MOMENTUM MIRROR

2-Year Development & Testing Plan

Physics-Based Ricochet Game with Measurable Skill Mastery

Project Start Date: February 10, 2026
Target Launch: Q1 2028
Version: 1.0
Status: Planning Phase

Table of Contents

1. Executive Summary

Momentum Mirror is a physics-based 2D ricochet game where players navigate levels using momentum-based movement. The game's unique value proposition lies in its measurable skill progression system, where motor mastery is quantified through efficiency, rhythm, and energy conservation metrics.

1.1 Project Overview

Unlike traditional puzzle games where solutions become trivial once discovered, Momentum Mirror creates an endless skill ceiling through physics-based mechanics that reward motor precision, timing, and rhythmic control. The game is designed with built-in observability for both human players and automated agents to measure and verify skill progression.

1.2 Key Innovation

The core innovation is the coupling of multiple skill metrics that prevent degenerate optimization strategies. Players cannot maximize efficiency without accepting risk, cannot maintain static rhythm in dynamic environments, and cannot spam inputs without losing efficiency. This creates genuine motor skill development rather than cognitive puzzle-solving.

1.3 Development Timeline

Phase 1 (Months 1-6): Core mechanics prototype and metric validation
Phase 2 (Months 7-12): Content creation and level design
Phase 3 (Months 13-18): Agent framework and testing infrastructure
Phase 4 (Months 19-24): Polish, balance, and launch preparation

1.4 Success Metrics

- Measurable skill progression across 90% of player base
- Agent can distinguish novice vs expert with 95% accuracy
- 50+ levels demonstrating mechanic scalability
- Average session length > 20 minutes (indicating engagement)
- Speedrunning community emergence (indicating skill ceiling)

2. Project Vision & Objectives

2.1 Vision Statement

Create a game where skill mastery is visceral, measurable, and infinitely improvable—where expert play looks and feels fundamentally different from novice play at the motor level, not just the strategic level.

2.2 Core Objectives

Primary Objective: Design a mechanic where motor skill improvement is necessary, observable, and quantifiable without human judgment.

Secondary Objective: Build an agent-testable framework that can verify skill progression and distinguish mastery from luck or brute force.

Tertiary Objective: Create a scalable content pipeline that produces levels demonstrating the full range of skill expression.

2.3 Design Principles

- Physically Felt Interaction: Every input must have immediate, satisfying feedback
- Low Floor, High Ceiling: Easy to understand, infinitely difficult to master
- Measurable Compression: Skill must leave numerical fingerprints in game state
- Metric Coupling: No single metric can be optimized independently
- Motor > Cognitive: Improvement comes from finger skill, not planning

2.4 Target Audience

Primary: Skill-focused gamers who enjoy mastery (Celeste, Super Meat Boy, Rocket League players)
Secondary: Speedrunning community
Tertiary: Casual players attracted by accessible controls and satisfying physics

3. Core Mechanic Design

3.1 Input System

Players control a circular character using swipe gestures (touch) or mouse drag (desktop). The swipe has three parameters:

- **Direction:** Angle of swipe from start to end point
- **Duration:** Time between touch-down and release (50ms - 1000ms)
- **Velocity:** Speed of swipe gesture, calculated from distance/duration

3.2 Physics Response

The character receives an impulse based on the swipe:

```
Impulse Strength = Duration × Velocity × Base Multiplier
```

- Character moves in straight line until collision
- Bounces preserve momentum with material-dependent coefficients
- Angular momentum affects trajectory on curved surfaces
- Momentum decays over time (friction)
- Some surfaces amplify momentum (springs), others dampen (cushions)

3.3 Surface Types

Different surfaces create strategic choices and prevent static solutions:

Surface Type	Restitution	Effect	Strategic Use
Standard Wall	0.85	Slight momentum loss	Predictable bounces
Spring Wall	1.3	Amplifies momentum	Chain high-speed sequences
Cushion Wall	0.4	Heavy dampening	Precise positioning
Curved Surface	0.9	Angular deflection	Complex path planning
Phase Wall	Variable	Changes over time	Rhythm adaptation

3.4 Metric Coupling Design

The critical innovation: metrics are coupled such that optimizing one without motor skill worsens another:

Efficiency vs Risk: Higher efficiency requires higher speeds and narrower margins. Slow, careful play is inefficient.

