Department of Electronic Engineering

FINAL YEAR PROJECT REPORT

BEngCE-2008/09-SYY-01

Development of a Program for Ms. Pac-Man Competition
(Independent Project II)

Student Name: Chan Kwong Choi
Student ID: 
Supervisor: Dr YUEN, Kelvin S Y
Assessor: Dr SO, H C
Bachelor of Engineering (Honours) in
Computer Engineering

Student Final Year Project Declaration

I have read the student handbook and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I declare that the work submitted for the final year project does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Student Handbook.

Project Title : Development of a Program for Ms. Pac-Man Competition
(Independent Project II)

Student Name : Chan Kwong Choi

Student ID :

Signature

Date :
# Contents

## List of Figure
1. Abstract p.1
2. Introduction p.2
   2.1 IEEE Ms. Pac-Man Competition p.2
   2.2 Result of 2007 IEEE Ms. Pac-Man Competition p.3
   2.3 Result of 2008 IEEE Ms. Pac-Man Competition p.4
3. Motivation p.5
4. Program Description p.6
   4.1 Screen Capture p.7
   4.2 Road Map Development p.8
   4.3 Ms. Pac-Man Location Detection p.13
   4.4 Ghosts Location Detection p.19
   4.5 Update Game States p.27
   4.6 Development of Algorithm p.30
5. Result p.35
6. Discussion p.36
   6.1 Limitation p.36
   6.2 Ideas for Improvement p.37
7. Conclusion p.39
8. Reference p.40
List of Figures

Figure 2.2_01 Result of 2007 IEEE Ms. Pac-Man Competition  p.3
Figure 2.3_01 Result of 2008 IEEE Ms. Pac-Man Competition  p.4

Figure 4.01 Flow of Software Controller  p.6

Figure 4.1_01 Ms. Pac-Man Game Image and Four Sub-Images  p.7

Figure 4.2_01 2D Array Road Map and Sub-Image 3  p.8
Figure 4.2_02 Relationship between 2D Array Road Map and Pixels  p.8
Figure 4.2_03 3 Different Game Level Images  p.9
Figure 4.2_04 The way to find color of power pills  p.10
Figure 4.2_05 6 Combinations of Previous and Current Ms. Pac-Man Locations  p.10
Figure 4.2_06 2D Array Road Map and Coordinate Values  p.12

Figure 4.3_01 Ms. Pac-Man with 4 Moving Directions  p.13
Figure 4.3_02 Example for Headdress and Nevus Pattern Hidden by Ghosts  p.13
Figure 4.3_03 14×14 Pixel Areas with Most Yellow Pixels  p.14
Figure 4.3_04 14×14 Pixel Area with Left Top Most  p.14
Figure 4.3_05 Searching around Ms. Pac-Man’s Previous Location  p.15
Figure 4.3_06 Ms. Pac-Man Travels through Tunnels  p.16

Figure 4.4_01 Ghosts’ 3 States  p.19
Figure 4.4_02 Ghost Overlap with Each Other  p.19
Figure 4.4_03 Ghost Overlap Sequence  p.20
Figure 4.4_04 Overlap with the one closest to itself  p.20
Figure 4.4_05 Ghost Overlap or Dead  p.21
Figure 4.4_06 Unstable Ghost Location Detection in Weak and In-Between States  p.22
Figure 4.4_07 Ghost House  p.23
Figure 4.4_08 Searching around Ghost’s Previous Location  p.23
Figure 4.4_09 Ghosts Travel through Tunnels  p.24

Figure 4.5_01 4 Sub-Images  p.27
Figure 4.5_02 Game Over Image  p.28
Figure 4.5_03 Game Image with word “READY!”  p.28

Figure 4.6_01 Example of “num”  p.31
Figure 4.6_02 An Example of Next Ghost Moving Direction  p.33
Figure 4.6_03 Full Moving Direction Estimation Table  p.34

Figure 6.2_01 8 Cases Consideration  p.37
1 Abstract

The aim of this project is to develop a program for the IEEE Ms Pac-Man Competition. The objective of the program is to earn as high a score as possible. A Software Controller consisting of a Detection Part and an Algorithm is developed. A Detection Part constructs a 2D Array Road Map such that the corresponding locations of Ms. Pac-Man and Ghosts are found from captured game images. An Algorithm is invented to make use of the 2D array Road Map and the location of Ms. Pac-Man and Ghosts to decide the direction Ms. Pac-Man should go at each moment. The highest score of this Software Controller is 30870 while the average score and the standard deviation are about 13513 and 6087 respectively. For comparison, the highest score and the highest average score from IEEE WCCI 2008 are 15970 and 11981 respectively. This report describes in details the development of the Detection Part and the Algorithm. Problems found during development are discussed. Finally, ideas on how to improve the Software Controller are reported.
2 Introduction

2.1 IEEE Ms. Pac-Man Competition

Ms. Pac-Man Competition is held by IEEE (Institute of Electrical & Electronic Engineering) since 2007. Participants need to develop a software controller to play Ms. Pac-Man game automatically. Winner will be the one whose software controller can achieve the highest score in 3 plays [1]. However, from IEEE WCII 2008 Ms. Pac-Man Competition, there are two ways to determine the winner [2]. They are so called “on-site” and “off-site”. “On-site” means participants attend the conference and run their software controller by their own computer for three times. “Off-site” means participants only send their software controller to IEEE and their software controller will be run by other computers for 10 times. No matter they are “on-site” or “off-site”, the software controller which achieves the highest score would be the winner.
2.2 Result of 2007 IEEE Ms. Pac-Man Competition

Figure 2.2_01 shows the result of 2007 IEEE Ms. Pac-Man Competition [3].

<table>
<thead>
<tr>
<th>Default</th>
<th>Wirth</th>
<th>Handa</th>
<th>Elno</th>
</tr>
</thead>
<tbody>
<tr>
<td>1330</td>
<td>1120</td>
<td>1000</td>
<td>650</td>
</tr>
<tr>
<td>2030</td>
<td>820</td>
<td>1300</td>
<td>1040</td>
</tr>
<tr>
<td>1940</td>
<td>760</td>
<td>2170</td>
<td>1310</td>
</tr>
<tr>
<td>2390</td>
<td>1250</td>
<td>1760</td>
<td>1790</td>
</tr>
<tr>
<td>2990</td>
<td>3370</td>
<td>1880</td>
<td>1360</td>
</tr>
<tr>
<td>2060</td>
<td>1700</td>
<td>1310</td>
<td>680</td>
</tr>
<tr>
<td>3810</td>
<td>1490</td>
<td>2270</td>
<td>620</td>
</tr>
<tr>
<td>3140</td>
<td>1990</td>
<td>2210</td>
<td>1830</td>
</tr>
<tr>
<td>1010</td>
<td>1380</td>
<td>1700</td>
<td>1370</td>
</tr>
<tr>
<td>1990</td>
<td>2830</td>
<td>1910</td>
<td>1160</td>
</tr>
</tbody>
</table>

| Ave     | 2269  | 1673  | 1751 | 1181 |
| Max     | 3810  | 3370  | 2270 | 1830 |

Figure 2.2_01: Result of 2007 IEEE Ms. Pac-Man Competition

In figure 2.2_01, “Default” is a software controller provided by IEEE as a reference for participants. The highest score of “Default” is better than other participants.
2.3 Result of 2008 IEEE Ms. Pac-Man Competition

Figure 2.3_01 shows the result of 2008 IEEE Ms. Pac-Man Competition [2].

<table>
<thead>
<tr>
<th>Entrant</th>
<th>Platform</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>High</th>
<th>Ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Southern Maine</td>
<td>Web/Mac</td>
<td>12780</td>
<td>15400</td>
<td>7500</td>
<td>9870</td>
<td>5480</td>
<td>15970</td>
<td>19750</td>
<td>11166.67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hanka</td>
<td>MS/Win</td>
<td>6610</td>
<td>10550</td>
<td>11320</td>
<td>12910</td>
<td>6990</td>
<td>7800</td>
<td>12930</td>
<td>9875.607</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dortmund</td>
<td>Web/Mac</td>
<td>4440</td>
<td>2380</td>
<td>2600</td>
<td>7190</td>
<td>3690</td>
<td>4260</td>
<td>7190</td>
<td>4130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>remote</th>
<th>Platform</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>High</th>
<th>Ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>San</td>
<td>Web/Win</td>
<td>15400</td>
<td>15010</td>
<td>14770</td>
<td>4150</td>
<td>12820</td>
<td>14450</td>
<td>8240</td>
<td>14190</td>
<td>8610</td>
<td>11920</td>
<td>15400</td>
<td>11981</td>
</tr>
<tr>
<td>Flensburg</td>
<td>MS/Win/Cpp</td>
<td>8640</td>
<td>4950</td>
<td>6520</td>
<td>3960</td>
<td>8720</td>
<td>8490</td>
<td>11990</td>
<td>5480</td>
<td>8050</td>
<td>7270</td>
<td>11990</td>
<td>731</td>
</tr>
<tr>
<td>Leandro</td>
<td>MS/Win</td>
<td>3860</td>
<td>5820</td>
<td>3510</td>
<td>4770</td>
<td>4580</td>
<td>5880</td>
<td>5550</td>
<td>7150</td>
<td>5260</td>
<td>8040</td>
<td>8040</td>
<td>5248</td>
</tr>
<tr>
<td>Ruck (Escape)</td>
<td>MS/Win</td>
<td>3190</td>
<td>5780</td>
<td>2660</td>
<td>9920</td>
<td>4190</td>
<td>3210</td>
<td>4880</td>
<td>1080</td>
<td>6120</td>
<td>8770</td>
<td>9930</td>
<td>5058</td>
</tr>
<tr>
<td>Pecool</td>
<td>MS/Win</td>
<td>1980</td>
<td>750</td>
<td>550</td>
<td>750</td>
<td>3020</td>
<td>1700</td>
<td>830</td>
<td>12590</td>
<td>12080</td>
<td>2000</td>
<td>12590</td>
<td>3625</td>
</tr>
<tr>
<td>Wirth</td>
<td>Web/Win</td>
<td>1900</td>
<td>1900</td>
<td>2500</td>
<td>3330</td>
<td>2320</td>
<td>2480</td>
<td>2290</td>
<td>4800</td>
<td>2800</td>
<td>1830</td>
<td>4800</td>
<td>2510</td>
</tr>
<tr>
<td>Shintakawa</td>
<td>Web/Win</td>
<td>1070</td>
<td>890</td>
<td>2500</td>
<td>4040</td>
<td>5520</td>
<td>3260</td>
<td>1350</td>
<td>2000</td>
<td>2500</td>
<td>1820</td>
<td>5520</td>
<td>2493</td>
</tr>
<tr>
<td>Jabbari</td>
<td>MS/Win</td>
<td>870</td>
<td>1590</td>
<td>2010</td>
<td>1480</td>
<td>2850</td>
<td>2990</td>
<td>4170</td>
<td>2000</td>
<td>3300</td>
<td>3020</td>
<td>4270</td>
<td>2994</td>
</tr>
<tr>
<td>Ruck (Ambush)</td>
<td>MS/Win</td>
<td>5490</td>
<td>3410</td>
<td>6390</td>
<td>2980</td>
<td>2780</td>
<td>3890</td>
<td>6380</td>
<td>5980</td>
<td>3860</td>
<td>5820</td>
<td>6390</td>
<td>4694</td>
</tr>
<tr>
<td>Sapin</td>
<td>Web/Win</td>
<td>Did not work</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.3_01: Result of 2008 IEEE Ms. Pac-Man Competition

The result in 2008 is much better than that in 2007. The highest score in 2008 is 15970 while the second and third are 15400 and 12930 respectively.
3 Motivation

The highest score for human player playing Ms. Pac-Man game is over 900,000 while the highest score for AI/CI Program in IEEE WCCI 2008 is only about 16,000. Hence, there is still a large area for development of AI/CI Programs. Besides, I see this project as an opportunity to experience development of AI Program. I believe that AI and CI would be widely used technologies in the future.
4 Program Description

This part aims to describe the development of the software controller. The software controller captures images from the Ms. Pac-Man game and passes captured images to do analysis. A 2D Array Road Map with corresponding locations of Ms. Pac-Man and Ghosts is constructed. Game Information is also updated. For instance, the number of marks at that moment, the number of lives Ms. Pac-Man remained, etc. An Algorithm is invented to make use of these information to decide the direction Ms. Pac-Man should go.

