AI Project Final Report - Pacman Game Play

Multi-Agent Search using Minimax with Alpha-Beta Pruning

Authors

Theodore Klausner (tjk223), Glenn Baevsky (ghb65), Akshay Yadava (aay29)

Presentation Link

https://drive.google.com/file/d/18qL0N7X9qPB5ny9VgNvJwQYlrfxkdYnE/view?usp=sharing

AI Keywords

Multi-Agent Search, Minimax Algorithm, Alpha-Beta Pruning

Application Setting

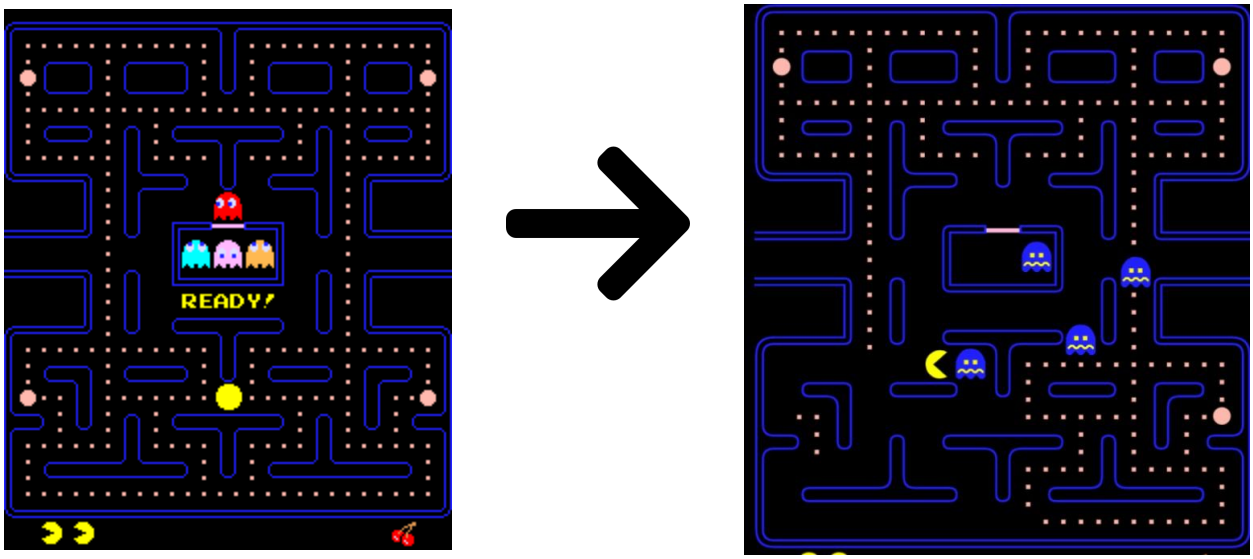
Open-source Pacman game code found at https://pacmancode.com/

Table of Contents

Project Description

For our 4701 project, our team chose to build an AI to play the Atari game Pacman. We initially dove into many ideas for how to build this AI including State-Space Search Algorithms and Reinforcement Learning (Q-Learning) implementations, but ultimately decided upon Multi-Agent Search. Pacman is a game consisting of a main player (Pacman) and four opponents (Ghosts). The objective is to complete each level by eating all the food pellets. There are also ways to maximize a player's points such as consuming ghosts after eating a power pellet or going for fruit. There are 14,600 points possible for each level, including the fruits that appear periodically throughout. The first fruit, the cherry, gives a small bonus of 100 points, progressing to higher levels like the Key level. You can receive as high as 5000 points per fruit eaten. Once you eat a power pellet, all the ghosts in the maze enter Freight Mode and run away from you. As shown in Figure 1, you can eat the ghosts for more points, effectively doubling the points from 200 to 1600 per ghost eaten.

Figure 1
Ghosts in Freight Mode



We decided to base our success on Pacman's ability to complete levels and increase its score as the AI develops. This meant strategically choosing Pacman's moves to eat all the pellets without being caught by a ghost. Given the amount of characters the game consists of, we thought Multi-Agent search would best help us achieve our goal.

Initial Implementations of our AI

Initially, we tried implementing different State-Space Search algorithms. This included depth-first search, breadth-first search, and iterative deepening, each time progressing a little better towards our goal, but ultimately falling short on what we wished to accomplish. Although this gave us great insight into constructing our search tree for future implementations, it did not perform well against all four opponents. From then, we knew we needed something to balance Pacman's decisions against the ghosts' moves.

From these search algorithms, we moved to Q-Learning which can value an action in a particular state without a model of the environment. This also led to performance issues which we discovered fairly quickly. At this point we knew we needed a model-based algorithm that can score future game states ahead of time.

