Module – Artificial Intelligence for Games

Project – Designing an MS Pac Man Controller

Group - 08

Rahul Verma -20175183 **Shivam Sood – 20050577**

Mithun Thakkar – 20017138 **Dishant Mewada - 17079349**

Introduction to MS Pac Man:

MS Pac Man is a maze arcade game developed by General Computer Corporation in 1982. It is the sequel to Pac-Man from 1980. In this game, the player is tasked with eating all of the pellets in an enclosed maze while avoiding four coloured ghosts. Eating large flashing "Power Pellets" will cause the ghosts to turn blue and flee, which can be consumed for bonus points. However, it is not as easy as it sounds, because this game involves randomness due to being probabilistic in nature. It is also considered to be a favourite game among AI researchers.

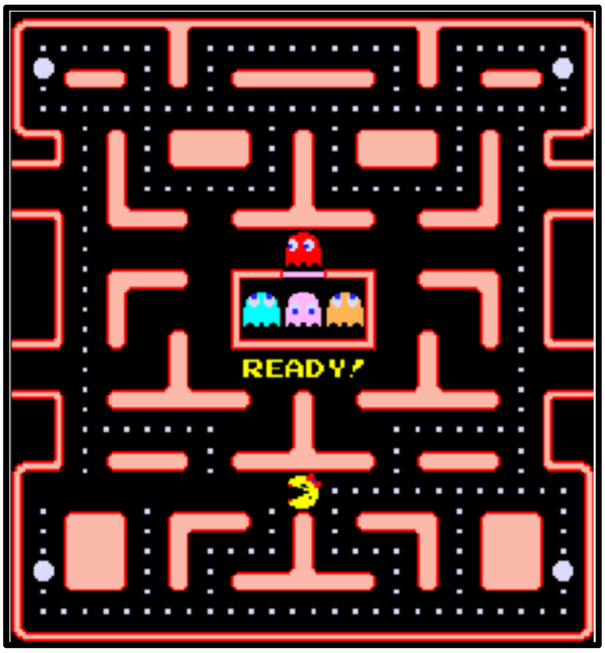


Fig 1: MS Pac Man Layout

The player navigates Pac-Man through a maze and has to collect all the dots (Pac-Dots) in order to complete the stage. Pac-Man is being chased by four ghosts in the game whose main objective is to kill her. The four ghosts, Blinky, Pinky, Inky and Sue, each have different behaviour depending on the mode of the ghosts. The ghosts change mode during game play from scattering to the corners of the maze, to chasing Pac-Man and also to being frightened when Pac-Man picks up a Power-Pellet.

Behaviour of the above mentioned of ghosts are as follows:

Blinky - The red ghost is named as Blinky and is considered to be the leader of ghosts. This ghost is considered to be aggressive and it follows Pac Man from behind.

Pinky – As the name suggests, the pink female ghost is Pinky and she uses ambush tactics to position herself in front of Pac-Man to surround him.

Inky – The cyan coloured ghost is Inky and has a fickle mood. He can be unpredictable. Sometimes he chases Pac-Man aggressively like Blinky; other times he jumps ahead of Pac-Man as Pinky would. Though the smartest, he lacks focus most of the time. In Pac-Man, Inky likes to appear in front of Pac-Man's face.

Sue – The orange ghost is named Sue, he will chase after Pac-Man in Blinky's manner, but will wander off to his home corner when he gets too close. He is designed to act stupid.

Objective of this project:

The objective of this project is to design a MS Pac Man controller based on Java and come up with strategies that will maximize the scores at each level. Here, we are specifically concerned with the average score obtained at each level.

Motivation:

The motivation comes straight away from the different resources, technologies, code-walkthroughs and algorithms learnt in the last 12 weeks. It is also quite challenging and exciting to apply AI techniques on such a simple game. Also, playing this game since childhood never gave us any perspective, which we did after reading a few related papers and working on code.

Introduction to Behavioral Trees:

This section gives an introduction to a type of tree search algorithm called Behavioural Trees. These trees consist of Edges (child) and Nodes (parent). In the case of MS PacMan, edges in the tree represent actions the agent takes to get from one state to another, with nodes representing states. This methodology helps in finding solutions to problems in a search space (here, the problem is, what should be our next move based on the data collected so far?).

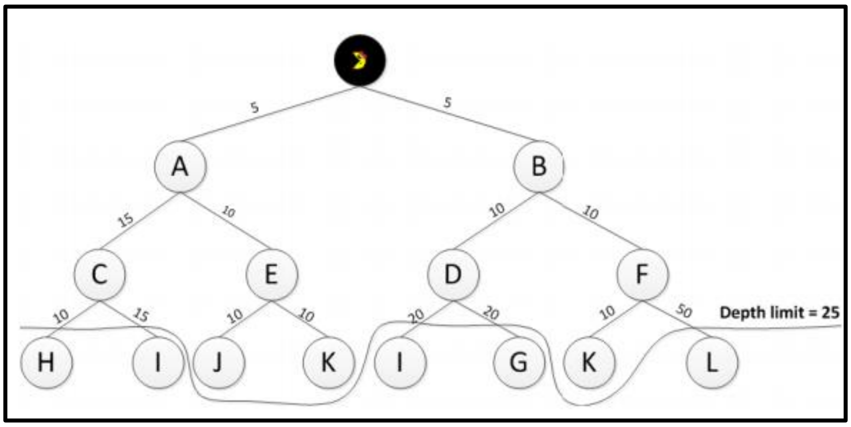


Fig 2: Behavioural Tree Example.

The above diagram represents an example tree search pattern with a depth limit of 25. It can be seen how the nodes are traversed and back-propagation are used to get the outcome. In MS PacMan, Each node stores all the information required to check and update the current state of the game. This is just one use-case, however, this tree search technique has been used at more class and subclass level in the Java code provided.

MS PacManCG12 Java Based Code Walkthrough:

Before we get into code, it is useful to understand how the maze is laid out and how the distance from one corner to another corner is calculated.

