Team Jörmungandr

• Introduction

Our project prototype is a classic snake game but we use the knowledge we learnt in this course, which is object oriented design and MVC in javafx. The gui is written in javafx and the other classes are all normal java class. It has all the functions in the user story that is completed. But the feature that makes the game so different is we have a hidden easter egg.

• User stories

As a player, I want to change the speed of snake movement.

As a player, I want to have time limit to get next food. (unimplemented)

As a player, I want to have obstacles. (unimplemented)

As a player, I want to be able to change the map size.

As a player, I want to have some wind in the background so that the game is more difficult. (unimplemented)

As a player, I want to have a score so that I can know how well I played.

As a player, I want to compare my score with other players so that I can feel better if I have a high score

As a player, I want to change the head of the snake so that I can have some fun (unimplemented)

As a player, I want to have a light background music so that I can focus on the game (unimplemented)

As a player, I want to have a variation of the snake colors so that I can have some visual difference. (the snake color is different from the food and grid color, but currently no option to change the color of the snake itself)

As a player, I want to play with others so that I can have fun with friends (unimplemented)

As a user, I want to have a clear interface so that I can know how to play the game.

As a player, I want to have the snake going without any obvious delay so that I won’t lose the game due to the game itself.

•OOD

We have 2 packages called Final Project, and gui.

The final project package contains the Snake class, the Point class, the Leaderboard class and SaveLeaderboard class and the Player class. The gui package contains the MVC structure and the GameGrid class.

Here are the classes in the final project package.

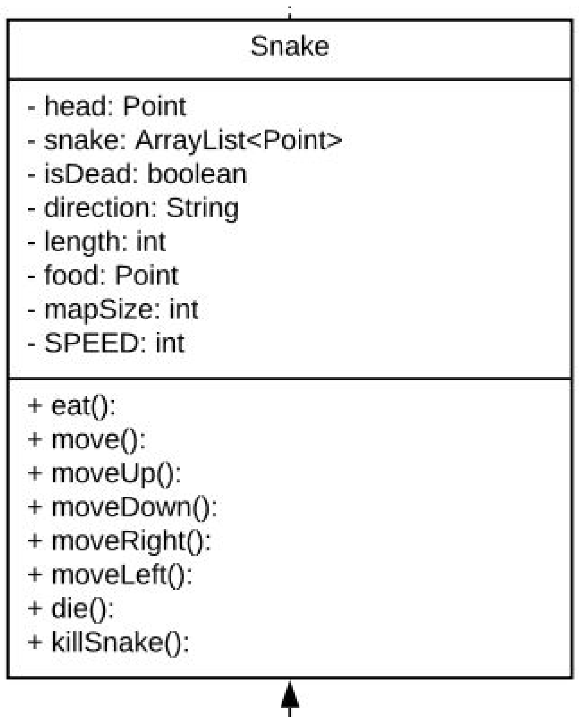
The Point class is the basic class that represent a single pixel of the map. And here is the CRC card for Point class.

Point	
Equals	
Set coordinate	
Get coordinate	

Since the Point class is the foundation of everything else, it does not interact with any other class. The Point class itself has x coordinates and y coordinates as parameters, the getter method that can get the coordinates, and an overridden equal method that can tell whether two point is pointing at the same pixel.

This class is so basic that you can not only use it in our snake game, you can use this class in any other programs that might use the idea of a pixel.

The Snake class is where we code the behavior of the snake.It has the function of let the snake eat, grow, move, and tell whether the snake is dead or not. The Snake class uses the Point class (where the arrow comes from), and is imported by the SnakeTask class (which the dotted line goes to).



