MULTI AGENT SYSTEM WITH ARTIFICIAL INTELLIGENCE

Sai Raghunandan G

Master of Science

Computer Animation and Visual Effects

August, 2013.

Contents

Abstract:

Artificial Intelligence in such is a vast area with various branches and methods to implement. Simulation of an environment where there are different agents interacting communication and responding comprises a multi agent system. Implementing such a system with robust communication system taking care of the message encapsulation and bandwidth has always been a challenge.
This project is one of those attempts to simulate an environment with multiple agents with different AI behavior. A simple pong game was implemented as a setup to the system and finally a simple maze game with the multi agent environment is simulated.

Chapter 1

Introduction

Problem Statement

To simulate an environment with multiple agents having Artificial Intelligent behaviour that interact with the environment and respond accordingly.

Artificial Intelligence has always a fascinating and challenging field. Its applications find roots in robotics, games and many other real world applications. There are different types of artificial intelligent systems which range from a completely self-learning system with least human interference and partial artificial intelligent systems with substantial human inputs.

Multi-Agent system on the other hand is an emerging system , with its applications spanning through gamut of fields, has been the topic of widespread research lately(Wooldridge 2004). Simulation of multi-agents with artificial intelligence has always been a challenging task to developers. My project chooses one such scenario to simulate a multi-agent behaviour using artificial intelligence. The behaviours are written in a different scripting language. The basic system set up is tested using a simple pong game, and later the system is developed to simulate another game with some more complexity involved in the behaviours of the agents.

Structure

The next section discusses the related work in this field, after a brief explanation of the technical background and the algorithms involved. The later section discusses the actual implementation of the project and the issues faced. Finally the discussion is concluded mentioning about the applications, results, summary and future work.

Chapter 2

Related Work

AI has been an interesting field of research right from the advent of early video games. As the complexity in the games started to improve the behaviours of the game agents evolved simultaneously. One such efficient game of early times which define simple yet powerful multi agent behaviour is Pac-Man(Pittman 2011). Though the AI behaviour is limited in that, it defines a very simple and interesting AI behaviour.

The recent work in the related field includes robotics, where different AI behaviour are researched and implanted. The field of robotics and the AI behaviour in using them can provide real time solutions to problems. Another field which carries the same context of simulation in which multiple agents interact and behave is the flocking system and crowd simulations. These are artificial intelligent systems with a large numbers of agents interacting and responding according to the environment. The other fields which are more profound in the same context are genetic programming and machine learning along with fuzzy logic algorithms which can define much more complex and interesting AI behaviours where the systems or the agents evolve and get to the solution.

Chapter 3

Technical Background

The following section will describe the technical aspects that are involved in the project along with the introduction to the algorithms that were implemented in the project

3.1 The Code

The entire project structure is divided into two segments. One segment is developing the system which contains the environment and handles the interaction and communication between the agents in it, and the other is the behaviors of the agents itself. The two segments were developed in two different programming languages enabling a more flexible system.

3.1.1 The AI System in C++

The environment is an important part in any Artificial Intelligent system. It should be capable of handling multiple-agents with different behaviors at the same time and must enable the agents to communicate among them and with the environment. The interaction with the environment is the key part which decides the resulting behavior of the agent. So the communication system should be well established with structured message passing and receiving settings. To conceive such a system in terms of coding a well suited programming language is required which can allow a better classification of the system entities. One of the high-level languages that is well suited for such a scenario is C++.The system engine is written in C++ as its object oriented programming enables easy classification of the classes and better handling of the memory. It is very efficient for graphics and also a platform independent programming. The communication and organization of different classes can be well structured. Each system entity is divided into a class of its own with its specific attributes and functions which will be discussed in detail in the following sections. The data abstraction and encapsulation features of it help to conceal the important attributes of the system entities and at the same time expose the essential features of the objects. And the message passing capability gives ease of communication.

3.1.2 Scripting in Python

The next important segment is the behavior of the agents. The behaviour should be simple to write, easy to read and should be flexible to change frequently without interfering with the main system engine. To enable such features Scripting is chosen to write the behaviors. Scripting can be defined as a simple programming language which can help to customize a specific task in any context in a simple way. It enables to write and refine the mechanics of the systems (Bourg and Seeman 2004).The C++ reads the behaviour from the scripting language. Normally scripts are run from within the program by Virtual machines. They enable the communication between the virtual machine and the C++ in which it is wrapped which makes the process of exporting and importing the data from the script easy(Buckland 2005).The other advantages of scripting language as explained in (Buckland 2005) are

1) They can be read easily from initialization files

2) The compile time can be avoided which saves time and helps increasing productivity

3) Its high level language uses syntax which can be easy to non-programmer to interpret and make changes if necessary in a development environment.

4) The behaviors written can be expanded easily whenever required increasing better extensibility.

The scripting language that is used in this project is Python. Python is a high level language which is easy to read and code. It has good interfacing compatibility with C++. The data from C++ is exported to the script in form of discreet variables, tuples or lists and imported back to C++ again after the processing from the script is done. This easy exchange of data helps if simplifying the interactions between the two languages. It is platform independent as it is converted into an intermediate code before the virtual machine executes it. Procedural code can be expressed in a rather natural way. The dynamic data types relieve the user from type testing each time he declares a variable. All these features make python a very suitable language for scripting, and it was used to define the Artificial Intelligent agent behaviours in this project.

3.2 Environment

As stated above, in any context where Artificial Intelligence is implemented, two factors that define the perception of Intelligence are the environment around and the other is the interaction of the agent with the environment.

3.2.1 Pong Game

In the pong game the interaction with the environment is not very significant. The only interactions involved are to check the collision with the bounding wall, the collision of the agent ”Ball” with the players and to check in which agents direction is the agent ”Ball” heading.

3.2.1.1 Collision with the Wall

A Sphere-Plane distance logic is used to calculate the distance between the wall and the agent (Weisstein 1999). Figure 3.1.1 shows the position of the point, the position of the plane and the normal from the plane. x_0 = (x_0 , y_0, z_0) is the point from which the distance is calculated, $v = (a, b, c)$ is the normal of the plane and $ax + by + cz + d = 0$ is the plane equation. Considering a point $x = (x, y, z)$ on the plane, the vector from x_0 to x is given by

$$w = -\begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix}$$

By projecting the point w onto v gives the distance from the point to the plane and dropping the absolute value signs gives the signed distance

$$D = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \qquad (1.1)$$

Given the radius r of the agent mesh the collision of the agent with the wall is identified when

$$D < r$$

When the collision is detected the response is different for each agent. The player agents when collided with the wall their velocity is made zero in Z axis so that they don't move further.

$$\begin{aligned}&\textbf{if } wallCollision \textbf{ then}\\&\quad velocity.z = 0\\&\textbf{end if}\end{aligned}$$