Maze - Each maze is stored as a (connected) graph; all nodes have neighbours, stored in an array of length 4. The maze can also be interpreted as an x, y coordinate. There are four ways (these functions are in-built) in which the distance between these nodes are calculated, which are as follows:

Path - This is the actual path distance to reach the target.

Euclidean Distance - Euclidean distance is the distance calculated using the Node's x and y coordinates. This distance is computed on the fly, instead of pre-computing.

Manhattan Distance - This is the absolute distance between a given x and y coordinate. This distance is computed on the fly, instead of pre-computing.

Shortest Path Distance - Shortest Path Distance returns the shortest distance from any node in the maze to any other node**.** On contrary to the other functions, this is the only built -in function which is pre-computed.

In order to design a controller and come up with strategies, it becomes very important to understand the actual working of the game at the technical level. Hence, this section will provide detailed information on some of the most important game classes and their related functions that are used under **MSPacManCG12**.

1. ***Executor Class* -** This is the executor class which is used to execute the game. We have been provided several options to execute the game which are as follows:
    - **Timed** - Runs the game with a time limit.
    - **Un-timed** - Runs the game without a time limit.
    - **With visuals** - Run the game with visuals.
    - **Without visuals** - Run the game without visuals.
    - **Delay** - The delay between time-steps.
    - **Number of trials** - The number of trials to be executed.

We have modified the runExperiment function in Executor class to include the alive time and average alive time for MS PacMan.

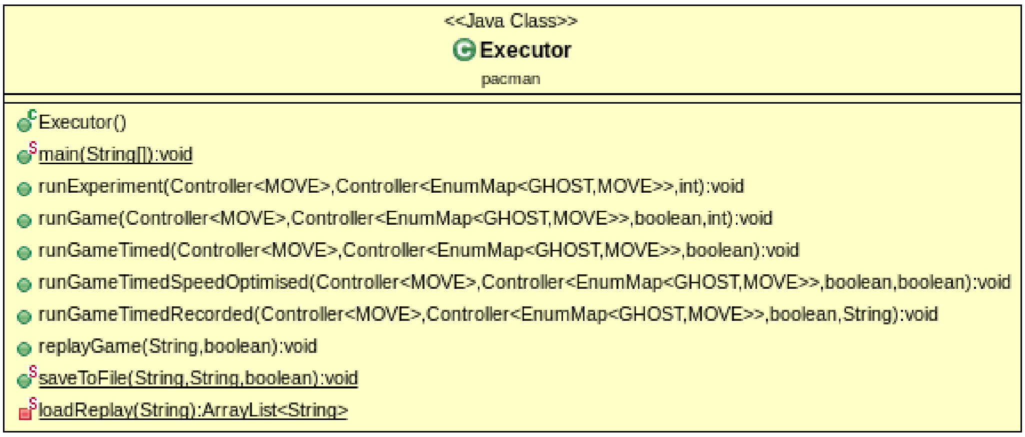

<<Java Class>>
Executor
pacman
Executor()
main(String[]):void
runExperiment(Controller<MOVE>,Controller<EnumMap<GHOST,MOVE>>,int):void
runGame(Controller<MOVE>,Controller<EnumMap<GHOST,MOVE>>,boolean,int):void
runGameTimed(Controller<MOVE>,Controller<EnumMap<GHOST,MOVE>>,boolean):void
runGameTimedSpeedOptimised(Controller<MOVE>,Controller<EnumMap<GHOST,MOVE>>,boolean,boolean):void
runGameTimedRecorded(Controller<MOVE>,Controller<EnumMap<GHOST,MOVE>>,boolean,String):void
replayGame(String,boolean):void
saveToFile(String,String,boolean):void
loadReplay(String):ArrayList<String>

fig 3: Executor Class Functionality.

2. ***Controller Class*** - Controller Class contains various methods which are passed to Executor class to use the controller in different execution modes. For example, a function called getMove() retrieves the move from the controller, this value is stored in the lastMove variable. Returned value is used to calculate the next move. It is the main class as it can be seen from the UML diagram, all other classes are called from this class as a single package.

3. **Human Controller Class and Keyboard Input Class** - These two classes when combined together, allows a human player to play the game using the arrow key of the keyboard. We have run HumanController.java to get familiarized with the gameplay and create strategies for our CustomControllerPacman class.

4. **BestPacManController and RandomSeedControllerPacMan Class –** We are implementing our custom made PacMan controller in these two classes.

    - BestPacManController class has the best minimum and maximum distance that we found
    - RandomSeedControllerPacMan which takes random minimum and maximum distance in suitable range.

