Project Proposal - Team Strawberry Cake

Osvaldo Moreno Ornelas and Paris Hom

Description of the Project

The goal and purpose of this project is to create smart, self learning agents that are capable of winning 97% of the time on every map against any opponent. The goal is also to create an agent that is able to optimize the paths it takes on any maze.

A couple challenges that we will face include: limit of 1,200 moves on every map, maps may have different amounts of food, and the algorithm we choose may not work on every map. The differing number of food will result in needing to capture more food before we can win early.

Some opportunities that arise are that we will be able to explore new algorithms or techniques such as MinMax, Alpha-Beta, and Reinforcement Learning. Additionally, we have the opportunity to explore and develop our own self learning agents through this project.

Requirements

PacMan Agent

- The PacMan agent should be able to increase the score with more training
- Using Keras-rl and OpenAI Gym library to help train a Deep-Q reinforcement learning model
- The PacMan agent should learn to avoid high casualty routes
- PacMan learns when is the best time to come back with food (avoid being eaten with a lot of captured food).
- PacMan agent should try to remain offensive and capture food while also avoiding enemy ghosts.
- The PacMan agent should learn how to get out of a trap before the opponent ghost agent eats PacMan.

Ghost Agent

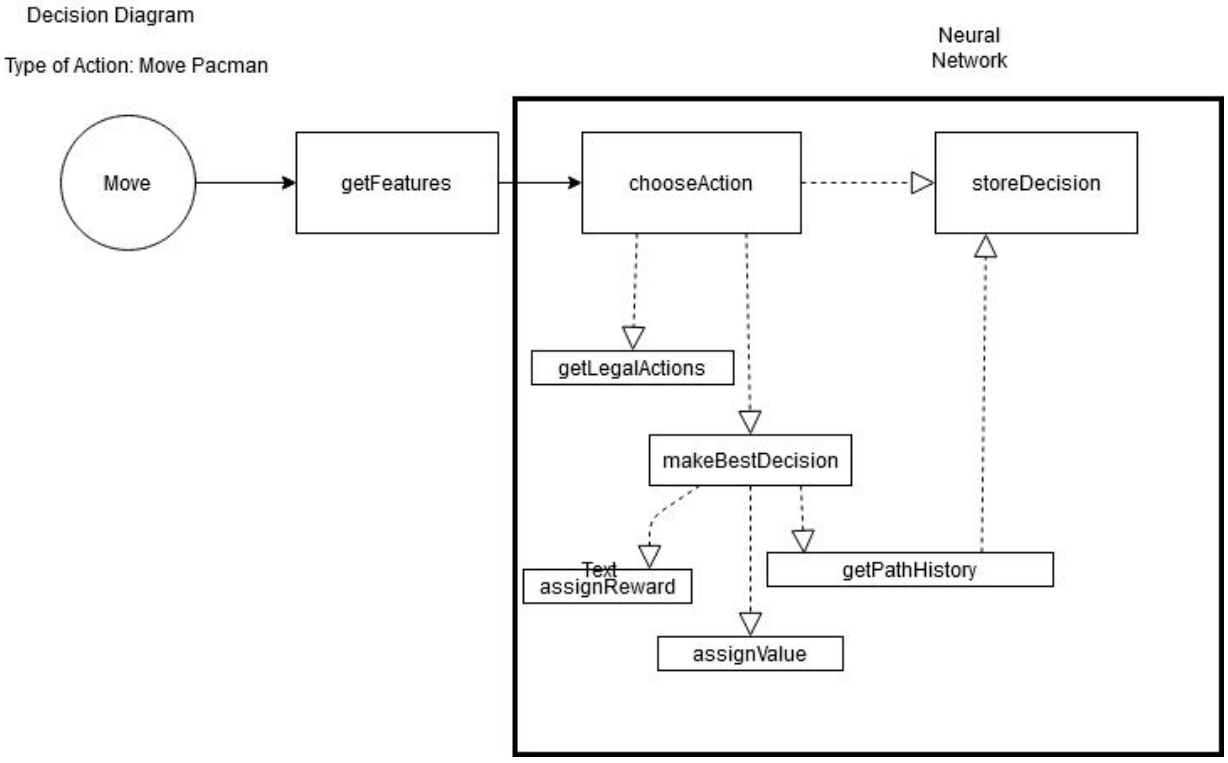
- The Ghost Agent should be able to accurately locate the opponent PacMan within 4 moves of entering our side.
    - The Ghost Agent will use a Hidden Markov Model to try to figure out where the enemy are through the noisy distance data
- The Ghost Agent should be able to eat the opponent PacMan agent before it returns back to its side.
- The Ghost Agent should be able to decide when is the best time to eat the opponent PacMan

