Team [INSERT BEE MOVIE HERE]

UTMIST AI² Tournament

Tommy Yu, Jun Bin Cheng

Agent Explanation

Our agent is a modified version of the *Based Agent*. By utilizing if-else statements, it mimics the decision making process of players in a platform fighter game based on the environment.

- Always head towards the opponent using 'w' and 'd'. If opponent is to your left, move left, if the opponent is to your right, move right.
- If off the edge of the platform, come back using 'w', 'd', and 'space' no matter what.
- Keep attacking using different moves, with which moves being decided to use according to an internal timer.

Experimentation

The original agent was based on the *Recurrent PPO (RPPO) agent* rather than the *Based Agent*. This section describes the experimentation that was performed to make the decision to switch to the Based Agent, as well as how the Based Agent was fine-tuned.

RPPO Agent

Our initial choice of the Recurrent Proximal Policy Optimization (RPPO) agent with an LSTM policy aimed to leverage temporal dependencies in combat sequences. For example, if an opponent frequently chains light grounded attacks into a heavy aerial down-attack, the LSTM could theoretically recognize this pattern and time evasive maneuvers to exploit vulnerabilities.

This approach aligned with platform fighter strategies where anticipating opponent behavior is critical to gaining advantage.

Research into platform fighter mechanics highlighted two core principles:

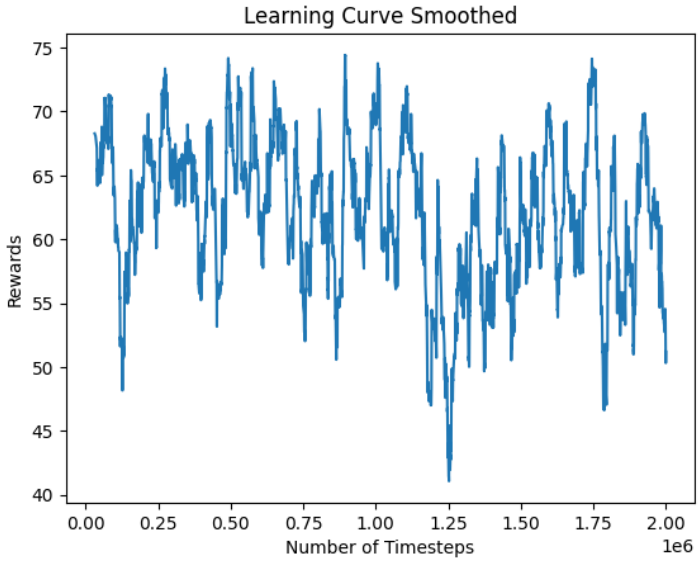
1. **Stage Control**: Players closer to the stage center (horizontally and vertically) gain a "state of advantage" [1], enabling juggling (airborne combos) and better reactive positioning against edge approaches.
2. **Damage Optimization**: Light attacks are optimal at low damage percentages for combo potential, while heavy attacks with high knockback excel at securing KOs at higher percentages [2].

Our original reward functions included:

- **Win**: Sparse reward/penalty for winning/losing
- **KO**: Sparse reward/penalty for getting a KO/getting KO'd
- **Stock Advantage**: Dense reward/penalize for every time step you have a stock advantage/disadvantage
- **Damage interaction**: Dense reward/penalization for applying/taking damage to opponent/yourself, scales with how much damage you already have
- **Combo**: Dense reward for every second opponent is hit while stunned
- **Vertical Danger Zone**: Dense penalty for being above a certain vertical distance from the stage
- **Horizontal Danger Zone**: Dense penalty for being above a certain horizontal distance from the stage
- **Advantage**: Dense reward for being closer to the middle of the stage than the opponent, <u>does not scale with distance</u>
- **Air Time**: Dense reward for every time step opponent is not on the ground
- **Move Efficiency**: Sparse reward for performing a light attack at smaller percentages and heavy attacks are larger percentages.

Entropy was initially set to 0.1 (exploration) and later reduced to 0.01 (exploitation). Additional experimental rewards (e.g., movement encouragement, anti-passive time penalties, whiff penalties) were discarded after inducing unintended behaviors, such as agents self-KO'ing to avoid penalties from missed attacks or prolonged matches.

Despite extensive hyperparameter tuning and ≈100 training hours, reward curves remained volatile and failed to converge.



From simulations, two major observations were made:

- **Performance Collapse Against the Based Agent**: Later iterations lost to the Based Agent they initially surpassed, suggesting overfitting to self-play strategies.
- **Non-Monotonic Skill Progression**: Later agents frequently lost to *earlier* versions of themselves, indicating a failure to discover robust policies.

We attribute both issues to **limited training diversity**: self-play created a feedback loop where agents optimized to exploit transient flaws in their immediate predecessors, rather than learning generalizable skills. This resulted in policies that were brittle, self-referential, and ineffective against opponents outside their narrow training distribution.

Given these challenges, we shifted focus to refining the Based Agent and it's rule-based (if-else) expert system. This decision prioritized interpretability and deterministic control over the black-box nature of RL, allowing direct encoding of stage control and combo principles without reward-shaping ambiguity. Subsequent efforts focused on enhancing the baseline agent's logic for juggling, edge-guarding, and adaptive attack selection.

Based Agent

During testing, we observed that the original Based Agent frequently lost stage control due to unchecked momentum. For example, after executing a dash or aerial attack, the agent

would often overshoot the stage's edge because it failed to decelerate properly. To address this, we added a **velocity threshold**:

- The agent now only attempts to move left/right if its horizontal velocity is between **0 and 1**.
- **Exception**: The threshold is bypassed if the agent is near the horizontal "danger zone" (i.e., close to the stage edge), allowing it to prioritize survival over momentum control.

Lessons Learned

- Reward functions in reinforcement learning are very difficult to optimize.

- Expert systems are not to be overlooked.

References

[1] Wisely, "The Beginner's Guide To Rivals of Aether 2," YouTube, https://youtu.be/vIOBWkHXOAw (accessed Feb. 2025).

[2] L. Brown and J. Crowley, "Perceptron Q-learning Applied to Super Smash Bros Melee," https://web.stanford.edu/class/aa228/reports/2018/final112.pdf