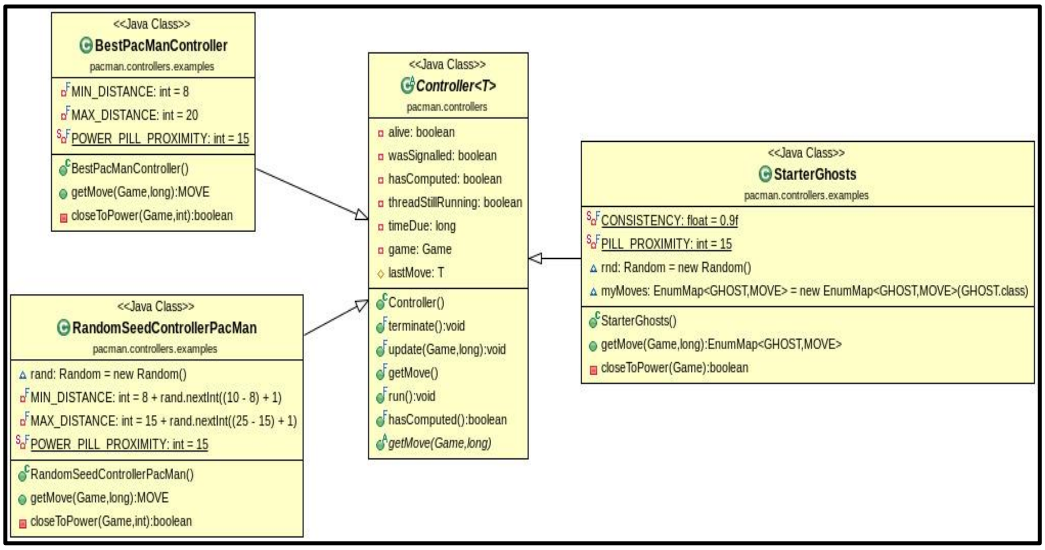

<<Java Class>>
BestPacManController
pacman.controllers.examples
MIN_DISTANCE: int = 8
MAX_DISTANCE: int = 20
POWER_PILL_PROXIMITY: int = 15
BestPacManController()
getMove(Game,long):MOVE
closeToPower(Game,int):boolean
<<Java Class>>
Controller<T>
pacman.controllers
alive: boolean
wasSignalled: boolean
hasComputed: boolean
threadStillRunning: boolean
timeDue: long
game: Game
lastMove: T
Controller()
terminate():void
update(Game,long):void
getMove()
run():void
hasComputed():boolean
getMove(Game,long)
<<Java Class>>
StarterGhosts
pacman.controllers.examples
CONSISTENCY: float = 0.9f
PILL_PROXIMITY: int = 15
rnd: Random = new Random()
myMoves: EnumMap<GHOST,MOVE> = new EnumMap<GHOST,MOVE>(GHOST.class)
StarterGhosts()
getMove(Game,long):EnumMap<GHOST,MOVE>
closeToPower(Game):boolean
<<Java Class>>
RandomSeedControllerPacMan
pacman.controllers.examples
rand: Random = new Random()
MIN_DISTANCE: int = 8 + rand.nextInt((10 - 8) + 1)
MAX_DISTANCE: int = 15 + rand.nextInt((25 - 15) + 1)
POWER_PILL_PROXIMITY: int = 15
RandomSeedControllerPacMan()
getMove(Game,long):MOVE
closeToPower(Game,int):boolean

Fig 4: UML Class Diagram of MS Pac Man.

To increase the score of Ms. PacMan we have 5 strategies as follows, these strategies are found under **BestPacManController** under **PacMan.controllers.examples**. All 5 strategies are discussed below with tree diagra

Strategy 1:

If the ghost is in close proximity, Ms. PacMan runs away from ghosts provided ghosts are non-edible. If the ghost is non-edible with liar time 0, we first compare it with minimum distance. We took minimum distance 8 since we found that minimum distance in range 8 to 10 gives better results. If the ghost node index and current node index are found to be less than minimum distance, then Ms. PacMan runs away from the ghost.

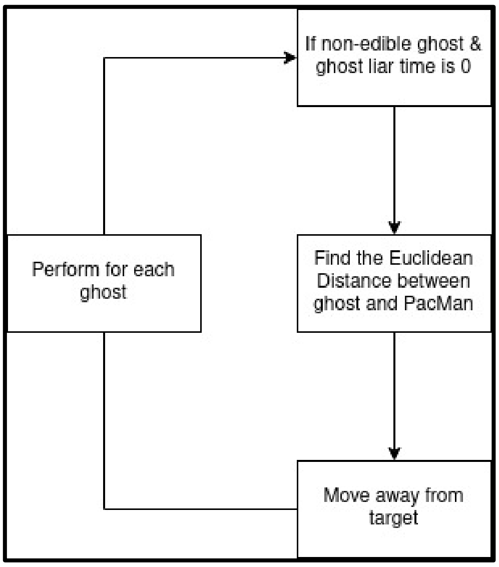

If non-edible ghost & ghost liar time is 0
Perform for each ghost
Find the Euclidean Distance between ghost and PacMan
Move away from target

Fig 5: Strategy 1.

We have found that Euclidean Distance between ghosts and MS PacMan gives better score compared to shortest path distance and Manhattan Distance.

Strategy 2:

Find the nearest edible ghost and go after them. First we calculate the maximum value of minDistance variable and also calculate the ghost edible time, if it matches the specified conditions, then the distance between the Node and the ghost is calculated. The ghosts are then eaten and the next move is calculated.

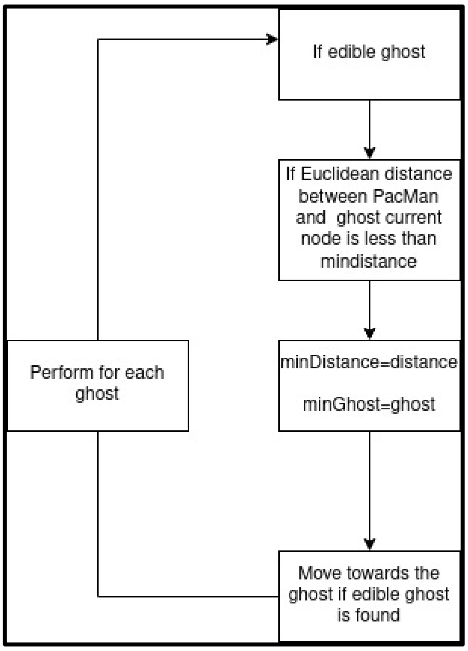

If edible ghost
If Euclidean distance between PacMan and ghost current node is less than mindistance
minDistance=distance
minGhost=ghost
Perform for each ghost
Move towards the ghost if edible ghost is found

Fig 6: Strategy 2.

