6CCS3AIN Coursework Report

1 Introduction

For this coursework I implement the Modified Policy Iteration algorithm as my MDP solver. My reward function considers food, ghosts (position, state, direction), deadends and capsules.

2 MDP Solver

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Figure 1: Formula for approximate policy evaluation stage. (From lecture notes)

$$\pi_{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s, a) U_i(s')$$

Figure 2: Formula for policy improvement stage. (From lecture notes)

First the approximate policy evaluation stage is repeated a fixed number of times to calculate utility values for each state, followed by a policy improvement step to find the new optimal policy for each state. This whole process reiterates until the policy map converges.

2.1 Reward Function

The reward function can be summarised as follows:

$$R(s) = \begin{cases} G(s), & \text{if } s\ is\ a\ ghost\ state \\ Capsule\ constant, & \text{else if } s\ is\ a\ capsule\ state \\ Food\ constant, & \text{else if } s\ is\ a\ food\ state \\ Empty\ constant, & \text{else} \end{cases}$$

In the initialization of the agent I select a set of values to calculate rewards according to map size, as this was said to be permitted.

2.2 Hiding From Ghosts

My algorithm uses a *danger zone,* where nearby reachable states for a ghost receive a negative reward. I implement a depth-limited, breadth-first-search to find these states.

I've noticed that ghosts only backtrack when they have no other choice, so I can prune certain branches by considering the ghost's current direction which I calculate from retaining a ghost's previous location and comparing it to the current. The time complexity in theory for BFS is $O(|V| + |E|)$, V = all reachable states, E = all possible moves. I expect considerably lower due to the depth limit only traversing certain moves.

The new reward value for a ghost state is calculated as follows, where the magnitude is proportional to the distance from the nearest ghost:

$$G(s) = -\frac{Ghost\ reward\ constant}{Distance\ from\ ghost + 1}$$

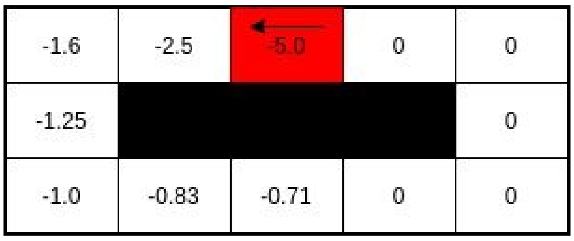


Figure 3: An example of generated reward values radiating from the ghost state in red is facing west.

2.3 Dead Ends

A situation I encountered when testing on Small Grid is where Pacman moves into a dead end for nearby food and is guaranteed death if a nearby ghost follows. To avoid this, in my reward function I give a large negative value to any deadends within a ghost's danger zone.

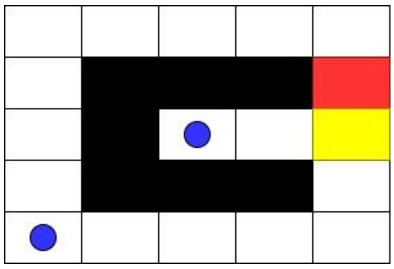


Figure 4: Example of the deadend situation, where if pacman (yellow) moves left for the food he risks dying if the ghost (red) follows.

2.4 Edible Ghosts

States with capsules are given a positive reward value. Edible ghosts don't produce a danger zone, instead they are given positive rewards to encourage Pacman to eat them. There is a time buffer so pacman doesn't risk chasing ghosts which are close to returning to normal state. The G(s) ghost reward function becomes:

$$G(s) = \begin{cases} Edible\ Constant, & \text{if s contains weak ghost} \\ -\frac{Ghost\ reward\ constant}{Distance\ from\ ghost + 1}, & \text{else} \end{cases}$$

When deciding whether to chase a ghost I check if it meets a minimum time remaining on its edible state, so that pacman avoids risk in chasing ghosts.

3 Statistical Analysis

All data was produced on a 3GHz CPU with 8GB of RAM, over 100 game runs unless stated.

3.1 Base Performance

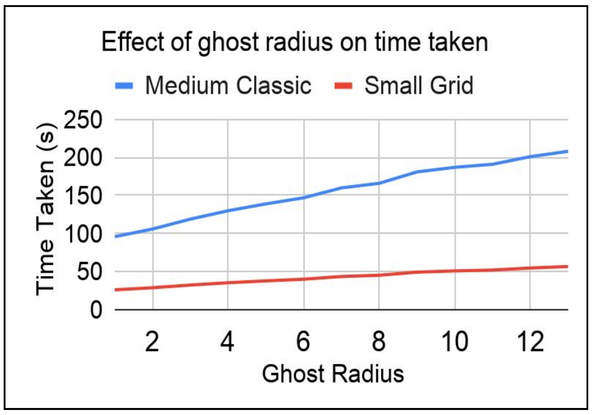
As a starting point I ran both maps without considering ghosts or deadends. From here I progressively add features and tune values.

