Snake Game Design

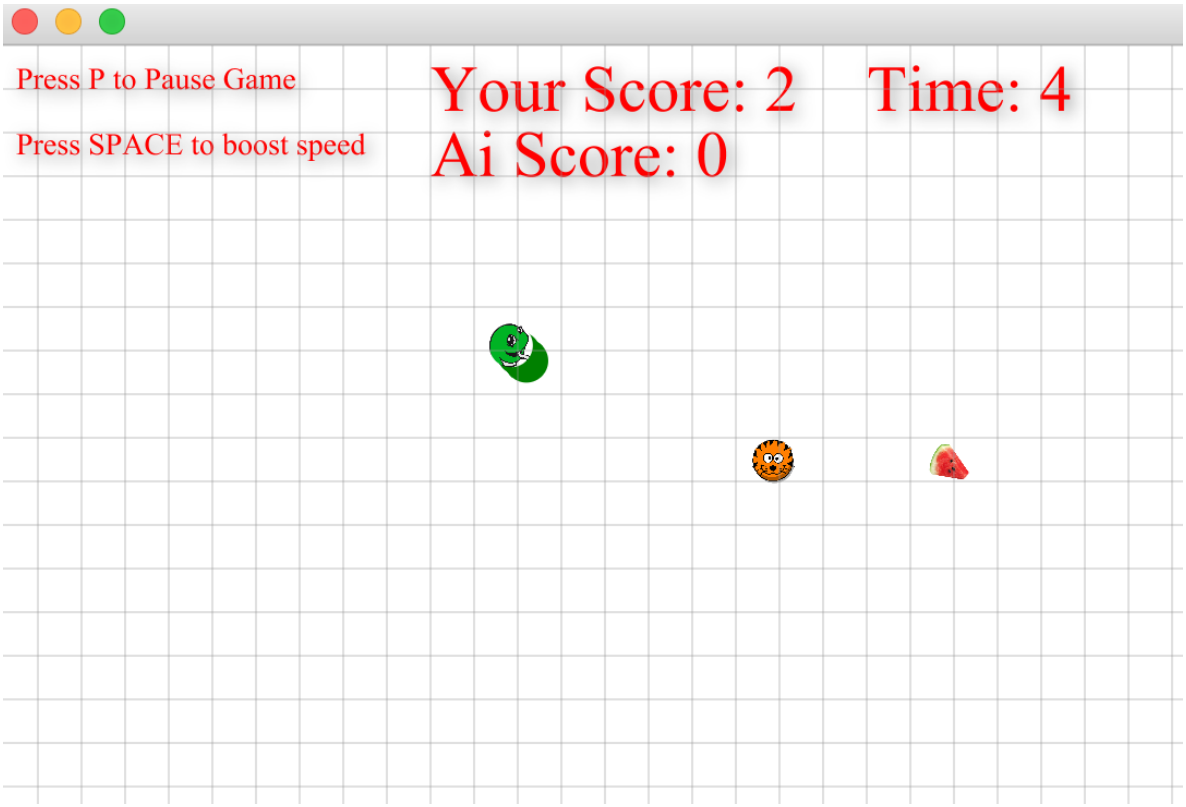

Figure 1: Snake Game Main Page

1 Game introduction

This is a snake game with a user snake (blue snake) and an AI snake (yellow Tiger). There is information at the top of the game page, such as scores, remaining time and keys information. The user needs to beat the AI snake by eating more food within 60 seconds. The user can control the snake to move with eight directions and can boost the speed by pressing SPACE on the keyboard. AI snake only focuses on the food, it moves directly to the food position. The snake can move beyond the wall, and come out again from the opposite wall. If the time is run out, game state will change to gameover, and show the scores information, the user can press R to restart the game.

2 Game states

In the beginning, the user can press S to start the game. During the running of the game, pressing P will pause the game, by pressing this key again it will restart from the pause state. If the game is over, pressing R can restart the game.

3 Game Design

3.1 Basic Requirements

This game is designed from a simple idea with following requirements:

- Snake can move
- Snake can grow
- Snake can eat food
- Snake can hit with other stuff

3.2 Snake class

In this class, the snake was defined as a list of coordinates. To save the coordinates, I designed the MyPoint2D class to save/set two numbers as the position. ArrayList is used to record positions. This class needs to have a testing as it's the basic logic of the snake game.

A first thing needed to consider is to decide the snake's movement, my design is to remove the last one and add the new head as the first element of this snake. If the snake only has one head, just set the first element with the new head without removing its previous tail.

