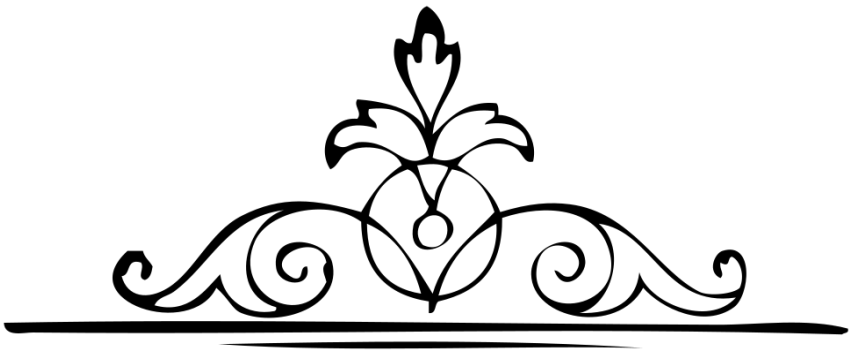
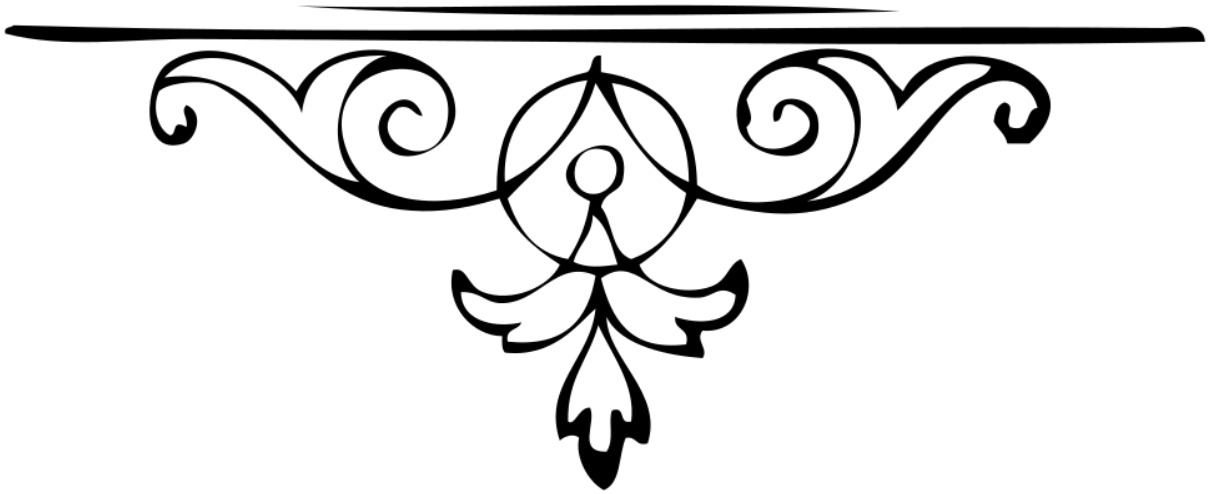


SNAKE GAME

Ethan Blits, Joscelyne Clasper, Andy Vu, Jessica Wuest



3/11/2023
EVERETT COMMUNITY COLLEGE

Overview:

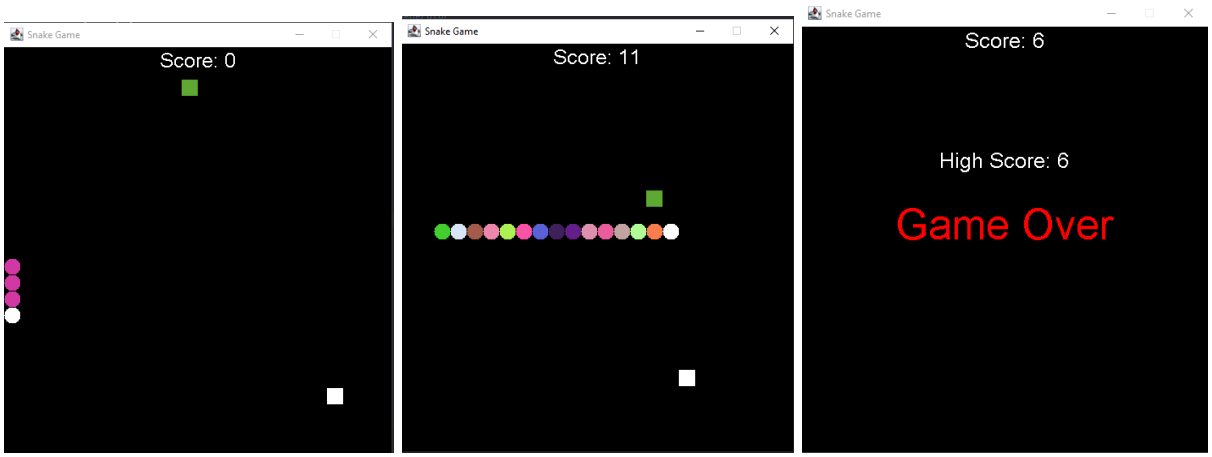
The program “Snake Game” is based on a popular game where the player is a snake and must eat apples to grow, and avoid walls to survive. In this version of the game, the player changes the direction of the snake by using the arrow keys on their keyboard. Apples and tomatoes appear on the screen, and the player must then avoid the tomatoes and “eat” the apples by navigating towards them. When the snake consumes an apple, it grows in size. If the player eats 10 apples and maintains a score above 10, the snake changes colors continuously. Eventually, the player loses the game by either running the head of the snake into its own tail or running into a wall. The player may also lose the game if the snake eats a tomato when the snake is too small to shrink.

