Hidden Markov Model Framework for Dynamic Difficulty Adjustment in MarioGPT

A Three-State Adaptive Level Generation System

1 Why HMM? The Core Problem

THere is a fundamental flaw in static difficulty: **the agent's true skill is invisible**. You *don't* see is the underlying skill state that produced those numbers.

This is exactly why Hidden Markov Models matter here. An HMM treats player skill as a **latent variable**—something real but not directly observable. Instead of reacting to raw metrics ("reward dropped, make it easier"), the HMM infers the *probability distribution over skill states* given a sequence of observations. This distinction is critical:

> Threshold-Based vs. HMM Approach
>
> **Threshold approach**: If completion_rate < 0.5, decrease difficulty.
> *Problem*: A single bad run triggers difficulty change. No memory, no context.
> **HMM approach**: Given the last 10 episodes of metrics, what is P(skill=High)?
> *Advantage*: Considers temporal patterns. A skilled player having one bad run stays in High state because the *sequence* still indicates high skill.

The HMM also provides **smooth probabilistic transitions**. Rather than binary switches (Easy/Hard), you get gradual shifts: "70% likely in Transition state, 25% in Low, 5% in High." This prevents the oscillation problem where difficulty ping-pongs between extremes.

For MarioGPT specifically, HMMs are ideal because:

1. MarioGPT's text prompts map naturally to discrete states ("few enemies" vs "many enemies")
2. Level generation is episodic—perfect for HMM's sequential observation model
3. The agent's learning process is inherently non-stationary; HMM parameters can adapt via Baum-Welch

2 The Three States: Definition and Purpose

The system operates on three hidden states, each corresponding to a difficulty tier and a MarioGPT prompt configuration:

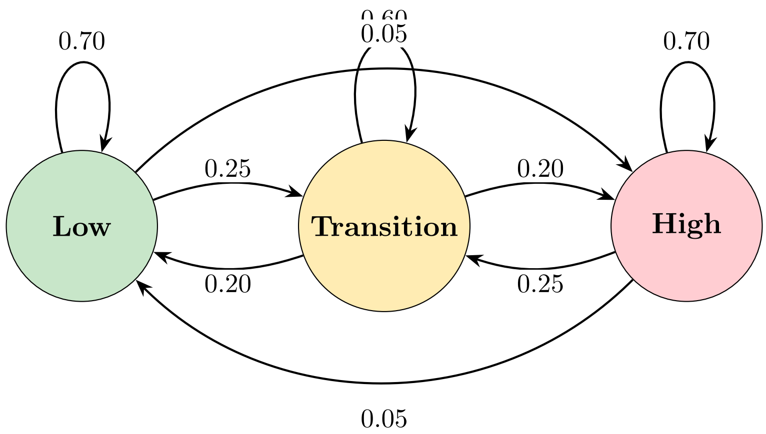
State	Level Characteristics	MarioGPT Prompt
Low (S_0)	Few enemies, no gaps, many pipes/powerups, flat terrain	`"few enemies, no gaps, many pipes, low elevation"`
Transition (S_1)	Moderate enemies, occasional gaps, some powerups, varied terrain	`"some enemies, few gaps, some pipes, medium elevation"`
High (S_2)	Many enemies, frequent gaps, few powerups, complex platforming	`"many enemies, many gaps, few pipes, high elevation"`

Why three states? Two states (Easy/Hard) create a binary system prone to oscillation. Four or more states add complexity without proportional benefit—MarioGPT's prompt vocabulary doesn't support fine-grained difficulty distinctions. Three states capture the essential dynamics: *building confidence* (Low), *assessment* (Transition), and *mastery challenge* (High).

3 State Flow: How Transitions Work

This is the core of the system. Understanding the flow between states is essential for both implementation and tuning.

