

Swype Hero

Gesture Recognition, Motor Learning, and the Development of a Multimodal Typing Interface

A Game Design, Technical Architecture, and Research Specification

Flyxion

Independent Researcher

October 2026

Specification version 0.2 (working draft)

Abstract

Swype Hero is a browser-based arcade game in which descending balloons are destroyed by drawing the geometric shapes they carry. The game is designed to be enjoyable on its own terms, and it is also designed as a curriculum: a sequence of gestures of increasing geometric and motor difficulty that eventually converges on the trajectories traced by words across a touchscreen keyboard. This document is both a game design and a technical specification. It defines the playfield, the balloon dynamics, the gesture lifecycle, the representation and geometry of gestures, a family of recognition algorithms with explicit confidence and rejection behavior, a learning and difficulty model, the construction of word trajectories from keyboard geometry, the software architecture of a browser implementation, a roadmap of later input devices, and an experimental research program for studying gesture recognition and motor learning with consent-based datasets.

Three commitments organize the whole. Recognition is independent of gameplay, so that the same engine can serve a typing tutor, a keyboard, or another application. Raw input is kept separate from interpreted gestures, so that recognition errors can be investigated and algorithms compared without rewriting recorded observations. Learning by the player and adaptation by the software are treated as distinct processes that must remain experimentally distinguishable. The specification states normative requirements with the conventions described below, marks the evidential status of its historical and behavioral claims, and defers every empirical question it cannot settle to a named hypothesis and an experimental protocol.

Motivation

Swype Hero is motivated by the observation that gesture-based text entry remains an underexplored form of human-computer interaction, despite the remarkable proficiency that users can develop through prolonged practice. The original Swype keyboard demonstrated that continuous movements across a keyboard could become a highly efficient means of producing language. Experienced users learned not merely to trace letters, but to develop distinctive motor strategies involving curves, sharp turns, abbreviated trajectories, and deliberate exaggerations that made otherwise ambiguous words recognizable. Over time, the relationship between the user and the recognition system could become increasingly specialized.

The discontinuation of Swype also illustrates a broader problem in software development: improvements in contemporary products do not necessarily preserve the capabilities of their predecessors. Gesture shortcuts, personalized dictionaries, and recognition behaviors developed through years of use can disappear when an application is abandoned or replaced. The resulting loss is not merely functional. It can invalidate a substantial investment in acquired motor proficiency.

Swype Hero approaches this problem through play. Inspired in part by gesture-drawing games such as Magic Touch, it begins with a simple premise: players destroy balloons by reproducing increasingly complex geometric shapes. What initially appears to be an arcade mechanic gradually becomes a curriculum in continuous gesture production, progressing from lines, angles, and loops to intricate trajectories resembling the shapes of written words. Eventually, the game can introduce actual keyboard-derived word gestures, transforming accumulated motor skills into practical text-entry proficiency.

The project also provides an experimental environment for investigating gesture recognition, motor learning, ambiguity, and personalization. Rather than assuming that users should reproduce a single ideal trajectory, it asks how recognition systems might accommodate the distinctive movements that individuals develop through practice. Its longer-term architecture anticipates alternative input methods, including mice, keyboards, joysticks, Etch A Sketch-style controls, and eye tracking, allowing a common gesture vocabulary to be explored across different physical interfaces.

The immediate objective is modest: to create an enjoyable, responsive browser game with progressively challenging gestures. The larger ambition is to investigate how a playful interaction can become a general method of acquiring computational fluency, and how software might preserve and extend that fluency rather than repeatedly requiring users to begin again.

Conventions of This Specification

Normative language. The words `MUST`, `MUST NOT`, `SHOULD`, `SHOULD NOT` and `MAY`, set in small capitals, are used in the sense of RFC 2119 [4]. Statements containing them appear inside numbered requirements. Text outside a numbered requirement is explanatory unless it says otherwise. A requirement binds an implementation only within the milestone that introduces it (Part XII); later parts describe designs for later milestones and are normative only when those milestones are adopted.

Evidential status. The project draws on published research, on the documented behavior of software, on the reported experience of long-term users, and on its own design decisions. These are different kinds of claim and the text marks them. A superscript ^[D] marks a claim documented in the published literature or in primary technical documentation. A superscript ^[O] marks a behavioral observation reported from practice that has not been independently verified. A superscript ^[H] marks a hypothesis that this project proposes to test, and each such hypothesis is stated as a numbered hypothesis with a protocol in Part X. A superscript ^[S] marks a design decision of this specification, which is a commitment and not a claim about the world. A superscript ^[C] marks a constant that awaits calibration: the requirement that uses it is executable and testable once a value is supplied, but the value, its unit and the distance to which it applies are fixed by a stated procedure (Appendix A) and not by this text. A requirement without that mark is testable as written. Unmarked statements are definitions, arithmetic consequences of definitions, or connective prose.

Coordinates and units. The playfield uses logical units defined in Chapter 4, with the origin at the top-left corner and the *y* axis pointing down, as in the canvas coordinate systems of browsers. Angles are in radians unless stated otherwise. Distances on the keyboard are measured in key widths. Times are in milliseconds and are taken from a monotonic clock.

Computed values. Every numerical value reported as computed in this document, including the word-geometry figure and tables of Part VI, is produced by the script `tools/wordgeom.py` distributed with the source (and the edge-case fixtures of Appendix H by `tools/fixtures_check.py`), from the definitions given in the text. Numerical constants chosen as defaults, such as thresholds and budgets, are starting points for calibration and are labeled as design decisions.

Notation. A gesture is written G and a library template T . The number of points after resampling is N . The recognizer's base distance is d . Three derived quantities are kept apart: the quotient pseudometric d_H under a common isometry group H , the directional matching cost c_g of a stroke against a candidate gesture g under its allowed set $\mathcal{S}_g$, and the symmetric confusability κ between templates of distinct gestures (Chapter 12). A fuller table of symbols is given in Appendix A.

Structure. Parts I and II specify the concept and the playable game. Part III develops the geometry on which everything else rests, and Part IV the algorithms that act on it. Part V treats learning, Part VI the passage from shapes to words, and Part VII presentation. Part VIII is the software architecture. Part IX anticipates other input devices, Part X is the research program, Part XI describes applications beyond the game, and Part XII sets out the roadmap and the scope of the first implementation. Appendices collect schemas, reference algorithms, protocols and the historical record.

Contents

Abstract	ii
Motivation	iii
Conventions of This Specification	iv
I Concept and Design Philosophy	1
1 Introduction and Project Objectives	2
2 Historical Background	4
2.1 Evidence classes	4
2.2 Gesture typing	4
2.3 Behaviors attributed to Swype	5
2.4 The consequence of discontinuing adaptive software	5
2.5 Magic Touch as a reference	5
3 Design Principles	7
II Core Gameplay	9
4 The Playfield	10
4.1 Layout	10
4.2 Logical coordinates	10
4.3 The drawing surface	11
4.4 Background and score display	11
5 Balloon Behavior	13
5.1 State and motion	13
5.2 Spawning	13
5.3 Difficulty and position	14
5.4 Outcomes	14
5.5 Duplicate gestures	14
5.6 Candidate set restriction	14

6	Gesture Execution	16
6.1	The gesture lifecycle	16
6.2	Minimum stroke	16
6.3	Recognition on release and during the stroke	16
6.4	Feedback and outcomes	17
6.5	Ink	17
7	Scoring and Progression	19
7.1	Score components	19
7.2	Not rewarding precision	19
7.3	Progression	20
8	Game Modes	21
III	The Geometry of Gestures	23
9	Gesture Representation	25
9.1	The trajectory	25
9.2	Path and execution	25
9.3	Derived quantities	26
9.4	Raw and derived data	26
10	Primitive Geometric Gestures	27
10.1	The initial vocabulary	27
10.2	Geometric complexity and motor difficulty	27
10.3	Criteria for introductory targets	28
11	Compound Gestures	29
11.1	A small algebra of shapes	29
11.2	Visual and execution complexity	29
12	Gesture Equivalence and Invariance	31
12.1	Which transformations preserve identity	31
12.2	Three notions of distance	31
12.3	Examples of collision	34
13	Curvature and Directional Information	35
13.1	Features that survive noise	35
13.2	Landmarks	35
13.3	Informative portions	35
14	Temporal Structure	37
14.1	What time contributes	37
14.2	Measures of execution	37
14.3	Which temporal properties to ignore	38

15 Gesture Complexity	39
15.1 A multidimensional measure	39
15.2 Computed example	40
15.3 From intuition to data	40
 IV Recognition Algorithms	 41
16 Input Normalization	43
16.1 The pipeline	43
16.2 Resampling	43
16.3 Scale normalization	44
16.4 Coordinate transformations	44
 17 Baseline Template Matching	 45
17.1 The recognizer	45
17.2 Cost and reproducibility	45
 18 Dynamic Time Warping	 47
18.1 Elastic alignment	47
18.2 Constraints and complexity	48
18.3 When warping helps and when it does not	48
 19 Geometric Feature Recognition	 49
19.1 Recognizing by structure	49
19.2 Comparison with pointwise matching	49
 20 Probabilistic Recognition	 50
20.1 From distances to probabilities	50
20.2 Likelihood model	50
20.3 Calibration	51
 21 Recognition Confidence	 52
21.1 The decision rule	52
21.2 Active candidates and the full vocabulary	53
21.3 Why an absolute gate is necessary	53
21.4 Setting thresholds and reporting	54
 22 Partial Trajectory Recognition	 55
22.1 Recognizing before the stroke ends	55
22.2 Prefix distance	55
22.3 Commit rule	55
22.4 A property of the library	56
22.5 Effects on play and on learning	57
 23 Personalized Gesture Recognition	 58

23.1 Two things that can be learned	58
23.2 Template adaptation	58
23.3 Protection against accidental learning	59
23.4 Prior adaptation	60
24 Recognition Evaluation	61
24.1 Measures	61
24.2 Evaluation across users	61
24.3 Benchmark datasets and procedures	62
V Learning and Difficulty Progression	63
25 Motor Learning Foundations	65
25.1 What is established	65
25.2 Learning is not adaptation	65
26 The Introductory Curriculum	67
26.1 Principle	67
26.2 Mastery	67
26.3 Pacing	67
27 Intermediate Gesture Families	69
27.1 Compound curves and multiple turns	69
27.2 Overlapping trajectories	69
27.3 Increasingly similar shapes	69
28 Advanced Gesture Families	71
28.1 Exercises	71
28.2 Dosage	71
29 Adaptive Difficulty	72
29.1 A competence model	72
29.2 Selecting challenges	72
29.3 Not a scalar	73
30 Gesture Fluency	74
30.1 Dimensions of fluency	74
30.2 Fluency is not fidelity	74
31 Abbreviation and Economical Movement	76
31.1 What abbreviation is	76
31.2 A formal frame	76
31.3 Teaching abbreviation	77
32 Retention and Transfer	78

32.1 Retention	78
32.2 Transfer	78
32.3 Three skills that are not interchangeable	79
VI From Geometric Shapes to Words	80
33 Keyboard Geometry	82
33.1 Layouts as data	82
33.2 The idealized word path	82
33.3 Repeated letters and neighbors	83
34 Word Shapes	84
34.1 Geometric identity of a word	84
34.2 Generating and classifying shapes automatically	84
35 Word Difficulty	86
35.1 Length is not geometric complexity	86
35.2 A curriculum by trajectory difficulty	86
36 Transitional Exercises	88
37 Visible Keyboard Assistance	90
37.1 Assistance levels	90
37.2 Fading	90
38 Vocabulary Learning	91
38.1 Classes of words	91
38.2 A transparent personal dictionary	91
39 Word Ambiguity	93
39.1 Geometric confusability	93
39.2 Geometry and priors	93
39.3 Training on ambiguity	94
40 Sentence Composition	95
40.1 From words to text	95
40.2 Spaces, punctuation, capitalization and correction	95
40.3 Word selection	95
41 Typing Proficiency	97
41.1 Speed and accuracy	97
41.2 Measures specific to gesture entry	97
41.3 Realistic tasks	98

VII Game Presentation and Interaction	99
42 Visual Design	101
42.1 A restrained visual language	101
42.2 Legibility under overlap	101
42.3 Contrast and sizing	101
43 Animation and Feedback	103
43.1 Motion of balloons	103
43.2 Popping, ambiguity and failure	103
43.3 Transitions	103
43.4 Reduced motion and flashing	104
44 Audio	105
45 Touchscreen Ergonomics	106
45.1 Occlusion	106
45.2 Screen size, handedness and gesture area	106
45.3 Edges and system gestures	106
45.4 Scale and recognition accuracy	107
46 Accessibility	108
46.1 Baseline requirements	108
46.2 Pause and stop	108
VIII Software Architecture	109
47 Technology Selection	111
47.1 Language and rendering	111
47.2 Priorities	111
47.3 Package structure	111
48 Game Loop and Simulation	113
48.1 Separation of stages	113
48.2 Fixed timestep	113
48.3 Determinism and replay	114
49 Input Abstraction	115
49.1 A neutral trajectory interface	115
49.2 The pointer adapter	116
50 Recognition Engine	117
50.1 Modular interfaces	117
50.2 What a result contains	118

51 Gesture Library	119
51.1 Contents	119
51.2 Versioning	119
52 Player Profiles	121
53 Local Persistence	122
53.1 Storage	122
53.2 Export, import and deletion	122
54 Performance	123
54.1 Budgets	123
54.2 Measurement	123
55 Testing	124
55.1 Levels of testing	124
55.2 Properties	124
55.3 Fixtures	125
56 Deployment	126
 IX Advanced Input Interfaces	 127
57 Keyboard-Controlled Drawing	129
57.1 A cursor moved by keys	129
57.2 Simultaneous directions and acceleration	129
57.3 Consequences for recognition and difficulty	129
58 Etch A Sketch Controls	131
58.1 Independent axes	131
58.2 Motor profile	131
59 Joysticks and Gamepads	132
59.1 Position and rate control	132
59.2 Interface and calibration	132
60 Alternative Pointing Devices	133
61 Eye Tracking	134
61.1 The difficulty of gaze as a pointer	134
61.2 Delimiting strokes	134
61.3 Gestures from fixations	134
62 Assistive Input Systems	136
63 Cross-Device Recognition	137

63.1 The question	137
63.2 Measures	137
X Experimental Research Program	138
64 Gesture Dataset Construction	140
64.1 Consent and minimization	140
64.2 Gesture data is biometric	140
64.3 Oversight	141
65 Human Variation	142
66 Learning Curves	143
66.1 Measurement	143
66.2 Models	143
66.3 The control that matters	143
66.4 Attrition	144
67 Human Adaptation versus Model Adaptation	145
67.1 The decomposition	145
67.2 Replay and its limits	145
68 Recognition under Degraded Information	147
69 Competing Recognizers	148
70 Historical Swype Comparison	149
70.1 Why compare	149
70.2 Feasibility and method	149
70.3 Limits of inference	149
71 Reproducibility and Open Datasets	151
71.1 Artifacts	151
71.2 Documenting data	151
71.3 Privacy-aware release	151
XI Future Applications	153
72 Standalone Gesture Keyboard	155
72.1 From engine to keyboard	155
72.2 Platform constraints	155
73 Gesture Command Languages	156
74 Personalized Motor Vocabularies	157

75 Input-Independent Interaction	158
XII Implementation Roadmap	159
76 Milestones	161
77 Scope of the First Implementation	163
77.1 Three architectural commitments	163
77.2 The coherent progression	163
XIII Appendices	165
A Notation and Symbols	166
B Gesture Library Schema	168
C Reference Algorithms	170
D The Simulation Step	173
E Coordinate Normalization Procedures	174
E.1 Device to logical coordinates	174
E.2 Normalized form	174
E.3 Keyboard coordinates	174
F Difficulty Metrics	175
G Input Adapter Contracts	176
H Test Fixtures	177
H.1 Format	177
H.2 Edge-case fixtures	177
H.3 A worked normalization	177
I Browser Compatibility Requirements	180
J Experimental Protocols	181
K Record of Historical Claims	183
Bibliography	185

Part I

Concept and Design Philosophy

Chapter 1

Introduction and Project Objectives

Swype Hero is a game in which balloons descend a screen and each carries a geometric symbol. The player destroys a balloon by drawing, anywhere on the screen, the shape that its symbol depicts. A recognized shape pops the matching balloon and scores; a balloon that reaches the bottom of the screen costs the player a life. The premise is deliberately close to that of *Magic Touch: Wizard for Hire*, a mobile game in which enemies are defeated by tracing the glyphs they display, and that game is the reference for the real-time feel of gesture-driven arcade play. Swype Hero differs from it in emphasis. Where a conventional gesture-arcade game uses a fixed and modest set of glyphs as a means of generating action, Swype Hero treats the glyph set as the subject matter. The vocabulary grows by design, the geometric complexity of the targets is measured and sequenced, and the long-range destination of the sequence is the trajectory that a word traces across a keyboard.

That destination explains why the project is a game and not an exercise program. A typing tutor in the usual sense takes a conventional drill, such as typing a displayed word, and attaches rewards to it. The approach taken here runs in the other direction. The activity that the player performs from the first minute is drawing continuous movements under time pressure and receiving immediate judgment, and the movements are chosen so that they gradually become the movements that gesture typing requires. By the time words appear as targets, the player has already rehearsed the component skills: starting and ending a stroke deliberately, turning sharply, drawing arcs and loops, abbreviating, and recovering from a misjudged stroke. The tutor is thus a consequence of the curriculum and not an overlay on top of the game.

Two requirements follow, and they conflict in useful ways. The first is that the initial application must be independently enjoyable. A player who has no interest in gesture recognition research, and who never learns that the game is part of one, should find it responsive, readable and fair, and should want to play again. The second is that the application must be a trustworthy research instrument, which demands that every input be preserved in raw form, that every recognizer decision be reproducible, and that every adaptive behavior be switchable and attributable. The specification resolves the tension by keeping the research apparatus beneath the game: data collection is optional and consent-based, adaptation is explicit, and nothing about the research changes what the player sees unless a mode says so.

Requirement 1.1 (Independence of enjoyment). The first playable release **MUST** be usable without account creation, network access after installation, or participation in any data collection. All research functions **MUST** be opt-in.^[5]

Scope and non-goals. The specification covers the game, the recognition engine, the curriculum, the progression from shapes to words, the browser architecture, later input devices, and the research program. It does not specify a commercial product, a monetization scheme, a social or competitive service, or a system-wide keyboard for mobile operating systems, although Chapter 72 states what such a keyboard would require. It does not claim that the curriculum improves typing speed. That is a hypothesis, stated as such in Part X, and the game is designed to produce the evidence that would confirm or refute it.

How the document is used. The document is intended to be developed incrementally. Part XII orders the work into milestones, and the first milestone, a playable prototype with a small gesture vocabulary and a transparent recognizer, is the immediate engineering target. Later parts record design intent early so that the first implementation does not foreclose it, in particular the separation of raw input from interpreted gestures and the separation of recognition from game logic.

Chapter 2

Historical Background

2.1 Evidence classes

This chapter and Appendix K concern a history in which documented mechanisms, reported behavior and plain speculation are easily confused, particularly for a product whose internals were proprietary and whose users formed strong impressions of how it worked. The tags defined in the Conventions are therefore applied with some care. A claim tagged ^[D] can be traced to a paper, a patent, a manual or other primary document. A claim tagged ^[O] describes what long-term users report about observed behavior and is retained because it motivates design and experiment, not because it is established. A claim tagged ^[H] is something the project proposes to measure.

2.2 Gesture typing

Continuous gesture typing, in which a single stroke across a keyboard indicates a word, has a documented research lineage. Zhai and Kristensson described shorthand writing on a stylus keyboard, in which a word is entered by sliding a stylus across the letters of the word, and described the resulting stroke pattern for a word as its shorthand form [33].^[D] Their SHARK2 system [16]^[D] recognized such strokes by comparing a user's stroke with templates generated automatically from the letters of each word in a large lexicon. SHARK2 compared strokes along two channels, one that was invariant to location and scale and measured shape, and one that used absolute position on the keyboard, and it combined the geometric evidence with a language model. A later review described the word-gesture keyboard as a paradigm that supports a gradual transition from visually guided tracing to recall of learned shapes [34].^[D] That transition, from tracing to recall, is exactly what Part VI seeks to train.

Commercial gesture keyboards followed. ShapeWriter was a commercial descendant of the IBM research line. Swype, created by Cliff Kushler and Randy Marsden, became widely known on Android devices around 2010, was acquired by Nuance Communications in 2011, and was discontinued by Nuance in 2018.^[D] Gesture typing was subsequently built into other keyboards, among them SwiftKey Flow and the glide typing feature of Gboard, so that the interaction survives in current products. What did not obviously survive was the

specific behavior of Swype, which is the subject of the project's historical interest. The dates here are given at the resolution of a year, and Appendix K lists what remains to be verified against primary sources before any of this chapter is cited as historical evidence.

2.3 Behaviors attributed to Swype

Experienced users of Swype and of other gesture keyboards report a set of behaviors that go beyond the published algorithms. They report that a stroke need not be completed for the intended word to be selected, so that a long word could be recognized when only its distinctive early portion had been drawn; that deliberate exaggeration, such as a swoop toward a distant letter or a sharply drawn corner on a letter that the language model might otherwise skip, increased the reliability of recognition; that the dictionary learned the user's vocabulary, so that words typed once became easier to produce; and that the keyboard offered gesture shortcuts for editing operations such as deleting a word or changing case.^[O] Each of these is plausible from the known mechanisms. Early selection is consistent with the use of a language model and a prefix-consistent search. Exaggeration is consistent with templates whose corners are made salient by a sharp turn. Vocabulary learning is a documented feature of predictive keyboards generally. The point is that none is documented in detail for Swype, and each is therefore treated here as an observation that motivates a design (Chapters 22, 31 and 23) and an experiment (Chapter 70).

2.4 The consequence of discontinuing adaptive software

An adaptive input method accumulates two kinds of state, and the difference between them matters for the design that follows. One is the personalization held in the software: the learned dictionary, the adjusted statistics of the user's strokes, the shortcuts. The other is the skill held in the user: the practiced movements, the chosen abbreviations, the sense of how far a swoop must go. The first lives in the application and, in the usual case, cannot be exported. When the application is discontinued, replaced by a differently behaving successor, or simply updated, the first kind of state is lost or invalidated, and the second kind is devalued because it was calibrated to a particular recognizer. The user is then required to learn again. The design response of this project is to make both kinds of state explicit and portable: personalization is stored in user-readable and user-owned form (Chapters 52 and 53), the recognizer is versioned and its version recorded with every decision (Chapter 50), and the claim that skill transfers across recognizers is itself an experimental question (Chapter 32).

2.5 Magic Touch as a reference

Magic Touch is relevant to this project for one narrow reason: it shows that quickly drawn glyphs can sustain real-time arcade play on a touchscreen without a separate selection step. The present specification adopts that observation and not the surrounding design. In particular it does not adopt the glyph set, the visual theme, the enemy structure or the

progression of that game, and it does not treat the game as a template. The glyph vocabulary of Swype Hero is derived from first principles in Part III, and the visual language is specified in Part VII.

Chapter 3

Design Principles

The project is governed by six principles. They are stated here in prose with the reasoning that supports them, and later chapters refer to them by number.

Principle 1 (Immediate feedback). The player sees the stroke respond to the finger within one display frame, and sees a decisive outcome shortly after the stroke ends. Feedback is never withheld to preserve the integrity of an experiment.

A drawing game fails if the ink lags the finger, and a learning game fails if the player cannot tell why a stroke was rejected. The principle therefore covers two time scales: the ink, which must follow the pointer without perceptible delay, and the outcome, which must be communicated clearly and quickly enough to affect the next stroke. Chapter 6 specifies three distinguishable outcomes so that a failure of the stroke can be told apart from a failure of the recognizer.

Principle 2 (Progressive difficulty). Difficulty increases chiefly by introducing new geometric features, and only secondarily by increasing speed, density or precision demands.

Speed is the cheapest source of difficulty in an arcade game and the least informative for learning. A player who is hurried is forced to repeat what they already know, whereas a player who meets a new feature, such as a sharp reversal or a self-crossing, must construct a new movement. The curriculum of Part V accordingly orders gestures by their features and uses time pressure only to consolidate.

Principle 3 (Meaningful motor learning). Practice in the game is intended to produce movement skill that persists and transfers, and the measurements of the game are designed to detect whether it does.

Principle 4 (Low interaction overhead). A stroke is the only action required to attack a balloon. There is no selection step, no mode switch and no confirmation.

The drawing surface covers the playfield, so the player never needs to aim at a balloon before drawing. This is a deliberate design choice, because the typing skill the game prepares for is also a one-stroke action without prior selection of a target.

Principle 5 (Device independence). The recognition algorithm depends on a trajectory and on nothing about how the trajectory was produced.

The same recognizer is meant eventually to accept strokes from a finger, a mouse, a keyboard-driven cursor, an analog stick or a gaze tracker. That is feasible only if the interface between input and recognition is a neutral trajectory representation (Chapter 9) and if any adjustment for a device is expressed as an explicit, inspectable parameter and never as a hidden special case.

Principle 6 (Accommodation without erosion). The recognizer should come to accept the movements a player has learned, but must retain the distinctions between gestures that the game depends on.

The last principle contains the project’s central tension. A recognizer that insists on a single ideal trajectory forces the player to keep reproducing it with indefinite precision, which is neither fun nor representative of how skilled gesture typists actually behave. A recognizer that adapts without limit, however, converges on whatever the player does, and in the limit accepts almost any stroke as almost any gesture, which removes the learning problem and ruins the game. The resolution adopted is to bound adaptation by a distinguishability constraint.

Definition 3.1 (Separation and floor). Let $\Lambda = \{T_1, \dots, T_K\}$ be a gesture library whose gestures declare allowed sets of transformations (Chapter 12), with confusability distance κ . The *separation* of the library is

$$\text{sep}(\Lambda) = \min_{g_i \neq g_j} \kappa(T_i, T_j),$$

the minimum being over templates of distinct gestures. A *separation floor* is a constant $\delta_{\min} > 0$, in normalized-stroke units, chosen so that gestures at confusability distance $\delta_{\min}$ are reliably told apart by players and by the baseline recognizer.

Requirement 3.1 (Distinguishability under personalization). After any personalization update, the personalized library Λ_u **MUST** satisfy $\text{sep}(\Lambda_u) \geq \delta_{\min}$. An update that would violate the floor **MUST** be rejected or scaled back along the update direction until the floor holds (Chapter 23).^[S] The value of $\delta_{\min}$ is a calibration constant^[C] and the requirement is testable once it is supplied.

The constant $\delta_{\min}$ is calibrated empirically (Chapters 24 and 69), and the mechanism that enforces the requirement is given in Chapter 23.

Part II

Core Gameplay

Chapter 4

The Playfield

4.1 Layout

The playfield is a two-dimensional rectangular region that fills the browser viewport, designed first for a phone held in portrait orientation. Balloons enter from a spawn band along the top edge and descend toward a failure line near the bottom edge. A balloon is a filled shape with a recognizable geometric symbol drawn on its face. Above the balloons, the player's strokes are drawn as ink on a transparent drawing surface that covers the whole playfield. A narrow status strip at the top shows the score, the remaining lives and the current streak; it is part of the layout and does not intercept strokes except in its small control area.

4.2 Logical coordinates

All simulation and recognition code operates in logical units, which are independent of the pixel size and the pixel density of the screen. The width of the playfield is fixed at $W_p = 1000$ logical units. The height H_p is determined by the aspect ratio of the viewport, subject to the bounds $1.2W_p \leq H_p \leq 2.4W_p$, and a letterbox or pillarbox fills the remainder when the viewport lies outside those bounds. The failure line is placed at $y = H_p - m_f$ with $m_f = 0.12H_p$, leaving a margin at the bottom for the resting hand and for system gestures. The spawn band occupies $0 \leq y \leq m_s$ with $m_s = 0.08H_p$ below the status strip.^[S] A transform between device pixels and logical units is computed once per layout change and applied in both directions; it is the only place in the program where device pixel ratio is visible.

Requirement 4.1 (Logical units). Simulation, scoring and recognition MUST be expressed only in logical units. Rendering MUST be the only stage that depends on device pixel dimensions.^[S]

4.3 The drawing surface

The surface that receives strokes covers the entire playfield, including the areas occupied by balloons and the status strip other than its control area. This follows Principle 4: the player draws, and the game decides which balloon the stroke concerns. The surface must not scroll, zoom or select text in response to touches, which in a browser is accomplished by declaring that the touch action of the element is none and by handling pointer events directly (Chapter 49). A margin of about 24 logical units along the left and right edges is not treated as a drawing region, because mobile systems reserve the screen edge for navigation gestures and a stroke that begins in that margin may be taken by the system.^[5]

4.4 Background and score display

The background is a quiet, low-contrast field that does not compete with the balloons or the ink, and it is specified in Chapter 42. The score, lives and streak are shown in the status strip in a fixed typeface at a size legible at arm's length, and a visual change at a level transition is brief. Nothing in the display is permitted to move or flash in a way that would draw the eye away from the descending balloons.