Strategy 3:

We created a new array list named Distance_From_Ghosts which will be used in strategy 4 as well. For each ghost, all three distances are added to this array list. If the distance from each ghost is less than max distance and ghost liar time is not equal to zero, PacMan moves away from the ghost. The flow diagram is as follows:

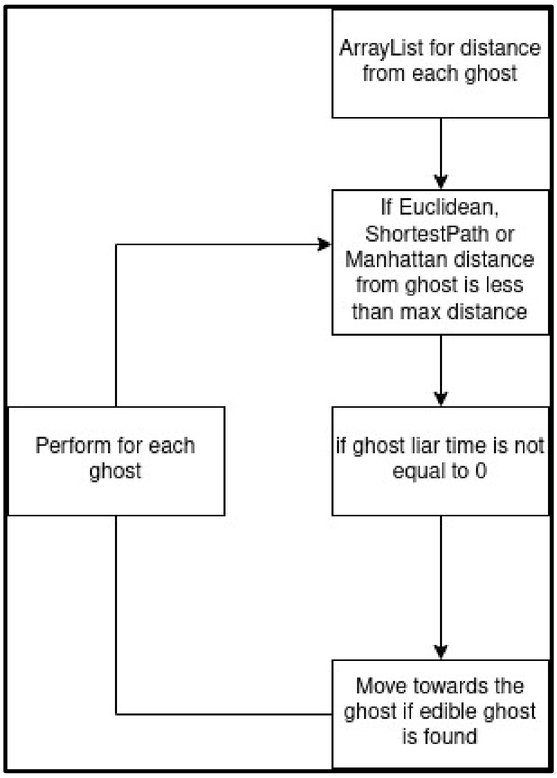

ArrayList for distance from each ghost
If Euclidean, ShortestPath or Manhattan distance from ghost is less than max distance
Perform for each ghost
if ghost liar time is not equal to 0
Move towards the ghost if edible ghost is found

Fig 7: Strategy 3.

Strategy 4:

We use the previously created Distance_From_Ghosts array list for this strategy. Iterating through each element in the array list, we check whether distance from each ghost is greater than minimum distance. If the condition is true, iterate through each power pill active indices. If power pill is close by, we add power pills indices into the targets array list otherwise we look for pills.

But if the ghost is in the range ie shorted path distance between PacMan and Ghost node index equals min distance from ghosts, then we move away from ghosts instead of collecting power pills.

The overall flowchart is shown below:

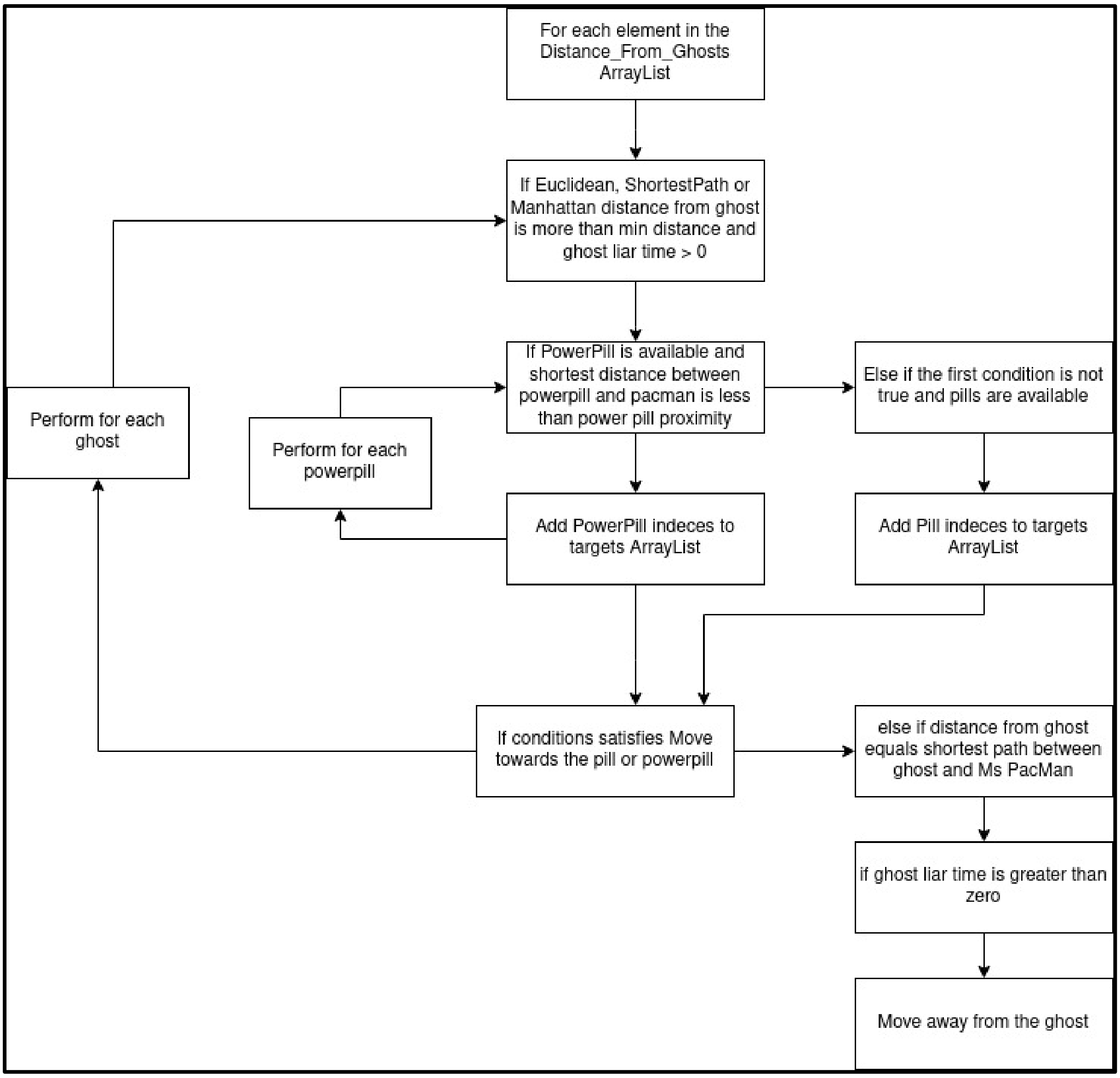

For each element in the Distance_From_Ghosts ArrayList
If Euclidean, ShortestPath or Manhattan distance from ghost is more than min distance and ghost liar time > 0
If PowerPill is available and shortest distance between powerpill and pacman is less than power pill proximity
Else if the first condition is not true and pills are available
Perform for each ghost
Perform for each powerpill
Add PowerPill indeces to targets ArrayList
Add Pill indeces to targets ArrayList
If conditions satisfies Move towards the pill or powerpill
else if distance from ghost equals shortest path between ghost and Ms PacMan
if ghost liar time is greater than zero
Move away from the ghost

Fig 8: Strategy 4.

