****

**CS401 – ARTIFICIAL INTELLIGENCE**

**“PROJECT REPORT”**

**PROJECT TITLE:**

**“Simulating Multi-Agent Planning using Pacman game”**

**GROUP MEMBERS:**

* Murad Popattia (K173722)
* Muhammad Samiullah (K173745)
* Muhammad Hassan Azhar (K173732)

**SECTION:** B

**INSTRUCTOR:** Dr. Fahad Sherwani

**DATE OF SUBMISSION:** 10-05-2020

**LANGUAGE USED:** Python 3.7.4

**Table of Contents**

Abstract 1

1.0 Introduction 1

2.0 Related Works 1

3.0 Features 2

4.0 Methods 2

4.1 Setting up GUI 2

4.2 Setting up Game Engine 2

4.3 Incorporating AI in pacman 2

4.3.1 Understanding the scenario 2

4.3.2 Can we make pacman play on its own? 3

4.3.3 Are reflex agents enough? 4

4.3.4 Adversarial Search Algorithms Used 5

4.4 Making the ghost ‘smart’ 6

5.0 Experimental Results 7

5.1 Maze: Contest Classic 7

5.2 Maze: Medium Classic 8

5.3 Maze: Minimax Classic 9

5.4 Maze: Open Classic……………………………………………………………………………………………10

5.5 Maze: Capsule Classic…………………………………………………………………………………………11

5.6 Maze: Small Classic…………………………………………………………………………………………...12

5.7 Maze: Test Classic…………………………………………………………………………………………….13

5.8 Maze: Trapped Classic………………………………………………………………………………………...14

5.9 Maze: Tricky Classic…………………………………………………………………………………………..15

5.10 Observations………………………………………………………………………………………………….16

6.0 Conclusions / Future Work……………………………………………………………………………………….18

7.0 Appendices………………………………………………………………………………………………………..19

8.0 Contributions……………………………………………………………………………………………………...23

9.0 References………………………………………………………………………………………………………...24

***Abstract:*** The game Pac-Man is a both challengeable and satisfactory video game that has been the focus of some important Artificial Intelligence research. The goal for using the game Pac-Man as a test bed in our experiment is that the Pac-Man game provides a sufficiently rich and challengeable platform for studying the AI in the computer game, and that it is simple enough to permit understanding of its characteristics. To achieve these desired goals we have used multiple informed search methodology like Minimax, ExpectiMax. Moreover we have used Alpha Beta pruning to reduce the search space. In addition to above, we have incorporated ghost agents to search smartly based on various variables such as distance from pacman, food locations etc. Finally, the experimental result from the game Pac-Man is presented to demonstrate the effectiveness and efficiency of these algorithms.

1. **Introduction**

This project was started to apply an array of AI techniques to playing Pac-Man. This project visualizes foundational AI algorithms namely informed state-space search. As Pacman is a zero sum game which means the pacman will maximize its score by eating all the coins in the maze meanwhile the ghost will try to minimize pacman’s score through interception. The algothrim uses adversial search tree to maximize pacman score. The algorithm in this project process the data provided by the game state. The game state is defined as the current position of the pac-man on the maze along with it’s neighboring coin and ghost position. After the game state has passed to the desired algorithm the result is used to increase the state-space search or move the agent if optimal solution is reached.

1. **Related works**

There are numerous projects that have in cooperated AI in games to make optimal computer player with the ability to produce 100% win rate. Some of games that have already in cooperated AI are listed below.

* **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
* **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
* **Go:** Human champions are now starting to be challenged by machines, though the best humans still beat the best machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.

Similarly the Pac-man game is also a good candidate for applying AI concepts. It can allow us to implement various search algorithms to produce high win rate for the Pac-man. The above game show similarity towards our project as they also consider future game states to determine their next move hence developing a search tree, which is key to our project in order to find the solution.

1. **Features**

Some application based features include:

* The GUI interface for selection Pac-man agent types, ghost agents and maze layout.
* The results of the game will be displayed on the terminal which is integrated into the GUI screen.
* Ability to run multiple instances of the game to generate cumulative result of the algorithm’s working.
* Algorithm visualization on pac-man game.
* Algorithm’s effectiveness shown through scoring based module.
* Ability to select the depth of search space.
* Ability to select number of ghost agents.
* Ability to toggle pac-man simulation.

1. **Methods**

**4.1 Setting up GUI:**

The GUI interface of the project is designed using a python GUI designing tool called pyqt5. Pyqt5 designer allows easy drag and drop GUI designing ability and also allows easy conversion of the user interface file into python code.

**4.2 Setting up game engine:**

The pacman game has been developed using the tkinter library in order to display all entities of the pacman world including coins, capsules, pacman, ghosts, mazes etc.