3.1 The State Transition Diagram



3.2 Transition Matrix

The transition matrix $\mathbf{A}$ encodes these probabilities:

$$\mathbf{A} = \begin{bmatrix} P(S_0 \to S_0) & P(S_0 \to S_1) & P(S_0 \to S_2) \\ P(S_1 \to S_0) & P(S_1 \to S_1) & P(S_1 \to S_2) \\ P(S_2 \to S_0) & P(S_2 \to S_1) & P(S_2 \to S_2) \end{bmatrix} = \begin{bmatrix} 0.70 & 0.25 & 0.05 \\ 0.20 & 0.60 & 0.20 \\ 0.05 & 0.25 & 0.70 \end{bmatrix} \quad (1)$$

3.3 Flow Mechanics Explained

3.3.1 Low → Transition (Probability: 0.25)

When it happens: Agent demonstrates consistent competence in Low difficulty. Metrics show high completion rate (>80%), low death rate (<1.0), and positive reward trend.

What triggers it: The HMM observes that emission probabilities favor the Transition state—the agent's performance is "too good" for Low. After several such observations, the posterior probability shifts toward Transition.

MarioGPT response: Prompt changes from `"few enemies, no gaps..."` to `"some enemies, few gaps..."`. The agent now faces moderate challenge.

3.3.2 Transition → High (Probability: 0.20)

When it happens: Agent handles Transition difficulty with ease. Completion rate remains high, deaths are minimal, and the agent completes levels faster than average.

What triggers it: Sustained high performance in Transition. The HMM's forward algorithm accumulates evidence that the agent has outgrown this difficulty tier.

MarioGPT response: Prompt escalates to `"many enemies, many gaps..."`. Maximum challenge engaged.

3.3.3 Transition → Low (Probability: 0.20)

When it happens: Agent struggles in Transition difficulty. Completion rate drops (<50%), deaths increase (>2.0 per level), reward trend turns negative.

What triggers it: The emission probabilities indicate the agent is overwhelmed. The HMM infers that the "true" skill state is likely Low, despite currently being in Transition.

MarioGPT response: Prompt softens to `"few enemies, no gaps..."`. Agent gets breathing room to rebuild confidence.

3.3.4 High → Transition (Probability: 0.25)

When it happens: Agent's performance degrades in High difficulty. Could indicate fatigue, policy degradation, or the difficulty being genuinely too hard.

What triggers it: Declining metrics over multiple episodes. The HMM doesn't react to a single bad episode—it requires a *pattern* of struggle.

MarioGPT response: Difficulty backs off to Transition, allowing recovery without fully resetting to Low.

3.3.5 Self-Transitions (Probabilities: 0.60–0.70)

Purpose: Stability. High self-transition probabilities ensure the system doesn't oscillate. An agent in Low state stays in Low unless there's *sustained* evidence of improvement. This is the "memory" that threshold systems lack.

3.3.6 Skip Transitions: Low ↔ High (Probability: 0.05)

When it happens: Rare. Only when metrics show extreme change—either sudden mastery or sudden collapse.

Why it's low: Jumping directly from Low to High (or vice versa) is disorienting. The Transition state exists precisely to smooth these shifts. The 0.05 probability allows for edge cases but discourages them.

4 The Transition State: Decision Point Metrics

The Transition state (S_1) is where the system decides the agent's trajectory. The following metrics form the observation vector that the HMM uses to compute emission probabilities.

4.1 Primary Metrics

Metric	How to Measure	→ Low Signal	→ High Signal
Completion Rate	Levels completed / levels attempted (last N episodes)	$< 40\%$	$> 80\%$
Death Rate	Total deaths / levels attempted	> 3.0	< 0.5
Reward Trend	Linear regression slope of rewards over window	Negative	Positive plateau
Time-to-Complete	Average frames to finish level	$> 2\sigma$ above mean	$< 0.5\sigma$ below
Progress Variance	Std dev of max x-position reached	High (inconsistent)	Low (consistent)

4.2 Composite Transition Score

These metrics combine into a single scalar $T \in [0, 1]$:

$$\boxed{T = w_1 \cdot \mathrm{CR} + w_2 \cdot \frac{1}{1+\mathrm{DR}} + w_3 \cdot \mathrm{RT}_{\mathrm{norm}} + w_4 \cdot \frac{1}{1+\mathrm{TTC}_{\mathrm{norm}}} + w_5 \cdot \frac{1}{1+\mathrm{PV}_{\mathrm{norm}}}} \tag{2}$$