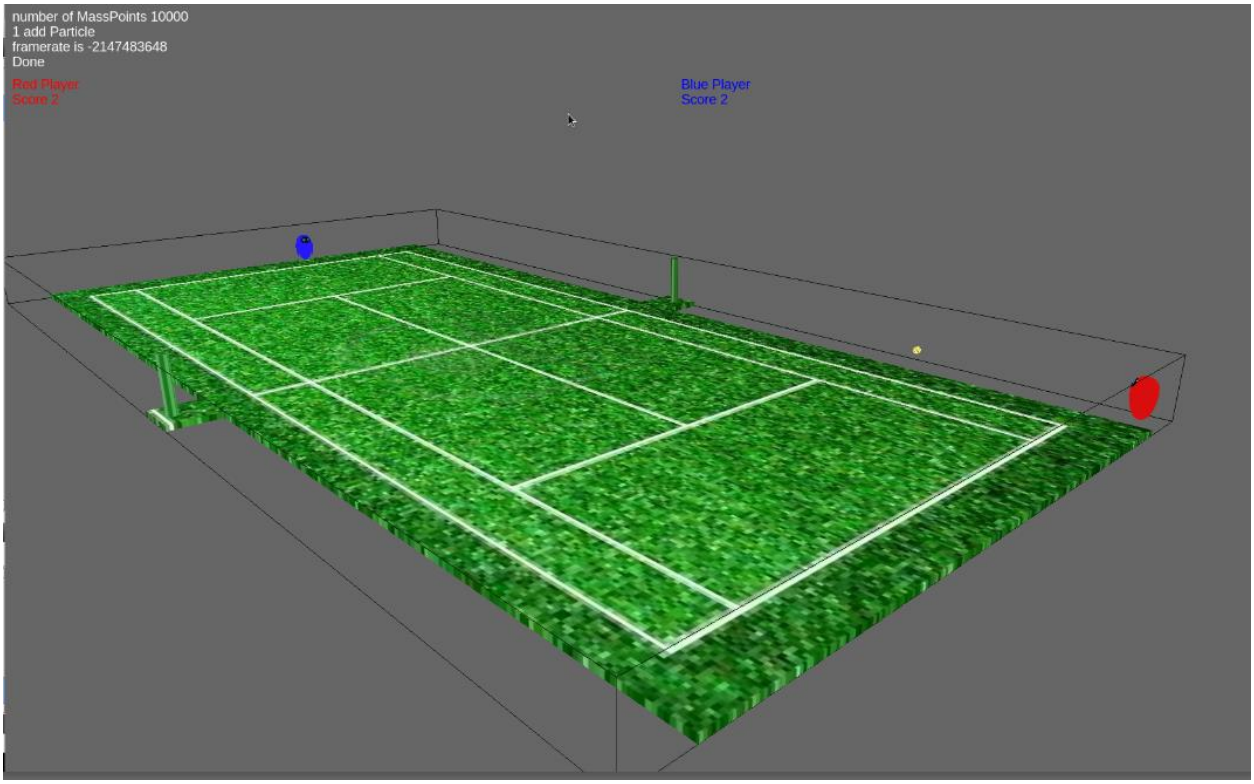

Figure 3.2.1 The Collision of the Ball with the wall

The collision response for the ”Ball” agent is such that if the wall it hits is in positive x direction or negative x direction, it means it has crossed the agent playing there. So it sets the reset status to an integer greater than zero which resets its position to the center of the court. Depending on the value of the reset status the score of the players is updated. If the wall is in either positive or negative z axis the “Ball” agent will reverse

```
if wallCollision then
  if wallNormal.x == −1 then
    reset = redPlayerReset
  else if wallNormal.x == 1 then
    reset = bluePlayerReset
  else
    velocity.z = −velocity.z
  end if
end if
```

3.2.1.2 Collision of an Agent with other Agent

The collision between the agents is detected using the sphere to sphere collision detection method. The logic used here is that the distance between the centres of the two spheres should be greater than the sum of the radii of the two spheres. If r1 and r2 are the radii of the two spheres and the P and Q are their centers respectively, then the condition to check for the collision is given by

$$(P - Q)^2 > (r_1 + r_2) \qquad (1.2)$$

This collision between agents is checked only for the “Ball” agents. When the collision is detected the “Ball” agent just reverses the velocity direction in x axis. To include a random nature in the force with which the “Ball” agent bounces back, another simple logic is implemented. The computer ticks are taken into track and each time they elapse after a particular interval, the magnitude of the velocity of the “Ball” agent in its corresponding direction is increased.

```
if agentCollision then
  if computerClock mod interval > interval−offset and computerClock
  mod interval < interval then
    int x = abs (velocity.x)
    x+= random (range)
    if velocity.x < -x then
      velocity.x = -x
    else
      velocity.x = x
    end if
  end if
  velocity.x = −velocity.x
end if
```

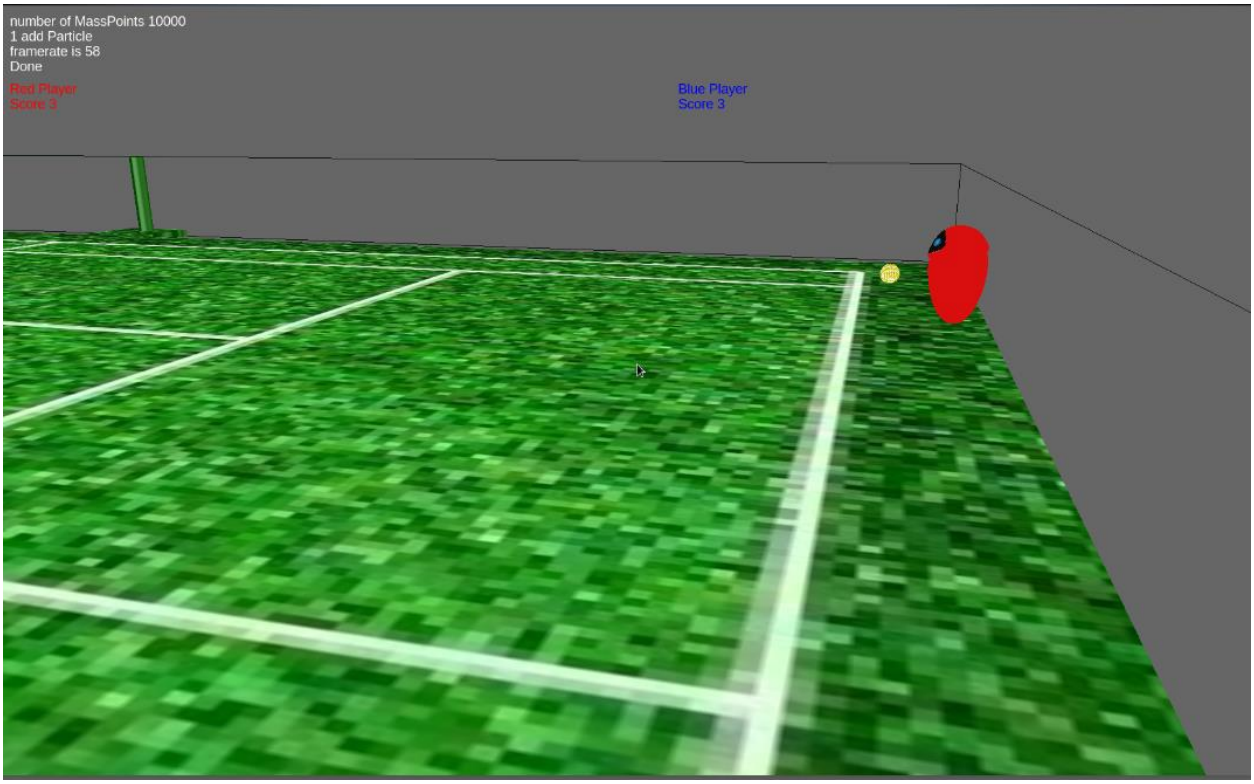

Figure 3.2.2 The collision of one agent with the another

3.2.1.2 Approaching Agent

The players have to check if the agent “Ball” is approaching them so that they can adjust their position in accordance to the position of the “Ball” agent. To be able to detect that a simple logic is implemented which checks the dot product of the velocity of the player agents with the velocity of the “Ball” agent. If the dot

product is negative it means the player agent and the “Ball” agent are facing each other and vice versa. The figure below shows the scenario where one agent remains unmoved when only the agent in which the Ball is headed to moves.