Figure 4_01 shows the flow of Software Controller.

![Flow of Software Controller](image)

Figure 4_01: Flow of Software Controller

More than 15 times per second
4.1 Screen Capture

Figure 4.1_01 shows a Ms. Pac-Man Game Image and 4 Sub-Images

Figure 4.1_01: Ms. Pac-Man Game Image and 4 Sub-Images

Four Sub-Images are captured from the Game Image. These four sub-images are then passed to do analysis.
4.2 Road Map Development

Road Map Development makes use of Sub-Image 3. Figure 4.2_01 shows a 2D Array Road Map and Sub-Image 3.

The width of the road is greater than 1 pixel. Thus, the center point of the road is chosen to represent the Road Map. The size of the 2D Array Road Map is 113 × 125 while the size of the Sub-Image 3 is 226×250 pixels. One unit size of 2D Array Road Map is equal to 4 pixels. Figure 4.2_02 describes this relation.
As shown in figure 4.2_02, pixel numbers 1, 2, 3 and 4 represent one unit of 2D Array while pixel numbers 5, 6, 7 and 8 represent another unit of 2D Array. The location of left top pixel is used to represent the location of 4 pixels. For instance, location of pixel number 1 and 5 are (12, 12) and (14, 12) respectively. The corresponding 2D Array Road Map for pixel number 1 is “array[12/2][12/2]” which is equal to “array[6][6]”. The corresponding 2D Array Road Map for pixel number 5 is “array[14/2][12/2]” which is equal to “array[7][6]”.

Figure 4.2_03 shows 3 Different Game Level Images

In Figure 4.2_03, many small dots and 4 big dots are shown at each Game Level. Small dots are called as pills while big dots as power pills. The color of pills and power pills changes in different Game Levels. Since color of pills and power pills are the same, if the color of power pills is found, the color of pills is also found.
Figure 4.2_04 shows the way to find color of power pills.

As shown in figure 4.2_04, pixel number 1, 2, 4, 9, 11 and 12 are in black color while pixel number 3, 5, 6, 7, 8 and 10 are not in black color. If this pattern is found, it is the power pills. The color in pixel number 3, 5, 6, 7, 8 and 10 is the color of power pills. The location of this power pills is pixel number 13 (12, 20) which is equal to “array[6][10]”.

Now, a 2D Array Road Map with pills and power pills is developed. In the Ms. Pac-Man game, pills and power pills disappeared when Ms. Pac-Man passes through. To indicate if the road has been passed through by Ms. Pac-Man, previous and current locations of Ms. Pac-Man are used. Since the location of Ms. Pac-Man updates more than 15 times per second, there are only 6 combinations between previous and current locations. Figure 4.2_05 shows these 6 combinations.
At this moment, a 2D Array Road Map with pills and power pills is developed. Whether the road has been passed through by Ms. Pac-Man can also be checked. Data and methods in the 2D Array Road Map are listed in Table 4.2_01.

Table 4.2_01: Methods and Data in the 2D Array Road Map

**Board.java**

Methods:
- public static void board_develop(BufferedImage image);
- public static void init_pre_value();
- public static void update_road();

Data:
- public static boolean[][] road = new boolean[113][125];
- public static boolean[][] pills = new boolean[113][125];
- public static boolean[][] power_pills = new boolean[113][125];
- public static int[] p_pills_x = new int[4];
- public static int[] p_pills_y = new int[4];
- public static boolean[][] turn_up = new boolean[113][125];
- public static boolean[][] turn_right = new boolean[113][125];
- public static boolean[][] turn_down = new boolean[113][125];
- public static boolean[][] turn_left = new boolean[113][125];
- public static boolean[][] pass = new boolean[113][125];

“Board.board_develop(sub-image3);” develops a 2D Array Road Map.

“Board.init_pre_value();” initials the previous location of Ms. Pac-Man in the Board.java

“Board.update_road();” updates boolean[][] pass.
Figure 4.2_06 shows a 2D Array Road Map and Coordinate Values.

Value of “x” is larger on the right hand side.

Value of “y” is larger at the bottom.

Consider location \((x, y)\) where \(0 \leq x \leq 112\) and \(0 \leq y \leq 124\).

“\(\text{Board.road}[x][y]\)” is true if location \((x, y)\) is a road.

“\(\text{Board.pills}[x][y]\)” is true if location \((x, y)\) consists a pills.

“\(\text{Board.power_pills}[x][y]\)” is true if location \((x, y)\) consists a power pills.

“\(\text{Board.p_pills_x}[i] \text{ and } p_pills_y[i]\)” is the location of power pills \((x, y)\) where \(0 \leq i \leq 3\).

“\(\text{Board.turn_up}[x][y]\)” is true if location \((x, y)\) is a road and \((x, y-1)\) is also a road.

“\(\text{Board.turn_right}[x][y]\)” is true if location \((x, y)\) is a road and \((x+1, y)\) is also a road.

“\(\text{Board.turn_down}[x][y]\)” is true if location \((x, y)\) is a road and \((x, y+1)\) is also a road.

“\(\text{Board.turn_left}[x][y]\)” is true if location \((x, y)\) is a road and \((x-1, y)\) is also a road.

“\(\text{Board.pass}[x][y]\)” is true if location \((x, y)\) has been passed through by Ms. Pac-Man.
4.3 Ms. Pac-Man Location Detection

Ms. Pac-Man Location Detection makes use of Sub-Image 3.