Rhythm vs Environment: Phase walls change timing windows. Static rhythm patterns become suboptimal as environments evolve.

Economy vs Precision: Fewer inputs require higher accuracy per swipe. Spam inputs reduces efficiency scores.

4. Technical Architecture

4.1 Technology Stack

Component	Technology	Justification
Game Engine	Phaser 3	Lightweight 2D, excellent physics, cross-platform
Physics	Matter.js	Precise collision detection, deterministic
Language	TypeScript	Type safety for complex metric calculations
Testing	Playwright	Reliable automation, cross-browser support
Metrics Store	SQLite	Local storage, fast queries, portable
Build Tool	Vite	Fast dev server, optimized production builds
Deployment	Static hosting	CDN distribution, low latency

4.2 System Architecture Diagram

[Diagram would be inserted here showing:]

- Input Layer (touch/mouse handlers)
- Physics Engine (impulse calculation, collision detection)
- Metrics Engine (real-time tracking of efficiency, rhythm, economy)
- Game State Manager (level progression, player stats)
- Rendering Layer (visual feedback, particle effects)
- Testing Harness (Playwright automation, metric verification)

4.3 Data Flow

The system follows this data flow for each player action:

1. Player performs swipe gesture

2. Input handler captures: start position, end position, timestamp

3. Calculate: duration, velocity, direction

4. Physics engine applies impulse to character

5. Collision detection tracks: bounce count, energy loss, distance traveled

6. Metrics engine updates: efficiency, rhythm entropy, input density

7. Render engine displays: trajectory, particle effects, metric feedback

8. State manager persists: action history, cumulative metrics, level progress

4.4 Determinism Requirements

For reliable agent testing, the physics simulation must be deterministic:

- Fixed timestep (60 FPS, 16.67ms per frame)
- Deterministic random seed for procedural elements
- IEEE 754 floating point consistency across platforms
- Replay system stores: input sequence, timestamps, random seed
- Verification: replayed inputs must produce identical outcomes

5. Development Roadmap

The development is structured in four 6-month phases, each with specific deliverables and success criteria.

5.1 Phase 1: Foundation (Months 1-6)

Timeline: Feb 2026 - Jul 2026

Objectives:

- Implement core swipe mechanics with variable duration/velocity
- Build deterministic physics engine
- Create metric tracking system
- Develop first 5 prototype levels
- Validate that metrics distinguish skill levels

Key Milestones:

Month	Milestone	Deliverable
1	Basic input system	Swipe captures duration + velocity
2	Physics prototype	Bouncing mechanics with different surfaces
3	Metrics MVP	Efficiency, rhythm, economy tracking
4	5 test levels	Cover full range of mechanics
5	Agent integration	Playwright can execute swipes
6	Validation study	Metrics distinguish novice vs expert

Success Criteria:

- Metrics compress with practice (expert = 2x novice efficiency)
- Agent can replay inputs with <1% variance in outcomes
- Playtesters report satisfying physics feel
- All 5 prototype levels completable, showing metric variance

5.2 Phase 2: Content & Balance (Months 7-12)

Timeline: Aug 2026 - Jan 2027

Objectives:

- Expand to 30 levels across difficulty tiers
- Implement level editor for rapid iteration

- Refine surface types and physics parameters
- Add visual polish and particle effects
- Create tutorial progression

Key Milestones:

Month	Milestone	Deliverable
7	Level editor	Internal tool for rapid level creation
8	10 beginner levels	Tutorial through basic mechanics
9	10 intermediate levels	Surface type combinations
10	10 advanced levels	Rhythm and precision challenges
11	Visual polish pass	Particle effects, transitions, feedback
12	Balance tuning	Metric targets calibrated across difficulty

Success Criteria:

- Smooth difficulty curve from beginner to advanced
- Each level demonstrates unique skill expression
- Level editor enables 1 level/hour creation rate
- Visual feedback clearly communicates physics state

5.3 Phase 3: Agent Testing Framework (Months 13-18)

Timeline: Feb 2027 - Jul 2027

Objectives:

- Build comprehensive agent testing suite
- Implement learning detection algorithms
- Create metric analysis dashboard
- Develop regression test suite
- Document testing methodology

Key Milestones:

Month	Milestone	Deliverable
13	Test automation	Playwright suite for all levels
14	Learning detection	Algorithms to identify skill progression
15	Metric dashboard	Real-time visualization of

		skill metrics
16	Regression tests	Automated verification of physics consistency
17	Variance analysis	Statistical tools for expert vs novice
18	Documentation	Complete testing methodology guide

Success Criteria:

- Agent can complete 100% of levels (deterministic verification)
- Learning detection accuracy >95% (novice vs expert classification)
- Full regression suite runs in <10 minutes
- Dashboard enables rapid identification of balance issues

5.4 Phase 4: Polish & Launch (Months 19-24)

Timeline: Aug 2027 - Jan 2028

Objectives:

- Expand to 50+ total levels
- Add progression systems (unlocks, achievements)
- Implement leaderboards and replay sharing
- Closed beta testing
- Marketing and community building
- Launch preparation

Key Milestones:

Month	Milestone	Deliverable
19	50 level library	Complete content for launch
20	Progression systems	Unlocks, stats, achievements
21	Online features	Leaderboards, replay sharing
22	Closed beta	100 player testing period
23	Final polish	Bug fixes, balance, optimization
24	Launch	Public release

Success Criteria:

- Beta retention rate >60% after 1 week
- Average session length >20 minutes
- Performance: 60 FPS on mid-range devices

- Community emergence: speedrun strategies shared online

6. Testing Strategy

Testing occurs at three levels: unit tests for components, integration tests for systems, and agent-based validation for skill metrics.

6.1 Unit Testing

Component-level verification:

- **Physics calculations:** Impulse formulas, collision detection, momentum transfer
- **Metric computations:** Efficiency, rhythm entropy, energy conservation
- **Input parsing:** Duration, velocity, direction from swipe data
- **Surface interactions:** Restitution coefficients, angular deflection

6.2 Integration Testing

System-level verification:

- End-to-end level completion
- Metric aggregation across multiple actions
- State persistence and loading
- Replay determinism
- Leaderboard integration

6.3 Agent-Based Validation

The core innovation: automated verification that the game mechanics actually measure skill improvement.

6.3.1 Baseline Establishment

Agent performs random exploration to establish baseline metrics:

```
// Baseline test: random swipes
for (i = 0; i < 100; i++) {
  await randomSwipe(level);
  recordMetrics();
}
baselineEfficiency = mean(metrics.efficiency);
baselineEntropy = mean(metrics.rhythmEntropy);
```

6.3.2 Learning Detection

Agent performs repeated attempts with variance injection to simulate human learning:

```
// Learning test: progressive refinement
for (trial = 0; trial < 50; trial++) {
  await executeStrategy(level, variance=0.1 - trial*0.002);
  recordMetrics();
}
// Verify compression
assert(metrics[49].efficiency > metrics[0].efficiency * 1.5);
assert(metrics[49].entropy < metrics[0].entropy * 0.5);
```

6.3.3 Expert Verification

Agent executes optimized strategies to verify expert metrics:

- Efficiency >0.85 (vs baseline ~0.4)
- Rhythm entropy <0.25 (vs baseline ~0.9)
- Input density <1.5x optimal (vs baseline ~6x)
- Energy conservation >75% (vs baseline ~40%)

6.4 Human Playtesting

Complement agent testing with human validation:

Phase	Scale	Focus
Alpha (Month 6)	5-10 testers	Core mechanic feel, metric correlation
Beta (Month 12)	50 testers	Level difficulty, progression pacing
Closed Beta (Month 22)	100+ testers	Balance, retention, community

7. Metric Implementation

The four core metrics that define skill mastery, with detailed implementation specifications.

7.1 Momentum Efficiency

Definition:

Ratio of progress toward goal divided by total impulse applied. Measures how economically the player uses their momentum.