Next, we tried a Reflex Agent Algorithm. This did not perform well due to its limited knowledge of the percept's history. Also, since the environment of the Pacman game changes very often, the rules the agent followed had to be updated frequently. The game state of Pacman is far from static. Ghosts change from being opponents to being a reward after consuming a power pellet. Some ghosts have a goal of hitting Pacman from the start while others randomly patrol the maze. Fruits to increase a player's score appear then disappear. Portals on the left and right sides of maze transport a player across the screen. All these facets had to be accounted for when building our AI which made Reflex Agent a poor choice. We needed something that could iterate upon a changing game state fast to account for all these adjustments as the Pacman moves from one position to another. This is how we ended up at our final solution, Multi-Agent Search.

Multi-Agent Search

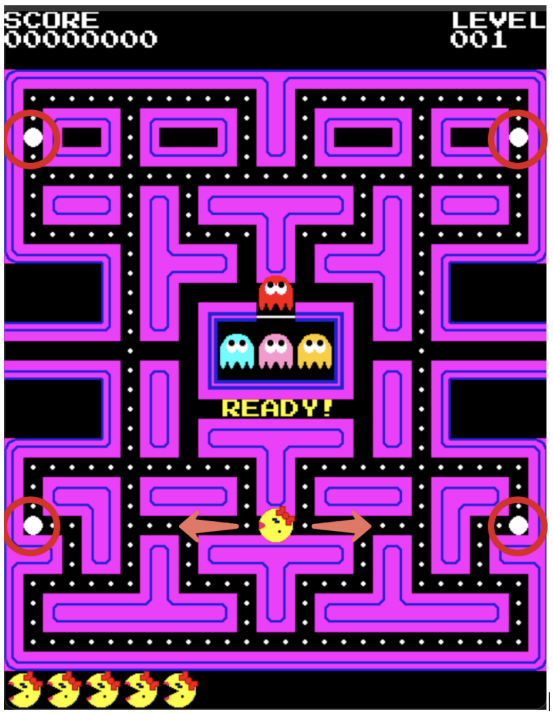
For our Multi-Agent Search, we decided to implement the Minimax Algorithm to simulate games a designated number of turns ahead. As our algorithm improved on time and efficiency, we were able to increase the amount of moves (depths in our game tree) the algorithm considered. The first step to our algorithm was getting a working game state that could be modified as agents made a succeeding move. In our game state we put Pacman's current position, the four ghosts' positions, a list of legal moves each of these five characters can move in without hitting a barrier or maze edge, a list of the pellets still available to be eaten by Pacman, and a frame counter. This gave us all the information we needed to eventually be passed into our game state evaluation function once a leaf node was reached.

To get each of these game state attributes we had to implement a lot of helper functions within the Pacman, Ghost, and Pellet classes in order to update a simulated game without

tampering with the game state of the live game. This initially posed a challenge for us as we were struggling to separate simulated games in the Minimax tree from the actual live game state. Since the source code of the game had no documentation, we struggled with understanding the purpose of each of these functions in the separate classes. We had to run many iterations of the game to understand which built-in functions dealt with updating the game's agents and items. After that, it was a matter of building new functions off of these functions to update our simulated game state.

Just like the actual game does, we treated each of the four ghosts and Pacman as a separate agent alternating whose turn it is to move to their next position. Each time one of these agents moved, the game state had to be updated before proceeding to the next depth in the tree. In order to handle this we created functions that returned the new position of these agents without updating their actual position in the game, and functions that returned all possible legal moves that could be made by that agent in their next iteration. Updating the list of pellets available was not so difficult. We simply checked whether or not Pacman collided with a pellet each time they moved and if so removed that pellet from the game state.

Figure 2
Highlighting the Power Pellet Positions and
Pacman's Legal Moves From its Current Position



Our biggest challenge in predicting future game states was tracking the individual ghost's movement. Each ghost in the game has its own personality, some head towards Pacman and

others randomly pick a goal and move towards that goal. Since these opponents do not act optimally to minimize the player's score, our Minimax algorithm would not have been perfect unless we accounted for this. Also, as levels advance, the opponents become smarter and faster as more ghosts flood to hit Pacman quicker. This made it so we could eliminate nodes on our game tree that dealt with ghosts moving in positions they would not necessarily move in…unless they entered Freight Mode. If at some point in our game tree path that Pacman had hit a power pellet, ghosts would start fleeing in unpredicted ways, making it very difficult to simulate those movements.