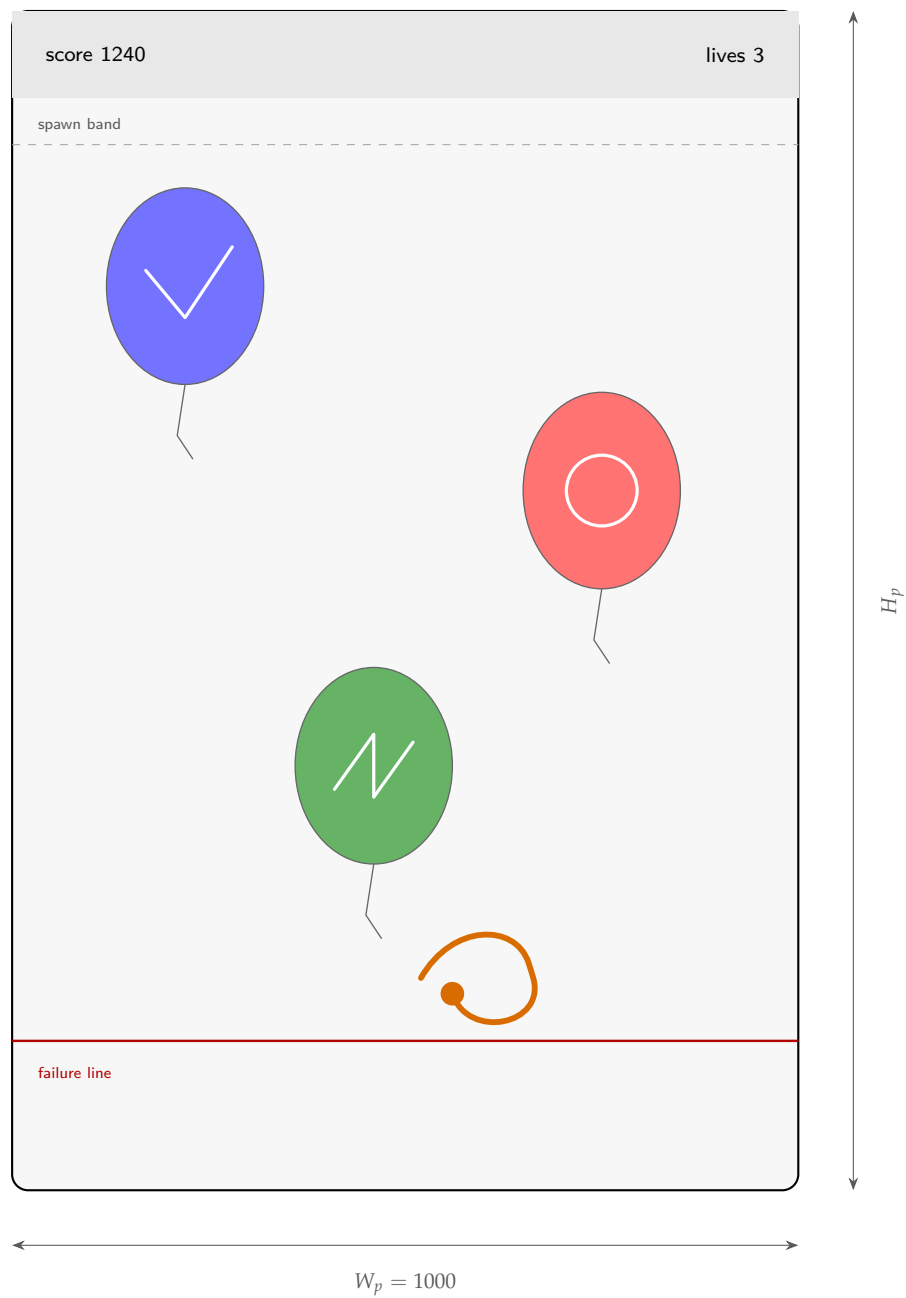


Figure 4.1: Schematic playfield in portrait orientation. Balloons carry glyphs and descend from the spawn band toward the failure line; the player's stroke (orange) is drawn on a transparent surface covering the whole field. The logical width is fixed and the height H_p follows the viewport aspect ratio.

Chapter 5

Balloon Behavior

5.1 State and motion

A balloon is a record with an identifier, a gesture identifier, a horizontal position, a vertical position, a descent speed, a radius, and a spawn time. Its horizontal position is fixed at spawn and chosen so that balloons do not overlap at birth; its vertical position decreases the distance to the failure line at a rate given by the descent speed. Horizontal drift is not part of the first implementation, because it adds visual noise without adding to the gesture-learning problem.

The descent speed is a function of elapsed play time and of the level. Let t be the time since the start of the current level, and let v_0 be the initial speed, a the acceleration and $v_{\max}$ the cap. Then

$$v(t) = \min(v_{\max}, v_0 + a t),$$

with $v_0 = 60$, $a = 0.8$ and $v_{\max} = 140$ logical units per second as starting values to be calibrated.^[S] Under these values, on a typical phone aspect ratio of about 16:9 (so $H_p \approx 1780$), a balloon takes roughly 10 seconds to cross the field at the speed cap and roughly 24 seconds at the initial speed, which gives a beginner ample time to draw and a practiced player room to spare.

5.2 Spawning

Balloons are generated by a spawner that draws the next spawn time from a distribution with a mean interval $\bar{s}$ that decreases slowly with the level, and that selects each balloon's gesture from the current *gesture pool*, the set of gestures eligible at the player's stage in the curriculum (Chapter 26). The interval is jittered, uniformly in $[0.7\bar{s}, 1.3\bar{s}]$ in the first implementation, so that arrivals are not so regular that the player settles into a rhythm unrelated to the shapes. An upper bound $B_{\max}$ on the number of simultaneous balloons is set by the level, beginning at two.

Requirement 5.1 (Seeded spawning). The spawner MUST draw all of its randomness from a seeded pseudorandom generator whose seed is recorded with the session, so that a session can be replayed exactly from the seed and the recorded input (Chapter 48).^[S]

5.3 Difficulty and position

The pressure on the player at a given instant depends on how close the lowest balloon is to the failure line and on how complex the gesture it carries. A balloon near the line that carries a simple gesture is a manageable pressure; a high balloon that carries a complex gesture is not yet pressure at all. The relationship is formalized with a time budget. For a balloon b at height y_b with descent speed v , the time remaining is $\tau_b = (y_f - y_b)/v$ where y_f is the failure line. If the gesture has estimated execution time $\hat{T}(g)$ (Chapter 11), the *slack* is $\tau_b - \hat{T}(g)$, and the spawner is constrained so that, for the player's estimated skill, the slack of a newly spawned balloon is positive with high probability. This makes the speed parameters a consequence of the gesture set and not a free dial that a designer must set blindly.

5.4 Outcomes

When a balloon is hit, it is destroyed, its score is awarded (Chapter 7) and the player's streak is incremented. When the player completes a stroke that is recognized as a gesture displayed by no balloon, nothing is destroyed, and the stroke is reported as a miss; a miss ends the streak, and in survival mode carries no further penalty. When a balloon reaches the failure line, it is removed, a life is lost, the streak ends, and the lost gesture is recorded in the player's history as a failure for the purposes of adaptive difficulty (Chapter 29). The asymmetry, that wrong strokes cost only the streak while missed balloons cost lives, is intentional. It keeps exploration cheap, which matters for learning, while preserving the pressure of the descent.^[5]

5.5 Duplicate gestures

Several balloons may display the same gesture. A single stroke recognized as that gesture destroys exactly one of them, and the rule for choosing which is: the balloon nearest the failure line among those displaying the recognized gesture, with ties broken by earlier spawn time. This rule is predictable, it favors the most urgent target, and it rewards no attention to balloon position that the player has not already invested. A later mode may allow a stroke to destroy all balloons of a gesture, as a power-up or as a consequence of a streak, but the default is single destruction.

Requirement 5.2 (Target selection). When a stroke is accepted as gesture g , the game **MUST** destroy the balloon displaying g whose vertical position is lowest on the screen, breaking ties by earliest spawn time, and **MUST** destroy no other balloon.^[5]

5.6 Candidate set restriction

A related design choice concerns the recognizer. The recognizer may be asked to classify a stroke against the whole library or only against the gestures currently displayed by living

balloons. Restricting the candidate set raises accuracy, because fewer classes compete, and it permits gestures that are very close in the full library to share a game. It also changes what the player learns, since the stroke is checked against what is on screen and not against the player's recall. The specification adopts the restricted candidate set as the default for survival mode and the full library for practice mode, and records the candidate set with every recognition result so that the effect can be measured (Chapter 24).^[5]

Chapter 6

Gesture Execution

6.1 The gesture lifecycle

A gesture begins when the pointer becomes active, continues while the pointer moves, and ends when the pointer is released or the interaction is cancelled. In terms of browser pointer events these are `pointerdown`, `pointermove` and `pointerup` or `pointercancel`. A stroke is therefore a sequence of timestamped samples delimited by two events, and the representation is given in Chapter 9. The game maintains a small state machine with four states.

In the *idle* state no pointer is active. A pointer-down event in the drawing surface, outside any control area, starts a stroke and moves to the *drawing* state, in which each movement appends a sample and extends the visible ink. Releasing the pointer moves to the *resolving* state, in which the recognizer runs on the completed stroke and returns an outcome. The *feedback* state then displays the outcome for a short, fixed time before returning to *idle*, and during this time a new stroke may already begin, so that feedback never blocks input. A cancel event discards the stroke without recognition and returns to *idle*. Only the first pointer is tracked in the first implementation; additional simultaneous pointers are ignored.

6.2 Minimum stroke

A touch that does not move far enough is a tap, not a gesture. A stroke **MUST** be treated as a candidate gesture only if its path length exceeds a minimum $\ell_{\min}$ and its extent exceeds a minimum $S_{\min}$, with starting values $\ell_{\min} = 40$ and $S_{\min} = 30$ logical units.^[S] Shorter strokes are discarded without any feedback other than the ink fading, because penalizing an accidental touch would be both unfair and discouraging.

6.3 Recognition on release and during the stroke

The default is that recognition occurs on release. It is simple, deterministic and equivalent to how the recognizer is evaluated in the benchmark (Chapter 24). Chapter 22 investigates the alternative in which a sufficiently confident partial stroke is accepted before it ends.

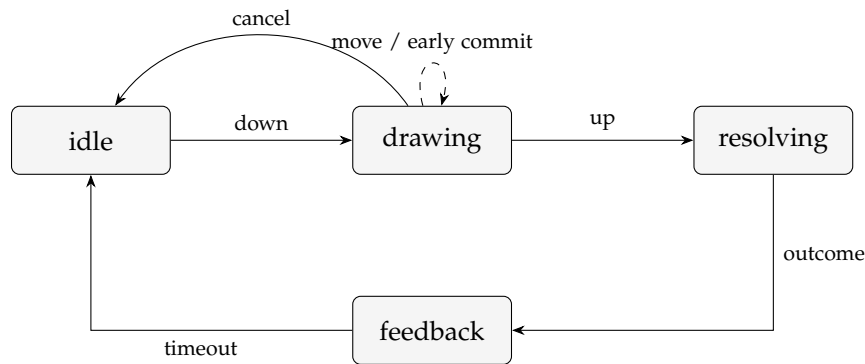


Figure 6.1: States of the stroke lifecycle. A new stroke may begin during the feedback state, so feedback never blocks input.

The early-commit variant changes the interaction in a visible way, because the stroke is cut short, and it changes the training in a less visible one, because a player rewarded for early commits will learn to draw only as much as is needed. That is the behavior that experienced gesture typists report as a skill (Chapter 31), and it is also exactly the behavior that makes full-stroke measurements uninformative. The specification therefore defines early commit as an optional recognition mode that is disabled in the first milestone and is recorded in the session metadata when enabled.

6.4 Feedback and outcomes

The recognizer returns one of three outcomes, and the game presents them so that they cannot be confused. An *accepted* outcome carries a gesture and a confidence and pops the corresponding balloon. An *ambiguous* outcome means that the stroke was close to more than one candidate and the recognizer declines to choose; the candidate balloons are briefly highlighted and no balloon is destroyed. A *rejected* outcome means that the stroke was not close to any candidate, or was malformed, and the ink dissolves. The distinction is essential for learning, since an ambiguous stroke tells the player that the shape was nearly right and perhaps merely too close to a neighbor, while a rejected stroke tells the player that the shape was wrong. Chapter 21 defines the decision rule.

Requirement 6.1 (Distinct outcomes). The three outcomes **MUST** be distinguishable without reliance on color alone, using differences of motion or shape in the feedback animation (Chapter 43), and an equivalent non-visual signal **SHOULD** be available (Chapter 44).^[5]

6.5 Ink

The ink is drawn as the stroke proceeds, as a smooth line through the sampled points with a fixed width of about 6 logical units, in a color that contrasts with both balloons and background. Rendering is incremental, adding each new segment to the existing path

without redrawing the entire stroke, which keeps the work per frame constant and the ink responsive on modest hardware.

Chapter 7

Scoring and Progression

7.1 Score components

The score for a destroyed balloon is the product of a base value, a speed factor and a streak multiplier, with an optional economy bonus. The base value $B(g)$ is a function of the gesture's complexity $C(g)$ from Chapter 15, so that a more demanding gesture is worth more, and it is defined by $B(g) = B_0 (1 + \beta C(g))$ with $B_0 = 100$ and $\beta = 2$.^[S] The speed factor rewards a prompt response and is defined by

$$\sigma = 1 + \lambda \operatorname{clip}\left(1 - \frac{t_{\text{resp}}}{t_{\text{avail}}}, 0, 1\right),$$

where t_{resp} is the time from the balloon's appearance to the pointer release that destroyed it, t_{avail} is the total time the balloon would take to reach the failure line, and $\lambda = 0.5$. The streak multiplier is $m(k) = 1 + \min(k, k_{\text{max}})/10$ for a streak of k consecutive destroyed balloons with $k_{\text{max}} = 10$.

7.2 Not rewarding precision

A game that rewards geometric precision rewards the wrong skill. A player who traces a shape slowly and exactly achieves a high recognition score and a poor game result, while a player who produces the same recognized shape in one smooth, economical stroke is exactly the player the curriculum is meant to produce. The specification therefore forbids any term in the score that increases monotonically with the closeness of the stroke to the template, once the stroke is accepted.

Requirement 7.1 (Precision neutrality). The score of an accepted stroke **MUST** be independent of the distance between the stroke and the template, except through the accept decision itself.^[S]

An economy bonus that rewards drawing only what is needed is permitted, and is defined from path lengths and not from template distance. Let $\ell(G)$ be the path length of the accepted stroke and $\ell_{\text{ref}}(g)$ the path length of the reference gesture. The economy factor is $\epsilon = \min(1, \ell_{\text{ref}}(g)/\ell(G))$, and the bonus is $B_{\text{econ}} = B(g) \kappa \epsilon$ with $\kappa = 0.1$. A stroke that

wanders is worth slightly less than one that goes directly, and a stroke shorter than the reference earns the full bonus.

7.3 Progression

Progression is the advance through the gesture pool, and it is driven by performance on gestures and not by score. A new gesture family is unlocked when the player's competence estimates on the current families (Chapter 29) exceed a mastery criterion, and the score is therefore a record and a source of play and not the control variable of the curriculum. This separates two things that games commonly conflate: the pleasure of a high score and the evidence of learning.

Requirement 7.2 (Replayable scoring). The score **MUST** be a deterministic function of the recorded input events, the seed and the recognizer version, so that any recorded session can be re-scored.^[S]

Chapter 8

Game Modes

A game mode is a configuration of six elements: the spawner policy, the candidate set, the failure rule, the scoring rule, the adaptation policy and the logging policy. Making the mode a configuration, and not a separate code path, means that new modes can be added without changing the recognizer or the simulation, and that experiments can be defined as modes.

Survival. The basic mode is the arcade game as described in the preceding chapters. It ends when lives are exhausted. It is the mode in which enjoyment is tested and the only mode of the first implementation.

Practice. The practice mode removes time pressure. A target gesture is displayed with its reference trajectory, the player draws it, and the outcome is reported with the stroke overlaid on the reference. The label of the stroke is known, because the target was prompted, and this is the mode in which supervised personalization is safe (Chapter 23).

Progressive training. The training mode is survival with a pool selected by the competence model, so that the player meets gestures at the edge of their ability and is given lighter pressure while a new gesture is consolidated.

Later modes. The specification anticipates timed challenges with fixed durations, restricted gesture families for focused drills, competing targets in which several balloons carry confusable gestures, and personalized exercises generated from a player's own error history. None is part of the first implementation, which concentrates on a single polished survival mode so that the central loop can be judged before it is extended.^[5]

Element	Survival	Practice	Progressive training
Spawner	Seeded, level-driven	Single target, repeated or cycled	Selected by competence model
Candidate set	On-screen gestures	Full library	On-screen gestures
Failure rule	Lives, balloon reaches line	None	Lives, generous
Scoring	Full	None, or accuracy only	Full, with mastery tracking
Adaptation	Off by default	May update templates	Updates competence estimates
Logging	Session record	Session record	Session record, learning curves

Table 8.1: The three modes of the first design, expressed as configurations.

Part III

The Geometry of Gestures

This part is the mathematical and experimental foundation of the project. It defines what a gesture is, what a gesture vocabulary should contain, which transformations preserve a gesture's identity, which features carry the information that a recognizer uses, and how the difficulty of a gesture can be measured. The definitions here are used without change by the recognizers of Part IV, the curriculum of Part V and the word geometry of Part VI.

Chapter 9

Gesture Representation

9.1 The trajectory

A *gesture* is a finite, time-indexed sequence of points in the plane,

$$G = \{(x_i, y_i, t_i)\}_{i=1}^N, \quad t_1 < t_2 < \dots < t_N,$$

where $(x_i, y_i) \in \mathbb{R}^2$ are logical coordinates (Chapter 4) and t_i is a timestamp in milliseconds from a monotonic clock. The sequence begins at the first sample after the pointer becomes active and ends at the sample at which it is released. Samples need not be equally spaced in time or in space, because both the spacing of events and the speed of the hand vary. The number N is not fixed; a stroke of two seconds on a device delivering 60 events per second has about 120 samples, while a high-rate mouse can deliver several times that.

The planar sample is a minimum. An implementation may attach further channels to each sample, so that a general sample is a vector $s_i = (x_i, y_i, t_i, \mathbf{c}_i)$ where $\mathbf{c}_i$ holds optional measurements such as contact pressure, contact width and height, stylus tilt, the pointer type, or device orientation. The recognizers of this specification use only the planar part, but the extended channels are preserved because they cannot be recovered later and because research questions about pressure and contact area are likely.^[5]

9.2 Path and execution

Two strokes can follow the same route at different speeds, with different pauses, and with different sampling. It is useful to separate what was drawn from how it was drawn. The *geometric path* of a gesture is the curve $\gamma : [0, 1] \rightarrow \mathbb{R}^2$ obtained by interpolating the samples and parameterizing by normalized arc length. The *execution* is the monotone map φ from time to arc length, so that the planar part of the gesture is $G(t) = \gamma(\varphi(t))$ with φ non-decreasing. Under this decomposition a stroke drawn twice as fast has the same path and a different execution, and a stroke with a pause has a φ with a flat interval. Part III treats the path and its features in Chapters 10 to 13, and the execution in Chapter 14. The recognizers of the first milestone depend only on the path.

Definition 9.1 (Geometric path equivalence). Two gestures G and G' are *path-equivalent* if their paths are equal as curves up to a monotone reparameterization. Path equivalence is the relation that a time-invariant recognizer must respect.

9.3 Derived quantities

Several quantities are computed from the samples and used throughout. Let $\Delta_i = (x_i - x_{i-1}, y_i - y_{i-1})$ for $i \geq 2$. The cumulative arc length is $s_1 = 0$ and $s_i = s_{i-1} + \|\Delta_i\|$, and the total path length is $\ell(G) = s_N$. The heading of the i th segment is $\theta_i = \text{atan2}(\Delta_{i,y}, \Delta_{i,x})$, and the turning angle at sample i is $\tau_i = \text{wrap}(\theta_{i+1} - \theta_i)$, where wrap maps an angle to $(-\pi, \pi]$. The discrete curvature at sample i is

$$\kappa_i = \frac{\tau_i}{\frac{1}{2}(\|\Delta_i\| + \|\Delta_{i+1}\|)},$$

and the instantaneous speed is $v_i = \|\Delta_i\| / (t_i - t_{i-1})$. With the logical coordinate system in which y points down, a positive turning angle corresponds to a clockwise turn on the screen.

9.4 Raw and derived data

The most important rule of the representation concerns what may be changed. A gesture as received from the input device is the *raw gesture*, and every other representation of it, whether smoothed, resampled, normalized, rotated or segmented, is a *derived gesture* computed from the raw one by a named, versioned procedure (Chapter 16). Derived gestures can be recomputed and discarded; the raw gesture is the observation, and it is the only thing in the system that cannot be regenerated.

Requirement 9.1 (Immutable raw input). The raw gesture **MUST** be stored exactly as received, **MUST** never be modified by any later processing, and **MUST** be retained with the device characteristics needed to interpret it (pointer type, device pixel ratio, viewport size, orientation, and the version of the input adapter). Every derived gesture **MUST** carry the identifier of the procedure and version that produced it.^[5]

This requirement is the foundation of the second architectural commitment of the project, that raw input remains distinct from interpreted gestures. It makes possible every retrospective analysis in Part X, in particular the comparison of recognizers on identical recordings and the study of whether a preprocessing choice caused a recognition error.

Chapter 10

Primitive Geometric Gestures

10.1 The initial vocabulary

The first gesture vocabulary is built from simple one-stroke shapes that a beginner can draw, that are visually distinct from one another, and that between them exercise the movements gesture typing will require. They are straight lines in the four principal directions and two diagonals, corners and hooks, arcs, circles in two senses of rotation, loops, and zigzags. A curriculum needs a library of about twenty entries to be worth playing, and the library below has twenty. The identifiers are stable and are used by the library schema of Appendix B.

10.2 Geometric complexity and motor difficulty

It is tempting to treat geometric complexity as motor difficulty, and the identification is not safe. A circle is geometrically simple, being a single curve of constant curvature, yet drawing a circle that closes cleanly in one stroke is motorically demanding for many people, who tend to overshoot the start or flatten one side. A zigzag has a more complicated description, yet the straight segments are easy and the reversals are abrupt but brief. The relationship between the geometry of a shape and the difficulty of producing it has two documented anchors. The first is the speed-accuracy tradeoff in goal-directed movement, summarized by Fitts's law [7].^[D] The second is the steering law, which models the time to move along a path of varying width and predicts that time grows with the integral of the inverse tolerance width along the path [1].^[D] Neither law predicts the difficulty of an arbitrary shape without calibration. The project therefore treats the relationship as an empirical one, and Hypothesis 10.1 states what is to be tested.

Hypothesis 10.1 (Difficulty is not path length). For gestures in the initial vocabulary, observed completion time and failure rate are better predicted by a model that includes the number and sharpness of direction changes and the presence of closure than by path length alone.^[H]

Id	Family	Description	Turns
line.h	Line	Horizontal stroke, left to right	0
line.v	Line	Vertical stroke, top to bottom	0
line.d1	Line	Diagonal, upper left to lower right	0
line.d2	Line	Diagonal, lower left to upper right	0
corner.l	Corner	Down then right, a right angle (the letter L)	1
corner.v	Corner	Down-right then up-right, acute (the letter V)	1
corner.caret	Corner	Up-right then down-right, acute (a caret)	1
corner.z	Corner	Right, down-left, right (the letter Z)	2
hook.j	Hook	Down then curving left (the letter J)	1
hook.u	Hook	Down, curve, up (the letter U)	1
arc.c	Arc	Open arc, concave right (the letter C)	0
arc.d	Arc	Open arc, concave left, anticlockwise from top (reverse C)	0
arc.s	Arc	Two opposed arcs (the letter S)	0
circle.cw	Circle	Closed circle drawn clockwise	0
circle.ccw	Circle	Closed circle drawn anticlockwise	0
loop.l	Loop	Rising stroke with one loop (the letter l in cursive)	0
loop.e	Loop	Small loop on a rightward stroke	0
zig.2	Zigzag	Right, down-left, right, down-left (two reversals)	3
zig.3	Zigzag	Zigzag with three reversals	4
spiral.in	Spiral	Inward spiral of one and a half turns	0

Table 10.1: The initial twenty-gesture vocabulary. The Turns column counts corner points, meaning sharp changes of direction, as distinct from continuous curvature.

10.3 Criteria for introductory targets

A gesture suits the introduction if it passes five criteria. It can be drawn in a single stroke of reasonable length. It has at most two corner points, so that the player has few things to remember. It has no self-intersection, so that its drawing order is unambiguous. It is at least $\delta_{\min}$ from every other gesture that appears with it at the same level, in the sense of Definition 3.1. Finally, it is described by a symbol that a player can recognize without instruction, which is a visual criterion tested informally in the first playtests. The straight lines, the corners, the open arcs and the circles satisfy all five; the zigzags, loops and spiral belong to the intermediate and advanced families of Part V.

Chapter 11

Compound Gestures

11.1 A small algebra of shapes

Compound gestures are built from the primitives of Chapter 10 by a few operations. *Concatenation* joins two paths end to start, with an optional specified join angle. *Repetition* traverses a path n times, as in a loop drawn twice or a zigzag with n reversals. *Reversal* traverses a path backward. *Reflection* mirrors a path about an axis. *Nesting* places one path inside another at a smaller scale, as in a spiral. *Crossing* requires that two parts of the path intersect. These operations generate, among others, the repeated loops, crossing trajectories, reversals and nested curves that the intermediate and advanced curriculum draws on, and they allow the library to be extended systematically instead of by ad hoc invention.

11.2 Visual and execution complexity

Two drawings may appear equally complicated while requiring very different motor coordination, and the distinction matters for choosing and ordering targets. The *visual complexity* of a shape is a property of the image: how much ink, how many parts, how many crossings. The *execution complexity* is a property of the movement: how fast it can be performed with a given accuracy, and how much coordination it demands. A figure-eight and a pair of side-by-side circles drawn as one continuous stroke are visually similar in the amount of ink and in the number of turns of the path. In the figure-eight, however, the direction of rotation reverses at the crossing, so that the hand must change from a clockwise to an anticlockwise sense mid-stroke, while the two circles of the other shape can be drawn with the same rotation sense and a retracing. Execution complexity can be modeled as a predicted completion time that sums, over the segments of the path, a steering-law term and a cost for each direction change,

$$\hat{T}(G) = \sum_k \left(a + b \frac{\ell_k}{W} \right) + c_{\text{turn}} \sum_i |\tau_i| + c_{\text{rev}} R(G),$$

where ℓ_k is the length of segment k , W is an effective path width set by the recognizer's tolerance, $R(G)$ counts reversals as defined in Chapter 15, and $a, b, c_{\text{turn}}, c_{\text{rev}}$ are constants to be fitted from player data.^[H] This predicted time is used by the spawner in Chapter 5 and

by the complexity measure of Chapter 15, and the fitting procedure is part of the research program (Chapter 66).

Chapter 12

Gesture Equivalence and Invariance

12.1 Which transformations preserve identity

A recognizer must decide which differences between two strokes are differences of gesture and which are differences of execution. A circle drawn small and a circle drawn large are the same gesture, and a circle drawn at the top of the screen and a circle drawn at the bottom are the same gesture. A circle drawn clockwise and a circle drawn anticlockwise may or may not be, depending on the game, and the question is of a different kind: it is a design decision about the vocabulary and not a tolerance for noise. For a given library the *invariance* of a gesture is the set of transformations that map it to a stroke that should be recognized as the same gesture.

The relevant transformations act on trajectories. The elements considered are translation, uniform scaling within an allowed range, rotation by an angle within an allowed range, reflection about an axis, reversal of the traversal direction, and, for closed shapes, a cyclic shift of the starting point. A gesture library specifies for each gesture the *allowed set* of these transformations that preserves its identity. The set is a group only when it contains no bounded range, and the distinction matters for what can be proved about it (below).

12.2 Three notions of distance

Earlier drafts used a single symbol, d_H , for three different things. They are separated here, because only the first has the properties of a metric and the other two should not be credited with them. Throughout, d is the mean pointwise Euclidean distance between two normalized point sequences of N points,

$$d(A, B) = \frac{1}{N} \sum_{k=1}^N \|A_k - B_k\|,$$

and the units of every distance in this chapter are normalized-stroke units, in which the larger side of the bounding box of a stroke has length one.

Transformation	Default	Reason to change
Translation	Always allowed	None; the playfield has no privileged position for shape
Uniform scale	Allowed within $[0.4, 2.5]$ of the reference size	A narrower range for gestures defined by size
Rotation	Not allowed	Allow only if the shape is rotation-defined, such as a plain line
Reflection	Not allowed	Chirality distinguishes C from reverse C, S from Z
Reversal of direction	Not allowed	Direction matters for tracing practice and motor learning
Start-point shift	Allowed for closed shapes only	A circle has no privileged start

Table 12.1: Default invariances. Each is configurable per gesture, and each configuration is part of the library data and not of the recognizer code.

The quotient pseudometric

Let H be a *group* of transformations of point sequences that are isometries of d , so that $d(hA, hB) = d(A, B)$ for every $h \in H$. Rotation by an arbitrary angle, reflection, reversal of the index order, and, for closed shapes, cyclic shifts of the starting point are all of this kind, and so are the groups they generate. The distance under H is

$$d_H(A, B) = \min_{h \in H} d(hA, B),$$

where the minimum is attained because H is compact. A bounded range of rotation, such as $\pm 15^\circ$, is *not* a group, since composing two admissible rotations can leave the range, and it is therefore excluded from this definition.

Proposition 12.1 (Pseudometric under a common group). *If H is a group of isometries of d , then d_H is symmetric, satisfies $d_H(A, A) = 0$, and obeys the triangle inequality, so it is a pseudometric.*

Proof. Symmetry: $d(hA, B) = d(h^{-1}hA, h^{-1}B) = d(A, h^{-1}B) = d(h^{-1}B, A)$ for every h , and h^{-1} ranges over H as h does, so $d_H(A, B) = d_H(B, A)$. Since the identity belongs to H , $d_H(A, A) \leq d(A, A) = 0$. For the triangle inequality choose h_1 and h_2 attaining $d_H(A, B)$ and $d_H(B, C)$. Then

$$d_H(A, C) \leq d(h_2h_1A, C) \leq d(h_2h_1A, h_2B) + d(h_2B, C) = d(h_1A, B) + d(h_2B, C),$$

where the equality uses the invariance of d under h_2 , and the first inequality uses $h_2h_1 \in H$. $\square$

The proposition is a statement about a common group acting on a common domain. It does not by itself describe a library in which different gestures declare different invariances, and in particular it does not apply to the pair of a closed shape, which admits start-point shifts, and an open stroke, which does not. The quotient d_H is used where the whole library shares one group, and as the reference against which the other two quantities are checked: when every gesture declares the same group H , the matching cost and the confusability below both reduce to d_H .