**4.3 Incorporating AI in pacman:**

**4.3.1 Understanding the scenario:**

Before establishing AI in pacman it was important to assess the environment type and the types of agents involved. The environmen analysis of the pacman can be shown below:

|  |  |  |
| --- | --- | --- |
| **Pacman Game** | | |
| **Observable** | Fully | Pacman knows all the aspects of environment, as state is maintained |
| **Deterministic** | Strategic | It is deterministic however, action of adversaries matter hence strategic |
| **Episodic** | Sequential | Each decision depends on the previous maintained state, hence sequential |
| **Static** | Dynamic | The environment is constantly changing i.e agents moving, coins reducing etc. |
| **Discrete** | Discrete | There are clearly defined percepts for the game |
| **Agents** | Multi-Agent | Multiple agents such as pacman, ghosts are involved |

Next, we defined goals for the Pacman that are:

* Eat all of the dots quickly.
* Get as many points as possible.
* Don't die

Some of the rational actions for pacman that would allow it to maximize its goals would include:

* Moving toward coins / food.
* Avoiding ghosts
* Moving toward capsules.

**4.3.2 Can we make pacman play in its own?**

The goal was to solve pacman i.e able to act rationally without user intervention. Firstly, the project involves pacman being a model-based reflex agent that suggests action based on the previous state. The pseudocode below highlights more about a model-based reflex agent.



In order for the agent to decide on an action based on the current game state, we use an evaluation function. An evaluation function is used to calculate the utility value to a game state. The evaluation function includes features of the game that provide evidence about the utility of the game state. These features are pre-determined based on expertise of the game.  The evaluation function used in this proejct involves the following features:

* Without any disturbance from the ghost, pacman would go towards the closest food location. In order to calculate the closest food, we have used breadth first search algorithm which always finds the shortest path first hence yields an optimal solution. Its space and time complexity is O(bd). As our search tree is depth limited, hence BFS was used.
* The implementation involves an exponential function for the ghost distance. As the ghosts get nearer to pacman, the utility decreases exponentially, hence pacman would avoid that. The more active ghosts are chasing pacman, the more it wants to eat the food
* Moreover, pacman would avoid a scared ghost, because eating it produces an active ghost in the center and lose the bonus points, so if there is a choice, it would tend to leave scared ghosts alone and take as much advantage of the scared time as possible.

**Evaluation Function:**

f(n) = [distance of closest food] + [distance from active ghost] + [distance from scared ghost]

**4.3.3 Are reflex-agents enough?**

As it is model based, hence the evaluation function takes in state-action pairs as input in order to evaluate actions from the current state. However, reflex agents choose an action based on the current perception of the world rather than planning or considering future consequences. This is a good rule in the short term for staying alive, but could lead pacman to be trapped in a corner.  Hence, we incorporate using a goal-based agent instead of reflex agent. Goal-based agents plan ahead by considering "what if" a certain action is taken. This needs a model including elements of the environment that are needed to measure success. For example, if the goal in pacman is to eat dots, then there needs to be a measure of how many dots are in the environment. If the goal is to score points, then points needs to be measured. Knowing how many points are scored for a given action, as compared to other actions, provides pacman with information for choosing an action.

Games with an adversary are generally what we think of when we think of games. In these games, there is another player, in our case ghosts, with motives alterior to ours and hence there is need for a search algorithm that determines the best state from a set of possible states considering both your goals and the actions of your adversary.

A typical search problem must include:

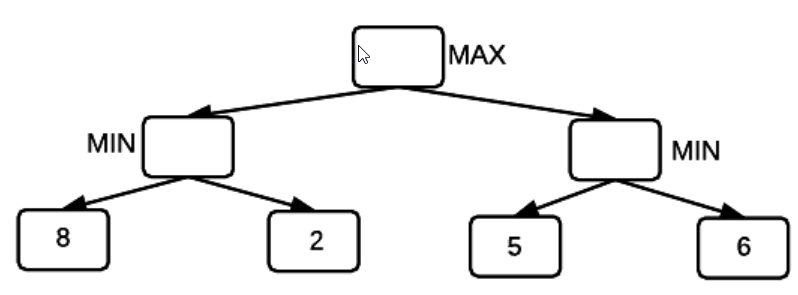
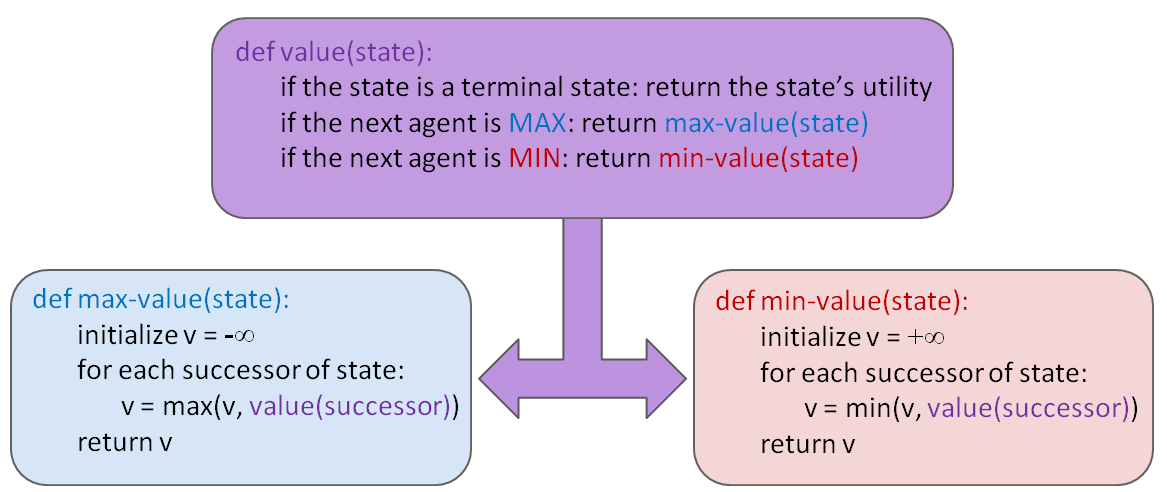
* States: S (start at s0)
* Players: P={1...N} (usually take turns)
* Actions: A (set of legal moves in state s)
* Transition Function: SxA → S (Successor functions that return a state based on a certain action)
* Terminal Test: S → {t,f} (States at the end of the game. Win or lose!)
* Terminal Utilities: SxP → R (defines the numeric value for game that ends in terminal state s for player p)

