Non-Deterministic AI in Games

Sai Raghunandan G
Master of Science Computer Animation and Visual Effects

November, 2013
Contents:

Abstract ................................................................. 3

1 Introduction
  1.1 Introduction....................................................... 5
  1.2 Objective.......................................................... 6
  1.3 Motivation.......................................................... 6
  1.4 Structure.......................................................... 7

2 Background and related Work
  2.1 Artificial Intelligence............................................. 9
  2.2 AI in games......................................................... 9
  2.3 Related Works.....................................................11

3 Technical Background
  3.1 Fuzzy Logic.......................................................15
  3.2 Finite State Machine.............................................19
  3.3 A* Algorithm.....................................................20

4 Implementation
  4.1 Implementation Overview......................................23
  4.2 Agent-Brain Model.............................................23
  4.3 Environment.....................................................24
  4.4 Artificial Intelligence...........................................27

5 Results and Analysis..................................................38

6 Conclusion..........................................................39

7 References..........................................................40
Abstract

Artificial Intelligence in games has always been an area of research and challenges. AI spans over several methods starting from simple chasing and evading, fuzzy-logic to much more complex neural networks and genetic algorithms. By using simple AI techniques games can be made much more interesting and the game characters unpredictable. Inspired from the paper “Real-Time Game Design of Pacman Using Fuzzy Logic”(Shaout et al 2006), this project is an attempt to recreate the classic arcade game Pacman by including fuzzy-logic based AI system and path finding algorithms, and compare it to its classical version and analyze the findings. This includes implementing the basic architecture, game rules and the variables used for the fuzzy system along with the path finding algorithms.

The game environment is manipulated to make it more interesting and complicated. The original Pacman game is implemented and then it is improved by adding fuzzy-logic. The results and analysis show that the altered version is better when compared to the original version, as its logic had limitations when the map was manipulated. The fuzzy-logic version of the game proved to be more adaptive and intuitive. My contributions include manipulating the game environment, implementing the fuzzy-system in Python to provide an ease in implementing the behaviors for the computer controlled entities, finding a solution for optimizing the path finding algorithm and present a comprehensive analysis of the comparison between the two versions.
Acknowledgments

I would like to thank my parents for everything. I would also like to thank Jonathan Macey and Mathieu Sanchez for their guidance and help throughout the project.
Chapter 1

Introduction

1.1 Introduction

Artificial Intelligence as the name suggests is the field of science which deals with the study and development of intelligent software or agents. AI in a sense can be interpreted as a science which deals with replicating the human cognitive powers in decision making and applying it in terms of computer programming. The artificial intelligent systems range from completely self-learning systems with least human interference to a partial artificial intelligent system with substantial human inputs. In the world of games AI has a very significant role to play. In the context of game, any output that comes out of the AI decision making process can be attributed to be behavior that one of the non-player characters in the game follows. As the present day game industries use different 3D techniques to achieve realism, that alone doesn't fulfill the objective of making the game feel real. The behaviors of the non-player characters should be made real to complete the sense of realism. By using methods and algorithms that build the AI systems, the game characters can be made realistic.

There are basically two types of AI methods (Bourg and Seeman 2004). Deterministic AI and Non-deterministic AI. Deterministic AI behaviors in games are of non-adaptive logic which are pre decided with no uncertainty connected to it. One of the best examples of a game with non-player characters having deterministic AI is Namco’s Pacman released in 1980. The four computer controlled opponents called ghosts use simple behaviors which change with the time elapsed irrespective of the player’s skill. This type of AI is very fast to develop and implement. The downside of such a behavior is that the game logic may be interesting in the beginning but in course of time they become predictable in most of the cases which makes the game experience less

![Figure 1.1 (a) Pacman](Image)

![Figure 1.1 (b) Tekken](Image)
effective. The next type of AI is the Non-deterministic, which, as opposing to the deterministic behavior is uncertain and unpredictable in nature. This kind of AI basically adapts to the game environment and takes decision. One such game with the non-deterministic AI is Namco’s Tekken released in 1994. This is a fighting game where the player has to combat the computer entity. The computer controlled fighter is unpredictable in terms of the moves or the powers it uses to defeat the player. The more unpredictable the game characters the more interesting the game. The nature of the unpredictability depends on the underlying AI technique employed. The degree of the success of a particular AI technique employed depends on the illusion of intelligence it creates. Though the human cognitive approach in decision making is too time taking and complex to develop and integrate in games, the illusion of intelligence in games can be based out of behaviorism. This mainly concentrates on the point that the entities have to take the most appropriate decision depending on the present environment state, which serves the purpose of making it look smart though they may not be actually smart.

Games having an interactive environment where the characters in the game dynamically react to the actions of the player are gaining prominence in the recent years. Games like Halo, Assassins Creed where the players work along with the game characters as a team, games with strategy like Rise of Nations and Age of Empires where the computer decides the strategy depending on the player’s skill are the kind of games that fascinate the players. All this is because of the advent of intelligence in the game logic. As the sense of intelligence can bring a whole new perspective to a game, the differences between the deterministic and non-deterministic AI effects on a game can be understood better if a single game is developed using both the AI techniques. In this project the game that is used for the discussion is Namco's classical Pacman. One of the papers by Shaout(Shaout et al 2006) discusses the implementation which is taken as a base for the thesis.

1.2Objective

The objectives of this project are to recreate the original Pacman game environment with the non-adaptive deterministic behavior and then improve it by including fuzzy Logic based AI in it to make it non-deterministic and analyze the differences. The project implementation is an adaptation of the paper “Real Time Game Design of Pac-Man using Fuzzy Logic” by Shaout et al. (2006).

1.3Motivation

Creating intelligent agents in the game is a challenging task to deal with. A more realistic life like AI behavior is not the one which is perfect in its decision making, but it is the one which can make mistakes like humans once in a while and take wrong decisions. Developing such kind of an AI behavior requires a very keen understanding on the human thinking and the way the world around is perceived. One means of quantifying computer’s intelligence is the Turing test (Turing 1950). The point of the test is that if the computer entity can convince the player that it is not controlled by computer, but by a human then it is qualified to be a good intelligent agent. One very good example of a game that convinces the players with the same effect is DOTA. Its a strategy game where the players play to build skill sets in each level to win over the opponents. This is also a multi-player game and one important point in the game regarding the non-player entities is that the players cannot easily distinguish between the other player entities and the computer entities. This is one of
the interesting implementation of AI which gives the human like sense to the non-player characters. One more influential game in creating convincing human like AI is *Halo*. Halo is a game in which the player has to complete a final mission which is accomplished in steps of objectives in each stage. The AI characters in this game are situated in a simulated world unlike the other First Person Shooter games where the players are normally a stationary camera. The AI characters in this game have senses that perceive the world around them, then they extract the information from their perception and interpret the data. After processing the interpreted data the AI entities take decisions based on that. After decisions are taken they choose their actions that can accomplish the decisions taken in the best possible way depending on the world around and their present state. Though this approach sounds similar to the normal conventions of agents perceiving their environment and reacting, this is different in the sense that these AI entities work with the same capabilities that a player has. They are not provided with the entire information of the world. This forms the key point in a way that since the non-player entities are forced to sense and perceive the world around them they are limited in ways just like a player is limited with his senses. Since their senses are limited, their perceptions are limited and thus they are capable of making mistakes. This makes them more life like, and thus there is a very little difference between the player and the non-player character that has AI.