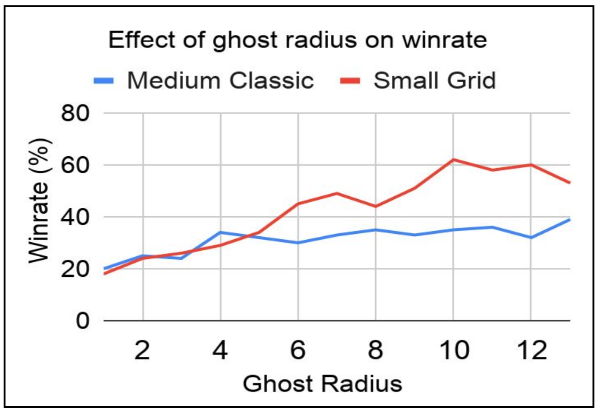
Medium Winrate (%)	Medium Time Taken (s)	Small Winrate (%)	Small Time Taken (s)
19	97.3	22	43.86

Table 1: base win rate and time taken on both maps.

3.2 Ghost Danger Zone

To find an effective ghost radius I began with an initial ghost value of -10. Time taken was a potential concern due to the BFS.

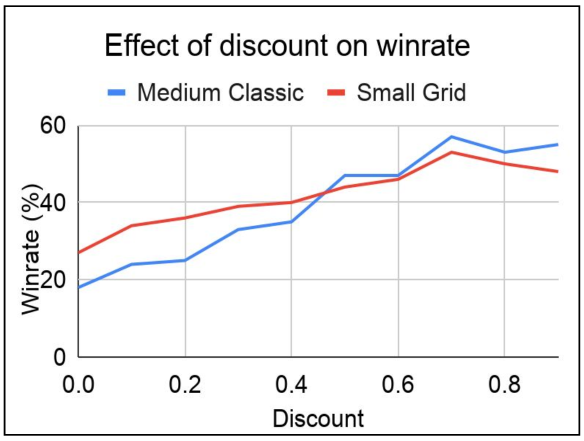


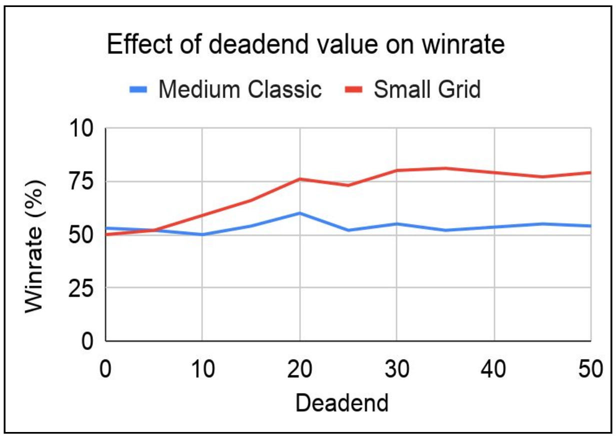


There is a clear increase in performance for both, however the win rate appears to plateau for medium classic after approximately a radius of 5 whilst plateauing around 9 for small grid, which seem like the optimal values. The win rate seems noisy which is expected due to nondeterminism. The time taken increases significantly with the ghost radius but large values still perform well within limits.

3.3 Deadend and Discount Value Tuning

The discount value is key as it decides how pacman should compare long term vs short term benefit when deciding a move with a value of 0. Pacman would only consider the adjacent states.

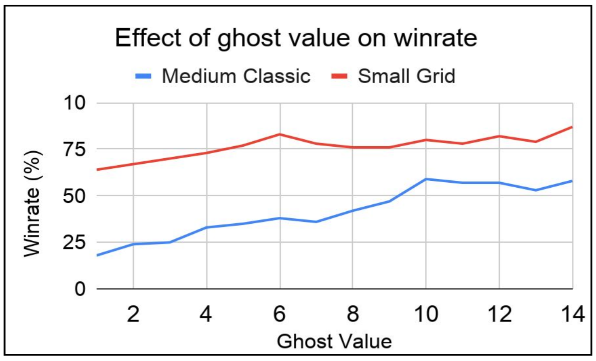


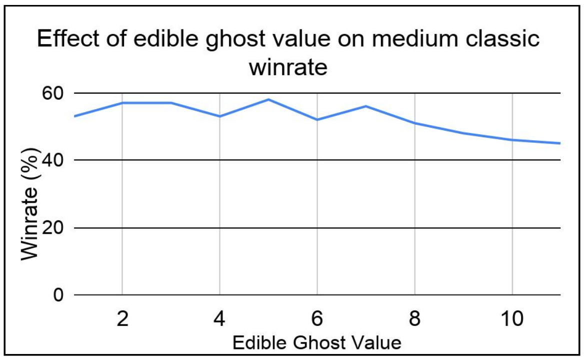


For discount, the win rate seems to increase until a plateau around discount values of 0.75 - 0.9 on both maps. Introducing deadends shows an improvement on Small Grid up to values of 30, but not Medium Grid which is expected as pacman would rarely enter those locations.

