MayZihan Zheng, Jianhua Yang

COMP 363

Dr Nick Hayward

**Snake Game: Final Project Report**

**Content:**

* **Project Participants**
* **Project Narrative**
* **Design consideration**
* **Testing and iterative design**
* **Restrictions, limitations, and constraints**
* **Conclusion**

**List of Project Participants:**

Project Members: Zihan Zheng and Jianhua Yang

In this snake project, Zihan and Jianhua are both focusing on the code development. We separate this project into several classes. For contribution, Zihan will mainly focus on GameView, GameController, SnakeApp, Setting, and Score classes. Meanwhile, Jianhua will mainly focus on Snake, Grid, Direction, Node and Login classes. All of our project members are doing presentation, report and DEV videos together. Both Zihan and Jianhua communicatied a lot during the project development to exchange ideas.  This project also cultivates our teamwork ability. In cooperation, each of us contributes our own strength and performs our duties.

**Abstract of the Project:**

Snake is a very classic game in the world. It can be said to be one of the pioneers of electronic games. In the late 1990s, as Nokia mobile phones became the world's mainstream mobile phone platform, Snake game became popular as the pre-installed game in Nokia phones. Although its gameplay was simple to the limit, the player used the arrow keys to manipulate a long snake to continuously swallow the beans. At the same time, the body of the snake continued to grow with the swallowed beans. The game is over when the body or barrier is over. Almost everyone knows and has played this game. It is no exaggeration to say that Snake is probably the game with the most players in the world. Most computer games nowadays are full of gorgeous special effects and cool game graphics. But the essence of the game is gameplay. This is the reason why we chose Snake Game as the project. We want to use what we have learned in computer science to restore this classic game.

**Project Narrative:**

We want to design the snake game through java language. Java is an object-oriented language. A Java program is a collection of a series of Objects. The objects cooperate with each other to complete specific functions through method calls. Snake is obviously a game suitable for java language. Its game rules are very simple, and the game interface is also very simple. In terms of controlling the game, we hope to implement the Model View Controller framework in the game. To explain MVC in details:

* Model is a model, and all data of the program is stored in the model.
* View is a view, which gets data from the model and outputs it.
* Controller is the controller, responsible for the business logic of the entire program, controlling data processing and output.

For example, snakes and food in the game, these specific game components belong to the Model in the MVC framework. Without these, the game cannot be run successfully. Separating the various parts of the program through this framework structure makes the context of the entire program clear, improves the efficiency of programming, and effectively reduces the coupling between classes and improves the scalability of the program . We want to combine Model, View and Controller by using the LinkList algorithm. We think that LinkList will combine each class very well, and it is easy to control.

We also hope to focus on user interaction in this game. We refer to the client of the online game, which allows the user to enter the account and password. Although we don't have a server that allows users to compete online (that would cost a lot of money), we want to store the user's game score and other information by entering the account and password to enhance user interaction. Users will maintain a sense of accomplishment when they reuse the app. Users should also be able to do some other adjustment in the client.

**Design consideration**

***Original ideas and concepts:***

We want to create an app called Snake Game, which is a classical game that users could control the snake to eat food to get points. The length of the snake will increase as the score increases. The game will end immediately when the snake touches a wall or itself, then the score will be settled. In the original idea, we hope to turn this game into an online game, which allows players to login to their accounts and save the player's history for a long time. On the basis of the above content, we also want to add a “custom setting” system which allows users to change the setting of the game, like windows size, music on/off, or even difficulty of game.

***Example of other projects*:**

We didn’t intend to separate all the functions into different classes first until seeing a lot of people use MVC (model-VIew-Controller) pattern in their program. By learning Jbberinger’s code in his github’s Snake repository, we found that using MVC is a way to separate concerns which let us easily maintain the application. For instance, we can split our work for different people at a time, and it will not affect one developer work to another developer work. Also, it can improve the readability of cour code. If we write those classes together and add a lot of comments afterwards, it will make our code particularly confusing. So inspired by Jbberinger, we decided to start our project through the MVC pattern.

***Primary goal of the chosen algorithms for this project:***

Since we want to use nodes to represent the snake, we can define a member variable of a collection type for Snake, and let the collection save all the nodes. Also we found that snakes keep getting longer because there are nodes inserted into the nodes’ collection. More important, we do not need to randomly visit a certain node in Snakes, so we decided to use linked lists as our data structure.

***Responded to feedback and review from the DEV week:***

We made two main changes after DEV week. The first change we made is to add a new class for our program. Since we want our application to look like a game client, we decided to add a login system which is a new window shown before the game starts, and users were allowed to type in “username” and “password”. Through the DEV week’s feedback and reading a lot of search materials, we finally decided to use the Java swing. Because the “javax.swing” package provides classes for java swing API such as JButton, JTextField, which could be used in our program. For example, after we extend JFrame in the class, we create some instances of JLabel, JTextField, and JButton and add them in the JFrame with setting parameters for specific locations. After that, we needed to link this class with other classes, so we decided to implement the ActionLIstener interface. The reason we choose ActionListener is we want the program to detect when the user clicks the action listener button. After receiving the request, we used a method called “actionPerformed” to call the above request. The second major change is the logic of the game. In the first DEV presentation, our game logic is that when the game is over, users can only choose to exit the game. However, this is not the line with a normal game logic, so through the feedback of DEV, we decided to add a *game loop*. In the GmaeView class, we wrote a method called “showGameOoverMessage”. By using the java class JOptionPane’s method, which is an easy way to pop up a standard dialog box that prompts users for a value or informs them of something, like “Yes”, “No”, or “Cancel” options. In our case, we used “showConfirmDialog” to return an integer “1” or “0”. After that, through the returned value, we could implement the *game loop* by if/else statement. Based on the feedback of DEV, we also wanted to add a storage system, but due to lack of ime, we did not complete it. But our idea is to create a new file, which stores the score record of each time and returns the historical scores of total. We think this method is feasible, but it takes a little more time.

***Changes impact:***

First, adding a new login interface makes our application more complete and safer. When we were designing the application, we hoped that our game could be the same as other online games, which allows players to log in to their accounts and obtain their own game records. Although the second part has not been realized, it is not far from success. The second change makes the game more playable. The game loop is implemented by asking the player if they want to continue the game, which allows users to quickly restart a new round of games. These two major changes have made our application a huge improvement.

**Testing and iterative design:**

During the test, we used a lot of "system.out.println" in the code, and basing on the return result could be used for understanding how the program worked. For example, When we finished writing the "createFood" method and the "eatFood" method, we found that the snake could not eat food anyway. By adding lots of “print” methods, we found that the coordinates of Food and the coordinates of Snake’s head are different. Through the returned value, we knew that there was a problem where we set up the node. After  we modified the paramarter in the Node, the problem was solved.

**Restrictions, limitations, and constraints:**

The current problem is that every time a game ends and the player chooses to restart, the original login interface will appear for a moment then disappear before jumping to the game window. We showed it in the last part of the video of the final demo, but we can't directly see this problem from the video. In addition, there is another problem that a round of the game is over, if the player chooses to restart, the game window of the previous round will not disappear or close, which means that as the player restarts more rounds, the more windows open. We think that the reason for these two problems may be due to the logic of the game loop.

**Conclusion:**

We successfully made our game run with a login user interaction. Due to time reasons, we failed to successfully put some of our ideas (such as the scoreboard) into our actual finished product. The completion of the game itself is satisfactory. We may continue to update this app in the future to reach the level we initially expected.

**Work Citation**

Jbberinger. Snake, commit 4271e93b399c01baa9367da80383c220e16e691b, 2018. GitHub, <https://github.com/jbberinger/Snake/tree/4271e93b399c01baa9e67da80383c220e16e691b/src>.