![Figure 1.1 (c) Dota](image1.png) ![Figure 1.1 (d) Halo](image2.png)

These games define a very well executed non-deterministic AI. That is the prime motivation behind this project. The games show how non-deterministic AI can bring interesting behaviors, which in turn help the game with a realistic feel. The project tries to prove the same point that non-deterministic AI is required to achieve a better sense of realism.

### 1.4 Structure

The project is structured in steps that were followed to understand the underlying AI involved. Then an implementation of the AI system with the required variables and game rules is carried out. The behaviors of the game characters are written in scripting language python. Finally they are integrated into the game play which is written in C++. An analysis is followed in the end.

**Chapter 2: Related Work** – Discusses the different AI method used in games, then focuses on games that use Fuzzy logic in particular and the related research to put forth the prominence of the AI used.
Chapter 3: Technical Background – This section deals with the explanation of the method implemented in the paper and the technical set up for the project. It explains the fuzzy based AI system and the state machine employed.

Chapter 4: Implementation– Discusses the implementation details of the project. The fuzzy system and the individual ghost behaviors are written in python first. The game engine is written in C++. The way the game engine exchanges information with the scripting language is explained.

Chapter 5: Results and Analysis– This section presents the result obtained from the implementation and the comparison between the deterministic AI and the non-deterministic AI that were implemented.

Chapter 6: Summary– Discusses the known bugs and issues faced, the limitations in the implementations and the future work that is intended before concluding the project.
Chapter 2

Background and Related Work

2.1 Artificial Intelligence

The present AI has its roots in the early philosophical quest to understand mind and body (Anderson 2003). Logic, which is an important concept that forms the basic of Artificial Intelligence was considered a science only after the introduction of Boolean algebra in mathematics. Before the word artificial intelligence came into use, there were some cases of development of some mechanical machines that were used for reasoning or playing chess.

Another notable contribution in the field of AI is from the man who is considered the Father of Computer Science Alan Turing. Though the name Artificial Intelligence was used for the first time by John McCarthy in 1956 two years after the demise of Alan Turing, the present day AI development is inspired from his radical and original ideas (BBC 2012). He was the first man to look at computers as machines that are capable of thinking. He proposed the famous turing test also known as the imitation game to measure the capability of thinking of a machine. His works inspired the later generations in developing new techniques and algorithms in the field of AI. Over the years the field of AI evolved into different branches. The major areas related to AI are Reasoning, Machine learning, Language processing, Computer Vision and Robotics. The traditional approaches that are normally employed are logic and search algorithms. In the recent times a more modern approach is being used which comprises of Neural networks, Bayesian networks, Genetic programming and Evolutionary algorithms. The time-line for the events of AI as mentioned by Anderson(2003) are shown in the table 1.

The relatively recent achievements in this field are in 2005 when a robot from Stanford drove for 131 miles autonomously along a desert trail. Following that two years later in DARPA urban challenge a autonomous vehicle steered through the urban traffic adhering to the traffic rules for a distance of 55 miles by a team from CMU and the latest milestone is the IBM’s question answering computer defeating the greatest Jeopardy champions in a Jeopardy quiz(The Economic Times 2013). There are humanoids that are already being built, and the research still continuous to build computers that can think on their own.

2.2 AI in Games

AI by itself is a vast field with its applications in many distinctive areas. One of them is games. In games AI can be interpreted in any way which is in most of the cases subjective. Like the path finding can be AI to a developer whereas deciding a strategy in a game depending on players skill may be AI to another developer. Irrespective of the interpretation there's always an underlying logic or algorithm which is used to achieve the desired result. In games the AI has been used and customized on a per case basis, since the introduction of intelligent characters in games.
The development of game AI can be discussed starting from one of the earliest games that was developed using AI, Chess. The AI in chess works with a search algorithm which goes through a series of states and a memory of already loaded data. The AI is used to evaluate the positions in the search tree using some heuristic functions. Its one of the basic approaches. Then the introduction of simple AI behaviors like in the game of Pacman was made which made the illusion of intelligence from moving computer entities.

As the computer hardware development accelerated, use of graphics in the game increased simultaneously and also use of different algorithms was made feasible in games in terms of computations. This can be seen in the first generation PlayStation games where the AI is seen to be quite complex but is easily integrated with the game. Some of the most common AI techniques employed are fuzzy based systems, finite state machines.

<table>
<thead>
<tr>
<th>year</th>
<th>Major developments/events in AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1931</td>
<td>Gödel shows that some mathematical theorems that are known to be true cannot be proven by mathematical and logical means</td>
</tr>
<tr>
<td>1937</td>
<td>Church-Turing Thesis states that all problems that a human can solve can be broken down into an algorithm</td>
</tr>
<tr>
<td>1943</td>
<td>McCulloch &amp; Pitts develop a model of artificial neurons</td>
</tr>
<tr>
<td>1948</td>
<td>Wiener publishes the book “Cybernetics” on information theory</td>
</tr>
<tr>
<td>1949</td>
<td>Hebb presents learning process for neural networks</td>
</tr>
</tbody>
</table>
| 1950 | • Shannon develops early chess-playing program  
• Turing states the idea for the turing test  
• Asinov states the three laws of robotics |
| 1951 | Minsky & Edmonds build SNARC – first neural network computer |
| 1956 | Dartmouth Conference – term “Artificial Intelligence” used for 1st time by McCarthy |
| 1958 | • McCarthy develops LISP – first dedicated AI programming language  
• Simon makes a number of predictions of the future of AI:  
  - computer will prove mathematical theorem (happened 1996)  
  - computer will be chess champion (happened 1997) |
| 1963 | ANALOGY by Evans solves problems from human IQ tests |
| 1965 | Weizenbaum programs ELIZA (early chatterbot) |
| 1967 | DENDRAL rule-based system for analysing molecular structures created |
| 1972 | PROLOG AI language developed |
| 1974 | MYCIN developed by Shortliffe – first expert system (medical diagnostics) |
| 1975 | learning program Meta-Dendral makes first scientific discoveries (chemistry) by a machine |
| 1983 | Laird and Rosenbloom work on SOAR – an AI architecture which is now also applied to computer games (SOAR Quakebot) |
| 1990 | Koza develops genetic programming (GP) – programs that evolve |
| 1996 | • Kasparov beats Deep Blue (world’s most powerful chess computer)  
• computer proves Robbins problem (1st creative proof of a mathematical theorem by a machine) |
| 1997 | • Deep Blue finally wins against Kasparov  
• first official Robo-Cup soccer match |

Table - 1 : The time line of Artificial Intelligence mentioned by E. Anderson (Anderson, 2003)
Then the games with intelligent environments were made where the agents in the game are provided with very limited information and the information is contained in the objects in environment. The agents have to gain access to those objects and then process it further. Examples for such games are SIMS and GTA vice city.

As the graphics developed the need for more realistic and conceivable games increased and that gave rise to the second, third fourth generation games where more complex search algorithms and techniques are being used. The path finding algorithms using navigation meshes can be quoted as the recent advances which employ the traditional path finding techniques in better way.

### 2.3 Related Work

The computer controlled Pacman ghosts in this project represent a class of intelligent agents. They can be classified as autonomous entities which are defined as systems inside an environment which perceive the environment around and act on it, and each has its own goals to pursue (Buckland 2005). The concept of autonomous agents is famous in games and computer graphics. Some of the work that was carried out in the same field is ‘Individual-Based Models’ by Craig Reynolds (Reynolds, 1999). This model is a subset of Multi-agent systems which is generally a system made of different agents with different behaviors that can interact among themselves and with the world around and they together make the system work. Each agent interaction in the system has a global effect on the system.

Another significant work by Craig Reynolds was a paper on distributed behavioral model (Reynolds, 1987) in which each agent forms a unit of a flock and follows some basic rules and together they form the flocking behavior. Other applications where we see intelligent agents in games are games such as FIFA, EA Sports Cricket, where each player is a part of a team and the environment and has his own role to play. All the players together try defending the player team.
Coming to the fuzzy system, there were several implementations using the fuzzy system in games. Some of the games that employed the fuzzy based systems are

**Unreal:** This game uses fuzzy State machines for the control of non-player characters. The behaviors of the computer entities such as to run away, or hide if wounded, call for back up or ambush the player when needed are controlled by strong state machines that run with fuzzy logic (Woodcock 1999).

**Call to Power:** Activision's *Call to Power* is noted to for its high use of serially connected Fuzzy state machines that guide the strategically thinking of the computer entities. The different civilizations portrayed in the game have different governing and militaristic bent. By using a fuzzy state machine each civilization can have a degree associated to it which alters its decision making capabilities depending on their respective unique traits. This was easily achievable using fuzzy and writing special code for each civilization was avoided (Woodcock 1999).

**S.W.A.T 2:** This is an action game that uses fuzzy logic which decides the enemy responses not only based on the situation but also on the personality traits of the particular enemy. Depending on the personality's present state of aggression, intelligence, courage the appropriate action is decided (Johnson and Wiles 2001).
Along with these games there are few other papers that deal with the real time implementation of fuzzy logic in games as mentioned by Shaout (Shaout et al 2006). One of them discusses the implementation of fuzzy logic in another similar game Battle City (Li et al 2004). This paper follows the same framework that is employed in the paper that is being followed in this project. A framework called as Belief-Desire-Intention model or BDI model. This model relies on the theory that the 'belief' is the knowledge of the environment, 'desire' is the goal of the agent and finally 'intention' is the action of the agent.

Another attempt in the same field as mentioned by Shaout (Shaout et al 2006) is design of intelligent game characters using behavior based control of mobile robots (Sanormoi and Sooraksa 2004). They propose a fuzzy based system to design artificial intelligent using a different architecture called the Takagi-Sugeno model.

While still speaking about fuzzy-logic one more important software that is based out of fuzzy-logic and is very prominent in the field of computer graphics is MASSIVE which is an abbreviated form of Multi Agent Simulation System In Virtual Environment. It is a Multi-agent simulation software which was developed in Weta Digital for the movie Lord of the Rings. It depends on the agent-brain based model where the brain works with a fuzzy network that controls the action of the agents. It was used in many award winning visual effects movies.
The methods used in this project include a Multi-agent system and fuzzy-logic. Both these methods have been implemented before and were successful with wide range of applications. It shows the potential with which this project can be improved further. The fuzzy-logic techniques that were used were formed out of basic fuzzy set rules and that simplicity in defining a world in terms of fuzzy set rules has been explored in this project. The agents system was made independent and as flexible as possible to be able to expand it in future and explore further possibilities.
Chapter 3

Technical Background

The implementation of this project requires the basic understanding of the underlying algorithms and the systems that it uses. This chapter describes the concepts and algorithms that were used in the project.

3.1 Fuzzy-logic

Fuzzy-logic forms the base of this project. This is the method that is mentioned in the paper. It is used to assist the computer controlled ghosts in the game to take decisions and choose a particular behavior depending on different game rules and variables. Fuzzy-logic was first introduced by Professor Lotfi Zadeh in his paper explaining out the fuzzy set theory in 1965. He explains “fuzzy logic is the means of presenting problems to computers in a way akin to the way humans solve them. The essence of fuzzy logic is that everything is a matter of degree” (Bourg and Seeman 2004, p-188).

It can be defined as a form of logic which has more than two truth values to deal with. Conventional Boolean logic has only two truth values either true or false, whereas fuzzy logic has variables whose truth values range from 0 to 1. They deal with the concept of partial truth which may be in between absolute truth (one) or absolute false (zero).

For example considering temperature as a parameter and defining two conditions cold and hot. Cold being anything between zero Fahrenheit to 50 Fahrenheit and hot being anything between 50 Fahrenheit to 100 Fahrenheit. Given a random temperature \( x \), the difference between conventional Boolean logic and fuzzy-logic can be defined as

1) If \( x \) is 45

   Boolean: \( 0 =< x < 50 \)
   Result: \( x \) is cold

   Fuzzy: \( x \) is 55% of cold and 45% of hot
   Result: \( x \) is 0.45 degree hot and 0.55 degree cold

2) If \( x \) is 70

   Boolean: \( 50 =< x <=100 \)
   Result: \( x \) is hot

   Fuzzy: \( x \) is 30% of cold and 70% of hot
   Result: \( x \) is 0.7 degree hot and 0.3 degree cold

It can be seen, fuzzy logic has degree of truth associated to its variables. If the value of \( x \) is 0 Fahrenheit then it would be defined as 1 degree cold and 0 degree hot. This is the basic definition on which the fuzzy logic operates. The advantage of fuzzy-logic is that all the variables that are used can be defined in linguistic terms which are non-numeric values that give a better ease in defining the facts and rules, like in the previous example the terms such as cold, moderate and hot can be used when
defining our rules and assessing them for the output. Then a set of fuzzy rules are applied on this linguistic variables to decide the result.