Figure 4.3_01 shows Ms. Pac-Man with 4 Moving Directions.

![Figure 4.3_01: Ms. Pac-Man with 4 Moving Directions](image)

In figure 4.3_01, a circle in green color is a head dress while a circle in purple color is a nevus. When Ms. Pac-Man does not partially overlap with ghosts, location of Ms. Pac-Man can be found through searching the headdress and nevus pattern. If the headdress and nevus pattern is found, the moving direction of Ms. Pac-Man is also found due to the different combination pattern of headdress and nevus.

Sometimes, Ms. Pac-Man overlaps with ghosts partially and partial or all headdress and nevus pattern is hidden by ghosts. Figure 4.3_02 shows an example for headdress and nevus pattern hidden by ghosts.

![Figure 4.3_02: Example for Headdress and Nevus Pattern Hidden by Ghosts](image)

Counting the number of yellow pixel within $14 \times 14$ pixel area is used to determine if this is Ms. Pac-Man. If a $4 \times 14$ pixel area consists of more than 25 yellow pixels and has the most number of yellow pixels compared with other $14 \times 14$ pixel areas, it is assumed that is the location of Ms. Pac-Man. Figure 4.3_03 explains the above idea.
In figure 4.3.03, the $14 \times 14$ Pixel Area in the left image has less yellow pixels than that in the right image. Thus, the correct location of Ms. Pac-Man is the center point (green color) of $14 \times 14$ Pixel Area in the right image.

Sometimes, the number of yellow pixels are the same in several $14 \times 14$ Pixel Areas. The location would be the one ($14 \times 14$ Pixel Area) with left top most. Figure 4.3.04 explains this idea.

In figure 4.3.04, although the location of Ms. Pac-Man should be the one in left image, the program has errors to choose the one in the right image. Program first chooses the one with the most left. If some of them are equal left, it chooses the upper one.

The reason why the number of yellow pixel should greater than 25 is by observing the change of body shape of Ms. Pac-Man and is just a rough estimation. However, this number performs quite well.
Now, idea on how to find location of Ms. Pac-Man is explained. Following talks about ways to reduce amount of time spent on finding location of Ms. Pac-Man.

Since location of Ms. Pac-Man only appear on the 2D Array Road Map, program only searches along the 2D Array Road Map which “Board.road[x][y]” is true. Besides, the program searches area around the previous location. If not find, search the whole 2D Array Road Map. Figure 4.3_05 shows an example of Searching around Ms. Pac-Man’s Previous Location.

Figure 4.3_05: Searching around Ms. Pac-Man’s Previous Location
Sometimes, Ms. Pac-Man travels through tunnels. At that moment, program cannot find the location of Ms. Pac-Man. It keeps the last updated Ms. Pac-Man location and sets Ms. Pac-Man is inside tunnels until Ms. Pac-Man comes out from tunnels. Figure 4.3_06 describes the idea.

Figure 4.3_06: Ms. Pac-Man Travels through Tunnels

Figure 4.3_06 shows four sequential images. In the first image, Ms. Pac-Man is coming into a tunnel. In the second image, most of Ms. Pac-Man’s body is inside the tunnel. Program cannot find Ms. Pac-Man at this moment and thus sets “in_tunnel” to be true. The location of Ms. Pac-Man at this moment is the same as that in first image. In the third image, Ms. Pac-Man is still travelling through tunnels and does not come out yet. In the last image, Ms. Pac-Man passed through tunnels. New location of Ms. Pac-Man is updated and “in_tunnel” is set to false.
Table 4.3_01 shows procedures to find Ms. Pac-Man Location and Moving Direction.

### Table 4.3_01: Ms. Pac-Man Location Detection Procedures

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>01.</td>
<td>Search around Ms. Pac-Man’s previous location and find headdress and nevus pattern. If found, update location and moving direction of Ms. Pac-Man and set Ms. Pac-Man is not inside tunnels.</td>
</tr>
<tr>
<td>02.</td>
<td>Search around Ms. Pac-Man’s previous location and count the number of yellow pixels within 14×14 pixel area. If the number of yellow pixel is more than 25 and consists the most number of yellow pixels, update the location of Ms. Pac-Man and set not inside tunnels. The moving direction of Ms. Pac-Man in this case can be determined by considering Ms. Pac-Man’s previous and current location.</td>
</tr>
<tr>
<td>03.</td>
<td>Search the whole image and see if Ms. Pac-Man’s headdress and nevus pattern can be found. If found, update the location and moving direction of Ms. Pac-Man and set it is not inside tunnels. Otherwise, set Ms. Pac-Man is inside tunnels.</td>
</tr>
</tbody>
</table>

From table 4.3_01, if Ms. Pac-Man cannot be found by 01, 02 would be used. If Ms. Pac-Man cannot be found by 02, 03 would be used. There are total 3 steps to find location of Ms. Pac-Man.
Table 4.3_02 shows methods and data of Ms. Pac-Man.

Table 4.3_02: Methods and Data of Ms. Pac-Man

Agent.java

**Methods:**
- public static void init_agent();
- public static void update_agent(BufferedImage image);

**Data:**
- public static int board_x;
- public static int board_y;
- public static int dir;
- public static boolean in_tunnel;

“Agent.init_agent();” initials the location of Ms. Pac-Man at each Game Level.

“Agent.update_agent(sub-image3);” updates the location of Ms. Pac-Man.

“Agent.board_x” and “Agent.board_y” is (x, y) coordinate in 2D Array Road Map.

“Agent.dir” is the moving direction of Ms. Pac-Man.

“Agent.in_tunnel” is true if Ms. Pac-Man is inside tunnels.
4.4 Ghosts Location Detection

Ghosts Location Detection makes use of Sub-Image 3.

Ghosts have 3 states. They are Strong, Weak and In-Between. Figure 4.4_01 shows these 3 states.

