Solving The Markov Decision Process within the Game of Pacman

6CCS3AIN - AI Reasoning and Decision Making

Reece Roberts
K20065137

Introduction

The Markov Decision Process (MDP) is a process to create optimal policies in a stochastic environment. It creates a policy for the agent to follow in order to find the most efficient method to achieve a goal *(Wakuta, 1992)*. From a mathematical perspective, the MDP has a set of states, a set of actions in each state, a transition model and a reward function. Each state in an environment is given a reward value based on the influence that state has on the environment, for example, any state that could potentially preemptively end a game and would result in the agent failing in achieving the goal will hold negative reward values.

The MDP factors in the randomness that the world has and includes this in the process to account for the possibility the selected action that the agent wants to complete may in fact fail. This means that a state that seems like the perfect transition state could potentially be a disastrous decision if failure to move to that state would result in reaching a negative terminal state. In order to find a solution to the Markov Decision Process a policy has to be selected.

A policy is the chosen action to take for every state in the environment. An optimal policy is one whereby the actions would seek to find the state that possesses the highest utility value, finding this for every state means that even when in a stochastic environment, if the agent was to execute an incorrect action, it wouldn't take long to go back to the optimal route.

In order to solve the MDP it is important to calculate the utilities of every state for an agent to allow the agent to understand the value that state has to it. The utility value is computed by taking the reward value of the current state, and summing it with the subsequent state that yields the highest reward, whilst simultaneously factoring in the values of states that could potentially be transitioned into given the stochastic nature of the environment.

In the game of Pacman, the goal is to eat all of the food on the map and avoid a collision with the ghosts that roam the environment, unless the ghosts are in a temporary scared state that is achieved when the pacman consumes one of the capsules in the map.

The MDP fits in with an altered version of Pacman where the user is the agent itself and when making a move the Pacman has a 20% possibility of making the incorrect move. The prerequisites are then set up for a MDP and a policy can be found that is optimal enough to achieve the goal of finishing the game without colliding with the ghosts.

Description

in order for the MDP agent to achieve a target goal with some degree of consistency, it was first imperative to deduce what the reward values would be for each state on the map. The first step to achieve this is to find all the states that are in the map. This was done by finding the maximum and minimum coordinates by calling the corners function in the API, from searching these values the most south-western and north-eastern coordinates can be found and from there all the coordinates can be stored.

The next obstacle is removing any wall coordinates that have been stored as these don't provide any use when finding the optimal policy. The remaining states (co-ordinates) were then broken down into categories. The categories were states that contained food, states with no food, states where there are ghosts that are not currently in a scared state, ghosts that were in a scared state and finally states that are within the direct vicinity of the ghost (coordinates that are within one x,y value from the aggressive ghosts). These categories are all stored in separate lists so no coordinates appear in two lists but all merge together into one dictionary that assigns them all a value of zero to initialise value iteration.

The ghosts are checked to see what state they are in as when they are in a scared state the ghost will have the same reward function as food, avoiding the reward function for scared ghosts being higher than the value of the food seemed reasonable as even though colliding with a scared ghost would lead to a higher average score, it wouldn't be optimal for the agent to be chasing the ghost and avoiding food as this could potentially have an overall adverse effect on the consistency of winning games. Setting the ghost to the same reward value as food means that the agent can capitalise on the opportunity if the ghost is nearby but also stops the agent veering away from food and putting itself into a potentially difficult situation.

The ghosts that were aggressive were set to a value of -2 in order to signify the importance of avoiding them. Since reaching these states would result in prematurely ending the game it is best to avoid them and also avoid their neighbouring states. The values given for the states around the perimeter of the ghost were -1.5 as they still are negative rewards but not to the same degree that the ghost is, but this value will still deter the Pacman from moving to this state. The non food states were given a value of -0.04 as it is not ideal for the Pacman to just wander around states it has already visited and that provide no benefit to the goal of the game unless they're bridging the gap between the Pacmans current state and the food states.

$$Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$$

Figure 1. The value iteration formula ***(Value Iteration, 2020)***

The food states were given a value of 2 in order to promote visiting these states over any other positive transitions in order to reach the goal state. From there the value iteration algorithm is implemented, the current values are saved as a copy in order

to allow each state to do the calculation without interfering with others. For every state, the legal moves around it are found and stored, if one direction is not a legal move, then the coordinates are assigned to the current state coordinates. Once the coordinates are stored then the utility value is calculated for all moves and the values are stored into a dictionary and the max value is then temporarily stored *(figure 1)*.

