# AIN Coursework MDP Solver

## Introduction

The Markovian Decision Process is a sequential decision problem for a fully observable, stochastic environment with the transition model and additive reward. In this coursework, the Markovian Decision Process method is applied to design an PACMAN that make moves in a non-deterministic environment, and see how many times it can win the games.

## Theoretical Base

## Bellman Equation

Bellman showed that a dynamic optimization problem in discrete time can be stated in a recursive, step-by-step form known as backward induction by writing down the relationship between the value function in one period and the value function in the next period. In Markovian Decision Process, a Bellman equation is a recursion for expected rewards. In the following Bellman equation, $R(s)$ represents the reward function, $\gamma$ represents the discount factor and $P(s'|s,a)$ indicates the transition model.

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

## Value Iteration

Value iteration is a dynamic programming algorithm that uses an iteratively longer time limit to compute time-limited values until convergence. It operates as follows:

1. $\forall s \in s$ , initialize $V_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence:

$$\forall s \in S, V_{k+1}(s) \leftarrow \max_a \sum_{s'} T\left(s, a, s'\right)\left[R\left(s, a, s'\right) + \gamma V_k\left(s'\right)\right]$$

## Strategy

The main strategy of the MDP agent is to assign positive rewards to the food, negative rewards to the ghosts, negative rewards to a blank space which is close to zero, and a large discount factor to the model. The first step to begin with is to build a map for storing the utility of each grid's coordinates, calculate the size of the map, and then add all the necessary staffs into the map. After setting all these parameters, I use the value iteration method to run the MDP agent based on the rules of Bellman equation. Then, try to generate the Maximum Expected Utility by Bellman equation, and determine the next move of the MDP agent based on MEU.

## Core code and function

**def registerInitialState(self, state):** This is a function gets run when I first invoke the pacman. This function gets run after an MDP agent object is created and once there is a game state to access.
**def final(self, state):** The final function gets run when each round of games ends. At the end of each round of game, the final function would clear the agent's parameters and reset the internal states.
**def mapSize(self, state):** This function combines the getLayoutHeight() function and getLayoutWidth() function, which calculate the right size of the map.
**def updateGhostToMap(self, state):** This is an important function in the MDPAgent class. The strategy I apply is that, if the ghosts is in the state of scare, just ignore them and keep finding food.
**def MaximumExpectedUtility(self, x, y, IteratedDirection):** This function calculate the maximun expected utility based on the four expected utility on each direction. And return the expected direction to decide next moves.
**def valueIteration(self):** This is the core algorithm of the MDP agent, it calculate the utility states in blank square. If the difference between the iterated map and the original map equals to zero, then the MDP process should be stopped.

## Analysis of the Experimental Data and conclusion

According to the instructions on the coursework introduction, I firstly control the pacman to run 25 games in a row in smallGrid and mediumClassic respectively, and the outcomes are shown below:

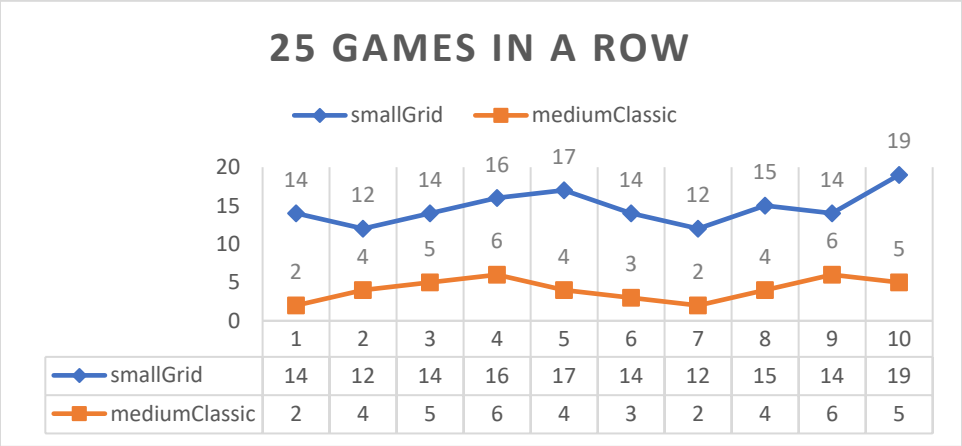


Figure 1. run 25 games in a row in smallGrid and mediumClassic

From the chart we can conclude that, the MDP agent performs much better in smallGrid than in mediumClassic if it runs 25 games in a row. In the 10 tests, the pacman could win up tp 19 times in 25 games, with the 58.8% average win rate in the smallGrid. However, in the mediumClassic, the pacman could only hold a 16.4% average win rate. These outcomes are quite reasonable, as the environment is non-deterministic, the pacman would face more complexity and uncertainty in the mediumClassic than in the smallGrid. The larger the map is , the harder for the pacman to win.

To further analyze the experimental data, I would control the pacman to run 50/100/200/500/800/1000 games in a row, respectively in smallGrid and mediumClassic. The data is as follows. And conclusion is obvious that, pacman gets a better performance in the smallGrid map, with average 61.6% win rate in the map. Another conclusion is that, the more games the pacman run in a row, the more stable performance the pacman will get.

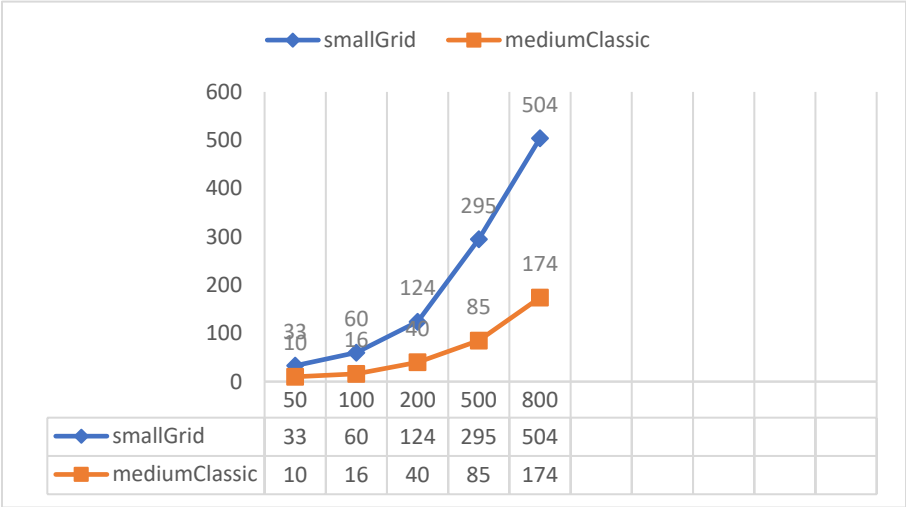


Figure2. the winning rates of pacman in smallGrid and mediumClassic