Remark 12.1 (The closed-form rotation angle). For centered point sequences a_i and b_i in the plane, the angle θ^* minimizing the *sum of squared* distances $\sum_i \|R(\theta)a_i - b_i\|^2$ is

$$\theta^* = \text{atan2}\left(\sum_i (a_{i,x}b_{i,y} - a_{i,y}b_{i,x}), \sum_i (a_{i,x}b_{i,x} + a_{i,y}b_{i,y})\right),$$

so that no iterative search is required.^[D] The objective as a function of θ is a constant minus $2\rho \cos(\theta - \theta^*)$ for some $\rho \geq 0$, which decreases with the circular distance from θ^* , so that minimization over an interval of angles is exact for squared distance and is attained at θ^* when it lies in the interval and otherwise at the nearest endpoint. For the mean of unsquared distances used by d the same angle is not in general the minimizer. Evaluating d at θ^* therefore gives an *upper bound* on the minimum over rotations, which is what the reference implementation computes and what this specification calls the rotation-aligned cost. It equals d_H exactly if d is replaced by the root-mean-square distance, for which Proposition 12.1 applies verbatim. The baseline keeps the mean distance, and the difference between the two is a quantity to be reported in the calibration of Chapter 24.

The gesture-specific matching cost

A library assigns to each gesture g an *allowed set* $\mathcal{S}_g$ of transformations of the input stroke (Table 12.1). The set contains the identity. It may contain a bounded interval of rotation angles, a bounded interval of scale, and a choice of reflection, reversal and start-point shifts, and it is *not* required to be a group. The matching cost of a stroke G against a template T of gesture g is

$$c_g(G, T) = \min_{s \in \mathcal{S}_g} d(sG, T),$$

and the distance of a stroke to the gesture is $d_g(G) = \min_m c_g(G, T_{g,m})$ over the templates of g . The transformation is applied to the input and the allowance is read from the *candidate*, so that the recognizer never needs to know which gesture the stroke was meant to be, which it cannot know. The cost is directional by construction. A stroke may be close to a gesture that tolerates rotation and far from one that does not, and the same two shapes can have $c_g(A, B) \neq c_{g'}(B, A)$ when the allowed sets differ. That asymmetry is a property of the vocabulary, not a defect, and no metric property is claimed for c_g . Scale is handled by normalization to a canonical size (Chapter 16) and, when a gesture permits a range of sizes, by the limit on aspect and extent in the library data, and not by minimization, since scaling is not an isometry of d .

The pairwise confusability

For the validation of a library, the question is whether some stroke could be taken for the wrong gesture, and this is a statement about both directions. For templates T_i and T_j of gestures $g_i \neq g_j$ the *confusability distance* is

$$\kappa(T_i, T_j) = \min\{c_{g_i}(T_j, T_i), c_{g_j}(T_i, T_j)\}.$$

It is the smaller of the two directional costs: it asks how close a stroke drawn exactly like one template can be brought to the other by the transformations that the *other* gesture tolerates,

Quantity	Definition and domain	Claimed properties
d_H	Minimum of d over a common isometry group H	Pseudometric (Proposition 12.1)
$c_g(G, T)$	Minimum of d over the allowed set $\mathcal{S}_g$ of candidate g , applied to the stroke	Non-negative; directional; no metric claim
$\kappa(T_i, T_j)$	Smaller of the two directional costs between templates of distinct gestures	Symmetric; non-negative; no triangle inequality

Table 12.2: The three distance notions of this specification. Recognition uses c_g , library validation and the separation floor use κ , and d_H is used where a common group makes the quotient meaningful.

and it takes the more dangerous direction. It is symmetric, non-negative and zero for a template compared with itself. It does not in general obey the triangle inequality, and the specification makes no claim that it does. When all gestures share a group H , $\kappa = d_H$.

Requirement 12.1 (Library validity). A gesture library **MUST** be rejected at load time if any pair of templates T_i, T_j of distinct gestures satisfies $\kappa(T_i, T_j) < \delta_{\min}$. The check detects, for example, a library that contains both a gesture and its mirror image while one of them declares reflection. The check is executable for any supplied value of $\delta_{\min}$; the value itself is a calibration constant^[C] in normalized-stroke units, and a library is valid only relative to the value that was used.^[S]

12.3 Examples of collision

The need for configurable invariance is clear from familiar pairs. Under rotation by 180° the digits 6 and 9 coincide, and under rotation by 90° the letters N and Z nearly do. Under reflection the letter S becomes a Z-like curve and C becomes a reversed C. Under reversal of direction a line from left to right is the same path as a line from right to left, and a circle drawn clockwise coincides with one drawn anticlockwise. The default of the library is conservative, with all of these off, because a stricter definition of identity costs the player only a requirement to draw the intended shape, while a looser one risks silently merging two gestures that the game treats as distinct. Where a gesture benefits from invariance, as with plain lines in the early levels where direction is not the lesson, the library switches it on for that gesture only.

Chapter 13

Curvature and Directional Information

13.1 Features that survive noise

Pointwise comparison treats every sample as equally informative, which is not true of a stroke. Along a straight segment the samples carry little information beyond the endpoints, while at a corner a few samples determine the character of the gesture. The turning angles τ_i and the discrete curvature κ_i defined in Chapter 9 concentrate the information at the places where the path changes direction, and they are invariant to translation and, up to scaling of κ , to scale, which makes them natural features. Their weakness is sensitivity to noise: a differentiated signal amplifies jitter in the samples, so that curvature computed from raw samples is unreliable and must be computed from a smoothed and resampled path.

13.2 Landmarks

A *landmark* is a point of a gesture at which the path has a locally informative feature: a corner, an inflection, an endpoint, a crossing, or a point of locally extreme curvature. A reliable procedure for corners is to compute the turning angle on the resampled path with a window w , so that the turn at index i is measured between the headings over $[i - w, i]$ and $[i, i + w]$, and then to keep the local maxima of the absolute turning that exceed a threshold τ_c , after non-maximum suppression within a distance w . Inflections are the sign changes of the smoothed curvature. Crossings are found by testing the segments of the resampled polyline pairwise, excluding adjacent pairs. The simplification of a polyline by the Douglas–Peucker procedure [6]^[D] provides an alternative route to corners: the vertices retained at a tolerance ϵ are the geometrically informative ones. The ordered sequence of landmarks, with their types and positions, is the representation used by the geometric feature recognizers of Chapter 19.

13.3 Informative portions

Which part of a gesture matters most for recognition is an empirical question that can be answered by occlusion. For a library Λ and a gesture T_i , remove a contiguous portion of

T_i by replacing it with the straight chord between its endpoints, and measure the change in the distance to the nearest competitor. A portion whose removal brings T_i close to another gesture is informative, and a portion whose removal changes nothing is redundant. Computing this for all portions yields an *informativeness profile* $\iota_i(s)$ along the arc length of each template, and the profile is used in three places: to choose the order in which parts of a gesture are drawn when the player is taught to abbreviate (Chapter 31), to decide where a partial stroke becomes unambiguous (Chapter 22), and to study how much information can be removed before recognition fails (Chapter 68).

Chapter 14

Temporal Structure

14.1 What time contributes

The execution of a stroke carries information beyond the path. The speed profile $v(s)$ along the path, the presence and length of pauses, the acceleration at the start and the deceleration at the end, and the overall rhythm are all measurable, and they differ between people and between occasions for the same person. The question is which of these should influence recognition. The answer adopted is conservative: recognition in the first milestones is purely geometric and time-invariant, in accordance with Principle 5 and with the aim of accommodating different users, since tempo varies far more across people than shape does.

Time enters the system in other roles. The pointer events supply timestamps by which the response time is measured for scoring. Pauses are used by input adapters to decide when a stroke has ended on devices that have no release event, such as a gaze tracker (Chapter 61). In the word keyboard, a deliberate dwell on a key is an optional way to indicate a repeated letter, as discussed in Chapter 33. Finally, the fluency measures of Chapter 30 are computed from the execution and are among the primary outputs of the learning research.

14.2 Measures of execution

The principal execution measures are as follows. The duration is $T = t_N - t_1$. The mean speed is ℓ/T , and the peak speed is $\max_i v_i$. A *pause* is a maximal interval in which the speed stays below v_{pause} for longer than t_{pause} , with starting values of 20 logical units per second and 120 milliseconds. The smoothness of the movement can be summarized by a dimensionless jerk, discussed in Chapter 30. A well-documented regularity of human drawing is that the speed along a curve decreases where the curvature is high, quantified by the two-thirds power law, which states that angular velocity is proportional to the two-thirds power of curvature, or equivalently that tangential speed varies as the negative one-third power of curvature [18].^[D] The regularity is useful for two purposes in this project: as a way of testing whether a recorded stroke has the kinematic character of a human movement and not a synthetic one, and as a prediction that curvature will cost time in the execution model of Chapter 11.

14.3 Which temporal properties to ignore

A recognizer that depended on speed would penalize a slow beginner and a hurried expert alike, and would make the distinction between gestures dependent on how they are drawn instead of on what they are. The specification therefore requires that two path-equivalent strokes (Definition 9.1) produce the same recognition outcome in the baseline recognizer, up to the effect of finite sampling. Temporal information may be used by an optional recognizer variant, labeled as such in the results, and the distinction between the two is preserved in the evaluation (Chapter 69).

Requirement 14.1 (Time invariance of the baseline). The baseline recognizer **MUST** depend on the planar coordinates of the stroke only, and **MUST** yield the same distance for two strokes that are path-equivalent and differ only in timing, within a numerical tolerance of 10^{-6} logical units after resampling.^[5]

Chapter 15

Gesture Complexity

15.1 A multidimensional measure

The complexity of a gesture is not one thing, and a single scalar built from path length would mislead in the ways Chapter 10 describes. The measure used here has five components, each normalized to lie between zero and one, combined linearly. For a gesture G ,

$$C(G) = \alpha L(G) + \beta T(G) + \gamma X(G) + \delta R(G) + \eta A(G),$$

with non-negative weights $\alpha, \beta, \gamma, \delta, \eta$ summing to one. The terms are measures of path length, turning complexity, self-intersection, reversal and ambiguity. Each is defined with a saturating transformation $\varsigma_c(x) = x/(x + c)$ that maps the non-negative reals into $[0, 1)$ and has a constant c for the value of x at which the transformed value is one half.

Definition 15.1 (Components). Let $D(G)$ be the diagonal of the bounding box of G , let τ_i be the turning angles of the resampled path, let $\text{cross}(G)$ be the number of proper crossings between non-adjacent segments of the resampled polyline, and let $\text{rev}(G)$ be the number of turning angles with $|\tau_i| \geq 5\pi/6$. The components are

$$\begin{aligned} L(G) &= \varsigma_{c_L}(\ell(G)/D(G) - 1), & T(G) &= \varsigma_{c_T}\left(\frac{1}{2\pi} \sum_i |\tau_i|\right), \\ X(G) &= \varsigma_{c_X}(\text{cross}(G)), & R(G) &= \varsigma_{c_R}(\text{rev}(G)), \\ A(G) &= \exp(-\text{sep}_\Lambda(G)/s_0), \end{aligned}$$

where $\text{sep}_\Lambda(G) = \min_{T \in \Lambda, T \neq G} \kappa(G, T)$ is the confusability distance from G to its nearest competitor in the library and $s_0 > 0$ is a scale constant.

The length term subtracts one so that a straight line, which has $\ell = D$, has zero length complexity. The ambiguity term depends on the library and not on the gesture alone, which is intentional: the same shape is harder to use in a library that contains a near neighbor. The default constants are $c_L = 1$, $c_T = 1$, $c_X = 1$, $c_R = 2$ and $s_0 = \delta_{\min}$, and the default weights are $\alpha = 0.2$, $\beta = 0.25$, $\gamma = 0.15$, $\delta = 0.15$ and $\eta = 0.25$. These are design decisions that provide a usable ordering before data exist.^[S]

15.2 Computed example

The definitions apply unchanged to word trajectories on the keyboard, where D is replaced by the fixed scale of the key width. Part VI uses the same components, and the quantities in Table 35.1 and Figure 34.1 are L -type path length, total turning, crossing count and reversal count computed from the idealized polyline of each word. These raw quantities, before the saturating transformation, are reported so that they can be inspected directly.

15.3 From intuition to data

The weights should not remain intuitive guesses. Let $y_{ug} \in \{0, 1\}$ record whether player u succeeded on a presentation of gesture g , and let $\phi(g) = (L, T, X, R, A)$ be the component vector. A logistic mixed-effects model

$$\text{logitPr}(y_{ug} = 1) = \mu + \theta_u - \mathbf{w}^\top \phi(g)$$

with a random effect θ_u for the player gives an estimate of $\mathbf{w}$, and normalizing the estimated weights to sum to one yields data-informed values of $(\alpha, \dots, \eta)$. The same data support estimating completion time with a model of the form in Chapter 11. The refit is a recurring task of the research program (Chapter 66), and the library records the version of the weights used to compute each difficulty value so that changes are traceable.^[5]

Part IV

Recognition Algorithms

This part defines how a stroke is turned into a decision. It begins with the normalization that makes strokes comparable, describes a deterministic baseline recognizer, and then develops three families of alternatives: elastic alignment, geometric features, and a probabilistic formulation that supplies calibrated confidence. The later chapters of the part treat rejection, early recognition, personalization and evaluation. Every recognizer in the part implements the same interface (Chapter 50) and is evaluated on the same recordings, so that the choice between them is an empirical matter.

Chapter 16

Input Normalization

16.1 The pipeline

Normalization converts a raw gesture of arbitrary length, position and size into a fixed-length representation in a canonical frame. It is a pipeline of pure functions, each taking a gesture and returning a gesture, with no hidden state, and the pipeline as a whole carries a version identifier. The stages, in order, are conditioning, smoothing, resampling, centering, scaling and optional rotation.

Conditioning removes samples that do not carry information: consecutive samples at the same position, and samples with a non-increasing timestamp, which can arise from event coalescing. *Smoothing* is optional and applies a low-pass filter to reduce jitter. For a pointing device the adaptive one-euro filter has the useful property that it reduces jitter at low speed and lag at high speed [5]^[D]; the baseline pipeline uses a simple three-point weighted average with weights $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ applied to interior points, and the filter variants are alternatives to be compared. *Resampling* replaces the stroke by N points equally spaced along its arc length; with the common choice $N = 64$ this suffices to represent the gestures of the vocabulary and is the value used by comparable recognizers [32].^[D] *Centering* translates the points so that their centroid is the origin. *Scaling* divides the coordinates by a size measure of the stroke. *Rotation* is applied only when the invariance group contains rotations.

16.2 Resampling

Let a conditioned stroke have points $p_1, \dots, p_M$ and total length ℓ . The resampled stroke has points $q_1, \dots, q_N$ with $q_k = \gamma(\frac{k-1}{N-1}\ell)$, where γ is the piecewise-linear arc-length parameterization of the stroke. The first and last points are therefore preserved exactly, and the result is independent of the original sampling density. A stroke with $\ell = 0$ cannot be resampled and is rejected before this stage. The algorithm is given in Appendix C.

16.3 Scale normalization

The size of a stroke is measured by $S(G) = \max(w, h)$ where w and h are the width and height of its bounding box, and the normalized coordinates are $q_k/S(G)$, so that the larger side of the bounding box has length one. Uniform scaling by the larger side preserves the aspect ratio, which is essential for distinguishing a line from a circle or an oval from a circle, and it is well defined for a straight line, whose bounding box has zero extent in one direction. Scaling each axis independently to a unit square, as some recognizers do, destroys that information and makes a horizontal line indistinguishable from a diagonal. A stroke with $S(G)$ below $S_{\min}$ is rejected as too small to normalize (Chapter 6).^[S]

16.4 Coordinate transformations

Pointer events deliver coordinates in CSS pixels relative to the viewport. The transformation to logical units is composed of a translation by the position of the playfield element, a scaling by W_p/width in CSS pixels, and no rotation, since the drawing surface is not rotated. On a device that changes orientation, the transformation is recomputed and strokes in progress are discarded. The device pixel ratio does not enter the logical coordinates at all, as stated in Requirement 4.1. For a keyboard overlay (Part VI) there is a second, fixed transformation from logical coordinates to key widths, determined by the layout of the displayed keyboard.

Requirement 16.1 (Preserve and version). Every stage of the normalization pipeline **MUST** be a pure function of its input and its parameters. The pipeline **MUST** be identified by a version string recorded with every recognition result, and the raw gesture **MUST** be retained alongside any normalized form.^[S]

Chapter 17

Baseline Template Matching

17.1 The recognizer

The baseline is the simplest recognizer that is useful. It is deterministic, it has no learned parameters other than the templates, and it runs entirely in the browser. The library Λ contains for each gesture g one or more templates $T_{g,1}, \dots, T_{g,m_g}$, each already normalized by the same pipeline as the input. Given a normalized stroke G with N points, the distance to a template is the mean pointwise Euclidean distance,

$$d(G, T) = \frac{1}{N} \sum_{k=1}^N \|G_k - T_k\|,$$

and the distance to a gesture is the smallest matching cost (Chapter 12) to any of its templates, where the cost minimizes d over the allowed set $\mathcal{S}_g$ that the candidate gesture declares,

$$d_g(G) = \min_m c_g(G, T_{g,m}), \quad c_g(G, T) = \min_{s \in \mathcal{S}_g} d(sG, T).$$

The cost is directional: each candidate applies its own allowance to the input, and no claim is made that $d_g(G)$ is a metric in G and g . The ranked list of gestures by $d_g(G)$, with ties broken by library order, is the output of the matching stage. It is the *geometric* ranking, and it is distinct from the posterior ranking of Chapter 20, which can differ from it once priors or gesture-specific tolerances enter. The decision stage that turns the ranking into an outcome is the subject of Chapter 21.

17.2 Cost and reproducibility

For a library of K gestures with m templates each, one recognition takes $O(KmN)$ operations before the invariance search, which for the first milestone ($K = 20$, $m = 1$, $N = 64$) is on the order of a thousand distance evaluations. Rotation invariance with the closed-form angle of Chapter 12 adds a constant factor, and reflection and reversal each double the count. The cost is negligible on any device that runs a modern browser, and Chapter 54 sets the budget formally.

The baseline is reproducible in principle because it contains no randomness. In practice, floating-point results depend on the implementation of elementary functions. The ECMAScript specification permits implementation-approximated results for transcendental functions such as sine, cosine and arctangent, so two engines may differ in the last digits.^[D] The consequence is that the specification asks for reproducibility to a tolerance and not to the bit. Fixtures (Appendix H) record expected distances to a tolerance of 10^{-9} and record the margin by which each decision exceeds its threshold, so that a decision that depends on a rounding difference is detected as a fragile fixture and not as a failure of the implementation.

Requirement 17.1 (Baseline determinism). Given the same normalized stroke, library and parameters, the baseline recognizer **MUST** return the same ranking on every conforming implementation, and its numerical distances **MUST** agree to within 10^{-9} .^[S]

Chapter 18

Dynamic Time Warping

18.1 Elastic alignment

The pointwise distance of Chapter 17 compares the k th point of the stroke with the k th point of the template. It is correct when the two have been resampled to correspond by arc length, and it is sensitive to local misalignment: a player who draws a corner slightly early shifts every subsequent point. Dynamic time warping (DTW) relaxes the correspondence, allowing a point of one sequence to be matched to several points of the other, subject to monotonicity [24].^[D] For sequences $a_1, \dots, a_n$ and $b_1, \dots, b_m$ with local cost $c(i, j) = \|a_i - b_j\|$, define the accumulated cost by the *symmetric* recursion, in which a diagonal step carries twice the local cost,

$$D(i, j) = \min\{D(i-1, j) + c(i, j), D(i, j-1) + c(i, j), D(i-1, j-1) + 2c(i, j)\},$$

with $D(0, 0) = 0$ and $D(i, 0) = D(0, j) = \infty$ for $i, j > 0$. The DTW distance is $D(n, m)/(n + m)$.

The weights are chosen so that the normalizer is exact and not a convention. Every admissible path from $(1, 1)$ to (n, m) accumulates local costs whose weights sum to $n + m$: a horizontal or vertical step increases $i + j$ by one and carries weight one, a diagonal step increases $i + j$ by two and carries weight two, and the first cell, reached by a diagonal step from $(0, 0)$, carries weight two. Hence $D(n, m)/(n + m)$ is a weighted average of local costs along the selected path, whatever the number of cells in that path. With the unweighted recursion, by contrast, paths of the same inputs have different numbers of cells, between $\max(n, m)$ and $n + m - 1$, and a fixed denominator $n + m$ is not an average over the path; it also places the value on a scale about half that of the pointwise distance. For $n = m$ the symmetric form agrees with the pointwise distance of Chapter 17 on the diagonal path, and since that path is admissible whenever the band contains the diagonal, the DTW distance is never larger than the pointwise distance of the same two sequences.^[D] The two recognizers are therefore on the same scale and can share thresholds as a starting point.

18.2 Constraints and complexity

Unconstrained DTW takes $O(nm)$ time and, if only the final value is needed, $O(\min(n, m))$ memory by keeping two rows. A global constraint restricts the alignment to a band around the diagonal. The Sakoe–Chiba band admits only cells with $|i - j| \leq w$, which reduces the time to $O(nw)$ and prevents the pathological alignments in which a long run of one sequence is matched to a single point of the other.^[D] A band width of about ten percent of the sequence length is a typical starting value and is tuned on recorded strokes. Lower bounds such as that of Keogh allow most templates to be discarded without computing the full alignment, which matters when the library is a lexicon of tens of thousands of words but not when it is twenty shapes [15].^[D]

18.3 When warping helps and when it does not

After arc-length resampling, DTW no longer compensates for differences in drawing speed, because speed has already been removed. What it compensates for is geometric misalignment, such as a corner placed earlier or later along the stroke and differences in the proportions of the parts of a shape, and it is applicable without resampling when the strokes are compared at their native sampling. The main failure mode is excessive flexibility. A warping path is free to match a long stretch of one stroke to a short stretch of another, and with a loose constraint two different shapes can acquire a deceptively small distance because a mismatched part is absorbed by warping. The effect is most harmful for shapes that differ in the number of repetitions, where a circle drawn once and a circle drawn twice can align at low cost. A tight band, a penalty on non-diagonal steps, or comparison on a feature sequence that includes the heading, in place of position alone, are the standard mitigations, and the comparison of DTW with the baseline is a primary experiment of Chapter 69.^[H]

Chapter 19

Geometric Feature Recognition

19.1 Recognizing by structure

A different approach discards the pointwise correspondence and describes a stroke by a small number of geometric facts about it. The turning function of a polygonal shape, which records the heading as a function of arc length, was introduced as an efficiently computable shape metric and has the properties of invariance to translation and scale, and a simple behavior under rotation, which adds a constant to the function [2].^[D] A recognizer built from the landmarks of Chapter 13 encodes a stroke as a string over a small alphabet whose symbols describe its structure: the compass direction of each straight segment, the sign of each turn, the presence of a closed loop, and the existence of a crossing. A gesture library then consists of such strings, and a stroke is recognized by an edit distance between the string of the stroke and each string of the library, with substitution costs that reflect how easily two symbols are confused, for example a higher cost for replacing a left turn by a right turn than for replacing east by northeast.

19.2 Comparison with pointwise matching

Feature recognizers have two attractive properties. They are interpretable, because a rejection can be explained in terms of a missing turn or an extra crossing, which is valuable for a game that must tell the player why a stroke failed. They are tolerant of the proportions of a shape, since only the structure is compared. They also have two weaknesses. They are brittle at their thresholds, since a corner of 44° is a straight segment and one of 46° is a corner when the threshold is 45° , so that tiny changes of the stroke produce different strings; and they discard metric information that the pointwise distance retains, such as the relative length of two sides of an L. The practical resolution, to be tested in Chapter 69, is a hybrid in which the feature string proposes a short list of candidates and a pointwise distance chooses among them.^[H]

Chapter 20

Probabilistic Recognition

20.1 From distances to probabilities

A recognizer that returns only a ranking cannot say how sure it is, and the game needs to know. The probabilistic formulation treats recognition as inference. For a stroke G and a candidate target w (a gesture, or later a word), the posterior probability is

$$P(w | G) \propto P(G | w) P(w),$$

which separates the geometric evidence $P(G | w)$ from the prior expectation $P(w)$ of how likely each target is a priori. In the first milestone the prior is uniform over the candidate set, and the posterior reduces to the normalized likelihood. In the word setting of Part VI the prior is a language model, and the interplay of the two terms is a central subject (Chapter 39).

20.2 Likelihood model

The likelihood is derived from the distance. A simple and defensible model treats the normalized stroke as the template plus isotropic Gaussian noise of standard deviation σ_w , so that, with $d_w = d_w(G)$ the matching cost of Chapter 17 to the nearest template of w and n_{eff} an effective dimension,

$$\log P(G | w) = -n_{\text{eff}} \log \sigma_w - \frac{d_w^2}{2\sigma_w^2} + \text{const.}$$

A shared σ is the initial choice, and per-gesture and per-user values of σ_w are estimated from recorded strokes, so that gestures that people draw consistently, such as straight lines, are held to a tighter tolerance than gestures that people draw variably, such as spirals. The effective dimension and the Gaussian assumption are modeling choices and not facts about hands, and their adequacy is checked by calibration (below). Combining the terms with a weight $\lambda \geq 0$ on the prior,

$$\log P(w | G) = \log P(G | w) + \lambda \log P(w) - \log Z,$$

where Z normalizes over the candidate set $\mathcal{C}$, so that the posterior is a softmax over $\mathcal{C}$. The weight λ makes explicit how much the prior is allowed to override the geometry, and it is reported with every experiment.

20.3 Calibration

A probability is useful only if it is calibrated, meaning that among strokes assigned posterior p about a fraction p are correct. Models of this kind are often miscalibrated, and a standard remedy is to divide the log-posterior by a temperature Θ fitted on held-out data [12].^[D] The expected calibration error, computed by binning the maximum posterior and comparing the mean confidence with the accuracy in each bin, is one of the evaluation measures of Chapter 24.

Chapter 21

Recognition Confidence

21.1 The decision rule

The decision stage converts the ranked and scored candidates into one of three outcomes. The system should be able to say that a stroke is ambiguous and not force a choice, and it should be able to say that a stroke belongs to no candidate. Let $\mathcal{C}$ be the candidate set. Each candidate g has a matching distance d_g (Chapter 17) and a posterior p_g (Chapter 20), and the two rankings need not agree, because priors and gesture-specific tolerances σ_g can promote a candidate that is not the geometrically nearest. The rule therefore names the candidates explicitly. The *posterior winner* is

$$g^* = \arg \max_{g \in \mathcal{C}} p_g,$$

with ties broken by the smaller d_g and then by library order, and the *nearest candidate* is $g_{\text{nn}} = \arg \min_{g \in \mathcal{C}} d_g$. Let $p^* = p_{g^*}$ and let p_2 be the largest posterior among the other candidates, with $p_2 = 0$ when $|\mathcal{C}| = 1$. Three thresholds govern the decision: an absolute distance gate d_{max} , a posterior threshold τ_{acc} and a margin $\mu > 0$.

Definition 21.1 (Decision rule). The outcome for a stroke and a candidate set $\mathcal{C}$ is

$$\text{outcome} = \begin{cases} \text{REJECTED}(\text{no_candidates}) & \text{if } \mathcal{C} = \emptyset, \\ \text{REJECTED}(\text{too_far}) & \text{if } d_{g^*} > d_{\text{max}}, \\ \text{ACCEPTED}(g^*) & \text{if } d_{g^*} \leq d_{\text{max}}, p^* \geq \tau_{\text{acc}} \text{ and } p^* - p_2 \geq \mu, \\ \text{AMBIGUOUS}(\{g : p_g \geq p_{\text{min}}\}) & \text{otherwise.} \end{cases}$$

Acceptance requires both posterior confidence and geometric support *for the same candidate*, g^* . When $g^* \neq g_{\text{nn}}$ the result records a prior override, so that the cases in which the prior decided against the geometry can be counted and audited.

The degenerate cases are defined and not left to the implementation. An empty candidate set yields a rejection with its own reason, since there is nothing to accept. With a single candidate the softmax gives $p^* = 1$ and $p_2 = 0$, so the posterior conditions hold trivially and the absolute gate alone decides between acceptance and rejection; this is correct but means

that a one-balloon screen is only as strict as $d_{\max}$, and it is a reason for the vocabulary-wide assessment of the next section. Exact ties in posterior give $p^* - p_2 = 0 < \mu$ and are therefore ambiguous. A stroke that fails normalization is rejected before scoring with the reason `malformed` or `too_short`, and never reaches this rule.

21.2 Active candidates and the full vocabulary

The candidate set is part of the input to the rule, and the outcome depends on it. A stroke that is accepted when three balloons are on the screen can be ambiguous when twenty different gestures are present, because the competitors that make it ambiguous were absent. This is a legitimate source of difficulty, since a crowded screen is a harder task, but it makes the outcome an unreliable measure of what the player has learned: a gesture can appear mastered only because its neighbors were not there.

The specification therefore distinguishes two recognitions of the same stroke. *Active recognition* uses the gestures of the balloons currently on screen as $\mathcal{C}$, and its outcome drives the game: the balloon that pops, the combo, the score. *Vocabulary recognition* uses every gesture that the player has been introduced to as $\mathcal{C}$, and its outcome is used for the competence estimates of Chapter 29, for the retention and transfer measures of Chapter 32, and for the research recordings. A stroke is a *transferable success* if it is accepted as the intended gesture under vocabulary recognition, and a *context-assisted success* if it is accepted under active recognition but not under vocabulary recognition. Both rates are reported, and the second is a direct measure of how much of the player's apparent performance is owed to the sparseness of the screen. Both recognitions are cheap at the library sizes of the first milestone, and the result records which scope produced it.