The names of Ghosts are Blinky (in Red color), Pinky (in Pink color), Inky (in Blue color) and Sue (in Orange color). When Ghosts are in Strong state, the location of these four ghosts can be found by searching their color. The corresponding moving direction for each ghost can also be found by looking at their eyes. For instance, in figure 4.4_01, Blinky is moving left, Pinky is moving down, Inky is moving up and Sue is moving right. However, Ghosts overlap with each other frequently. Sometimes just part of its body, sometimes the whole body is hidden by other ghosts. Figure 4.4_02 shows how ghosts overlap with each other.

In figure 4.4_02, the body of Inky is partially hidden by Pinky in left image while the whole body of Inky is hidden completely by Pinky in right image.
By observation, Blinky is always at the top, then Pinky, Inky and Sue. Figure 4.4_03 shows this observation.

Base on this observation, program first searches Blinky’s location since Blinky would not be hidden by other strong state ghosts. Then, it searches Pinky’s location, Inky’s location and last Sue’s location. When Ghosts overlap with others, program will set Ghost overlap with the one which is closest to it. Figure 4.4_04 shows an example for this idea.

In figure 4.4_03, Sue overlaps with both Pinky and Blinky. Since Blinky is closer to Sue than Pinky, Sue is set to overlap with Blinky but not Pinky.

Before a Ghost hidden by another Ghost completely, it must be partially overlap with others. Therefore, when a Ghost overlaps with others completely and cannot be found by program at that moment, program will find the previous overlap record and see if a Ghost overlapped with others in the previous moment. If so, program will set the location of Ghost
same as the one overlaps with it. If not, program will set that Ghost to be dead. Figure 4.4_05 describes this idea.

![Figure 4.4_05: Ghost Overlap or Dead](image)

In the upper image of figure 4.4_05, Sue overlaps with Inky. In the lower image, Sue disappeared and cannot be found by program. Program finds that Sue overlaps with Inky at the previous moment and thus sets Sue’s location same as Inky.

When Ghosts are partially hidden by others and both eyes cannot be found by program, program will count the number of color of a ghost within $14 \times 14$ pixel area. If the number of color is more than 8, they are assumed to be a Ghost. For instance, in figure 4.4_05, both eyes of Sue are hidden by Inky. Program counts the number of orange (Sue) color within $14 \times 14$ pixel area. Since the number of orange color in this case is more than 8, program updates the location of Sue and its moving direction by considering Sue’s previous and current location. Also, program sets Sue overlaps with Inky.
When Ghosts are in Weak or In-Between states, the program cannot differentiate which one is Blinky, Pinky, Inky and Sue. For instance, the location of Weak state Blinky may be set as Weak state Pinky and so on. Also, the moving direction of Weak state Ghosts cannot be found due to unstable Ghost Location Detection. Figure 4.4_06 explains this idea.

![Figure 4.4_06: Unstable Ghost Location Detection in Weak and In-Between States](image)

In figure 4.4_06, Ghost 0 and 1 overlap with each other while Ghost 2 and 3 are alone in the left image. However, after a few steps, in the right image, the program detects that Ghost 2 and 3 overlap with each other and Ghost 0 and 1 are alone. Due to this unstable location detection, the moving direction of Weak state Ghosts cannot be determined by considering a ghost’s previous and current location.
Following talks about ways to reduce amount of time spent on finding location of Ghosts. Figure 4.4_07 shows a ghost house.

Since Ghosts appear in the Ghost House or on roads, program only searches area of Ghost House and along roads. To reduce the amount of time, program would search around area of ghosts’ previous location first. If not found, program would search the whole image. Figure 4.4_08 shows this idea.

In figure 4.4_08, a big blue square is the area program searches around based on ghosts’ previous location.
Sometimes, Ghosts travel through tunnels. At that moment, program cannot find the location of Ghosts. It keeps the last updated Ghost Location and set Ghosts are inside tunnels until Ghosts comes out from tunnels. Figure 4.4.09 describes this idea.

Figure 4.4.09 consists of 4 sequence images showing Pinky traveling through tunnels. In the first image, program can detect the location of Pinky. In the second image, program cannot detect the location of Pinky and thus keeps Pinky’s previous location and sets Pinky inside tunnels. In the third image, Pinky is still inside the tunnel and program just keeps Pinky’s location unchanged. In the last image, Pinky comes out from tunnels and program can detect its new location. Thus, program updates Pinky’s location and sets Pinky not inside tunnels.
Table 4.4.01 shows procedures to find location and moving direction of Ghosts.

Table 4.4.01: Procedures for Ghosts Location Detection

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01.</td>
<td><strong>Search Ghosts one by one.</strong> The sequence is Blinky, Pinky, Inky and Sue.</td>
<td></td>
</tr>
<tr>
<td>02.</td>
<td><strong>Search Area of Ghosts’ House.</strong> If all 4 ghosts are found in Ghosts’ House, location detection is completed. Otherwise, do 03.</td>
<td></td>
</tr>
<tr>
<td>03.</td>
<td><strong>Search Area around previous Ghost location.</strong> If cannot find the ghost, check if the previous position is close to tunnels. If so, set the ghost is inside tunnels. If not, check if the ghost overlaps with others in the previous moment. If so, set the ghost location same as the one overlap with it. Otherwise, do 04.</td>
<td></td>
</tr>
<tr>
<td>04.</td>
<td><strong>Check the whole image and see if ghost is found.</strong> If it cannot be found, set the ghost to be dead.</td>
<td></td>
</tr>
</tbody>
</table>

Special cases:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01.</td>
<td><strong>If ghosts are inside tunnels,</strong> search the two ends of tunnels. If cannot find, search the whole image.</td>
</tr>
<tr>
<td>02.</td>
<td><strong>If ghosts are set to be dead,</strong> search the whole image.</td>
</tr>
</tbody>
</table>

By using the procedures in table 4.4.01, the program can find the location and moving direction of Ghosts accurately when Ghosts are in Strong state no matter when they are travelling through tunnels or overlap with others completely. The program can also find the location of Ghosts when they are in Weak or In-Between states. However, when Weak or In-Between state Ghosts are travelling through tunnels or overlap with each other completely, the program do not know. Besides, the moving direction of ghosts in Weak and In-Between states cannot be found.
Table 4.4_02 shows the methods and data of Ghosts.

