CSC 413 Project Documentation

Summer 2021

Haoyuan Tan(Sunny)

918274583

CSC413.01

csc413-su21/csc413-tankgame-CiYuan53: csc413-tankgame-CiYuan53 created by GitHub Classroom

Table of Contents

1 Introduction

1.1 Project Overview

This project is aiming to make a real world game, and workable, playable and carefully designed game. It gives the coder opportunities to really make a real application for users, not just interacting with the terminal.

1.2 Introduction

This project is a Tank Game, where two tanks are competing to take the other down by shooting the other down. Both tanks are limited in the same map and they can shoot the breakable wall in order to get to other areas. While breaking the breakable wall, there will be items dropped. The tank can pick up those items that can modify their tanks and grants them stronger strength to take the other down.

2 Development Environment

Java Version(SDK): Azul 16.0.1 For Mac M1

IDE used: Intellij IDEA 2021.1.2 (Ultimate Edition)

3 How to Build/Import your Project

1. Make sure your IDE supports Java and install Java's jdk
2. At the home page of your IDE, select Open Project
3. Use this repository folder as the root of this project
4. Click File -> Project Structure and select the correct SDK
5. Click Project Structure->Artifact
6. Click +->JAR->From modules with dependencies
7. Select Launcher as main class and click Apply
8. Back to IDE, click Build->Build Artifacts
9. Now there is a jar version of this project in /out/artifacts

4 How to Run your Project

1. Click the Launcher(which is main) showing on the left side
2. Click run ***.main()

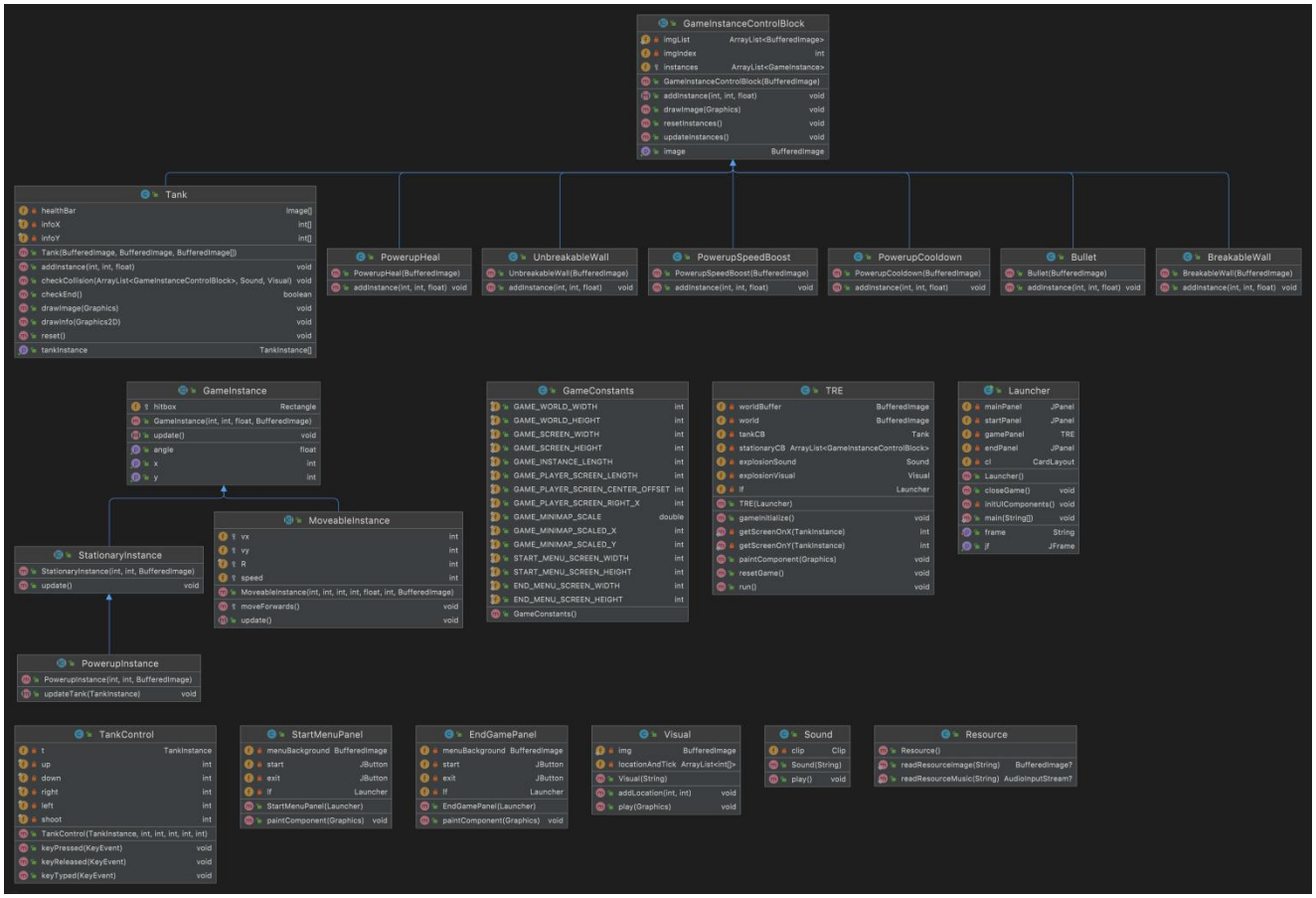
OR if you build the jar

3. Go to JAR folder in this repository
4. Double click to run it OR right click in the IDE and select run
5. Player 1 use Arrow key to control and Enter to shoot
6. Player 2 use WASD key to control and Space to shoot

5 Assumption Made

Users are assumed to know how to use control and shoot key to play like an RPG game.

6 Class Diagram



7 Class Description

7.1 GameConstants

This class contains all numbers that is final and used by the program, specifically for the sizes of elements, such as screen's size and map instances' size. It can keep most elements to be consistent in terms of displaying.

7.2 Launcher (Entry Point)

This class is the entry point of the whole program. It contains the methods to initialize and set up the screen panel for start, game and end. And it will initialize the gaming elements before actually start. It provides a static method called setFrame() so we can switch between the panels when game is started or ended.

7.3 Folder "menus"

7.3.1 StartMenuPanel

This class holds the contents for the panel of starting page. It displays the title image of our game, and also draws two buttons for user to click. Button "Start" will redirect the current panel to the exact game that users can play. Button "Exit" will end this game safely.

7.3.2 EndGamePanel

This class holds the contents for the panel of ending page. It displays the ending page when the game ends. It also draws two buttons for user to click. Button "Restart" will redirect you to the game panel and also reset all instances in the game, including the map, to the initial state. Button "Exit" will end this game safely.

7.4 Folder "game"

7.4.1 TRE

This class holds the contents for the panel of gaming. It runs the game and draws all instances onto the map and display it. It display three main screens: two split screens that are for each user's view, and one as a minimap for users' reference. It limits where the map is going to be displayed to avoid painting errors, especially at the edges.

It also reads all images and music from the resource folder and stores them in the corresponding classes. And it will read the map setting from the given excel file. Then it creates control blocks for all types of instances and draws all the instances.

It is also responsible for running the game. It will reset the whole game first, then by updating every 10ms and check collisions and other actions in order to make the game actually "running". And it will end when finding a tank has a health of 0.

7.4.2 GameInstance (Abstract)

This class is the parent class of all instances, both the stationary and moveable. It has the basic variables, x and y positions, the angle and the important hitbox for collision. It provides only the getters so the outside codes can read the positions. It also requires the child class to include the update() method.

7.4.3 GameInstanceControlBlock (Abstract)

This class is the parent class of all control blocks for instances. It is aiming to store the instances for each type by using the abstract GameInstance class as reference. And as a parent, it stores all images in a static ArrayList for each control block, where each control block is in charge of one type of instances.

This abstract class makes sure its child has an ArrayList of instances to store the corresponding active instances. Also, it requires the child to provide a way to add instances.

It has a general drawImage() function that will be called from TRE, to draw all instances from the control block. As well as the update() function to update all instances. It also provides a function to reset the instances array, and this is aiming to help re-initialize the map when resetting game.

There is also a rule for this control block class, which is it must include another class that is extending from GameInstance so it can stores instances easily within the control block. I design it to be like this so we can easily manipulate the instances using the control block.

7.4.4 Tank (Core)

This class is one of the core classes, which holds everything about the tank. It is extending from GameInstanceControlBlock to be in charge of tanks. This means this class includes an instance class extending from MoveableInstance, which is true tank.

As a control block, it stores the tanks into the list. And it has much more functionalities than other control blocks. It modify drawImage() to draw everything of the tanks, including their bullets. It checks the health of the tank to tell if the game ends. It also contains a new drawInfo() function to draw all extra information that is not the tank itself onto the screen, these are the health bar, live count and other status. It is aiming to tell the change and current status of the tanks for better experience to users.

Moreover, it is in charge of the collision checking, because the tank is the main one that is going to cause collisions (bullets are also shoot from tank). The collision check takes the place of the border checking because all instances including the walls have hit-box. It checks tank to tank, tank to map instance, tank to bullet. When there is a bullet collides on an instance, it will cause a explosion effect using Visual class, playing a explosion sound using Music class and remove if it is hitting the breakable wall. If the breakable wall is removed, it may also generate a new PowerUp at the original position.

The instance class includeed in the control block is the true tank instance. It contains the very beginning position for resetting purpose, and old position from each update for restoring position during collision. It also stores boolean for key being pressed in order to perform specific task while the user prompt the tank to do so. It also stores the RPG elements, which are the status: health, live, speed and cooldown time. It is designed that the tank can't shoot until the cooldown time is ended. It also has a new moveBackward() method so it can move backward using a similar logic to moveForward(). It also sets the hitbox when it is moving so that it can always use the correct position for hitbox. Lastly, it has multiple functions for status getter and changer. For the changer methods, they are different than the setters, because they has their own way to modify the tanks.

7.4.5 TankControl

This class is implemented with KeyListener class so it can read the prompt from user's keyboard in order to toggle the boolean of each key in the TankInstance class for each tank so that the tanks know what to do when updating.

7.4.6 Folder "effect"

This folder contains all class related to elements that makes the game beautiful, such as drawing an effect, or playing an effect.

7.4.6.1 Resource

This class contains all static methods that is used to read the files from resource folders using the given path and handle the possible error such as no such file exists.

7.4.6.2 Sound

This class is to store one clip from the resource folder using the path, and play the sound when it is prompted at play() function.

7.4.6.3 Visual

This class is to store a visual effect that is not going to be permanently drawing on the screen. It stores the effect image, and draw it when it is prompted at play() function. It can store multiple locations to display the effect. It will draw an image of the effect in each position it is currently holding. And it only draws for 100 ticks for each location, after it, it removes that location from the list so it won't draw it again.

7.4.7 Folder “moveable”

This class contains all classes related to moveable instances. (Except the tank)

7.4.7.1 MoveableInstance (Abstract)

This class is extending from GameInstance, but modified for holding more information for a moveable instance. It contains the general moveForward() method to move and uses a round up vx and vy calculated based on the angle. And it also sue the speed as a multiplier to the change of movement.

7.4.7.2 Bullet

This class is extending from GameInstanceControlBlock and it include an instance class extending from MoveableInstance. It will store all created bullet instances and update them each tick after they are existed. Unlike tanks, the bullets should automatically move forward after it was shoot, so we have the moveForwad() call in update().

7.4.8 Folder “stationary”

7.4.8.1 StationaryInstance (Abstact)

This class is extending from GameInstance, but not doing something special. It only makes the update() do nothing because they shouldn’t. Its only purpose is distinguish the stationary instances from the moveable instances. This may help if we want to develop more on the stationary instances so we can easily find them using this class.

7.4.8.2 PowerUpInstance (Abstract)

This class is extending from StationaryInstance. It inherits the same property from its parent, but this class is aiming for specifically for power up items. So other than distinguishing purpose, it also aims to make its child to have a way to modify the tank.

7.4.8.3 BreakableWall & UnbreakableWall

These two classes are both extending from GameInstanceControlBlock. They are like all other control blocks, which they both have their own instance class and both of them are extending from StationaryInstance for distinguishing purpose, specifically for the images. And nothing else so far but we can easily modify separately if we need.

7.4.8.4 PowerUpHeal & PowerUpCooldown & PowerUpSpeed

These three classes are all extending from GameInstanceControlBlock. It has the similar logic like above Wall. But they all have a special method, which updateTank() to change some status of the tank by calling the changer methods from TankInstance class.

8 Project Reflection

In this project, I spent a lot of time optimizing the code, and changing my design so that it fits my need. I thought about a lot, even though they are not really used. But I tried to make it flexible enough, as we can see, there are a few folders for different parts. But I don't think I make it my best work, because tried a design I never used before, which is the Control Block. I didn't have enough time to optimize my project structure for it. Especially on the Tank class, it has a lot of things compared to other control blocks, also the tank instance itself is excessive. However, stationary control blocks are much more simpler and even I feel like my design is not useful when applying to this kind of simple instances. But I think practice new ideas and designs are pretty important for a programmer. I will try to practice this design and also other designs.

9 Project Conclusion

This project is my very first complete application, not the simple one using the terminal in previous classes. It actually has a GUI so that the users can interact with. It is a good taste when using/playing the application we make ourself. Even though I don't have enough time to make it my current best, but I still implement all functions I want and the required ones into the game.