Final Project

Object Oriented Programming - COMP6699001

Project Name: **Classic Pac**

Name: **Avariq Fazlur Rahman**

Student ID: **2502043002**

Class: **L2AC**

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Avariq Fazlur Rahman

Avariq Fazlur Rahman

Project Specification

This project is based on a classic, old, yet very popular arcade game ; Pacman. This project was made to relieve the experience of the old classic Pacman game, and unchanged as it is fully reminiscent to the old game. This project is a single-player, arcade-like game that uses keyboard controls to move the character (in this case, Pacman) to collect as many edibles as we can whilst avoiding the enemy (in this case, the Ghosts). The way this game works is exactly like the arcade version and it uses keyboard inputs to move the character. This game also has a pretty small maze-like map to make it look simple yet still quite challenging to play.

Input

1. User directional movements (Player 1)

Output

1. Movement of Pacman (Player 1)

Solution Design

- Main Game

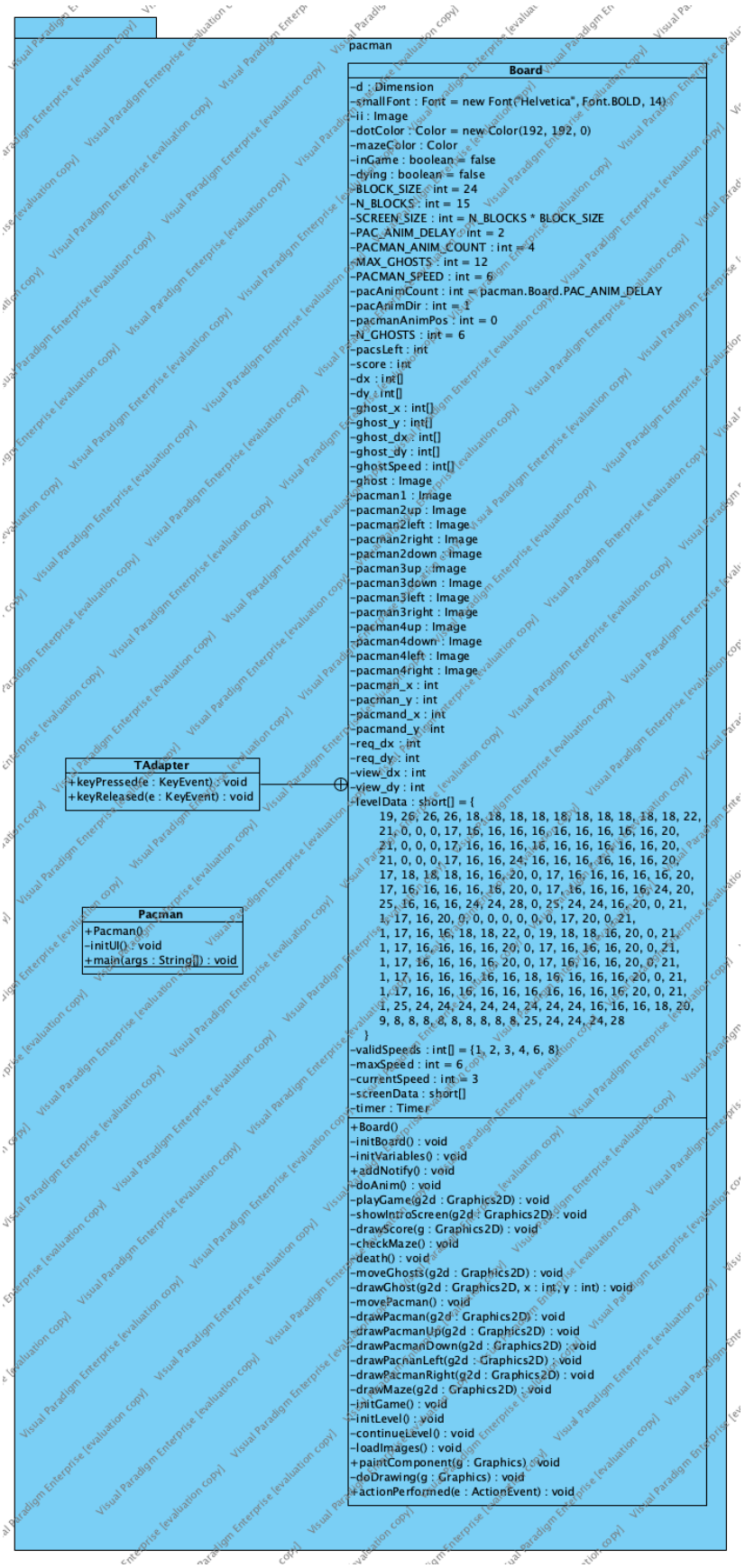
Main Game

This is the window that shows the main game. As far as graphics go, like stated above, this game looks exactly like the arcade version of Pacman. We have 4 yellow Pac-mans, 6-12 Ghosts, a maze map, and the edibles. We also have a start screen before we enter the main game but that is only a start sequence as basically we press a button to start the game. Once

your life is depleted in the game, there is also a Game Over sequence where it asks if you want to play again or exit the game.