Strategy 5:

One of the issues faced during the game was that Ms. PacMan was getting stuck at one neutral position. So, we created strategy 5, where if Ms. PacMan is found with a neutral move, we try to move her in different directions according to the last element in the targets array list.

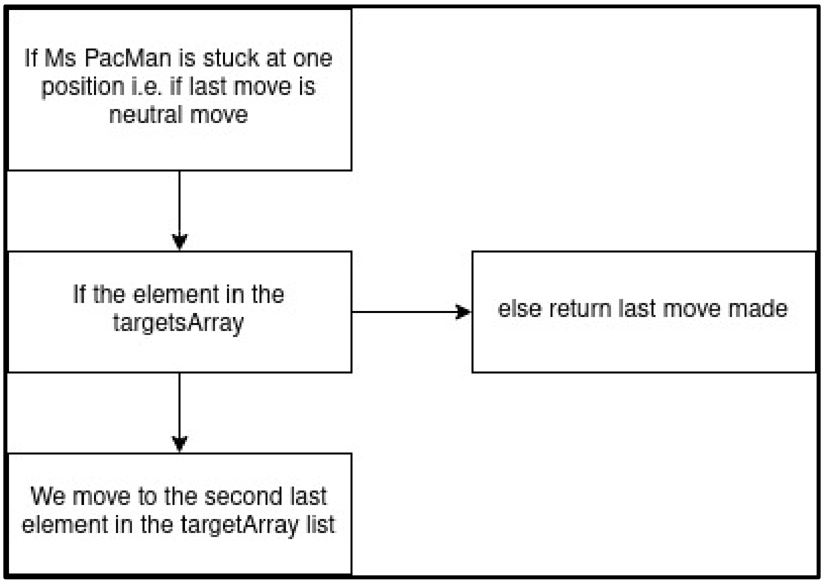

If Ms PacMan is stuck at one position i.e. if last move is neutral move
If the element in the targetsArray
else return last move made
We move to the second last element in the targetArray list

Fig 9: Strategy 5.

Results:

We have two custom classes made. One for experimenting with random min and max distance and the other one for the best values that we found.

Overhere, we are running 100 experiements with BestPacManController class and the results are as shown below

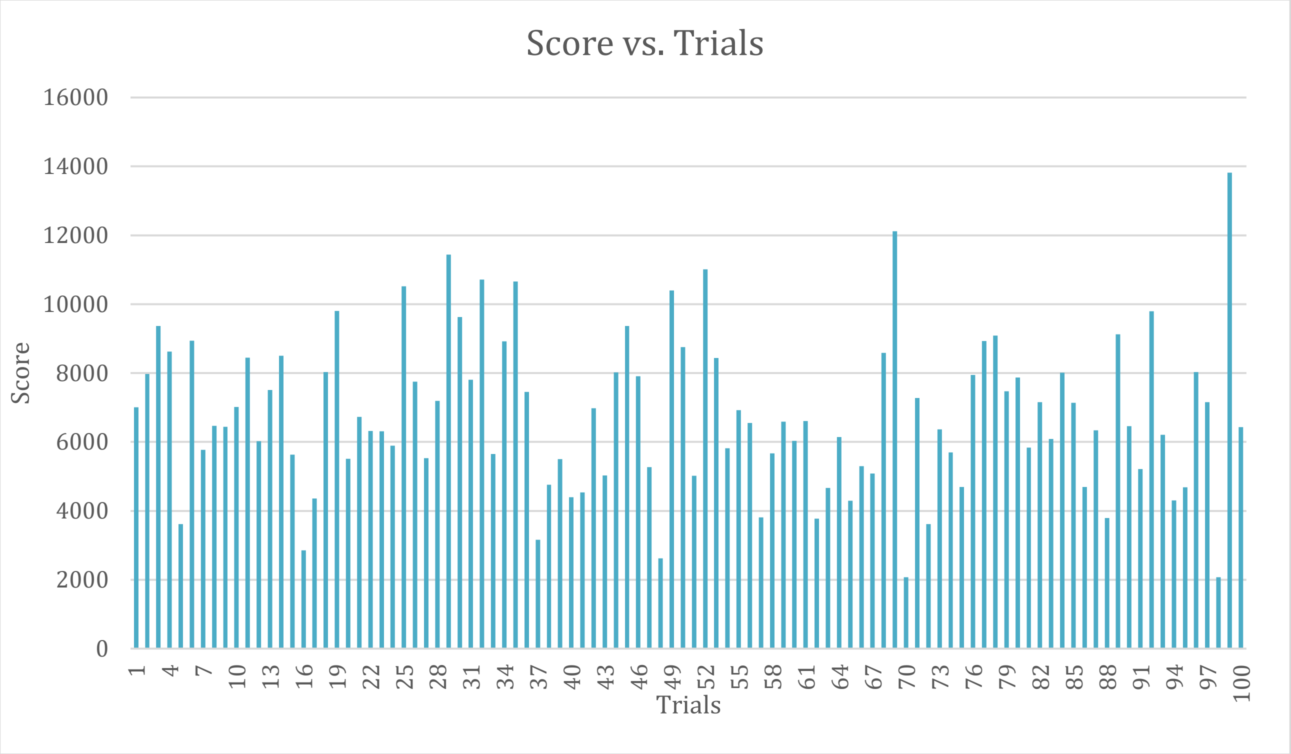


Fig: Bar graph of Score against Trials.

The above bar graph shows how score changes with respect to the number of Trials, this is obvious and is due to the probabilistic nature of the game. At times, it was also observed that there is a spike in score, but that wasn't consistent.