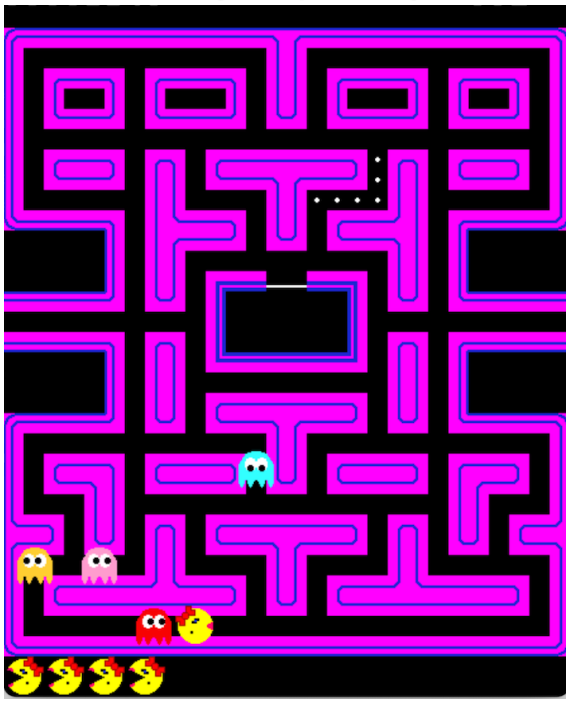
The source code defined this way of movement as Scatter, where the ghosts pick a random direction to move away from Pacman to avoid being eaten. This logic posed a huge struggle for us we were not able to overcome in our game simulations. As we started to get to higher depths in our Minimax tree, our runtime increased significantly. To combat this issue we implemented a key feature to our AI: Alpha-Beta Pruning. Alpha-Beta Pruning further decreased our runtime as we got lower depths, pruning large sections of the tree that were unnecessary to iterate through.

Evaluation Function

After Minimiax was implemented for our Multi-Agent search, we then were tasked with creating an evaluation function. This function scores a game state at a given time in order to return the most optimal move for an agent at that position. The main goal of the game is to to stay alive as long as possible by evading enemy ghosts while maximizing points. When Pacman is traversing the search space, there are many factors it must assess in order to make the most optimal move. Initially, we decided to implement the evaluation function by assessing the total distance to each pellet and each ghost in a given game state. For the ghosts, we decided to sum the total Manhattan distance of a ghost to Pacman's current position. Greater distances to the ghosts are seen as better and thus receive a higher score for that game state. For tracking the location of food pellets, we similarly implemented this with the Manhattan distance, though shorter locations to food pellets were seen as better and further distances were worse. This led us to add the reciprocal of the distance to food pellets in a new game state to our evaluation function score. Although we had some issues with weighting the various components of our evaluation function, these two factors created our first initial implementation of our function.

Figure 3

Pacman running away from ghosts



Upon running a few initial tests of our algorithm, the AI was actively trying to eat pellets while maximizing its distance to ghosts. This caused an issue to arise. Even when there were surplus pellets in a local region and some located in a different quadrant of the board, the AI would still try to move in the direction of pellets further away, while undermining the pellets located locally to the agent. To combat this issue, we added another factor to our evaluation function, the amount of pellets located in a local region. This worked by summing up the scores of pellets in a radius of 50 units to Pacman in this static game state. We then continuously decrease this radius by 10 and recalculate how many pellets remain locally. The intuition was that this factor of the evaluation function would now weigh local pellets more than the total pellets located on the board, prioritizing those closer to the agent. Upon implementing this into our algorithm, we found that the Pacman AI would now target local pellets even if there were some present in a different location of the board, solving our initial problem. Still, a new problem arose. Even though we assessed the total distances to ghosts in this game state, when a ghost came near a Pacman, it would not always move in the optimal direction to maximize its distance to the ghost because of the presence of these other factors.

This led us to implement one more factor into our algorithm, an insurance to check that the ghost will choose not to move to a game state that is within a certain distance to a ghost. This part of the algorithm returned an extremely low score when processing the Manhattan distances

to the ghost. In some instances, all the possible moves a Pacman could make in a given game state would be within the given radius to a ghost, thus not having a proper way of evading them in that situation. One final assurance check added to the algorithm was that whenever a ghost was within 5 space locations to the Pacman, the only criteria of the evaluation function would be to maximize Pacman's distance to the nearest ghost. This final factor completed our implementation of our evaluation function.