Modules

PacMan Modules

- Purpose: PacMan agent should be able to learn through trial and error the best action to take at every state.
- Our PacMan agent will need a decision maker. This will be created by using Deep-Q Reinforcement Learning on our agent.
- Reward Assigner to train our model with rewards and penalties
    - We will give rewards for our PacMan increasing our score (bringing back food)
    - We will penalize dying
- We need a value assigner so the agent can decide which action to take
    - This will calculate value of every action
- In order to store a Deep-Q Neural Network, we will be using a file. We will add and update the Neural Network file with every game (episode) that we play.
    - The Neural Network will be referred to by the Q value calculator to decide which action to choose.
    - This will help us train our model as we store more episodes

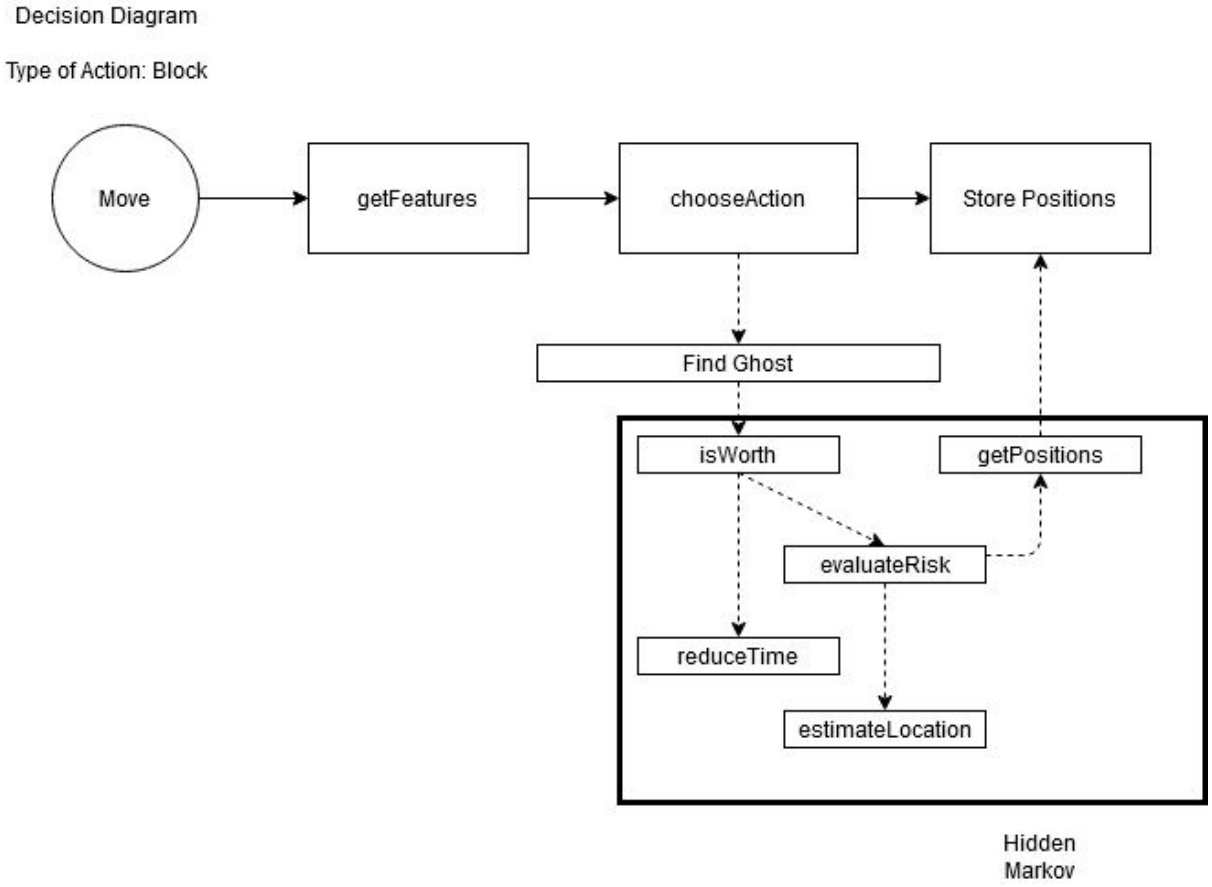
Action Diagram



Ghost Modules

- Purpose: Ghost agent should be able to defend our team's food and prevent the opponent from increasing their score.
- Ghost finds where opponent PacMan is within 4 moves of entering.
    - Using a Hidden Markov Model, estimates where enemy PacMan is.
    - Ghost will choose the action that minimizes the distance to PacMan the greatest
- Ghost should be able to calculate if it is worth it to get eaten by PacMan or run away
    - Calculate function for how far away from start and compare to how long Ghost will remain scared for
    - Reduces amount of time that the enemy can use to increase team score
- Store most recent found position
    - Allows Ghost to have a reference of the most recent location, in case it moves more than 6 distance

Action Diagram



We will test how these components are working by seeing if PacMan continues to go on a path that results in death. We can see if PacMan is learning by seeing if the win rate increases as well as the number of food. We can also test if the performance drops on different maps or opponents.

As for the Ghost Agent, we can test if the component is working if the Ghost Agent is able to locate the opponent PacMan within 4 moves of entering our side. The Ghost Agent will head towards this location and should find its opponent. Additionally, the Ghost agent should be able to quickly eat the enemy PacMan after finding its location.

Milestones

- Milestone 1: Create self learning agents, will require storage ability.
    - We will be able to test this Milestone by seeing if our agents continue to pursue a “bad” route, or if it learns and tries a different route.
    - This new route may or may not be a route that results in a higher score, but should just show that our agent learns that the “bad” route results in penalties.
- Milestone 2: Agent that wins 90% of time in all maps versus the BaselineTeam.
    - We will be able to test this Milestone by running our agent against the BaselineTeam on every map included in the layouts file 100 times and looking at the win rate for each map.

Detailed Design

The technical details of the modules described above. Similar to those of an API Example[1]. This will be updated as modules are added or deleted.

Pacman
On a high level standpoint, our PacMan agent should be able to learn and decide on the best path to take. Our team has decided on developing an agent that uses Deep Q-learning to meet this goal. We will be using Keras-rl and OpenAI Gym to help train a Deep-Q reinforcement learning model for our PacMan model. To gather data, we will run many episodes (matches) on every map listed in the layouts folder against the BaselineTeam. We can also gather data from running against our Agent0 implementation.

We will first create a “cheatsheet” of rewards and penalties. For example, we will assign a reward for increasing the number of points for our team. We will give higher reward for more points so that our agent learns to return back to base after consuming food. We will penalize the agent for dying. We will do this so our agent learns to avoid being eaten by an opponent.

We then need to create and implement the value function to calculate the Q-value. We can also use this to verify that we are properly assigning rewards and penalties. We will also have to decide the step size of the Q-value function for overriding old information.

In order to create the neural network to approximate the Q-value function, we need to preprocess and store game states into a Deep-Q Network file. This will then return the Q-value of all the possible actions in the current state. Our Agent will use a Greedy method and select the highest Q-value action to perform and store the transition state. We can then perform a gradient descent to minimize loss. We would repeat this until we are able to train our model to meet our Milestones.

After training our model with Reinforcement Learning, our model should be able to score higher points than the Agent0 agents. We do not know exactly how the PacMan agent will be able to learn to do this. We can assume from our own knowledge that a higher score would be a result of learning to change routes, the different states of enemy ghosts, and returning back to base carrying food.

Ghost

From a high level standpoint, our Ghost agent should be able to defend our team's food and prevent the opponent team from increasing their score. In order to do this, the Ghost agent should be able to quickly find the enemy PacMan after it enters our base. We will do this by trying to predict where the enemy has entered our side. We will use a Hidden Markov Model in order to implement this. Our Agent should be able to figure out where the opponent is within 4 moves of entering.

In order to implement a Hidden Markov Model, we need to define some parameters. The possible states will be defined by gameStates. Possible observations are possible coordinates of the enemy PacMan (through noisy distances). State Transitions will be defined by our Ghost agent moving North, East, South, West. The output probabilities will be defined as the probability that the enemy PacMan is at the coordinate observed.

After using the Hidden Markov Model to locate where the enemy PacMan is, our agent will pursue using the action that results in the closest maze distance. This will be done using a Greedy Strategy that minimizes the distance to PacMan the greatest.

Additionally, the Ghost Agent should be able to calculate if it is worth it to get eaten by the enemy PacMan or run away. This is because it may be worth it to be eaten quickly than try to avoid or run away. We can determine how far away from the respawn/starting point is from PacMan's current location and compare it to how long our Ghost will remain scared. This will reduce the amount of time that the PacMan can use to increase their score.

References

[1]: List Objects - https://docs.python.org/3/c-api/list.html
[2]: Reinforcement Learning and Implementation - https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/
[3]: Reinforcement Learning Tutorial - https://www.youtube.com/watch?v=LzaWrmKL1Z4
[4]: Q-Learning Tutorial - http://mnemstudio.org/path-finding-q-learning-tutorial.htm
[5]: I/O operations in Python using Pickle - https://towardsdatascience.com/optimized-i-o-operations-in-python-194f856210e0
[6]: MDP vs Q-Learning - https://www.guru99.com/reinforcement-learning-tutorial.html#:~:text=Two%20types%20of%20reinforcement%20learning,given%20sample%20data%20or%20example.
[7]: Markov Decision Process Model - https://www.geeksforgeeks.org/markov-decision-process/
[8]: PyTorch - https://pytorch.org/
[9]: TensorFlow https://www.tensorflow.org/
[10]: TensorFlow Training - https://www.tensorflow.org/guide/keras/train_and_evaluate
[11]: Deep Q-Learning - https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/
[12]: Keras-rl - https://github.com/keras-rl/keras-rl