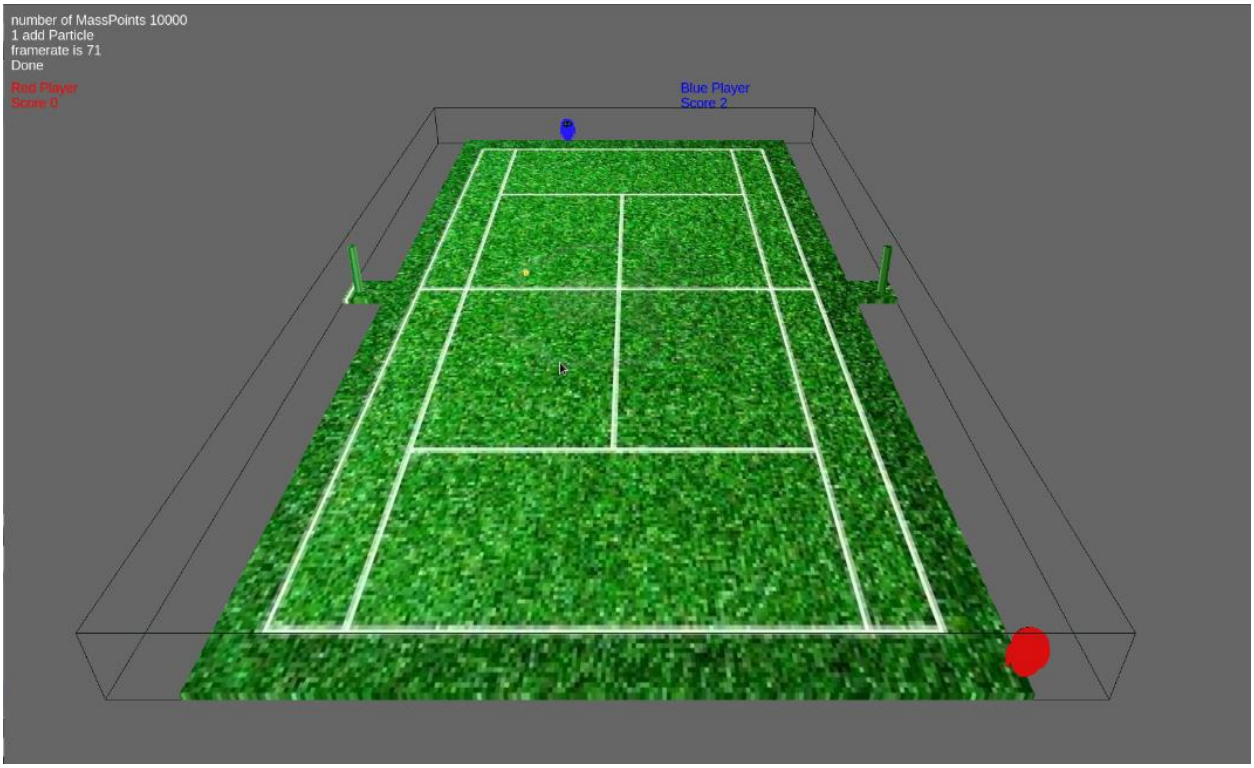

Figure 3.2.3 The Approaching agent.

3.2.2 Maze Game

The maze game has more interaction between the environment and the agents. There is message passing, maze solving, and path finding algorithms that are implemented. A graph network for the map traversal and collision detection depending on the number of nodes occupied by each agent in the graph is used.

3.2.2.1 Graph Network

The environment in the maze game is composed of a graph network with nodes having indices. A given node has an occupied status which contains the number of agents that are occupying that node. A node can have an obstacle placed in its position. At any given point it can have a minimum of three neighbouring nodes to a maximum of eight neighbouring nodes. An agent on a particular node can chose between its neighbouring nodes depending on the state and conditions. Each agent traverses only one node at a time while moving in the maze. The agents are short sighted in the sense they can think only about the next tile traversing along the direction of present node and will decide the direction after reaching the present node. With this pretext the algorithms that are used will be explained.

3.2.2.2 Maze Solving Algorithm

In the maze game, the agents have different states. One of those states is the ”Random” or ”Search” state in which they follow the maze solving algorithm to traverse the graph network. The wall follower rule is implemented for the maze following. The rule is that the agent always tries to follow the wall and makes sure

that the wall is always on its right side. The graph network has four directions associated with it. East, West, North, South. The three rules in the wall following are

1) If there is no wall on the right side Turn Right

2) If there is a wall on your right and no wall in front of you Go Straight

3) If there is a wall on your right and wall in front of you Turn left

Initially the agents are placed in a direction heading east. They continue in the same direction till they encounter a wall. The wall is an obstacle which is placed on a given node position in the graph network. So each node is checked if an obstacle is present in its position or not. Once they encounter a wall, they follow the wall following logic. The algorithm is given below

```
if nextNodeIsFree then
  position = nextNodeposition − presentPosition
  presentPosition+ = position
  velocity = nextNodeposition − presentPosition
  if (position)^2 < 0.1 then

if not initialWallFound then
  if nextNodeisfree then
    direction = east
  else
    direction = north
    initialWallFound = true
  end if
end if
if initialWallFound then
  Switch(direction)
  Case: east
  if nodeOnSouthisfree then
    direction = south
  else
    direction = east
  end if

  Case: west
  if nodeOnNorthisfree then
    direction = north
  else
    direction = west
  end if
```

Case: $north$
if $nodeOnEastisfree$ **then**
$direction = east$
else
$direction = north$
end if
Case: $south$
if $nodeOnWestisfree$ **then**
$direction = west$
else
$direction = south$
end if

end if
end if
else
$intialWallFound = true$
Switch($direction$)
Case: $east$
$direction = south$
Case: $west$
$direction = north$
Case: $north$
$direction = east$
Case: $south$
$direction = west$
end if

This algorithm is implemented in python script. The graph network data is exported to python and the new direction and node position are read from the script. Figures 3.2.1, Figure 3.2.2, Figure 3.2.3 show the wall following of a green agent.

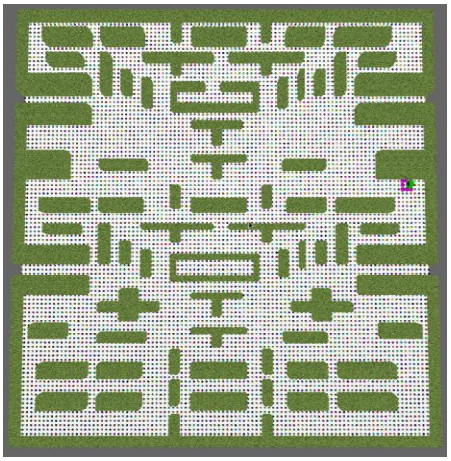

Figure 3.2.1

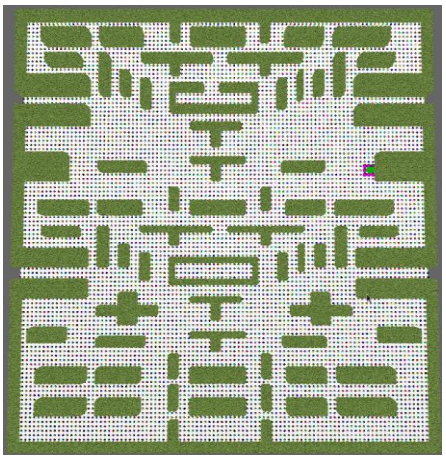

Figure 3.2.2

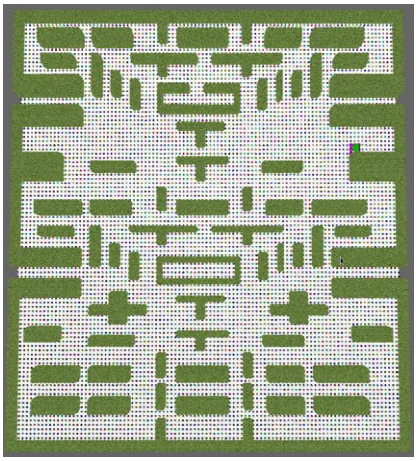

Figure 3.2.3

3.2.2.3 Collision Detection

The collision detection in the graph network is based on the fact that at a time a given node and its neighboring nodes should be occupied only by a single agent. If the number exceeds, i.e., if the node is occupied by more than one agent, then it is discarded and the next neighbouring node is taken into consideration. This is a simple approach when the size of the agents is standard. This doesn't require special collision detection methods. Every node has an attribute called occupied status, which holds an integer value of the number of agents that are accessing that node. Each time an agent traverses the graph and sets a new node, that new node and all of its neighbour's occupied status is increased by one. At the same time the previous node and the previous node neighbours occupied status is decreased by one as the agent has left that node and entered a new node. In this way the nodes occupied status is set and de-set as the agents leave the previous node and move to next node. So in each loop when the node is checked if its available to traverse or not the occupied status condition should be checked in a way that it should be less than or equal to one to be able to traverse through in addition to the condition that, the node doesn't have an obstacle on it. In this way the collision detection is checked for the maze game. This detection is made only when the agents are in "Chase" state. In the "Random" or the "Search" state, the occupied status of the nodes is not set. The agents only recognize the node that the user is traversing through but not the nodes that the other agents are traversing.