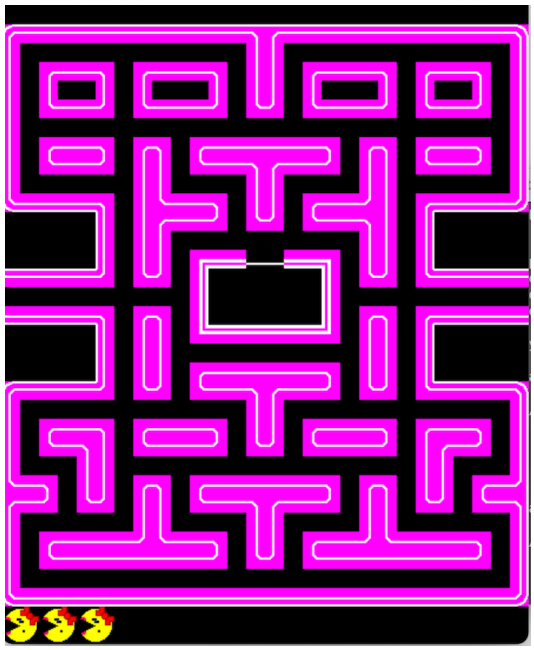
Evaluation

For the evaluation of our system, we took a three prong approach and sought to answer the following questions. How well does the pacman perform? This question aims to assess not only the maximum score attained by playing, but also how efficiently the AI could complete a level. We also wanted to assess how the pacman fled ghosts when they were in the normal mode in addition to seeking them out when in frenzy mode. This question sought to validate the pacman's flexibility: a single minded AI would only seek food and run away from ghosts, but an optimal AI would also chase the ghosts after eating a power pellet. These questions were answered by both user testing and by running 50 simulations of our AI. We asked 50 users to play pacman without giving them prior knowledge of what we were after. We wanted to show unbiased results for human gameplay. After having them complete a level, we asked them what their goals were for completing a level. This is analogous to the goals we set for our pacman AI, i.e., balancing level completion with maximization of score. We then asked how they would play differently given different scenarios: play without losing a life, play by maximizing your score, play by completing the level as quickly as possible. We compared the results of the human trials with the average of the AI after 50 runs. We also did a baseline test for our AI by programming a random state machine that randomly chose the next action for the AI.

We recorded our results and found the following: the pacman with random moves had a mean score of 800.32 with a maximum score of 1200, the AI pacman had a mean score of 2432.49 and a maximum score of 3400, and the human trials had a mean score of 2243.67 and a maximum score of 4000. We can conclude from these results that the AI pacman had a statistically significant run when compared to its randomized counterpart. These results also show that the AI pacman beat the human trials on average. The maximum score results show that a human was able to best the AI, showing that the AI could be improved further. The qualitative feedback we gained from the human trials after asking the aforementioned questions could drive improvement for our AI. We found that humans were far more strategic than the AI taking into

account such strategies like “bading” the ghosts to their advantage. Most human testers however simply avoided the ghosts and performed similarly to the AI, attempting to eat all the pellets and complete the level quickly. This showed that our AI performed adequately, not the best AI in the world, but performed similar to how a human would.

Figure 4
Completed level



To fully evaluate our AI system, we wanted to not only look at the statistics (as described above) but also how the different parts came to be. We started off with a pacman game that was only controllable by player input and ended with an automated entity. Most of the initial challenges were in integrating the AI algorithms to the existing source code. It took some time to get acquainted with the open source code and to learn how to automate the pacman entity. However once we passed this road block, we were able to try advanced AI algorithms (as described above) and see how the pacman performed. Overall, the different parts of the system (automated pacman entity, state space search, and best action evaluation) came together quite successfully. A challenge that prevented us from making a perfect AI that would maximally solve every level was the non determinism of the ghosts. The algorithm we ended up going with (Minimax) was based on the fact that the enemies played optimally; however, the enemies in the game were in fact controlled by pseudorandomized state machines. Each ghost had its own goal and to reach that goal, randomly chose a possible action based on the legal options at that position. This prevented the Minimax algorithm from taking into account the actual future

positions of the ghosts; instead we had an estimate of their future positions. This estimate however worked out and was good enough to effectively have the AI pacman evade ghosts and successfully complete levels.

To make that perfect AI, we would have had to either change the ghosts behavior (thus making them deterministic) or go with a different optimization algorithm. If we were going to do this again, we would have looked into building an algorithm that used probabilities to look at the expected position of the ghosts. Our AI was successful if compared against a novice player but was by no means an expert. So our success is based on the fact that we were not only able to automate the pacman to move but also able to have it on average play as well as a novice player. It was able to use the goals we coded (avoid ghosts and maximize points) to navigate the pacman world. In conclusion, we evaluate our pacman game as a success within the scope of the aforementioned conditions.

References

1. Open-Source Pacman Game Code
   https://pacmancode.com/

2. Playing Pacman with Multi-Agents Adversarial Search Research Paper
   https://davideliu.com/2020/02/13/playing-pacman-with-multi-agents-adversarial-search/

3. ResearchGate Paper on Pacman AI
   https://www.researchgate.net/figure/The-multi-agent-search-project-emulates-classic-Pac-Man-but-using-smaller-layouts-and_fig2_228577256