Formula:

```
Efficiency = Σ(distance_toward_goal) / Σ(impulse_magnitude)
```

Implementation Details:

- Track character position every frame
- Calculate dot product of movement vector with goal-direction vector
- Accumulate positive dot products (backward movement = 0 contribution)
- Divide by cumulative impulse strength
- Update in real-time, display post-completion

Typical Values:

Novice: 0.35 - 0.50
Intermediate: 0.55 - 0.75
Expert: 0.80 - 0.95
World-class: >0.95

7.2 Energy Conservation

Definition:

Percentage of momentum preserved through collisions relative to path complexity. Experts hit walls at optimal angles, minimizing energy loss.

Formula:

```
Conservation = (final_momentum / initial_momentum) ×
               path_complexity_factor
```

Implementation Details:

- Record momentum magnitude before and after each collision
- Calculate loss ratio: (momentum_after - momentum_before) / momentum_before
- Normalize by path complexity (number of required bounces)
- Weight by collision angle (perpendicular = high loss penalty)

Typical Values:

Novice: 30 - 45% (heavy losses from poor angles)
Intermediate: 50 - 70%
Expert: 75 - 90%
World-class: >90%

7.3 Rhythmic Consistency

Definition:

Entropy of inter-swipe timing intervals. Expert players develop consistent rhythm; novices have erratic, high-variance timing.

Formula:

```
Rhythm_Entropy = σ(inter_swipe_times) / μ(inter_swipe_times)
```

Implementation Details:

- Record timestamp of each swipe
- Calculate differences: Δt[i] = timestamp[i] - timestamp[i-1]
- Compute standard deviation σ and mean μ of Δt array
- Lower values indicate more consistent rhythm

Typical Values:

Novice: 0.75 - 1.20 (highly variable timing)
Intermediate: 0.40 - 0.70

Expert: 0.15 - 0.35
World-class: <0.20

7.4 Input Economy

Definition:

Ratio of actual swipes used to theoretical optimal swipe count. Prevents input spam and rewards precision.

Formula:

```
Input_Density = actual_swipes / optimal_swipes
```

Implementation Details:

- Each level has a pre-calculated optimal swipe count (speedrun TAS)
- Count player swipes
- Divide actual by optimal
- Values near 1.0 indicate mastery; high values indicate spam

Typical Values:

Novice: 4.0 - 8.0 (lots of corrections)
Intermediate: 2.0 - 3.5
Expert: 1.2 - 1.8
World-class: <1.3

8. Content Pipeline

8.1 Level Design Philosophy

Each level must demonstrate specific skill expression opportunities. Levels are not puzzles with single solutions, but arenas for performance.

8.2 Level Categories

Category	Count	Focus	Example Challenge
Tutorial	5	Introduce mechanics	Learn swipe duration control
Efficiency Focused	12	Optimal pathing	Reach goal with minimal impulse
Rhythm Focused	12	Timing consistency	Phase walls require rhythm adaptation
Precision Focused	12	Narrow margins	Thread needle gaps at high speed
Hybrid	9	Multiple skills	Combine efficiency + rhythm

8.3 Level Creation Workflow

Design: Sketch layout, identify skill focus

Prototype: Build in level editor, test core path

Calibrate: Run agent tests to determine optimal metrics

Validate: Human playtest to verify fun factor

Balance: Adjust surface types, goal position until metrics align

Polish: Add visual elements, particle effects

8.4 Quality Assurance Checklist

Every level must pass:

- Agent can complete level (determinism verification)
- Metric spread: expert efficiency >2x novice
- Multiple valid paths exist (not single-solution puzzle)
- Completion time: 15 seconds (expert) to 120 seconds (novice)
- Visual clarity: goal visible, path readable
- Performance: maintains 60 FPS

8.5 Difficulty Progression

Levels are ordered by required skill level, measured objectively:

Difficulty Score = (1 - expert_efficiency) × input_density × (1 / rhythm_tolerance)

Levels are sorted by this score, ensuring smooth progression curve.

9. Agent Testing Framework

9.1 Framework Architecture

The agent testing system verifies game mechanics through automated gameplay and metric analysis.