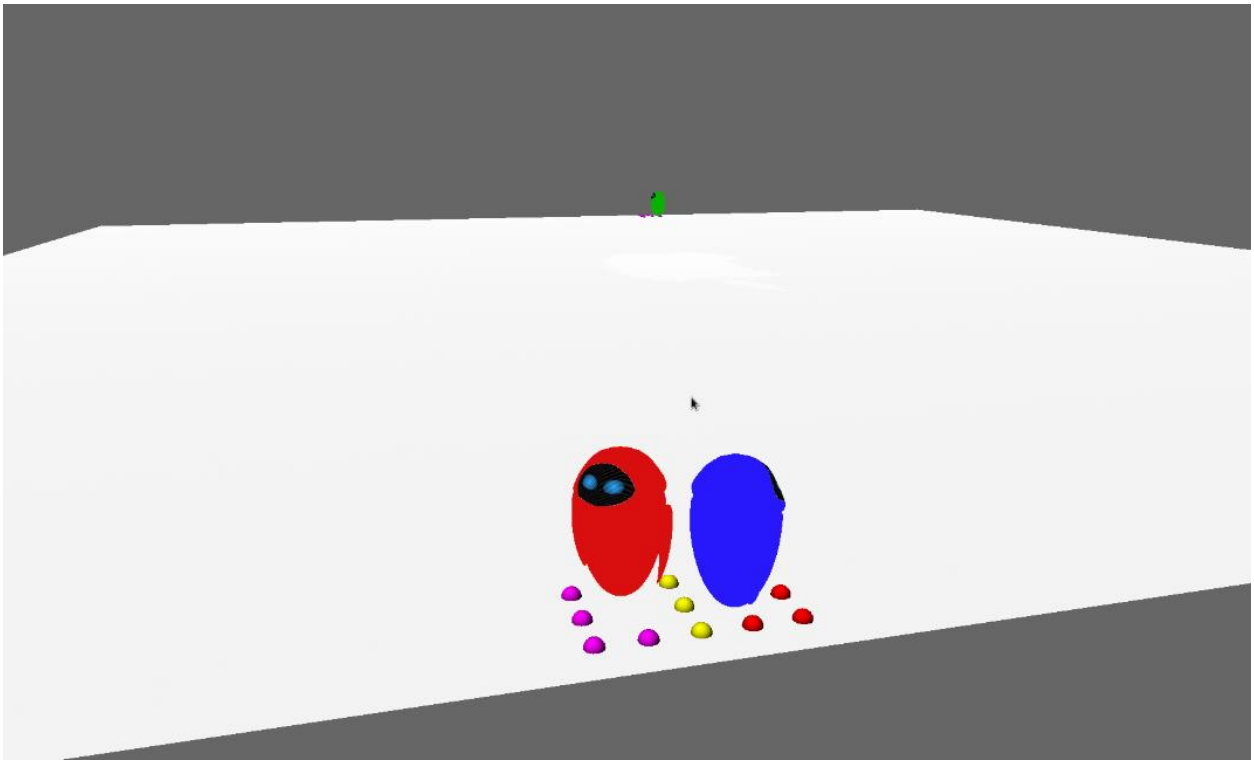

Figure 3.2.4 Collision Detection between two agents. The yellow nodes indicate the occupied status is greater than one.

3.2.2.4 Path Finding Algorithm

The next algorithm the agents follow is the path finding algorithm when the agents are in “Chase” state. In this state the AI agents follow the user agent. For them to follow the user agent, they need to know the shortest possible path. Each agent is allocated a target position. This can be a moving target or a static target. Static target is the home position of the agents. That is in the “home” state they go back to their home positions which are the positions that are initialized during their creation. Dynamic target is the position of the user, which the AI agents chase. The shortest path is calculated on the fly. The AI agent’s present node has neighbouring nodes connected to it. Out of all the neighbours the neighbours that are free to traverse are selected i.e., the nodes without obstacles on them. From the selected set of the neighbouring nodes, the linear distance between the target position and the position of the each node is calculated. Then the node with the shortest distance between the target and the agent is selected and that node becomes the next node for the agent for traversal. The same logic is used when the target is stationary or when the target is moving. In the course of the selection of the shortest distance node, sometimes the previous node is selected as the next node, and this result in the agent moving back and forth in the same position. To avoid this toggling behaviour an extra condition is written such that if the next selected shortest path node is same as the previous node of the agent or if it’s any of the nodes that are pointing backwards it is not used. The next node with the shortest path is taken. The agent should try going forward always. Only in a position where the only possible node that is available is the node that is pointing backwards, then that is taken. A priority queue is used for storing the shortest distance from each node along with the index of the node, so that it sorts the data by itself and when

it pops out the data, we get the node that has the shortest distance from the target. The algorithm is given below.

```
if nextNodeIsFree then
  position = nextNodeposition − presentPosition
  presentPosition+ = position
  velocity = nextNodeposition − presentPosition
  if (position)^2 < 0.1 then
    for every node in the neighbourslist do
      if theNextNodeIsFree and occupiedStatus <= 1 then
        position = targetPosition − nodePosition
        enqueue (position)^2 and nodeIndex in priorityQueue
      end if
    end for
    if not(QueueIsEmpty) then
      nextNode is pop prioirtyQueue
      if nextNode is pointing backwards then
        nextNode is pop prioirtyQueue
      end if
    end if
    node = nextNode
  end if
else
  for every node in the neighbourslist do
    if theNextNodeisfree and occupiedStatus <= 1 then
      position = targetPosition − nodePosition
      enqueue (position)^2 and nodeIndex in priorityQueue
    end if

    end for
    if not(QueueIsEmpty) then
      nextNode is pop prioirtyQueue
    end if
    node = nextNode
  end if
```

In this way, the different states in the AI follow two different algorithms to traverse the graph network.

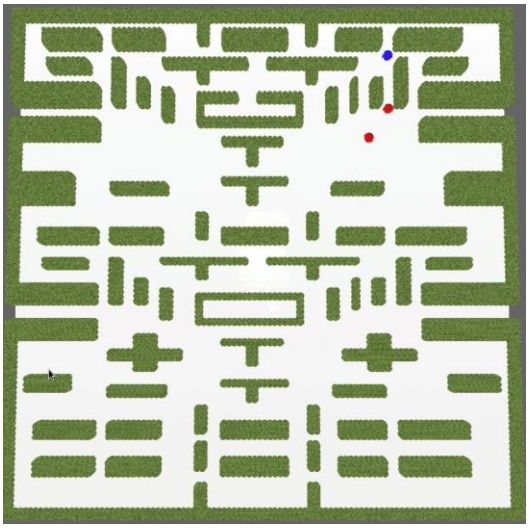

Figure 3.2.5 The Red agents following the Blue agent using the path following algorithm

4 Implementation

4.1 Setting up the System

The main objective of the project is to build a system which supports a multi-agent behaviour in which the agents interact with the environment and depending on the response from the environment their behaviour is decided. The system should support message passing among the agents. The behaviour is written in a scripting language so that it can be tailored as per the need. The next goal is to simulate a game environment using the system which can display the behaviors and interactions. To set up the initial system a simple behavior and a simple game Pong is chosen. After successfully implementing the Pong game, the system is further developed which supports better interactions and a maze game is implemented to demonstrate the same.

4.2 The Map

To start with the system a suitable map must be chosen. While developing the initial setup for the pong game, no map is created but an OBJ file from Maya has been exported to create the court for the game. There are no obstacles when it comes to the pong game, except for the walls of the bounding box of the court. In the maze game a more detailed and structured approach is followed to create the map and the environment. The maze game is based on a graph network, where the agents traverse on the nodes of the graph. To enable a flexible way of defining the maze for the game environment, the structure of the maze is read from a text file, which is done by the parser class. The text file contains a n x n rows and columns of 0's and 1's. Zeroes are the positions where there are no obstacles whereas one's are the positions where there are obstacles. The data from the text file is stored in the form of a vector of integers of zeros and ones. When the nodes in the graph network are created, they are created with the same dimensions of nxn each node has an attribute onOff which indicates if a node contains an obstacle or not. If the value of onOff is zero, it means the node is free from obstacle, but if the onOff value is one it means that there's an obstacle on the node. To assign the onOff status to the nodes in the graph, the obstacles data from the parser class is sent to the graph class. By iterating through that data from the parser class, if a given index has the value '1', the onOff status of the node at the corresponding index in the graph class is set to '1'. After assigning the onOff status to all the nodes in the graph class, the corresponding positions of the nodes with their onOff status '1' are sent to the map class. The map class stores the positions of these nodes which have an obstacle at that particular position, and it draws the obstacle in each iteration. In this way, in the initialization of the program, the obstacles data from the text file is read, the corresponding obstacle positions data is read from the graph data and is passed to the map class which draws the obstacles. The map class contains the required attributes to hold the data for drawing the obstacles.

