Using a Markov Decision Process and Value Iteration Algorithm to Win Pacman in a Non-Deterministic World

Gregory Verghese

StudentID: 1410855
King's College London

December 9, 2018

Contents

Introduction

This report details a utility agent based approach for trying to win the classic game Pacman. Pacman exists in a maze-like, fully observable world which contains food, ghosts and walls and to win the game, Pacman must eat all of the food whilst avoid being eaten by ghosts. However, in this version of the game, the environment is non-deterministic and any action that Pacman decides to take has a stochastic outcome with an associated probability of success (achieving the desired move) and failure (an unintended move). For this reason, the agent uses a Markov Decision Process approach, MDP, to navigate the game. In particular, a value iteration algorithm is implemented to solve the Bellman equation, calculate utilities and find the optimum policy to guide Pacman to eat all of the food and avoid harm from ghosts. This report finds that Pacman wins with an average win rate of 65% and 48% on the small and medium layouts respectively.

The report is split into four main sections. The first section, the methodology, explains the concepts behind a MDP and the value iteration algorithm. The second section outlines the agents strategy, behaviour and particular implementation of the MDP. The third section presents a number of tests and analysis to support certain decisions made about parameter values and provide a comparison of the MDP performance vs a search and random strategy in a non-deterministic environment. The final section discusses the shortcomings of our methodology and areas for improvement.

1 Methodology

In a deterministic world, a goal state can be reached through a sequence of actions with certainty and a searching algorithm can be used to find a route to a desired goal. Unfortunately, most instances of the world are uncertain and stochastic - this type of environment is defined as non-deterministic. This section introduces the MDP technique for dealing with non-deterministic environments.

1.1 Markov Decision Process

MDPs are non-deterministic search problems. In a deterministic world, one can achieve it's goal with certainty by following the fixed set of actions that take it to the desired state. Unfortunately, with non-determinism and stochastic motion the environment does not always permit a solution to be executed as planned, often leading to unintended states. MDP's outline a framework to make sequential decisions in an uncertain world, formally MDPs consist of the following.

1. $s \in S$: a set of available states in the environment. The agent will belong to one state at each time step.
2. $a \in A$: a set of actions where action, a, takes the agent from state ,s, to state s'.
3. $P(s'|s, a)$: a transition model, that describes the outcome and probability of each action in each state.
4. $U(s)$: utility function describes the utility for each state.
5. $R(s)$: reward function that determines rewards an agent receives for being in certain states.

It is worth noting, the transition model is Markovian, and therefore the probability of reaching state, s', only depends on the current state, s, and not on any prior history. Given this definition, the dependency of the transition function can be reduced from a sequence of past states to just the current state, as shown in equations 1 and 2.

$$P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, ...S_0 = s_0) \tag{1}$$

$$P(S_{t+1} = s'|S_t = s_t, A_t = a_t) \tag{2}$$

1.1.1 Utility

Utility is the measure of the desirability of a state and in this game the agent is dealing with sequential decisions that determine the utility of any given state, that is, the utility is the sum of discounted rewards over this future sequence. For the agent to be rational, it should always strife to maximize its utility [Russell and Norvig, 2016].

1.1.2 Policy

Since the transition function is stochastic in nature with well defined probabilities of outcomes, an agent may not end up at it's goal state given a fixed set of sequential actions. Therefore the goal of an MDP framework is to prescribe what action an agent should do for any of the states which it may reach in the future. The policy lays out a clear action at each state, if the agent ends up in a state it wasn't expecting, it has a clear instruction of what action it should take [Russell and Norvig, 2016]. Furthermore, the agent will behave optimally if a optimal policy is taken, this is a policy that maximizes expected utility (MEU), U*(s), as defined by equation 3.

$$U*(s) =_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{3}$$

1.1.3 The Bellman Equation

Here the Bellman equation is presented for calculating utilities. As mentioned previously, the utility of a state is the expected sum of discounted rewards received by the agent from future states. This definition leads to the bellman equation which states that there is a direct relationship between the utility of a state and the utility of its neighbours. The Bellman equation, as defined in equation 4, defines the utility of a state as the reward received for being in that state plus the MEU of its neighbours, assuming a rational agent would pick this state, discounted by some constant decay factor, gamma. Gamma acts as a weighting to states according to their distance from the start state [Bellman, 1957].