UML Diagram

As understood, this project uses classes. There is the main class, where we go ahead and start the game and there is the main game class, where the game physics, all the characters, and the movement is taken care of. The UML diagram of this project looks a little bit like this :


pacman
Board
-d : Dimension
-smallFont : Font = new Font("Helvetica", Font.BOLD, 14)
-ii : Image
-dotColor : Color = new Color(192, 192, 0)
-mazeColor : Color
-inGame : boolean = false
-dying : boolean = false
-BLOCK_SIZE : int = 24
-N_BLOCKS : int = 15
-SCREEN_SIZE : int = N_BLOCKS * BLOCK_SIZE
-PAC_ANIM_DELAY : int = 2
-PACMAN_ANIM_COUNT : int = 4
-MAX_GHOSTS : int = 12
-PACMAN_SPEED : int = 6
-pacAnimCount : int = pacman.Board.PAC_ANIM_DELAY
-pacAnimDir : int = 1
-pacmanAnimPos : int = 0
-N_GHOSTS : int = 6
-pacsLeft : int
-score : int
-dx : int[]
-dy : int[]
-ghost_x : int[]
-ghost_y : int[]
-ghost_dx : int[]
-ghost_dy : int[]
-ghostSpeed : int[]
-ghost : Image
-pacman1 : Image
-pacman2up : Image
-pacman2left : Image
-pacman2right : Image
-pacman2down : Image
-pacman3up : Image
-pacman3down : Image
-pacman3left : Image
-pacman3right : Image
-pacman4up : Image
-pacman4down : Image
-pacman4left : Image
-pacman4right : Image
-pacman_x : int
-pacman_y : int
-pacmand_x : int
-pacmand_y : int
-req_dx : int
-req_dy : int
-view_dx : int
-view_dy : int
-levelData : short[] = {
19, 26, 26, 26, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
21, 0, 0, 0, 17, 16, 16, 24, 16, 16, 16, 16, 16, 16, 20,
17, 18, 18, 18, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 20,
17, 16, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 16, 24, 20,
25, 16, 16, 16, 24, 24, 28, 0, 25, 24, 24, 16, 20, 0, 21,
1, 17, 16, 20, 0, 0, 0, 0, 0, 0, 17, 20, 0, 21,
1, 17, 16, 16, 18, 18, 22, 0, 19, 18, 18, 16, 20, 0, 21,
1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,
1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,
1, 17, 16, 16, 16, 16, 16, 18, 16, 16, 16, 16, 20, 0, 21,
1, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 21,
1, 25, 24, 24, 24, 24, 24, 24, 24, 24, 16, 16, 16, 18, 20,
9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 25, 24, 24, 24, 28
}
-validSpeeds : int[] = {1, 2, 3, 4, 6, 8}
-maxSpeed : int = 6
-currentSpeed : int = 3
-screenData : short[]
-timer : Timer
+Board()
-initBoard() : void
-initVariables() : void
+addNotify() : void
-doAnim() : void
-playGame(g2d : Graphics2D) : void
-showIntroScreen(g2d : Graphics2D) : void
-drawScore(g : Graphics2D) : void
-checkMaze() : void
-death() : void
-moveGhosts(g2d : Graphics2D) : void
-drawGhost(g2d : Graphics2D, x : int, y : int) : void
-movePacman() : void
-drawPacman(g2d : Graphics2D) : void
-drawPacmanUp(g2d : Graphics2D) : void
-drawPacmanDown(g2d : Graphics2D) : void
-drawPacmanLeft(g2d : Graphics2D) : void
-drawPacmanRight(g2d : Graphics2D) : void
-drawMaze(g2d : Graphics2D) : void
-initGame() : void
-initLevel() : void
-continueLevel() : void
-loadImages() : void
+paintComponent(g : Graphics) : void
-doDrawing(g : Graphics) : void
+actionPerformed(e : ActionEvent) : void
TAdapter
+keyPressed(e : KeyEvent) : void
+keyReleased(e : KeyEvent) : void
Pacman
+Pacman()
-initUI() : void
+main(args : String[]) : void

Implementation and explanation of code

```
private int pacman_x, pacman_y, pacmand_x, pacmand_y;
```