The fuzzy rules can be set of rules that help in decision making. Considering the previous example, let us say the decision to be taken is whether the fan should be switched on or off depending on the temperature. So the fuzzy rules can be

1) If cold switch on the fan with low speed
2) If hot switch on the fan with high speed

So here the degree of the truth value of cold and hot can be related to the speed of the fan which helps in the smooth transition from low speed to high speed when the temperature is increasing instead of an abrupt transition in case of Boolean logic. If there are more than two linguistic variables involved the logical operators AND or OR or NOT are also included in the rules. For example if there is another linguistic variable called wind, then the fuzzy rules may be defined as

1) If cold AND windy switch off the fan
2) If cold AND NOT windy switch on the fan with low speed
3) If hot AND windy switch on the fan with medium speed
4) If hot AND NOT windy switch on the fan with high speed

In this way the fuzzy rules help us deciding the output.

The three basic rules in fuzzy logic are

1) Covert the crisp input variables to Fuzzy Input
2) Apply the Fuzzy rules and calculate the output
3) Convert the fuzzy output to Crisp output

**Figure 3.1** Fuzzy rule-based interface (Buckland 2005, p-17)
Membership functions:

The crisp inputs are mapped to fuzzy inputs using functions called membership functions. This is called fuzzification. The membership functions calculate the degree for the crisp inputs in terms of linguistic variables. These are normally linear or ramp functions. Defining the membership functions depends on the requirement. The following are different membership functions that were used in this project.

1) Grade function:
If \( x \) is the input and \( a \) is the lower limit and \( b \) is upper limit

\[
\mu_a(x) = \begin{cases} 
0, & x < a \\
\frac{x-a}{b-a}, & a \leq x \leq b \\
1, & x > b 
\end{cases}
\]

![Figure 3.2 Fuzzy Grade Function](Alonso 2013)

2) Reverse Grade Function:
Input is \( x \) and lower limit is \( c \) and upper limit is \( d \)

\[
\mu_a(x) = \begin{cases} 
0, & x > d \\
\frac{d-x}{d-c}, & c \leq x \leq d \\
1, & x < c 
\end{cases}
\]

![Figure 3.3 Fuzzy Reverse Grade Function](Alonso 2013)

3) Triangle Function:
Input \( x \), lower limit is \( a \), upper limit is \( b \) and value \( m \) given as \( a < m < b \)
4) Trapezoid Function:
Input $x$, lower limit $a$, lower support limit $b$, upper support limit $c$ and upper limit $d$ given as $a < b < c < d$

$$\mu_a(x) = \begin{cases} 
0, & x \leq a \\
\frac{x - a}{m - a}, & a < x \leq m \\
\frac{b - x}{b - m}, & m < x < b \\
0, & x \geq b
\end{cases}$$

Figure 3.4 Fuzzy Triangle Function (Alonso 2013)

After finding the degree of truth for the variables using the membership functions, the fuzzy set rules are applied on these variables to decide their weightage in each possible output decision. As mentioned above we are using logical rules for the game variables in this project and we use logical operators in the rules. So we define the functions for the logical operators as well. AND is defined as minimum of two given variables, OR is defined as maximum of two variables and NOT is defined as the value obtained after subtracting the variable from one.

Finally after deciding the output for each possible output action in terms of fuzzy variables, they are converted back to crisp output. This is called defuzzification. This can be done in different ways, such as the average maximum method or centroid method. The method followed in this project is described in implementation sections.

The fuzzy-logic application is elaborated in the coming sections.
3.2 Finite State Machine

Finite State Machine or FSM is a machine that is abstract in nature. It can exist in one of the many predefined states. The change in the states determined by a set of predefined conditions can also be defined using FSM's. The behavior of the machine at a given instance depends on the present state that its in. It is one of the important tool to create the sense of intelligence. The advantages of FSM's are mentioned as (Buckland 2005)

1) Relatively simple and quick to code
2) Debugging is easy
3) Overhead in terms of computations is relatively low
4) Intuitive and Flexible

One of the famous early finite machines is Turing Machine. The underlying concept of FSM's is that the behaviors of a computer object or agent can be broken down in small chunks of small states or behaviors. A good example as mentioned by Buckland (2005) for a finite state machine is a light switch. It has two states associated to it On and Off. By tripping the switch down the state is transitioned from On to Off and by tripping it up the state is changed from Off to On. Though there is no particular behavior associated to its states, this basic concept can be used in games to define what different states can do in a FSM.

![Figure 3.6 Finite state machine – light Switch(Buckland 2005,p-44)](image)

The ghost’s behavior in the Pacman game has always been a classical example of explaining FSM. We use that in this project to define the behaviors that they follow depending on the logic.

The FSM's can be coupled with fuzzy-logic. The fuzzy-logic can decide the state the FSM changes to based on a set of rules. This provides a better decision making scheme. The same has been used in this project.
3.3 A* Path Finding Algorithm

Path finding is one of the frequent issues that most of the game developers find hard to deal with. A poor algorithm for path finding can make the agents look artificial. So care should be taken in deciding the right algorithm. One of the most efficient algorithms for path finding is A*. It is considered because it guarantees to find the best part between the start and end point given a path exists between them. The algorithm works on nodes, polygons or points that define the map. The map used in this project is node based. It uses a best-first search to find the path with a least cost from the source node to the destination node. It starts with the source node it traverses through the adjacent nodes following a least total distance or cost, keeping a track of the alternative path segments that come in the way in a priority queue in sorted order. It uses heuristic functions for each node, which determines the order in which the nodes in the tree or graph are visited for search, to direct the search towards the destination node. The efficiency in the algorithm depends on the heuristics used. The cost function is basically a sum of two costs
1) The past path cost which is the calculated distance to the present node from the source node
2) The heuristic cost function which estimates the distance from the present node to goal node.