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{4}$$

1.1.4 Value Iteration

The value iteration algorithm is a method for computing the utility of states and solving a MDP computationally. The Bellman equations are non-linear which makes solving them a complex task to solve algebraically, value iteration uses an iterative approach to find a solution to the equation. Firstly, the utilities are initialized to an arbitrary value, using this set of utilities a new set is calculated under the bellman equation and this update procedure continues until convergence of the utilities is reached [Russell and Norvig, 2016]. Once convergence is reached, an optimal policy can be deduced from the utilities using the MEU. This updated algorithm is shown in equation 5.

$$U(s)_{i+1} \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{5}$$

2 Agent Strategy and MDP Implementation

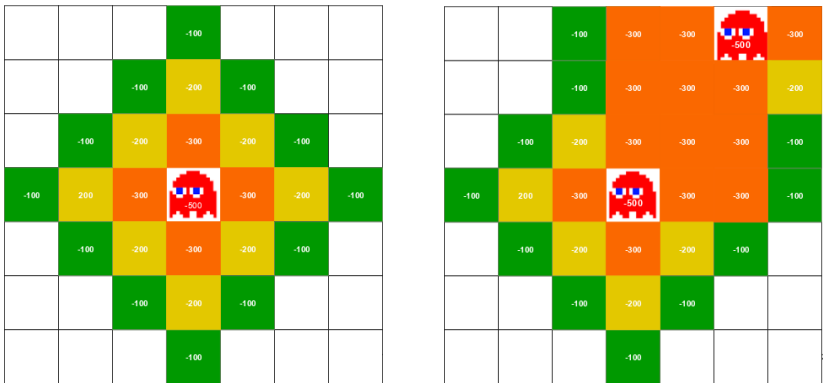
The MDP agent defined in this program implements the above ideas to try to win Pacman. The implementation consists of four main classes that Pacman uses, the main class, the MDP agent, makes use of the other two to try to understand it's world and form a policy for moving. Each of these four classes is discussed below.

2.1 MDPAgent Class

The MDP agent controls the agents decisions. There are three main steps that the agent executes. Firstly, the world is fully observable, so to understand it's environment, it calls the PacWorld() class to build a map of the world, Pacman uses this information to design its moving strategy. Secondly, it calls the MDPSolver() class, passing it a list of the nodes in the map to form the states in the MDP. Finally, it obtains the optimal policy for Pacman by getting the MEU move from the MDPSolver() based on the utilities calculated in the previous step. It then passes this move to the api.makeMove() method. Following the motion model set up in this game, Pacman may not make the intended move. Despite this, Pacman will rerun the same decision making process on the next move, finding the optimal move for this new state from the new optimal policy.

2.1.1 Ghost Effect

The agent uses the MEU principle to build an optimal policy and decide which neighbour state to move to at each time step in the game. The utilities calculated with the value iteration method follow the Bellman equation defined in the previous section. Each states utility is therefore a function of its surrounding states and reward. The reward function sets the reward values to either incentivise or disincentivise states according to their status. However, many instances may arise where Pacman has a choice between an open area or an area containing food and ghosts. Since the MEU is used in the Bellman equation, the latter case will result in a higher utility than the former as the food will be weighted heavily in the Bellman calculation. Pacman will head towards this dangerous area in search of the food states, ignoring the danger posed by ghosts. To mitigate this problem, the agent instigates a ghost effect method using the ghost information held in the map. The agent builds a ghost safety zone around the ghosts and marks these locations as ghost danger squares. The MDP reward function sets the reward values incrementally larger than -500 according to the number of steps from the ghost. Figure 1 below demonstrate this stepped increase in reward with a ghost zone radius of 3 steps. As figure 1 shows, the reward is -500 at the ghost location but diminishes according to the number of steps away from the ghost.