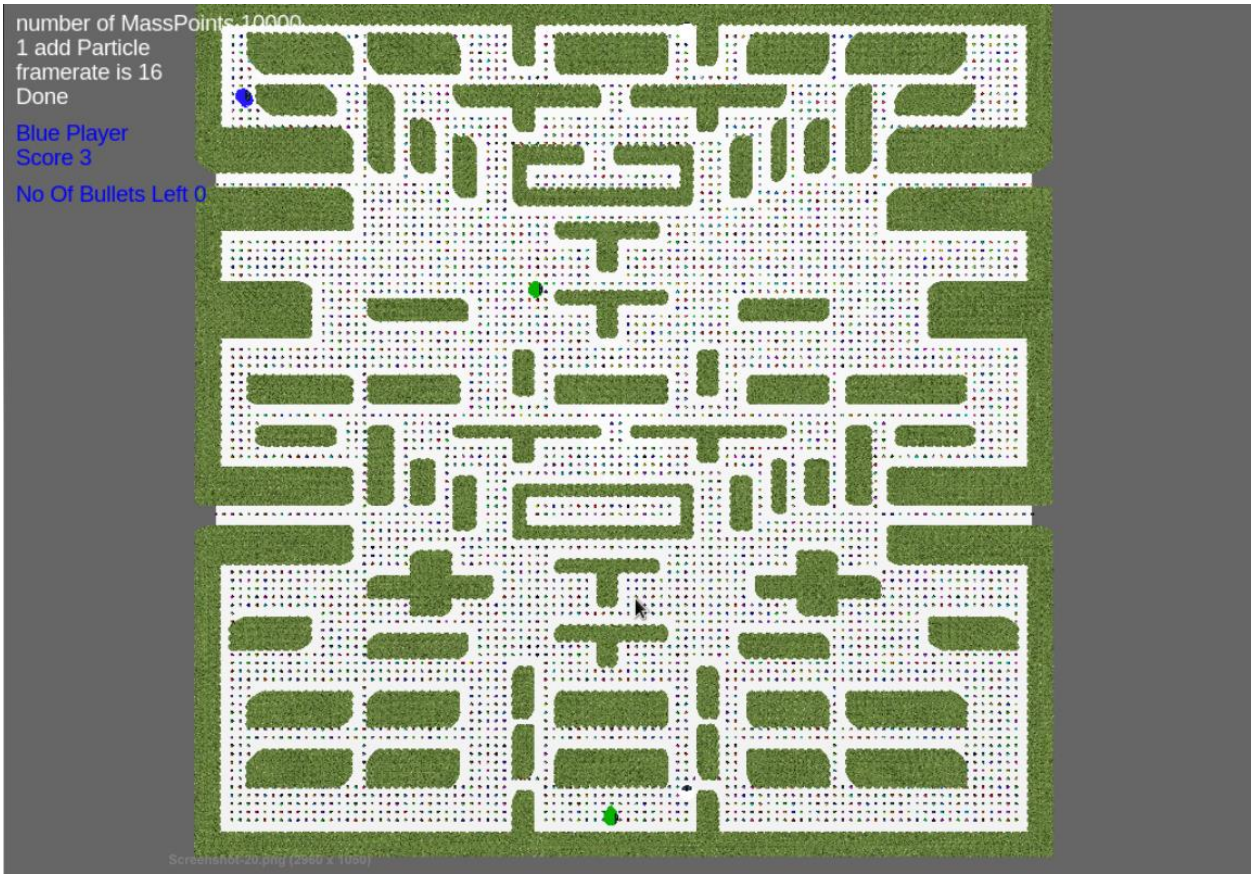

Figure 4.1.1 The Map with the obstacles and nodes.

4.3 The Graph

The graph network is one of the important parts of the maze game. It comprises the grid on which the agents move. It is the basis for the maze solving and the shortest path algorithms that are used. The graph data is maintained by the graph class, which has the the Graph nodes and Graph edges as its member variables.

4.2.1 The Graph Nodes

Graph nodes are the primitives which form the graph network. This data is contained in the GridNodes class. The nodes are divided into two dimensions width and depth which is calculated from number of entries from the text file that are read from the parser class. Each node has an index, position attributed to it. Each node is connected to its neighbouring node by an Edge. A given node can have a maximum of three nodes to eight nodes as its neighbors. An agent traverses from node to one of its neighboring node along the edge connecting them.

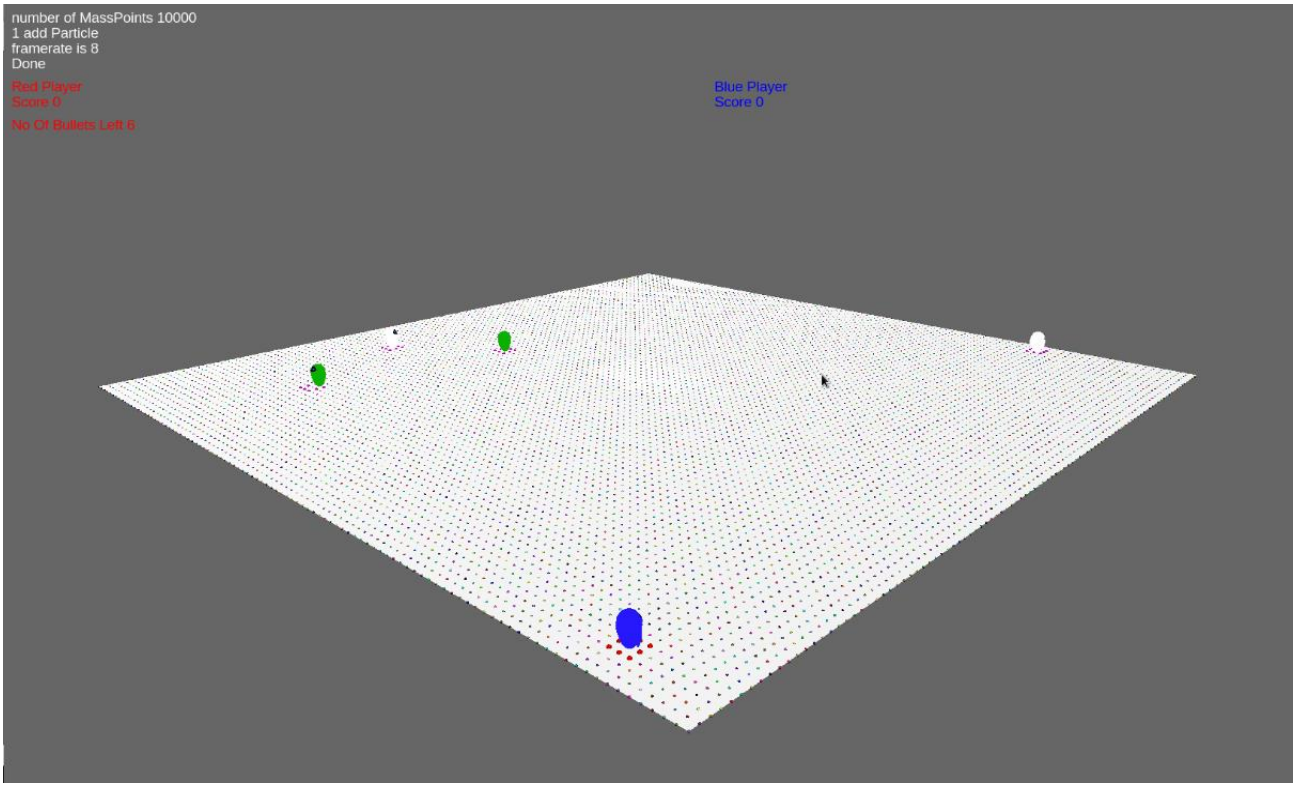

Figure 4.2.1 The Grid Nodes and the agents traversing on them

4.2.2 The Graph Edges

The edges in the graph are the connections between the nodes. The have a to-index and from-index and the cost which is the distance between the nodes associated to it. For a given node all the edges are calculated and stored during initialization. If the node is located in a given position (x,y) the possible nodes that share the edge with this node are