Recommended weights: $w_1 = 0.25$, $w_2 = 0.20$, $w_3 = 0.25$, $w_4 = 0.15$, $w_5 = 0.15$. **Interpretation**:

- $T > 0.65$: Strong signal for High state
- $0.35 < T < 0.65$: Ambiguous; stay in Transition
- $T < 0.35$: Strong signal for Low state

4.3 Emission Model

Each state has a Gaussian emission distribution over T:

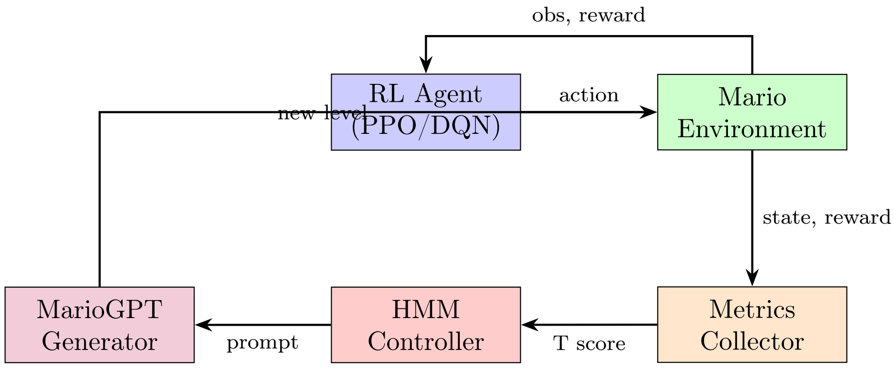
$$B_{\mathrm{Low}}(T) \sim \mathcal{N}(\mu = 0.25, \sigma = 0.15) \tag{3}$$
$$B_{\mathrm{Transition}}(T) \sim \mathcal{N}(\mu = 0.50, \sigma = 0.12) \tag{4}$$
$$B_{\mathrm{High}}(T) \sim \mathcal{N}(\mu = 0.75, \sigma = 0.15) \tag{5}$$

The tighter variance for Transition ($\sigma = 0.12$) reflects that this state has a narrower "expected" performance band.

5 Agent Architecture

5.1 Single Agent (Recommended)

Use **one RL agent** that experiences adaptive difficulty. The HMM operates as an external controller—it doesn't modify the agent's policy, only the environment.



5.2 Why Not Multiple Agents?

You could train three separate agents (one per difficulty) and use the HMM to select which agent plays. This is unnecessarily complex:

- No transfer learning between difficulties (jumping skills learned in Low don't help High agent)
- Three times the training compute
- Switching agents creates discontinuities in behavior

A single agent naturally transfers skills across difficulties and produces a cleaner learning curve.

6 Implementation

6.1 HMM Update Cycle

Listing 1: Core HMM Controller

```
import numpy as np

class HMM_DDA:
    def __init__(self):
        # Transition matrix
        self.A = np.array([
            [0.70, 0.25, 0.05],   # From Low
            [0.20, 0.60, 0.20],   # From Transition
            [0.05, 0.25, 0.70]    # From High
        ])
        # Emission parameters: (mean, std) for T score
        self.emission = [(0.25, 0.15), (0.50, 0.12), (0.75, 0.15)]
        # State probabilities
        self.belief = np.array([1.0, 0.0, 0.0])  # Start in Low
        self.states = ["Low", "Transition", "High"]

    def gaussian_pdf(self, x, mu, sigma):
        return np.exp(-0.5 * ((x - mu) / sigma)**2) / (sigma * np.
            sqrt(2 * np.pi))

    def update(self, T_score):
        """Update belief given new observation T_score."""
        # Prediction step: apply transition matrix
        predicted = self.belief @ self.A
```

```
        # Observation step: compute emission probabilities
        emissions = np.array([self.gaussian_pdf(T_score, mu, sig)
                              for mu, sig in self.emission])

        # Bayes update
        self.belief = predicted * emissions
        self.belief /= self.belief.sum()  # Normalize

        return self.states[np.argmax(self.belief)]

    def get_prompt(self):
        """Return␣MarioGPT␣prompt␣for␣current␣most␣likely␣state."""
        prompts = {
            "Low": "few␣enemies,␣no␣gaps,␣many␣pipes,␣low␣elevation
               ",
            "Transition": "some␣enemies,␣few␣gaps,␣some␣pipes,␣
               medium␣elevation",
            "High": "many␣enemies,␣many␣gaps,␣few␣pipes,␣high␣
               elevation"
        }
        return prompts[self.states[np.argmax(self.belief)]]
```