(a) safety area around a single ghost (b) safety are around two ghosts

Figure 1: (a) shows how rewards are set for the ghost zone. (b) when two ghosts are close by, there is an additive effect, indicating squares that are close to both are more dangerous.

2.2 PacWorld Class

The environment in which Pacman exists is fully observable and locations of food, walls and ghosts is available at each move step of the game. Pacman takes advantage of this transparency by instantiating the PacWorld Class and building a map. The map is formed of nodes that contain food, walls, corners, ghosts and Pacman. The set of nodes that don't contain walls are used directly as the states for the MDP. To help speed up Pacman, a function to reduce the size of the map is used. It builds the boundaries of the map by setting the edges slightly larger than the minimum and maximum location of either food, ghosts or Pacman. This, at times, reduces the number of states used in value iteration.

2.3 Node Class

Each node within the map is an object belonging to the node class. A node has the following attributes

1. x: x coordinate of the node
2. y: y coordinate of the node
3. status: indicates node type, with the following codes
    - w: wall node
    - f: food node
    - g: ghost node
    - o: open node (no ghosts or food)
    - gz: node within ghost safety zone
4. utility: utility value of the node
5. reward: reward value. Defined by reward function and dependant on status.

2.4 MDPSolver Class

MDPSolver class defines the implementation of the MDP and value iteration algorithm. The agent uses a MDP because Pacman is operating in a non-deterministic environment. The uncertainty leads the agent to try and find an optimum policy based on the MEU for each of the states in the world. By adjusting certain rewards associated with different states, the utilities and subsequent policy can be formed to guide Pacman away from ghosts and towards food. The MDP class has a transition function, reward function, set of states, set of actions and a decay factor implemented as attributes. Below is an outline of how each part of the MDP is implemented in this program. Testing and analysis of the various parameters is presented later to support the choice of value.

2.4.1 Actions

The set of actions Pacman can take consists of North, South, East and West. Each action takes Pacman to one of the neighbour states, for this reason we consider each action when calculating the utility of the current state. A move may be illegal due to walls, in this instance the probability is multiplied by the utility of the current state.

2.4.2 States

Each state in the world represents a physical node of the map and has a corresponding status, utility and reward. The Pacman world does not contain any terminal states corresponding to physical nodes of the map, instead, the only terminal state represents a map where all the food is eaten. For this reason, value iteration is performed on every node of the map that is not a wall. At each move, a new map is created reflecting any changes to the underlying world such as ghosts at new locations or food removed from a space where Pacman has been.

2.4.3 Transition Function

The transition function defines the stochastic motion model. In this model, the transition function defines an 80% chance of Pacman reaching the target state, s, if action, a, is taken and a 20% chance of ending up in a state perpendicular to the target. This model is encapsulated in the class transitionModel attribute.

2.4.4 Rewards

The reward of a state is dependant upon the node status. The reward function is constructed to incentivise food, penalize open spaces and disincentivise ghosts. The reward function sets the following schema for rewards:

1. Food: +10
2. Open: -1
3. Ghost: -500
4. Ghost Area: The following are set as rewards for number of steps from ghosts
    - 4: -50
    - 3: -100
    - 2: -200
    - 1: -300

2.4.5 Decay Function

The utilities in equation 2 are discounted by the decay factor gamma. This constant sets how far in the future the agent can see. A lower gamma gives states further away less significance in the model, causing the agent to prefers those states that are closer. On the contrary, a higher gamma gives states further away a higher weight in the utility calculation. As demonstrated in the testing section, a high gamma leads to a more successful win rate, reasons for specific gamma will be discussed later.

2.4.6 Optimal Policy

Once the MDP solver has run value iteration and the utilities have converged, the MDP object is passed back to the getAction() function. From here Pacman calls the getMEU() function within the MDPSolver to get the optimum policy based on the MEU for the state Pacman is in. The action corresponding to the optimal policy is then passed to the api.makeMove() function and this is the move that Pacman makes.