Functionality: Entertainment

Implementation:

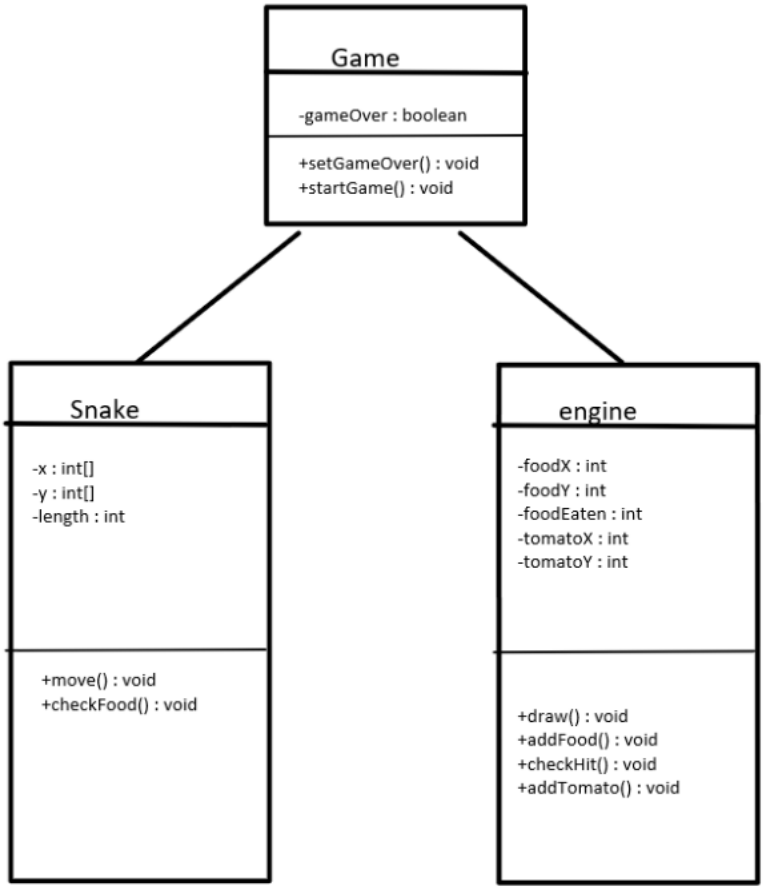
The SnakeGame class begins by including the main method which sets up the GameFrame. The GameFrame class creates a new instance of GamePanel while setting up the game window for the Panel to be executed onto. The GamePanel class contains the SnakeGame program. This implements ActionListeners to read user inputs to control the snake's movement. Throughout the GamePanel class, there are checks to see if the boolean "running" remains true. This boolean can be set to false by losing the game. As soon as "running" is set to false, the text "GAME OVER" is displayed. The "play" method initially sets "running" to true. During this time the "draw" method checks the boolean and creates the snake's body. After the snake's body is created, the colors are then assigned and the score text is written at the top. After the "play" method is executed, there are two check object methods: "checkFood", and "checkTomato". "checkTomato" first sees if there are any additional tomatos in the window before running "addTomato". The "addFood" method creates random units in the game's window space at random positions for the snake to collide with. Upon collision, this adds points to the score text at the top. After 10 food collisions, the color of the snake's body is altered to change colors continuously until "gameOver" The "addTomato" method also checks for collisions, however once collided the length of the snake as well as the score decreases by 1. Once the length reaches -1, the gameOver method is called. If any additional "addFood" collisions were recorded, they are displayed on the "GAME OVER screen" The "gameOver" method runs if at any point the snake collides with itself, or the wall.