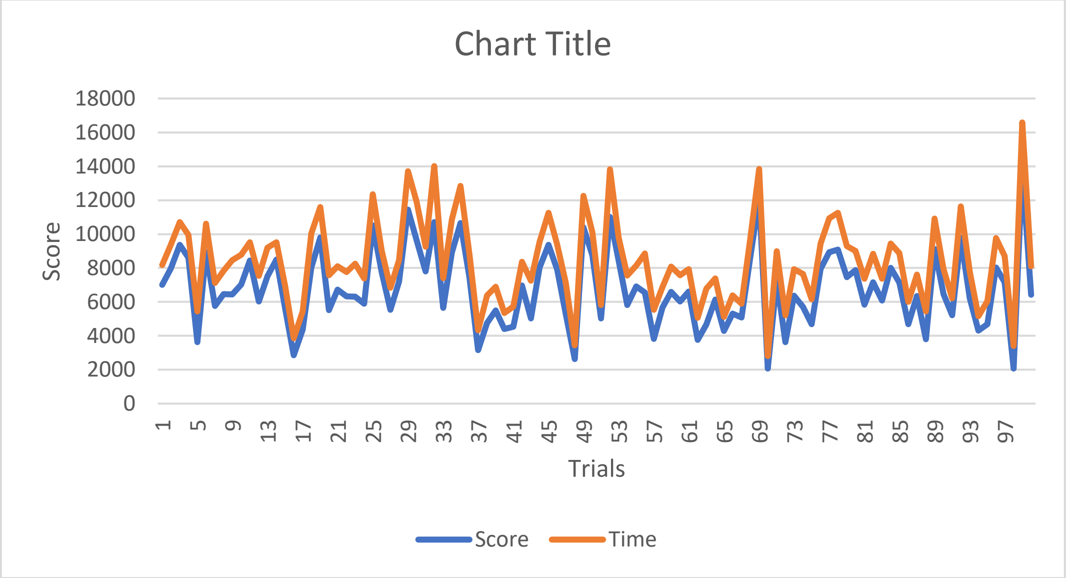


Fig: Graph of Score & Time against Trials.

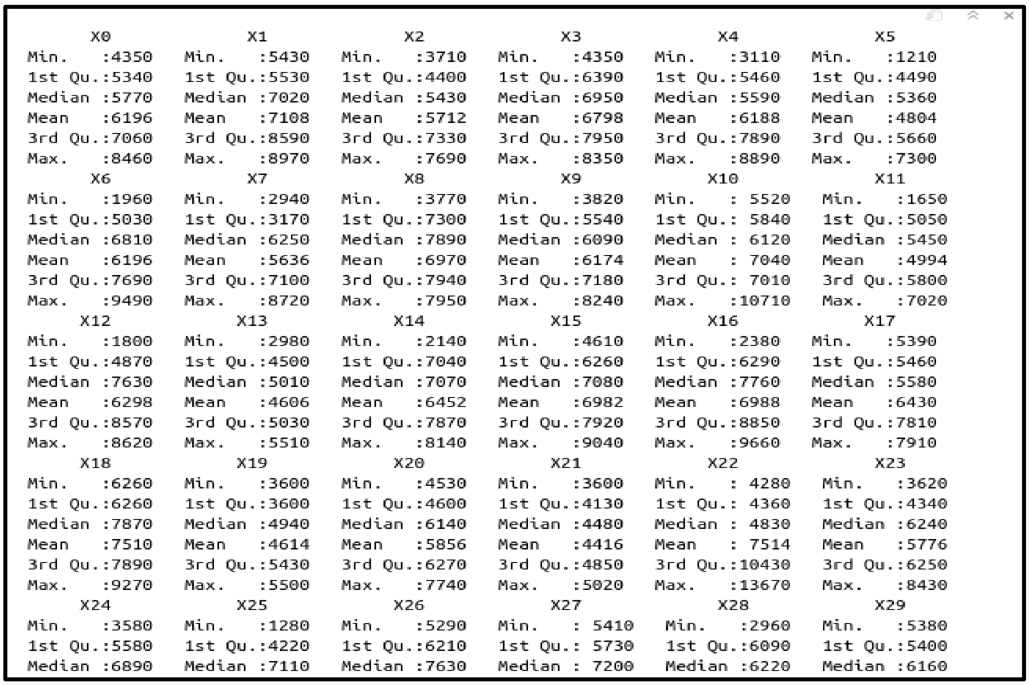
The above graph shows the relationship between the Time & Score with respect to number of trials.

It is evident from the graph that Time and Score both have a similar tendency when compared to number of trials. At times, high scores were achieved in less duration and other times, it was the opposite.

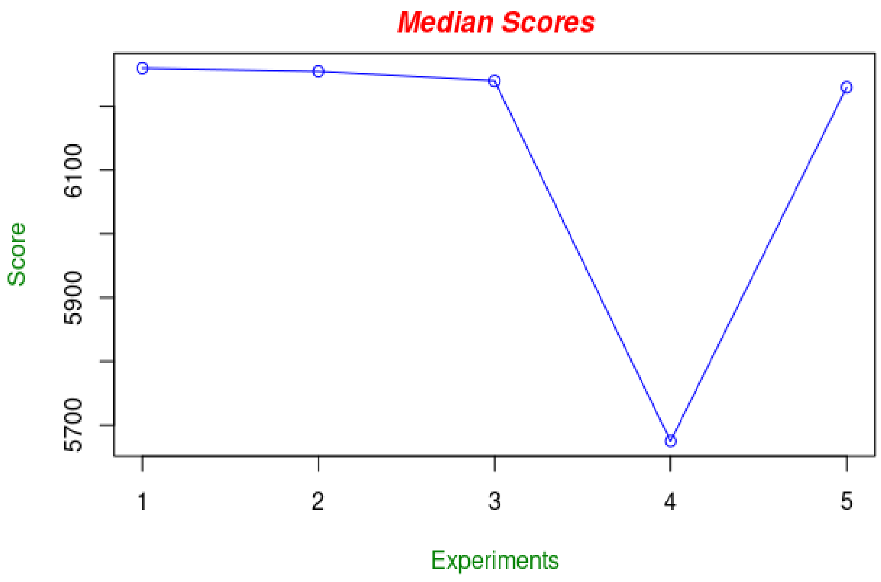
Number of Experiments vs. Score in R:

In the code, we are running 5 experiments with 100 multiple runs each, all with different random seed i.e. random MinDistance and MaxDistance in a suitable range.