3 Testing

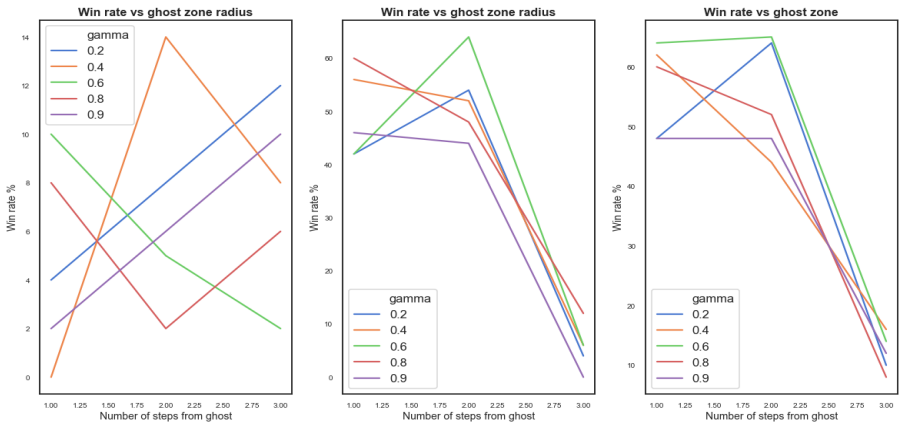
There are two parts to the testing regime, each part is split further by layout. In the first, the tests are focused on optimization of parameters of the MDP. The second, demonstrate the performance of a MDP compared to other strategies. For the purposes of statistical significance, each test consists of twenty independent trials of ten games.

3.1 Parameters

3.1.1 Small Grid

i) **Win rate vs ghost safety zone radius for each gamma**

This test compares the win rate vs the ghost zone radius across different values of gamma. Figure 2 plot (a) shows that there is no clear relationship between win rate, distance and gamma values at a low number of iterations. This is because at 5 iterations, the utilities have not converged enough. Plot (b) and (c) show that at 10 iterations or more there is a clear relationship between ghost zone radius and win rate for all gamma discount factors. A radius of 1 and 2 is successful at achieving a win rate of between 50-60% but at 3 and beyond the win rate drops significantly. Since the small grid contains much fewer states to iterate over than the medium classic, if the radius is bigger than two the zone is too big and it floods too many states with low ghost rewards, overwriting the food utility and so it is never attractive for Pacman. Pacman becomes trapped in his own ghost zone and eventually the ghost will catch Pacman. Plot (b) and (c) also show that a gamma value is not as effective in the small grid. This is discussed further in the next test.

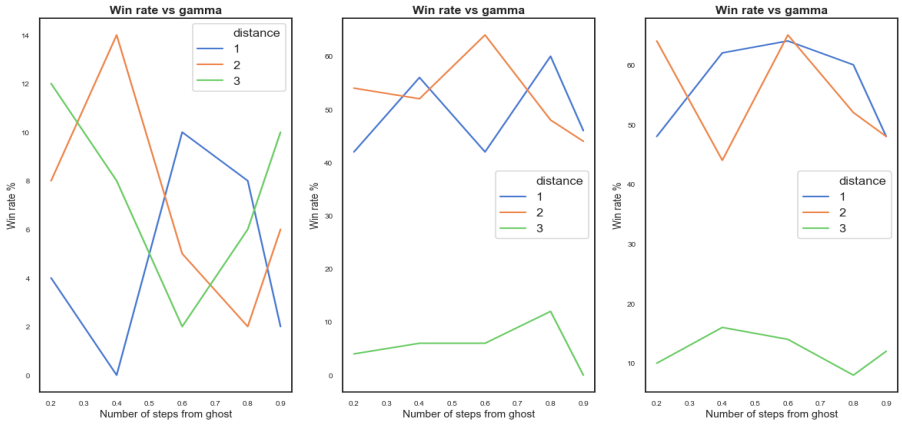


(a) 5 iterations (b) 10 iterations (c) 20 iterations

Figure 2: Win rate vs ghost radius for each gamma discount factor

ii) **Win rate vs gamma discount factor for each ghost radius**

Figure 3 shows that gamma has little effect on a small grid and remains fairly constant. Since gamma is a discounting factor that penalizes states further away, it is expected that gamma will have less influence differentiating utilities of states in the bellman equation. These plots further illustrate that distance is the overriding factor on the small grid for win rate.