The first two variables store the x and y coordinates of the Pacman sprite. The last two variables are the delta changes in horizontal and vertical directions.

```
private final short levelData[] = {
    19, 26, 26, 26, 18, 18, 18, 18, ...
};
```

These numbers make up the maze. They provide information out of which we create the corners and the points. Number 1 is a left corner. Numbers 2, 4 and 8 represent top, right, and bottom corners respectively. Number 16 is a point. These numbers can be added, for example number 19 in the upper left corner means that the square will have top and left borders and a point (16 + 2 + 1).

```
private void doAnim() {

    pacAnimCount--;

    if (pacAnimCount <= 0) {
        pacAnimCount = PAC_ANIM_DELAY;
        pacmanAnimPos = pacmanAnimPos + pacAnimDir;

    if (pacmanAnimPos == (PACMAN_ANIM_COUNT - 1) || pacmanAnimPos == 0)
{
            pacAnimDir = -pacAnimDir;
        }
    }
}
```

The `doAnim()` counts the `pacmanAnimPos` variable which determines what pacman image is drawn. There are four pacman images. There is also

a PAC_ANIM_DELAY constant which makes the animation a bit slower. Otherwise the pacman would open his mouth too fast.

```
boolean finished = true;

while (i < N_BLOCKS * N_BLOCKS && finished) {

    if ((screenData[i] & 48) != 0) {
        finished = false;
    }

    i++;
}
```

This code is part of the checkMaze() method. It checks if there are any points left for the Pacman to eat. Number 16 stands for a point. If all points are consumed, we move to the next level. (In our case, we just restart the game.)

Next we will examine the moveGhosts() method. The ghosts move one square and then decide if they change the direction.

```
if (ghost_x[i] % BLOCK_SIZE == 0 && ghost_y[i] % BLOCK_SIZE == 0) {
```

We continue only if we have finished moving one square.

```
pos = pacman_x / BLOCK_SIZE + N_BLOCKS * (int) (pacman_y / BLOCK_SIZE);
```

This line determines where the ghost is located; in which position/square. There are 225 theoretical positions. (A ghost cannot move over walls.)

```
if ((screenData[pos] & 1) == 0 && ghost_dx[i] != 1) {
    dx[count] = -1;
    dy[count] = 0;
```

```
    count++;
}
```

If there is no obstacle on the left and the ghost is not already moving to the right, the ghost will move to the left. What does this code really mean? If the ghost enters a tunnel, he will continue in the same direction until he is out of the tunnel. Moving of ghosts is partly random. We do not apply this randomness inside long tunnels because the ghost might get stuck there.

```
if (pacman_x > (ghost_x[i] - 12) && pacman_x < (ghost_x[i] + 12)
        && pacman_y > (ghost_y[i] - 12) && pacman_y < (ghost_y[i] + 12)
        && inGame) {

    dying = true;
}
```

If there is a collision between ghosts and Pacman, Pacman dies.

Next we are going to examine the `movePacman()` method. The `req_dx` and `req_dy` variables are determined in the `TAdapter` inner class. These variables are controlled with cursor keys.

```
if ((ch & 16) != 0) {
    screenData[pos] = (short) (ch & 15);
    score++;
}
```

If the pacman moves to a position with a point, we remove it from the maze and increase the score value.

```
if ((pacmand_x == -1 && pacmand_y == 0 && (ch & 1) != 0)
        || (pacmand_x == 1 && pacmand_y == 0 && (ch & 4) != 0)
        || (pacmand_x == 0 && pacmand_y == -1 && (ch & 2) != 0)
        || (pacmand_x == 0 && pacmand_y == 1 && (ch & 8) != 0)) {
    pacmand_x = 0;
```

```
    pacmand_y = 0;
}
```

The Pacman stops if he cannot move further it his current direction.

```
private void drawPacman(Graphics2D g2d) {

    if (view_dx == -1) {
        drawPacnanLeft(g2d);
    } else if (view_dx == 1) {
        drawPacmanRight(g2d);
    } else if (view_dy == -1) {
        drawPacmanUp(g2d);
    } else {
        drawPacmanDown(g2d);
    }
}
```