The final cost obtained from this calculation is used to sort the nodes in the queue which is used to determine the order of nodes to visit. The heuristic cost function is basically a Euclidean distance function or a Manhattan distance function. The implementation and efficiency in terms of time will be discussed in the coming sections. Below is the Pseudo code for this algorithm that is described in the

If (Target == Source) then
  Return INVALID;
  Add Source to Open List (GCost = 0, HCost = cityblock dist from Source to Target,
  Parent = Null)
While (Open List is not empty)
  Current = item in Open List with lowest (GCost + HCost)
  If (Current == Target) then
    Break;
  End If;
  For (each of 4 squares adjacent to Current)
    If (square is not a wall) then
      If (square not already on Open List) then
        Add square to Open List (GCost = Current GCost + 1,
        HCost = cityblock dist from square to Target,
        Parent = Current)
      Else
        If (Current GCost + 1 <
        GCost of square on Open List) then
          GCost of square on Open List = Current GCost + 1;
        End If;
      End If;
    End If;
  End For;
  Move Current from Open List to Closed List;
End While;
If (Open List is empty) then
  Return INVALID;
End If;
Retrace Path from Current to Source;
Return direction from Source to the best path to Current;
The fuzzy-logic and the FSM form the core elements in the implementation. The fuzzy-logic selects the state that should be chosen and the FSM transitions to the new state. The path finding algorithm is used by the agents in different cases like chasing the player or navigating to the home position. It enhances the behavior of the agents. The implementation of this algorithm was done with reference to source Buckland’s (Buckland 2005) book.
Chapter 4

Implementation

Pacman is one of the most successful games. The premise of the game is relatively simple. The player controls and guides the Pacman in a 2D grid containing pellets. The aim of the game is to clear all the pellets in the map in each level trying to avoid the ghosts that follow the Pacman. The ghosts have different behaviors associated with them, and the decision to follow a certain behavior depends on different rules. The topic of discussion in this project is the AI which assists the ghosts to decide a particular behavior. A systematic approach was described in the paper “Real Time Game Design of Pac-Man Using Fuzzy Logic” (Shaout et al 2006) to create fuzzy-logic operated ghosts. The implementation that was carried in this project used certain sections of the paper as reference and the remaining part was carried out in a different approach. The game environment that was discussed in the paper is the original Pacman grid environment that has single column grids. But in this project the map is modified to a much broader space without single columns so that the movements of the intelligent agents can be tracked in a better way. The paper discusses the implementation of the fuzzy-logic Pacman alone, but this project also includes an implementation of the original Pacman which is carried out with the help of different sources (Pittman 2011). Since the focus of the project is to implement a non-deterministic behavior for the agents, in most of the cases care was taken to follow the original game rules but small deviation from the game rules can be noticed. The only difference between the original Pacman AI and the approach explained in this project is the way the agents choose between their behaviors or states. The remaining design for both the games is the same. With this in mind, the following paragraphs explain the common implementation details of the project starting from the approach followed in conceiving the idea and then the Artificial Intelligence part which marks the difference between the deterministic and non-deterministic behaviors.

Figure 4.1 Original Pacman Grid (Pittman 2011)

Figure 4.2 Grid that is used in this Project
4.1 Implementation Overview

There are some basic things that form the minimum requirements for any game. Some of them are the underlying Map for the game, the non-living entities in the game like the pellets, the living entities which in this case are Pacman and the agents and an Environment which takes care of everything right from the interaction between the agents to handling different functions in the world around the agents. This project follows the same model. It has an underlying graph network as a map, the different entities in the game like the User agent, the AI agents and an Environment which loads the map, calculates the shortest path, loads the agents passing on the controls, just to name but a few. Coming to the technical details, the entire game engine is written in C++ except for the behaviors of the AI agents which were written in scripting language.

4.2 Agent: Body-Brain model

The agent is the important entity in the game. Though there isn't much to discuss about the agents that are controlled by the user, Pacman in this case, this section outlines the model followed for the AI agents. The individual agents in the game make up for the game experience on the whole. The agents in this game are classified as autonomous agents as mentioned in the earlier sections. Each agent has some physical properties that can be called as its body and then it has a decision making part which can be called as its brain. Body has no intelligence associated with it, whereas the brain has no physical attributes.

Agent-Body:

As mentioned above this part takes care of the physical properties of the agent. This includes information like the present position where the agent is situated, the velocity with which it is moving. As the scope of this project is limited to the game, there aren't many physical properties that this game requires. But this model is a more general approach which can be used for Multi-agent systems. This part of the code is written in C++ as a part of the game engine.

Agent-Brain:

This is the important component of the game which is responsible of the decision making process. The brain receives inputs from the body, as well as the environment around, processes the information and decides the necessary behavior. Depending on the AI that it uses, the inputs vary. For a deterministic AI the inputs are few as the decision process is relatively simple and for the non-deterministic the inputs it requires is more which will be dealt in the AI part. Apart from the inputs it requires the common output that the brain gives back to the body is the next relative position that the body should be moving to and the velocity with which it moves. The behaviors of the brain uses Finite state machine. At a given time the brain can be in one state or one behavior. And the way it chooses that state depends on the AI again. The different behaviors that the brain can choose are given below

1) Scatter: This behavior is random in nature. The ghosts just wander around the map. When the agents are loaded, they are assigned with a home position. When in Scatter mode the agents keep moving towards their home position.
2) Chase: In this mode, the agents chase the user i.e, Pacman in an attempt to capture and kill it. This requires the agents to access the current position of the Pacman in each game cycle. Agents can have a same chase logic or different chase logic depending on the AI used.

3) Defense: In this mode, the agents defend the area of the map that has the highest number of pellets.

4) Shy: In this mode, the agents move away from each other.

5) Frightened: In this mode the agents run away from the Pacman, avoiding being killed by it.

The finite state machine uses these states. Further details regarding which AI uses which states and the logic used in each state is discussed in the Artificial Intelligence section. The behaviors of the agents are generally volatile in nature when compared to the physical properties. To enable an ease in providing flexibility to edit the behaviors, they are written using a scripting language. A scripting language enables to write and refine the mechanics of the systems (Bourg and Seeman 2004). The game engine in C++ reads the behavior 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. 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. These features make Python a very suitable language for scripting.

4.3 Environment

This is the important part of the game which handles all the functions in the game. The environment keeps track of all the agents, entities and the map. It updates the agents and world in each update cycle. It reads the text file information, passes it to the map to load. The map takes care of
setting the obstacles in the map and then uses a network of graph nodes and edges to generate the
required level for the game. An overview of the important classes is given below

1) Map:

The Pacman game is based on a 2D grid. To recreate the same environment a graph network is
used. The agents traverse on the nodes of the graph. 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 and the
maze dimensions. Zeroes represent positions without obstacles on the map whereas ones represent
positions where there are obstacles. This data is stored in the form of a vector of integers of zeros and
ones and the maze dimensions are stored as integers. The map uses the dimensions to generate nodes.
When the nodes in the graph network are created, they are created with the same dimensions of n x n
nodes. 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
there’s an obstacle on the node. After generating the nodes, the map class passes the obstacles
vector of zeros and ones to the graph class. The graph class iterates through the obstacles data, 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 positions of the
nodes with their onOff status ‘1’ are saved in a vector named obstaclepositions and is sent back to the
map class. The map class uses this information to draw the obstacles. 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.