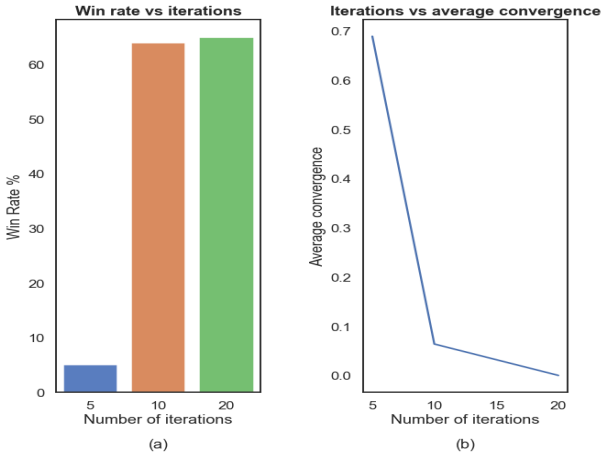


(a) 5 iterations (b) 10 iterations (c) 20 iterations

Figure 3: win vs gamma discount factor for different ghost zone radius

iii) **Win rate vs number of iterations**

This test aims to explore an optimal convergence tolerance factor given win rate and the time constraints on Pacman. Figure 4 shows that win rate is low at a small number of runs, corresponding to low convergence between state utilities. Once a small convergence factor has been reached the win rate plateaus as suggested by plot (a). Similarly, in plot b the convergence factor decreases significantly from 5 to 10 iterations but after this, it plateaus as the number of iterations increases. Given this, the convergence rate has been set at 0.1 which is around 14 runs.



(a) (b)

Figure 4: Plot (a) Number of iterations vs win rate. Plot (b) The average convergence tolerance limit vs number of iterations

3.1.2 Medium Classic

i) **Win rate**

This test compares the win rate vs the ghost zone radius and the win rate vs the gamma decay value. Figure 5 plot (a) shows that a higher gamma value leads to a higher win rate. At a low gamma, Pacman is less concerned with states in the future, as reflected in the small utilities, and he has low foresight. Consequently, he only becomes aware of ghosts and danger late on, leading to more situations where he heads towards ghosts and thus collides with them. At a gamma of 0.8, the win rate starts to plateau. Figure 5 plot (b) shows that a radius of one for the ghost zone leads to a poor win rate but at a radius of 2-4 steps, the win rate is high and stays roughly the same between 40-50%.

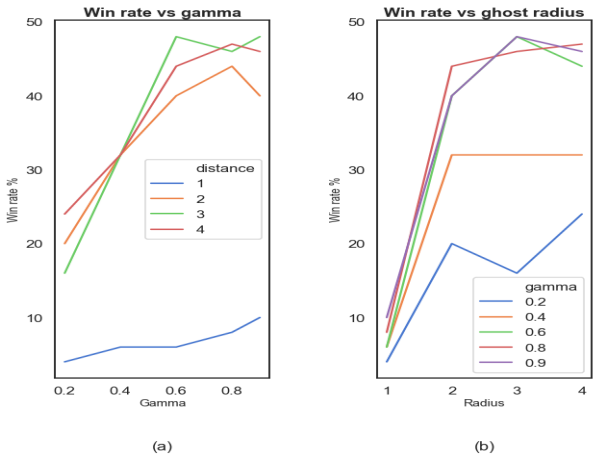


Figure 5: ((a) Win rate vs gamma (b) Win rate vs ghost zone radius

ii) **Win rate vs number of iterations**

This test looks at the average convergence tolerance vs number of iterations and number of iterations vs win rate in order to determine a reasonable convergence tolerance in the value iteration step. As can be seen in figure 6, a increase in the number of iterations from 20 upwards does increase the win rate but by a small amount.