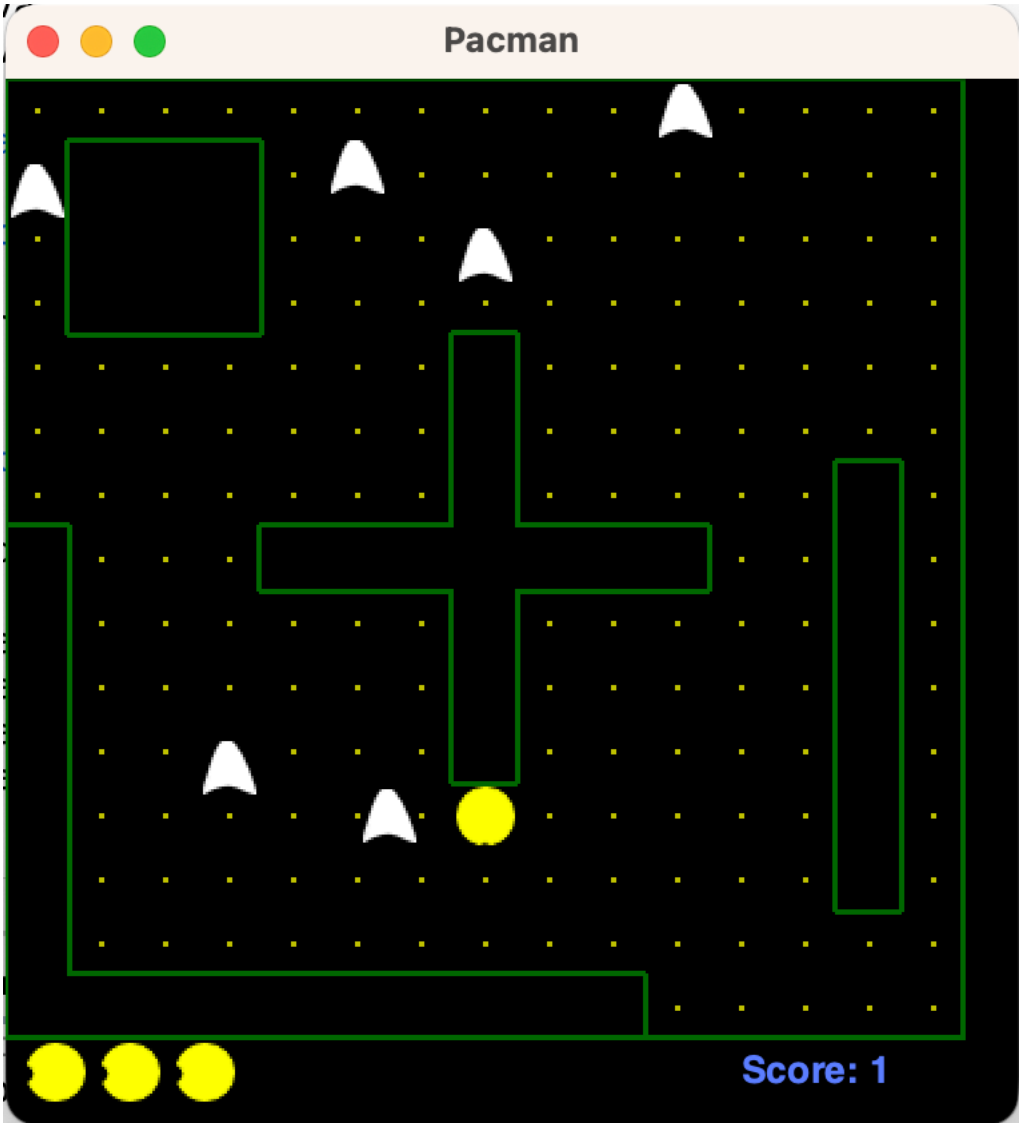
There are four possible directions for a Pacman. There are four images for all directions. The images are used to animate Pacman opening and closing his mouth.

The `drawMaze()` method draws the maze out of the numbers in the screenData array. Number 1 is a left border, 2 is a top border, 4 is a right border, 8 is a bottom border and 16 is a point. We simply go through all 225 squares in the maze. For example we have 9 in the `screenDataarray`. We have the first bit (1) and the fourth bit (8) set. So we draw a bottom and a left border on this particular square.

```
if ((screenData[i] & 1) != 0) {
    g2d.drawLine(x, y, x, y + BLOCK_SIZE - 1);
}
```

Finished Product

As the above section of this report stated the codes that were used, this was the end result of all of those codes above.



Lessons Learned

This project has taught me quite a lot about Java and how it works. Mostly from using classes, interfaces, and GUIs. It's also taught me a lot about 2D game developing. Whilst this is only the simple version, it's a good foundation moving forward for if I will create something in the future. GUI is also pretty important and I'm very glad that I got to learn it during this project as it is quite interesting and I look forward to more projects that

will require GUIs and designing interfaces. Overall, this project has taught me a lot about developing Java programs and has got me quite acquainted to Java and it's underlying features.

Resources

https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html

https://www.javatpoint.com/java-awt

https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html

Resource PNGs found from Google

https://www.youtube.com/watch?v=om59cwR7psI&list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq