Introduction

The task is to design and implement an MPD solver to work in UC Berkeley's pacman environment. For my MDP solver, I selected value iteration to implement.

Method

Data structures

Mapping

As the 3 main dictionaries utilised have complex mappings, they are listed in a figure 1.

{} : symbolise a dictionary data structure

Identifier	Data structure	Representation
self.rewardDict	{tuple -> int}	{(x,y) -> reward}
self.stateDict	{tuple -> {str -> [tuple]}}	{(x,y) -> {direction -> [(x1,y1),(x2,y2),(x3,y3)]}}
self.utilDict	{tuple -> float}	{(x,y) -> utility}

Fig. 1. Data structures used for rewards, states and utilities

Constant variables are capitalised and initialized as global variables to allow parameter tuning to be easier and centralised in one place.

Grid Population

As the MDPAgent program initialises, a set variable map is populated according to the current grid's maximum height and width, calculated from the populateGrid() function.

Set Key Parameters

Within the function registerInitialState() , depending on the layout, the self.discountFactor and self.ghostBuffer values are different. The self.discountFactor represents the gamma function and the self.ghostBuffer represents the radius around the ghost.

Pacman's Action

Everytime the pacman moves, these functions are carried out beforehand.

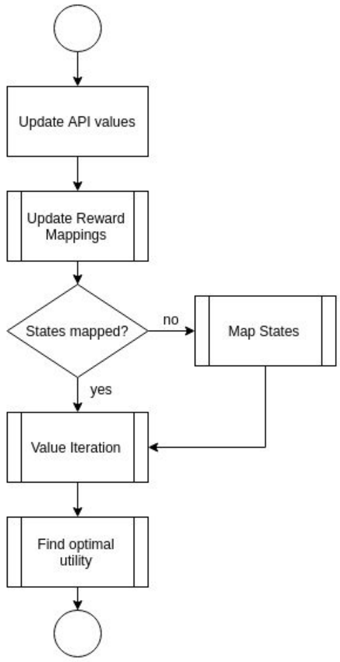


Fig. 2. Flowchart of MDPAgent.getAction()

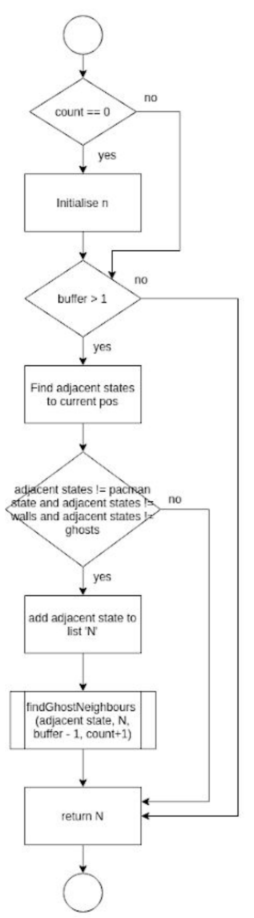
API Value Update

The pacman's, the food's, the capsules' and the ghost states' are stored in variables and each time the pacman moves, these variables must be updated. This is to decrease the amount of state parameters used for functions.

Reward Mapping

The reward dictionary self.rewardDict is initialised with every state's reward as -1. The reward dictionary is also updated with the rewards for the locations containing food and capsules. Each reachable corner in the self.map are also given a reward value to give pacman a direction to work towards.
Every ghost within the board is then found and assigned reward depending on whether they are aggressive or edible. The radius around the ghosts are also given a reward based on the nature of the ghost calculated in findGhostNeighbours().

mapGhostNeighbourRewards() then assigns each neighbour a reward value. The ghost's proximity has an attracting or repelling effect on the pacman's movements.



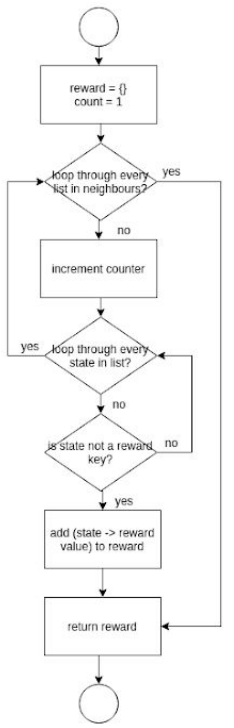


Fig. 3. Flowchart of MDPAgent.findGhostNeighbours()	*Fig. 4. Flowchart of mapGhostNeighbourReward()*

State Mapping

Every state that isn't a wall is mapped to possible future states when an action is executed.

Value Iteration

The value iteration for the MDPAgent is done within the function valueIteration().