1) The node at (x+1,y)

2) The node at (x,y+1)

3) The node at (x+1,y+1)

4) The node at (x+1,y-1)

5) The node at (x-1,y)

6) The node at (x,y-1)

7) The node at (x-1,y-1)

8) The node at (x-1,y+1)

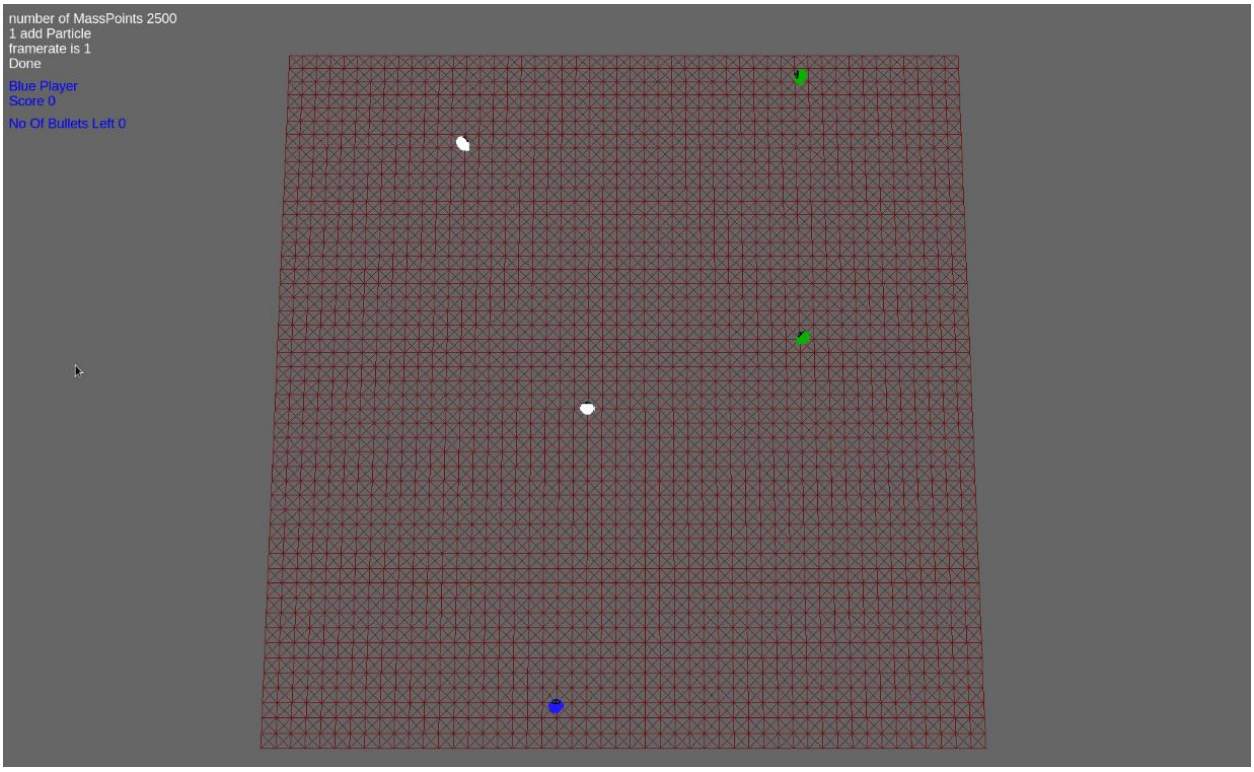

Figure 4.2.2 The Grid Nodes connected by their corresponding Edges

Any agent on a given node has to choose among these neighbouring nodes for its traversal depending on the state. The GraphEdge class contains the related to the edges, which is a member variable of the Graph class. The graph class takes care of creating the nodes and assigning the edges and passing the information to the environment class depending on the requirement. The graph network is an efficient way of representing a map, which enables ease of movement to the agents moving in it. The obstacle avoidance and the collision detection can be included in the net- work itself without having a special algorithm for detecting the collisions and avoiding them. As mentioned previously, whenever the onOff status of a node is set to '1', it is removed from the set of nodes that are available for traversal, as a result navigating through the graphs becomes easy and the obstacle avoidance becomes the integral part of the nodes traversal itself. The same logic can be applied to the collisions between agents as well. As mentioned earlier more than one agent cannot be on a given node at a time. This makes the collision detection between them easy. In addition to these advantages, the path finding and maze solving algorithms are easy to be implemented on a graph node. Because of these advantages and convenience in their structure the graph is used for the maze game.

4.3 The Environment

The environment is the most important part of the system. All the functions and attributes of the environment are taken care by the environment class. The environment is responsible for the interaction be- tween the agents and with itself. It manages the AI agents and the user agent. The AI agents on their own cannot perceive things in the environment. In each iteration the AI agents query the environment different questions like the collision detection, approaching agent's message from the other agents. Depending on the response from the query, their behaviour is decided. In case of the pong game, the "Ball" agent always queries the environment

to check for the collision with the bounding wall and the collision with the agents. Depending on the query response, its behaviour is decided. The player agents have to check the collision with the bounding wall as well and along with that, they have to know if the ball is approaching in their direction. These are queried and the environment replies them back with the appropriate reply. In the maze game, the AI agents have to know if the user agent is within its vicinity. If the user comes with in the vicinity, they change their state and they pass the message to the other agents to change their state. This communication has to go through the environment. The environment replies to the query in each iteration. More over messages from one agent to other are sent through the environment in each iteration. The environment class also updates the target position of the AI agents of the maze game with position of the user. When the agents are chasing the user agent, this target position is used to calculate the shortest path to reach the user. The environment also takes care of passing the information from the text file to the map class, for the obstacle generation and then the graph class to process the nodes and pass the information to the agents. In this way the environment plays a very significant role in this system.

4.4 The Agents

The next important part of the system is the agents. The agents are classified into AIagents and UserAgents which is taken care by the agent's class. The environment class communicates with the agents class for any information regarding agents. All the required functions for creation and processing the agents are present in the Agents class.

4.3.1 The Artificial Intelligent Agents

The AIagents class contain all the artificial intelligent agents that are present in the system. In the Pong game, the two types of agents that are present are the "Player" agents which hit the ball, and the "Ball" agent which moves over the court. In the maze game the types of agents are "Army" and the "Scout". The Scout is the leader agent to the army agents. Each artificial intelligent agent has a brain associated with it. Brain is an object of the class Brain. Each type of agent has a script associated with it, which is passed to it at the time of creation of the agents. The type of the agents is also decided at the time of creation. In the maze game, the agents are created with an army ID. Each agent belongs to an army, and each army has a single Scout which is the leader of the army. The army agents follow the message they receive from the Scout agents. These messages are passed by the environment. The state of the army agents depends on the message they receive from the Scout. The brain object of the AI agent is the main class which does the exchange of data for the AI processing. It collects the data from its parent i.e., the AI agent class. The AI agent class queries the environment through the Agent class in each iteration and that data is used by its brain class. The brain class converts the data from C++ to a form which can be read by python, bundle the data into Lists and tuples along with the single variables, export it to python. Once the python script is run the resultant values are read back from python converted into the form that is readable in C++ and are passed back to the AI agent.

4.3.1.1 Python

The actual scripting of the behavior is done in Python. This is a normal text file. The system variables in this are set by the brain class and are passed to this. Once the variables are set, the script utilizes them manipulates them depending on the state, and this data is read back by the Brain class.

4.3.1.2 States

The agents in the Pong game do not have any states. They move along the Z axis of the plane tracking the movement of the ball in Z axis. Whereas the agents in the maze game have three states associated with them 1) Random: In the random state, the army agents move along the maze wandering using the maze solving algorithm. They traverse along the plane following the wall. The collisions are not enabled in this state. The same state is named as Search for the Scout which wanders around.

