Fundamental Object Oriented Programming 2021
Final Project

Make It On Time

李宜璟	陳冠廷	陳倢堂	鄭仲語
B07401012	B07902025	B07902018	B07508005

Summer 2021

1 Assignment of responsibilities

1. 陳倢堂: Menu Design, Map Design, Game Coding, Report
2. 陳冠廷: Structure Design, Map Design, Game Coding, Report
3. 鄭仲語: Game Mechanic Design, Map Design, Game Coding, Report
4. 李宜璟: Game Mechanic Design, Structure Design, Art Design, Map Design, Game Coding, Report

Special thanks to 葉冠廷 for his help on artworks in the game.

2 Object Oriented Design

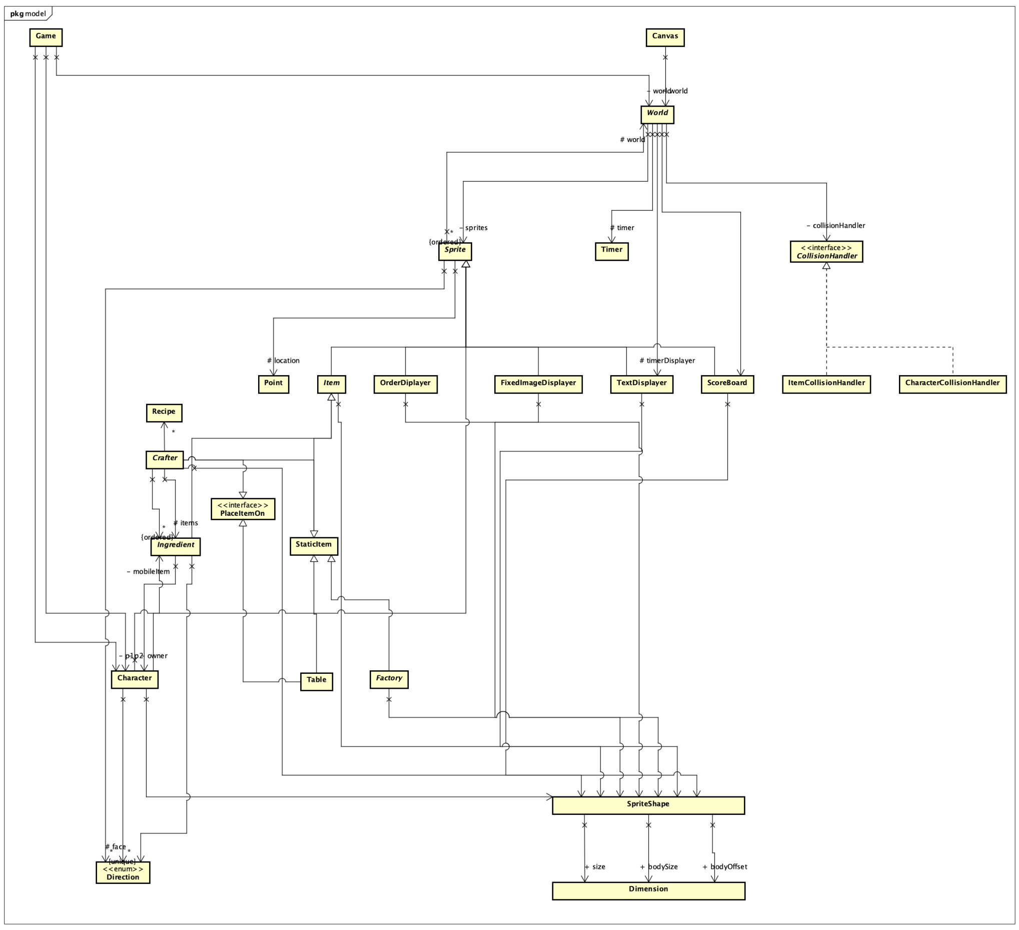
In this section we'll provide a brief illustration for OOD in our final project **Make It On Time**. Although we won't go through the detailed design and responsibility for each class, we do provide a simplified version of class diagrams, which demonstrate the relations between our classes.
There are three major division of classes in our design:

1. The **model** of the world, which includes all sorts of objects inside the game world and encapsulates their relations.
2. The **controller**, which is the interface between player input and the game world.
3. The **View**, which is responsible for displaying the world to the player.