- **Test Controller:** Orchestrates test execution, manages test suites
- **Input Simulator:** Generates swipes with specified parameters
- **Metric Collector:** Captures and stores performance metrics
- **Analysis Engine:** Statistical analysis of metric distributions
- **Assertion System:** Validates expected vs actual outcomes
- **Reporting:** Generates test reports and visualizations

9.2 Test Types

9.2.1 Determinism Tests

Verify physics consistency:

```
test('Physics determinism', async () => {
  const seed = 12345;
  const inputs = generateInputSequence();

  const run1 = await executeLevel(level, inputs, seed);
  const run2 = await executeLevel(level, inputs, seed);

  expect(run1.finalPosition).toEqual(run2.finalPosition);
  expect(run1.metrics).toEqual(run2.metrics);
});
```

9.2.2 Metric Validation Tests

Verify metrics distinguish skill levels:

```
test('Efficiency distinguishes skill', async () => {
  const noviceRuns = await runNoviceStrategy(level, 20);
  const expertRuns = await runExpertStrategy(level, 20);

  const noviceAvg = mean(noviceRuns.map(r => r.efficiency));
  const expertAvg = mean(expertRuns.map(r => r.efficiency));

  expect(expertAvg).toBeGreaterThan(noviceAvg * 1.8);
  expect(expertAvg).toBeGreaterThan(0.80);
});
```

9.2.3 Learning Detection Tests

Verify metrics compress with practice:

```
test('Learning detected', async () => {
  const runs = await progressivePractice(level, 50);

  const earlyAvg = mean(runs.slice(0, 10).map(r =>
r.efficiency));
  const lateAvg = mean(runs.slice(40, 50).map(r =>
r.efficiency));

  expect(lateAvg).toBeGreaterThan(earlyAvg * 1.5);

  const trend = linearRegression(runs.map(r => r.efficiency));
  expect(trend.slope).toBeGreaterThan(0.01);
});
```

9.3 Input Strategies

The agent uses different strategies to simulate skill levels:

Strategy	Characteristics	Simulates
Random	Random angles, durations, timing	Complete novice
Noisy Optimal	Near-optimal with 20% variance	Learning player
Refined	Near-optimal with 5% variance	Intermediate
TAS (Tool-Assisted)	Frame-perfect, optimal	Theoretical maximum

9.4 Continuous Integration

Agent tests run automatically on every code commit:

- Unit tests: Component verification (<1 minute)
- Physics regression: Determinism verification (2 minutes)
- Metric validation: All levels, all strategies (10 minutes)
- Learning detection: 50-trial progression tests (15 minutes)
- Full suite: Nightly build (30 minutes)

10. Risk Management

10.1 Technical Risks

Risk	Severity	Mitigation
Determinism failure across platforms	High	Implement fixed-timestep, IEEE 754 compliance tests, cross-platform verification
Physics tuning taking excessive time	Medium	Parameterize all physics constants, build tuning dashboard
Metric coupling breaking with edge cases	Medium	Extensive agent testing, statistical validation
Performance issues on low-end devices	Low	Profile early, optimize rendering, web workers for metrics

10.2 Design Risks

Risk	Severity	Mitigation
Mechanics not actually fun despite metrics	High	Early playtesting (Month 6), pivot if satisfaction scores low
Difficulty curve too steep	Medium	Agent-measured difficulty scores, automated balancing
Content creation too slow	Medium	Invest in level editor tooling, procedural generation research
Skill ceiling too low (solved quickly)	Low	Phase walls, dynamic elements prevent static solutions

10.3 Market Risks

Risk	Severity	Mitigation
Competition releases similar mechanic	Medium	Our USP is measurable mastery, unique even if mechanics copied
Audience too niche	Low	Accessible controls broaden appeal beyond hardcore
Lack of virality	Medium	Replay sharing, leaderboards, speedrun community focus

10.4 Contingency Plans

If key risks materialize:

Month 6 playtest fails: Pivot to simplified mechanic, extend Phase 1 by 2 months

Agent testing reveals metrics don't distinguish skill: Emergency sprint to redesign metrics, delay Phase 2

Performance unacceptable: Switch to native engine (Unity/Godot), 3-month delay

Content creation too slow: Reduce launch target to 30 levels, expand post-launch

11. Resource Requirements

11.1 Team Structure

Role	FTE	Responsibilities
Lead Developer	1.0	Architecture, physics, metrics engine
Level Designer	0.5	Content creation, balance (ramp up Phase 2)
QA/Agent Engineer	0.5	Testing framework, automation (ramp up Phase 3)
Visual Designer	0.25	UI/UX, particles, polish (Phase 4)
Sound Designer	0.1	Audio effects, feedback (Phase 4)

11.2 Technology Costs

- Development tools: Free (VSCode, Phaser, TypeScript)
- Hosting: ~$20/month (static CDN)
- Testing infrastructure: ~$50/month (CI/CD, test runners)
- Analytics: Free tier (Plausible or similar)
- Domain: ~$15/year

Total technology costs: ~$840/year (~$1,680 over 2 years)

11.3 Time Allocation

Estimated hours per phase:

Phase	Development	Testing	Design	Total
Phase 1	600	200	200	1000
Phase 2	400	300	500	1200
Phase 3	300	700	100	1100
Phase 4	500	400	300	1200
Total	**1800**	**1600**	**1100**	**4500**

At 40 hours/week: approximately 2.15 years of full-time work

12. Launch Strategy

12.1 Pre-Launch Activities

Month 18-20	**Build landing page, collect emails**
Month 21	Announce closed beta, select testers
Month 22	Closed beta, gather feedback
Month 23	Dev blog posts about metrics system
Month 24	Trailer, press outreach

12.2 Launch Platforms

Platform	Distribution	Rationale
Web (Primary)	Itch.io, own domain	Broadest reach, instant play
Steam	If web successful	Monetization, wishlist building
Mobile	Post-launch expansion	Touch-native controls

12.3 Marketing Messaging

Core positioning:

"The first game where your improvement is measured, not guessed. Watch your skills grow through hard data, not just high scores."

Key differentiators:

- Unique mechanic: Momentum-based ricochet with motor skill mastery
- Measurable progression: See your efficiency, rhythm, economy improve
- Infinite skill ceiling: Speedrunning potential, community competitions
- Accessible entry: Simple controls, satisfying physics from minute one

12.4 Community Building

- Discord server: Launch Month 20, build pre-release community
- Dev blog: Technical posts about metric system, attract skill-game fans
- Reddit: r/gamedev, r/speedrun, r/IndieDev
- YouTube: Dev logs, mechanic breakdowns, expert gameplay
- Speedrun integration: Leaderboards, ghost racing, world record tracking

13. Success Criteria

13.1 Technical Success Metrics

Metric	Target	Importance
Agent Testing	95% accuracy distinguishing novice vs expert	Core validation of metric system
Physics Determinism	100% replay consistency	Enables reliable testing
Performance	60 FPS on mid-range devices	Smooth gameplay experience
Test Coverage	>90% code coverage	Stability and maintainability

13.2 Player Experience Metrics

Metric	Target	Importance
Skill Progression	90% of players show metric improvement	Core mechanic validation
Retention	60% return after 1 week	Long-term engagement
Session Length	>20 minutes average	Deep engagement
Completion Rate	>70% complete 10+ levels	Difficulty curve balance

13.3 Launch Success Metrics

Metric	Target	Importance
Week 1 players	1,000+	Initial traction
Month 1 players	5,000+	Sustained interest
Community engagement	Discord 100+ members	Community building
Speedrun emergence	3+ documented strategies	Skill ceiling validation
Press coverage	5+ indie game sites	Visibility

13.4 Post-Launch Goals

- Month 3: 10,000 total players
- Month 6: Speedrun community with documented world records
- Month 12: 50,000 total players, considering mobile port
- Long-term: Establish as reference for skill-based game design

Appendix A: Technical Specifications

A.1 Physics Constants

Constant	Value	Purpose
Base Impulse Multiplier	1.5	Scales swipe to momentum
Gravity	0 m/s²	Top-down perspective
Friction Coefficient	0.02 per second	Gradual momentum decay
Standard Wall Restitution	0.85	Slight bounce loss
Spring Wall Restitution	1.3	Momentum amplification
Cushion Wall Restitution	0.4	Heavy dampening
Curved Surface Angular Factor	0.15	Trajectory deflection
Phase Wall Period	3.0 seconds	Timing window cycle