Requirement 21.1 (Two scopes). Every stroke in a play session MUST be scored against the active candidate set for gameplay, and the competence model, the retention measures and the stored recordings MUST use the outcome against the vocabulary candidate set. The scope of every result MUST be recorded.^[S]

21.3 Why an absolute gate is necessary

The posterior is a normalized quantity. A softmax over candidates assigns probabilities that sum to one regardless of how far the stroke is from every candidate, so that a meaningless scribble that happens to be slightly nearer to one template than to the rest can receive a posterior close to one. A decision based only on p_1 and the margin would accept it. This is the open-set problem: the stroke may belong to none of the candidates. The absolute gate $d_{\max}$ handles it by rejecting any stroke that is not close enough to any template, independently of the competition among the candidates. An alternative that makes the rejection part of the probabilistic model adds a null class to $\mathcal{C}$ with a broad likelihood, whose posterior is the probability that the stroke is not any gesture. The two are equivalent in intent; the gate is simpler to explain and to set from data, and it is the specified behavior.^[S]

21.4 Setting thresholds and reporting

The thresholds are set from the trade-off between two kinds of error. Accepting a wrong gesture frustrates a player by popping the wrong balloon, and declining a correct gesture frustrates a player by ignoring a good stroke. The trade-off is summarized by the risk-coverage curve of selective classification, which plots the error rate among accepted strokes against the fraction of strokes accepted as the threshold varies [11].^[D] In the game, the costs are asymmetric. A wrong acceptance may destroy a balloon that the player did not intend, which is worse than an ambiguous outcome, and the thresholds are therefore set to produce a low error among accepted strokes at the cost of a higher ambiguity rate, with starting values $\tau_{\text{acc}} = 0.8$, $\mu = 0.3$, $p_{\text{min}} = 0.15$ and d_{max} set at the ninety-fifth percentile of genuine strokes in the calibration set.^[S] All four are calibration constants^[C] in the units of the distance they gate (normalized-stroke units for d_{max} , probability for the others), and Appendix A lists the procedure that fixes each.

Requirement 21.2 (Explicit non-acceptance). A recognizer **MUST** be able to return the Ambiguous and Rejected outcomes of Definition 21.1, **MUST** report the scores that produced the outcome, and **MUST NOT** select a gesture when the acceptance conditions fail.^[S]

Chapter 22

Partial Trajectory Recognition

22.1 Recognizing before the stroke ends

The question of this chapter is whether a stroke can be recognized before it is finished, so that the remaining movement is unnecessary. Long-term users of gesture keyboards report that a long word is often selected when only its early or distinctive portion has been drawn, and that this property is part of what makes the technique fast.^[O] The claim is plausible, because a lexicon is a set of paths that diverge from one another, and after enough of a path has been drawn only one word remains consistent with it. The problem is to define when the game or keyboard may commit.

22.2 Prefix distance

For a stroke prefix $G_{1:k}$ and a template T , the prefix distance is the minimum, over prefixes of the template, of the distance between the stroke prefix and that template prefix,

$$d_{\text{pre}}(G_{1:k}, T) = \min_m d(\rho(G_{1:k}), \rho(T_{1:m})),$$

where ρ is resampling to a common number of points. An efficient computation uses open-end dynamic time warping, in which the alignment of the stroke must start at the beginning of the template but may end anywhere. The scale of the stroke is unknown while it is incomplete, which is a difficulty for shape-channel recognition and not for a recognition against fixed coordinates, such as a keyboard whose layout is displayed; for that reason early recognition is most natural in the word setting, and the gesture setting uses a coarse estimate of the scale or a candidate set small enough that the ambiguity does not arise.

22.3 Commit rule

A commit rule decides when to accept a prefix. The one specified requires that the leading candidate hold its position and its confidence. For a prefix at step k with posterior $p_1^{(k)}$ and margin $p_1^{(k)} - p_2^{(k)}$, the stroke commits to the leading candidate g at the first k for

which, over h consecutive updates, g has been the leading candidate, $p_1^{(k)} \geq \tau_{\text{early}}$, and $p_1^{(k)} - p_2^{(k)} \geq \mu_{\text{early}}$. The threshold τ_{early} is set higher than τ_{acc} because an early commit cannot be revised, and h guards against a momentary alignment. The *earliness* of a commit is the fraction $f = s_k / \ell_g$ of the *template* consumed, where s_k is the arc length drawn so far and ℓ_g is the length of the template of the committed gesture. This is deliberately a template fraction. The length that the player intended to draw is not observable once the system commits, because the stroke is cut short, so a fraction of the intended movement can be measured only in a *shadow* mode in which the commit decision is logged but not enforced and the player finishes the stroke. Experiments that compare earliness with the movement that would have followed use shadow mode.

22.4 A property of the library

Early recognizability is a property of the library as well as of the recognizer. For a library Λ of templates and a fraction $f \in (0, 1]$, let $T[f]$ denote the prefix of T of fraction f of its length, and treat $T_i[f]$ as the idealized partial stroke of a player who follows T_i exactly up to that point. The recognizer does not compare it with the same fractional prefix of a competitor. It scores it with the prefix distance d_{pre} defined above, which searches over *all* prefixes of the competitor with an open end, so that a competitor is a threat if the partial stroke fits any of its initial portions. The *prefix separation* of T_i at f is therefore

$$\text{ps}_i(f) = \min_{j \neq i} d_{\text{pre}}(T_i[f], T_j),$$

computed with the same alignment, band and distance as the recognizer, and the *earliest unambiguous fraction* is

$$\text{EUF}(T_i) = \inf\{f : \text{ps}_i(f') \geq \delta \text{ for all } f' \geq f\}.$$

A template with a small EUF can be recognized early; one with $\text{EUF} = 1$ cannot be recognized before the stroke is complete. Defined this way, EUF is a property of the library that uses the same operation as the recognizer, and it cannot predict an early commit that the recognizer is unable to make. It is nevertheless an optimistic bound: it uses noise-free strokes, a threshold δ on distance and no prior, whereas the commit rule uses posteriors and a run of h updates, so that the observed earliness f of a commit is expected to lie above EUF and the size of the gap is itself a result. The word setting also fixes the scale and position of the stroke through the displayed keyboard, which is why the measure is defined there and not for shape gestures. The distribution of EUF over a lexicon measures how much a gesture keyboard can truncate strokes, and the question that Hypothesis 22.1 asks is how far the distribution lies below one for realistic vocabularies.

Hypothesis 22.1 (Early commit is available). For a lexicon of common English words on the default keyboard layout, the median earliest unambiguous fraction is substantially below one, so that a recognizer with a commit rule of the form above reaches a correct decision on most words before the end of the stroke, without raising the error rate among committed words by more than a stated margin.^[H]

22.5 Effects on play and on learning

Early commit has costs that are not computational. The stroke is cut off, so the player no longer controls its end, and a player who lifts early, as a habit, will be recognized less reliably under a recognizer that does not support early commit. A game that rewards early commits will teach players to abbreviate, which is a design aim (Chapter 31), and also changes what the recordings contain. Because this affects every analysis that compares strokes, the mode is recorded in the session metadata and the analyses that assume complete strokes are restricted to sessions without early commit.

Chapter 23

Personalized Gesture Recognition

23.1 Two things that can be learned

A recognizer may adapt to a user in two different ways, and the two must not be conflated. It can learn the *geometry* of the user's gestures, adjusting the templates and the tolerances so that the way this person draws a circle is recognized as a circle. It can also learn the *prior*, adjusting the probability $P_u(w)$ that the user intends each target, so that targets the user selects often are favored. Geometry adaptation changes how strokes map to shapes. Prior adaptation changes which shape is chosen when the strokes are equivocal. They have different risks and different benefits: geometry adaptation can erode distinctions (Principle 6), while prior adaptation can make rare targets unreachable. They are stored in separate structures, logged separately and can be switched off independently, and the experiments of Chapter 67 depend on this.

23.2 Template adaptation

A user-specific library Λ_u is initialized from the canonical library Λ and updated by an exponential average. The update is carried out in a fixed *canonical frame*, because interpolating between sequences is meaningful only when they are expressed in the same coordinates, and a distance that is minimized over transformations does not define such a frame. When a stroke G with a known label g is observed, and its matching cost to the nearest template of g satisfies the learning gate $c_g \leq d_{\text{learn}}$, the stroke is first aligned once to the *canonical* template T^0 of that slot by the transformation that attains the matching cost,

$$\hat{G} = s^* G, \quad s^* = \arg \min_{s \in \mathcal{S}_g} d(sG, T^0),$$

and the personalized template is then moved toward the aligned stroke by pointwise interpolation,

$$T \leftarrow (1 - \eta) T + \eta \hat{G},$$

where both are normalized with the same number of points, and η is the adaptation rate. The frame is thus fixed by T^0 and never by the current personalized template, so that

the template cannot drift by following its own alignment. If the stroke is farther than $d_{\text{new}} > d_{\text{learn}}$ from every template of g but still acceptable, it is added as a further template up to a capacity m_{max} , replacing the least recently matched. The rate η decays with the number of updates, beginning near 0.2 and settling near 0.02, so that a template is flexible at first and stable later.^[S]

23.3 Protection against accidental learning

Four protections prevent the library from learning the wrong thing. First, updates are gated on the label: only strokes whose label is known with confidence, by the prompt of a practice exercise or by an explicit confirmation, may adjust the geometry. In survival mode the label is the recognizer’s own output, which creates a feedback loop in which the system reinforces its own decisions, and adaptation is therefore disabled by default there.^[S] Second, the update is rate-limited per session, so that a bad session cannot move the library far. Third, each personalized template is held within a radius ρ_{drift} of its canonical version,

$$d(T_u, T^0) \leq \rho_{\text{drift}},$$

where d is the plain mean pointwise distance in the canonical frame, with no minimization over transformations. In this frame $d(T, T^0)$ is a norm of the difference $T - T^0$, so it is homogeneous: scaling the difference by λ scales the distance by exactly λ , and an update that leaves the ball is retracted onto it by the radial rescaling $T^0 + (\rho_{\text{drift}}/r)(T - T^0)$ with $r = d(T, T^0)$. This is a retraction onto the ball and not the nearest point, which for a norm other than the Euclidean one it need not be, but it is feasible, deterministic and idempotent. The drift bound limits how far a template may move, and it does not by itself keep two gestures apart. The separation floor is therefore checked *independently*, on the confusability distance κ of the whole personalized library (Requirement 3.1): the candidate template is accepted only if the floor holds with it in place, and otherwise the step is halved along the update direction, down to a minimum step, after which the update is rejected. No monotonicity of the floor along the step is assumed, and the returned template is checked, not inferred. Fourth, every change is reversible, because the history of updates is retained and the user can reset a gesture or the whole profile.

Requirement 23.1 (Bounded and reversible adaptation). Template adaptation **MUST** be gated on a label known independently of the recognizer, **MUST** be computed in the canonical frame of the slot, **MUST** be bounded by a drift radius from the canonical template measured in that frame, **MUST** preserve the separation floor as an independent check on κ , **MUST** be recorded as a history of updates, and **MUST** be resettable by the user. The constants η , d_{learn} , d_{new} and ρ_{drift} are calibration constants.^{[C][S]}

23.4 Prior adaptation

The prior $P_u(w)$ is estimated from counts $n_u(w)$ of the user's selections, smoothed with a Dirichlet prior centered on the population prior $P_0(w)$,

$$P_u(w) = \frac{n_u(w) + a P_0(w)}{\sum_v n_u(v) + a},$$

with concentration a controlling how many observations are needed before the personal evidence outweighs the population. The count structure is the personal dictionary of Chapter 38, and it is inspectable and editable by the user.

Chapter 24

Recognition Evaluation

24.1 Measures

A recognizer is evaluated on a labeled dataset of strokes. For a library with K gestures let C_{ij} be the number of strokes of true gesture i that the recognizer accepted as gesture j , with separate counts A_i of strokes ambiguous and J_i of strokes rejected. The *accuracy* is the fraction of strokes accepted as the correct gesture, the *false-accept rate* is the fraction accepted as a wrong gesture, and the *false-reject rate* and *ambiguity rate* are the fractions rejected and ambiguous respectively. The confusion matrix is C augmented with an ambiguity column and a rejection column. For each gesture, precision and recall are computed in the usual way from C , and the macro-average over gestures is reported alongside the micro-average to prevent frequent gestures from masking poor performance on rare ones.

Latency is measured as the time between the release event, or between the last sample for an early-commit decision, and the availability of the outcome. Computational cost is the processor time per recognition on a reference device, reported as a median and a ninety-fifth percentile over repeated runs after warm-up, because mean values hide the tail that causes a visible hitch. Calibration is measured by the expected calibration error and the risk-coverage curve introduced in Chapter 21.

24.2 Evaluation across users

Strokes by the same person are not independent observations, and treating them as independent understates the uncertainty. Two procedures follow. The first is that generalization to new users is measured by leaving out all strokes of a user at a time (leave-one-user-out cross-validation), in contrast to splitting strokes at random, which mixes a user's strokes between the training and test sets and measures performance on already-seen users. The second is that confidence intervals are computed by a cluster bootstrap that resamples users, with replacement, and then strokes within users, so that between-user variation contributes to the interval. A report that gives only the pooled accuracy is incomplete; the distribution of per-user accuracy is reported, with the minimum and the lower quartile, because the experience of the worst-served users is what an accessible game is judged by.

24.3 Benchmark datasets and procedures

A benchmark consists of a dataset, a fixed set of splits, an evaluation harness and a reporting template. The dataset is stored as one JSON object per stroke (Appendix B), with a manifest giving its checksum. Splits are generated from a recorded seed and are never regenerated. The harness takes a recognizer implementation, identified by name and version, runs it over the splits, and writes a results file with the measures above. Comparisons between recognizers use the same splits, the same candidate sets and the same tuning budget, with hyperparameters chosen by nested cross-validation inside the training folds only. The reference benchmark is built from the data collected under Chapter 64, and until such data exist the harness is exercised on synthetic strokes generated by perturbing the templates, which tests the machinery and not the recognizers.^[5]

Part V

Learning and Difficulty Progression

This part treats the player as a learner. It states what is known about the acquisition of coordinated movement, specifies a curriculum that advances by geometric feature, develops a model of player competence that selects the next challenge, and defines the measures by which improvement is recognized and attributed. The recurring concern is attribution: when the success rate rises, it must be possible to say whether the player improved or the software did.

Chapter 25

Motor Learning Foundations

25.1 What is established

Several regularities of motor skill acquisition bear directly on the design. The first is the power law of practice, which describes the time to perform a task as a decreasing power function of the number of repetitions, $T(n) = a + b n^{-c}$ with a the asymptotic time and $c > 0$ the learning rate [22].^[D] It is an empirical description with known limitations, among them that curves aggregated over many learners tend to look more like power functions than the individual curves do, and the research program fits several forms and compares them (Chapter 66). The second is the description of skill acquisition as a progression from a cognitive stage, in which the learner attends to the movement and errors are large, through an associative stage, in which the movement is refined, to an autonomous stage in which performance requires little attention [8].^[D] The third is schema theory, which proposes that what is learned for a class of movements is a relation between parameters and outcomes that generalizes beyond the movements practiced, and which motivates varying the size and the orientation of practiced gestures [26].^[D]

Two further results shape the curriculum. Practice in which the tasks vary from trial to trial, as opposed to practice in blocks of a single task, produces poorer performance during acquisition and better retention and transfer, a finding known as the contextual interference effect [27].^[D] It argues for interleaving gestures in the survival mode instead of drilling one at a time. And frequent, detailed feedback can improve performance during practice while impairing the learning that shows up in retention, because the learner comes to depend on it; this is the guidance hypothesis [25].^[D] It argues for reducing visual assistance as proficiency grows (Chapter 37).

25.2 Learning is not adaptation

The game contains two systems that can improve over time: the player and the recognizer. Improvement of the observed outcome, such as the proportion of strokes that pop a balloon, is a function of both. The distinction that organizes the research program is that between *human learning*, a change in the strokes the player produces, and *model adaptation*, a change in how strokes are interpreted.

Definition 25.1 (Identification problem). Let $y(u, t, v)$ be the success rate of user u at time t under recognizer version v . Observed improvement $y(u, t_2, v_2) - y(u, t_1, v_1)$ conflates a change of behavior ($t_1 \rightarrow t_2$) with a change of recognizer ($v_1 \rightarrow v_2$). The effects are separately identified only if designs are used in which one argument is held fixed while the other varies.

Chapter 67 gives the designs. The requirement on the game is simpler and applies from the first milestone: it must be possible to hold the recognizer fixed.

Requirement 25.1 (Frozen recognizer option). Every adaptive behavior of the recognizer **MUST** be switchable, and a session **MUST** record the identity and state of the recognizer used, so that a session can be scored again under a frozen recognizer.^[5]

Chapter 26

The Introductory Curriculum

26.1 Principle

The introduction presents simple shapes with generous time and little competition, and difficulty grows by introducing new geometric features and not by raising speed arbitrarily (Principle 2). A level is defined by the feature it introduces, by the set of gestures that exhibit it, and by a mastery criterion, and the player moves on when the criterion is met and not when a number of minutes has elapsed.

26.2 Mastery

A gesture is considered mastered when the player's estimated success probability on it is high with confidence, and when the time to complete it is comfortably within the budget. With a Beta posterior on the success probability π_g of gesture g , built from the successes and failures of recent presentations with exponential forgetting, mastery requires that the lower end of the posterior, for example its fifth percentile, exceed 0.75 and that the median response time fall below half of the time available. A level is completed when all of its introduced gestures are mastered. Gestures that were mastered are retained in the pool at a lower frequency, which gives the distributed practice that retention requires (Chapter 32). The criterion rests on the competence model of Chapter 29.

26.3 Pacing

The introduction has to be enjoyable for a player who does not know that it is an introduction. It is therefore paced by the game and not by a lesson. A new gesture first appears alone, displayed with its reference trajectory drawn faintly beside the glyph for the first appearances, and the aid fades after a few successes. Failure on the first appearances is not penalized with a lost life. After that, the gesture joins the pool and competes with the others, and the interleaving effect (Chapter 25) comes into play without any explicit change of mode.^[5]

Level	Feature introduced	Gestures added	Concurrent balloons
1	Straight stroke, horizontal and vertical	<code>line.h</code> , <code>line.v</code>	1
2	Diagonal direction	<code>line.d1</code> , <code>line.d2</code>	2
3	Single sharp corner	<code>corner.l</code> , <code>corner.v</code> , <code>corner.caret</code>	2
4	Open curvature	<code>arc.c</code> , <code>arc.d</code> , <code>hook.u</code> , <code>hook.j</code>	2
5	Closure and rotation sense	<code>circle.cw</code> , <code>circle.ccw</code>	2
6	Two corners in one stroke	<code>corner.z</code> , <code>zig.2</code>	3
7	Curvature of changing sign	<code>arc.s</code>	3
8	Loop on a path	<code>loop.l</code> , <code>loop.e</code>	3
9	Repeated reversal and nested curvature	<code>zig.3</code> , <code>spiral.in</code>	3

Table 26.1: A first curriculum over the twenty gestures of Table 10.1. Levels one to five form the introduction; levels six to nine open the intermediate family of Chapter 27.

Chapter 27

Intermediate Gesture Families

The intermediate stage introduces gestures that combine features and whose discrimination from one another is itself a task. It contains compound curves, strokes with several turns, strokes that overlap their own earlier parts, and sets of gestures that are deliberately similar.

27.1 Compound curves and multiple turns

Compound curves, such as the S-shaped arc and the loop on a path, require the player to coordinate a change of curvature sign or to produce a closed sub-shape in the middle of a stroke. They are the first gestures in which the stroke has an internal structure that must be executed in order. Multiple-turn gestures, such as the zigzags, require the player to count: two reversals and three reversals are different gestures, and the most common error is to omit or add a segment. The game can help by making the glyph show the segment count clearly, and the recognizer is tuned so that a missing segment is a rejection and not a near-match (Chapter 21).

27.2 Overlapping trajectories

A stroke that crosses itself introduces an ambiguity of order. A figure-eight can be started at the crossing or at either lobe, and drawn in two rotation senses. Only some of these variants are the intended gesture. The library therefore carries a set of admissible variants for such gestures, each a template with an invariance configuration (Chapter 12), and the glyph conveys the intended start and direction through a small marker. The presence of variants raises the question of whether admitting many variants makes a gesture too easy, and the answer is controlled by the separation floor: any variant added must keep the library above $\delta_{\min}$.

27.3 Increasingly similar shapes

The most important property of the intermediate stage is that it teaches discrimination. A circle and an oval, a V and a U, a caret and an arch, a plain corner and a corner with a

hook: each pair demands that the player attend to a feature that distinguishes them. The design device is the *contrast pair*, in which the two gestures are introduced together and made to appear in the same level, and in which the recognizer's decision between them is reported to the player with a clear indication of which candidate the stroke approached. A pair is a good contrast pair when its separation is only slightly above the floor, since then the player must draw carefully enough to make the distinction, and not so far above it that the distinction is trivial.^[5]

Chapter 28

Advanced Gesture Families

The advanced stage prepares the player for the shapes of words. A word trajectory is typically a long, multi-segment path with several sharp turns, frequent reversals and occasional self-crossings, drawn through specific positions on a keyboard, and no single primitive prepares for the whole. The advanced exercises are therefore built to exercise the relevant features, in combination and under time pressure.

28.1 Exercises

Self-intersection exercises use shapes that cross themselves one or more times, such as a figure-eight, a pretzel-like trace, and shapes with a looped tail. They train the control of a stroke that must pass over its own earlier part without hesitation. *Repeated direction change exercises* use long zigzags and sawtooth paths with five or more segments, and they train rhythm and the placement of reversals at a given distance from one another. *Intricate trajectory exercises* combine all of these in shapes derived by simplifying real word paths, so that the player meets word-like geometry before meeting words (Chapter 36). *Abbreviation exercises*, described in Chapter 31, show a full shape but accept a reduced one, and they are the point at which the curriculum starts to reward economy over copying.

28.2 Dosage

The advanced exercises are demanding, and a curriculum that presents too many together would discourage players. They are introduced one family at a time, in short sets of balloons within otherwise familiar play, and the competence model decides when each family has been seen enough (Chapter 29). The aim is that the player experiences the advanced material as an extension of the game they already know.

Chapter 29

Adaptive Difficulty

29.1 A competence model

The next challenge is chosen from an estimate of what the player can do. The estimate is computed from outcomes against the full learned vocabulary (Requirement 21.1), so that a gesture does not appear mastered merely because its neighbors were absent from the screen. The estimate is not a single level and not the total score. Both are too coarse: a total score mixes speed, luck and persistence, and a single level cannot represent a player who is fluent with lines and arcs and unreliable with reversals. The model used instead represents the player's competence as a vector over the geometric features of Chapter 15.

Let $\phi(g) \in \mathbb{R}^m$ be the feature vector of gesture g , including the component values (L, T, X, R, A) and indicators for closure and rotation sense. The probability that player u draws g acceptably is modeled as

$$\Pr(y_{ug} = 1) = \text{logistic}(b_0 + \theta_u^\top \phi(g)),$$

where the vector θ_u has a coordinate for each feature, negative where the feature is difficult for the player. For a single scalar feature and a gesture-specific difficulty b_g this reduces to the one-parameter logistic model of item response theory, $\Pr(y = 1) = \text{logistic}(\theta_u - b_g)$, and the vector form is its multidimensional generalization.^[S] The posterior over θ_u is maintained by online Bayesian logistic regression with a Gaussian prior, which produces both an estimate of competence and a measure of its uncertainty, and the posterior at the start of play is the population prior of players.

29.2 Selecting challenges

Learning is best supported when tasks are difficult enough to require effort and not so difficult that they cannot be done, a principle known in the literature as desirable difficulty [3].^[D] The selection policy operationalizes it. For each gesture in the eligible pool, the model predicts a success probability $\hat{\pi}_g$ and its uncertainty, and the selector draws gestures with weight highest for those whose predicted success lies in a target band, taken as $[0.6, 0.9]$ at the outset, and with a bonus for gestures whose competence is uncertain, so that every part

of the pool is eventually explored. Gestures that were mastered recur at low frequency for retention. Constraints on the selection keep the pool coherent: no more than a fixed number of new features in a single balloon wave, no two highly confusable gestures simultaneously unless a contrast pair is intended, and no gesture for which the player's recent failure rate is very high until a simpler related gesture has been practiced.

29.3 Not a scalar

The selector exposes the competence vector to the rest of the system, so that the player can be shown, if they wish, which features they are strong in, and so that the research program can observe how competences develop (Chapter 66). No single number is presented as the player's level, and the level labels of Table 26.1 are curriculum stages and not ratings.

Requirement 29.1 (Transparent estimates). The competence estimates **MUST** be stored in the player profile as inspectable values with uncertainties, **MUST** be resettable by the player, and **MUST NOT** be used for any purpose that is not disclosed in the interface.^[S]

Chapter 30

Gesture Fluency

30.1 Dimensions of fluency

Fluency describes how a gesture is drawn, as distinct from whether it is recognized. Four dimensions are measured. *Speed* is the path length divided by duration, and it must be compared across gestures of different length and complexity, so the useful quantity is the ratio of duration to the predicted time $\hat{T}(g)$ of Chapter 11. *Smoothness* is the absence of unnecessary changes in acceleration. A dimensionless measure of it is the squared jerk integrated over the movement and normalized by duration and path length, $J = \frac{T^5}{\ell^2} \int_{t_1}^{t_N} \|\ddot{x}(t)\|^2 dt$, which is unaffected by changes of scale and tempo [13].^[D] A related simple count is the number of speed peaks in the stroke, since a smooth movement has few. *Consistency* is the repeatability of the path across repetitions, measured as the mean distance of each normalized stroke to the user's centroid template for that gesture. *Effort* is the amount of movement used, the path length relative to the reference path, with a small penalty for turning.

30.2 Fluency is not fidelity

Fidelity is the distance $d_H(G, T)$ between the stroke and the reference, and it is the quantity the recognizer is built on. Fluency and fidelity are partially independent. A stroke traced slowly and exactly has high fidelity and poor fluency. A stroke drawn quickly and economically that departs from the reference in the proportions of its parts has low fidelity and high fluency, and it may still be accepted. The distinction matters because training for fidelity produces tracing, which is a different skill from the fluent production that gesture typing depends on, and because a recognizer that accommodates the player's habits (Chapter 23) can improve fidelity by moving the template, which would be mistaken for improvement by the player if only fidelity were measured. The research program therefore reports fluency measures separately from recognition outcomes and fits them in the learning curves of Chapter 66.

Hypothesis 30.1 (Fluency gains precede fidelity gains). In extended practice with a frozen recognizer, improvements in smoothness and consistency begin earlier and continue longer

than improvements in the distance to the canonical reference.^[H]

Chapter 31

Abbreviation and Economical Movement

31.1 What abbreviation is

Players who use gesture keyboards for long periods report that they stop reproducing the full path of a word. They make long, flat sweeps across the middle of a word where the letters lie on a line, cut corners that the recognizer tolerates, exaggerate corners that carry information, and omit the last part of a word that the vocabulary makes obvious.^[0] These behaviors are what *abbreviation* means here: a systematic reduction of the stroke relative to the canonical path, which preserves its recognizability. The specification treats abbreviation as a skill to be taught and measured, and not as an error to be eliminated.

31.2 A formal frame

An *abbreviation operator* $\mathcal{A}$ maps a template T to a reduced stroke $\mathcal{A}(T)$. The operators considered are truncation, which drops the final portion; vertex skipping, which removes landmarks of low informativeness (Chapter 13) and replaces them by chords; corner exaggeration, or a *swoop*, which replaces a sharp corner by an extended excursion that overshoots it; loop elision, which replaces a loop by a tight turn; and straightening, which replaces a gentle curve by its chord. An abbreviation a of gesture g is *recoverable* in library Λ if the recognizer, applied to a , returns g with at least the acceptance confidence, which is the condition under which the abbreviation can be used in play. The most economical recoverable abbreviation is the shortest stroke satisfying that condition,

$$a^*(g) = \arg \min_{a \in \mathcal{A}^*(T_g)} \ell(a) \quad \text{subject to} \quad \text{outcome}(a) = \text{ACCEPTED}(g),$$

where $\mathcal{A}^*$ is the closure of the operators under composition. The informativeness profile of Chapter 13 predicts which parts can be dropped, and the prefix separation of Chapter 22 predicts where truncation is safe.

31.3 Teaching abbreviation

Abbreviation exercises display the full shape and accept a reduced stroke, and they report which reductions were made, so that the player gains an explicit sense of the margin. The reward is the economy bonus (Chapter 7), which rewards short recognized strokes without rewarding closeness to the template. If the recognizer personalizes, an abbreviation that a player uses repeatedly can be recorded as an additional template of the gesture, subject to the drift bound and the separation floor (Requirement 23.1). The danger to be managed is that abbreviations of different gestures converge on one another, since removing detail from two different shapes can make them alike, and the separation floor is the formal guard against it.

Hypothesis 31.1 (Abbreviation reaches a stable economical form). With practice, players converge on abbreviated forms that are shorter than the canonical path by a stable proportion and that remain recoverable, and the forms are consistent within a player and differ between players.^[H]

Chapter 32

Retention and Transfer

32.1 Retention

A skill that is lost after a few days is of little use, and a game that cannot show retention cannot claim to teach. Retention is measured by testing performance on a gesture after an interval without practice and comparing it with performance at the end of practice. The intervals tested are one day, one week and one month. The comparison uses the *savings* measure of memory research, in which relearning after a lapse is compared with original learning: savings are large if relearning is much faster than the original learning. The curriculum protects retention by distributed practice, in which mastered gestures return to the pool at spaced intervals, and by the contextual interference of interleaved presentation (Chapter 25).

