A Pac-Man Agent Using Monte Carlo Tree Search

Member: Xiang Luo

1 Background

Pac-Man is a classic maze-chasing game where the player navigates Pac-Man to eat pellets while evading ghosts. I'm not good at this kind of game, thus would like to develop an autonomous agent capable of strategic navigation to complete the game.

I was inspired by a "chasing" story in a recent Genshin Impact main quest. In this project, the character Rerir takes the role of Pac-Man, while the ghosts are represented by Traveler, Aino, Albedo, and Flins. I designed and drew these characters using pixel art to integrate this theme into the game engine.

2 Solution

Monte Carlo Tree Search (MCTS) was selected as the core algorithm for the agent. MCTS is a decision-making algorithm commonly used in adversarial search. It is particularly effective in game environments where the entire state space is too large to be fully explored, e.g., AlphaGO.

The project consists of a Pac-Man game engine (built with index.html and associated web technologies) and a Python-based backend (navi.py) for the agents. Due to time constraints, the current implementation focuses primarily on the MCTS agent. While placeholders for A* and Deep Q-Network (DQN) agents exist, they currently operate using random movement logic.

The basic game engine was initially drafted using generative AI (Gemini) and subsequently modified to support specialized agent communication. The movement logic for the ghosts was informed by the behavioral patterns described on *pacmanonline*. For the MCTS implementation, I referred to the foundational work of Tom Pepels et al. regarding state definitions for Pac-Man agents, and consulted a BFS Pac-Man agent implementation to refine the agent's pathfinding and state-evaluation logic.
Furthermore, I referred to the AlphaZero implementation in the LightZero framework for the MCTS tree structure and the Predictor + Upper Confidence Bound applied to Trees (PUCT) formula. As a variation of the standard UCB formulation, PUCT integrates a prior probability into the selection phase, as shown below:

$$PUCT = \frac{Q_i}{N_i} + C \cdot P \cdot \frac{\sqrt{N_{total}}}{1 + N_i}$$

By incorporating this prior probability P, Rerir can prioritize branches with higher potential based on heuristic guidance, leading to more efficient state exploration.

3 Requirements

VS Code or other platforms that can run Python scripts.
Websockets: `pip install websockets`. (My version: Python: 3.11.14, websockets: 15.0.1.)
Since the game engine will run in a web and agent runs in Python terminal, websockets can connect them.

4 Running Process

Note: If anything goes wrong, refresh the browser or use Ctl + C to stop and then restart the python.

Try the game yourself (optional):
1) Double-click the HTML file (index.html) to launch the game in your default browser. You can also use another browser you prefer.
2) Move Rerir (light blue character at the bottom of the maze) in any direction (J: left, L: right, I: up, K: down) to avoid enemies and eat pellets (small yellow points) and heart-fragments (larger red chunks) in the maze.
3) Win the game: eat all pellets and heart-fragments in the maze and be caught by enemies no more than twice. You will see a message pop up: "Rerir Werewolf!" If you are caught by enemies three times, the message will be: " YOU ARE ARRESTED! (Rerir has run out of lives.)"

Use agents:
1) Open the HTML file in a browser.
2) Run navi.py in VS Code terminal. `python navi.py`
If succeed, terminal will automatically return: "AI WebSocket server started, listening on ws://localhost:8765...".
3) Back to the browser. There are two ways to test agents:

- Enemies staying in the cage: You can find the skills part on the right side of the screen. Click "Apply" to use the method.
- Enemies chasing: Move Rerir (light blue character at the bottom of the maze) in any direction (J: left, L: right, I: up, K: down) to wake the enemies. Then apply agents by click "Apply" in the skills part on the right side of the screen.

Currently MCTS is the only functional AI agent. Both the A* and DQN navigation modes currently default to random movements.
4) Watch the agent moving.

5 Result

The performance of the MCTS agent was functional but fell short of expectations. In my manual testing across approximately one hundred trials with active enemy pursuit, the agent failed to secure a victory, achieving an average final score of roughly 1100.

Two specific behavioral issues were identified and addressed slightly during development:

1. Bottom Area Bias: Rerir occasionally exhibited a horizontal bias in the bottom line of the maze, moving left and right instead of progressing upward. To address this, I implemented bonus scores to encourage upward movement.

2. Oscillation Loops: Because next-move selection relies on heuristic scores, Rerir often entered "back-and-forth" loops when neighboring tiles had identical or nearly identical values. The agent could not autonomously break these cycles.
Here are my solutions: **Oscillation Penalty Solution:** A specific penalty was added to discourage repetitive oscillating patterns. **Enemy Threats Solution:** Observation showed that Rerir often breaks these loops when enemies approach. This is because the highest priority assigned to enemy avoidance effectively "re-values" the available movements and forces Rerir to run away.

The agent performs significantly better on long, straight paths than in confined areas with multiple walls. This difference probably result from two factors: **1. Computation Latency:** The evaluation process is time-intensive. If Rerir is moving upward while the agent calculates a left turn, Rerir may pass the intersection before the decision is finalized, resulting in a collision with the now-present wall. **2. Environmental Complexity:** The presence of four simultaneous chasing enemies creates a high-pressure environment that Rerir failed to get out of their surrounding.

These two factors are also the directions for me to improve the agent in the future.

6 Reference

[1] Pepels, Tom, Mark HM Winands, and Marc Lanctot. "Real-time monte carlo tree search in ms pac-man." *IEEE Transactions on Computational Intelligence and AI in games* 6.3 (2014): 245-257.
[2] PacmanOnline.org. "Pacman Online Game Manual." *Pacman Online Official Website* (2024): Accessed Nov 12, 2025. https://pacmanonline.org/?lang=manual.
[3] Ech-Chader, Othmane. "bustersAgents.py." *GitHub Repository: pacman-RL* (2021). Accessed Nov 3, 2025. https://github.com/othmaneechc/pacman-RL/blob/main/pacman/bustersAgents.py.
[4] OpenDILab. "ptree_az.py." *GitHub Repository: LightZero - A lightweight and efficient MCTS/AlphaZero algorithm toolkit* (2023). Accessed Dec 20, 2025.
https://github.com/opendilab/LightZero/blob/main/lzero/mcts/ptree/ptree_az.py.