Game Design Document: ASPECT

Version: 1.6
Genre: High-Performance Vertical Scrolling Rhythm Game (VSRG) / Typing Hybrid Platform: PC (Keyboard) Resolution: 1280 x 720 (Standard HD)

---

1. Executive Summary

ASPECT is a minimalist, high-skill rhythm game that challenges both rhythmic precision and typing speed. Unlike traditional rhythm games where keys are bound to static lanes, ASPECT utilizes Dynamic Key Mapping, where every falling note is a randomized character from the entire keyboard. The game features an Auto-Map Generation engine that analyzes audio files to create persistent, beat-synced maps without manual intervention.

---

2. Gameplay Mechanics

2.1 The Dynamic Lane System

The play area consists of 8 vertical lanes used for visual organization.

- Lane Width: 160px per lane.
- Input Logic: Notes are not column-locked. When a note spawns, it is assigned a random lowercase letter (a-z).
- Unique Key Constraint: The spawner tracks all letters currently on screen. It will never spawn a duplicate letter until the existing one is cleared, ensuring no input ambiguity.

2.2 Hit Objects

- Beats (Circles): Standard circles with a bold letter in the center, synced to the song's onsets.
- Hold Notes (Sliders): Generated during sustained frequencies or long vocal notes.
- Chords: Multiple circles appearing on the same horizontal line, typically synced to heavy percussion or downbeats.

---

3. The Flashpoint (Dynamic Typing Endurance)

Flashpoints are high-intensity typing sections that occur at musically appropriate intervals.

- Variable Frequency: The number of Flashpoints per map is not fixed. Short tracks may have one, while marathon tracks may have four or more.

- Dynamic Placement: Instead of fixed percentages, Flashpoints are triggered by Audio Intensity Dips or Structural Breaks identified by the generation engine.
- Transition: Rhythm notes stop. Lanes slide to the screen edges. A central text field appears.
- Survival: Maintain a Words Per Minute (WPM) average above the Star Requirement. If the timer expires and your WPM is below the goal, it results in an Insta-Death.

---

4. Algorithmic Beat Mapping (ABM) Technical Logic

This section outlines how the engine programmatically generates a map from a raw audio file.

4.1 Required Libraries & Tools

- Librosa: Core functions for onset detection and tempo estimation.
- NumPy: Used for fast mathematical processing of spectral data.
- SciPy: Utilized for signal filtering (e.g., isolating bass frequencies).

4.2 The Generation Pipeline

1. Tempo Estimation: Finds the global BPM and initial beat offset.
2. Onset Detection: Identifies sudden changes in energy via Spectral Flux.
3. Flashpoint Identification: The engine looks for "Energy Valleys" (sections with low RMSE but high rhythmic consistency) to insert Flashpoint markers.
4. Quantization: All detected onsets are shifted to the nearest 1/4 or 1/8 note based on the calculated BPM.

---

5. Map Formatting Template (JSON)

The map file is a persistent record of the ABM analysis. Once generated, it is saved so the map is identical every time it is played.

JSON

```
{
  "metadata": {
    "title": "Song Name",
    "artist": "Artist Name",
    "generated_by": "ASPECT_Engine_v1.6",
    "difficulty_rating": 8.2,
    "drain_rate": 7.5
  },
  "audio_config": {
    "file": "audio.mp3",
    "bpm": 180,
    "offset_ms": 240
```

```
 },
 "flashpoints": [
   {
     "start_ms": 45000,
     "end_ms": 60000,
     "target_wpm": 120
   }
 ],
 "hit_objects": [
   { "time_ms": 1000, "lane": 3, "type": "beat" },
   { "time_ms": 1500, "lane": 5, "type": "beat" },
   { "time_ms": 2000, "lane": [1, 8], "type": "chord" },
   { "time_ms": 3000, "lane": 2, "type": "hold", "end_ms": 4500 }
 ]
}
```

---

6. Engine Logic & Edge-Case Handling (The "Brain")

This section defines how the game handles input conflicts and ensures high-performance fairness.

6.1 Active Key Registry (Unique Key Constraint)

To prevent input ambiguity, the engine uses a Global Key Registry.

- Logic: When a note is queued for spawning, the engine checks a HashSet of currently active letters. If Letter_X is present, it re-rolls to another letter.
- The "Wait-List": If a note is cleared (Hit or Miss), its letter remains in a 100ms cooldown buffer before it can be used again to prevent accidental double-tapping.

6.2 Simultaneous Input (Ghosting & Chords)

- Conflict Resolution: If a player presses two keys at the exact same millisecond, the engine resolves the lowest note on the screen first.
- Chord Mapping: In a Chord, the engine selects letters from different keyboard quadrants (e.g., Left Hand: 'W', Right Hand: 'P') to prevent finger crowding.

6.3 Despawn & Miss Logic

- Out-of-Bounds: Notes are removed 200px below the Strike Line.
- Miss Trigger: A "Miss" is only registered if the note crosses the Strike Line + the Grace Window OR if the player presses the wrong key while a note is in the hit-zone.

6.4 Input Debouncing

- Logic: If a user "mashes" a key that isn't currently assigned to a falling note, the engine ignores the input entirely rather than penalizing the HP.

---

7. Mathematical Systems & Difficulty Algorithms

7.1 ASPECT-Rating (Star Level Calculation)

The difficulty is calculated by aggregating several Intensity Factors.
Difficulty Formula (Pseudocode):
Python

```
def calculate_star_rating(map_data):
    nps_weight = (total_notes / song_length) * 1.2
    bpm_multiplier = map_data.bpm / 120
    chord_density = (total_chords / total_notes) * 2.5
    flashpoint_impact = (len(map_data.flashpoints) * 0.5) + (target_wpm / 100)

    # Final Star Rating (1.0 - 10.0 scale)
    star_rating = (nps_weight * bpm_multiplier) + chord_density + flashpoint_impact
    return round(star_rating, 2)
```

7.2 Scoring & Combo Logic

Score is calculated using an exponential combo multiplier to reward perfect streaks.
Formula:
$$Score = \text{Value} \times (\text{Combo} \times \text{DifficultyMultiplier})$$

- Perfect: 300 pts
- Great: 100 pts
- Miss: 0 pts (Breaks Combo)

7.3 The HP System (Passive Drain)

Health is managed via a value between 0 and 100.
HP Logic (Pseudocode):
Python

```
def update_hp(delta_time, rating, star_level):
    # Constant Passive Drain
    drain_amount = (star_level * 0.5) * delta_time
    hp -= drain_amount

    if rating == "PERFECT":
        hp += (10 - star_level) * 0.6
```

```
elif rating == "MISS":
    hp -= (star_level * 2.0)

hp = clamp(hp, 0, 100)
```

---

8. User Interface (UI) Specifications

Element	Position	Visual Style
Health Bar	Top Left	Horizontal bar (#38BDF8).
Combo Count	Bottom Left	[Number]x. Size: 64pt.
Accuracy	Top Right	Percentage to two decimals.

---

9. Implementation Notes for Coding AI

- Audio Synchronization: The game clock must be tied to the audio_stream_position.
- Note Pooling: Reuse note objects to prevent memory fragmentation.
- Coyote Time: Implement a 50ms grace window for late keypresses.
- JSON Persistence: If a song is loaded without a .json file, the game triggers the ABM pre-processor once and saves the output.