Machine Learning I

Tutorial 1 - February 15th, 2022

Cooper Ang(100480324) & Niv Magril(100479662)

Exercises

Answers are Italicized

1. What information is shown in the interface? And in the terminal? Which position does Pac-Man occupy initially?

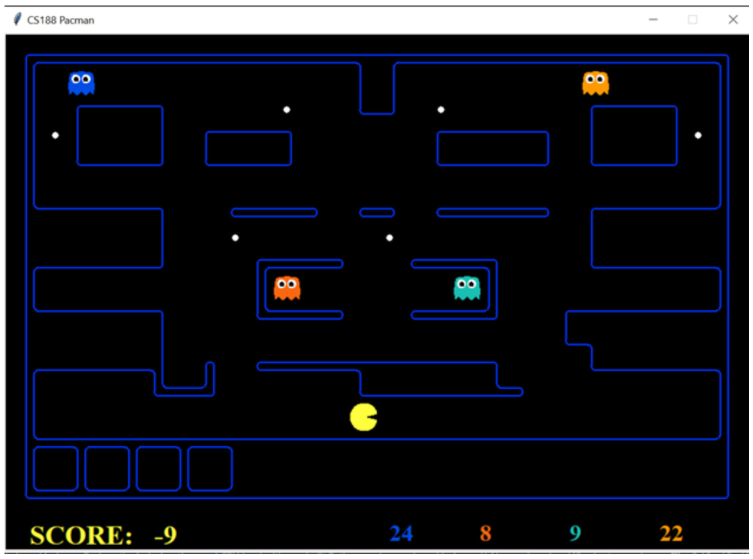
   The information that is shown on the interface is the score and the distance to each of the ghosts. In the terminal when a pac dot is eaten "REMOVE" is printed to the console. The position of the Pac-Man is initially occupied at (12, 10).

2. In your opinion, which data could be useful to decide what Pac-Man should do on each tick?

   The current position and direction of the pac-man, the position of the walls, distance/location of the ghosts, current tick, and the position of the pac-dots.

3. Take a look to the layout folder. How are the mazes defined? Create a new maze, save it and execute the game on it. Include a screenshot of the maze in the document.

   Mazes are defined as a ".lay" file, using some characters to specify if *there are walls (%), ghosts (G), and pac dots (.).*



4. Execute the agent BasicAgentAA with the following command: python busters.py -p BasicAgentAA Describe which information is shown in the screen about the game state. Which information do you consider the most useful?

   The width and height of the map is provided, the pacman position, the legal actions (directions, and stopping), the pacman's current direction, the number of ghosts, a list of Booleans of the living ghosts, a list of the positions of the ghosts, the ghost directions, the distance to the ghosts, the number of pac-dots, the distance to the closest pacdot, a map (where T is a wall and F is movable space), and the score. All information is useful, but to play the game it is most useful to know the position of the pacman, the legal actions, the map, and the score. With this bare amount of information, the game can be completed, the additional information enhances the awareness of the agent to be able to complete the game with a higher score.

5. Program a method called printLineData() inside the BasicAgentAA agent from the bustersAgents.py file. This method should return a string with the information from the Pac-Man state you consider relevant. Then, you call this method from the main loop of the game in Game.py to write the

information to a file. For each tick, you should store a line with all the considered information, splitting each data with commas. Moreover, each time a new game starts, the new lines must be appended below the old ones. You should not rewrite the file when a new game starts.

6. Program an “intelligent” Pac-Man. To do so, modify the method chooseAction() from the BasicAgentAA class. As you can see, by now it just act randomly. Your new Pac-Man should chase and eat all the ghosts in the screen. Compare the results obtained by executing the game with the static and random ghosts (-g RandomGhost). Play several games and determine the number of ticks (on average) that your agent needs to eat the ghosts.

   On average using our default maze and stationary ghosts we find that our agent takes 33 ticks to complete the game. On the other hand, when using random ghosts we find that it takes 50 ticks on average to terminate the game. Of course this result varies depending on how far the ghosts are to each other upon the consumption of each ghost and the maze we choose.

7. The agent you just programmed does not use any machine learning technique. What advantages do you think machine learning may have to control Pac-Man?

   By applying machine learning techniques we can opt for a more adaptable “intelligent” agent. Specifically using previous results we may construct a model to decide the optimal ghost consumption order, pac-dot consumption, and direction/moves that our agent should choose. Using input vectorized features or attributes - such as wall location matrix, distance and direction of the nearest ghost, and predictable paths of the ghosts - we may optimize our agents abilities and “train” a model to attempt to play pacman, which may or may not be *better than what is implemented in the lab. For games such as pacman an approach of reinforcement learning could be helpful. A machine learning approach may result in a more adaptable approach to complete the game, which would rely not on hardcoded ordering of consumption and neglecting to eat the pac-dots as seen in the currently coded approach.*

Description of Methods

Description of the method implemented to save the game state on each tick.

First on startup if not created already a csv file will be initiated with a header providing the schema to the csv. To save the game state in a csv the agent printLineData method is called on every tick. Using the provided methods from the GameState class the pacman position, pacman direction, direction of all ghosts, distance of all ghosts, a flattened version of the walls, the wall dimensions, the position of all the ghosts, the legal actions of the pacman, a flattened version of the food/pac-dots, a boolean array of the living status of the ghosts, and the score are collected. Once collected they are processed to a format that is readable in a csv. A string with the information from the tick is returned from the printLineData method. In the loop of the game the returned string is appended to the already opened file.

Description of the behavior programmed for the BasicAgentAA.

The agent uses the breadth first search (BFS) algorithm to find the shortest path to the targeted ghost when provided the ghost's location. This is implemented in the findPathBFS method in the BasicAgentAA. The BFS algorithm is a way to traverse a graph and find the path to a target. In this case a grid which first searches the current node (the pacmans location), and then explores all nodes which are neighbors to that searched node. A queue is used to keep track of which node to search next and its corresponding neighbors, and an additional 2D grid is used to ensure a node is not visited twice. The implementation of BFS here uses the boolean 2D grid of the walls to inform which of the adjacent cells of the current node are searchable. The BFS searching algorithm A dictionary of parent child relationships is kept to provide a *way to backtrack and find out which was the shortest path once the ghost is found. Once located a path is returned, and the path is then converted to the cardinal directions used in the game. A variable keeps track of the path, and a new path is computed if the ghost moves from the last known position. The agent implementation will go for ghost 1, 4, 3, 2 - in this order. The consumption of pac-dots are not implemented in this agent.*

Conclusion

Through this lab we explore a non-machine learning approach to the game of pacman, and through this report we illustrate the difficulties and redundancies that can occur when neglecting machine learning methods. Through the progression of this course we will develop skills to implement machine learning methods instead of the hand crafted methods as presented in this lab. As mentioned in the seventh question we believe a more adaptable program can be achieved for the tasks completed. In future labs we hope to explore ML methods in environments similar but not limited to our Pacman environment.