3.4 Ghost Reward Value Value Tuning

I have noticed that higher edible ghost values are forcing pacman into the ghost spawning location after eating a ghost causing a drop in winrate.

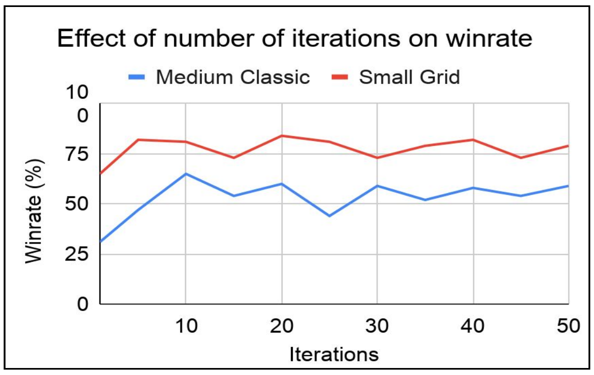


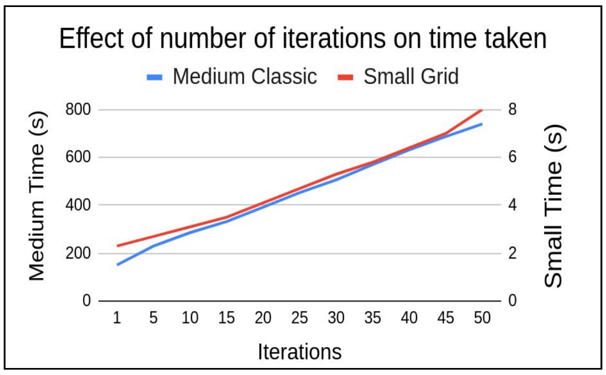


The win rate seems to plateau after a certain threshold for each ghost reward value, whilst winrate decreases for high edible ghost values.

3.5 Iterations

The iteration value affects how utility values can properly "propagate" to far regions of the map, but with diminishing returns and more iterations increase computation time.

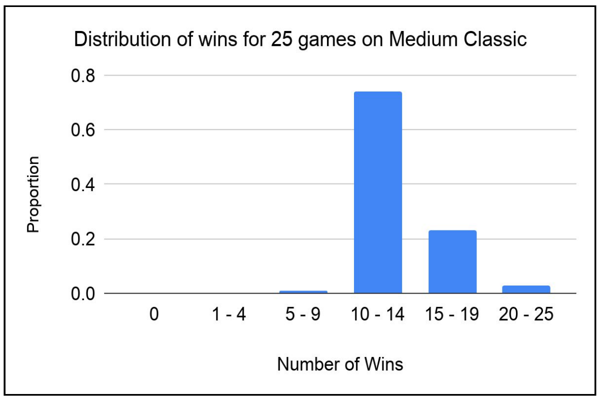




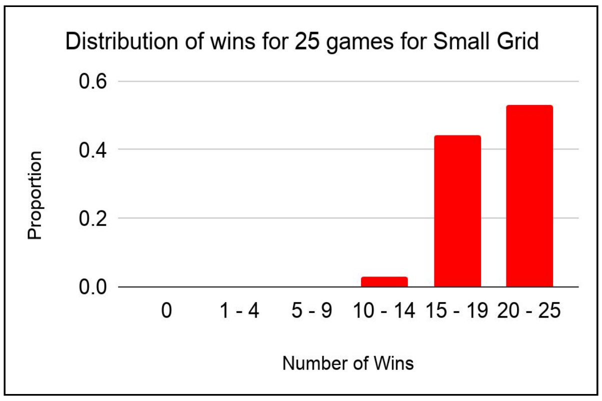
After 10 iterations there seems to be no significant performance gain.

3.6 Variance

Lastly, to see the variance of my algorithm for the 25 games runs that it will be tested on, I ran 40 sets of 25 games for each map and the distribution of win categories for each map.



standard deviation: 1.98936, range: 8 to 24 wins



standard deviation: 1.93252, range: 13 to 25 wins

4 Conclusion

I feel that my algorithm gives satisfactory results for this coursework on average; 78% on Small Grid and 58% on Medium Classic whilst completing well within time limits. A worry is that due to the non-deterministic nature of the game my performance varies between win bands significantly on Small Map. I see unexplored possibilities to improve performance on the larger maps, such as scaling food values depending on the amount of remaining food, and avoiding corridors that could lead to being trapped by ghosts. There is also more room for experimentation with ghost chasing.