Part 4: Final Report

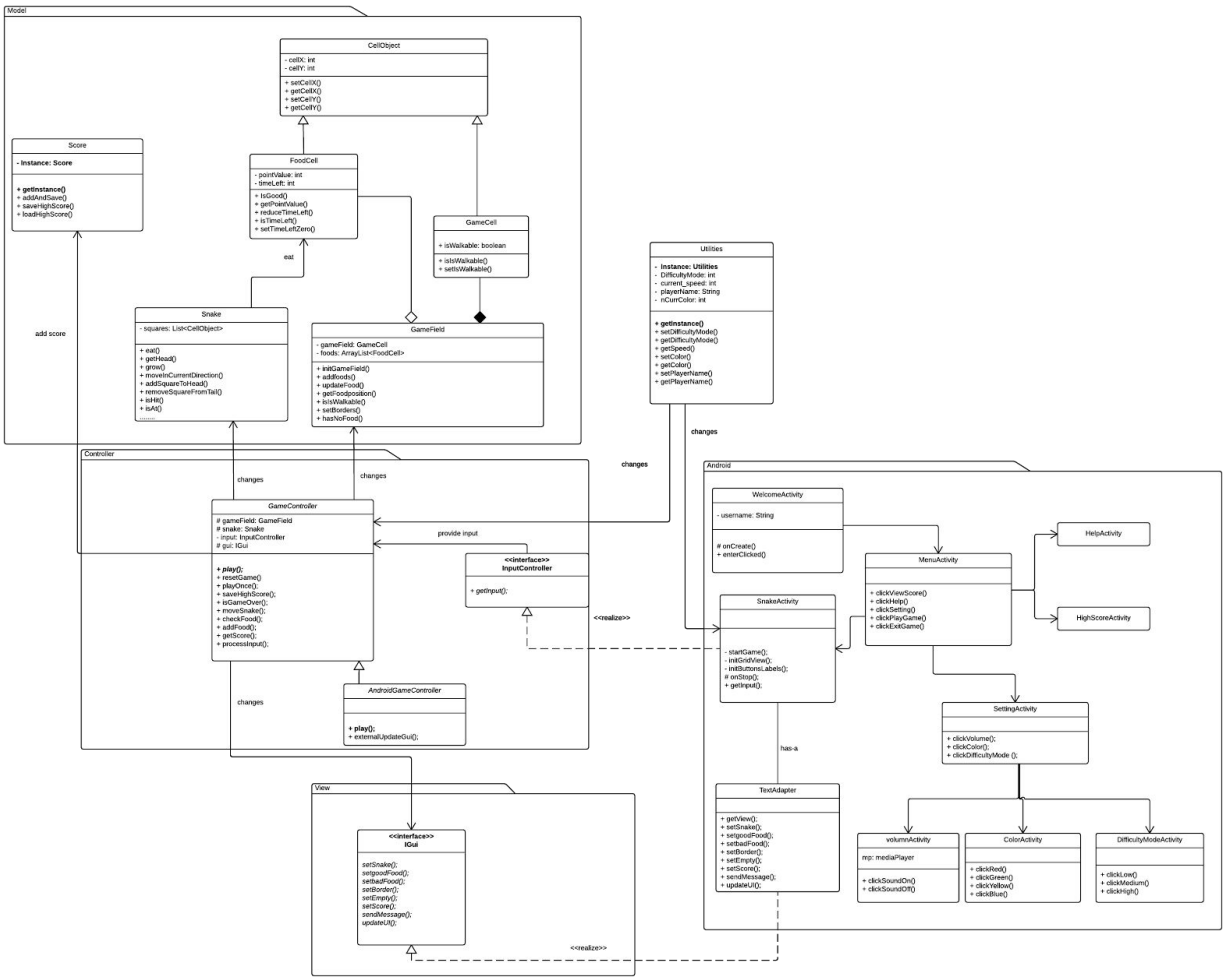
Yun Zhou
Tiantian Xie
Sorayya Niazi

Part 1: Final state of our project

We have implemented all the use cases listed in our project part 2. Here are the features we implemented:

1	The user can enter a username to play the game. No password is needed.
2	The user can read help instructions.
3	The user can modify the volume of the background music..
4	The user can modify the background color.
5	The user can modify the difficulty mode of the game.
6	The user can view the top ten scores.
7	The user can pause and resume the game.
8	The user can restart the game when game is over.
9	The user can exit the game whenever he or she wants.

Final Class Diagram



Benefits of designing before coding:

1. From the design phase, we formed an overall perspective of the whole project. UML diagrams help to illustrate clearly all the design we have. During our implementation, we almost follow our class diagram to write code.
2. In the design phase, we decided to use MVC framework, which separate the project in several different parts with high cohesion and low coupling. This structure design is extremely helpful for dividing the coding tasks to group members, which eventually improved our efficiency in a significant manner.
3. Another benefit of designing is that it facilitates future modification or/and adding features. For example, if we want to implement the snake game app on desktop, these

UML diagrams are great references, from which we know where we should do the modification and how could we do the modification to the code in a reasonable way.

4. Last but not the least! During design phase, we could think what design patterns will facilitate our implementation and take advantage of them appropriately. If we skip the design phase and write code directly, it will be inconvenient and troublesome for us to think design patterns, the relations between classes when we are focusing on the detail of coding.

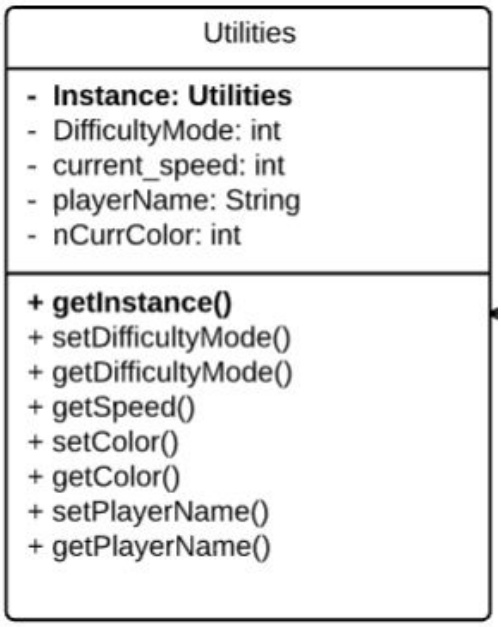
Part 2: Design Pattern

1. **Singleton**

We have used singleton design pattern in Utilities class and Score class.

Utilities:

The Utilities class is used to store global variables accessed frequently in the whole program which are set in Android activities. The variables include the username, difficulty mode, initial speed, background color and etc. We need and only need one instance of Utilities class. The "instance" is just an interface to get and set that global stuff. Every "instance" has the same ability, since it gets and sets the same variables -- and in fact, any update through one "instance" will be reflected in every other.

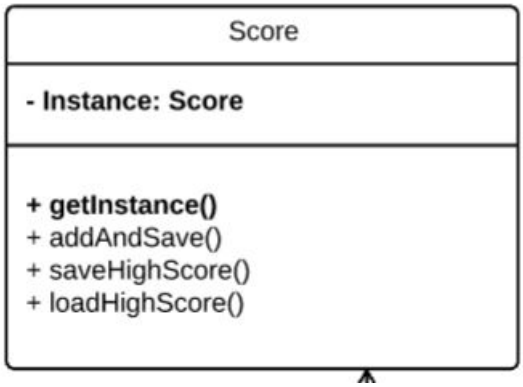



```
    private static Utilities instance = null;


    public static Utilities getInstance() {
        if (instance == null) { // first call
            instance = loadSettings();
            if (instance == null) { // could not load
settings
                instance = new Utilities();
            }
        }
        return instance;
    }
```

Score:

And we also use singleton in our Score class.

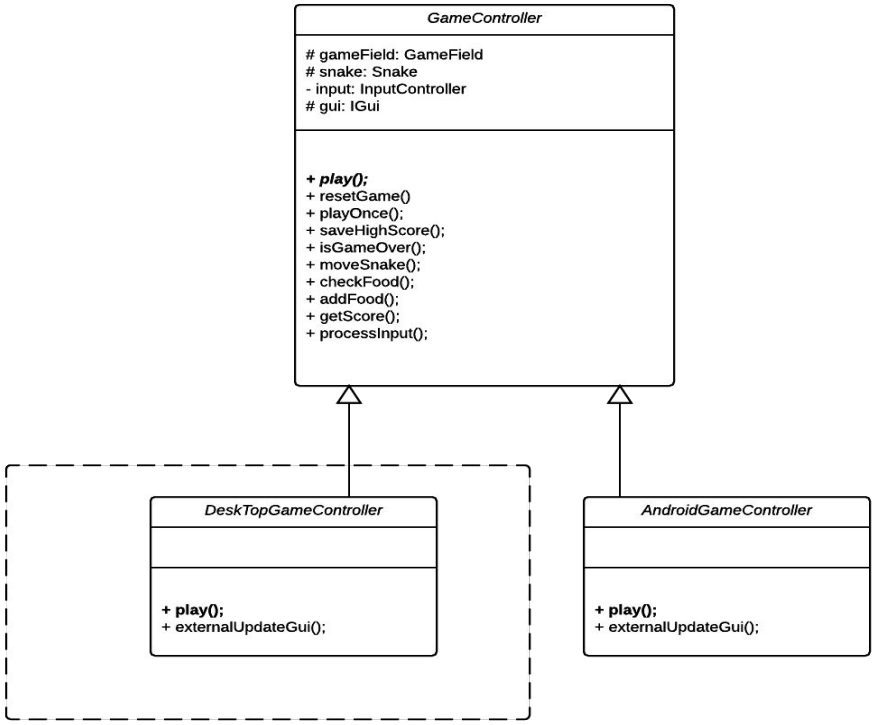



```
private static Score instance = null;


public static Score getInstance() {
   if (instance == null) { // first call
      instance = loadHighScore();
        if (instance == null) { // could not load
highscore
                instance = new Score();
        }
    }
    return instance;
}
```

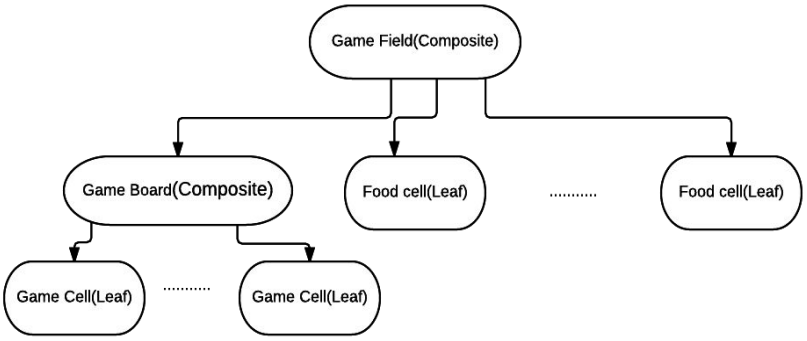
2. **Strategy**

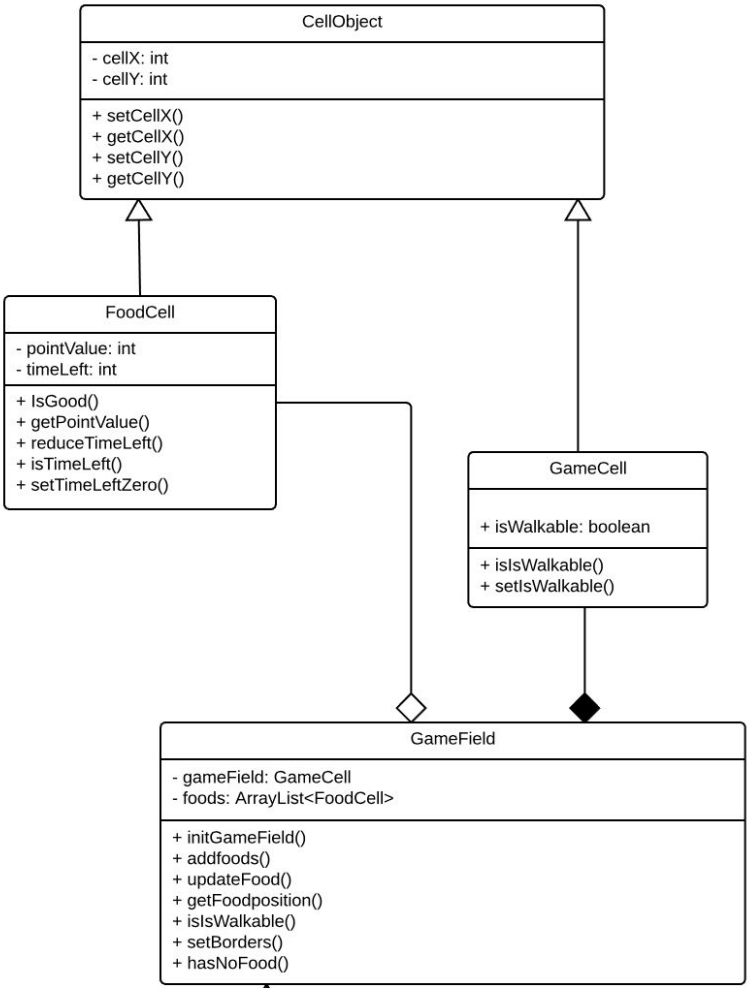
Strategy design pattern is used in the Controller part. In this project, we implement the greedy snake game on Android platform. But in the future, we want to further this project and implement this game on Desktop. In this way, strategy design pattern, which follow the “open for extension, closed for modification” principle, is used here for further extension to desktop version of this greedy snake game.



3. Composite

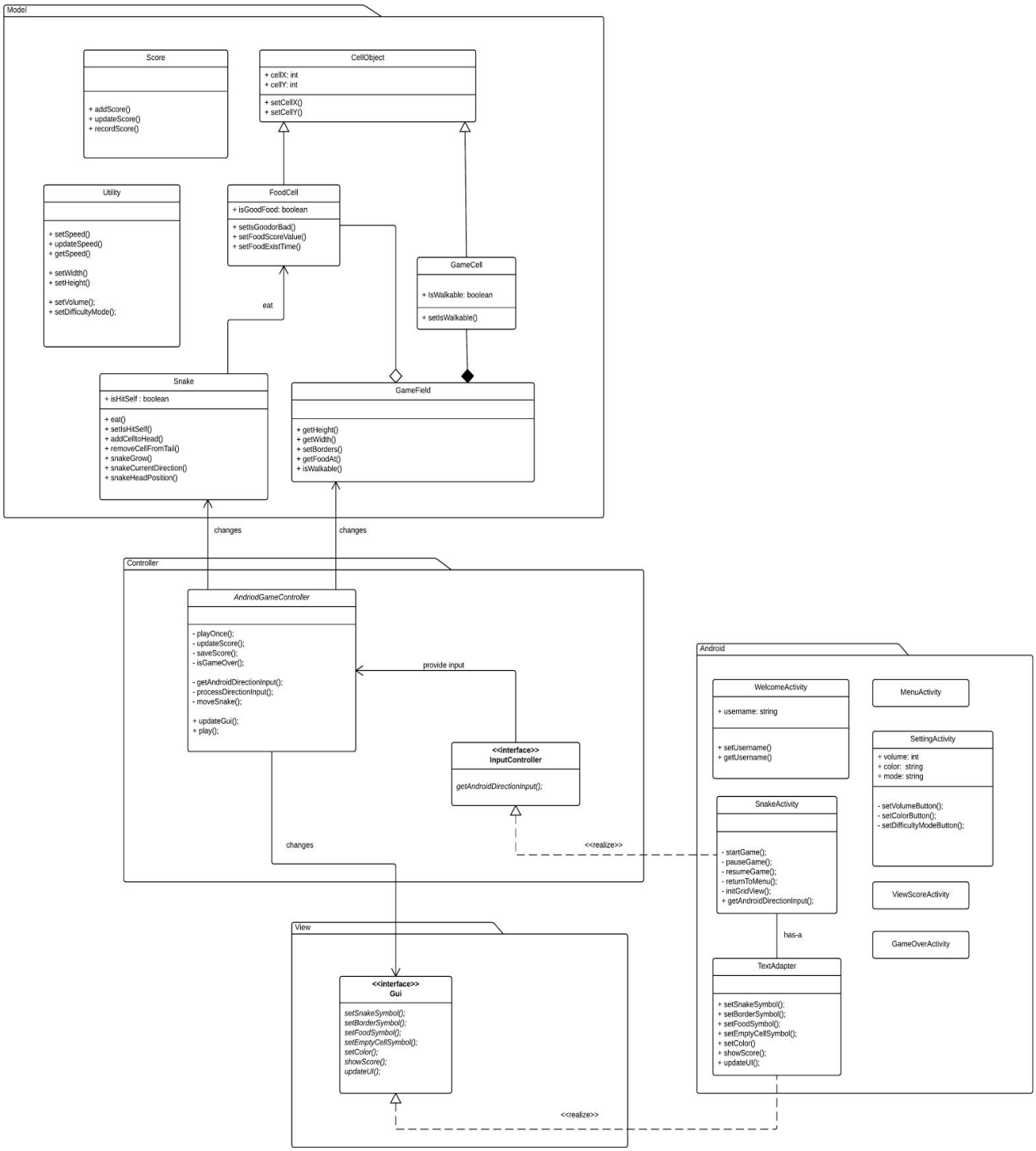
The composite design pattern is used in the model part in MVC. In the model part, the food cell and game cell are both cell objects. The game field include the gameboard which is consist of game cells and a list of food cells. So here we use composite design pattern of the game field. The tree structures are shown below.





Part 3: Comparison with previous design

Previous version Class Diagram



What changes in Class Diagram and Why:

There are three main changes in the Class Diagram:

1. In the Controller part, insteading of using solely a concrete AndroidGameController class to implement all functionalities, we use an abstract GameController class, and then let the AndroidGameController class inherit the GameController and implement the play() method. The reason we did this change because this could be more flexible for extension if we want to add a desktop version of this game--we could simply create another subclass of the GameController named DeskGameController and then implement the play() or some other methods in a different way.
2. In the Android part, the relations between different classes are clearer and more reasonable. In the previous version, we did not show the relations between different "Activity" classes(eg. relation between MenuActivity class and SettingActivity class) . In the final version, we think it is better to show these relations.
3. Utilities class have more reasonable relations with other classes. The reason we did this change because the flaw was pointed in the grading interview.

Part 4: Lesson learned from this project

1. Design phase is very important. Software development is not just about writing code. Good design are of great significance for a good software product. From this project, we have obtained more comprehensive understanding of UML.
2. Deep understanding and good use of design patterns will positively influence the implementation in a significant manner. We applied 5 design patterns in this project, which makes things easier in implementation phase and makes our code structure much better. From this project, we have obtained a deeper understanding of design pattern--when to use them, how to implement them and what benefits they will impose to your coding.
3. Good team dynamics is critical, without which we can not deliver any valuable results. Our group met some problem when we were doing project part 2. Then we fixed it and collaborate with higher efficiency afterwards. We found that deep and sincere communication is a good way to eliminate misunderstanding. Software development is always a group work and we have to improve our people skills continually in order to work as a valuable person in the future working environment.