32.2 Transfer

Transfer asks whether practice in one condition improves performance in another. Three kinds are distinguished. *Transfer between devices* is the question whether a player who learned on a phone performs well with a mouse, and conversely. *Transfer between recognizers* is the question whether the skills learned under one recognizer remain effective when the recognizer is replaced by a new version, which is the situation of a user whose keyboard is updated, and it is the experimental form of the problem that motivates the whole project. *Transfer between input methods* is the question whether the shape vocabulary learned by touch carries to a keyboard-driven or joystick-driven cursor (Part IX). Each is quantified by a transfer ratio. If $P_{B|A}$ is performance on condition B after practice on condition A , $P_{B|B}$ the performance after the same amount of practice on B itself, and $P_{B|0}$ the performance on B without practice,

$$\text{TR}(A \rightarrow B) = \frac{P_{B|A} - P_{B|0}}{P_{B|B} - P_{B|0}},$$

which is zero if practice on A confers no advantage on B and one if it is as good as practice on B .

32.3 Three skills that are not interchangeable

The word-gesture setting involves three distinct skills, and improvement in one does not establish improvement in another. *Tracing* is reproducing a trajectory that is visible. *Recall* is reproducing a learned trajectory with no guide. *Generation* is producing the trajectory of a newly presented word from knowledge of the spelling and the keyboard geometry, which requires a mapping from letters to positions to movement that neither of the other two exercises. A player can become fluent at drawing complicated glyphs without becoming faster at gesture typing, because the third skill is not trained by the first. The transitional exercises of Chapter 36 and the keyboard assistance of Chapter 37 are the designed bridge between them, and the bridge is an empirical question and not an assumption.

Each skill is therefore assessed separately, with probes that isolate it: tracing against a displayed path, recall of a previously learned path with the guide removed, and generation for words not seen in practice. The central claim of the curriculum, that geometric practice contributes to gesture typing, is tested against a control that receives an equal amount of direct word-gesture practice (protocol P7 of Appendix J). Without that comparison an improvement in typing speed would show only that practice improves performance, and nothing about the geometric content of the practice.

Requirement 32.1 (Records that support transfer studies). Every session record **MUST** identify the device, input method, recognizer version and pipeline version, and the scope of every recognition result (active or vocabulary, Requirement 21.1), so that retention and transfer can be analyzed without reconstructing conditions from memory.^[S]

Part VI

From Geometric Shapes to Words

This part is the central transition of the project, from an arcade game about shapes to a tutor for typing. Its method is to make the shapes of words a continuation of the shapes already learned: a word is a path on a keyboard, a path has the geometric properties developed in Part III, and the same recognizers, complexity measures and curriculum principles apply. What is new is the keyboard, which fixes the coordinates, and the language, which supplies the prior.

Chapter 33

Keyboard Geometry

33.1 Layouts as data

A keyboard layout is a table that assigns to each character a key center and a key size in a common coordinate system. The unit of length is the width of a key. The default layout is the QWERTY layout as it appears on a phone in portrait orientation, which has three rows of ten, nine and seven keys, each row centered, so that the rows are offset by 0, 0.5 and 1.5 key widths relative to the first. Key j in row r then has center $(o_r + j + \frac{1}{2}, r \pi_{\text{row}} + \frac{1}{2})$ where o_r is the row offset and π_{row} is the vertical pitch between rows, taken as 1.3 key widths to reflect the greater height of phone keys. A physical keyboard has different offsets, in which the second row is shifted by a quarter of a key and the third by three quarters, and the layout is a parameter of the system and not a constant. Alternative layouts, such as AZERTY, QWERTZ and Dvorak, are further tables, and the property of Chapter 39 that geometric confusability depends on the layout is a reason to keep layouts as data.

Requirement 33.1 (Layouts as data). A layout **MUST** be specified by a data file giving, for each character, the key center and size in key widths, and **MUST** be identified by a name and version recorded with every word trajectory and recognition result.^[5]

33.2 The idealized word path

The *idealized path* of a word $w = c_1 c_2 \dots c_n$ is the polyline through the centers of the keys of its letters in order. Consecutive repeated letters correspond to the same key and contribute a single vertex, so that the path of *hello* has four vertices, at *h*, *e*, *l* and *o*, and the path of *pizza* has four, at *p*, *i*, *z* and *a*. A word consisting of a single key, such as *a*, is a point and not a path, and is entered by a tap. The straight segments of the polyline pass over keys that the writer does not intend, and this is the central ambiguity of gesture typing: a stroke from *o* to *r* passes over *u*, and the geometry alone cannot say whether *u* was meant. The lexicon and the language model resolve it, as the *or* and *our* entry of Table 39.1 shows.

33.3 Repeated letters and neighbors

Repeated letters have several possible conventions. The default is to collapse them and to rely on the lexicon to recover the double letter. An alternative, useful in teaching, is a deliberate small loop or a brief dwell at the key to signal repetition, and the dwell uses the temporal information that Chapter 14 otherwise excludes. Neighboring keys cause a second class of ambiguity, because a stroke aimed at one key may land within the area of its neighbor. Writers' errors in aiming are modeled by a Gaussian distribution around the key center, with a standard deviation of about a third of a key width as a starting value to be estimated from data,^[H] which is used in two ways: to generate synthetic strokes for testing, and to define the tolerance of the location channel in the probabilistic recognizer.

Chapter 34

Word Shapes

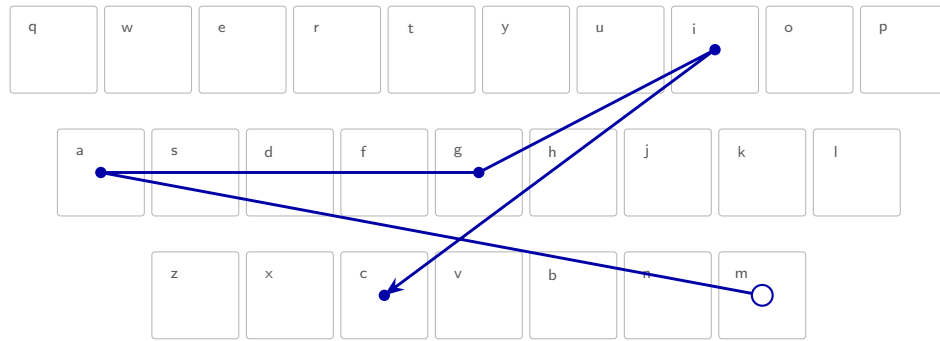
34.1 Geometric identity of a word

Every word has a shape in two senses. Its *located path* is the idealized path in the coordinates of the keyboard, which is what a recognizer that uses absolute position compares with the stroke. Its *free shape* is the path after normalization for translation and scale, which is what a player sees as the form of the word independent of where it is drawn. The two correspond to the two channels of the SHARK2 recognizer, a location channel and a shape channel,^[D] and a gesture-typing system can use either or both. Because the keyboard is displayed, the location channel is available in a typing tutor and gives much stronger discrimination; the free shape is the representation used in the early transitional exercises of Chapter 36, where the keyboard is not yet shown.

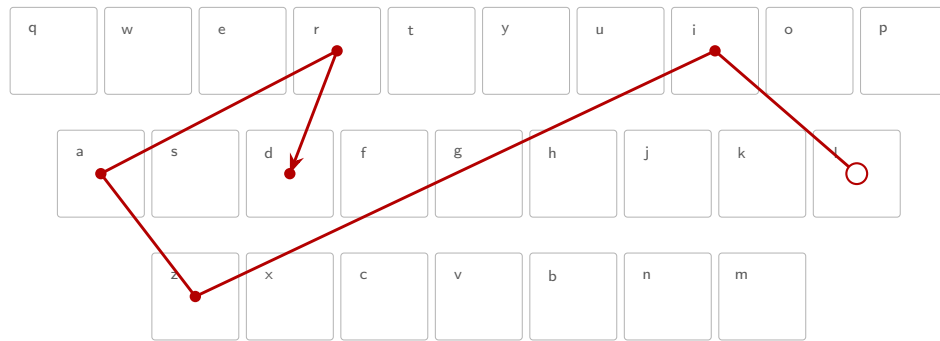
Word shapes are readily described informally. The word *magic* has a trajectory that doubles back on itself with a crossing, which makes it feel like a pretzel, and *lizard* winds in a continuous curve from the lower left to the upper right and back. The informal description captures something real but should be checked against the computation. Figure 34.1 gives the idealized paths, and Table 35.1 the measures. For *magic* the computation gives one proper self-crossing and two near-reversals, consistent with the intuition of a doubling-back shape. For *lizard* it gives no self-crossing and no near-reversal, but a total turning of just over one full turn, so that the winding is real while the form is not knotted. The informal descriptions are therefore supported in the first case and refined in the second, which is the benefit of treating them as hypotheses about geometry and not as definitions.

34.2 Generating and classifying shapes automatically

The specification requires that word shapes be computed, not drawn by hand. Given a layout and a lexicon, the generator produces for each word the idealized path, the resampled located path with N points, the free shape, and a descriptor containing the quantities of Definition 15.1. The classifier assigns each word to a shape class from the descriptor. A coarse and useful taxonomy follows from the descriptor: *linear* words, whose total turning is below half a turn and which have no reversal, such as *pizza*; *arched* words, which turn smoothly in one sense, such as *lizard*; *zigzag* words, which have two or more near-reversals,



magic: $L = 18.30$, $T/2\pi = 1.02$, $X = 1$, $R = 2$.



lizard: $L = 13.92$, $T/2\pi = 1.06$, $X = 0$, $R = 0$.

Figure 34.1: Idealized trajectories for *magic* (top) and *lizard* (bottom) on the default mobile QWERTY layout. The open circle marks the first key and the arrow marks the direction of traversal. Values of L (path length in key widths), $T/2\pi$ (total absolute turning in turns), X (proper self-crossings) and R (near-reversals) are computed by the reference definitions of Chapter 33 and Chapter 15.

such as *banana*; and *knotted* words, which have at least one self-crossing, such as *magic* and *keyboard*. The classes are a reporting convenience for the curriculum and for the choice of examples, and they are not an input to the recognizer.

Requirement 34.1 (Reproducible shape generation). The word-shape generator **MUST** be deterministic, **MUST** take the layout and the lexicon as explicit inputs, and **MUST** emit descriptors tagged with the layout version, the lexicon version and the generator version.^[S]

Chapter 35

Word Difficulty

35.1 Length is not geometric complexity

A curriculum for words that ordered them by the number of letters would follow the convention of conventional typing instruction and would misrepresent the task. The geometry of the stroke depends on where the letters lie and not on how many there are. Table 35.1 shows the effect: *pizza* and *magic* both have five letters, yet the path of *magic* is nearly twice as long (18.30 against 9.72 key widths) and turns more than three times as much (1.02 turns against 0.29). Two six-letter words, *lizard* and *banana*, differ in path length by a factor of more than two, and *banana* contains four near-reversals while *lizard* contains none. A seven-letter word, *minimum*, has four self-crossings and five near-reversals, while the seven-letter *gesture* has one crossing and one near-reversal.

The pattern is general in the convenience sample of the project. For the 170 words of the sample described in Chapter 39, the Pearson correlation between the number of letters and the path length is $r = 0.68$, with total turning $r = 0.82$, and with the number of self-crossings $r = 0.48$. Length in letters is a fair predictor of how much turning a word demands and a weak one of whether the path crosses itself, and in every case it leaves a large part of the variance unexplained. The sample is a hand-chosen list of common words and not a frequency-weighted corpus, so the coefficients describe that sample and are reported as an illustration of the point and not as an estimate for English.

35.2 A curriculum by trajectory difficulty

The difficulty of a word has two parts. The *production difficulty* is the predicted execution complexity of its path, from the same model as for shapes (Chapter 15): length, turning, crossings and reversals, computed in key widths so that it is comparable between words on one layout. The *discrimination difficulty* is the ambiguity term, which for a word depends on the lexicon in use, since a word with many geometric neighbors is harder to get recognized and requires more careful drawing. A word's combined difficulty is

$$C(w) = \alpha L(w) + \beta T(w) + \gamma X(w) + \delta R(w) + \eta A_{\Lambda}(w),$$

Word	Letters	Keys	L	$T/2\pi$	X	R
the	3	3	5.72	0.44	0	1
word	4	4	13.39	0.69	0	1
magic	5	5	18.30	1.02	1	2
lizard	6	6	13.92	1.06	0	0
hello	5	4	11.76	0.81	0	1
quick	5	5	15.57	0.85	0	1
pizza	5	4	9.72	0.29	0	0
banana	6	6	29.72	1.99	0	4
minimum	7	7	16.59	2.33	4	5
keyboard	8	8	26.73	2.27	7	2
gesture	7	7	13.03	1.23	1	1
typing	6	6	12.03	1.03	0	1

Table 35.1: Geometric measures of selected words on the default mobile QWERTY layout (Chapter 33). Letters is the length of the written word, Keys is the number of vertices after collapsing repeated letters, L is path length in key widths, $T/2\pi$ is total absolute turning in full turns, X is the number of proper self-crossings and R is the number of near-reversals (turns of at least 150°).

with A_Λ computed against the active lexicon Λ . The word curriculum introduces words in increasing order of $C(w)$ within frequency bands, so that a player first meets common words with simple paths and later meets rare words and complex paths, and the ordering is recomputed whenever the lexicon changes. Frequency enters as a second axis, not as a component of C : a word that is common is useful to learn early even if it is geometrically hard, and a word that is simple but rare is useful later. The player-facing sequence is a compromise controlled by the competence model (Chapter 29).

Chapter 36

Transitional Exercises

The transition from shapes to words is the most delicate step of the curriculum, and it is designed in stages, each of which changes one thing. The stages move the player from tracing, through recall, toward generation, the three skills distinguished in Chapter 32, and only the last stages exercise the mapping from spelling to movement on which gesture typing depends. Whether the earlier stages contribute to that mapping, or only to the motor fluency of drawing, is what the comparison with direct word practice (protocol P7) is designed to show, and the design here makes no claim that it does.

In the first stage the player is shown a *word-derived shape*, the free shape of a word drawn as an abstract glyph with no keyboard and no letters. The task is the one the player already knows: draw the glyph. The shapes are chosen from the advanced families of Chapter 28 and drawn from real words, so that the geometry is that of words, but nothing about spelling is revealed. The player is learning the movements.

In the second stage a faint keyboard appears behind the glyph, positioned so that the shape lies over the letters it traverses. The shape is still presented as a shape. Players who are inclined to notice, notice that it passes through letters, and the stage depends on that noticing being spontaneous.

In the third stage, the keys that the path of the glyph traverses are highlighted in order as the player draws, so that the letters of the word are revealed by the motion. The word has not been displayed, and the player may read it from the highlighted keys as they would read a word from the movement of a pen.

In the fourth stage the word is displayed as text, and the glyph is removed, with the keyboard and an optional reference trajectory remaining (Chapter 37). The player draws the word. The shape and the spelling are now associated, since the same movement has been paired with both.

In the final stage the word is given by a cue that does not display the spelling, such as a short definition, a picture or a sentence with a gap, and the keyboard is shown without the reference trajectory. The player must recall the spelling and produce the movement. Retrieval practice of this kind is among the most reliable ways to consolidate learning.^[D]

The stages are not a fixed sequence for every word. A word advances through them at the rate of the player's competence on it, so that a player may be at the final stage for common words and at the first for new ones within a single session. Each word carries its

stage in the profile (Chapter 52), and the game selects balloons accordingly. A player may skip stages if the competence model shows that they are not needed.^[5]

Chapter 37

Visible Keyboard Assistance

37.1 Assistance levels

The player's use of a keyboard overlay is graded, and the grades are defined so that assistance can be withdrawn smoothly. At the highest level the keyboard is displayed with the complete reference trajectory of the target word drawn over it. At the next level the keyboard is displayed with only the first and last keys marked. At the next, the keyboard is displayed without any marks. At the next, the keyboard is hidden but the playfield retains the layout, so that the player must draw in the remembered positions. At the lowest, there is no overlay and the layout is neither displayed nor remembered by the interface, which is the condition of a skilled user of a gesture keyboard who types in a notes application with no feedback. A hint, in the form of a brief display of the reference trajectory, remains available on request at a small cost in score.

37.2 Fading

Assistance is reduced according to performance on each word. The reasoning follows the guidance hypothesis of motor learning (Chapter 25): assistance that is always available becomes a crutch, and what is learned under guidance may not be retained when the guidance is removed.^[D] The rule of the first design reduces the assistance level of a word by one after each run of k successes at the current level and increases it by one after m consecutive failures, with $k = 5$ and $m = 3$ as starting values, and the quantity is stored per word. The effect of fading is itself an experiment: Hypothesis 37.1 compares groups with a fixed assistance level and a performance-driven fade.

Hypothesis 37.1 (Performance-driven fading improves retention). Players whose assistance is reduced according to their performance show better retention of word trajectories after a one-week interval without assistance than players with a constant level of assistance, at equal practice time.^[H]

Chapter 38

Vocabulary Learning

38.1 Classes of words

The vocabulary is organized in classes that correspond to different purposes. *Common words* are the several thousand most frequent words, which make up the greater part of ordinary text and whose paths are practiced first. *Uncommon words* are infrequent but standard words, and they are where discrimination and the weakness of strong priors are most visible. *Proper nouns* such as names of people and places are often absent from a default lexicon and are exactly the entries a user needs to add. *Technical vocabulary* includes abbreviations, identifiers and specialized terms for a domain. *User-defined entries* are words the user adds, including words that are not words of any language but are used regularly, such as handles, codes and shorthand.

38.2 A transparent personal dictionary

The personal dictionary is a plain list the user can read and change. Each entry has a word, a weight that expresses how likely the user is to write it, the date it was added, the source of the entry (imported, typed, or learned from use) and an optional note. The list is stored in a documented tabular text format with one entry per line and tab-separated fields, with a header naming the fields, and it can be imported, exported, edited and deleted in bulk. Nothing is learned that does not appear in the list, and no entry in the list is hidden. The aim is the opposite of a keyboard whose learned vocabulary is a black box and is lost on uninstalling it, which is the situation described in Chapter 2.

Requirement 38.1 (Personal dictionary transparency). The personal dictionary **MUST** be fully visible to the user, **MUST** support import and export in a documented plain-text format, **MUST** support editing and deletion of any entry, and deletion of an entry **MUST** remove every statistic, template and count derived from it.^[5]

Bulk import accepts word lists with optional weights from text files, and it validates every entry against the layout, so that entries containing characters the layout does not provide are reported and not silently dropped. A newly added word is associated with

a located path computed by the generator of Chapter 34 and is available for training as a balloon target after a configurable number of exposures.

Chapter 39

Word Ambiguity

39.1 Geometric confusability

Two words are geometrically confusable when their idealized paths are close. For this chapter the distance is the location-channel distance d_{loc} , the mean Euclidean distance in key widths between the 32-point arc-length resamplings of the two paths, which is a practical measure of how well a stroke aimed at one word can also be read as the other. The *confusability graph* of a lexicon has the words as vertices and an edge between two words whose d_{loc} is below a threshold ε . The degree of a word in this graph is its number of geometric neighbors, and it is the quantity that enters the ambiguity term of the word difficulty (Chapter 35).

The convenience sample of 170 common words gives a concrete picture. Table 39.1 lists the ten closest pairs. The closest, *or* and *our*, are at zero distance: the three keys *o*, *u* and *r* lie on one line of the top row, so the path of *or* passes over *u* and is the same polyline as the path of *our*. No geometric evidence can separate them, and the choice is made entirely by the language model and the context. The next pairs, such as *the* and *three*, *might* and *night*, and *horse* and *house*, differ by one letter whose key lies close to the path of the other word, and they are familiar sources of error to users of gesture keyboards. The sample is small and its pairs are not exhaustive, since the larger the lexicon the more near neighbors appear, and the full analysis is a task for the lexicon of the research program.

39.2 Geometry and priors

The resolution of an ambiguous stroke combines the geometric likelihood with a prior from the language, as in Chapter 20. Let $P(w \mid c)$ be the probability of word w in context c of preceding words, estimated by a unigram frequency or an n -gram model, with a neural language model as a later option. The posterior is $P(w \mid G, c) \propto P(G \mid w) P(w \mid c)^\lambda$. The balance, set by λ , is a design variable with a familiar failure mode: if the prior dominates, a frequent word overrides a clear geometric match for a rare word, and the user who has drawn the rare word with care sees the common one inserted. If the likelihood dominates, noise in a stroke produces odd words that the context would have excluded. The right value depends on the vocabulary and the user, and the personal prior P_u (Chapter 23) shifts it for

Word	Word	d_{loc}
or	our	0.000
the	three	0.119
might	night	0.168
horse	house	0.215
still	will	0.312
some	stone	0.387
he	her	0.417
well	will	0.428
loop	pop	0.442
light	might	0.449

Table 39.1: The ten geometrically closest word pairs in the 170-word convenience sample of Chapter 39, under the location-channel distance d_{loc} (mean pointwise distance in key widths between 32-point arc-length resamplings of the idealized paths; pairs whose letter counts differ by more than three are not compared).

words the user writes often. The entropy of the posterior, $-\sum_w P(w \mid G, c) \log P(w \mid G, c)$, is a convenient summary of the ambiguity of a stroke and is what the interface uses to decide whether to present alternatives.

39.3 Training on ambiguity

An ambiguous pair is a teaching opportunity. A player who learns that *the* and *three* are close learns also how far to extend the stroke to indicate the second. The tutor generates *contrast sets* from the confusability graph, balloons that carry words with neighbors in the same set, and reports which of the two the stroke favored, in the same manner as the contrast pairs among shapes (Chapter 27).

Chapter 40

Sentence Composition

40.1 From words to text

Real typing consists of sequences of words with spaces and punctuation, and the tutor extends the single-word exercises in two steps. First, a sentence is presented as an ordered chain of word balloons, drawn from a phrase set, and the player produces the words in order, with the space inserted automatically after each word as gesture keyboards conventionally do. Second, the player copies or composes a sentence in a text area with no balloons, under the control of the same recognizer, and the interface reports the text entered and the corrections made.

40.2 Spaces, punctuation, capitalization and correction

Several decisions fix the semantics of text entry. A word is committed when the stroke is released, or at an early commit if the mode is enabled (Chapter 22), and the space is appended on commit. Punctuation is entered by tapping the punctuation keys, and a trailing punctuation mark removes the automatically inserted space before it. Capitalization is automatic at the start of a sentence, and a later extension provides a gesture to capitalize a word (Chapter 73). Correction is the most important of these, because errors will occur and the cost of recovering from them determines the practical speed. Three correction modes are provided: deletion of the last word by a backspace gesture or key, replacement of the last word by an alternative suggested by the recognizer, and character-level editing of the text. The tutor measures how many gestures a correction costs and reports it as part of proficiency.

40.3 Word selection

When the posterior over words is not sharply peaked, the interface offers the next most likely words in a strip, and the player may select an alternative by tapping it. The use of the strip is the practical counterpart of the AMBIGUOUS outcome of the arcade game, and the training treats it as a skill: selecting an alternative promptly is faster than redrawing, and

the strip is a component of realistic typing.

Chapter 41

Typing Proficiency

41.1 Speed and accuracy

The standard measure of text entry speed is words per minute, with a word defined as five characters including spaces, so that for a transcribed phrase of $|S|$ characters entered in T seconds the speed is $\text{WPM} = \frac{(|S| - 1)/T \cdot 60}{5}$, where the first character is excluded because the timing starts at its entry [28].^[D] Accuracy is reported by the total error rate, which combines the errors left in the final text and the errors that were made and corrected,

$$\text{TER} = \text{CER} + \text{UER} = \frac{\text{INF}}{\text{C} + \text{INF} + \text{IF}} + \frac{\text{IF}}{\text{C} + \text{INF} + \text{IF}},$$

where C counts correct characters in the transcribed text, INF counts incorrect characters that were corrected, and IF counts incorrect characters that were not [28].^[D] The two components are reported separately, because a fast user who leaves errors uncorrected and a careful user who corrects them all have different profiles that a single rate would hide.

41.2 Measures specific to gesture entry

The measures of conventional keyboards need adjusting for a gesture keyboard, in which one stroke produces a word. The *gestures per character* is the number of strokes used divided by the number of characters in the final text, the analogue of keystrokes per character [19].^[D] It is a measure of the efficiency of the interface and the language model together, since a perfect system would need one gesture per word and so a value of about one sixth for typical English. The *correction cost* is the number of additional gestures and taps spent on correction per word. The *gesture efficiency* of a stroke is the ratio of the ideal path length to the executed path length, capped at one, and it measures the economy of movement of Chapter 31. *Sustained performance* is the speed and error rate in successive five-minute windows of a longer session, which shows fatigue and settling.

41.3 Realistic tasks

Proficiency is evaluated over tasks of increasing realism: the isolated word of a balloon, a chain of words from a phrase set, the transcription of short phrases, and free composition. The phrase set used for transcription follows the established set of phrases designed for text entry evaluation, which are memorable, representative of English letter frequencies, and contain no punctuation that would complicate measurement [20].^[D] Composition tasks, in which the participant writes their own text, are included to capture the cost of thinking and of correction, which transcription removes. The reports of Chapter 66 relate these measures to practice.

Part VII

Game Presentation and Interaction

This part specifies how the game looks, moves, sounds and feels in the hand. The requirements are modest in the first milestone and deliberately concrete, because presentation is where the principle of immediate feedback (Principle 1) is either honored or lost.

Chapter 42

Visual Design

42.1 A restrained visual language

The visual design aims at legibility under the conditions of play: a small screen, a moving target, a finger that hides part of the picture, and a player whose attention is divided between the descending balloons and the stroke being drawn. The design is therefore restrained. There are few colors, with one hue per role and not per gesture, there is no decorative motion, and the balloons carry high-contrast glyphs on plain fills. The color of a balloon is not a code for its gesture; gestures are told apart by their shapes, since color would give a free cue that disappears when the game moves toward words.

42.2 Legibility under overlap

The playfield will frequently show several balloons and a stroke at once, and the stroke will often pass over balloons. Three devices keep the picture comprehensible. The ink is drawn in a layer above the balloons with a thin contrasting outline, so that it is visible on any fill. The glyph of each balloon is drawn with a halo, a wider stroke in the fill color beneath the glyph, so that the glyph remains legible when ink crosses it. The balloons are not allowed to overlap one another at spawn, and the spawner enforces a minimum separation. In the dense case, in which several balloons are close to the failure line, the lowest balloon, which is the next to be chosen by Requirement 5.2, is marked with a small, quiet indicator so that the player knows which of identical balloons will be hit.

42.3 Contrast and sizing

The design follows the contrast success criteria of the Web Content Accessibility Guidelines [31]:^[D] text, including score and lives, has a contrast ratio of at least 4.5:1 against its background, and graphical objects that carry meaning, including glyphs and the failure line, at least 3:1. Balloon glyphs have a minimum size of 12 percent of the playfield width, which on a phone of 360 CSS pixels is about 43 pixels and which keeps simple glyphs legible at arm's length. A high-contrast theme, in which fills are replaced by outlines on a black or

white field, is a setting of the first release.

Requirement 42.1 (Redundant encoding). No information **MUST** be conveyed by color alone. Outcomes, balloon urgency and the identity of a gesture **MUST** each be conveyed also by shape, position, motion or text.^[5]

Chapter 43

Animation and Feedback

43.1 Motion of balloons

Balloons move at the speed defined in Chapter 5, and their position is updated every simulation step and rendered by interpolation between steps so that motion is smooth at any display rate. There is no easing on the descent, since constant speed is what makes the time remaining readable. A balloon that is close to the failure line is shown with a subtle change that does not rely on color, such as a slow pulse in its outline, to signal urgency.

43.2 Popping, ambiguity and failure

The three outcomes of Requirement 6.1 are given distinct motion. A pop is a brief expansion of the balloon and a burst of small fragments, lasting about 250 milliseconds, followed by a short score indicator rising from the position of the balloon. An ambiguous outcome produces a brief horizontal shake of each candidate balloon and a thin ring around it, lasting about 300 milliseconds, with the stroke fading rather than dissolving. A rejection dissolves the ink along its length from the end to the start over about 200 milliseconds, with no motion of the balloons. An ambiguous stroke should not read as a failure without explanation, and the specification does not settle how much explanation fits inside the arcade rhythm. The first prototype is the instrument for finding out: it tries the candidate highlight alone, the highlight with the nearer candidate marked, and a short text or glyph hint available on request, and it records which of these players understand without slowing down. Until that is measured the minimal highlight of this section is the default, and any richer feedback is an option that never blocks input.^[H] These durations are design targets, and the specification asks that all three can be told apart in a screen recording at one quarter speed, which is a practical test that the differences are in the motion and not in the color.^[S]

43.3 Transitions

Transitions between levels are short, 600 milliseconds or less, and consist of a change in the status strip and a brief announcement of the new gesture family, with the reference

trajectory of any new gesture shown beside its glyph. Play is not paused for the transition unless the player has asked for a slower pace, so that the arcade rhythm is not broken.

43.4 Reduced motion and flashing

When the browser reports the user preference for reduced motion, through the corresponding media query, the game replaces the shake, the fragments and the pulse by static indications, such as an outline change and a brief icon, which preserve the distinction between outcomes without movement.^[D] No element flashes more than three times in any one-second period, in accordance with the corresponding guideline [31].^[D]

Chapter 44

Audio

Audio provides a second channel for the outcomes and for progression, and it is optional. The sounds are short, a quarter of a second or less, and synthesized in the browser with the Web Audio interface and not loaded from files, which keeps the package small and the sounds consistent across devices. A rising two-note figure marks a pop, a low single note an ambiguity, a brief soft noise a rejection, a falling figure the loss of a life, and a short ascending run the passage to a new level. The pitch of the pop may rise with the streak, which gives an audible sense of momentum without adding a score display.