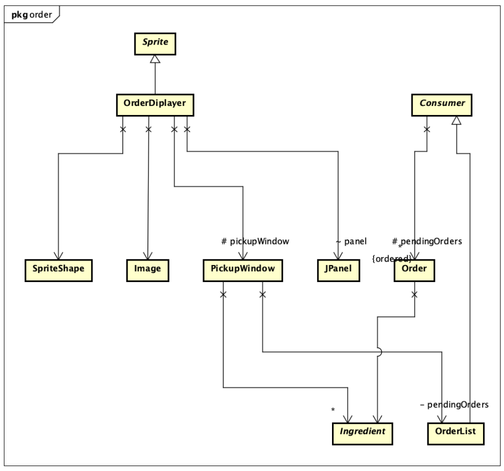
2.1 Model

The model of our game world is implemented in the class file **World**, which contains the following attributes. as shown in *Figure 1-(a).*

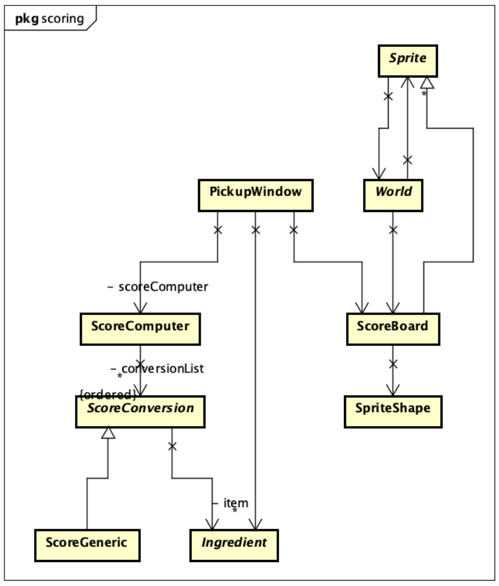
1. *Sprite*, a collection of the **Sprite** residing in our game world, they can further be classified into several classes



(a) A simplified class diagram of **World**.



(b) A simplified class diagram of **OrderList**.



(c) A simplified class diagram of **Scoreboard**.

Figure 1: Class diagrams of the game model in this project.

(a) **Character**, which can be controlled by players and is able to interact with other items, e.g. picking and releasing **MobileItem**, and collision with **StaticItem**.

(b) **Item**, which includes all sprites other than player in our game, divided into

i. **MobileItem**, or equivalently **Ingredient** in our design, which can be picked up and moved with **Character**. Also they can interact with **StaticItem** to be discarded and crafted into a new **Ingredient**.

ii. **StaticItem**, which isn't mobile but may be equipped with different function, allowing them to interact with **MobileItem** and **Character**. (See the following subsection for details)

2. **OrderDisplayer** is a special sprite, which shows the incoming order, requiring character to deliver them.

3. **Scoreboard** is also a special sprite, demonstrating the score player had gotten via controlling **Character** to complete the orders.
   The simplified class diagram of **OrderDisplayer** and **Scoreboard** are shown in *Figure 1-(b), (c)*, and worth noting is that they can interact with **Ingredient** (or completed order the player make) via a static item **PickUpWindow** is our design.

4. **Timer**, which counts down the game time.

5. **TextDisplayer** and **FixedImageDisplayer**. As the name suggested, these items can display text and images in the **World** as background.

6. **CollisionHandler**, which is responsible to handle the overlap the rigid body between sprites.

2.1.1 Detail about Item Relations

Several type of static items are implemented with some properties allowing them to interact with mobile item, or **Ingredient**. These includes,

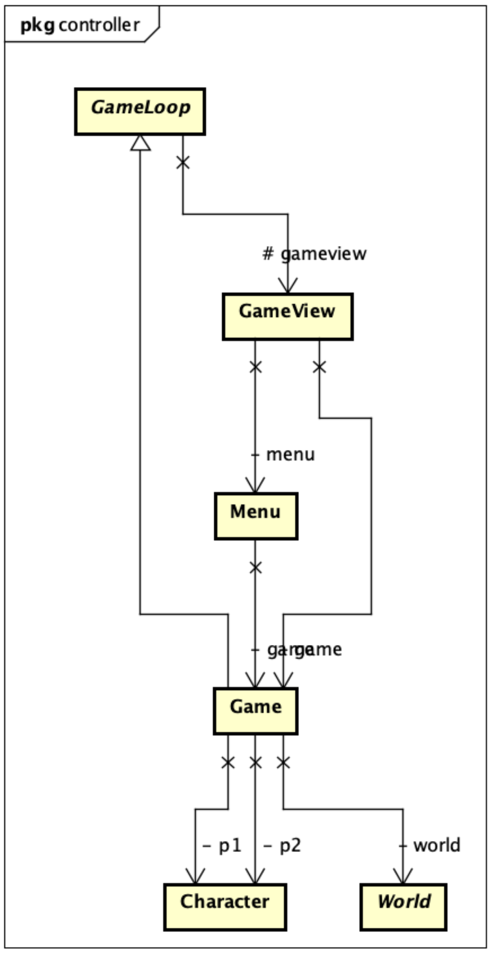
1. **Factory**, an abstract class, which encapsulate the function of limitlessly produce ingredient.
2. **Crafter**, an abstract class, which encapsulate the function of transforming ingredient(s) into new ingredient. Inside the **Crafter** are several **Recipe** attributes enclosing the transform formula.
3. **PlaceItemOn**, an interface which allows ingredient(s) to be released on this item.

2.2 Controller

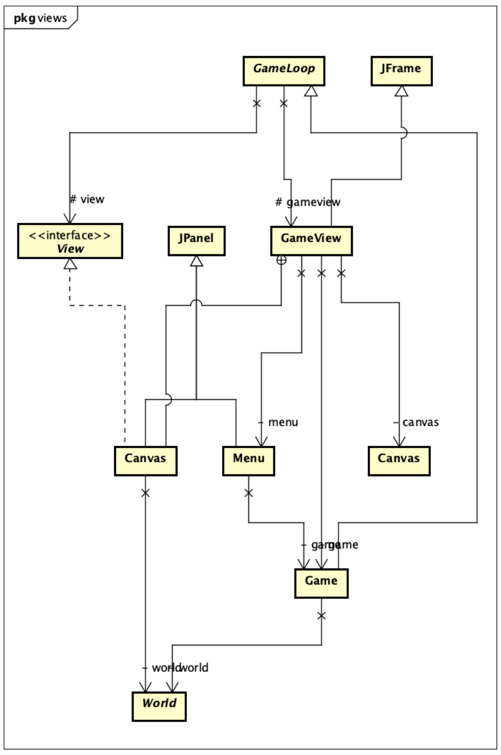
As shown in *Figure 2-(a)*, class **Game** implementing **GameLoop** operates the game flow, and **GameView** allows player to interact with **Game** and **Menu**, which in terms alters the **Character** and **World**.

2.3 View

As shown in *Figure 2-(b)*, class **GameView** utilizes **Canvas** to render the world and the action of the character in it.



(a) A simplified class diagram of game controller in this project.



(b) A simplified class diagram of game view in the project.

Figure 2: Class diagrams of the game control and view in this project.

3 Advantage of Design

1. **Open-Closed Principle**

   We achieve OCP in **Ingredient**, **Recipe**, **Factory**, **StaticItem**, **World**. One can add any ingredients by extending the **Ingredient** class as

```java
public class IngredientOneWants extends Ingredient {
    public IngredientOneWants(Point location, SpriteShape shape) {
        super(location, shape, " IngredientOneWants ");
    }
}
```

   One can creates any recipe by extending the abstract **ConcreteRecipe** class

```java
public class SomeIngredientRecipeOneWants extends ConcreteRecipe {
    public ApplepieRecipe(SpriteShape productShape) {
        super(productShape, "ingredientNeeded1", "ingredientNeeded2",
            ... );
    }
    protected Ingredient getResult() {
        return new SomeIngredientOneWants(new Point(0, 0),
            productShape);
    }
}
```

One can creates any ingredient factory by extending the abstract **Factory** class

```java
public class IngredientFactoryOneWants extends Factory {

    public IngredientFactoryOneWants(Point location, SpriteShape
        shape, SpriteShape productShape) {
        super(location, shape, productShape, "eggbasket");
    }

    @Override
    public MobileItem produceItem() {
        RawEgg newItem = new IngredientOneWants(new Point(0, 0),
            productShape);
        this.world.addSprite(newItem);
        return newItem;
    }
}
```

One can creates any static item by extending the abstract **StaticItem** class public class

```java
public class StaticItemOneWants extends StaticItem {

    public StaticItemOneWants(Point location, SpriteShape shape) {
        super(location, shape);
        ImageRenderer imageRenderer = new ItemImageRenderer(this);
        idle = new WaitingPerFrame(4, new
            Idle(imageStatesFromFolder("assets/item/staticItemOneWants",
            imageRenderer)));
    }
}
```

Finally, one can also creates a customized map and world by extending the abstract **World** class.

2. Not many outer source code and packages are utilized, make our codes rather ”lightweight”.

4 Disadvantage of Design

4.1 File IO

Beacuse we use java.IO.File to access our assets, it is nearly impoosible to package the whole game as a file. A proper way to load image from .jar file is to use getClassLoader().getResourceAsStream(). However, since the utility is design to load state by all file name, it is not likely possible to do so.

4.2 Design limit

We us java AWT as our GUI engine, and because it is quite old package, some of our design is limited by its ability.

5 Packages Utilization

No outer source package other than java AWT is imported in this project. But we can have to attribute and thank TA Waterball for providing the template code for 2D game design in package **Java-Game-Programming-with-FSM-and-MVC**, especially for the design of **FiniteStateMachine** packge **fsm**, and all parts related to image rendering.
All picture and icons designs are download from free database **flaticon**, with attribution to creator **Freepik**.

6 How to Play

After entering the game menu, the player can determine the number of players (1-2) and which game world to play (No.1 -No.5). Then, the player can click the start button to start the game.

During the game, one can see that the character(s) and the game map are in the middle of the window. On the right side of the window, there are a timer, a scoreboard, and recipes. The food orders are shown at the bottom of the window, the player can finish the order and get a score by following the recipe to cook the food and deliver it to the pickup window. If the player delivers one food that is not in the orders, he/she would get score punishment by deducting score by 10. The orders would increase during the time and the maximum number of orders is 5.

Both players are controlled by the keyboard.

- Player1: Move **W/A/S/D**. Pickup food: **Q**. Place food: **E**
- Player2: Move **I/J/K/L**. Pickup food: **U**. Place food: **O**

In the game, the player(s) can get food ingredients from 7 items. There are **EggBasket**, **BreadBasket**, **CheeseBlock**, **SpinachGarden**, **PieBox**, **FruitBasket** and **TomatoBasket**.

Despite the fact that the **FruitBasket** can provide random fruit (apple, banana, and orange), others can only provide one kind of food which is simply shown by their names.

There are 4 kinds of items where food can be placed.

1. **WoodPlatform** for placing at most one food.
2. **TrashCan** for abandoning food.
3. **ApplePieStove**, **SaladBowl**, **SandwichMaker**, and **FriedEggStove** for cooking food.
4. **PickupWindow** for delivering food.

When time is up, the game would enter the end page and the player could click play again to enter the menu to start a new game.