The adversial search tree used in this project consists of game states, where nodes in the tree are determined by actions in the game and specifics of the tree depend on the rules of the game. The solution that the algorithm provides maps states to actions. The project shows an extensive use of all the search problem variables during the agent design.

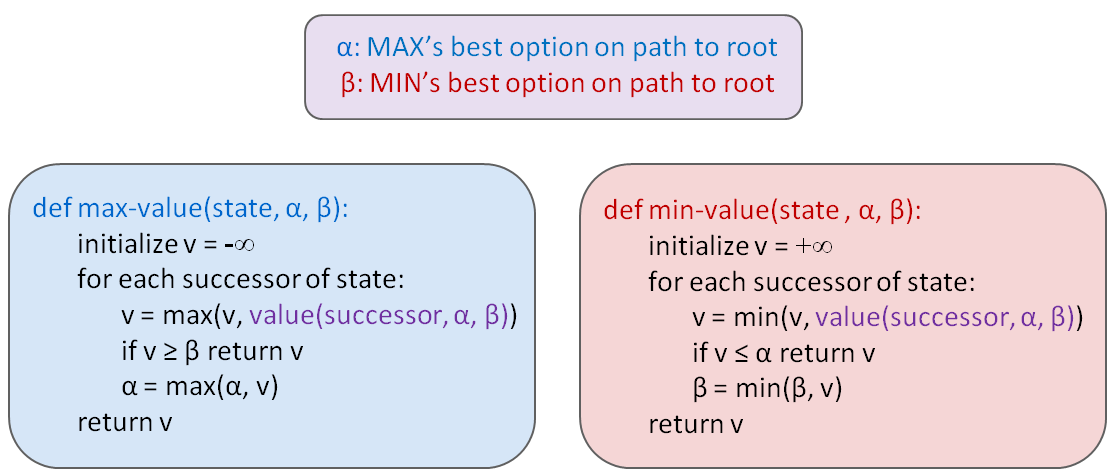
**4.3.4 Adversarial Search Algorithms used:**

We have used minimax, alpha-beta pruning and expectimax algorithms for evaluation of the game state.

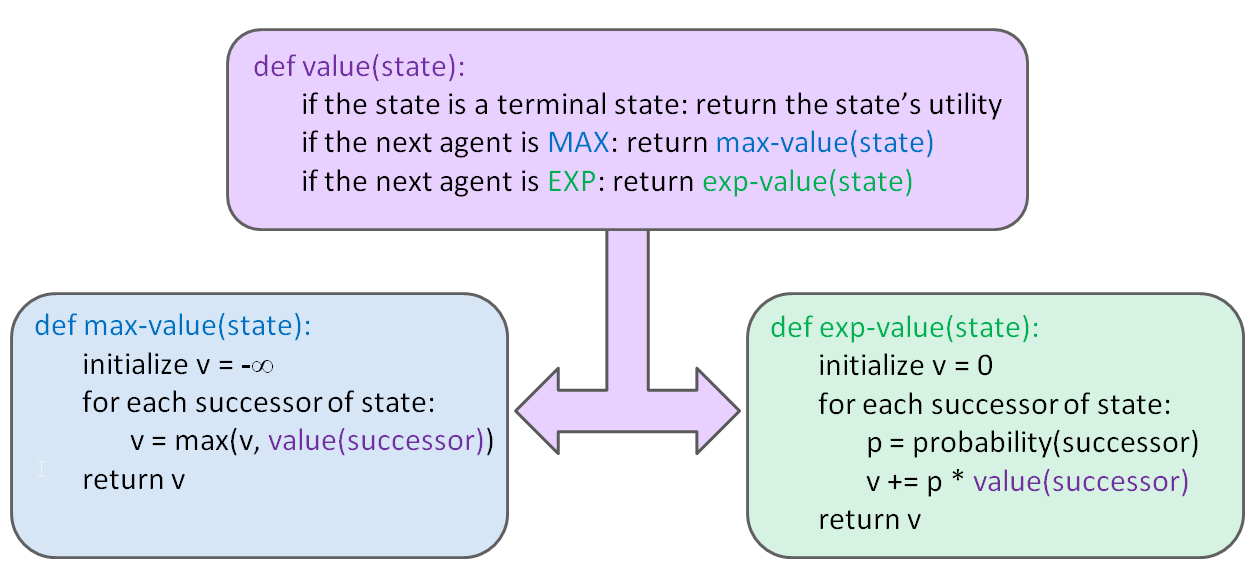
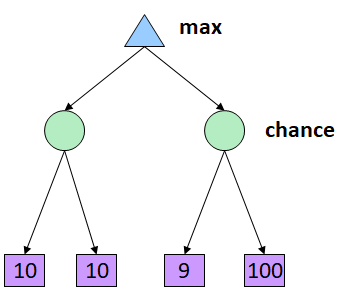
In minimax search, we start with a game tree of all possible futures. At the bottom of the tree, or the bottom level being explored, are the outcomes for the game and the utility for that outcome. The minimax algorithm requires a recursive traversal of the game tree. Starting at the root, MAX selects the action that returns the maximum of the minimum values of its children. The algorithm begins with a call to minimax(), with the node to evaluate, the current depth, and whether it's MAX or MIN making the decision as arguments. The diagram provides the pseudocode for the algorithm:



In any search tree, there are branches in the tree that cannot possibly contain the optimal strategy for the game, and it is possible to detect these branches and prune them so that time is not wasted on searching them. Pruning useless branches from the search tree makes minimax much faster, in some cases, exponentially faster. The alpha-beta algorithm finds the optimal minimax solution while avoiding subtrees that won't contain the optimal solution. Alpha-beta pruning sets two bounds on values observed during computation. The diagram below provides the pseudocode for the algorithm:



In some cases, there is a chance the adversaries(minimizer) might not be optimal hence the maximizer can take advantage of it instead of going for optimal solution, and might end up having a higher utility. The expectimax algorithm, a variation of the [minimax](https://en.wikipedia.org/wiki/Minimax) algorithm, uses the same approach. The chance nodes involved here take the average of all available utilities giving us the ‘expected utility’. The diagram below provides the pseudocode for the algorithm:

The problem is that in realistic we cannot search the entire game space tree as it is computationally intensive. Hence the solution is a depth limited search. Instead, we search only to a limited depth in the tree and replace terminal utilities with an evaluation function for non-terminal positions. This can help us to save the computational cost of evaluating all possible states. However, evaluation functions are always imperfect. The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters. This is an important example of the tradeoff between complexity of features and complexity of computation.

In this project, the search is limited to a fixed depth, and once that depth is reached, the path that leads to the highest evaluation value is selected. A deeper search will yield better results, but will also require more time. The evaluation function used here for the goal-based agents evaluates states rather than actions, as we were for the reflex agent. Instead of evaluating successor states for only the pacman, we evaluate successor state for all the agents involved. Moreover, unlike reflex agents, we are evaluating the states

**4.4 Making the ghosts ‘smart’:**

We have used two types of adversaries in the project i.e random and smart. The random adversaries or ‘ghosts’ used, chooses an action based on a random distribution. However, we have made a smart version of the ghost that chooses its action based on the distance from pacman. In case of scared, it flees to the position with the maximum distance from the pacman, while in active mode, it will proceed to the position having the minimum distance from the pacman. Moreover, for effectiveness, we have incorporated a probability with of attacking and fleeing into the ghosts, so that it selects its action based on its probabilistic nature.

**5.0 Experiments/Results/ Discussion**

When designing a rational/intelligent agent, we keep in mind PEAS analysis. PEAS analysis of the pacman game is as follows:

**Agent 🡪Pacman :**

* **Performance measure:** average score, win rate, computation time
* **Environment:** maze containing coins, power capsules, ghosts
* **Sensors:** screen display
* **Actuators:** arrow keys(in case of user)

This section highlights the experimental results and insights gained from them. The experimental setup includes running a 10-fold iteration 10 times on each agent (5 with random ghosts, 5 with smart ghosts) and then recording the average score, average win rate, average time taken. The evaluation function is the same for both reflex as well as goal-based agents in order to avoid biasness of evaluation. Depth for the goal-based agents has been limited to depth = 2 as increasing depth gave too much longer computation times. The diagrams below summarize all the experimental results for all the mazes.

**5.1 Maze: Contest Classic**

**5.2 Maze: Medium Classic**

**5.3 Maze: Minimax Classic**

**5.4 Maze: Open Classic**

**5.5 Maze: Capsule Classic**

**5.6 Maze: Small Classic**

**5.7 Maze: Test Classic**

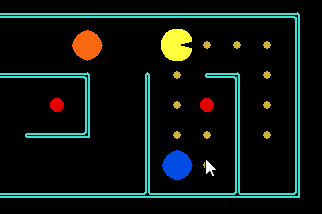
**5.8 Maze: Trapped Classic**

**5.9 Maze: Tricky Classic**

**5.10 Observations:**

The observations gained from the experiments supported the theoretical view explained in the method section.