Browsers do not allow audio to start until the user has interacted with the page, so the audio context is created or resumed on the first touch of the game. On devices that support it, a short vibration may accompany the outcomes; the interface for this is not available in every browser, and in particular is not available on iOS Safari at the time of writing, so haptics are an enhancement and never the only signal.^[D]

Requirement 44.1 (Audio is optional and redundant). All audio **MUST** be switchable, the game **MUST** be fully playable with the sound off, and every outcome communicated by sound **MUST** also be communicated visually.^[S]

Chapter 45

Touchscreen Ergonomics

45.1 Occlusion

The finger that draws a stroke covers the part of the screen under and just beyond it, and the player cannot see the ink immediately under the fingertip. For a game whose target is on screen while drawing, this has two consequences. The balloons that matter must not be hidden by the drawing hand, which argues for a layout in which balloons descend in the upper two thirds of the screen and the stroke is drawn mostly in the lower third, and the design therefore invites strokes to be made in the lower part of the playfield, although it does not require it, and the surface remains the whole playfield. The ink should be drawn with a short trailing glow that extends behind the fingertip, so that the shape being drawn is visible even when the tip is covered.

45.2 Screen size, handedness and gesture area

The physical size of a gesture depends on the screen. A gesture of fixed logical size is physically larger on a tablet than on a phone, and the relationship between physical size and accuracy is not monotone: very small gestures suffer from the noise of the finger relative to the shape, and very large ones from the effort of covering distance. A starting target is that a typical gesture occupy between 15 and 40 millimeters, and the recognizer's tolerance and the minimum stroke (Chapter 6) are set in physical terms through the device's reported pixel density where this is reliable.^[H] Handedness affects occlusion, since a right-handed player's hand covers the area to the right and below the fingertip, and a setting mirrors the placement of the status strip and of any hint display to keep them clear of the hand. Tablets introduce palm contact, and the first implementation ignores additional pointers once a stroke has begun, which provides a basic rejection of the palm.

45.3 Edges and system gestures

Mobile operating systems reserve the screen edges for navigation, and a stroke that begins at the edge may be taken by the system and cancelled. The specification therefore uses margins

at the left and right edges as non-drawing regions (Chapter 4) and handles cancellation events by discarding the stroke without penalty. The same applies when the browser interface intervenes, as when the address bar appears or the user interface of the system is summoned.

45.4 Scale and recognition accuracy

The relationship between the scale of a gesture and the accuracy of its recognition is a measurable property of the system. The jitter of the sampling and the wobble of the hand have approximately fixed magnitude in physical units, so that the relative noise in a normalized gesture falls as the gesture grows. The dependence is measured in the research program by recording strokes at instructed scales (Chapter 65), and it determines the recommended size of the drawing area on each device class.

Chapter 46

Accessibility

46.1 Baseline requirements

The initial release establishes a baseline and does not attempt to support every future input device. The baseline consists of five adjustable properties and a commitment. *Timing* is adjustable by a speed scale from one quarter to full speed, applied to the descent speed and the spawn rate together, and by an optional untimed practice mode, because time pressure is often the main barrier to participation for players with motor or cognitive impairments. *Contrast* is adjustable through the high-contrast theme. *Target size* is adjustable: the glyph size and the minimum stroke can be enlarged, and any interactive control has a minimum target size of 24 by 24 CSS pixels, which is the minimum of the corresponding success criterion [31].^[D] *Reduced motion* is a setting that follows the system preference by default. *Alternative control* is represented in the first release by the pointer abstraction, which already accepts a mouse and a stylus, and by the commitment to the keyboard-driven drawing of Chapter 57.

The commitment is that the recognition engine and the game never assume a touchscreen. That includes the requirement that every function reachable by a gesture is also reachable by a control, so that a player who cannot draw a stroke can still pause, restart, change settings and export data.

46.2 Pause and stop

Moving content that lasts more than five seconds and starts automatically must be able to be paused, stopped or hidden, which for a game means that a pause control is always available and is reachable by keyboard.^[D] Pausing stops the simulation and the clock, so that no life can be lost during a pause, and the game state is preserved if the page is hidden, as when the player switches applications.

Requirement 46.1 (Accessibility baseline). The first release **MUST** meet the contrast ratios of Chapter 42 and **MUST** provide a speed scale, a high-contrast theme, adjustable glyph and control sizes, a reduced-motion mode, a pause control operable without a pointer, and redundancy of outcome signals (Requirements 42.1 and 44.1).^[S]

Part VIII

Software Architecture

This part specifies how the game is built. Its organizing idea is a strict separation between four layers: input, which turns a device into trajectories; recognition, which turns trajectories into decisions; simulation, which turns decisions and time into game state; and presentation, which turns state into pixels and sound. Each layer communicates with the next through a small, versioned, serializable interface, which is what makes replay, testing, comparison of recognizers and the later addition of devices possible.

Chapter 47

Technology Selection

47.1 Language and rendering

The implementation is a browser application written in TypeScript. The language brings static types to the interfaces that matter most, in particular the trajectory and the recognition result, and it compiles to JavaScript that runs without a plugin on every target. Rendering uses the Canvas 2D interface, which is sufficient for the flat shapes and strokes of the game, has predictable performance on mobile devices, and has a simple and well-understood model of drawing. The alternatives were considered. Scalable vector graphics in the document tree suit static diagrams but add per-element overhead for dozens of moving balloons and a fast-growing stroke. WebGL offers more throughput than the game needs, at the price of complexity and of context-loss handling that is not warranted by the visual design of Part VII. The choice can be revisited if profiling shows a need, and the architecture isolates rendering behind an interface so that the change would not touch simulation or recognition.^[5]

47.2 Priorities

Four properties order the choices. *Predictable input handling* favors the pointer events interface over separate mouse and touch handlers, as detailed in Chapter 49. *Low latency* favors work that is incremental and does not allocate in the frame loop. *Offline capability* favors a static application with all assets bundled and a service worker that caches them (Chapter 56). *Portability* favors the absence of runtime dependencies in the core packages and the use of widely supported features, with the compatibility list of Appendix I.

47.3 Package structure

The code base is divided into packages whose dependencies point in one direction. The *core* package contains the gesture representation, normalization, recognizers, library and scoring-independent mathematics; it has no dependency on the browser and runs under a command-line runtime for testing and for the evaluation harness. The *sim* package contains

the game simulation and depends on the core. The *input* package contains adapters for devices and depends on the browser. The *ui* package contains rendering, audio and settings and depends on all of the preceding. The *lab* package contains the evaluation harness and the experiment tooling and depends on the core. The direction of the dependencies is the second architectural commitment in executable form: the recognizer cannot know about the game because it is below it.

Chapter 48

Game Loop and Simulation

48.1 Separation of stages

The program is organized around a loop that separates five concerns: sampling input, advancing the simulation, recognizing gestures, scoring, and rendering. Input events arrive asynchronously from the browser and are appended, with their timestamps, to a queue; they are not processed in the event handler beyond that. The simulation advances in fixed steps. Recognition runs when a stroke ends or, in early-commit mode, on each new sample. Scoring runs when recognition produces an outcome. Rendering runs once per display frame and reads the state without changing it.

48.2 Fixed timestep

The simulation uses a fixed step of $\Delta = 1/60$ second, accumulating real time and advancing the state in whole steps. This makes the simulation independent of the display rate, so that a 120 Hz display does not run the game faster, and it makes the evolution of the state a function of the step count and the inputs alone. The rendering interpolates between the previous and the current state by the fraction of a step left in the accumulator.

Listing 48.1: The main loop (pseudocode).

```
accumulator = 0
previous = clock()
each display frame:
    current = clock()
    frame = min(current - previous, MAX_FRAME)    // guard against long
    stalls
    previous = current
    accumulator = accumulator + frame
    while accumulator >= STEP:
        events = queue.takeUpTo(simTime + STEP)  // by event timestamp
        state = step(state, events, STEP)        // includes recognition,
    scoring
        simTime = simTime + STEP
        accumulator = accumulator - STEP
```

```
render(state, accumulator / STEP)
```

48.3 Determinism and replay

Deterministic behavior is a requirement because it turns a recorded session into a test and into an experimental instrument. The simulation uses no source of nondeterminism. It reads no wall clock, since time is the step counter times Δ , and it takes randomness only from a seeded generator of which a small and well-known example is the following.

Listing 48.2: A small seeded generator (mulberry32).

```
export function mulberry32(seed: number): () => number {
  let a = seed >>> 0;
  return () => {
    a = (a + 0x6d2b79f5) >>> 0;
    let t = a;
    t = Math.imul(t ^ (t >>> 15), t | 1);
    t ^= t + Math.imul(t ^ (t >>> 7), t | 61);
    return ((t ^ (t >>> 14)) >>> 0) / 4294967296;
  };
}
```

A session record consists of the seed, the configuration (mode, library version, recognizer version, pipeline version, parameters) and the timestamped input events, and replaying it through the same code reproduces the same state at every step. Because floating-point evaluation of elementary functions may differ across engines (Chapter 17), the replay guarantee holds on one engine and is checked to a tolerance across engines. The simulation's own arithmetic is restricted to addition, subtraction, multiplication and comparison, which are exactly specified for double-precision numbers, and recognition, which uses trigonometric functions, is the stage at which a divergence is tolerated and monitored.

Requirement 48.1 (Replayable sessions). A session record **MUST** contain every input needed to reproduce the session, **MUST** be sufficient to reproduce the sequence of game states exactly on the same engine, and the simulation **MUST NOT** read the wall clock or any source of randomness other than the seeded generator.^[5]

Chapter 49

Input Abstraction

49.1 A neutral trajectory interface

The input layer converts the events of a device into a stream of stroke events with a common shape, so that nothing downstream depends on the device. A stroke event has a kind, which is one of begin, sample, end and cancel, a stroke identifier, a position in logical coordinates, a timestamp from a monotonic clock, and an optional bag of extended channels. The adapters of Part IX all produce this stream, and the recognizer receives only the accumulated trajectory.

Listing 49.1: The neutral input interface.

```
export type PointerKind = "touch" | "mouse" | "pen" | "keyboard" | "gamepad"
    | "gaze" | "other";

export interface Sample {
  x: number; y: number; t: number;           // logical units, milliseconds
  pressure?: number; width?: number; height?: number; tilt?: [number, number]
};

export interface StrokeEvent {
  kind: "begin" | "sample" | "end" | "cancel";
  strokeId: number;
  sample: Sample;
}

export interface InputAdapter {
  readonly id: string;
  readonly version: string;
  readonly kind: PointerKind;
  start(target: HTMLElement, sink: (e: StrokeEvent) => void): void;
  stop(): void;
}
```

49.2 The pointer adapter

The first implementation supports touch, mouse and pen through a single adapter built on the pointer events interface of the browser, which delivers a unified stream for all pointing devices.^[D] The adapter sets the touch action of the drawing element to none, so that the browser does not scroll or zoom in response to the stroke, and it captures the pointer on the begin event so that the stroke continues even if the pointer leaves the element. It records the pointer type and ignores events from a second simultaneous pointer. Many browsers deliver pointer events at the display rate, which on a 60 Hz screen discards the extra samples that touch controllers produce at a higher rate. The interface can return the coalesced events that make up one delivered event, and the adapter uses them to recover the full sample stream; the browser may also offer predicted events, which are used only for drawing the ink slightly ahead of the finger and never for recognition, since a prediction is not an observation.^[D] The timestamps of the events come from the high-resolution clock of the browser, not from the system date, and are converted to milliseconds from the start of the stroke when stored.

Requirement 49.1 (Adapter contract). An input adapter **MUST** emit events in non-decreasing timestamp order, **MUST** emit exactly one end or cancel per begin, **MUST** convert coordinates to logical units, **MUST** record the device characteristics needed to interpret the samples, and **MUST NOT** perform any smoothing or resampling.^[S]

The last clause keeps the raw gesture raw (Requirement 9.1): smoothing belongs to the normalization pipeline, where it is named, versioned and replaceable.

Chapter 50

Recognition Engine

50.1 Modular interfaces

The recognition engine is a composition of stages with explicit interfaces: preprocessing, candidate generation, scoring, confidence estimation and personalization. Each stage can be replaced independently, which is what permits the competing-recognizer experiments of Chapter 69.

Listing 50.1: Recognition interfaces.

```
export interface Preprocessor {
  readonly id: string; readonly version: string;
  apply(raw: Sample[]): Sample[];           // pure
}

export interface Scorer {
  readonly id: string; readonly version: string;
  score(stroke: Sample[], candidates: Template[]): Score[]; // distance
  // per template
}

export interface Score { gestureId: string; templateId: string; distance:
  number; }

export type Outcome =
  | { kind: "accepted"; gestureId: string; confidence: number }
  | { kind: "ambiguous"; candidates: { gestureId: string; posterior: number
    }[] }
  | { kind: "rejected"; reason: "too_short" | "too_far" | "malformed" | "
    no_candidates" };

export interface RecognitionResult {
  outcome: Outcome;
  ranking: { gestureId: string; distance: number; posterior: number }[];
  posteriorWinner: string | null; // g*: may differ from the nearest
  // candidate
  nearest: string | null;        // g_nn
  priorOverride: boolean;
```

```
candidateSet: string[];
scope: "active" | "vocabulary";
recognizer: { id: string; version: string };
pipeline: { id: string; version: string };
library: { id: string; version: string };
}

export interface Recognizer {
  readonly id: string; readonly version: string;
  recognize(raw: Sample[], candidates: string[]): RecognitionResult;
}
```

50.2 What a result contains

A recognition result is a self-describing record. It contains the outcome, the full ranking with distances and posteriors, the candidate set that was considered, and the identity and version of the recognizer, the pipeline and the library. The contents are specified so that a result can be explained and, with the raw gesture, reproduced, without consulting the code that produced it. This is what makes a gameplay session a dataset: the pair of raw gesture and result is the unit of analysis in Part X.

Requirement 50.1 (Self-describing results). Every recognition result **MUST** identify the recognizer, pipeline and library versions that produced it, **MUST** list the candidate set and its scope, and **MUST** include the ranking from which the outcome was derived, with the posterior winner and the nearest candidate named separately.^[S]

Chapter 51

Gesture Library

51.1 Contents

The gesture library is a versioned data file, separate from the code, containing for each gesture a stable identifier, one or more reference trajectories, the invariance configuration, difficulty metadata and recognition constraints, together with the library's own name, version and the identifiers of the pipeline and the layout it was built under. Keeping the library as data allows it to be extended without a release of the program, to be replaced by a personalized version, and to be validated by the rules of Requirement 12.1 when it is loaded. The schema is given in Appendix B; an entry has the following form.

Listing 51.1: A library entry (abbreviated). The component values for the circle are computed from Definition 15.1 with the default constants; the ambiguity term and the total depend on the rest of the library.

```
{
  "id": "circle.cw",
  "family": "circle",
  "description": "Closed circle drawn clockwise",
  "templates": [{ "id": "circle.cw/0", "points": [[0.5,0.0],[0.4975,0.0498],
    "..."] }],
  "invariance": { "translate": true, "scale": [0.4, 2.5], "rotate": null,
    "reflect": false, "reverse": false, "startShift": true },
  "difficulty": { "version": "w1", "L": 0.550, "T": 0.500, "X": 0.0, "R":
    0.0, "A": "...", "C": "..." },
  "constraints": { "minLength": 160, "closed": true }
}
```

51.2 Versioning

The library follows semantic versioning with a specification of what constitutes each kind of change. A change to a template or to an invariance is a change of recognition behavior and is a minor version at least; a change to the schema is a major version; a correction of a description is a patch. Because skill and personalization are tied to a library (Chapter 32), a

profile records the library version under which each observation was made and a migration function is provided between versions.

Chapter 52

Player Profiles

A player profile is a record of what the game knows about the player. Its parts are separated by their purpose. The *progression record* holds the curriculum stage, the set of mastered gestures and the unlocked modes. The *preferences* hold accessibility settings, audio, handedness, layout and any assistance defaults. The *competence model* holds the posterior over the competence vector of Chapter 29 and the per-gesture statistics from which it is derived. The *personal dictionary* (Chapter 38) holds the vocabulary and its weights. The *personalized templates* hold the user's adapted library Λ_u and the history of updates (Chapter 23). Each part carries its own version and can be exported, reset or deleted independently.

A profile is local. It is identified by a random identifier generated on the device and not by an account, and it contains no personal data beyond what the player enters. Transfer between devices is by explicit export and import of the profile files (Chapter 53). If the player consents to the contribution of data to research (Chapter 64), a separate pseudonymous participant identifier is created for that purpose and is not linkable to the profile identifier by the research data alone.

Requirement 52.1 (Profile ownership). Every part of a profile **MUST** be exportable in a documented format, **MUST** be deletable by the player, and **MUST** be usable without any network connection or account.^[S]

Chapter 53

Local Persistence

53.1 Storage

Profiles and training data are stored in the browser on the device of the player. Small values such as settings fit the simple key-value storage of the browser. Larger and structured data, including recorded strokes and session histories, use the transactional object store of the browser (IndexedDB), which has far higher limits and supports indexed queries. Data held in browser storage is subject to eviction under storage pressure, and the specification asks for persistent storage with the corresponding request to the browser; the browser may refuse, and the application must behave sensibly in that case.^[D] Some browsers also limit the lifetime of script-writable storage for sites that the user does not visit or interact with regularly, and apply exceptions to applications that have been installed to the home screen, so that installation as a web application is recommended to players who intend to build a long-term profile.^[D]

Because browser storage can be lost, and because the motivation of the project is the preservation of the player's investment, the application encourages export. It displays the date of the last export, prompts for an export at intervals, and writes a complete profile as a single file in an open format that can be re-imported on any device.

53.2 Export, import and deletion

An export is a JSON document with a manifest listing its parts, their versions, the library and pipeline versions, and a checksum for each part. An import validates the manifest, rejects parts whose major versions it cannot migrate, and reports what it will replace before replacing it. Deletion is complete and immediate: deleting a part removes it from storage, from any caches, and from the next export, and a final control deletes the entire profile and all recorded strokes. The data stored for research, if the player has consented to it (Chapter 64), is stored under the same controls and is deleted by the same action.

Requirement 53.1 (Complete deletion). A deletion request **MUST** remove the targeted data from every store, cache and queue of the application on the device, and **MUST** remove from any research upload queue any data not yet uploaded.^[S]

Chapter 54

Performance

54.1 Budgets

Performance targets are stated as budgets and measured against a reference device class. A frame at 60 Hz has a budget of 16.7 milliseconds, and the application's own work, including simulation steps, rendering and input handling, is limited to 8 milliseconds at the ninety-fifth percentile on a reference mid-range phone of the preceding five years. The visible latency of the ink, from the pointer event to the pixel update, is limited to one frame beyond the display pipeline. A recognition on release, for the first-milestone library, is limited to 10 milliseconds at the ninety-fifth percentile, and a recognition against a lexicon of tens of thousands of words, in the later word milestones, to 50 milliseconds. Memory use during play is limited to 100 megabytes of heap, and the compressed size of the application's code and data for the first milestone to 300 kilobytes. All of these are design decisions and starting points.^[S]

54.2 Measurement

Timing uses the high-resolution clock of the browser. Browsers reduce the resolution of this clock, for security reasons, to a figure that may be as coarse as a tenth of a millisecond or worse, and the benchmark therefore repeats operations in batches and reports per-operation values computed from the batch time.^[D] The benchmark procedure is as follows. After a warm-up that is long enough for the engine to compile the code, each operation is run in at least thirty batches, and the median, the ninety-fifth percentile and the interquartile range are reported. Measurements are taken on the reference device with the application in its production build, with the browser in a fresh profile and the device at a stable temperature, since throttling under heat changes the results. The result of each benchmark is a file that records the device, the browser version, the build identifier and the figures, and the files are retained so that regressions are visible.

Chapter 55

Testing

55.1 Levels of testing

Four levels of testing are specified. *Unit tests* check the pure functions of the core package against hand-computed values, including the derived quantities of Chapter 9 and each stage of the normalization pipeline. *Property-based tests* state invariants that must hold for all inputs of a class and check them on generated cases with a library for property-based testing. *Recorded-gesture regression tests* run recorded strokes through the recognizer and compare the outcomes and distances with stored expectations. *Integration tests* replay recorded sessions through the full simulation and compare the final state, and a hash of the state at fixed steps, with stored values.

55.2 Properties

Property-based testing is especially suited to the geometry. The resampling of any stroke with positive length returns exactly N points, the first and last of which equal the first and last of the stroke. The path length of the resampled stroke is at most the path length of the original, and approaches it as N grows. Normalization is idempotent: normalizing a normalized stroke leaves it unchanged within tolerance. The distance d is symmetric and zero for identical inputs, and d_H is a pseudometric under a common group (Proposition 12.1). The matching cost c_g is tested for what it is, directional, and the confusability κ is tested for symmetry and for being zero on identical templates, with no triangle inequality asserted. The symmetric DTW distance never exceeds the pointwise distance of equal-length sequences when the diagonal lies in the band. The decision rule returns a defined outcome for an empty candidate set, a single candidate and tied posteriors. Translating or uniformly scaling a stroke does not change its normalized form. A stroke that is a template, perturbed by noise below a bound, is recognized as that template. The result of the baseline recognizer is unchanged when the timestamps of a stroke are altered monotonically (Requirement 14.1). These properties are the executable form of the specifications of Part III.

55.3 Fixtures

A regression fixture is a recorded stroke together with the expected outcome, the expected distances to a tolerance, and the margin by which the decision clears each of its thresholds. A fixture with a small margin is marked fragile, because a harmless difference in arithmetic could flip its outcome, and fragile fixtures are reviewed and not simply updated. Appendix H gives the format and the edge-case fixtures that every reference algorithm must have: degenerate strokes, empty and singleton candidate sets, ties, repeated points and conflicting invariance settings.

Chapter 56

Deployment

The application is a static site. All of its files are served from static hosting with no server-side logic, which makes it inexpensive to host, simple to mirror, and independent of any service whose discontinuation could strand its users, which is the situation the project aims to avoid. The application is built as a progressive web application: a web app manifest describes its name, icons and display mode so that it can be installed on the home screen of a phone, and a service worker precaches the application shell, the library and the assets, so that the game runs with no connection after the first visit. Updates are delivered by a versioned cache. A new version is downloaded in the background and activated at the next start, with a notice, so that a session in progress is never changed beneath the player.

Versioning is semantic and applies separately to the application, the library, the pipeline and the recognizers, since each has its own compatibility consequences for profiles and recorded data (Chapter 32). A release procedure runs the full test suite and the benchmark, compares the benchmark with the previous release, builds a reproducible production build, and publishes it with a changelog that states whether recognition behavior has changed. The application loads no third-party scripts, collects no telemetry by default, and sets a content security policy that forbids loading code from other origins; research data collection is opt-in and uses a documented endpoint described in Chapter 64.

Requirement 56.1 (Offline and self-contained). After its first load, the application **MUST** run without network access, **MUST** load no third-party code, and **MUST NOT** transmit any data unless the player has opted in to research participation.^[5]

Part IX

Advanced Input Interfaces

This entire part belongs to later development phases, after the touch-and-mouse game has become playable and stable. It is written now for one reason: the principle of device independence (Principle 5) is a promise about the architecture, and the promise is credible only if it is tested against devices that behave very differently from a finger. Each chapter takes a device, states how it would produce a trajectory, and identifies the question that the device raises for the recognizer, the difficulty measure or the curriculum.

The common structure is as follows. Every device is wrapped in an input adapter (Requirement 49.1) that emits the stroke events of Chapter 49. The adapter must supply the three things that a pointer supplies without effort: a position that evolves over time, a way to begin a stroke, and a way to end one. Where a device cannot supply them directly, the adapter synthesizes them from a control, such as a key or a button, and documents the synthesis. Because normalization resamples by arc length, differences in speed between devices do not enter the recognizer, but differences in the accuracy and the smoothness of the path do, and they are the subject of the cross-device study of Chapter 63.

Chapter 57

Keyboard-Controlled Drawing

57.1 A cursor moved by keys

A player who cannot or does not wish to use a pointer can draw with the keyboard. The adapter maintains a virtual cursor whose position is changed by held keys. Two key sets are provided: the arrow keys, and the Vim-style movement keys *h*, *j*, *k* and *l* for left, down, up and right, which are familiar to many keyboard-oriented users. A separate key, by default the space bar, acts as the pen: pressing it begins a stroke at the current cursor position, and releasing it ends the stroke. Moving without the pen held repositions the cursor without drawing, so that a gesture can be begun anywhere on the playfield.

57.2 Simultaneous directions and acceleration

Diagonal movement requires that two direction keys can be held at once, and the adapter tracks the state of every key and does not rely on key repeat events, whose rate and delay are set by the operating system and differ between machines.^[D] The velocity of the cursor is the vector sum of the unit vectors of the held direction keys, normalized so that a diagonal is not faster than an axis-aligned movement. Acceleration is configurable. With a constant speed the cursor moves at v_0 in logical units per second. With acceleration the speed grows with the time τ for which a direction has been held, $v(\tau) = \min(v_{\max}, v_0 + a\tau)$, and returns to v_0 when all direction keys are released, which allows long travel to be quick and short, careful movements to be slow. The cursor position is updated at a fixed rate, with a default of 120 samples per second, independently of the display, so that the sample density is uniform and the trajectory is a faithful record of the path.

57.3 Consequences for recognition and difficulty

A keyboard-driven trajectory has special geometry. Its segments are straight between changes of the held keys, its directions are quantized to eight when only the four keys are used, and its curves are approximated by staircases of segments. A circle is drawn as a sequence of alternations between directions, and the effort of producing a smooth circle is

large. The arc-length resampling and the smoothing of the pipeline remove much of the staircase, but a recognizer tuned on finger strokes has a tolerance that may be wrong for this device, which makes the cost of the gestures on this device a different function from the cost with a finger. The conclusion is that the difficulty measure is device-dependent, $C_{\text{device}}(G)$, and the weights of Chapter 15 are fitted per device class (Hypothesis 58.1). The curriculum may also include a vocabulary of gestures designed for the device, since a polyline with few direction changes is easy to draw with keys and a smooth curve is not.

Chapter 58

Etch A Sketch Controls

58.1 Independent axes

The Etch A Sketch is a drawing toy in which two rotary knobs control a stylus, one the horizontal position and the other the vertical, so that every line is the product of two coordinated rotations. It is a useful model for a class of inputs in which the two coordinates are controlled independently, and it appears in the project for two reasons. Such an input is easy to implement on any device with two analog controls, and it has a distinctive motor profile that tests the claim of device independence.

In an implementation with virtual knobs, the player rotates two on-screen dials, each by dragging in a circle, and the angle θ_x of one dial gives the horizontal position $x = x_0 + k_x\theta_x$, and the angle θ_y of the other gives the vertical position $y = y_0 + k_y\theta_y$. With physical controllers the two dials are two controls with a continuous range, such as the two sticks of a gamepad in position mode, two rotary encoders, or two sliders. The pen is up or down according to a button, with a convention to be fixed by the first prototype.

58.2 Motor profile

Under this control the horizontal and vertical lines are the easiest shapes, since each requires the turning of one dial, and the diagonal is harder, since it requires both dials to turn together at a matched rate. A circle requires the two dials to turn sinusoidally with a quarter-period phase difference, $\theta_x \propto \cos \omega t$ and $\theta_y \propto \sin \omega t$, which is a difficult coordination, and a figure-eight requires a two-to-one frequency ratio between the dials. The ordering of difficulty is therefore different from the ordering for a finger, in which the circle is easier than the diagonal zigzag of many segments. The same vocabulary, drawn with the same recognizer, has a different cost, and the measure of the cost for this device is the main output of the study.

Hypothesis 58.1 (Device-dependent cost). The rank order of gesture difficulty differs significantly between a finger on a touchscreen, a keyboard-driven cursor and a two-dial controller, so that a difficulty model fitted on one device predicts the order of difficulty on another substantially less well than a model fitted on the device itself.^[H]

Chapter 59

Joysticks and Gamepads

59.1 Position and rate control

An analog stick maps a displacement of the stick to either the position of the cursor or its velocity. In position control, the stick position, scaled to the playfield, is the cursor position, so that the stick behaves like a small pointing surface. In rate control, the stick displacement sets the velocity of the cursor, so that releasing the stick stops the cursor and a long movement requires holding the stick for some time. The distinction is a familiar one in the study of input devices, and the two modes have different strengths: position control is direct and fast for short, absolute movements, while rate control gives access to a large area with a small device and is used by joysticks on consoles and by pointing sticks on laptops.^[D] The gamepad adapter provides both, and the choice is a setting with rate control as the default because it matches the common experience of gamepad users.

59.2 Interface and calibration

The browser exposes gamepads through a polled interface and not an event stream, so the adapter reads the state of the connected device on each simulation step and produces samples at the simulation rate, which is a uniform and convenient sampling.^[D] Analog sticks have a dead zone, because a stick at rest does not report exactly zero, and the adapter applies a radial dead zone with a small default radius and a rescaling of the remaining range, since a dead zone that is applied to each axis independently distorts the diagonal. A calibration step records the rest position and the maximum displacement of each stick. A designated button begins and ends a stroke, preserving appropriate control over the initiation and completion of the gesture, so that the stroke is delimited by an intention and not inferred from the stick position, which would make every movement a stroke.

Chapter 60

Alternative Pointing Devices

Several other devices produce planar positions, and the pointer abstraction accommodates them with differences of detail. A *trackball* produces relative motion by the rotation of a ball, and gives smooth, scalable movement with the hand at rest; its acceleration curves are set by the operating system and make the physical distance of the movement a poor predictor of the cursor distance, which the arc-length normalization discards. A *touchpad* produces relative motion by default and may be placed in a mode with absolute coordinates; its tracking is smooth but its sampling is controlled by the driver, and some drivers apply smoothing and prediction that cannot be turned off. A *stylus* produces absolute positions with higher resolution than a finger, together with pressure and tilt, and its hover state offers a sample sequence before contact that can be recorded and used to detect the intention to begin a stroke. A *mouse* has a sampling rate that ranges from 125 Hz for basic devices to 1000 Hz or more for gaming devices, and delivers many more samples than the display consumes, which the coalesced events interface of the pointer adapter recovers. Each device can be recorded by the research tools (Chapter 64) with its identity, so that the effects of the device on recognition can be measured and not assumed.