Table 4.4_02: Methods and Data in Ghost.java

Ghost.java

Methods:
public static void init_ghost();
public static void update_ghost(BufferedImage image);

Data:
public static int[] board_x = new int[4];
public static int[] board_y = new int[4];
public static int[] dir = new int[4];
public static int[] status = new int[4];
public static boolean[] in_tunnel = new boolean[4];

“Ghost.init_ghost();” initials Ghosts values in the beginning of each Game Level.

“Ghost.update_ghost(sub-image3);” updates the location of Ghosts.

“Ghost.board_x[i]” and “Ghost.board_y[i]” are (x, y) coordinate in the 2D Array Road Map.

“Ghost.dir[i]” is the moving direction of each Ghost.

“Ghost.status[i]” shows if a Ghost is in Strong, Weak, In-Between state or Dead.

“Ghost.in_tunnel[i]” is true if [i] is inside tunnels.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: Neutral</td>
<td>0: Strong</td>
</tr>
<tr>
<td>1: up</td>
<td>1: Weak</td>
</tr>
<tr>
<td>2: right</td>
<td>2: In-Between</td>
</tr>
<tr>
<td>3: down</td>
<td>3: Dead</td>
</tr>
<tr>
<td>4: left</td>
<td></td>
</tr>
</tbody>
</table>
4.5 Update Game States

Update Game States makes use of all sub-images.

Besides 3 major parts (constructing 2D Array Road Map, finding locations of Ms. Pac-Man & Ghosts and Developing an Algorithm ), other minor parts are combined into Game States. Table 4.5_01 lists all minor parts.

Table 4.5_01: Lists of Game States

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Position of Ms. Pac-Man game</td>
</tr>
<tr>
<td>02</td>
<td>Game Over</td>
</tr>
<tr>
<td>03</td>
<td>Initial program parameters</td>
</tr>
<tr>
<td>04</td>
<td>Number of Marks</td>
</tr>
<tr>
<td>05</td>
<td>Number of Lives remained</td>
</tr>
<tr>
<td>06</td>
<td>Game Level</td>
</tr>
<tr>
<td>07</td>
<td>Display Update</td>
</tr>
</tbody>
</table>

In Table 4.5_01, there are 7 Game States. Following explains Game States one by one.

Figure 4.5_01 shows the four sub-images.

**Figure 4.5_01: 4 Sub-Images**

01. Position of Ms. Pac-Man game

Before starting playing Ms. Pac-Man game, program checks if position of Ms. Pac-Man game is placed properly. This is done by checking Sub-Image 1.

02. Game Over
To check if Game is over, Sub-Image 4 would be used. Figure 4.5_02 shows Game Over Image.

![Game Over Image](image)

Figure 4.5_02: Game Over Image

An ellipse in white color is shown on Figure 4.5_02. When Game does not start yet or game over, a word “CREDIT” is always shown at the left bottom corner.

03. Initial program parameters

When “READY!” is shown in the game image, program initials parameters such as the initial location of Ms. Pac-Man & Ghosts. Figure 4.5_03 shows game image with a word “READY!”.

![Game Image with word “READY!”](image)

Figure 4.5_03: Game Image with word “READY!”

“READY!” is shown when entering a new Game Level or after Ms. Pac-Man is killed. If entering a new Game Level, program reconstructs 2D Array Road Map and initials location of Ms. Pac-Man & Ghosts. If Ms. Pac-Man is killed by Ghosts and continues to play the game, program keeps the 2D Array Road Map and only initials location of Ms. Pac-Man & Ghosts.
04. Number of Marks

Sub-Image 2 is used to find number of Marks.

05. Number of Lives Remained

Sub-Image 4 is used to count the number of Lives Remained.

06. Game Level

The Software Controller inserts coins and starts playing Ms. Pac-Man game automatically. When it starts the game, the program initials the Game Level as 1. Every time when “READY!” is shown, compare the number of Lives Remained to previous record. If the number of Lives Remained is the same as the previous record, this is a New Game Level. Thus, update the Game Level plus 1. If the number of Lives Remained is less than previous record, update the previous number of Lives and keep the Game Level unchanged.

07. Display Update

Every time, after updating 2D Array Road Map and location of Ms. Pac-Man & Ghosts, update the Display to show changes.

These 7 Game States are done by Game_Flow.Functions.java, Pacman.Pacman.java and Objects.Display.java
4.6 Development of Algorithm

The main idea of the algorithm is to estimate the next Ghosts’ moving direction. If distance between Ms. Pac-Man and Ghosts are too close, Ms. Pac-Man is in danger. When Ms. Pac-Man eats pills, power pills and Weak state Ghosts, it earns Marks. The responsibility of the Algorithm is to find out which moving direction is not danger and can earn more Marks.

About the estimation of Ghosts’ next moving direction, a thought is that no matter which direction a Ghost moves, if it wants to kill Ms. Pac-Man, it needs to choose a direction which can approach to Ms. Pac-Man at least one time.

There are many considerations for developing an algorithm. For instance, when to eat power pills, the moving speed of Ms. Pac-Man & Ghosts in different situations, the length of time for Ghosts change from Weak to Strong state. During development of algorithm, some important considerations may be missed. After development of algorithm, some considerations may be found useless. As a result, the program structure should allow writers to modify program without big change. Hence, “Recursion” is used as a core part in the program.

4 main factors are developed in the Algorithm. Table 4.6_01 lists these four factors.

