OOP Final Project Report

Topic

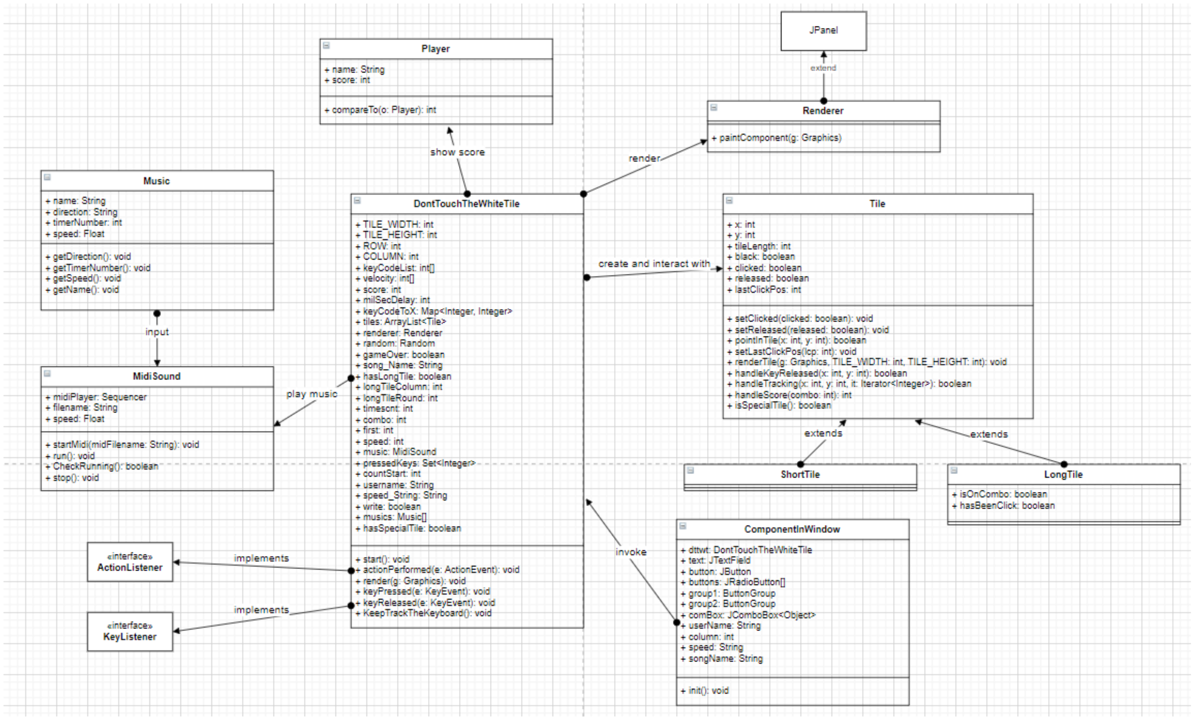
Don't touch the white tile.

組員分工表

組員	學號	分工
張鈞堯	b07902022	start page and key event handling
林則仰	b07902006	tile creation and rendering
王泳錡	b07902029	music part and scoreboard

Software Design

Design Diagram



If the image is not clear, please refer to this url. https://drive.google.com/file/d/1uKH6YFBHB6ax4PQry27pzmgLWPRV7AAH/view?usp=sharing

Game Flow

[In class DontTouchTheWhiteTile]

First it will invoke class `ComponentInWindow` to get the needed parameters.

[In class ComponentInWindow]

This class will render a screen that provides several options for the user to select. Once the **Play** button is clicked, this screen will be disposed and a new game will be initialized by the paramters.

[In class DontTouchTheWhiteTile]

The game start the music user selected first.

[In class MidiSound]

First it read the music direction and open it. Then setting the sequence pointer to the head of the song file. Then adjust the song speed. Finally, start playing the music.

[In class DontTouchTheWhiteTile]

Next, we will maintain an arraylist `tiles` of Tile object to represent the tiles in game. In the beginning, there will be $C \times (H+1)$ short tiles in this arraylist and both of them are white. At the first 3 seconds, the game will countdown three seconds to remind user that game is going to start. After that, in every 0.02s, the game will call funciton `ActionPerformed` since this class implements class `ActionListener`. We use a variable `timescnt` to record the time in game.

Everytime the keys with keycode we specified in game (eg: 70, 32, 74, 82) detected keyPress, we will add the keycode into a HashSet `pressedKeys`. Meanwhile, everytime funciton `ActionPerformed` was called, do`timescnt++`. In `ActionPerformed`, we also subtract a constant from every tiles' value of `y` to preform the effect that tiles slide. After subtracting, if there are tiles whose value of `y` is greater then `ROWS * TILE_HEIGHT`, we will remove it from the arraylist. In this condition, we will produce new tiles and add them into arraylist `tiles`.

In addition, we will check all keycodes in `pressedKey` mentioned above everytime `ActionPerformed` is called and set all of their relative tile to`tile.clicked=true`. If their relative tile is short tile, set `tile.released=true` since a short tile can't be pressed more than once. Next, we add the `score` by the type of tile and the current `combo` (If the tile player pressed is white, `combo` will become 0, too). Furthermore, everytime the keys with keycode we specified in game was released, the game will check if its relative tile is `tile.released`. If so, that means this tile is a short tile since we've set it to `true` when it was pressed. Otherwise, it is a long tile and we will remove it from `pressedKey`. By this, we can make sure that the long tile that is pressed but hasn't released should keep adding `score` every 0.02s.

After the modifications of tiles in `ActionPerformed`, the game will rerender so we can see the tiles' displacement and the new tiles.

After 60 seconds since game starts, the game will end and screen will show a scoreboard. Players need to press `R` (keyCode : 82) to restart the game, and `timescnt` and `score` will set to 0.

The advantages of our design

1. We've set several rules to prevent some illegal tile patterns.
2. We've done the synchronization mechanism to handle multiple keypress at the same time.
3. We've done OCP on different kinds of tiles.

The disadvantages of your design

1. Since the package `javax.sound.midi` only support paino music, the song we can use is limited.

2. After considering several issues, we decide to use a file to store our score instead of a DB. Thus, if one wants to move the program (perhaps for deployment), he also needs to move the score file, too.

Other packages that we have used

```
// handle action and key event
java.awt.event.ActionEvent;
java.awt.event.ActionListener;
java.awt.event.KeyEvent;
java.awt.event.KeyListener;

// render the frame
java.awt.Graphics;
javax.swing.JPanel;

// create and render the window
javax.swing.JFrame;
javax.swing.Timer;

// handle the music
javax.sound.midi.*

// construct the starting window
javax.swing.*
```

How to play our 2D game

First, compile and run the program by typing

```
make run
```

Or typing the following commands:

```
javac -cp . -sourcepath src -d ./out/ ./src/*.java
java -cp out/ dontTouchTheWhiteTile/DontTouchTheWhiteTile
```

Once the program is running, you will see the starting window. You can adjust your settings here. After that, you can click the play button to play. If you don't type anything the default will be

```
username: player
speed: slow
Columns: 3
music: Canon
```

For each column from left to right, the keyboard setting is like the following section.

```
Column = 3: F [space] J
Column = 4: D F J K
```

Once you enter the game, you'll see the grids and a red line. This red line represent the vertical position of where you keypress. The target for you is to keypress all the black tiles as many as you can. For the short tile, you just need to keypress it once. For the long tile, you should keypress it until it reaches the end. If you keypress the white tile, your combo will be 0 and you score will be deducted.

The duration of the game is one minute. When the game is finished, there will be a scoreboard showing the top 5 score of the same setting. If you want to play again you can press key `R` to restart the game.