The extended channels of the sample (Chapter 9) become useful with a stylus. Pressure may be used to delimit a stroke more accurately than a contact threshold, to distinguish deliberate strokes from incidental contact, and to define a graded quantity that a later curriculum could exploit, such as emphasis on a corner. The specification records the channel and leaves its use to future work.

Chapter 61

Eye Tracking

61.1 The difficulty of gaze as a pointer

An eye tracker provides an estimate of the point at which the user is looking, and gaze offers an input method for users who cannot use their hands. It is also the most difficult device in this part. The eye does not move smoothly: it holds still in fixations lasting a few hundred milliseconds and jumps between them in saccades, so that a trace of the gaze point is a sequence of clusters joined by rapid leaps and not a drawn line. Estimates of the gaze point are noisy and biased, with typical errors from half a degree of visual angle for dedicated hardware to larger values for a webcam, and sampling rates range from the 30 Hz of a camera to hundreds of hertz for dedicated trackers. The eye also looks at what attracts attention and not only at what it is told to draw, which is the Midas touch problem: a system in which everything that is looked at is acted upon cannot distinguish looking from commanding [14].^[D]

61.2 Delimiting strokes

The adapter must supply the start and end of a stroke by a mechanism other than a button. The options are dwell activation, in which a fixation of longer than a threshold at a start region begins a stroke; an explicit blink or a wink, which is detectable by some trackers; a switch operated by any available body movement; and a smooth-pursuit design, in which the user follows a moving target, because the eye's pursuit of a moving object produces a smooth trace that the system can match to the target's motion and that can be distinguished from ordinary looking [30].^[D] The first implementation in this direction would use dwell and switch activation, which are the simplest, and would record its pursuit variants for research.

61.3 Gestures from fixations

Because gaze is fixated, the natural representation of a gaze gesture is a polyline whose vertices are the successive fixation points and not a continuous curve. The geometric feature

recognizers of Chapter 19 are suited to this: a gesture is a sequence of landmarks with their positions, and the recognizer compares the sequence with the landmark sequences of the library, with a tolerance that matches the accuracy of the tracker. The curriculum for gaze would consist of gestures with few vertices and generous tolerances, such as lines and corners, and would reach the word shapes only when the tracker is accurate enough to discriminate the keys. Gaze data is sensitive, because it reveals attention and in some conditions identity, so the research use of any gaze data is subject to a stricter consent procedure than that of strokes (Chapter 64).

Chapter 62

Assistive Input Systems

Some users can operate only one or two switches, which may be pressed by a hand, a head movement, a breath or a muscle. Input by switch is usually by scanning: the interface highlights options in sequence, and the user selects the highlighted option by pressing the switch. A gesture game for switch users cannot be a drawing game in the continuous sense. It can nevertheless use the same vocabulary and recognizer by changing the manner in which a gesture is composed. In one scheme the player composes a gesture from primitives chosen by scanning, each primitive being a direction and a length, and the sequence is rendered as a trajectory which is passed to the recognizer like any other. In another, the player selects among balloons directly by scanning and the game provides the shape association as a visual game of matching, so that the curriculum of shapes remains available without the drawing.

The scope of this chapter is deliberately limited. The specification records the intention that the neutral trajectory interface not preclude such devices and that every function of the game be reachable by a control that a switch can operate, and it does not commit to a design. The design of assistive input is a task that requires the participation of the people who would use it, and the specification therefore defers it to a consultative process once the first milestones are complete.^[S]

Chapter 63

Cross-Device Recognition

63.1 The question

The central question of the part is whether a common gesture vocabulary can remain useful across fundamentally different input mechanisms. The question has a practical side, in that a player who changes from a phone to a laptop should not have to learn again, and a research side, in that it tests whether the vocabulary is a property of the shapes or of the hand that draws them.

63.2 Measures

Let $\text{acc}(A \rightarrow B)$ denote the recognition accuracy on strokes from device B for a recognizer trained or calibrated on device A , and let $\text{acc}(B \rightarrow B)$ be the accuracy of the matched case. The *cross-device gap* is $\Delta(A, B) = \text{acc}(B \rightarrow B) - \text{acc}(A \rightarrow B)$, and a vocabulary is *portable* between the devices if Δ is below a stated tolerance for all gestures. The gap decomposes into two contributions that the design can address differently. The first is a shift in the geometry of the strokes, so that the same gesture is drawn with different tolerance or proportions, and it can be addressed by per-device tolerance parameters and, where there is enough data, by per-device templates. The second is a shift in the difficulty, so that a gesture that is easy on one device is not on another (Hypothesis 58.1), and it cannot be addressed by the recognizer, only by the curriculum, which has to choose different gestures for different devices.

Hypothesis 63.1 (A portable core vocabulary). There exists a subset of the initial vocabulary, containing at least half of its gestures, whose cross-device gap between touch and each of the keyboard-driven and gamepad devices is below five percentage points after per-device calibration of the tolerance alone.^[H]

Requirement 63.1 (Device metadata). Every recorded stroke MUST identify the input adapter and its version, the device kind, and any per-device calibration in force, so that the cross-device analysis can be performed from the records alone.^[S]

Part X

Experimental Research Program

This part describes how the claims of the preceding parts are to be tested. The game is the instrument: every stroke it records is a measurement, and the architecture of Part VIII was designed to keep those measurements raw, versioned and reproducible. The part moves from the collection of data to the questions it can answer, and ends with the conditions under which the data and the analyses can be shared.

Chapter 64

Gesture Dataset Construction

64.1 Consent and minimization

Data collection is optional and requires an affirmative decision by the player, made at a moment when the player has been told what is recorded, for what purpose, how long it is kept, and how to withdraw. The consent screen is separate from the game and is not required to play. Collection follows data minimization: only what the research questions need is recorded. For each stroke the record contains the raw samples with timestamps and any extended channels, the target label and the candidate set, the recognition result, the identifiers of the recognizer, pipeline, library and layout, the mode and the session seed, and the device characteristics of Requirement 63.1 including the pointer type, the viewport size, the pixel density and the orientation. The record contains no name, no email address, no network address that the application itself stores, no free text beyond the task, and no location. The participant is identified by a random pseudonymous identifier that the player can reset.

64.2 Gesture data is biometric

Timing and shape of strokes are individually distinctive. Research on touch input has shown that the way a person swipes and taps can serve as a behavioral biometric for continuous authentication, with identification accuracy well above chance on small populations [9].^[D] The practical consequence is that stroke data cannot be treated as anonymous merely because it carries no name: with enough strokes it may be linked to a person, or to another dataset of the same person. The data is therefore handled as personal data. It is stored with access controls, transmitted over an encrypted connection, kept for a stated period, and covered by the right of the participant to withdraw and to have the data deleted. The relevant legal frameworks depend on the participant's jurisdiction, and examples are the Canadian PIPEDA and the European GDPR, with which the procedure is designed to be consistent.

64.3 Oversight

Research on human participants ordinarily proceeds under the oversight of an ethics review board. This project is conducted by an independent researcher and has no institutional board. The specification therefore commits to a published consent and data-handling protocol, to a plain-language description of the study that the participant sees before consenting, and to inviting external review of the protocol before data collection on a large scale begins, by an ethics committee or a qualified reviewer who is independent of the project. Studies involving children, or gaze data, are not to be started without such review.^[S]

Requirement 64.1 (Consent-based collection). Research data collection **MUST** be off by default, **MUST** require a separate affirmative consent that states what is collected and for how long, **MUST** be withdrawable at any time with deletion of data not yet released, and **MUST NOT** be required for any function of the game.^[S]

Chapter 65

Human Variation

People draw the same shape in different ways, and the extent and structure of the differences is a central object of study. Four kinds of variation are separated. *Geometric variation* is the difference in the paths of different people for the same gesture, summarized by the distance between a person's mean normalized stroke for a gesture and the population mean, and by the dispersion of strokes about a person's own mean. *Timing variation* is the difference in duration, speed profile and pauses. *Consistency* is the within-person dispersion of strokes of a gesture, and is the quantity that determines how tight the recognizer's tolerance can be for that person. *Preferred abbreviations* are the habitual reductions of the strokes (Chapter 31).

The appropriate model is hierarchical. A stroke G_{ugk} of person u , gesture g and repetition k is modeled as arising from a population template for the gesture, a person-level deviation, and within-person noise. In symbols, with normalized strokes treated as points in a vector space of dimension $2N$,

$$G_{ugk} = T_g + b_{ug} + \epsilon_{ugk}, \quad b_{ug} \sim \mathcal{N}(0, \Sigma_b), \quad \epsilon_{ugk} \sim \mathcal{N}(0, \Sigma_\epsilon),$$

so that the ratio of the between-person to the within-person variance, per gesture, says how much is gained by personalizing the templates and how much is lost by treating all users alike. The analysis also examines variation with physical conditions, notably the screen size and the scale of the stroke (Chapter 45), handedness, and the pointer type.

Hypothesis 65.1 (Personalization gain tracks between-person variance). The recognition accuracy gained by personalizing templates is larger for gestures with a higher ratio of between-person to within-person variance, and negligible for gestures with a low ratio.^[H]

Chapter 66

Learning Curves

66.1 Measurement

The learning of a gesture is observed as the change in the measures of Chapter 30 and in the success probability over repeated presentations. Because the study aims to cover days, weeks and months, the design uses the natural play of the game and not scheduled sessions alone, with a small number of standardized probe tasks at intervals, in which the same gestures are presented in the same conditions so that the comparison across time is free of variation in the difficulty of what was shown. The probe tasks use a frozen recognizer (Requirement 25.1).

66.2 Models

For each person and gesture several functional forms are fitted: the power law of Chapter 25, an exponential $T(n) = a + b e^{-cn}$, and a hyperbolic form, and they are compared by predictive accuracy on held-out repetitions, since the form of the curve is an empirical question and the aggregate-curve caveat applies. The fit is hierarchical, with parameters for the person and the gesture drawn from population distributions, which stabilizes the estimates for people with few observations. The quantities of interest are the asymptote, the rate, and the time to reach the mastery criterion.

66.3 The control that matters

A learning curve shows that players improve, and not why. For the project's central question, whether the geometric curriculum yields transferable gesture-typing skill, the necessary comparison is between players trained on the geometric curriculum and players given an equal amount of direct word-gesture practice, with a third group on a non-gesture control for the effect of play alone (protocol P7). The comparison is on unseen words, without a guide, at the end of training and after a retention interval, and it is separated into the three skills of tracing, recall and generation (Chapter 32). This is the most consequential empirical question in the specification, and the first release is built so that it can be asked: sessions

record the arm, the scope of every recognition and the full raw strokes.^[H]

66.4 Attrition

Longitudinal studies of voluntary play suffer from attrition: players stop. The analysis cannot treat the people who remain as representative of the people who began. The record therefore notes the last activity of each participant, the analysis reports the number of participants contributing to each time point, and a sensitivity analysis compares the estimated curves under assumptions about why players stop. The statements about months of practice are restricted to the subset of players who continue, and are described as such.

Chapter 67

Human Adaptation versus Model Adaptation

67.1 The decomposition

The identification problem of Definition 25.1 is addressed by experimental design. Let $y(t, v)$ be the success rate for a given user population at time t of the user's practice, scored by recognizer version v . Two times $t_1 < t_2$ and two versions v_1, v_2 define a table of four values. The total change from (t_1, v_1) to (t_2, v_2) decomposes as

$$y(t_2, v_2) - y(t_1, v_1) = \underbrace{[y(t_2, v_1) - y(t_1, v_1)]}_{\Lambda \text{ (learning)}} + \underbrace{[y(t_1, v_2) - y(t_1, v_1)]}_{M \text{ (model)}} + \underbrace{I}_{\text{interaction}},$$

where $I = y(t_2, v_2) - y(t_2, v_1) - y(t_1, v_2) + y(t_1, v_1)$. The learning term is the change in success caused by the change in the user's strokes when the recognizer is held fixed at v_1 , and the model term is the change caused by the new recognizer applied to the earlier strokes. The interaction measures the extent to which the benefit of the new recognizer depends on how practiced the user is, and a large positive value would mean that adaptation helps experts more than novices, or the reverse.

67.2 Replay and its limits

The terms can be estimated by replay, because the raw strokes are preserved: strokes recorded at time t are re-scored by any recognizer version v offline, and the table is filled by computation. The method has a limit that must be stated. The strokes recorded under recognizer v_1 were drawn by a player who was receiving feedback from v_1 , and who may have adjusted their strokes in response, so that the recording is not what the player would have drawn under v_2 . Replay holds the strokes fixed and cannot capture this closed-loop effect. The remedy is a live experiment: the recognizer version, frozen or adaptive, is assigned at random to each session, or to each participant for a period, and the outcomes are compared between arms. The two designs answer different questions and both are used.

Replay isolates the effect of the recognizer on a given set of strokes, and the randomized design measures what the player does when the recognizer is different.

The randomized design must respect a property of the subject that a recognizer does not share: a player cannot be reset. A crossover, in which every participant experiences both arms in sequence, confounds the arm with the period and with carryover, because skill acquired under the first arm persists into the second. Counterbalancing the order balances period effects across the sample on average, but it does not remove carryover, which can differ between the two sequences, and it cannot restore the unfamiliarity of the gesture set. The primary design is therefore parallel, with each participant assigned to one arm for the whole study, and a crossover is used only as an exploratory secondary design with a washout whose adequacy is verified, a model with explicit period and sequence terms, and a pre-specified rule for discarding the second period (protocol P3).

Requirement 67.1 (Experimental assignment). Studies of adaptation **MUST** record the arm, the recognizer version and its adaptive state at the start and end of every session, and assignment to arms **MUST** be randomized and recorded, with the randomization seed held separately from the recognizer.^[5]

Chapter 68

Recognition under Degraded Information

How much of a gesture can be removed before recognition fails is a measure of its redundancy and of the robustness of the recognizer, and it bears directly on partial recognition (Chapter 22) and on the choice of sampling rate and resolution for devices of low quality. The experiments degrade a set of strokes by a set of controlled operations and record the accuracy as a function of the degradation. The operations are truncation of the beginning or of the end of the stroke to a fraction f of its length; deletion of a contiguous portion; deletion of random samples to a fraction f ; subsampling in time to a lower rate; quantization of positions to a coarser grid, simulating low spatial resolution; and the addition of Gaussian noise of a given standard deviation.

For each operation the curve of accuracy against the retained fraction is estimated for each recognizer. The summary quantity is the *critical fraction*, f^* , the smallest retained fraction at which the accuracy remains at least 95 percent of its value on undegraded strokes. A recognizer with a small f^* under truncation is suitable for early commit. The occlusion profiles of Chapter 13 are compared with the degradation curves: if the profile identifies the informative portions correctly, deleting the portions of high informativeness should reduce accuracy faster than deleting those of low informativeness at the same length, and this is a test of the profile and of its use in teaching abbreviation.^[H]

Chapter 69

Competing Recognizers

The recognizers of Part IV are to be compared on equal terms. The contestants are the baseline template matcher of Chapter 17, the elastic matcher of Chapter 18, the geometric feature recognizer of Chapter 19, the probabilistic recognizer of Chapter 20, and personalized variants of each. Established gesture recognizers from the literature are included as external baselines where their published definitions can be implemented faithfully, in particular the single-template stroke recognizers that are widely used in prototyping [32, 29] and the feature-based statistical recognizer of Rubine [23].^[D]

The comparison follows a fixed protocol. All contestants receive the same recorded strokes, the same normalized inputs where the algorithm permits, and the same candidate sets. Hyperparameters are chosen on training folds by nested cross-validation, with equal budgets. Evaluation follows Chapter 24: leave-one-user-out splits, cluster-bootstrap intervals, and the full set of measures including latency, rejection quality and calibration. Paired differences between two recognizers on the same strokes are tested with a paired procedure, such as the test of McNemar for paired classification outcomes [21],^[D] with the caveat that strokes from one person are correlated, so that the bootstrap over participants is the primary inference and the paired test is supporting.

Two principles guard against self-deception. The recognizer that the project prefers for its simplicity, the baseline, receives the same tuning effort as the others. And a recognizer that wins on accuracy but loses on explanation is not declared the winner without a statement of what the game loses: the geometric feature recognizer can tell the player which part of a stroke failed, and for a learning game that capability may outweigh a small loss of accuracy.^[S]

Chapter 70

Historical Swype Comparison

70.1 Why compare

The project began from a remembered capability, and the comparison with surviving Swype installations is an attempt to turn memory into measurement. The questions are the behaviors reported in Chapter 2: the tolerance of the recognizer to inaccurate or abbreviated strokes, the effect of a personalized vocabulary, the behavior of gesture shortcuts, and the effects of long use on recognition.

70.2 Feasibility and method

The comparison is feasible only where legitimate copies of the software can be run, and its feasibility depends on obtaining them lawfully. It is not assumed that any copy can be obtained or redistributed, and the project does not distribute the software. Where a copy is available, it can be run on a device or an emulator of the appropriate version of the operating system, and it is a black box: the experimenter can supply strokes and observe the output text and, in some versions, the list of alternatives, but cannot observe internal scores. The method is therefore behavioral. A harness presents the same strokes to the historical software and to the new recognizer. Strokes are generated from the idealized word paths with controlled perturbations, which are noise, abbreviation operators (Chapter 31) and truncation, and they are injected by the input system of the device with timing that mimics a human stroke. The measured quantity is the acceptance curve: the fraction of words recovered as a function of the degree of perturbation, for each perturbation type. Personalization effects are measured by comparing the acceptance of words in the default lexicon with the acceptance of words that have been added by use.

70.3 Limits of inference

The comparison is kept as a software-behavior experiment (protocol P6). Human performance on surviving installations, which would ask how experienced users actually stroked, is a different study with different participants and different inferences (protocol P8), and

it is not required by any milestone. Conclusions from this comparison are observations about the behavior of one version of one proprietary product, under synthetic strokes. They cannot establish how the product worked internally, and they say nothing about how real users stroked. They are labeled ^[O] in this specification and are used to calibrate the design and to test the recollection that early selection and abbreviation were tolerated. Appendix K lists the claims that this experiment can and cannot support.

Chapter 71

Reproducibility and Open Datasets

71.1 Artifacts

Reproducibility rests on a small set of public artifacts. The *recognizers and the evaluation harness* are released as open source with fixed version identifiers. The *libraries, layouts* and *lexicons* used in studies are released with checksums. The *protocols* of the studies, including the consent text, are published before the data collection begins, and deviations are recorded. The *analysis code* is published with the results it generated, and a result is not reported without the command that regenerates it from the data. The *datasets* are released in the form described below.

71.2 Documenting data

Each dataset is accompanied by a datasheet that records its motivation, composition, collection process, preprocessing, intended uses, limitations, distribution and maintenance, following the practice of documenting datasets proposed for machine learning research [10].^[D] The datasheet is part of the release and is versioned with the data. A manifest gives the checksum of each file, and the splits used in published results are included.

71.3 Privacy-aware release

The release of raw strokes is constrained by Chapter 64: they are biometric, and perfect anonymization of time-series data of this kind cannot be guaranteed. The release is therefore tiered. At the first tier, aggregate statistics, learning-curve parameters and per-gesture summaries are public without restriction. At the second tier, normalized strokes with timing removed and with participant identifiers replaced by randomly assigned codes that are not reused across releases are public with a documented warning. At the third tier, raw strokes with full timing are released only to qualified researchers who agree to terms that prohibit re-identification, unless a participant explicitly consents to open release of their own data, in which case their strokes may be released in the open tier. The tiers, and the participants' choice among them where it applies, are stated in the consent procedure. Because the work

of the project is otherwise placed in the public domain, the default license for public artifacts is a public domain dedication, and the exceptions are those data for which the consent does not permit it.^[S]

Part XI

Future Applications

The applications described in this part are consequences of the architecture and not goals of the first milestones. They are included to state the requirements that they would place on the recognition engine, so that nothing in the earlier design makes them harder than necessary, and to mark where their feasibility depends on conditions outside the project's control.

Chapter 72

Standalone Gesture Keyboard

72.1 From engine to keyboard

A gesture keyboard is a recognition engine joined to a keyboard layout, a lexicon, a language model and a text-editing interface. Almost everything but the interface already exists in the specification by the end of Milestone 4: the layout as data (Chapter 33), word trajectories and their ambiguity (Chapters 34 and 39), the probabilistic recognizer and the commit rules (Chapters 20 and 22), the personal dictionary (Chapter 38) and the sentence semantics (Chapter 40). What a practical keyboard adds is a set of constraints on the engine. It must respond with imperceptible delay on a modest device, so the recognition against a full lexicon must meet the budget of Chapter 54, which a naive scan of every template does not and which therefore requires indexing, a prefix structure over the lexicon, and the lower bounds of Chapter 18. It must be multilingual, with layouts and lexicons for other languages and with the composition of accented characters. It must handle the many kinds of text a person writes, which include numbers, symbols, emoji and mixed-language text, for which a gesture is inappropriate and the keyboard must fall back to taps.

72.2 Platform constraints

A keyboard that is used in other applications must run as an input method on the operating system, and the routes to this are controlled by the platform. On Android an input method is a service that the user installs and selects, and it can take strokes across the whole screen of the keyboard. On iOS a custom keyboard is an extension with limits on memory and on network access, and access to the network and to shared storage requires an explicit permission by the user. A web application cannot be a system keyboard. A keyboard has access to everything that is typed, including passwords and private messages, so its privacy design is not optional: the recognition and the personal dictionary run on the device, the application does not transmit what is typed, and the personal dictionary remains in the open, user-readable format of Chapter 38 and exportable. The constraints of the platforms are those of the time of writing and must be reviewed again before any work on this milestone.^[S]

Chapter 73

Gesture Command Languages

A gesture keyboard is capable of more than words. Strokes can express commands such as capitalizing a word, selecting text, copying, cutting, pasting, moving the cursor and deleting. The earlier research on shorthand gestures applied this idea to commands by letting a user enter a command by drawing the shape of its name on the keyboard after a distinguished activation, so that the vocabulary of commands is the vocabulary of words and is learned through the same skill [17].^[D] The design problems are those of mode and of collision. A command stroke must be distinguished from a word stroke, which is accomplished by an explicit activation (a dedicated key, a region of the keyboard or a modifier held with another finger) and not by recognition alone, because a command that is mistaken for a word, or the reverse, is expensive in a way that an ordinary recognition error is not. And a command vocabulary must not collide with the lexicon, which the library validation of Requirement 12.1 enforces by treating commands as entries of the same library.

Beyond the commands of conventional editing, the engine permits a language of gestures that is learned in the game and used in the keyboard. Capitalization, for example, may be a short stroke upward at the start of a word, and a space or a punctuation mark may be a stroke at the end of one. These conventions are designs for later milestones. The specification records the requirement that gesture commands be defined as library entries, with invariance and difficulty metadata, and be introduced by the same curriculum as the other gestures, so that the player's skill in the arcade game is the skill that the keyboard uses.

Chapter 74

Personalized Motor Vocabularies

The player may define gestures of their own. A user-defined gesture is created by drawing a shape several times, from which a template and a tolerance are estimated, and by assigning it a meaning, such as a word, a phrase, a command or a macro. The creation procedure enforces the separation floor of Requirement 3.1: a new gesture too close to an existing one is refused, with the nearest competitor shown, and the player may redraw the new shape, choose another or accept an explicit relaxation of the neighbor's tolerance. The number of repetitions required is set by the within-person variance of the strokes, so that a person who draws consistently needs fewer.

Two questions are worth stating. The first is whether the learned associations of a person can be transferred between applications and devices. The answer is conditional on the exchange format: a vocabulary exported in the open format of Appendix B, with the library version, layout version and the pipeline it was built under, can be imported into any conforming implementation, and the recognizer then performs the same computation. The second is whether transfer is meaningful for the player, whose skill is tied to the behavior of one recognizer (Chapter 32), so that moving the vocabulary to a different recognizer gives it a different meaning. The specification therefore treats portability of data and portability of skill as separate properties, and proposes to measure the second. A vocabulary is accompanied by a conformance test, a set of recorded strokes with their expected outcomes, so that a receiving implementation can verify that it interprets the vocabulary as the sender's did.

Chapter 75

Input-Independent Interaction

The broadest application is an interface whose vocabulary is independent of the device. If a stroke is a trajectory, and a trajectory can be produced by a finger, a mouse, a keyboard, a joystick, a pair of dials or a gaze, then a vocabulary learned with one can be used with another. A person with a temporary injury of the hand, or a permanent disability, or a person using an unfamiliar device, would retain their skill, subject to the device-dependent cost of Hypothesis 58.1 and the portability of Hypothesis 63.1.

The possibilities beyond text entry include navigation of menus by shapes, the control of applications by learned command gestures, and the use of gestures as a lingua franca between assistive devices and ordinary software. These ideas are speculative, and some will fail for reasons that cannot be seen from here: the expressiveness of gesture vocabularies may not scale beyond a few hundred entries without becoming a memory burden, and for many tasks a visible menu is simply better. The project does not rely on them. What the architecture contributes is the conditions under which they could be tested cheaply, namely an open trajectory representation, a recognizer that is separable from any application, a transparent personal vocabulary and a research program that measures transfer and cost instead of assuming them. If gesture-based computation becomes a way of acquiring fluency that survives the abandonment of any particular product, it will have done so through those properties, which are the substance of the specification.^[S]

Part XII

Implementation Roadmap

The milestones order the work so that each ends in something that can be tried. The first is the immediate engineering target, and the later ones are stated at the level of detail that is useful now, with the expectation that each will be refined when the preceding one is complete. For each milestone, the exit criteria are the conditions under which it can be considered complete, and they are stated so that a judge other than the author could decide.

Chapter 76

Milestones

Milestone 1: Playable arcade prototype

The first milestone is the playable prototype and is the first implementation target. It provides touch and mouse input through the pointer adapter, descending balloons with the behavior of Chapter 5, a vocabulary of ten to twenty geometric gestures from Table 10.1, the baseline recognizer of Chapter 17 with the decision rule of Chapter 21, scoring as in Chapter 7, and a restart after the loss of the last life. It includes the survival mode only, the accessibility baseline of Requirement 46.1, local storage of settings and best score, installability and offline play. It includes recording of raw strokes and session replay, since these are cheap to build early and costly to retrofit. The exit criteria are that a new player can begin without instruction, that the median latency from release to outcome is within budget on the reference device, that a recorded session replays to the same final state, that the recognizer attains the accuracy target on the synthetic and the first recorded sets, and that a group of players asked to play for ten minutes report that they would play again, which is the only test of the independence of enjoyment (Requirement 1.1) that the milestone can run.

Milestone 2: Expanded gesture vocabulary

The second milestone adds more complex shapes, the ambiguous and rejected outcomes with their feedback, the practice mode, and the replay of recorded gestures for inspection. It introduces the confidence calibration and the evaluation harness of Chapter 24 and the first comparisons of recognizers on recorded data. The exit criteria are that the vocabulary reaches the full twenty gestures at the separation floor, that the three outcomes are distinguishable by players in a test of the kind described in Chapter 43, and that the harness reproduces a published table of results from a fixed dataset.

Milestone 3: Adaptive gesture curriculum

The third milestone adds the competence model, the selection of challenges, the gesture history, and the measurements of learning over time, together with the optional and consent-based data collection of Chapter 64. The exit criteria are that the competence model predicts held-out success with better calibration than a model with no player term, that the curriculum can be run in a frozen-recognizer arm, and that the consent and deletion procedures have been externally reviewed.

Milestone 4: Word trajectories

The fourth milestone introduces the keyboard-derived shapes of Part VI: word balloons, the lexicon and personal dictionary, keyboard overlays with fading assistance, transitional exercises and the typing exercises and measures of Chapter 41. The exit criteria are that word shapes are generated reproducibly, that the recognizer meets its latency budget against a lexicon of the target size, and that a first study of the curriculum reports its effect on typing measures with the uncertainty that the sample permits.

Milestone 5: Alternative input systems

The fifth milestone adds the adapters of Part IX: arrow-key and Vim-style cursor control, two-dial controls, joysticks and gamepads, and experimental gaze input. The exit criteria are that each adapter satisfies the adapter contract (Requirement 49.1), that the cross-device study of Chapter 63 has produced a first estimate of the gap and of the cost differences, and that assistive devices have been considered in consultation with the people who would use them.

Milestone 6: Research and standalone applications

The sixth milestone carries out the comparative recognition experiments of Chapter 69, develops personalization beyond the adaptation of the first design, packages the recognition engine as a reusable library with a stable interface, and evaluates a standalone gesture keyboard (Chapter 72). The exit criteria depend on results that do not yet exist and are not stated here.

Chapter 77

Scope of the First Implementation

Although the complete specification may be long, the first implementation should be deliberately small. It should contain a functioning playfield, approximately twenty gestures, a transparent recognition algorithm, and enough progression to establish whether the core interaction is enjoyable. Everything else in this document is conditional on that answer. A specification that describes in detail a game that no one wants to play has no value, and it is better to learn this from a small prototype than from a large one.

77.1 Three architectural commitments

Three decisions are cheap to make correctly at the beginning and expensive to repair later, and they are fixed here as commitments of the first implementation.

Principle 7 (Recognition independent of gameplay). The recognizer is a library that knows nothing about balloons, scores or levels. It receives strokes and candidate sets and returns results, and it can be used unchanged in a typing tutor, a keyboard or another application.

Principle 8 (Raw input distinct from interpreted gestures). The system preserves enough information to investigate recognition errors and to compare algorithms, and never rewrites its recorded observations. Every interpretation is a derived object carrying the identity of the procedure that produced it.

Principle 9 (Learning distinct from adaptation). Players become better at producing gestures, and the software can become better at interpreting them. These are separate processes, and they remain experimentally distinguishable because every adaptive behavior can be frozen and every session records the state of the recognizer.

The first corresponds to the package structure of Chapter 47, the second to Requirement 9.1, and the third to Requirement 25.1 and the designs of Chapter 67.