2) Chase: In the Search state of the Scout, if it comes across the user, the Scout changes its state from Search to Chase and passes a message to the army agents to Chase the user. As they receive the message Chase from the scout, they change their state from Random to Chase. In chase mode the agents follow the shortest path algorithm trying to chase and hold the user.

3) Home: In Home state, they follow the same shortest path algorithm. The only difference is that instead of following the user position as tar- get they follow the home position which is the initial position they are created during their creation. The communication between the environment and the agents and the response from the environment decides the state and the message received by the agents.

4.3.1 The User Agents

Another category of agent is the user agent which is completely con- trolled by the user. The UserAgent class takes care of the functions and attributes of the user agent. The user agent is created in the agent class and initialized. The controls of the user agent are maintained by the UserControls class. This receives the input of keyboard press events, processes it and sends the corresponding control response to the UserAgent class. The user agent then moves along the grid depending on the input. The user can move in any direction including diagonal traversing. It has to collect points wandering around the maze trying to avoid the “white” agents which are scouts. It can collect bullets to shoot at the agents in its defence. All these controls are read from the keyboard.

4.4 The Game Entities and Models

The Game entities are the part of the game, which the user has to collect as a part of the game. These game entities are either Coins which boost up the score of the user and bullets which he has to collect in order to shoot the agents. The game entities are stored in the Game entities class which distributes the entities randomly over the grid. All the meshes for the game are stored in a class called models which keeps a track of each mesh and its texture. All these meshes are loaded and stored in the models class during the initialization of the system. Each class has a reference to this models class. Each agent can get its related mesh from the models class to draw.

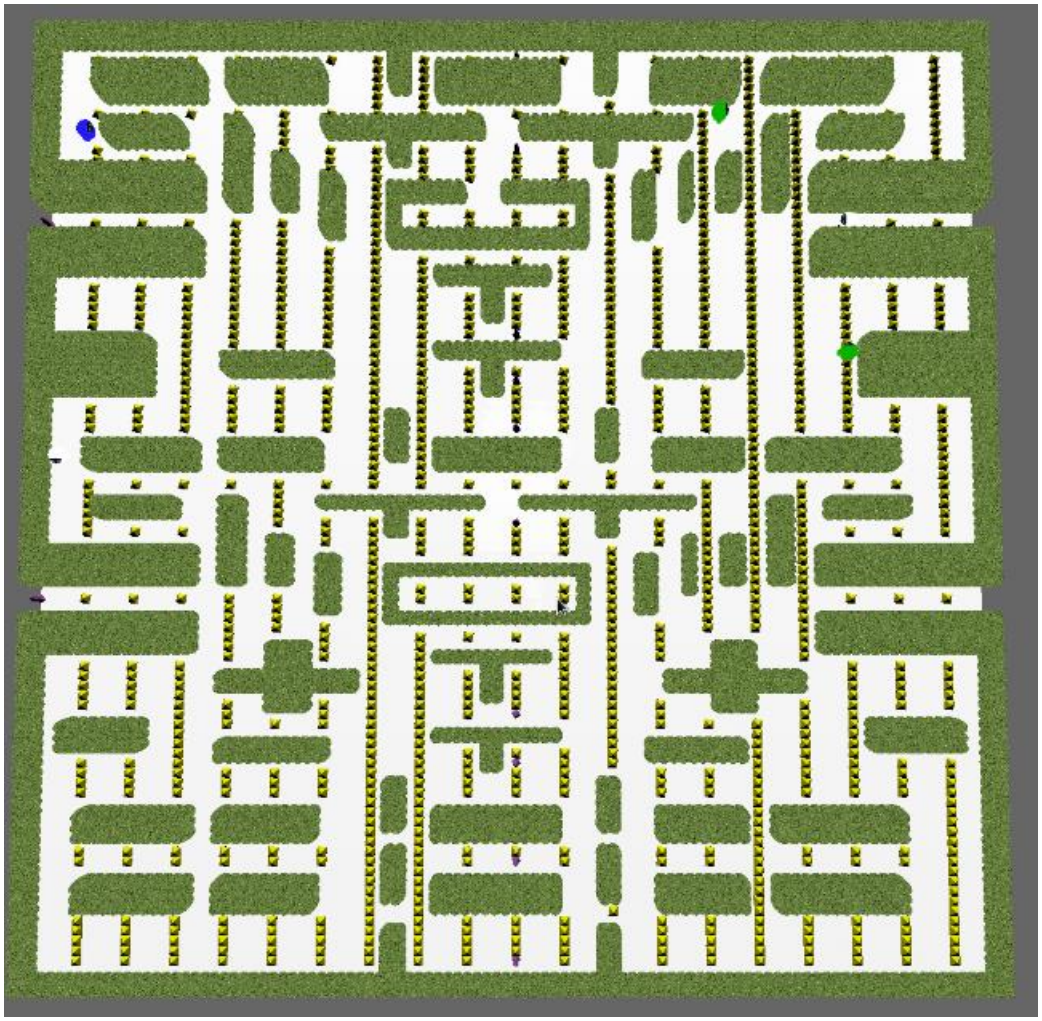

Figure 4.4.1 the complete map with the game entities.

4.5 Game Play

4.5.1 The Pong Game

The Pong game is completely independent. No user input is involved. The RedPlayer and BluePlayer move along the court in the positive and negative z axis trying to track the Ball agent that is coming in their direction. The environment intimated the players if the Ball agent is approaching their direction and passes the position of the Ball agent to the players. Depending on this information the players adjust themselves to hit the Ball coming their way. Once in a particular interval of time, the velocity with which the player hits the Ball increases by a constant value and once that interval elapses the Ball agent moves with the same normal velocity. This is included to involve the uncertainty in the speed with which the Ball agent bounces.

