**PacManAI – Maintenance and Code enhancement**

700102 ACW1

This report details the steps taken to implement a Moving Target Search heuristic algorithm in an existing C# Solution. The provided solution contained the full sets of logic, features, and user interfaces required to simulate a game of Ms Pac-Man, with varying options for controlling Ms Pac-Man ranging from random movement to the use of an MCTS algorithm. The aim was to implement the Moving Target Search to affect the behavior of one ghost with the expectation that it will perform with greater accuracy in seeking Ms Pac-Man.

It was crucial to determine a strong understanding of how the existing implementation functions. This included assessing the functionality of member methods and which abstract or virtual classes various objects inherited from, and was arguably the longest part of this task. A class diagram was generated using Visual Studio to assist with the understanding of the class structure of solely the *PacManGameLogic* project. All other projects in the solution were not deemed relevant for controlling the logic of the game itself, which is what needs to be manipulated to change ghost movement.

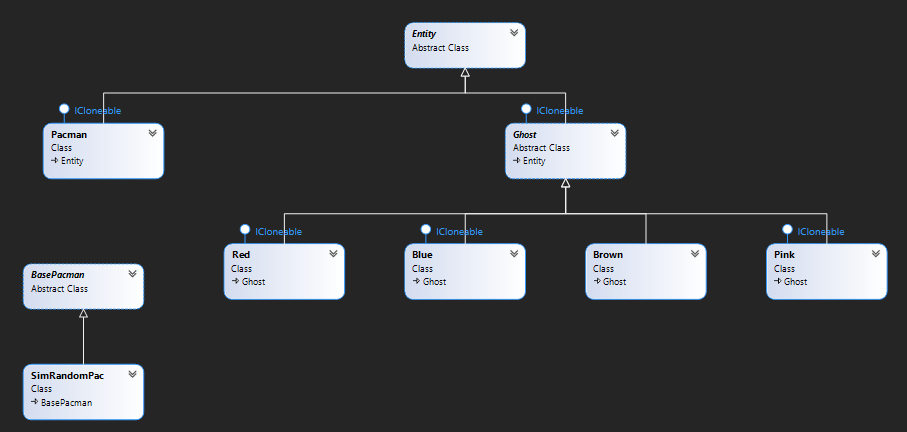


Figure 1: Pacman.GameLogic class diagram.

Additionally, there were two namespaces to consider: *GameLogic*, and a child namespace *GameLogic.Ghosts*. The class diagram (*fig. 1*) makes it clear that the ghosts inherit from *Ghost* and *Entity* classes. Only one ghost needed to make use of the MTS algorithm; red was chosen as its movement logic was considered to be the least efficient out of all ghosts in the application. Changes of movement functionality was required in Red.cs, and overriding some abstract and virtual methods would help to facilitate this.

Table 1: List of changes made across the VS Solution\*

|  |  |
| --- | --- |
| **File** | **Contents changed** |
| Red.cs | * Included Reversal method. * Changed contents of Move method. |
| Ghost.cs | * Changed Reversal method to virtual. * Included TryGoInverseAllowed method. |
| GhostMTS.cs | * Implemented the GhostMTS class. * getGhostLocation method. * getPacmanLocation method. * rankedDirections method. |
| GameState.cs | * Changed Ghosts member array from size 4 to size 1. * Changed GameState constructors so that only one ghost (red) would be added to the Ghosts member array. |
| PacmanAI: UncertainAgent.cs | * In UncertainAgent, changed UseSmart to true. |

\*Not all contents listed are utilised in the final implementation in submitted code. Some changes were made temporarily and reverted.

//**WRITE ABOUT GHOSTMTS HERE**

The content from the overridden *Move* method in the *Red* class was replaced with calls to new functions written in a new file: GhostMTS.cs. As seen on line 45 of Red.cs, an array of Direction types is returned, and these movement directions are considered in turn, with the first being the most optimal direction to travel, and the last being the least optimal direction to travel. Direction is an enumerated type, and each Entity (or child) object has a member direction and nextDirection.

Initially, a foreach loop was used to attempt to move the ghost in each direction. This called the TryGo method in Ghost.cs and checked if the node in the direction given was a wall. If this was the case, then the direction was not allowed and false would be returned. The object’s nextDirection variable would be set to the first succeeding direction. It was therefore clear that nextDirection would become Direction when the move method was called.

With the aim of improving efficiency, a new function was added to Ghost.cs. TryGoInverseAllowed had almost additional functionality to the aforementioned TryGo method, however, the part that checks if the requested direction is the inverse of the current direction was removed, which would allow changes of direction mid-path. In some circumstances, this improved performance, but in others it hindered the ghost as it would sometimes turn around when it would not have been appropriate to do so.

Ultimately, this method the original TryGo method were abandoned. In reading the (since removed) MoveAsRed function, it was found that MoveInFavouriteDirection already implemented a similar solution to move in the correct direction. This was implemented in Red.cs, removing all logic from its Move method, and improving efficiency by reusing code.

This solution worked quite well, however, there was an issue where the ghost would randomly change direction when in pursuit of Ms Pac-Man. Eventually, it was realised that the direction was being reversed by GameLogic calling a ghost’s Reversal method from Ghost.cs. Removing this may have broken existing functionality, so instead the virtual keyword was used, and the method was overridden just for the red ghost. This method was left empty, and the direction change issue was resolved.

To test ghost performance, the number of ghosts added to the game was reduced to 1. Additionally the UncertainAgent was set to always be “smart”, as the “dumb” solution was not useful for testing, and there was a 50% chance of either being implemented at runtime. Locations of these changes can be seen in *Table 1*.

The performance of the red ghost has been vastly improved by implementing MTS, and it regularly exceeds the performance of other ghosts by catching Ms Pac-Man the fastest. However, this time could be further reduced by allowing the ghost to use tunnels on the map to its advantage. The Pacman.cs code gives slight detail to how tunnels could be identified. Additionally, the *stall* direction alongside allowing inverse directions could have been used to catch Ms Pac-Man more effectively. However, this solution quickly goes wrong when Ms Pac-Man pauses and waits, or goes back & forward along a path, as it causes the ghost to follow the same behaviour! Further algorithmic design could have relieved this issue by only allowing the ghost to change direction after a set time interval. The amount of overlapping functionality within the *entity* class means that multiple implementations may have similar performance, even if different member methods are utilised.