77.2 The coherent progression

Swype Hero does not have to begin as a typing tutor. It can begin as an entertaining arcade game in which the player acquires a progressively richer vocabulary of movements, and

those movements can later become the geometry of written language. The progression is from drawing shapes, to recognizing shapes, to mastering word trajectories, to producing language through learned movement. This outline is therefore treated as the table of contents of a substantial, versioned specification, while the first playable prototype remains the immediate engineering objective.

Part XIII

Appendices

Appendix A

Notation and Symbols

Symbol	Meaning
$G = \{(x_i, y_i, t_i)\}_{i=1}^N$	A gesture: planar samples with timestamps (Chapter 9)
$\ell(G), s_i$	Total path length; cumulative arc length at sample i
$\theta_i, \tau_i, \kappa_i$	Heading, turning angle and discrete curvature
v_i	Instantaneous speed
$T, T_{g,m}$	A template; the m th template of gesture g
Λ, Λ_u	The gesture library; the personalized library of user u
N	Number of points after resampling (default 64)
d	Base distance: mean pointwise Euclidean distance between normalized sequences
d_H	Quotient pseudometric under a common isometry group H (Proposition 12.1)
$\mathcal{S}_g, c_g(G, T)$	Allowed transformation set of gesture g ; directional matching cost of a stroke against a template of g
$\kappa(T_i, T_j)$	Symmetric confusability distance between templates of distinct gestures
$\delta_{\min}$	Separation floor (Definition 3.1)
$\text{sep}(\Lambda)$	Minimum pairwise κ over templates of distinct gestures
g^*, g_{nn}	Posterior winner; geometrically nearest candidate (Definition 21.1)
$\hat{G}, T^0$	Stroke aligned to the canonical template; canonical template of a slot
$C(G)$	Complexity (Definition 15.1) with components L, T, X, R, A
$\hat{T}(g)$	Predicted execution time of gesture g
$P(w G)$	Posterior of target w given gesture G
λ	Weight of the prior in the posterior
$\tau_{\text{acc}}, \mu, d_{\text{max}}$	Acceptance threshold, margin and absolute distance gate
$\eta, \rho_{\text{drift}}$	Template adaptation rate; drift radius
$\text{EUF}(T)$	Earliest unambiguous fraction (Chapter 22)
d_{loc}	Location-channel distance between word paths, in key widths
W_p, H_p	Logical width and height of the playfield

Constants that await calibration

Requirements that depend on a constant tagged ^[C] are testable only relative to a supplied value. The constant, the distance to which it applies, its unit, and the procedure that fixes it

are listed so that each can be implemented as a parameter now and set from data later.

Constant	Applies to	Unit	Fixing procedure
$\delta_{\min}$	κ	normalized-stroke	Smallest κ at which players and the baseline separate two gestures with a stated error (P1)
$d_{\max}$	d_{g^*}	normalized-stroke	Ninety-fifth percentile of genuine strokes in the calibration set
$\tau_{\text{acc}}, \mu, p_{\min}$	posterior	probability	Risk–coverage curve at a fixed ambiguity rate (P1)
σ_w, Θ	likelihood, softmax	normalized-stroke; none	Maximum likelihood per gesture; temperature on held-out data
$d_{\text{learn}}, d_{\text{new}}$	c_g	normalized-stroke	Quantiles of genuine strokes of each gesture
$\rho_{\text{drift}}, \eta$	d in canonical frame	normalized-stroke; none	Largest drift that preserves the floor on recorded libraries (P3)
Band half-width w	DTW	samples	Tuned on training folds (Chapter 69)
s_0 , weights of C	complexity	normalized-stroke; none	Fit to predicted execution time (Chapter 66)

Appendix B

Gesture Library Schema

The library is a JSON document whose structure is given below as TypeScript declarations, which serve as the normative description, followed by the requirements that a loader must enforce. All numbers are IEEE double-precision values in the units stated. Template points are stored already normalized by the pipeline named in the library header.

Listing B.1: Normative schema of a gesture library.

```
export interface GestureLibrary {
  schema: "swypehero.library/1";
  id: string; // e.g. "core-shapes"
  version: string; // semantic version
  pipeline: { id: string; version: string; points: number }; //
    normalization used
  layout?: { id: string; version: string }; // for word
    libraries
  separationFloor: number; // delta_min under which the library was
    validated
  gestures: GestureEntry[];
}

export interface GestureEntry {
  id: string; // stable identifier, unique in the
    library
  family: string;
  description: string;
  templates: { id: string; points: [number, number][] }[]; // length =
    pipeline.points
  invariance: Invariance;
  difficulty?: { version: string; L: number; T: number; X: number; R: number
    ; A: number; C: number };
  constraints?: { minLength?: number; maxLength?: number; closed?: boolean
    };
  meaning?: { kind: "symbol" | "word" | "command"; value: string };
}

export interface Invariance {
  translate: boolean;
  scale: [number, number] | null; // allowed ratio range, null = exact
```

```
    size
rotate: [number, number] | null; // allowed radians, null = none
reflect: boolean;
reverse: boolean;
startShift: boolean;           // cyclic start-point shift, closed
    shapes only
}
```

Loader obligations. A loader **MUST** check that each template has exactly the number of points declared in the pipeline header; that identifiers are unique; that each gesture has at least one template; that the scale and rotation ranges are well formed; that the library satisfies Requirement 12.1 at its declared separation floor; and that a closed-shape flag is accompanied by a start-shift invariance only when the template is closed to within a stated tolerance. A loader **MUST** refuse a library whose major schema version it does not support, and **SHOULD** report every violation and not only the first.

Exchange of personal vocabularies. A personal vocabulary is a library whose entries carry a meaning and whose header names the library from which it was derived. Together with a conformance set (Appendix H), it is the unit of exchange between implementations (Chapter 74).

Appendix C

Reference Algorithms

The algorithms below are reference definitions. An implementation may be organized differently, provided it computes the same values within the tolerance of Requirement 17.1.

Listing C.1: Arc-length resampling and normalization.

```
function resample(P, N):                                     // P: points with len(P) >= 2, N
    >= 2
    cum[0] = 0
    for i = 1 to len(P) - 1:
        cum[i] = cum[i-1] + dist(P[i-1], P[i])
    total = cum[len(P) - 1]
    if total == 0: fail("zero length")
    j = 0
    for k = 0 to N - 1:
        s = total * k / (N - 1)
        while j < len(P) - 2 and cum[j+1] < s: j = j + 1
        seg = cum[j+1] - cum[j]
        u = 0 if seg == 0 else (s - cum[j]) / seg
        Q[k] = P[j] + u * (P[j+1] - P[j])
    return Q

function normalize(raw, N):
    P = condition(raw)                                     // drop repeated positions, non-
    increasing times
    P = smooth(P)                                           // named filter, may be identity
    if pathLength(P) < L_MIN or extent(P) < S_MIN: fail("too small")
    Q = resample(P, N)
    S = max(width(Q), height(Q))
    c = centroid(Q)
    return [ (q - c) / S for q in Q ]
```

Listing C.2: Template matching under an invariance group.

```
function score(G, library, candidates):                   // G normalized
    result = []
    for g in candidates:
        best = infinity
        for T in library[g].templates:
```

```

        for s in allowed(library[g].allowedSet, G, T): // angle clamped
            to the declared interval,
                                                    // flips,
reversal, shifts, as the candidate allows
        best = min(best, meanPointDistance(s(G), T)) // matching
cost c_g(G, T)
    append (g, best) to result
sort result by distance, ties by library order
return result

```

Listing C.3: Band-constrained dynamic time warping.

```

function dtw(a, b, w): // Sakoe-Chiba band of half-width
    w
    n = len(a); m = len(b); w = max(w, abs(n - m))
    D[0][0] = 0; all other D = infinity
    for i = 1 to n:
        for j = max(1, i - w) to min(m, i + w):
            c = dist(a[i], b[j])
            // symmetric weights: path weights sum to exactly n + m
            D[i][j] = min(D[i-1][j] + c, D[i][j-1] + c, D[i-1][j-1] + 2 * c)
    return D[n][m] / (n + m) // weighted average cost along
the optimal path

```

Listing C.4: Posterior and decision rule (Definition 21.1).

```

function decide(dist, prior, P): // dist[g]: matching distance of
each candidate g
    if candidates is empty: return Rejected("no_candidates")
    for each g: logL[g] = -n_eff * log(sigma[g]) - dist[g]^2 / (2 * sigma[g]^2)
    logp[g] = logL[g] + lambda * log(prior[g])
    p = softmax(logp / Theta) // Theta: calibration temperature
    // posterior winner; ties by smaller distance, then library order
    gstar = argmax over g of (p[g], -dist[g], -libraryIndex[g])
    gnn = argmin over g of (dist[g], libraryIndex[g])
    p2 = 0 if only one candidate else max{ p[g] : g != gstar }
    if dist[gstar] > D_MAX: return Rejected("too_far", nearest = gnn)
    if p[gstar] >= TAU_ACC and p[gstar] - p2 >= MU:
        return Accepted(gstar, p[gstar], priorOverride = (gstar != gnn))
    return Ambiguous({ g : p[g] >= P_MIN })

```

Listing C.5: Early commit on a stroke prefix.

```

function onSample(stroke, library, candidates, state):
    if arcLength(stroke) < L_PREFIX_MIN: return Continue
    scores = prefixScores(stroke, library, candidates) // open-end
alignment
    r = decide(scores, prior, P)
    if r is Accepted(g) and g == state.lead and p >= TAU_EARLY and margin >=
MU_EARLY:

```

```
        state.run = state.run + 1
    else:
        state.run = 0; state.lead = leader(scores)
        // fraction of the committed gesture's TEMPLATE consumed; the intended
        length is unobservable
        if state.run >= H: return Commit(g, fraction = arcLength(stroke) /
        templateLength(g))
        return Continue
```

Listing C.6: Bounded template update (Chapter 23).

```
function update(T, T0, G, allowedSet, eta, rho, library, delta_min):
    Ghat = alignTo(T0, G, allowedSet) // fixed canonical frame
    : attains  $c_g(G, T0)$ 
    T1 = (1 - eta) * T + eta * Ghat // pointwise
    interpolation, same N
    r = meanPointDistance(T1, T0) // plain distance, no
    minimization
    if r > rho: T1 = T0 + (rho / r) * (T1 - T0) // exact: the distance
    is a norm in this frame
    lam = 1
    while lam >= 1/64: // floor is an
    independent check on kappa
        Tc = T + lam * (T1 - T) // T and T1 are in the
        ball, so is Tc
        if floorHolds(library with T replaced by Tc, delta_min): return Tc
        lam = lam / 2
    return T // reject the update
```

Appendix D

The Simulation Step

The main loop is given in Listing 48.1. The step function advances the state by one fixed interval, and is a pure function of the state, the events and the generator.

Listing D.1: One simulation step.

```
function step(state, events, dt):
    state.time = state.time + dt
    for e in events:
        if e.kind == "begin": state.stroke = newStroke(e)
        if e.kind == "sample": state.stroke.append(e.sample)
        if e.kind == "cancel": state.stroke = none
        if e.kind == "end":
            state.stroke.append(e.sample)
            result = recognizer.recognize(state.stroke.raw, candidatesOf(
state))
            state = applyOutcome(state, result)           // pop, ambiguity,
rejection; score; streak
            log(state.stroke.raw, result)
            state.stroke = none
    for b in state.balloons: b.y = b.y + speed(state, b) * dt
    for b in state.balloons where b.y >= state.failureLine:
        remove b; state.lives = state.lives - 1; state.streak = 0;
    recordMiss(b)
    if state.time >= state.nextSpawn and count(state.balloons) < state.
maxBalloons:
        state = spawn(state, state.rng)                 // seeded generator
only
    if state.lives == 0: state.phase = "over"
    return state
```

Appendix E

Coordinate Normalization Procedures

E.1 Device to logical coordinates

Let the drawing element have its top-left corner at (o_x, o_y) in CSS pixels relative to the viewport and a width of w_{css} CSS pixels. A pointer event with client coordinates (c_x, c_y) is converted to logical coordinates by

$$x = (c_x - o_x) \frac{W_p}{w_{\text{css}}}, \quad y = (c_y - o_y) \frac{W_p}{w_{\text{css}}}.$$

The same factor is used on both axes, so that logical units are square, and $H_p = h_{\text{css}} W_p / w_{\text{css}}$. The element's offset and size are read when the layout changes and not on every event. The device pixel ratio is used only to size the backing store of the canvas.

E.2 Normalized form

For a conditioned stroke P of positive length resampled to $Q = (q_1, \dots, q_N)$, the normalized stroke is $\hat{q}_k = (q_k - \bar{q}) / S$ with $\bar{q} = \frac{1}{N} \sum_k q_k$ and $S = \max(w, h)$, where w and h are the width and height of the bounding box of Q . The normalized stroke has centroid at the origin and the larger side of its bounding box of length one. Rotation, if enabled, is applied afterward, by the angle that minimizes the distance to the template (Chapter 12) or by the indicative angle of the stroke, the angle from the centroid to its first point, where the invariance configuration asks for a canonical orientation.

E.3 Keyboard coordinates

For word trajectories, logical coordinates inside the keyboard region are converted to key widths by a fixed affine map from the displayed keyboard rectangle. For a keyboard rectangle with origin (k_x, k_y) and width w_k in logical units, with $u = w_k / 10$ the key width for a ten-key row, the point (x, y) maps to $((x - k_x) / u, (y - k_y) / u)$. The location-channel distance of Chapter 39 is computed in these coordinates, without further normalization of translation or scale.

Appendix F

Difficulty Metrics

The complexity measure is defined in Definition 15.1. The table below collects the default constants, which are design decisions and are versioned with the library through the identifier of the weight set (the field `difficulty.version` of Listing B.1).

Quantity	Symbol	Default
Half-value constants	c_L, c_T, c_X, c_R	1, 1, 1, 2
Ambiguity scale	s_0	$\delta_{\min}$
Weights: length, turning, crossing, reversal, ambiguity	$\alpha, \beta, \gamma, \delta, \eta$	0.20, 0.25, 0.15, 0.15, 0.25
Reversal threshold	(none)	150°
Base score	B_0, β	100, 2
Speed bonus	λ	0.5
Economy bonus	κ	0.1
Streak cap	$k_{\max}$	10
Target success band	(none)	[0.6, 0.9]
Mastery	(none)	fifth percentile of $\pi_g \geq 0.75$

Appendix G

Input Adapter Contracts

An adapter is the only code that touches a device. Its obligations are those of Requirement 49.1, and are restated here as a checklist of what a conformance test verifies. The adapter must produce, for each stroke, exactly one begin event followed by zero or more sample events and exactly one end or cancel event, with non-decreasing timestamps. The first sample of a stroke is carried by the begin event and the last by the end event. Coordinates are logical. The adapter identifies itself and its version and declares the device kind and the nominal sample rate. When the device does not supply a natural begin and end, the adapter documents the control it synthesizes them from. When two strokes would overlap, as with multiple pointers, the adapter delivers the first and ignores the others until it ends. An adapter performs no filtering that cannot be reconstructed from the raw samples, and any such filtering is declared in its version string and is subject to Requirement 9.1.

A conformance test drives an adapter with a scripted device, a sequence of device events with known timing, and compares the emitted stroke events with the expected sequence. The scripted devices of the first milestone are a pointer script with and without coalesced events, and a keyboard script with held and released keys in both orders.

Appendix H

Test Fixtures

H.1 Format

A fixture is a JSON object with a name, the identifiers of the pipeline, library and recognizer, the raw samples, the candidate set, and the expected results with their tolerances.

Listing H.1: A regression fixture (the five-point test profile of the worked example below).

```
{
  "name": "corner.l/worked-example",
  "pipeline": "baseline/1+n5", "library": "test-shapes/1.0.0", "recognizer":
    "template/1",
  "samples": [[0,0,0],[0,100,400],[100,100,800]],
  "candidates": ["corner.l", "line.v"],
  "expect": { "distance": { "corner.l": 0.0, "line.v": 0.404625 },
    "tolerance": 1e-6, "fragile": false }
}
```

H.2 Edge-case fixtures

Every reference algorithm of Appendix C has fixtures in each of the classes below: degenerate strokes, empty and singleton candidate sets, ties, repeated points, and conflicting invariance settings. The values are produced by `tools/fixtures_check.py`, which implements the algorithms independently of the prose, and the hand-checkable ones are derived in the table.

H.3 A worked normalization

The following example can be checked by hand and is part of the unit tests. The stroke runs from (0,0) down to (0,100) and then right to (100,100), a right-angled L of two equal sides. Its path length is 200. Resampled to $N = 5$ points at arc lengths 0, 50, 100, 150, 200, the points are (0,0), (0,50), (0,100), (50,100) and (100,100). The bounding box is 100×100 , so $S = 100$ and the scaled points are (0,0), (0,0.5), (0,1), (0.5,1), (1,1). The centroid is

$(0.3, 0.7)$, and the normalized points are

$$(-0.3, -0.7), (-0.3, -0.2), (-0.3, 0.3), (0.2, 0.3), (0.7, 0.3).$$

Normalizing the same stroke drawn at twice the size, or at a different position, gives the same five points, which is the content of the scale and translation properties of Chapter 55. Against the vertical line, whose normalized five-point template is $(0, -0.5), (0, -0.25), (0, 0), (0, 0.25), (0, 0.5)$, the pointwise distances are 0.3606, 0.3041, 0.4243, 0.2062 and 0.7280, and their mean, the distance d of Chapter 17, is 0.4046, the value recorded in the fixture above.

Class	Setup	Expected
Empty candidates	$\mathcal{C} = \emptyset$	Rejected, reason <code>no_candidates</code>
Single candidate, near	$\mathcal{C} = \{A\}, d_A = 0.10 \leq d_{\max} = 0.30$	Accepted, $p^* = 1$; the gate alone decides
Single candidate, far	$\mathcal{C} = \{A\}, d_A = 0.40 > d_{\max}$	Rejected, <code>too_far</code>
Posterior tie	A, B with $d_A = d_B = 0.10$, equal priors	Ambiguous $\{A, B\}$; margin $0 < \mu$
Prior override	$d_A = 0.10, d_B = 0.12$, priors $0.1, 0.9, \sigma = 0.1, \lambda = 1, \Theta = 1$	$p_A = 0.1216, p_B = 0.8784$; $g^* = B \neq g_{\text{nn}} = A$; Accepted(B) with the override flag
No override	as above with equal priors	$p_A = 0.5548, p_B = 0.4452$; Ambiguous $\{A, B\}$
Zero-length stroke	all samples at one point	Rejected before scoring, <code>malformed</code>
Repeated points, non-increasing times	the worked example with each point duplicated	Same normalized points and distance as the worked example
DTW by hand	$a = ((0,0), (2,0)), b = ((0,0), (1,0), (2,0))$	$D(2,3) = 1$, distance $1/5 = 0.2$; path weights $2 + 1 + 2 = 5 = n + m$
DTW below pointwise	any equal-length pair with the diagonal in the band	$\text{dtw} \leq d$
Conflicting invariances	<code>line.h</code> allows rotation, <code>line.v</code> does not	$c_{\text{line.h}}(v, h) = 0$, $c_{\text{line.v}}(h, v) = 0.4243$ ($N = 5$); $\kappa = 0$; library rejected at load
Drift retraction	T^0 the vertical line, $\hat{G}$ shifted by $0.2, \eta = 0.5, \rho = 0.05$	Drift 0.1 before, exactly 0.05 after retraction
Floor violation	an update whose every step down to $1/64$ violates the floor	Template returned unchanged

Table H.1: Edge-case fixtures. Values to four places; tolerance 10^{-6} .

Appendix I

Browser Compatibility Requirements

The implementation depends on the browser features below. All are supported by current releases of the major engines at the time of writing, with the variations noted. The specification lists the features and the intended behavior when one is missing, and does not list version numbers, which change faster than this document.

Feature	Used for	If unavailable
Pointer Events with capture	Unified touch, mouse and pen input	Not supported: the game does not start and says so
Coalesced pointer events	Full sample rate from touch controllers	Fall back to delivered events; sample rate falls to display rate
<code>touch-action: none</code>	Prevent scroll and zoom while drawing	Required
Canvas 2D, <code>requestAnimationFrame</code>	Rendering and loop	Required
High-resolution clock	Timestamps and benchmarks	Fall back to millisecond clock with a warning
IndexedDB	Profiles and recorded strokes	Fall back to simple storage with a size limit
Persistent storage request	Resist eviction	Continue; prompt for export more often
Service workers and manifest	Offline play and installation	Play online only
Web Audio	Synthesized sound	Silent play, with visual signals
Gamepad interface	Gamepad adapter	Adapter unavailable
Reduced-motion media query	Respect user preference	Provide the setting manually
Vibration interface	Optional haptics	Omit haptics

Appendix J

Experimental Protocols

The protocols below are summaries from which full, versioned protocols are written before data collection, and each is registered, with its analysis plan, before the data are seen.

P1, comparison of recognizers. Strokes from at least thirty participants, each drawing every gesture of the library at least twenty times in practice mode with prompted targets, are collected on at least two device classes. The strokes are split by participant into folds. Every recognizer is tuned on training folds and evaluated on the held-out fold with the measures of Chapter 24. The primary outcome is the false-accept rate at a fixed ambiguity rate, and the secondary outcomes are accuracy, latency and calibration error. Intervals are cluster bootstrap intervals over participants.

P2, learning curves. Participants play for at least four weeks with a frozen recognizer in the training mode. A probe of twenty standardized presentations is given at the start and at fixed intervals. The outcomes are the parameters of the curve families of Chapter 66 fitted by hierarchical models, with the last-activity record used for the attrition analysis.

P3, human versus model adaptation. A crossover is unsuitable as the primary design, because motor learning cannot be reset between periods: skill acquired in the first period carries into the second, so period and carryover effects are confounded with the arm. The primary design is therefore parallel. Participants are randomized to a frozen arm or an adaptive arm for the whole study, stratified by baseline probe performance and device class, and the primary comparison is the difference in the change of success rate on the standardized probes. A crossover is permitted only as a secondary, exploratory design, with a washout period whose adequacy is checked by a return of the baseline probe, a model with explicit period and sequence terms, and a pre-specified rule for discarding the second period if carryover is detected. Changing familiarity with the gesture set is handled by a fixed probe set that is held out of play, and the replay decomposition of Chapter 67 is computed for each participant as a secondary analysis, with the closed-loop limitation reported.

P4, degraded information. Recorded strokes from P1 are degraded by the operations of Chapter 68 and re-scored by each recognizer, giving curves of accuracy against retained fraction and the critical fraction for each recognizer and operation.

P5, cross-device transfer. Participants practice on one device class and are tested on another, with a control group that practices on the test device, giving the transfer ratio of Chapter 32 and the cross-device gap of Chapter 63.

P6, historical comparison. A harness generates strokes from word paths with controlled perturbations and presents them to the historical software and to the new recognizers, giving acceptance curves (Chapter 70). The protocol is a software-behavior experiment and nothing more: it measures how one version of the software responds to controlled perturbations, and it says nothing about how experienced users actually stroked. Human performance on surviving installations is a separate study (P8), undertaken only where such installations can be lawfully run, with its own participants, consent and analysis plan.

P7, geometric curriculum versus direct word practice. The central educational hypothesis is that the geometric curriculum contributes something beyond practice time. Participants are randomized to one of three arms of equal prescribed practice time: the geometric curriculum of Parts V and VI, direct practice of word gestures on the keyboard with matched difficulty and the same game wrapper, and a no-gesture control that plays the game with a non-gesture input. The primary outcome is gesture-typing speed and accuracy on a fixed set of unseen words, tested on the keyboard without any guide, at the end and after a retention interval; the secondary outcomes are the three separate skills of reproducing a visible trajectory, recalling a learned trajectory without a guide, and generating the trajectory of a newly presented word. The geometric arm is said to help only if it outperforms the direct-practice arm at equal time; an advantage over the no-gesture control alone is not evidence for the curriculum.

P8, human performance on surviving installations. An optional study of experienced users on lawfully obtained historical software, separate from P6, measuring their acceptance curves and speed on natural, unperturbed word lists. It is not required by any milestone.

Appendix K

Record of Historical Claims

This appendix separates what is documented about gesture keyboards and Swype from what is reported and from what is hypothesized, as the Conventions require. It is a working record: entries marked as needing verification are to be checked against primary sources before they are used in any publication. The patents held by the developers of Swype and ShapeWriter, and the surviving implementations, have not been analyzed in this version and are listed as open tasks.

Bibliography audit status. In version 0.2 two entries were checked against an institutional record of the publication: Kristensson and Zhai (CHI 2007), for which the page range was added, and Zhai and Kristensson (CHI 2003), whose existing details were confirmed. The remaining entries, the historical dates in the table below and the patent details have not been audited against primary sources, and the audit is a precondition of any publication.

Claim	Status	Basis	To verify
Shorthand writing on a stylus keyboard; SHARK2 recognition with shape and location channels and a language model	[D]	Published papers [33, 16]	Confirm details of channel integration against the paper
The word-gesture keyboard as a paradigm supporting transition from tracing to recall	[D]	Published review [34]	Quote the exact formulation if cited
Swype created by Kushler and Marsden; wide Android availability around 2010	[D]	Public histories	Primary announcements and dates
Acquisition by Nuance in 2011; discontinuation in 2018	[D]	Public announcements	Exact dates and scope (platforms) of discontinuation
Recognition before a stroke is complete	[O]	Reports of long-term users	Measurement on surviving installations (P6)
Exaggerated corners and swoops improve recognition	[O]	Reports of long-term users	Measurement (P6) and mechanism (Chapter 31)

Appendix K. Record of Historical Claims

Claim	Status	Basis	To verify
Learned personal vocabulary improves recognition of added words	[O]	Reports; general feature of predictive keyboards	Measurement on surviving installations (P6)
Gesture shortcuts for editing operations	[O]	Reports of long-term users	Inventory of the shortcuts in the last releases
Patents and internal algorithms	None	Not analyzed	Patent search and review

Bibliography

- [1] J. Accot and S. Zhai. Beyond Fitts' law: models for trajectory-based HCI tasks. In *Proceedings of CHI '97*, pages 295–302. ACM, 1997.
- [2] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216, 1991.
- [3] R. A. Bjork. Memory and metamemory considerations in the training of human beings. In J. Metcalfe and A. Shimamura, editors, *Metacognition: Knowing about Knowing*, pages 185–205. MIT Press, 1994.
- [4] S. Bradner. Key words for use in RFCs to indicate requirement levels. RFC 2119, IETF, 1997.
- [5] G. Casiez, N. Roussel, and D. Vogel. The one-euro filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of CHI '12*, pages 2527–2530. ACM, 2012.
- [6] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
- [7] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.
- [8] P. M. Fitts and M. I. Posner. *Human Performance*. Brooks/Cole, 1967.
- [9] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: on the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.
- [10] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.
- [11] Y. Geifman and R. El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems 30*, 2017.
- [12] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of ICML*, PMLR 70:1321–1330, 2017.

- [13] N. Hogan and D. Sternad. Sensitivity of smoothness measures to movement duration, amplitude, and arrests. *Journal of Motor Behavior*, 41(6):529–534, 2009.
- [14] R. J. K. Jacob. What you look at is what you get: eye movement-based interaction techniques. In *Proceedings of CHI '90*, pages 11–18. ACM, 1990.
- [15] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.
- [16] P.-O. Kristensson and S. Zhai. SHARK²: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of UIST '04*, pages 43–52. ACM, 2004.
- [17] P.-O. Kristensson and S. Zhai. Command strokes with and without preview: using pen gestures on keyboard for command selection. In *Proceedings of CHI '07*, pages 1137–1146. ACM, 2007.
- [18] F. Lacquaniti, C. Terzuolo, and P. Viviani. The law relating the kinematic and figural aspects of drawing movements. *Acta Psychologica*, 54(1–3):115–130, 1983.
- [19] I. S. MacKenzie. KSPC (keystrokes per character) as a characteristic of text entry techniques. In *Mobile HCI 2002*, LNCS 2411, pages 195–210. Springer, 2002.
- [20] I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03 Extended Abstracts*, pages 754–755. ACM, 2003.
- [21] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.
- [22] A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson, editor, *Cognitive Skills and Their Acquisition*, pages 1–55. Erlbaum, 1981.
- [23] D. Rubine. Specifying gestures by example. In *Proceedings of SIGGRAPH '91*, pages 329–337. ACM, 1991.
- [24] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.
- [25] A. W. Salmoni, R. A. Schmidt, and C. B. Walter. Knowledge of results and motor learning: a review and critical reappraisal. *Psychological Bulletin*, 95(3):355–386, 1984.
- [26] R. A. Schmidt. A schema theory of discrete motor skill learning. *Psychological Review*, 82(4):225–260, 1975.
- [27] J. B. Shea and R. L. Morgan. Contextual interference effects on the acquisition, retention, and transfer of a motor skill. *Journal of Experimental Psychology: Human Learning and Memory*, 5(2):179–187, 1979.

- [28] R. W. Soukoreff and I. S. MacKenzie. Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric. In *Proceedings of CHI '03*, pages 113–120. ACM, 2003.
- [29] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. Gestures as point clouds: a \$P recognizer for user interface prototypes. In *Proceedings of ICMI '12*, pages 273–280. ACM, 2012.
- [30] M. Vidal, A. Bulling, and H. Gellersen. Pursuits: spontaneous interaction with displays based on smooth pursuit eye movement and moving targets. In *Proceedings of UbiComp '13*, pages 439–448. ACM, 2013.
- [31] World Wide Web Consortium. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation, 2023.
- [32] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In *Proceedings of UIST '07*, pages 159–168. ACM, 2007.
- [33] S. Zhai and P.-O. Kristensson. Shorthand writing on stylus keyboard. In *Proceedings of CHI '03*, pages 97–104. ACM, 2003.
- [34] S. Zhai and P.-O. Kristensson. The word-gesture keyboard: reimagining keyboard interaction. *Communications of the ACM*, 55(9):91–101, 2012.