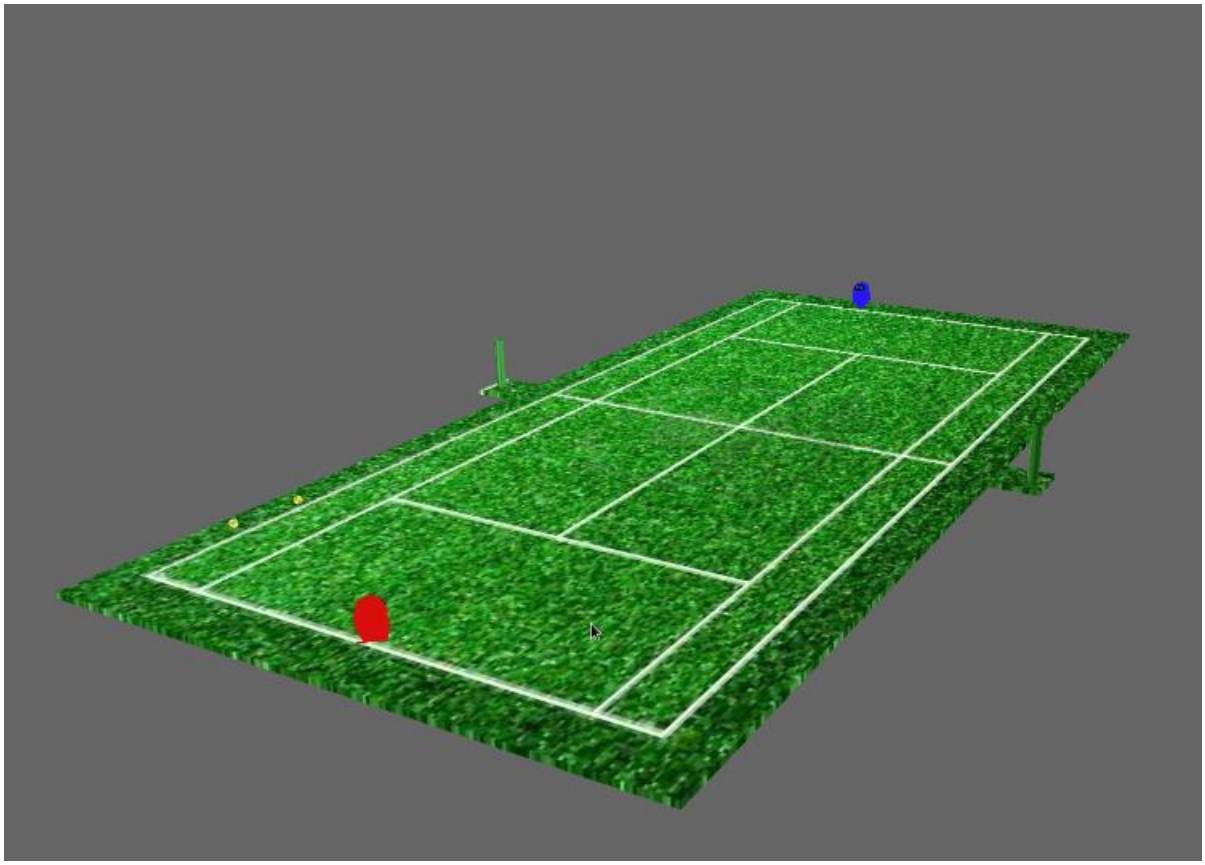

Figure 4.5.1 The Pong Game

4.5.2 The Maze Game

The Maze game has a detailed set of rules. At the start of the game, the user is positioned at a random place on the map. A set of army contains a particular number of army agents and a single Scout agent as their leader. The agents that belong to a particular army follow the messages of the Scout from the same army. So each set of army has a discreet communication system with its Scout and no interference is there. The army agents are Green in color and the Scout agents are White in color. Once the game starts, the Army agents and the Scout agents randomly wan- der on the grid. The user has to move along the grid collecting coins, avoiding the vicinity of the Scouts. The user can also collect bullets, which are relatively few when compared to the coins, for his defense. If the agent comes within the vicinity of the Scout agent, the Scout turns red, indicating that the Chase state is activated. It then sends a message to the other agents in its army to chase the user. The user has to evade the agents. He can shoot them if he collects the bullets. But the bullets are limited so he has to carefully use them. If the agents get near him, then he is stranded and the game is reset. Another way of provoking the agents is that, if the agents shoot the scout from a distance, the agents in his army are alerted and they start chasing the user. In this way the user has to evade all agents by either killing them or avoiding them and collect all the coins to win the game. The moments of the user are controlled by the keys

1) Up: Moves Forward

2) Down: Moves Backward

3) Left: Moves Left

4) Right: Moves Right

5) B: Shoot

Chapter 5

Applications and Results

5.1 Results

The output of the project displays the mulit-agent behaviour scenario of a game environment. The message passing and the state changing of the agents function successfully giving an impression of the communication between agents and the artificial intelligence of the agents. The use of the simple wall follower algorithm adds up to the behaviour of the agents, and the path finding algorithm inspired from the Pac-Man game is very efficient in terms of finding the shortest path without the computational complexities and its ease of implementation and calculation. Overall a simple game with a convincing AI behaviour was successfully implemented.

5.2 Applications

The applications of Artificial Intelligence are many. Especially in games and robotics they define the landmarks. Simulating a multi-agent behaviour always finds its way in many scenarios. The project can be expanded further and developed into a complete game with more interesting behaviours. In its simple form it can be modified into a game on mobile platform. Crowd simulations are multi-agent behaviors on a larger scale.

Chapter 6

Conclusion

6.1 Summary

• The objective of the project is to simulate an environment with multi- agent behaviour, where the behaviour is scripted

• The objective is achieved successfully where a simple simulation of the game environment with agents interacting with the environment and message passing among themselves and state changing has been implemented.

• The behaviour is written in python a system was setup to communicate with the script to export and import data from the script.

• The graph network is successfully used with simple and efficient path finding and maze solving algorithms.

• A pong game was implemented to test the basic functionality of the system.

• Though the basic behavior is implemented a much more convincing environment can be simulated.

• The behaviours can be made more complex and interesting. The complexity of the game and levels can be increased.

• The system can be made much more robust and flexible giving a better scope of extending it. Fuzzy logic can be implemented in the behaviours of the agent, which will give a more convincing result.

• As a lot of time was spent researching it resulted in simple environment behaviour with basic structure. With little bit more attention and effort this project has a potential of being developed into a system which can simulate a high end game or a crowd interaction.

6.2 Issues and Bugs

The basic game behaviour works well. The interaction with the environment and the response output is as per the logic. There were some problems that were encountered during the development of the project. The first problem encountered was creating the environment. The AI agents should be able to traverse through the environment finding the path and avoiding the obstacles. After few failed attempts to implement the same thing using a bounding box the graph environment was chosen. As stated earlier it's easier to implement and the obstacle avoidance becomes an integral part of the node traversal. The other problem encountered was the implementation of the shortest path finding algorithm. Initially with an impression, that the path finding on graphs is easy, the Breadth First Search algorithm was implemented. The disadvantage with that algorithm was that, as the graph network gets dense, the algorithm has to traverse through a lot of edges before finding the shortest path. If the requirement of path finding is not dynamic then this would have been useful as the shortest path can be calculated before the start of the game and stored and can be utilized whenever required. But the requirement of the game is such that the user would be moving and in each iteration a shortest path is to be calculated. After spending some time researching on different options a simple path finding logic which was implemented in the 'Pac-Man' was considered. The behaviours of the agents in the game are inspired from

the moments of the Ghost characters in the Pac-Man game. The ghosts in the Pac-Man game calculate the shortest path in each iteration (Pittman 2011). This involves a grid and choosing between the nodes with the shortest distance to the target node. This proved efficient as the computation over head is decreased to the calculation of the distances from just eight nodes (the maximum number of nodes a node can have as a neighbor) when compared to the traversal through all the edges in each iteration. The basic idea was taken from the game, but it was customized depending on the requirement of the game. All the constraints in the original algorithm were not implemented. But because of this customization there were few bugs that were encountered during its implementation. The original logic suggests that the ghost cannot turn back at any time, in the normal state. It has to always choose a node which doesn't point backwards. This logic is perfect in the scenario where the ghost moves in a single column with wall on both sides. But in this game the columns between to wall are more than one. Because of this constraint I could only restrict the movement of the AI agents backwards only up to some extent. Not all possibilities in the reverse direction were considered. Due to this, the AI agents sometimes toggle back and forth while trying to reach the target position with the calculation of the shortest path. This is a small logical error which should be handled in a better way. But as the user is always on move, normally this toggling doesn't occur frequently. The algorithm gives a very good impression of chasing the user in the shortest path, which achieves the objective of artificial intelligence.

6.3 Future Work

I would like to continue developing the game into a much more complex scenario with interesting behaviors. Research on different AI methods like fuzzy logic and other algorithms like genetic algorithms and genetic programming and continue my quest in this field.

References:

1. Ferber, J., 1999. *Multi-Agent Systems An Introduction to Distributed Artificial Intelligence* Essex: Longman. The basic principles of Multi agent systems. The introduction to an agent, its attributes and features.

2. Wooldridge, M., 2004. An Introduction to MultiAgent Systems. Sussex: John Wiley & Sons. Types of agents. An introduction to the environment, agents and objects.

3. Buckland, M., 2005. Programming Game AI by Example. Sudbury: Jones and Barlett. State driven agent behavior. Creation of autonomously moving agents.

4. Bourg, D.M. And Seeman, G., 2004. AI for Game Developers. Sebastopol: O'Reilly. Chasing and Evading in agents, Flocking behavior.

5. Halleux, J., 2003. MetaAgent, a Steering Behavior Template Library. Available from: http://www.codeproject.com/Articles/4125/MetaAgent-a-Steering-Behavior-Template-Library [Accessed 10 June 2013]. History of behaviors and simple rules in behavior.

6. Shiffman, D., 2012. The Nature of Code. California: Creative Commons. Available from: http://natureofcode.com/book/chapter-6-autonomous-agents/ [Accessed 09 June 2013] Different autonomous behaviors, following simple rules.

7. Pittman, J., 2011.*The Pac-Man Dossier*. Available http://home.comcast.net/~jpittman2/pacman/pacmandossier.html#Table_Of_Contents [Accessed on 29th July 2013].

8. Weisstein, E., 1999. *Point-Plane Distance*. Eric Weisstein. Available from http://mathworld.wolfram.com/Point-PlaneDistance.html [Accessed on 21th June 2013]

9. Anon., *Python.* Netherlands. Available from http://www.python.org/about/website/ [Accessed on 1st June 2013]