We are first getting score from each run and plotting the random seed experiments (each with 100 runs) vs score.

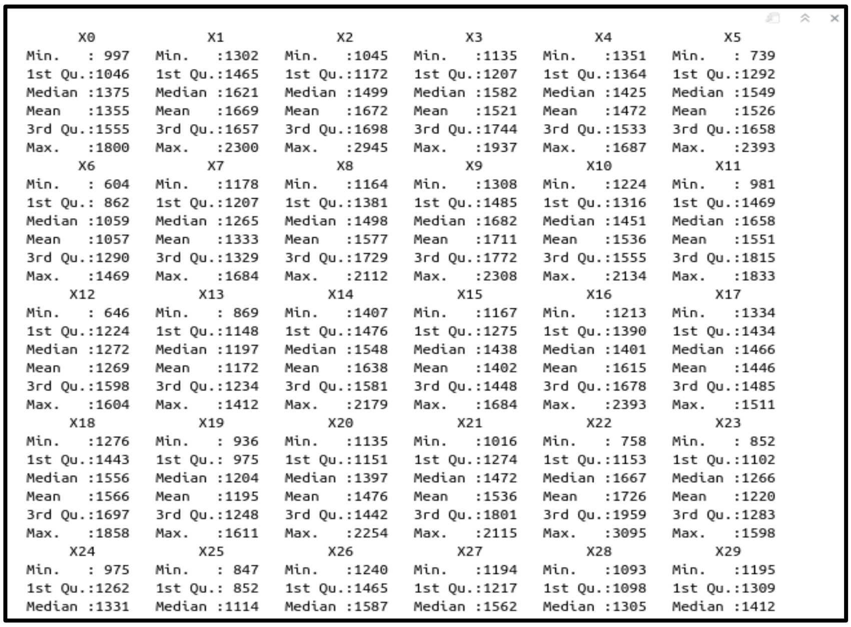


```
       X0              X1              X2              X3              X4              X5
 Min.   :4350    Min.   :5430    Min.   :3710    Min.   :4350    Min.   :3110    Min.   :1210
 1st Qu.:5340    1st Qu.:5530    1st Qu.:4400    1st Qu.:6390    1st Qu.:5460    1st Qu.:4490
 Median :5770    Median :7020    Median :5430    Median :6950    Median :5590    Median :5360
 Mean   :6196    Mean   :7108    Mean   :5712    Mean   :6798    Mean   :6188    Mean   :4804
 3rd Qu.:7060    3rd Qu.:8590    3rd Qu.:7330    3rd Qu.:7950    3rd Qu.:7890    3rd Qu.:5660
 Max.   :8460    Max.   :8970    Max.   :7690    Max.   :8350    Max.   :8890    Max.   :7300
       X6              X7              X8              X9             X10              X11
 Min.   :1960    Min.   :2940    Min.   :3770    Min.   :3820    Min.   : 5520    Min.   :1650
 1st Qu.:5030    1st Qu.:3170    1st Qu.:7300    1st Qu.:5540    1st Qu.: 5840    1st Qu.:5050
 Median :6810    Median :6250    Median :7890    Median :6090    Median : 6120    Median :5450
 Mean   :6196    Mean   :5636    Mean   :6970    Mean   :6174    Mean   : 7040    Mean   :4994
 3rd Qu.:7690    3rd Qu.:7100    3rd Qu.:7940    3rd Qu.:7180    3rd Qu.: 7010    3rd Qu.:5800
 Max.   :9490    Max.   :8720    Max.   :7950    Max.   :8240    Max.   :10710    Max.   :7020
      X12             X13             X14             X15             X16             X17
 Min.   :1800    Min.   :2980    Min.   :2140    Min.   :4610    Min.   :2380    Min.   :5390
 1st Qu.:4870    1st Qu.:4500    1st Qu.:7040    1st Qu.:6260    1st Qu.:6290    1st Qu.:5460
 Median :7630    Median :5010    Median :7070    Median :7080    Median :7760    Median :5580
 Mean   :6298    Mean   :4606    Mean   :6452    Mean   :6982    Mean   :6988    Mean   :6430
 3rd Qu.:8570    3rd Qu.:5030    3rd Qu.:7870    3rd Qu.:7920    3rd Qu.:8850    3rd Qu.:7810
 Max.   :8620    Max.   :5510    Max.   :8140    Max.   :9040    Max.   :9660    Max.   :7910
      X18             X19             X20             X21             X22              X23
 Min.   :6260    Min.   :3600    Min.   :4530    Min.   :3600    Min.   : 4280    Min.   :3620
 1st Qu.:6260    1st Qu.:3600    1st Qu.:4600    1st Qu.:4130    1st Qu.: 4360    1st Qu.:4340
 Median :7870    Median :4940    Median :6140    Median :4480    Median : 4830    Median :6240
 Mean   :7510    Mean   :4614    Mean   :5856    Mean   :4416    Mean   : 7514    Mean   :5776
 3rd Qu.:7890    3rd Qu.:5430    3rd Qu.:6270    3rd Qu.:4850    3rd Qu.:10430    3rd Qu.:6250
 Max.   :9270    Max.   :5500    Max.   :7740    Max.   :5020    Max.   :13670    Max.   :8430
      X24             X25             X26             X27              X28             X29
 Min.   :3580    Min.   :1280    Min.   :5290    Min.   : 5410    Min.   :2960    Min.   :5380
 1st Qu.:5580    1st Qu.:4220    1st Qu.:6210    1st Qu.: 5730    1st Qu.:6090    1st Qu.:5400
 Median :6890    Median :7110    Median :7630    Median : 7200    Median :6220    Median :6160
```