2) Graph:

The graph network is one of the important part of the environment. It comprises the grid on
which the agents move. It is the basis for the shortest path algorithms that were implemented. The
graph data is maintained by the graph class, which has the the Grid nodes and Graph edges as its
member variables.

3) Grid Nodes:

The nodes of the graph are the primitives which form the graph network. This data is contained
in the GridNodes class. The nodes are generated in two dimensions, width and height which are read
from the parser class. Each node has an index and position attributed to it. Each node is
connected to its neighboring node by an Edge. A given node can have a minimum of two nodes to
maximum of four nodes as its neighbors. An agent traverses from node to one of its neighboring node
along the edge connecting them.

4) Graph Edges:

The edges in the graph are the connections between the nodes. They 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)

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

Any agent on a given node has to choose among these neighboring nodes for its traversal depending on the state. The GraphEdge class contains information 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 as required. The graph network is an efficient way of representing a map given the static nature of the game. The obstacle avoidance and the collision detection can be included in the network 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. More than one agent cannot be on a given node at a time, this makes the collision detection between them easy.

5) User Agent (Pacman)

The user agent or the Pacman which is controlled by the user. The UserAgent class takes care of the functions and attributes of the Pacman. The environment creates an instance of the user agent in the beginning of the game. The controls of the Pacman 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 Pacman then moves along the grid depending on the input. The user can move in any direction.

6) The Game Entities and Models

The Game entities are the part of the environment, which Pacman has to collect as a part of the game. These game entities are three types

   i) Pellets – which are to be cleared by Pacman
   ii) Power Pellets – which Pacman can take to kill the ghosts
   iii) The moving Pellets (represented as teddy bears in the game)- The are bonus pellets which boost the score

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 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.
4.4 Artificial Intelligence

This is the essence of the project. The intelligence in the agents and the rules for the decision making are described here.

1) Non-deterministic AI

As mentioned earlier the non-deterministic AI is based on fuzzy logic. This section of the project uses the methods outlined in the paper. A framework similar to the one named Beliefs-Desire-Intentions is used here which works on deriving rational behaviors depending on logical theories. The 'Beliefs' is also known as knowledge that the intelligent agents require. In this context the beliefs are the location of the player, the location of the other game entities and the information which indicates the skill of the player (good, bad or medium). Coming to 'desire' it is categorized as the goals that the intelligent agents have and in this case their goal is to intercept the Pacman. Intentions are the course of action they have to carry out and that includes maintaining distance from other ghosts, chasing Pacman and defending the pellets.

The states of the ghosts that are associated with this AI are scatter, defense, hunting, shy and fright. When the Pacman takes a Power Pellet, the ghosts enter the fright state. In this state they avoid Pacman. Other than that in normal conditions the fuzzy-logic based rules select one of the behavior for each ghost in each cycle. The defense, Shy are the new behaviors that are introduced in this approach.

In defense state, the ghosts try to defend the area of the map with the highest number of pellets. The method that is used to calculate that is, at first the entire map is divided into nine overlapping sections based on the following fractions of the depth and height of the map. The three sections in the x axis and z axis are given x is the width and z the height

1) From 0 to x/2
2) From x/4 to 3x/4
3) From x/2 to x

Similarly
1) From 0 to z/2
2) From z/4 to 3z/4
3) From z/2 to z

After dividing the map, compute the total number of pellets in each section. Out of them, select the section with the highest number of pellets. Then the position of the pellet that is present in the middle of that section is read and is passed on to the ghosts. The ghosts traverse to that position to guard that section of map. The A* star algorithm is used to travel to the middle pellet position. In the design the SpatialPartition class takes care of dividing the map into overlapping section and returning the position of the middle pellet.

In Shy state, the ghosts move away from the closest neighboring ghost. The calculations involved are first the Manhattan distance from a given ghost to all other ghosts is calculated, then the position of the ghost that is closest is considered. Next the available adjacent nodes of the given ghosts present graph node are taken, and the node which is farther from the position of the closest ghost is considered as the
next node and the ghost moves in that direction.

In fright state the same Shy logic is used but the only difference is that, the instead of the closest ghost position the users position is considered. The A* star algorithm is also used in scatter state to find out the shortest path to the home position and also in hunting state to calculate the shortest position to the users position. The Scatter state is selected when none of the other states are selected.

Linguistic Terms:

The fuzzy-rules use a lot of linguistic variables. They are non-numeric values which we used to define the inputs. The linguistic variables that were used in this project are classified into three types distance, rate and time.

Distance:

The two types of distances that are associated with the game are the distance between the ghosts and the player and the distance between the each ghost pair. Since its a 2D grid, the Manhattan distance method suits well in calculating the distance and also it is not computationally expensive. The distance can be divided into three areas near, medium and far. Now the distance is relative to the size or dimensions of the map used. So they are compared against the variable called level_size which is the sum of width and height. These distances in terms of crisp inputs are mapped to the fuzzy inputs using membership functions. The same membership function is used for the two types of distances. The function is given below.

![Figure 4.3 Membership Function for Distance (Shaout et al 2006,p-321)](image)

The linguistic terms that the distance defines are

- pacman_near
- pacman_med
- pacman_far
- ghost_near
- ghost_med
- ghost_far
These are derived from the membership function above.

**Time:**

The next linguistic term is Time. The unit defined for the time is 'tick' which is the one cycle of the Pacman or the ghost moment. There are two types of time variables. The first one defines the amount of time elapsed since the Pacman has eaten a pellet defined as pellet time. The other one is the time elapsed since he lost a life defined as average life time. There are again three regions in this linguistic term. Short time, medium time and long time. The time variables are compared against another variable called the base_time which is the shortest time taken by Pacman to reach from one corner of the map to the other end. This depends on the ticks taken by him to traverse one node multiplied by the level_size. If Pacman traverses one node per tick then the level_size is equal to base_time. The membership functions for the pellet time and average life time are given below.

![Figure 4.4 Membership Function for Pellet Time](image)

The linguistic terms from this function are:
- pellet_short
- pellet_med
- pellet_long

![Figure 4.5 Membership Function for Average Life Time](image)

The linguistic terms from this function are:
- time_life_short
- time_life_med
Rate:

There is only one rate variable defined that is pellet rate, which is nothing but the number of pellets eaten per the number of ticks elapsed. The pellet rate can be a decreasing function, but in some bonus conditions when additional pellets are generated, the pellet rate can increase. The pellet rate has three areas defined bad, medium and good. The membership function for this variable is a little bit different than others. The medium rate is a triangle function with its peak at 0.5. The good pellet rate is the direct ratio of the pellets eaten to ticks consumed, and to avoid the rate exceeding the range [0,1] it is made sure that it is clipped to this range whenever it exceeds it. The bad rate is simply a complement of the good rate. The rate membership function is given below.