Table 4.6_01: Four Main Factors in Algorithm

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01.</td>
<td>Marks for each Direction</td>
</tr>
<tr>
<td>02.</td>
<td>Ms. Pac-Man’s Moving Speed</td>
</tr>
<tr>
<td>03.</td>
<td>Length of time for a Ghost keeps in Weak State</td>
</tr>
<tr>
<td>04.</td>
<td>Ghosts’ Moving Direction Estimation</td>
</tr>
</tbody>
</table>

Following describes the Four Main Factors one by one.
01. Marks for each Direction

Different situations would be awarded with different marks. Table 4.6.02 shows different situations and its corresponding marks awarded.

Table 4.6.02: Different Situations and its Corresponding Marks award

<table>
<thead>
<tr>
<th>Situations</th>
<th>Award</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Eat Pills</td>
<td>+10 + num</td>
</tr>
<tr>
<td>B. Eat Power Pills</td>
<td>-1,000</td>
</tr>
<tr>
<td>C. Many Pills Left and Eat Power Pills</td>
<td>-15,000</td>
</tr>
<tr>
<td>D. Eat Weak Status Ghost</td>
<td>+75,000 + num×100</td>
</tr>
<tr>
<td>E. M&amp;G Distance &lt; 3</td>
<td>-600,000</td>
</tr>
<tr>
<td>F. M&amp;G Distance &gt;=3 &amp; &lt;=10</td>
<td>-10,000</td>
</tr>
</tbody>
</table>

“M&G Distance” means Distance between Ms. Pac-Man & Strong State Ghosts.

“num” is an integer that inversely proportional to distance between Ms. Pac-Man and Objects. Figure 4.6.01 shows an example of num.

In figure 4.6.01, distance for Ms. Pac-Man to pills A is shorter than to pills B. Thus, value of “num” for pills A is greater.

A. With “num”, Ms. Pac-Man would eat pills which are closer to her.

B. In order only to eat power pills when Ghosts are sufficiently close to Ms. Pac-Man. Such that after eating power pills, Ms. Pac-Man can eat Weak State Ghosts.
C. If many pills are left, try to eat pills first. This is to prevent all power pills are eaten by Ms. Pac-Man while still many pills are left which increases difficulties for Ms. Pac-Man to complete that Game Level.

D. To encourage Ms. Pac-Man to eat Weak State Ghosts and also the closest one.

E. To avoid Ms. Pac-Man from going towards to Strong State Ghosts.

F. To avoid Ms. Pac-Man staying too close with Strong State Ghosts.

02. Ms. Pac-Man’s Moving Speed

When Ms. Pac-Man is eating pills, its’ moving speed is a bit slower than Strong State Ghosts. When Ms. Pac-Man turns to two sides of its moving direction, its moving speed is faster than Strong State Ghosts. Due to these two observations, table 4.6_03 shows two situations and its corresponding speed awarded.

<table>
<thead>
<tr>
<th>Situations</th>
<th>Award</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Eat Pills</td>
<td>-10</td>
</tr>
<tr>
<td>B. Turning Two sides of its current moving direction</td>
<td>+100</td>
</tr>
</tbody>
</table>

If Speed <= -10, Ms. Pac-Man stay at the same location for 1 step.

This Speed Calculation is just an estimation used for Algorithm. The performance in the Ms. Pac-Man game is totally difference.

A. To slow down Ms. Pac-Man’s moving speed.

B. To allow Ms. Pac-Man to eat more pills without slowing down the speed.

03. Length of time for a Ghost keeps in Weak State

The Length of time for a Ghost to change from Weak State back to Strong State is different in each Game Level. However, in the Algorithm, there is only one constant
value for this maintenance. The length of time is 200 which means length of time for Ms. Pac-Man walks 200 Steps. One step is equal to one unit in 2D Array Road Map which is the same as 2 pixels distance. Therefore, after Ms. Pac-Man walking 400 pixels distance, it is assumed that Ghosts would change from Weak State back to Strong State.

04. Ghosts’ Moving Direction Estimation

With a thought that Ghosts would approach to Ms. Pac-Man, figure 4.6_02 shows an Example to estimate the next Ghost moving direction.

\[
\begin{align*}
\text{Length}_x &= \text{Ghost}_x - \text{Agent}_x \\
\text{Length}_y &= \text{Ghost}_y - \text{Agent}_y \\
(\text{Length}_x, \text{Length}_y)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Prediction Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\uparrow)</td>
</tr>
<tr>
<td>((+, -))</td>
</tr>
<tr>
<td>((-), +)</td>
</tr>
<tr>
<td>((-), -)</td>
</tr>
</tbody>
</table>

Figure 4.6_02: An Example of Next Ghost Moving Direction

In figure 4.6_02, \(\text{Length}_x\) is equal to \(x\)-coordinate of Ghost minus \(x\)-coordinate of Ms. Pac-Man in 2D Array Road Map. \(\text{Length}_y\) is equal to \(y\)-coordinate of Ghost minus \(y\)-coordinate of Ms. Pac-Man in 2D Array Road Map.

The Prediction Table in figure above consists of 2 different moving direction estimations. The left one is for Blinky and Pinky while the right one is for Inky and Sue.

In the right hand side of the figure, suppose Blinky is moving up. The moving direction estimation on the left hand side would be used. If Ms. Pac-Man is on the left top corner which means \(\text{Length}_x \geq 0\) and \(\text{Length}_y \geq 0\), program will estimate Blinky moving left. If
Blinky cannot move left, program will estimate it moving up. If Blinky cannot move up, program will estimate it moving right. If Ms. Pac-Man is on the right top corner which means Length\_x < 0 and Length\_y ≥ 0, program will estimate Blinky moving right. If Blinky cannot move right, program will estimate it moving up. If still Blinky cannot move up, program will estimate it moving left.

Figure 4.6_03 shows a full Moving Direction Estimation Table.

The recursion in the program Algorithm stops if it has repeated 150 times or Marks is less than -300,000 which means Ms. Pac-Man would be killed if it walks along this road. Direction with the highest Marks would be set as the moving direction for Ms. Pac-Man.

Program of Algorithm is written in Algorithm.PlayGame.java
5 Result

Table 5.01: 50 Plays by Software Controller.

<table>
<thead>
<tr>
<th></th>
<th>1st 10 Plays</th>
<th>2nd 10 Plays</th>
<th>3rd 10 Plays</th>
<th>4th 10 Plays</th>
<th>5th 10 Plays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plays</td>
<td>12450</td>
<td>16300</td>
<td>11930</td>
<td>5280</td>
<td>5160</td>
</tr>
<tr>
<td></td>
<td>13420</td>
<td>14480</td>
<td>16420</td>
<td>4860</td>
<td>12000</td>
</tr>
<tr>
<td></td>
<td>11540</td>
<td>17660</td>
<td>5760</td>
<td>26600</td>
<td>5880</td>
</tr>
<tr>
<td></td>
<td>4670</td>
<td>7290</td>
<td>11970</td>
<td>19400</td>
<td>7930</td>
</tr>
<tr>
<td></td>
<td>15150</td>
<td>7010</td>
<td>8750</td>
<td>8130</td>
<td>8200</td>
</tr>
<tr>
<td></td>
<td>30870</td>
<td>17410</td>
<td>9470</td>
<td>8770</td>
<td>14380</td>
</tr>
<tr>
<td></td>
<td>2680</td>
<td>6800</td>
<td>18560</td>
<td>15430</td>
<td>15190</td>
</tr>
<tr>
<td></td>
<td>17400</td>
<td>19830</td>
<td>18790</td>
<td>13810</td>
<td>31720</td>
</tr>
<tr>
<td></td>
<td>21120</td>
<td>15160</td>
<td>18550</td>
<td>13340</td>
<td>3110</td>
</tr>
<tr>
<td></td>
<td>18490</td>
<td>3240</td>
<td>17600</td>
<td>16490</td>
<td>16130</td>
</tr>
<tr>
<td>Average</td>
<td>14779</td>
<td>12518</td>
<td>13780</td>
<td>13211</td>
<td>11970</td>
</tr>
<tr>
<td>Max</td>
<td>30870</td>
<td>19830</td>
<td>18790</td>
<td>26600</td>
<td>31720</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>8070.848</td>
<td>5825.396</td>
<td>4797.678</td>
<td>6764.279</td>
<td>8273.41</td>
</tr>
</tbody>
</table>

The highest score of each 10 Plays is greater than the highest score in IEEE WCCI 2008 Ms. Pac-Man Competition. However, the Standard Deviation is large, which means the performance of the software controller is not stable.
6 Discussion

In this section, Limitation of the Software Controller would be discussed. Then, ideas on how to improve the performance would be mentioned.

6.1 Limitation

From Result, the Standard Deviation of each 10 plays is large. As mentioned in the Result, this is due to unstable performance of the Software Controller playing Ms. Pac-Man Game. The reason for this unstable performance is highly related to the estimation of Ghosts’ next moving direction.

The estimation of Ghosts’ next moving direction only considers 4 cases which are both Length_x & Length_y are positive, either Length_x or Length_y are positive and both Length_x & Length_y are negative. Due to this rough estimation of Ghosts’ next moving direction, Ghosts’ next moving direction cannot be estimated accurately.

In fact, it is impossible to estimate Ghosts’ next moving direction 100% accurate since Ms. Pac-Man game is a non-deterministic game [1]. The meaning of non-deterministic is that even the moving path of Ms. Pac-Man is the same for two plays, the moving path of Ghosts can be different.

As a result, the performance of the Software Controller sometimes performs better and sometimes poorer.
6.2 Ideas for Improvement

Refer to the limitation of the Software Controller, two ideas can be used to improve the performance. They are stated in the following one by one.

First Idea:

The current Software Controller only considers 4 cases for estimating Ghosts’ next moving direction. To increase the accuracy of estimation, 8 cases consideration can be used. Figure 6.2_01 shows 8 cases consideration.

![Figure 6.2_01: 8 Cases Consideration](image)

However, obviously, the 8 cases consideration is not guaranteed to have a high accuracy of estimation. That’s why Second Idea is stated.

Second Idea:

Since Ms. Pac-Man game is non-deterministic, to estimate Ghosts’ next moving direction is meaningless. Human players who can gain a high score in Ms. Pac-Man game never estimate the next moving direction of Ghosts but they will choose a road that no matter how Ghosts move, Ms. Pac-Man can still be safe.
To perform this idea, program needs to calculate all combinations of Ghosts’ next moving direction. Table 6.2_01 shows a comparison of current Software Controller and the first & second idea.

Table 6.2_01: Comparison of Software Controller and First & Second Idea

<table>
<thead>
<tr>
<th></th>
<th>Software Controller</th>
<th>First Idea</th>
<th>Second Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Poor</td>
<td>Better than Poor</td>
<td>Good</td>
</tr>
<tr>
<td><strong>Computational Time</strong></td>
<td>Constant (Short)</td>
<td>Constant (Short)</td>
<td>Long</td>
</tr>
</tbody>
</table>

A Road that can allow Ghosts turning up, right, down and left

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Blinky</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Pinky</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Inky</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Sue</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The estimated performance of second idea is no doubt the best. However, the Computational Time is much longer than the current Software Controller and First Idea. Consider a road which can allow Ghosts turning up, right down and left, Each Ghost has 3 choices (Ghosts cannot turn back). The program of current Software Controller and First Idea would only do calculation for 1 time while program for Second Idea would do it for 81 times. Therefore, the estimated performance of Second Idea is good but the actual performance cannot be guaranteed to be well. However, it is worth to try out the Second Idea.
7 Conclusion

A Software Controller is developed to play Ms. Pac-Man game. The highest score for this Software Controller is about 32,000 which is about twice of the highest score in IEEE WCCI 2008 Ms. Pac-Man Competition. However, this score is still far from the highest score by human player. The performance of the Software Controller is unstable which is implied by high value of Standard Deviation. The reason for this instability is mainly due to inaccurate Estimation of Ghosts’ Next Moving Direction.
8 Reference

[1] “Information for Ms. Pac-Man Competition”
http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html, April 16, 2009.
