**Java Snake game project: Brian**

Intro (rationale):

-Interest in topic

-The objective

Modern games are filled high quality graphics and stunning gameplay, but in order for the entertainment to develop that much there was a long period of trial, error, and improvement within the gaming coding industry.

Analysis:

* What are the requirements? What am I trying to fulfill?

I want to build a quick and simple game of snake that may help children cope with numbers and basic cognitive skills. I plan to begin by building a game of snake, the one everyone knows, to a more complex game in which user is required to be able to distinguish colors. E.g. if the snake eats the black apple it will die whereas eating red apple will extend its length.

-quick and simple entertainment

-bonding game for kids to play

- to entertain siblings

* Make the requirements/rationale very specific

Specs/requirements (criteria for success):

* The snake must start with 3 blocks
* The program must accept user’s input through arrow keys (up, left, right)
* Program must hold exit button that allows quit game
* Score display
* Snake must increase its size by one block when food is eaten
* Every time food is eaten, another one is generated on a random spot
* Game over message is displayed if the snake hits itself or the wall
* After a few stages the game should start spawning black apples, which will lead to game over when eaten.
* Score
* Who wins
* Will run on any platform with java virtual machine

Screen mock ups:

UML diagrams:

Modular diagram – hierarchy

A flow diagram of sort to show how the program 🡪 the logic of the program

Screen designs:

Testing success criteria:

“Action” – what happens within the testing, different actions for the test, which is the criteria I am looking for.

What I want to do 🡪 success criteria

Imported code:

In order to create a modified version of snake, I have implemented a basic source code for snake, written by Jaryt Bustard.The code holds two components: Snake.java and Renderpanel.java. The codes are as follows:

Renderpanel.java:

package snake;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Point;

import javax.swing.JPanel;

@SuppressWarnings("serial")

/\*\*

\* @author Jaryt Bustard

\*/

public class RenderPanel extends JPanel

{

public static final Color GREEN = new Color(1666073);

@Override

protected void paintComponent(Graphics g)

{

super.paintComponent(g);

Snake snake = Snake.snake;

g.setColor(GREEN);

g.fillRect(0, 0, 800, 700);

g.setColor(Color.BLUE);

for (Point point : snake.snakeParts)

{

g.fillRect(point.x \* Snake.SCALE, point.y \* Snake.SCALE, Snake.SCALE, Snake.SCALE);

}

g.fillRect(snake.head.x \* Snake.SCALE, snake.head.y \* Snake.SCALE, Snake.SCALE, Snake.SCALE);

g.setColor(Color.RED);

g.fillRect(snake.cherry.x \* Snake.SCALE, snake.cherry.y \* Snake.SCALE, Snake.SCALE, Snake.SCALE);

String string = "Score: " + snake.score + ", Length: " + snake.tailLength + ", Time: " + snake.time / 20;

g.setColor(Color.white);

g.drawString(string, (int) (getWidth() / 2 - string.length() \* 2.5f), 10);

string = "Game Over!";

if (snake.over)

{

g.drawString(string, (int) (getWidth() / 2 - string.length() \* 2.5f), (int) snake.dim.getHeight() / 4);

}

string = "Paused!";

if (snake.paused && !snake.over)

{

g.drawString(string, (int) (getWidth() / 2 - string.length() \* 2.5f), (int) snake.dim.getHeight() / 4);

}

}

}

Snake.java:

package snake;

import java.awt.Dimension;

import java.awt.Point;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.ArrayList;

import java.util.Random;

import javax.swing.JFrame;

import javax.swing.Timer;

/\*\*

\* @author Jaryt Bustard

\*/

public class Snake implements ActionListener, KeyListener

{

public static Snake snake;

public JFrame jframe;

public RenderPanel renderPanel;

public Timer timer = new Timer(20, this);

public ArrayList<Point> snakeParts = new ArrayList<Point>();

public static final int UP = 0, DOWN = 1, LEFT = 2, RIGHT = 3, SCALE = 10;

public int ticks = 0, direction = DOWN, score, tailLength = 10, time;

public Point head, cherry;

public Random random;

public boolean over = false, paused;

public Dimension dim;

public Snake()

{

dim = Toolkit.getDefaultToolkit().getScreenSize();

jframe = new JFrame("Snake");

jframe.setVisible(true);

jframe.setSize(805, 700);

jframe.setResizable(false);

jframe.setLocation(dim.width / 2 - jframe.getWidth() / 2, dim.height / 2 - jframe.getHeight() / 2);

jframe.add(renderPanel = new RenderPanel());

jframe.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

jframe.addKeyListener(this);

startGame();

}

public void startGame()

{

over = false;

paused = false;

time = 0;

score = 0;

tailLength = 14;

ticks = 0;

direction = DOWN;

head = new Point(0, -1);

random = new Random();

snakeParts.clear();

cherry = new Point(random.nextInt(79), random.nextInt(66));

timer.start();

}

@Override

public void actionPerformed(ActionEvent arg0)

{

renderPanel.repaint();

ticks++;

if (ticks % 2 == 0 && head != null && !over && !paused)

{

time++;

snakeParts.add(new Point(head.x, head.y));

if (direction == UP)

{

if (head.y - 1 >= 0 && noTailAt(head.x, head.y - 1))

{

head = new Point(head.x, head.y - 1);

}

else

{

over = true;

}

}

if (direction == DOWN)

{

if (head.y + 1 < 67 && noTailAt(head.x, head.y + 1))

{

head = new Point(head.x, head.y + 1);

}

else

{

over = true;

}

}

if (direction == LEFT)

{

if (head.x - 1 >= 0 && noTailAt(head.x - 1, head.y))

{

head = new Point(head.x - 1, head.y);

}

else

{

over = true;

}

}

if (direction == RIGHT)

{

if (head.x + 1 < 80 && noTailAt(head.x + 1, head.y))

{

head = new Point(head.x + 1, head.y);

}

else

{

over = true;

}

}

if (snakeParts.size() > tailLength)

{

snakeParts.remove(0);

}

if (cherry != null)

{

if (head.equals(cherry))

{

score += 10;

tailLength++;

cherry.setLocation(random.nextInt(79), random.nextInt(66));

}

}

}

}

public boolean noTailAt(int x, int y)

{

for (Point point : snakeParts)

{

if (point.equals(new Point(x, y)))

{

return false;

}

}

return true;

}

public static void main(String[] args)

{

snake = new Snake();

}

@Override

public void keyPressed(KeyEvent e)

{

int i = e.getKeyCode();

if ((i == KeyEvent.VK\_A || i == KeyEvent.VK\_LEFT) && direction != RIGHT)

{

direction = LEFT;

}

if ((i == KeyEvent.VK\_D || i == KeyEvent.VK\_RIGHT) && direction != LEFT)

{

direction = RIGHT;

}

if ((i == KeyEvent.VK\_W || i == KeyEvent.VK\_UP) && direction != DOWN)

{

direction = UP;

}

if ((i == KeyEvent.VK\_S || i == KeyEvent.VK\_DOWN) && direction != UP)

{

direction = DOWN;

}

if (i == KeyEvent.VK\_SPACE)

{

if (over)

{

startGame();

}

else

{

paused = !paused;

}

}

}

@Override

public void keyReleased(KeyEvent e)

{

}

@Override

public void keyTyped(KeyEvent e)

{

}

}

Code:

Evaluation:

As the progam requires logic and graphic interface: the OOP will hold multiple classes:

-main – the program is ran from here – it sets the main loop for the game

-Game Instance - holds game logic

- Snake – Holds everything, including information, which has to do with the snake

UML

Final code:

Testing: