Fundamental Object Oriented Programming (NTU, Spring 2021) - Final Project Report

周寬	周健烽	胡皓雍	鄭明淵
B06902085	B08902087	B08902110	B08902114

2021/07/01

1 Introduction

This report is about some detail of our game **Shaaark**. Shaaark is a 4 key Rhythm game. Theme of the game is based on the famous Virtual YouTuber Gawr Gura. In our game, we include some songs which are created by Gawr Gura. The main selling point of our game is of course, the cuteeee Gawr Gura!

2 Our Design

In our design, we can divide it to 7 part about folder `src/`.

- `FileHandler/`: It is a folder that contain some classes for the management of files.
- `controller/`: It is a folder that contain some classes for controlling the overall process of the game.
- `media/`: It is a folder that contain some classes for management of media (e.g. playing music).
- `menu/`: It is a folder that contain some classes for the menu page in the game.
- `model/`: It is a folder that contain some classes for the model such as sprite and screen etc.
- `note/`: It is a folder that contain some classes for the note split in the game.
- `track/`: It is a folder that contain some classes for track.
- `utils/`: It is a folder that contain some classes for some utilities the game use. (e.g. image state)

- `views/`: It is a folder that contain some classes for manage the view of game.

Next, the following is about the relation between the classes and the responsibility of the classes and interfaces.

- For classes in folder `FileHandler/`:
  - `class FileHandler`: This class responsible for management of files which use in the game. (In this game, this class usually use for management of beatmap file)
- For classes in folder `audio/`:
  - `class AudioPlayer`: This class is responsible for management of audio which use in the game. (In this game, this class usually use for management of song and effect music)
- For classes in folder `menu`:
  - `class Intro`: This class is responsible for showing the flickering text "Press any key to continue" in the title screen.
  - `IntroImageRenderer implements ImageRenderer`: The flickering effect is in fact a 2 state looping sequence. This class implements `ImageRender` that we need in `CyclicSequence`.
- For classes in folder `note`:
  - class Note: This class interit abstract class Sprite, responsible for management note in the track, which manage style and the location of note. When the note overborder (do not remove by press key), the note will be removed by this class.
  - class NoteDatabase: This class interit class Thread, responsible for management all notes in the beatmap.
- For classes in folder `model`:
  - `abstract class Sprite`: This abstract class define some abstract method for the elements which will show in the frame. (In this game, track and note etc. inherit this class)
  - `class SpriteShape`: This class responsible for recording the size and offset for the sprite.
  - `class Screen`: This class responsible for recording and management of sprites in the screen.
- For classes in folder `fsm/` (this class created by TA):
  - `interface State`: The basic building block of a finite state machine.

  - `interface StateMachine extends State`: Provides finer specs of methods such as trigger for the state to be useful in a finite state machine.
  - `class ImageState implements State`: A special State that represent an Image.
  - `interface ImageRenderer`: An interface that is needed to instantiate an `ImageState`.
  - `FiniteStateMachine implements StateMachine` A fully functional finite state machine that is defined by a list of states and a set of transition rules.

- For classes in folder `utils/` (this class created by TA):
  - `class ImageStateUtils`: This class responsible for the image processing for the status.

- For classes in folder `track/`:
  - `class track`: This class inherit `abstract class Sprite`, responsible for the controlling the every event trigger in the track such as the note hit and the click effect.
  - `class Border`: This class inherit `abstract class Sprite`, responsible for management of border.
  - `class TrackButton`: This class inherit `abstract class Sprite`, responsible for button in the track.
  - `class ClickEffect`: This class inherit `class CyclicSequence`, responsible for the detail of the click effect.
  - `class NoEffect`: This class inherit `class CyclicSequence`, responsible for the detail of the no click effect.
  - `class TrackImageRenderer` This class inherit `interface ImageRenderer`, responsible for rendering image for track.

- For classes in folder `views/`
  - `class GameView`: This class responsible for the overall view in game such as menu switching, key trigger etc.

- For classes in folder `controller/`
  - `abstract class GameLoop`: This abstract class provide the base game looping (update $\rightarrow$ render $\rightarrow$ loop it forever) and the base template for a game.
  - `class Game`: This class inherit `abstract class GameLoop`, responsible for running the our game Shaaark which contain menu, song menu and playing screen that include 4 tracks and some notes base on the sheet music. Also, the remove note by press key will working in this class.

- `class Main` This class responsible for the creating the Game and the GameView to start the game Shaaak.

And then, some implementation detail for the beatmap file. In file `assert/song/reflect/sheet.out`, this file content like following.

```
125
0000
0000
0000
...
...
```

In the above content, first line contain a number which is the BPM of the beatmap. After first line, the each lines is the whether there are note for each beat in the track. Zero for no note, one for having note. For the time interval of each beat, we can use following formula.

$$\text{time interval of each beat} = \frac{60000}{\text{BPM} \times 4} \text{ ms}$$

We believe that is a OPC of the beatmap.

3 Advantages of our design

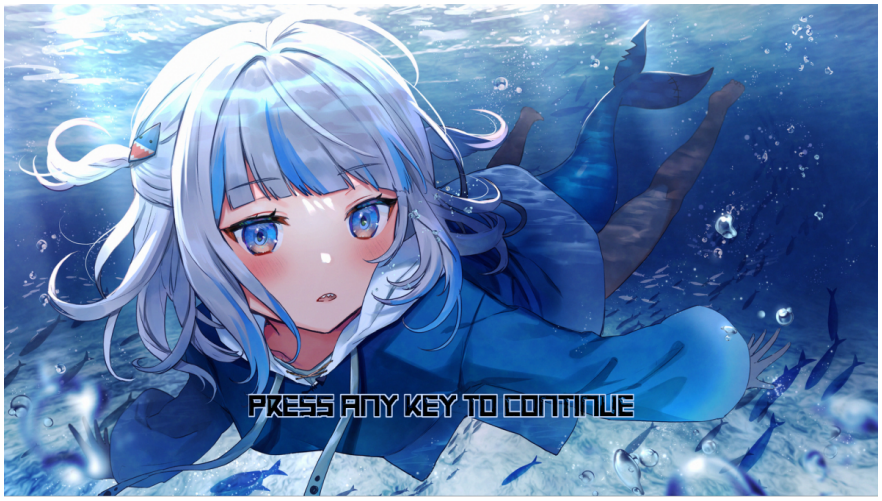
- As a fan game, our game contains dozens of interesting easter eggs which make our game stands out from ordinary rhythm games.
- Tried our best to fulfill the requirements of the Open-Closed Principle.
- We can create customized beatmap base on our beatmap format.

4 Disadvantages of our design

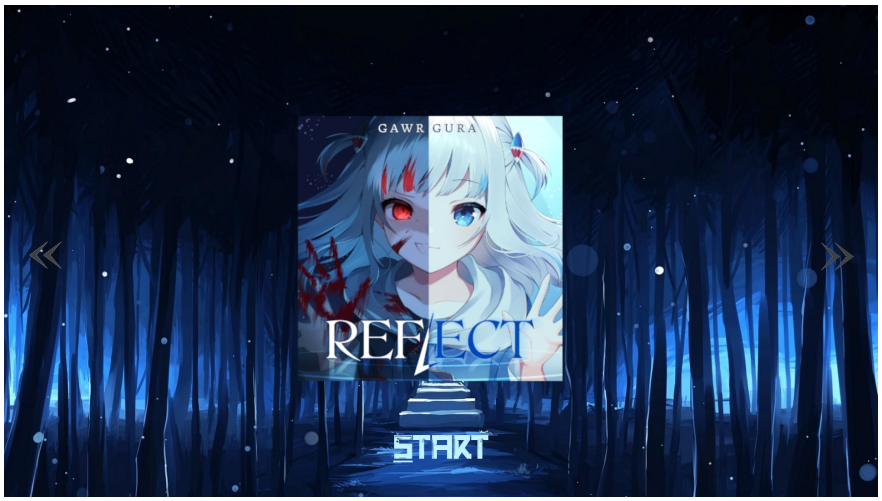
- Few settings that can be customized by the players. (e.g. notes drop speed, offset adjustment)
- Lack of some common types of notes such as sliding and holding.
- Add new Song is not easy for our design.

5 How to play our 2D game

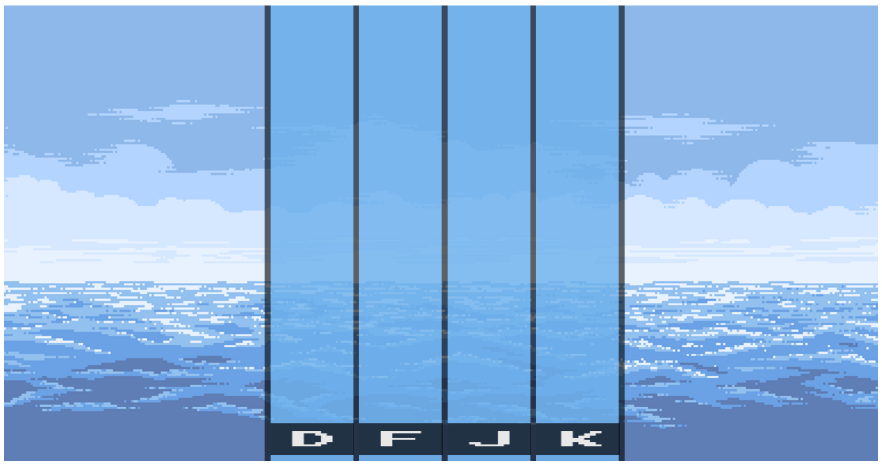
You can use command `make run` to start our game. You will see the start menu of our game as following screenshot.



You can press Enter to enter the song select menu as following screenshot.



In the song select menu you can press left key or right key to switch to the song you want to play. After selected, you press Enter to start to song.



In above screenshot, there are 4 tracks. D key, F, key, J Key and K key corresponds to each track from left to right. Notes will drop from top of tracks. The player needs to press the corresponding key when the Note falls on the black square in the track.

6 Responsibilities Assignment of Team Members

- For the coding part of our game, 周寬, 周健烽 and 鄭明淵 3 people finish it.
- For the art design part, 胡皓雍 finish it.
- For report, all members finish it.