![Figure 4.6 Membership Function for Pellet rate(Shaout et al 2006,p-322)](image)

The linguistic terms from this function are:
- pellet_rate_good
- pellet_rate_med
- pellet_rate_bad

Behaviors and Fuzzy-Rules:

Now as we have the linguistic terms defined, the fuzzy rule set should be defined. The fuzzy-logic model follows the Mamdani fuzzy model which is explained below. As mentioned above for each ghost fuzzy-logic system selects a behavior for them. Each behavior is related to a set of fuzzy rules that define the likelihood of selecting that behavior. Several linguistic terms that are a derivative from the game progress parameters and the performance of the player are used here. The following list of images define the set of fuzzy rules that are associated with each behavior.

Hunting Behavior
• If (pacman_near AND skill_good) then hunting_behavior
• If (pacman_near AND skill_med AND pellet_med) then hunting_behavior
• If (pacman_near AND skill_med AND pellet_long) then hunting_behavior
• If (pacman_med AND skill_good AND pellet_long) then hunting_behavior
• If (pacman_med AND skill_med AND pellet_long) then hunting_behavior
• If (pacman_far AND skill_good AND pellet_long) then hunting_behavior

Figure 4.7 Rules for Hunting (Shaout et al 2006, p. 320)

Defense Behavior

• If (pacman_far AND skill_bad AND ghost_far AND pellet_short) then defense_behavior
• If (pacman_far AND skill_bad AND ghost_far AND pellet_med) then defense_behavior
• If (pacman_far AND skill_bad AND ghost_med AND pellet_short) then defense_behavior
• If (pacman_far AND skill_bad AND ghost_med AND pellet_med) then defense_behavior
• If (pacman_far AND skill_med AND ghost_far AND pellet_short) then defense_behavior
• If (pacman_med AND skill_bad AND ghost_far AND pellet_short) then defense_behavior

Figure 4.8 Rules for Defense (Shaout et al 2006, p. 320)

Shy Behavior
• If (pacman_far AND skill_bad AND ghost_near AND pellet_short) then shy_ghost_behavior
• If (pacman_far AND skill_bad AND ghost_near AND pellet_med) then shy_ghost_behavior
• If (pacman_far AND skill_bad AND ghost_med AND pellet_short) then shy_ghost_behavior
• If (pacman_far AND skill_bad AND ghost_med AND pellet_med) then shy_ghost_behavior
• If (pacman_far AND skill_med AND ghost_near AND pellet_short) then shy_ghost_behavior
• If (pacman_far AND skill_med AND ghost_near AND pellet_med) then shy_ghost_behavior
• If (pacman_med AND skill_bad AND ghost_near AND pellet_short) then shy_ghost_behavior

**Figure 4.8 Rules for Shy (Shaout et al 2006,p-320)**

Random Behavior

\[
\text{If NOT (hunting_behavior) AND NOT (shy_ghost_behavior) AND NOT (defense_behavior) then random_behavior}
\]

**Figure 4.9 Rules for Random (Shaout et al 2006,p-320)**

As we see there is a term included in all the three behaviors called Skill. Skill is the parameter that is used to assess the player's performance at three levels good skill, medium skill and bad skill. There is a separate set of fuzzy rules to derive the player’s skill. They are given as

• If (time_life_short OR pellet_rate_bad) then skill_bad
• If (time_life_medium OR pellet_rate_medium) then skill_medium
• If (time_life_long AND pellet_rate_good) then skill_good

**Figure 4.10 Rules for Skill (Shaout et al 2006,p-320)**

The skill should be calculated first before calculating a behavior. The logical operators are used to calculate the output. The AND uses the minimum operator and the OR uses the maximum operator. For the fright state the only condition required is that Pacman should take a power pellet. If that condition is true, then the FSM chooses that behavior for the ghosts, and the shy logic is used for the ghosts to stay away from Pacman. The fright state has a timer associated to it, and when the timer elapses the behavior selected by fuzzy-logic is followed by the ghosts.

Defuzzification:

Now for each behavior we have a set of rules, and in each cycle the individual fuzzy rule value for each rule is calculated. These individual fuzzy rules pertaining to a single behavior are combined using the
maximum operator (AND operator). The resultant value is a defuzzified output for each value in the range of [0,1]. This method of defuzzifying it is computationally less expensive than other methods. After getting the resultant values for all the behaviors, they are multiplied with a particular weight value to give precedence or priority to certain behaviors. And after multiplying them with a weight, the behavior with the highest output value is chosen. A selection example is mentioned below.

![Defuzzification Table](Shaout et al 2006, p.322)

In this way the Mamdani model of fuzzy-logic is used to select the behaviors of the ghosts in each cycle. This gives the non-deterministic behavior for the ghosts as the decision are dynamically calculated depending on the game progress and other parameters. If the Brain of the agents is following this AI then the inputs it would be requiring from the game engine are:

- The number of ticks for pellet
- The distance between its neighbors
- The distance between Pacman and the ghost
- The average life ticks
- The base time
- The level size
- The middle pellet in the section with most pellets
- The home position
- The navigation data
- The Pacman position

Since the behaviors of the ghosts were written in scripting language, the fuzzy-system is also written in scripting language to make it easy in a way that the behaviors can just read the fuzzy functions as functions from different files. This makes sense as in future if another membership function is required, it can simply be written in the fuzzy file and can be read by the behaviors.

2) Deterministic AI

This is the second kind of AI which doesn't have a particular system as such to decide the behaviors, but chooses between its different behaviors depending on the time elapsed. The behaviors that are associated with this behavior are scatter, hunting and fright. In hunting state, the ghosts chase the Pacman. But to give an illusion of intelligence in the agents four different logic were used for each ghost. The logics are:

1) Ghost 1 – Always chases the Pacman with the Pacman's present node as target.
2) Ghost 2 – The target node to this ghost is the node which is two nodes ahead of Pacman in the direction it is heading.

3) Ghost 3 - This ghost uses the complex algorithm out of the four. It requires the position of the node that is two nodes ahead of Pacman, in the direction Pacman is heading and then the position of the Ghost 1. A vector to from the Ghost 1's position is drawn to the node that is two nodes ahead of Pacman. Now this vector is doubled in its vector length extending out in direction opposite to the direction Ghost 1 is facing. This new position is target to the Ghost 3.
4) Ghost 4 – This ghost toggles between chasing the Pacman and heading back to its home position by checking its Euclidean distance from Pacman. If the distance between them crosses an offset it starts chasing Pacman by making his present tile as target tile. But if the distance is within the offset, it goes back to scatter state heading to its home position.