A.2 Input Parameters

Constant	Value	Purpose
Swipe Duration Min	50 ms	Quick flick
Swipe Duration Max	1000 ms	Extended drag
Velocity Calculation	distance / duration	Pixels per millisecond
Impulse Formula	duration × velocity × base_multiplier	Final momentum
Input Buffer	100 ms	Prevent accidental double-swipes

A.3 Rendering Specifications

- Resolution: 1920×1080 base, responsive scaling
- Frame Rate: 60 FPS locked
- Character: 32px circle, gradient fill
- Trajectory Preview: Dotted line, 0.3s lookahead
- Particle Effects: Bounce sparks, trail effects
- UI: Minimal HUD, metric display post-completion

Appendix B: Metric Formulas

B.1 Complete Metric Definitions

Momentum Efficiency:

```
efficiency = Σ(max(0, movement_vector · goal_direction)) /
Σ(impulse_magnitude)

where:
  movement_vector = position[t] - position[t-1]
  goal_direction = normalize(goal_position - current_position)
  impulse_magnitude = sqrt(impulse_x² + impulse_y²)

Range: [0, 1.0], higher is better
Typical expert: >0.85
```

Energy Conservation:

```
conservation = (1 - Σ(momentum_loss) / initial_momentum) ×
complexity_factor

where:
  momentum_loss[i] = max(0, momentum[i-1] - momentum[i])
  complexity_factor = optimal_bounces / actual_bounces

Range: [0, 1.0], higher is better
Typical expert: >0.75
```

Rhythmic Consistency:

```
rhythm_entropy = σ(inter_swipe_times) / μ(inter_swipe_times)

where:
  inter_swipe_times[i] = timestamp[i] - timestamp[i-1]
  σ = standard deviation
  μ = mean

Range: [0, ∞), lower is better
Typical expert: <0.30
```

Input Economy:

```
input_density = actual_swipe_count / optimal_swipe_count

where:
  optimal_swipe_count = TAS (tool-assisted speedrun) benchmark
  actual_swipe_count = player inputs used

Range: [1.0, ∞), lower is better
Typical expert: <1.5
```

B.2 Composite Skill Score

Overall skill rating combining all metrics:

```
skill_score = (efficiency × 0.4) +
              ((1 - rhythm_entropy) × 0.3) +
              ((conservation) × 0.2) +
              ((1 / input_density) × 0.1)

Normalized to [0, 100]

Rating Scale:
  0-20:  Novice
  21-40: Beginner
  41-60: Intermediate
  61-80: Advanced
  81-95: Expert
  96-100: Master
```

Appendix C: Level Design Guidelines

C.1 Design Principles

- Every level must be completable by novice players (even if inefficiently)
- Multiple paths to goal should exist (no single-solution puzzles)
- Skill expression opportunities must be visible and testable
- Visual clarity: player can see goal and critical surfaces
- Challenge emerges from execution, not obscurity

C.2 Tutorial Progression

Level	Teaches	Description
Level 1	Basic swipe	Single straight bounce to goal
Level 2	Duration control	Requires specific impulse strength
Level 3	Multi-bounce	3 bounces to reach goal
Level 4	Surface types	Introduce spring and cushion walls
Level 5	Precision	Narrow passage requiring accuracy

C.3 Difficulty Scaling

Levels increase difficulty through:

- Increased required bounces (3 → 8 → 15)
- Narrower passages (larger margin → tighter tolerances)
- Surface variety (all standard → mixed types)
- Phase walls (static → dynamic timing windows)
- Distance to goal (short paths → longer traversals)
- Momentum preservation requirements (casual → precise angles needed)

C.4 Testing Checklist

Before finalizing a level, verify:

☐ Agent can complete (determinism)

☐ Multiple solution paths exist

☐ Novice completion time: 30-120 seconds

☐ Expert completion time: 10-30 seconds

☐ Metric spread: expert efficiency >1.8x novice

☐ Visual clarity: goal visible from start

☐ Performance: 60 FPS maintained

☐ Difficulty score calculated and appropriate

☐ Playtested by 3+ people

☐ Fun factor rating >7/10

Document History

Version 1.0 - February 10, 2026

Initial comprehensive execution plan

END OF DOCUMENT