For movement, eight directions were set as the snake's directions. Every step, head will move with a speed to next position according to the 8 directions.

To detect the collision with other position, an equation is used as the detection method.
This method is to check whether if the head hit the specific position by considering the object size (the size of food, snakehead).

This class runs methods to test the snake's movement, growing, collision with another position. It is very useful to test the snake movements in this class without adding

```
// check collision with one position by checking the four coordinates
boolean isHitWithPosition(MyPoint2D pos) {
    head = snakebody.get(0);
    return head.getX() < pos.getX() + Constants.objectSize
            && head.getX() + Constants.objectSize > pos.getX()
            && head.getY() < pos.getY() + Constants.objectSize
            && head.getY() + Constants.objectSize > pos.getY();
}
```

Figure 2: Collision detection method

any graphics contents as if the coding interface is correct, designing the graphic unit will be much easier and with more accuracy.

3.3 Screen class

This is the graphics unit with sample logic from the snake. This class is to add all components to the screen, update the snake position and game states, it will also monitor the key states to move the user's snake with correct direction.

This class can be illustrated by following steps. First, it will initialise all settings, such as the background lines, loading the food images, setting the initial position of snake and food and so on. Second, it will update all game logic, such as snake movements, counting the time, checking collision and boosting, monitoring the game over states. Third, all components will be drawn into the screen. At the same time, it tracks the key states to change game states and snakes directions. The reason for separating the game logic and the drawing part is to better control the game states without using the timeline to pause and start. The setting of pause and stop the game states is very easy to design in this structure. The game logic can stop updating, but the graphics contents are still shown on the screen. Previously, I put the timeline and keyframes in this class and found that it is better to put the main run loop in other classes as this game can have many screens. For this reason, I moved the timeline to SnakeApp class, and call the update method of the game screen to update snakes and draw components.

To design the countdown, I use a variable to count how many times of the game logic runs as every KeyFrame runs 0.05s as the duration time. Every 20 times of updating will be a period of one second. Previously, I used two variables to record the start time and stop time by using System.currentTimeMillis(), but the design has a problem that pausing the game cannot stop counting.

Drawing the snake can be made by keeping fill rectangles. I used the GraphicsCon-

text to draw snakes and foods. Every frame, the previous tail will be cleared. Although this design works, I think there can be a better way to draw these things without clearing the previous tail. The ideal design can be clearing all stuff on the screen and redraw everything in the next frame. I distinguished the user snake and ai snake by adding a boolean variable into their classes to mark their identity. The head of snake will be drawn first as this will put the head in front of bodies. Different images and body colour are used to fill snakes and food to make it attractive. Five game states control methods are used to change game state.

Music is added to this game to make the design process with more fun and also increase the attraction of the game.

Although monitoring key states can be put into other classes, I leave it in this class as this game uses simple key codes. The HashMap of KeyCode is made to save the key code and its states such as pressed/released. Boosting speed of snake used these key states. Moving the snake and changing game states need to know which key has been pressed, so I put the method of changing directions of the snake and dealing with game states in the event of setOnKeyPressed.

4 GameInfo, Food, AIsnake, MyPoint2D, Constants

GameInfo class is designed to show messages on the screen. Other methods mainly depend on the method of drawText to get the Text object with specific settings. I used the DoubleProperty type to monitor changes of its value by adding addListener in this property. With this type, the game info messages can be updated very smoothly.

Food class is just a position with methods to get/set its values. AIsnake is extending from Snake class by adding updateDirection method to renew its directions according to Food's position. MyPoint2D is just a class to include two values as the coordinate of objects with basic setting and getting methods. Constants class contains the important values of screen size and object size.

5 Submission version

Source code: Although the original code works fine in Eclipse, I disabled initialising the images patterns and music in Screen class in the final submission version as these two functions can lead to some errors when I compile it using Mac terminal. Now the code works fine with Mac terminal by using commands in "Run Instruction .txt" which located in the folder containing this file.

snakeGame.jar: This edition comes from source codes including Music and Images by exporting the Runnable JAR file in Eclipse.

6 Libraries and references

IDE: Eclipse

External Libraries: JavaFx-8

References:

1: `https://www.youtube.com/watch?v=ZiCa13y9D4s&t=832s`

2: `https://www.youtube.com/watch?v=nK6l1uVlunc&t=38s`

3: `https://github.com/silence1772/JavaFX-GreedySnake`

In the beginning, I used these resources to help me gain knowledge of how to combine the graphics modules with the game. I declared that I understood the snake game fully and re-wrote it all by myself.