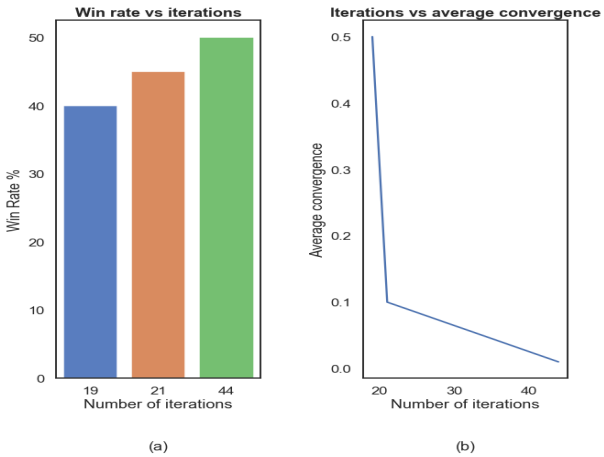


Figure 6: Plot (a) Number of iterations vs win rate. Plot (b) The average convergence tolerance limit vs number of iterations

In plot (b), the convergence tolerance decreases significantly for a small number of iteration steps, around 20, but the the rate of reduction decreases significantly as the iterations get larger. Given these small variations in win rate and convergence tolerance and the time constraints on Pacman to win. A convergence tolerance of 0.2 is chosen for the value iteration.

3.2 Strategy

This test aims to demonstrate the effectiveness of Markov Decision Processes in a non-deterministic world. As can be seen in 7, our value iteration algorithm significantly outperforms both a random agent strategy and a strategy based on an A* food search and ghost survival mechanism. The latter achieved a win rate of ca. 87% in the equivalent deterministic medium classic Pacman world [Verghese,]. Here the A* strategy only reaches around 16% and random strategy fails to win. This demonstrates that a Markov Decision Process computed with a value iteration algorithm is a successful approach to winning Pacman in an uncertain environment.

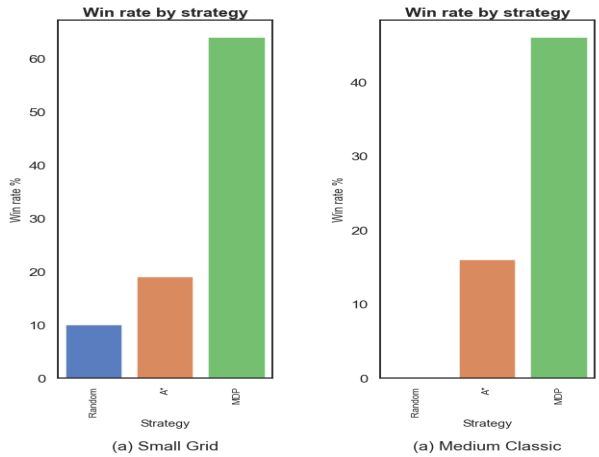


Figure 7: (a) Win rate vs strategy on small layout (b) Win rate vs strategy on medium layout

4 Further Work

The required win rate is achieved with the chosen parameters for both layouts. However, future work could explore how to get a higher score by testing the parameters over a larger range of values and to optimize the model. It should also be noted that whilst the time taken to achieve two wins in the medium layout is within the stated time limit, it is slower than desired. Future work could look at redundancy in the implementation, model parameters and other possible strategies for reducing the time. One such strategy, could be a map reduction method that reduces the map used in value iteration based on where the ghost, food and Pacman states are. Work would need to be done on how this affects the win rate. Finally, other more creative strategies could be explored to achieve an optimal score. One such strategy might be based on an A* search algorithm, where a shortest route to food is determined based on some heuristic. The states that compose the route could be rewarded by the reward function in such a way so as to incentivise Pacman to follow them.

References

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684.

[Russell and Norvig, 2016] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[Verghese,] Verghese, G. An a* agent approach to winning pacman in a deterministic world.