Intelligent Pac-Man Agents Using Search, Adversarial Planning, and Reinforcement Learning

Ananya Sunil Nair
University of Minnesota
nair0177@umn.edu

December 2025

1. Problem Description and Contribution

The goal of this project is to design and evaluate intelligent agents capable of autonomously playing the classic arcade game *Pac-Man*. The objective for each agent is to maximize score by collecting pellets while avoiding ghosts. The Pac-Man environment is dynamic and partially observable, making it well suited for exploring classical AI techniques such as uninformed search, heuristic search, adversarial search, and reinforcement learning.

My primary contribution is the design and implementation of a complete Pac-Man simulation framework in Python, along with multiple intelligent agents:

- A **RandomPlayer** baseline
- **Search-based agents**: BFS, DFS, UCS, Greedy Best-First Search, and A*
- **Adversarial agents**: MinimaxPlayer and AlphaBetaPlayer
- An **AdvancedPlayer** using Q-learning (reinforcement learning)

I also implemented a unified evaluation framework to compare these agents quantitatively using metrics inspired by the UC Berkeley CS188 Pac-Man project. All agents interact with the same environment, ensuring a fair comparison. All code written specifically for this project is located in a single Colab-compatible Python notebook, with each agent implemented as a separate class (e.g., `RandomPlayer`, `BFSPlayer`, `MinimaxPlayer`, `RLPlayer`). No external Pac-Man codebases were used.

2. Related Work and Supporting Literature

This project is inspired primarily by the Pac-Man projects developed for the UC Berkeley CS188 Artificial Intelligence course. These projects demonstrate how classical AI algorithms can be applied in a game-based environment to illustrate planning, decision-making, and learning.

Key references include:

- Russell and Norvig's *Artificial Intelligence: A Modern Approach*, which provided theoretical background on search algorithms, adversarial search, and reinforcement learning.
- Sutton and Barto's *Reinforcement Learning: An Introduction*, which guided the implementation of Q-learning and reward-based policy optimization.
- The UC Berkeley AI Pac-Man Project website, which influenced the choice of evaluation metrics such as win rate, survival time, and score.

Additional clarification on algorithm implementation details was obtained from official Python documentation and NumPy references, particularly for data structures and numerical operations.

3. Software Requirements

The project is implemented entirely in Python and requires the following software:

- **Python 3.8+**
- **NumPy** for numerical operations
- **Matplotlib** for plotting performance metrics
- **Pygame** for visualization and animation
- **Google Colab** (recommended) or a local Python environment

No proprietary software or datasets are required.

4. Instructions for Running the Project

The project is designed to run seamlessly in Google Colab.

1. Open the provided Colab notebook.
2. Install required dependencies by running:

```
pip install pygame numpy matplotlib
```

3. Run all code cells sequentially to initialize the Pac-Man environment.
4. Select the desired agent (e.g., RandomPlayer, AStarPlayer, RLPlayer).
5. Execute the evaluation cell to run multiple games and collect metrics.
6. Visualization windows and Matplotlib plots will display agent behavior and performance.

5. Results

Agents were evaluated using the following metrics:

- Average score over multiple games
- Win rate (percentage of games completed without capture)
- Average survival time (number of steps)
- Learning curve for the reinforcement learning agent

The RandomPlayer consistently performed the worst, serving as a baseline. Search-based agents such as BFS and UCS improved path efficiency, while Greedy Best-First Search achieved faster but sometimes suboptimal performance. A* consistently outperformed other search agents due to its informed heuristic.

MinimaxPlayer and AlphaBetaPlayer demonstrated improved ghost avoidance, with Alpha-Beta pruning significantly reducing computation time while preserving optimality.

The AdvancedPlayer using Q-learning achieved the highest overall performance after training, surpassing all other agents in both score and win rate. The learning curve showed steady improvement, validating effective policy learning.

Final Scoreboard

Agent	Win %	Avg Score	Avg Steps
Random	0.0	-181.20	102.0
BFS	100.0	82.40	18.6
DFS	0.0	-193.00	153.4
UCS	100.0	81.00	20.0
Greedy	100.0	80.40	20.6
A*	100.0	77.60	23.4
Minimax	0.0	-200.00	200.0
AlphaBeta	0.0	-200.00	200.0
RL	0.0	-189.80	150.2

Table 1: Performance metrics of all Pac-Man agents evaluated over multiple games.

6. References

1. Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

2. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

3. UC Berkeley AI Pac-Man Projects. `http://ai.berkeley.edu/project_overview.html`