Screenshots:

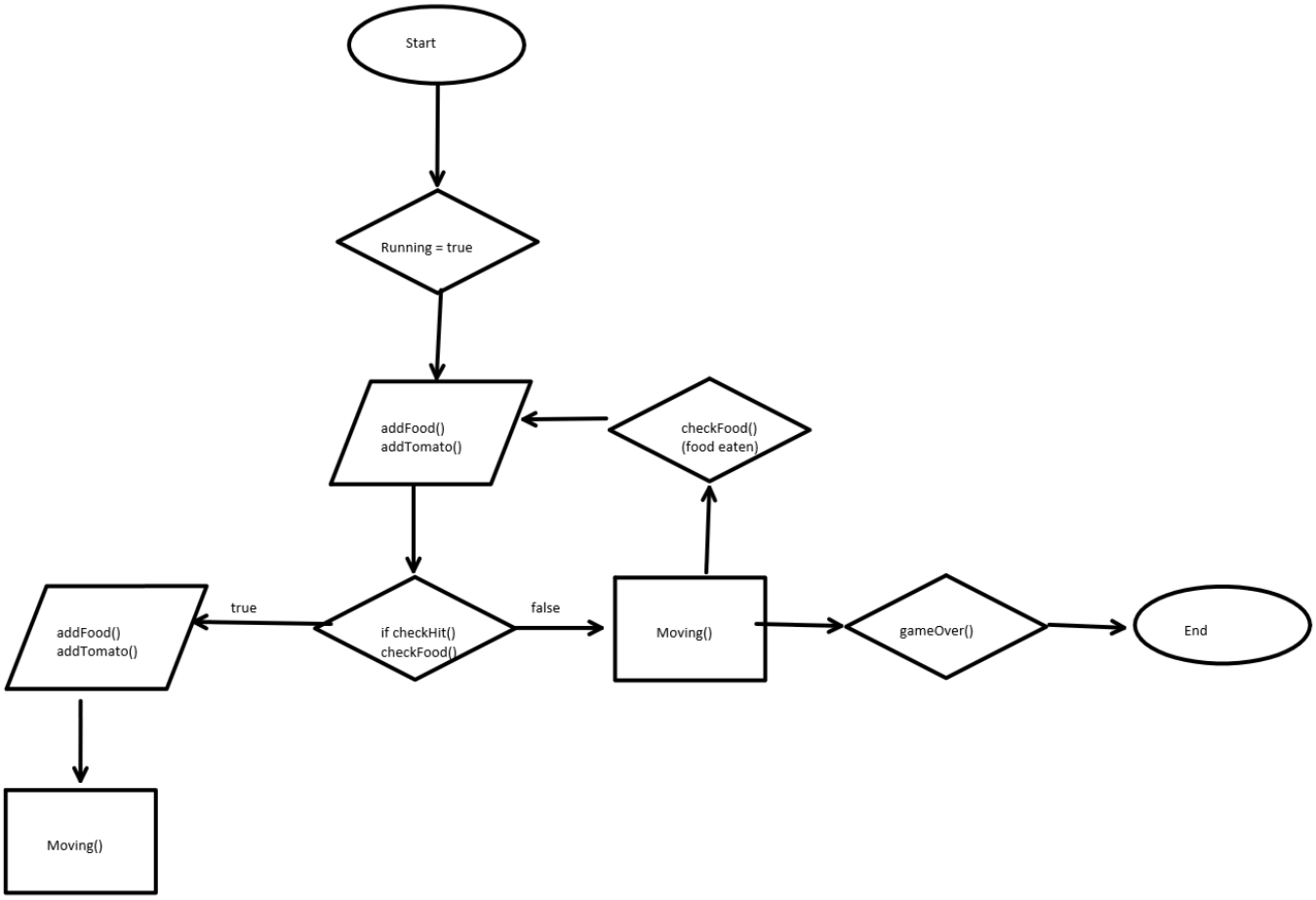


On startup **After 10 score** **On game end**

UML Diagram:



Flow Diagram:



Notes/Comments (Optional):

```
for (int i = 1; i < length; i++) {

    if (foodEaten >= 10) {
        r = rand.nextFloat();
        g = rand.nextFloat();
        b = rand.nextFloat();
    }

    graphics.setColor(new Color(r, g, b));
    graphics.fillOval(x[i], y[i], UNIT_SIZE, UNIT_SIZE);
}
```

```
//    float r = (float) (rand.nextFloat() + 0.01);
//    float g = (float) (rand.nextFloat() + 0.01);
//    float b = (float) (rand.nextFloat() + 0.1);
    6 usages
    float min = (float) 0.01;
    3 usages
    float max = (float) 1.0;
    2 usages
    float r = min + rand.nextFloat() * (max - min);
    2 usages
    float g = min + rand.nextFloat() * (max - min);
    2 usages
    float b = min + rand.nextFloat() * (max - min);
```

We encountered a problem when we implemented random colors for the snake on startup and when coming back down from a score of 10. There was a small chance that the random float would choose a dark color and the snake would disappear or be hard to see. We solved this issue by adding a limit to the float our random generator could choose.

```
        try {
            FileWriter myWriter = new FileWriter( fileName: "highscore.txt");
            myWriter.write(String.valueOf(a));
            myWriter.close();
            System.out.println("Successfully stored high score.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

We also had issues when implementing the reader/writer into our code for our local high score system. When writing an integer into a file, it would appear as letters; this is due to our writer not translating properly and, instead, writing the representation for an integer into our file. In order to fix this, we converted our integers into strings using “myWriter.write(String.valueOf(**a**));” (***a*** being out parameter for our writer method).