When the max transition state is stored the utility value is then calculated by adding the states reward value with the product of gamma (the gamma value was assigned as 0.8 which was found to be the optimal value through testing) and the max value. This is then repeated for every state and repeated 100 times (as convergence didn't seem to be computationally feasible after trialing different values, 100 seemed to be the optimal number to balance consistency and computational feasibility).
The MDP is then part of the decision for which state the pacman is going to move into next. The coordinates are found for all legal moves that the pacman can make and the utility value for each move is calculated based on the product of the utility value calculated from the value iteration along with the probability of moving into that location, this is then also repeated for moves that could be accidentally made when taking that action.

All the values are collated for every move and the state that yields the highest utility is the one that the Pacman would make the move on and thus solving the Markov Decision Process.

Results

When deciding which values were optimal to have a successful MDP agent the key values to test on are the gamma value and number of iterations. These were tested at different values to measure their efficiency with regards to the average score and number of wins, these were measured over 50 games in order to mitigate any abnormalities.

The number of iterations for value iteration was measured to see at what number of iterations can be effective whilst avoiding too much computational cost. For the number of iterations the results were derived from the small grid layout.

iterations	Wins over 50 games	Average Score
25	30	176
50	36	55.12
100	34	135
200	34	120

Figure 2. Table displaying the number of wins and average score for each increment of iterations

The selected value was set at 100 iterations as it wasn't too slow to run the agent whilst simultaneously keeping a consistent number of wins *(figure 2)*.

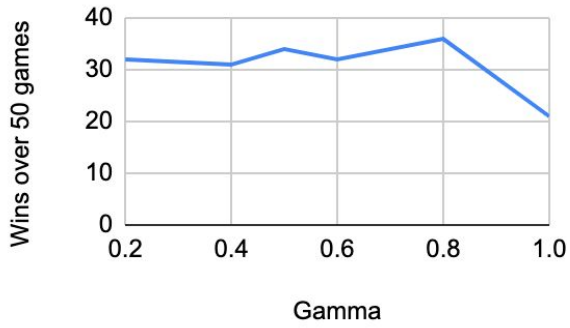

Figure 3. Graph that shows the number of wins over 50 games at each increment of Gamma

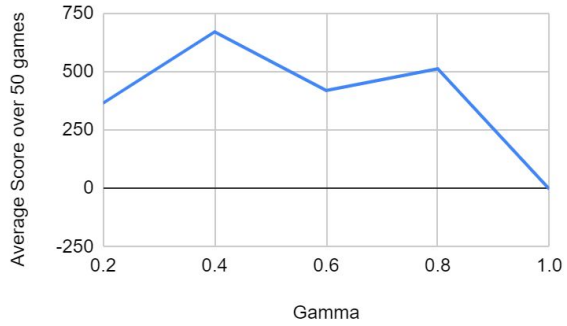

Figure 4. Average score over 50 games at different gamma values

Once the iterations were set the next thing to do was find the optimal gamma value. The different gamma values were tested to which value had the highest win rate, after testing the value that yielded the highest result was 0.8 with the highest win rate and the second highest average score *(figure 3 & 4)*.

Over 50 games the MDP agent with the most optimal values had a win rate of 40% in the medium classic and 68% in small grid. Considering the random nature of the game's mechanics when making moves I believe this win rate is quite effective and enables the agent to solve the MDP problem to a reasonable level. The agent utilises the value iteration method to make reasonable decisions by inferring from the reward values around the map to complete the goal of consuming all the food without colliding with a ghost.

References

Cs.cmu.edu. 2020. Value Iteration. [online] Available at: <https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node19.html> [Accessed 9 December 2020].

Wakuta, K., 1992. Optimal stationary policies in the vector-valued Markov decision process. Stochastic Processes and their Applications, 42(1), pp.149-156.