When pacman was made to play complex maps such as ‘Contest Classic’, ‘Medium Classic’, ‘Capsule Classic’, ‘Small Classic’ and ‘Tricky Classic’, the goal based agents were able to perform quite well having a higher average score and a higher win rate. The obvious reason for this was that goal-based agents evaluate action based on their future states as well till a certain depth, and hence were able to choose more wisely. For example, given the ‘Capsule Classic’ scenario below:



A reflex agent does not look-ahead and hence the most feasible option would be to go right. However, goal-based agents would choose to go down as going right would cause them to be locked in a dead-end if a ghost follows. Hence, we see that evaluating to a certain depth, pacman is able to perform better.

The next crucial observation to be seen is that when pacman was made to play the ‘Open Classic’ maze, we see that an expectimax agent was able to achieve a higher score as compared to a minimax agent when against a random ghost. Minimax agents are unaware that the ghost does follow it and is random and hence, tries to play safe. Although this will yield a higher win rate, however the average score decrease as pacman thrashes around. However, in case of an expectimax approach , pacman is able to take advantage of the non-optimal behaviour of the adversary, hence yielding a higher score. Moreover, expectimax agents involve taking a chance which might benefit in some scenarios. For example, given the ‘Trapped Classic’ scenario below:



A minimax agent will commit ‘suicide’ first instead of taking the chance to go left and decrease its utility(score) in hopes that the blue agent would proceed down and it can eat all the food. However, expectimax agents involve chance and that is why we see that the expectimax agent has a slightly better win rate (37%) as compared to minimax agents that did not win a single game!

Another theoretical observation to be noticed included that instead of traversing the whole game tree, alpha-beta pruning can reduce the time taken by pruning unecessary nodes. A vivid result can be seen in the time taken to process the ‘Tricky Classic’ maze where an alpha-beta agent decreased the time for processing by approximately 40%! Calculation can be seen below:

Lastly, we analyze the performance of all the agent types used with pacman in order to analyze which agent is more inclined at winning with a reasonable average score and minimum computational time. Hence, combining all the statistics gained from random and smart agents, we analyze which agent performs better in a real life scenario:

Hence, we come to the conclusion, that with a better evaluation function expectimax agents are able to achieve a higher score and have an overall better win rate against both smart and dumb adversaries but are time consuming. However, reflex agents are much quicker in computation but lack in giving a better win rate. Alphabeta agents on the other might be a suitable options for those that require a quicker computation and a reasonable win rate despite some cases where the win rate is 0%.

**6.0 Conclusion/ Future Work**

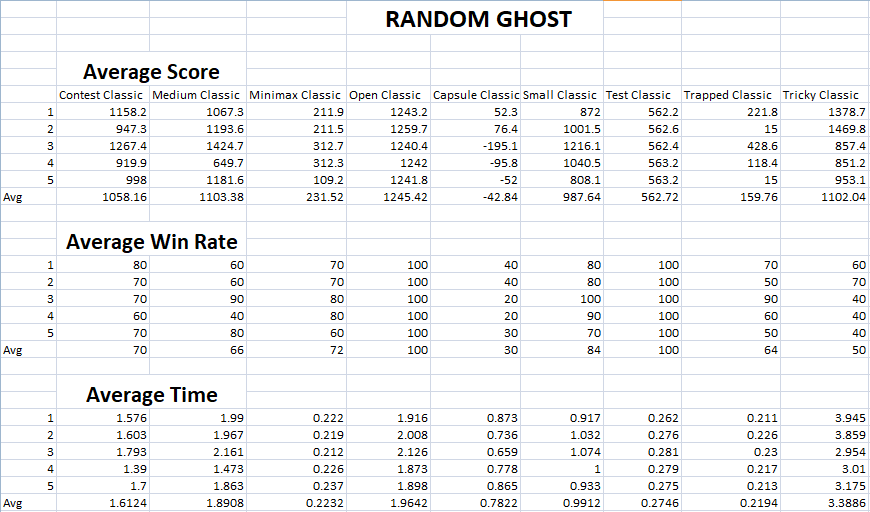
In this article we have proposed a method that plays Pac-Man according to the defined algorithms using game state and search tree to perform actions using particular module to maximize its score. The used algorithm can uncover action combination to allow our agent to pursue multiple goals. Furthermore decisions depend on the current observations and future state of action. The policy of the agent is represented as a list of if-then rules with priorities. Through observation we concluded that the worst agent was the reflex agent as it only compares current game state. Moving on to mini max the algorithm evaluates states rather than actions, as was the case for the reflex agent. Mini max agents evaluate future states whereas reflex agents evaluate actions from the current state, however mini max is memory and computational thus can only traverse through a depth of 3 or 4 on modern computers. Therefore, Alpha beta pruning can be used to remove non-optimal game state to reduce the search tree in order to improve the search time and depth. Minimax and alpha-beta work well, but they both assume an adversary who makes optimal decisions. Expectimax is useful for modeling probabilistic behavior of agents who may make suboptimal choices. Random ghosts are of course not optimal minimax agents, and so modeling them with minimax search may not be appropriate. This expectimax pacman will no longer take the min over all ghost actions, but the expectation according to a perceived model of how the ghosts act.