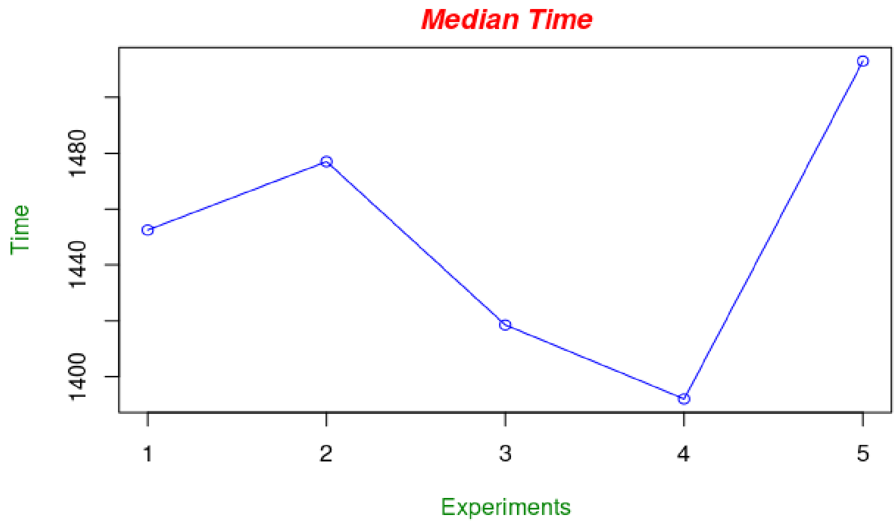


Number of Experiments vs. Time:

Over here, we have the same experiments but plotting the alive time for PacMan.



```
      X0              X1              X2              X3              X4              X5
 Min.   : 997   Min.   :1302   Min.   :1045   Min.   :1135   Min.   :1351   Min.   : 739
 1st Qu.:1046   1st Qu.:1465   1st Qu.:1172   1st Qu.:1207   1st Qu.:1364   1st Qu.:1292
 Median :1375   Median :1621   Median :1499   Median :1582   Median :1425   Median :1549
 Mean   :1355   Mean   :1669   Mean   :1672   Mean   :1521   Mean   :1472   Mean   :1526
 3rd Qu.:1555   3rd Qu.:1657   3rd Qu.:1698   3rd Qu.:1744   3rd Qu.:1533   3rd Qu.:1658
 Max.   :1800   Max.   :2300   Max.   :2945   Max.   :1937   Max.   :1687   Max.   :2393
      X6              X7              X8              X9             X10             X11
 Min.   : 604   Min.   :1178   Min.   :1164   Min.   :1308   Min.   :1224   Min.   : 981
 1st Qu.: 862   1st Qu.:1207   1st Qu.:1381   1st Qu.:1485   1st Qu.:1316   1st Qu.:1469
 Median :1059   Median :1265   Median :1498   Median :1682   Median :1451   Median :1658
 Mean   :1057   Mean   :1333   Mean   :1577   Mean   :1711   Mean   :1536   Mean   :1551
 3rd Qu.:1290   3rd Qu.:1329   3rd Qu.:1729   3rd Qu.:1772   3rd Qu.:1555   3rd Qu.:1815
 Max.   :1469   Max.   :1684   Max.   :2112   Max.   :2308   Max.   :2134   Max.   :1833
     X12             X13             X14             X15             X16             X17
 Min.   : 646   Min.   : 869   Min.   :1407   Min.   :1167   Min.   :1213   Min.   :1334
 1st Qu.:1224   1st Qu.:1148   1st Qu.:1476   1st Qu.:1275   1st Qu.:1390   1st Qu.:1434
 Median :1272   Median :1197   Median :1548   Median :1438   Median :1401   Median :1466
 Mean   :1269   Mean   :1172   Mean   :1638   Mean   :1402   Mean   :1615   Mean   :1446
 3rd Qu.:1598   3rd Qu.:1234   3rd Qu.:1581   3rd Qu.:1448   3rd Qu.:1678   3rd Qu.:1485
 Max.   :1604   Max.   :1412   Max.   :2179   Max.   :1684   Max.   :2393   Max.   :1511
     X18             X19             X20             X21             X22             X23
 Min.   :1276   Min.   : 936   Min.   :1135   Min.   :1016   Min.   : 758   Min.   : 852
 1st Qu.:1443   1st Qu.: 975   1st Qu.:1151   1st Qu.:1274   1st Qu.:1153   1st Qu.:1102
 Median :1556   Median :1204   Median :1397   Median :1472   Median :1667   Median :1266
 Mean   :1566   Mean   :1195   Mean   :1476   Mean   :1536   Mean   :1726   Mean   :1220
 3rd Qu.:1697   3rd Qu.:1248   3rd Qu.:1442   3rd Qu.:1801   3rd Qu.:1959   3rd Qu.:1283
 Max.   :1858   Max.   :1611   Max.   :2254   Max.   :2115   Max.   :3095   Max.   :1598
     X24             X25             X26             X27             X28             X29
 Min.   : 975   Min.   : 847   Min.   :1240   Min.   :1194   Min.   :1093   Min.   :1195
 1st Qu.:1262   1st Qu.: 852   1st Qu.:1465   1st Qu.:1217   1st Qu.:1098   1st Qu.:1309
 Median :1331   Median :1114   Median :1587   Median :1562   Median :1305   Median :1412
```



Conclusion:

After implementing the strategies as described in the first half of this report, we are confident that we were able to increase the score of the game to a maximum of 7000 on an average. However, this may not be the best score achieved, but it is definitely better than the original score which was in the range of 3000-3500. We can conclude that, how search algorithms are useful in finding solutions in the search space and solve optimization problems.