6.2 Metrics Collection

Listing 2: Computing Transition Score

```
def compute_T_score(episode_buffer, window=10):
    """Compute␣transition␣score␣from␣recent␣episodes."""
    recent = episode_buffer[-window:]

    # Completion rate
    cr = sum(1 for ep in recent if ep['completed']) / len(recent)

    # Death rate (inverted and bounded)
    dr = sum(ep['deaths'] for ep in recent) / len(recent)
    dr_score = 1 / (1 + dr)

    # Reward trend (normalized slope)
    rewards = [ep['reward'] for ep in recent]
    slope = np.polyfit(range(len(rewards)), rewards, 1)[0]
    rt_score = (np.tanh(slope / 10) + 1) / 2  # Normalize to [0, 1]

    # Time to complete (inverted, normalized)
    times = [ep['frames'] for ep in recent if ep['completed']]
    if times:
        avg_time = np.mean(times)
        ttc_score = 1 / (1 + avg_time / 1000)
    else:
        ttc_score = 0.2  # Penalty for not completing

    # Progress variance (inverted)
    positions = [ep['max_x'] for ep in recent]
    pv = np.std(positions) / (np.mean(positions) + 1)
    pv_score = 1 / (1 + pv)

    # Weighted combination
    T = 0.25*cr + 0.20*dr_score + 0.25*rt_score + 0.15*ttc_score +
       0.15*pv_score
```

```
    return T
```

6.3 Training Loop

Listing 3: Main Training Loop with DDA

```
def train_with_dda(agent, hmm, mariogpt, total_episodes=100000):
    episode_buffer = []

    for ep in range(total_episodes):
        # Generate level from current HMM state
        prompt = hmm.get_prompt()
        level = mariogpt.generate(prompt=prompt, temperature=0.7)
        env = level_to_gym_env(level)

        # Run episode
        metrics = agent.train_episode(env)
        episode_buffer.append(metrics)

        # Update HMM every 10 episodes
        if ep % 10 == 0 and len(episode_buffer) >= 10:
            T = compute_T_score(episode_buffer)
            new_state = hmm.update(T)

            print(f"Ep {ep}: T={T:.3f}, State={new_state}, "
                  f"Belief={hmm.belief.round(2)}")
```

7 Expected Results

With HMM-DDA, your learning curve should transform from two diverging lines into a single curve oscillating within the “flow zone”:

Metric	Current (Static)	With HMM-DDA
Easy final reward	∼+95	N/A
Hard final reward	∼-50	N/A
Target reward range	—	40–70 (flow zone)
Convergence time	20k (Easy only)	40–60k (adaptive)
State transitions/episode	—	0.05–0.15 (stable)

The key indicator of success is **flow zone percentage**: what fraction of episodes have rewards in the 40–70 range? Target: >60%.

8 Tuning Guide

If oscillating too fast: Increase self-transition probabilities (0.70 → 0.80) or widen emission variances.

If stuck in one state: Decrease self-transition probabilities or tighten emission variances.

If transitions feel abrupt: Add exponential smoothing to belief updates: $\text{belief}_{\text{new}} = \alpha \cdot \text{belief}_{\text{update}} + (1 - \alpha) \cdot \text{belief}_{\text{old}}$ with $\alpha = 0.3$.

If MarioGPT levels vary too much: Lower generation temperature ($0.7 \rightarrow 0.5$) for more consistent outputs.