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

For the implementation of the Bellman equation, the program iterates through the utility dictionary's keys which represents a square within the map that is not a wall and calculates the utility of each action to the next state taken from that square. Then the maximum of those utilities will be multiplied by the gamma function and the result will have the reward of the current state added onto it.

After every state in the utility has been iterated through, we find out whether the values of utility have changed significantly from the previous iteration through comparing it with the delta value - if delta is smaller than the stability variable, then return the utility dictionary. If not, then iterate through the state values again and reassign utilities.

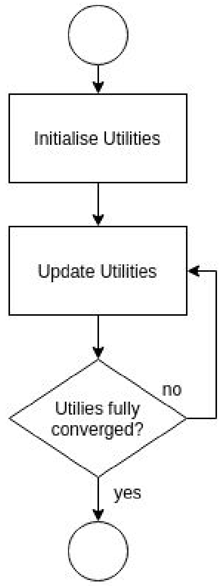


Fig. 5. Flowchart of MDPAgent.valueIteration()

Optimal Utility

The optimal utility is found through selecting the maximum expected utility out of pacman's adjacent state's utilities. In findNextMove() the program finds the optimal utility state and translates that into a Direction action. This is fed into the api.makeMove() and pacman then moves onto the new state and we move onto the next step.

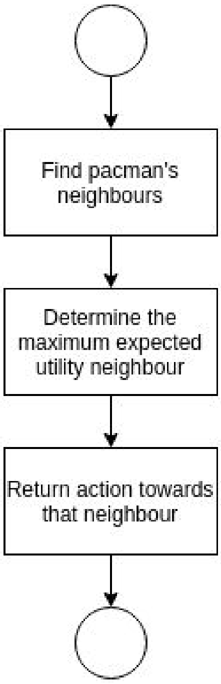


Fig. 6. Flowchart of MDPAgent.findNextMove()

Parameter Analysis

Fine tuning parameters are done to optimise win rate in the smallGrid and mediumClassic layout. The parameters tuned are the discount factor in the bellman's equation (1) and the ghost buffer which is the distance from a ghost to assign reward values. My strategy is to test several different settings available for the discount factor and ghost buffer on smallGrid and mediumGrid by running 2000 games. Then the parameter with the maximum average win rate for each layout will be picked as the best parameter.

The discount factor is a value ranging from 0 to 1. The closer the value is to 1, the further ahead the agent 'sees into the future', thereby increasing convergence time in value iteration. The best values for both layouts are in bold. My ghost buffer value while testing was 1 for smallGrid and 3 for mediumClassic.

Discount factor	Win rate (%)
0.0	1.30
0.1	46.00
0.2	59.65
0.3	64.40
0.4	64.20
0.5	66.40
0.6	65.50
0.7	**68.45**
0.8	68.25
0.9	67.50

Fig. 7. Win rate of discount factor values on smallgrid

Discount factor	Win rate (%)
0.0	0.20
0.1	44.10
0.2	45.65
0.3	52.50
0.4	54.75
0.5	55.05
0.6	56.10
0.7	58.25
0.8	57.45
0.9	**58.50**

Fig. 8. Win rate of discount factor values on mediumclassic

The ghost buffer parameter sets the distance away from the ghost a pacman would begin to move away from the ghost. I have chosen to test this parameter from 1 to 4 inclusive as any value above 5 would be unnecessary. The best values for both layouts are in bold. My discount factor while testing for smallGrid is 0.6 and 0.8 for mediumClassic.

Ghost buffer	Win rate (%)
1	66.70
2	**67.45**
3	62.15
4	51.90

Fig. 9. Win rate of ghost buffer values on smallgrid

Ghost buffer	Win rate (%)
1	40.15
2	55.10
3	**59.75**
4	53.40

Fig. 10. Win rate of ghost buffer values on mediumclassic

In conclusion, based on the tests conducted independently on the discount factor and ghost buffer, the optimal parameter settings for both smallGrid and mediumClassic layouts are specified in figure 11.

The optimum discount factor on mediumClassic is larger than on smallGrid may be because mediumClassic is a significantly larger environment for the pacman and therefore to eat all the food and survive, long term solutions are favoured to be successful. The ghost buffer is also larger on mediumClassic than on smallGrid as it benefits pacman more to be further away from hostile ghosts and eat edible ghosts to move the threat further away from their current position.

Parameter	smallGrid	mediumClassic
Discount factor	0.7	0.9
Ghost buffer	2	3

Fig. 11. optimum parameter values for smallgrid & mediumclassic layouts