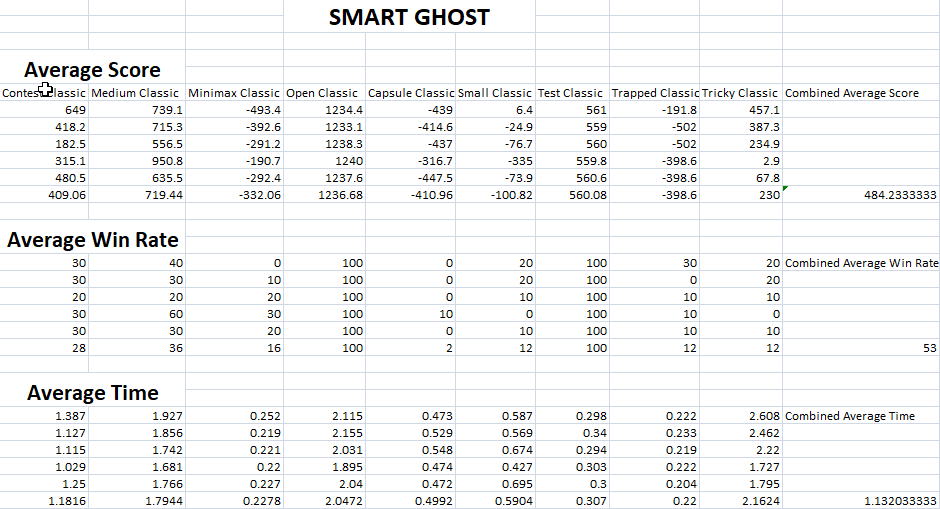
Our project can be a useful benchmark in considering different representations and approaches to evolving a Pac-Man playing agent. It is expected that future work would need to be able to improve on the performance (fitness) of the agent. However it will also be interesting to compare the complexity (dimensionality, interpretability, computational requirements) of alternative approaches, to the rule-based approach developed above. Finally, we conjecture that general aspects of the approach taken here to developing an adaptive agent for Pac-Man may eventually lead to techniques that can be applied successfully to other real-time computer game.

**7.0 Appendices**

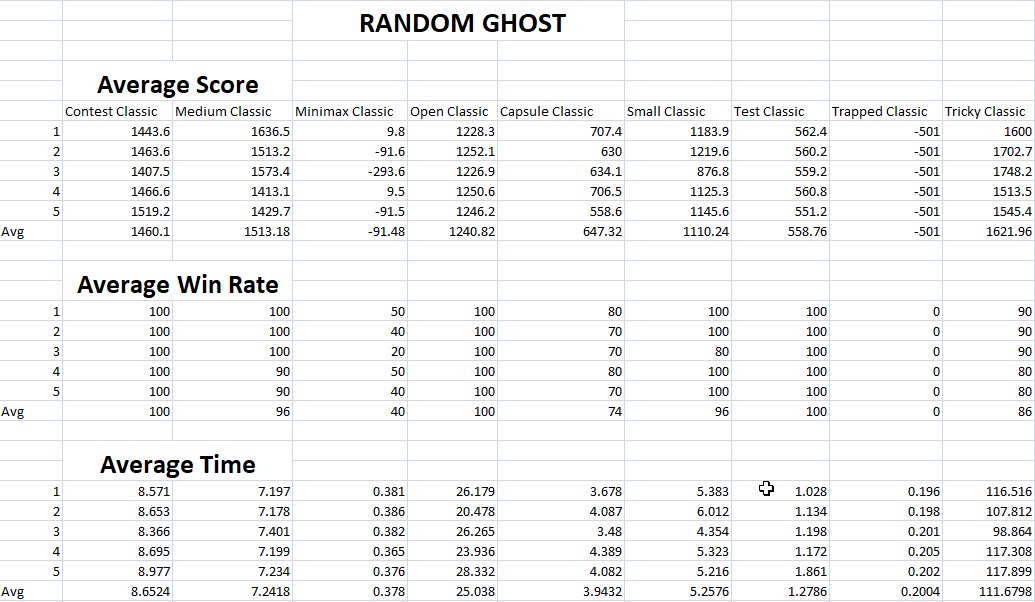
Recorded results for the experiment are as follows:

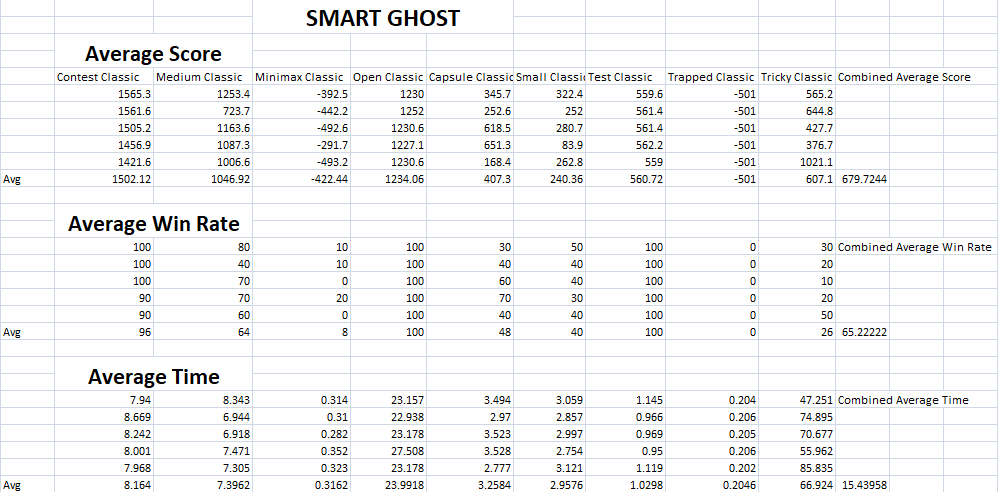
**REFLEX AGENT:**



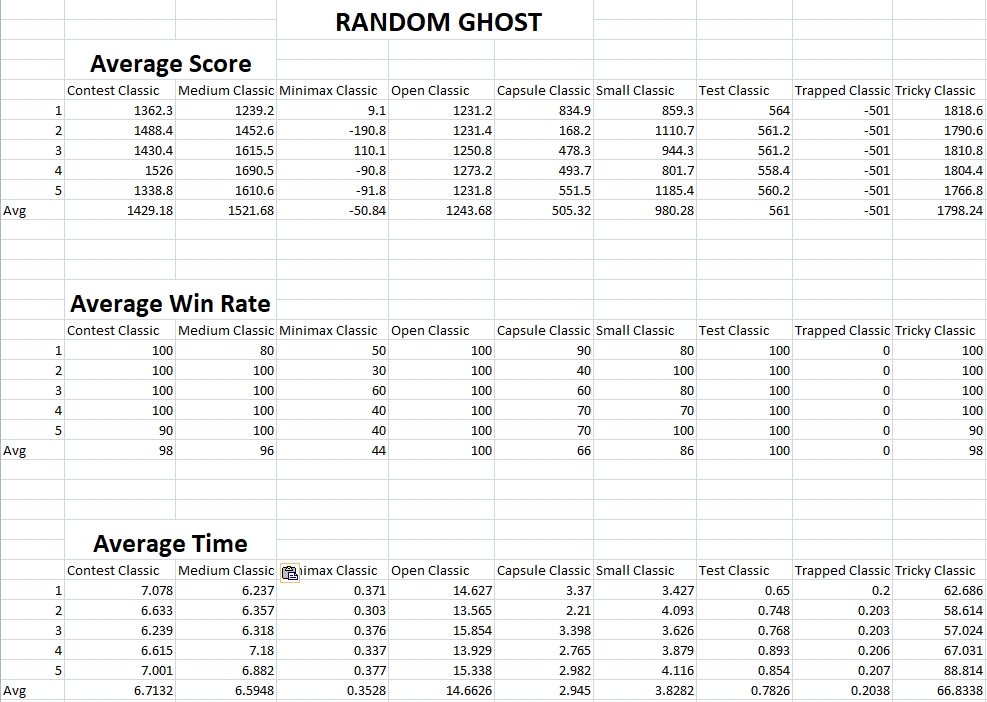


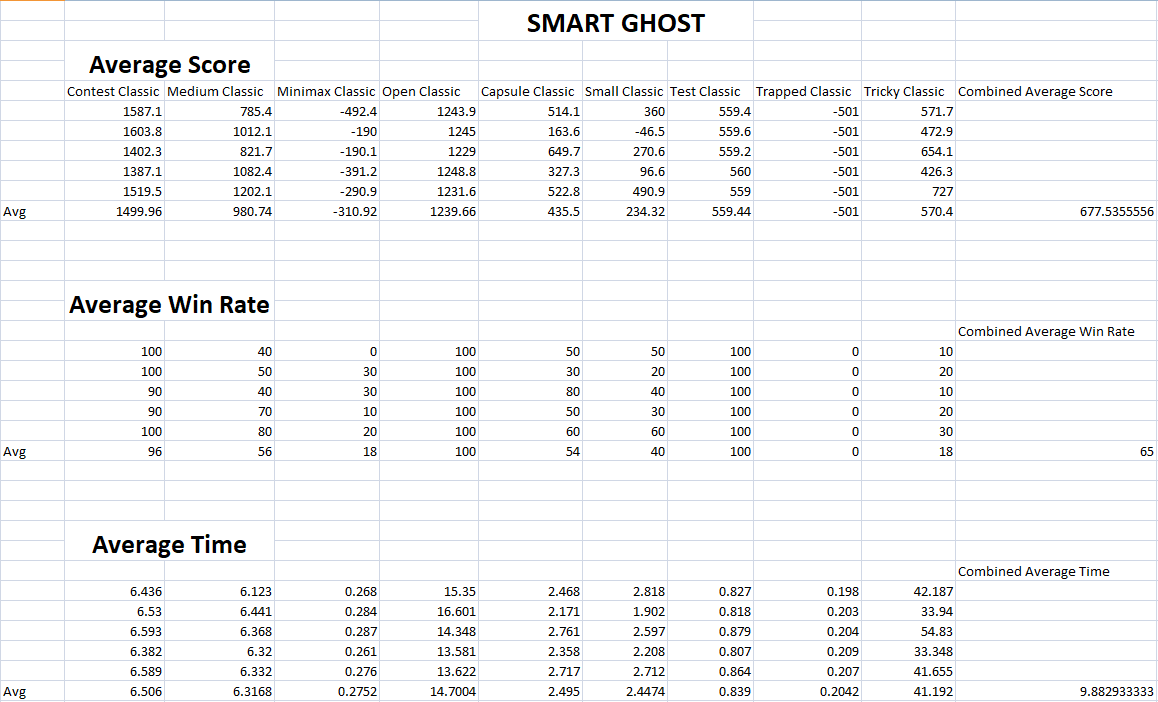
**MINIMAX AGENT:**



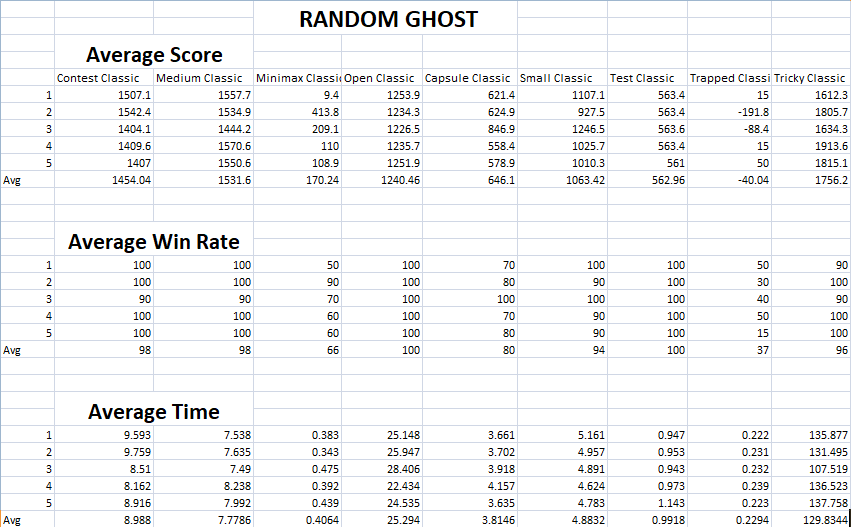
****

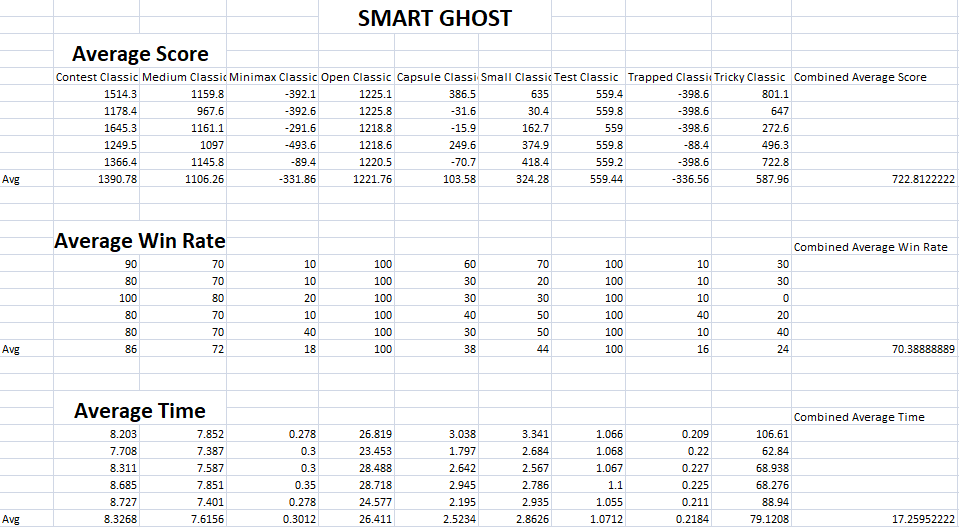
**ALPHA BETA AGENT:**



****

**EXPECTIMAX AGENT:**

****

****

**8.0 Contributions**

Murad Popattia:

* Developing AI for Pacman (Minimax, AlphaBeta)
* Developing evaluation function
* Developing random ghost agents
* Making game rules
* Collection of experimental data
* Making project report

Muhammad Samiullah:

* Developing AI for Pacman (Reflex, Expectimax)
* Developing smart ghost agents
* Making game states, game flow classes
* Making of command parser

Muhammad Hassan Azhar:

* Development of GUI
* Developing maze layouts and maze classes
* Developing graphics display for the pacman
* Making diagrams from data

**9.0 References**

[1]: M. Gallagher and A. Ryan, "Learning to play Pac-Man: an evolutionary, rule-based approach," The 2003 Congress on Evolutionary Computation, 2003. CEC '03., Canberra, ACT, Australia, 2003, pp. 2462-2469 Vol.4, doi: 10.1109/CEC.2003.1299397.

[2]: D. Robles and S. M. Lucas, "A simple tree search method for playing Ms. Pac-Man," 2009 IEEE Symposium on Computational Intelligence and Games, Milano, 2009, pp. 249-255, doi: 10.1109/CIG.2009.5286469.

### [3]: Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man I. Szita, A. Lorincz