In the original Pacman a very different logic is used for chasing the Pacman. The AI agent’s present node has neighboring nodes connected to it. Out of all the neighbors only the neighbors that are free to traverse are selected i.e., the nodes without obstacles on them. From the selected set of the neighboring nodes, the linear distance between the target position and the position of the each adjacent 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. This work perfectly in a 2D grid with single columns. But since changes were made to the map, to compensate for that the A* algorithm was used for the shortest path in hunting state. And even in the scatter mode A* is used for the same reason.

For the fright state, in the original game the ghosts follow random path, choosing one of the adjacent nodes to its present node in a random fashion. But again because of the map, the fright state uses the same logic that its non-deterministic AI uses.

The decision of the behaviors chosen follows a certain pattern where a certain amount of time is given to each behavior and once that time is elapsed, they change the state. And this pattern depends on the level as well. As an example table 2(Pittman 2011) is shown below. When the ghosts are in fright state the states respective timers are halted until the fright state timer comes to zero. Once it comes to zero, the ghost come out of this state and resume from their old state timer value and continue from there.

![Figure 4.16 State Change Table(Pittman 2011)]

The logic in most of the cases depends on either the time elapsed or the pellets eaten by Pacman. Initially when the game starts the moment when the ghosts leave their home also depends on a timer or pellets count so that the ghosts leave their home sequentially. One more important factor that assists the logic is the speed of the ghosts. It varies when they are in fright state and any other state.

There aren't many details to explain the deterministic AI as it doesn't follow any certain set of rules to decide the state changes. If the Brain of the agents is following this AI then the inputs it would be requiring from the game engine are

- The Pacman position
- The position of node two nodes ahead of Pacman in the direction it is heading
- The Time elapsed since the game started
- The timer values that are associated with each state
- The position of Ghost 1 (This information is used only by Ghost 3)

Performance Issue with A*:

Since most of the states use the A* algorithm for shortest paths. If they are following Pacman at the same time in each iteration the ghosts ask for the shortest path, it increases the computation time as
a result of which the jerks in the game were seen. Though the algorithm is efficient in finding the shortest path, the use of it in each update cycle by agents simultaneously had its effect on the game play. To overcome this problem the shortest paths for the entire graph from each node to all the nodes in the network is precalculated using another tool written in C++ for all the levels that are used in this game. All those shortest paths are written in to a file and at the beginning of the program that file read and the shortest path data is stored in a table. And whenever it is required by the agents in the program the table is looked up to find the shortest path and the data is passed through them. This method is very efficient as it saves the computation time, it is generated only once and as the map is static there is no need to change the data unless you plan to change the structure of map. When calculating the shortest paths, as A* requires a heuristic function, in this case the heuristic is made zero as this is not a directed search and it is a search from a single node to all the other nodes. Calculating the shortest paths using that tool relatively takes a long time, but is worth it as it makes the game play smooth. The path of the input file is passed as a command line argument and this tool outputs the data into a file that should be mentioned in the program.

The Game Play:

When the program is started the path of the files which contains the level map files is read. The file is read and the map is loaded. Then the agents are allocates with their initial positions by assigning an index number to them which corresponds to a node in the network. The AI is to be chosen between deterministic and non-deterministic. The level of difficulty is selected. The initial score is set to zero and the number of pellets taken is set to zero and all the parameters are set to zero. During the course of play if the ghosts capture Pacman their positions are reset to their home positions and the game resumes and one life of Pacman is reduced. If all the pellets in a given level are taken, the level is completed and the game initializes the next level. Else if the player dies three times the game is over. There are teddys that pop in the maze once in a while which boost the score. There are three levels designed for this project.
Chapter 5
Results and Analysis

5.1 Game Result

The non-deterministic logic works very well. The state change in the ghosts is dynamic and it changes as the parameters of the game play changes. In the defuzzification by changing the weights, certain behaviors were repeated more frequently. Since the map has been made broad maneuvering Pacman became more exciting as there is lot of ground to cover. Different weights are associated to the different levels of difficulty and the behavior patterns and the intelligence of the ghosts was evaluated. A small change in the weight values can bring significant change to the way the behaviors are chosen. The ghosts seemed to be recognizing the presence of the Pacman as in some instances, moving towards them triggered the hunting behavior in them and as the Pacman changed the direction the state of the ghost changed. Such interesting behaviors were observed which were predictable at times and unpredictable in most of the cases.

The deterministic AI seemed interesting in the beginning, but it is noticed that as the map for the game is changed, it changed the illusion of the intelligence. In a 2D grid with single column sandwiching Pacman is easy compared to a spacious map. In some instances the four different logics that the ghosts were following were evident, which were not very evident in the original Pacman. The ghosts were heading to a position other than the position of Pacman. And at times they seemed to be crowded by moving together as a group as they are no longer in a single column and they don't have a shy behavior.

5.2 Comparison

The difference between the deterministic and non-deterministic AI very significant and quite evident. The deterministic behavior as the name suggested was predictable after two three time game play. Though the four different behaviors were added to the ghosts to add to the realism of the game, it was proved that they only work in a restricted environment. If the map structure changes, the realism is no more intact and it requires more amount of time and logic to recreate the realism for the new structure again. On the other hand fuzzy logic can be used irrespective of the map, be it a restricted map or a well-defined map. In the deterministic AI the behavior of the ghosts will be same in any given condition or position of Pacman no matter how many times that condition is repeated. But when it comes to non-deterministic AI, the behavior is an effect of the players skill and pellet consumption. So if a given condition is repeated twice they may not react in the same way. To make the player believe the difficulty level increases in as the game progresses, the only way that can be done in deterministic AI is either by increasing the time they stay in hunting state or by either increasing their speed which makes it difficult to avoid. But in non-deterministic approach the weights can be manipulated easily and different levels can be created without actually interfering with speed of the ghosts. The fuzzy-rules can be also be expanded to add new rules and interesting functionality to the game. It is relatively simple to include new fuzzy sets or change the old ones.
Chapter 6

Conclusion

An attempt to highlight the effect of deterministic and non-deterministic AI on the game play experience was carried out. The non-deterministic AI was implemented using fuzzy-logic systems following the methods listed in the paper “Real Time Game Design of Pacman Using Fuzzy-logic”(Shaout et al 2006). The implementation was successful with interesting results. The ghosts had a better illusion of intelligence in the non-deterministic AI. Its relatively flexible way of changing the rules or adding rules to include new behaviors makes it more interesting. It can be improved further by making a dynamic map and adding more behaviors for the FSM and fuzzy-logic systems. It can also be used along with other traditional AI techniques to experiment with.
References


8) Bourg, D.M. and Seeman, G. 2004. AI for Game Developers. 1St ed. Sebastopol: O'Reilly.