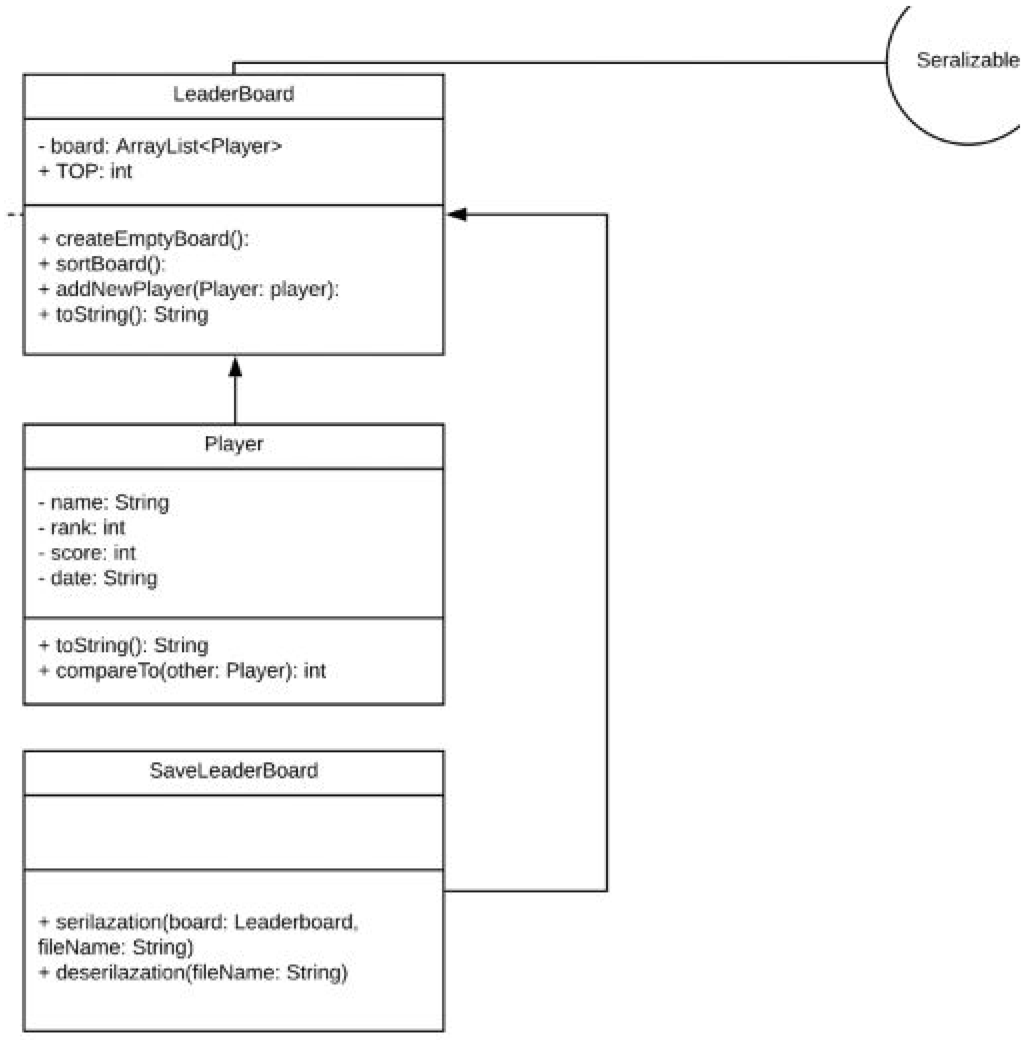
The snake uses an ArrayList of points to represent the snake and a specific point head. The fields also contains a string representing the direction, a boolean tells the status of the snake(live or died) with default value false, integers represent the length and the speed of the snake, and the size of map. And a point stores the food position.

This class has the methods that allow users to move the snake, the length of the snake is part of the score, which is also what the user wants. We tried our best to write the methods as independent as possible, so that the processing of the computer is simple and the game runs smoothly. The speed in the field of this class is changeable so that the user can have more level of difficulties. The die methods ensures the snake dies properly so the game ends.

The Player class is a class that create a player object. The class has fields of name, rank, score and date.

The Player class has only two functions, which is create a new player and compare two players so that we can sort players in the leaderboard.

The Leaderboard class creates the leaderboard we used in the game. This class implements serializable so that you don’t lose your leaderboard after you quit the game. The player class has an ArrayList of players in its field, and also a final integer named TOP to store the number of players in the leaderboard. The diagram below is the group of classes that provide the leaderboard function



As for methods, this class has 4 methods. The createEmptyBoard method is called in the constructor when there was no other leaderboard stored in this computer. The sortBoard method is used to sort the players in the leaderboard. The addNewPlayer method takes in a player object as parameter and add this player to the leaderboard, this method also calls sortBoard inside it. The last method is toString method, it returns a string of players, each player occupies a line and this string is what the leaderboard gui shows.

The leaderboard class system, which also include the player class and the saveLeaderBoard class, is independent from the snake, so this class can be extract from this game and use as a leaderboard in other games. And it also let the user have a leaderboard so that they can store their scores and compare their scores with other players.

The SaveLeaderBoard class is just two methods that serialize and deserialize the leaderboard object, with two methods to do each of them independently.

The classes introduced below are all in the gui package.

We used the MVC method to implement the snake game. The GameMain class uses the controller, view and the model, as shown by the graphs below.

The GameModel class contains the logic we used in the game, including the Snake class and the Leaderboard class.

GameModel	
Get snake Get leaderboard Refresh model	Snake Leadaerboard

The GameModel class simply import those two classes, and add a refreshModel method so that every time you replay the game you get a new snake instead of the one from the last game.

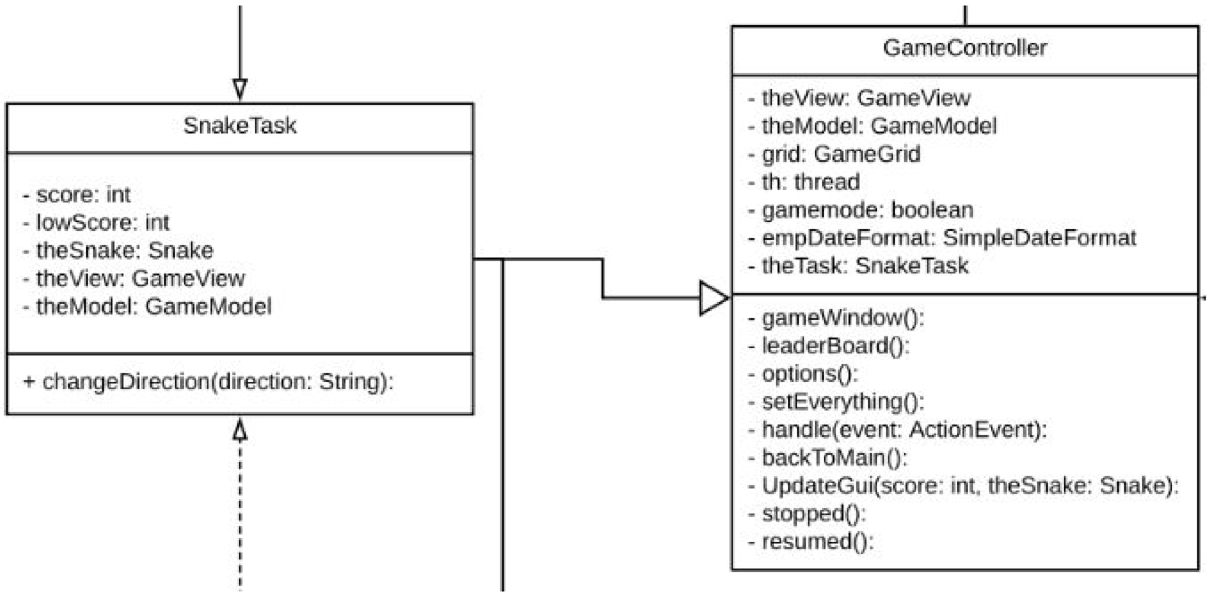
The GameView class is where we put all the visible things in the game.

GameView	
Set game GUI	GameModel

In the field we have the GameModel, a group of buttons including the play button, the options button, the leaderboard button and the back button. In addition, we also have a VBox rootnode.

According to the way of MVC programming, we don’t have any methods here except for getters.

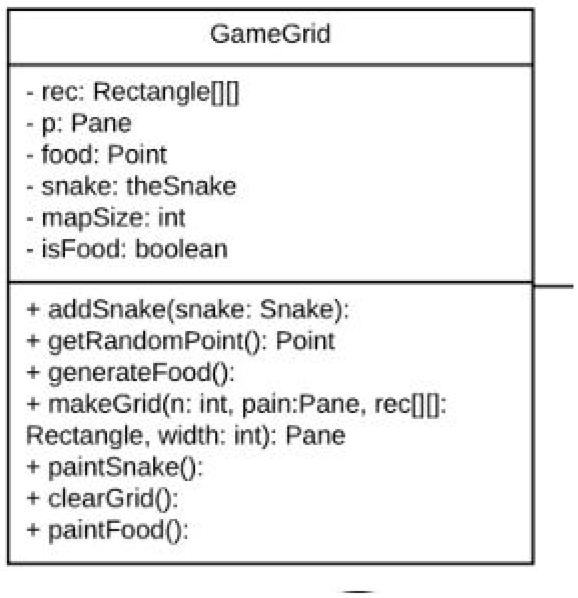
The GameController class is where we handle the user input. In its field it has the GameView, GameModel, GameGrid, a thread, a task, and .etc. We have a list of methods that deal with the event of clicking the corresponding button instead of putting everything in the handle method. In the option method and setEverything method, we have the code that let the user set the map size and speed, which match the requirement of changing size and speed. The gameWindow method is where we setup the gui for the actual game. The stopped and resumed methods handles the stop function which the client want.



In the GameController class, we also have a SnakeTask class that create a task object. That is the main game logic. This class have a run method, and it keeps running while the snake is alive, and update the gui in every while loop use the updateGui method. This SnakeTask is the key that make the game run normally, and make the users can have a complete game after he or she click the play button.

The updateGui method reprint the GameGrid using the new snake and the food position.

The GameGrid class create a GameGrid object, which is a 2d array of rectangles. In the field it also has a snake object, a pane, a point food, an integer mapSize and a boolean isFood. Here is the UML diagram for GameGrid, it is related with the GameView class.



In this class we have a getRandomPoint method, that can be used to generate food or initialize the position of the snake. The addSnake method add the snake in the grid, the tow paint method paint the food and the snake, and the cleanGrid method is used to clear the screen and it is used when we run the updateGui method. The mapSize in the field of the class is changeable, that match the user story of changing map size.

At last, we put an easter egg in the game, which is also a part of client requirement, but the easter egg is not in any of our word documents. Because the charm of an easter egg in the game is the process of finding it. It would be boring if someone tells you where the easter egg is hidden. The only hint for the easter egg is our team name Jörmungandr, the answer is hidden in the myth.