CSC 413 Project Documentation

Summer 2021

Vicente Pericone

918693144

CSC413.01

https://github.com/csc413-su21/csc413-tankgame-ViP-Cente

Table of Contents

1 Introduction

1.1 Project Overview

This project is a video game that has many game objects that are used and drawn on the screen for players to interact with

1.2 Introduction of the Tank game

The tank game has two tanks that can be controlled by two different people where the objective is to reduce to enemy's health to zero and their lives to zero before they eliminate you. There are also powerups across the map that players can grab and use.

2 Development Environment

I am on Java 8 and using IntelliJ. For this project, I used libgdx and some of the libraries they offered.

3 How to Build/Import your Project

After you clone the repository, open up your IDE and open the project and choose the build.gradle file that's in the csc413-tankgame-ViP-Cente folder. This will build the gradle project and you'll be able to see all the classes correctly.

To build the jar, I ran the "gradlew desktop:dist" command in the terminal and the jar appears in the "desktop/build/libs" folder.

4 How to Run your Project

To run the project, you can either run the jar file in the jar folder or run the DesktopLauncher method.

	Player 1	Player 2
Forward	W	UpArrow
Backward	S	DownArrow
Rotate left	A	LeftArrow
Rotate Right	D	RightArrow
Shoot	F	Enter

5 Assumption Made

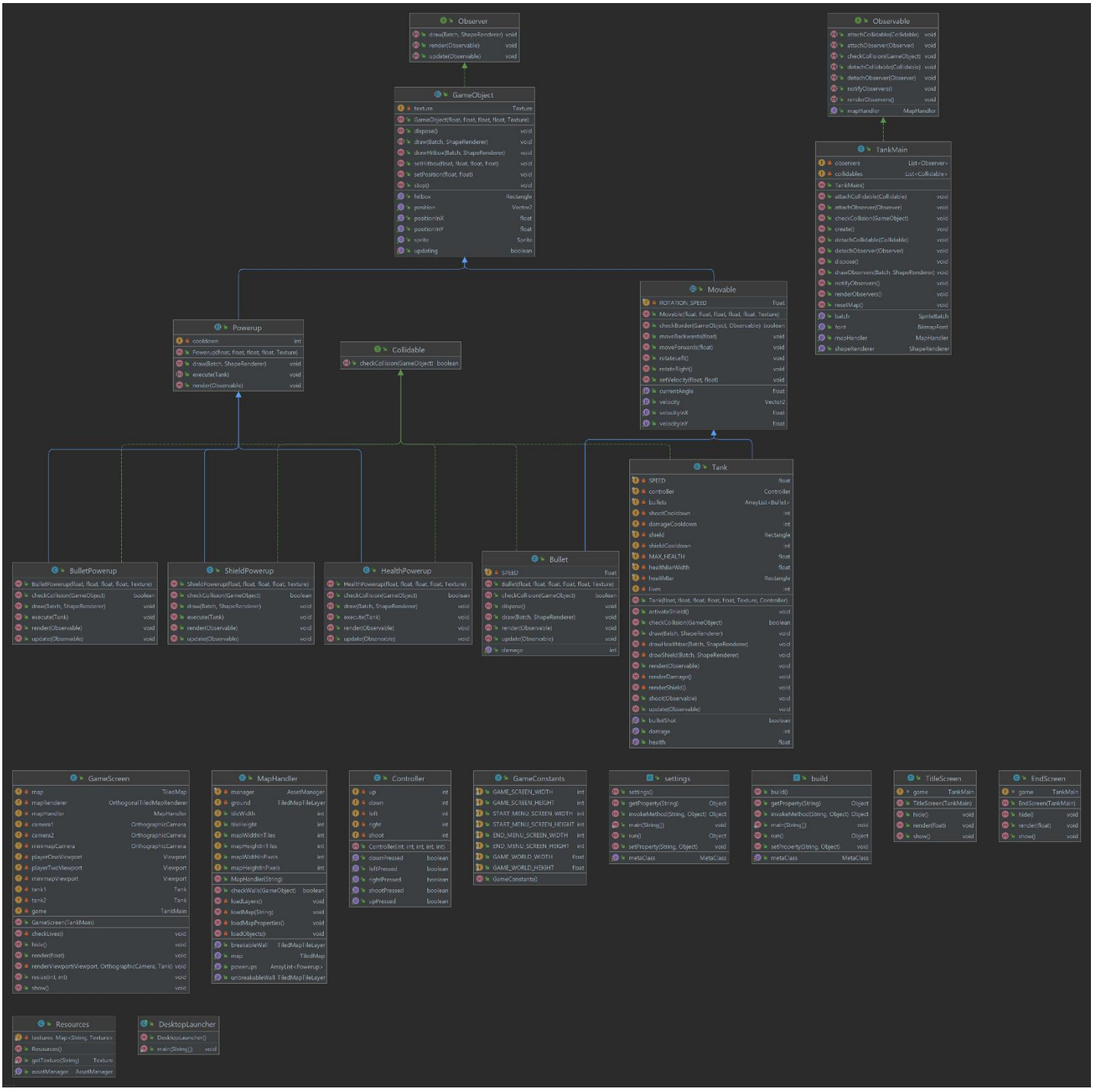
I assumed that the only players are the tanks and that's why the tanks have a lot of methods within it.

6 Implementation Discussion

So design wise, the VirtualMachine interacts with runtimestack based on the execute() function that it is calling at the current ProgramCounter. This programCounter is a pointer to where the current part of the program(Mock Language X) is at. The ByteCodes extends to the abstract class ByteCode that each

have their own execute functions that the VirtualMachine will use. The CallCode, GotoCode, FalseBranchCode also implements the interface AddressLabel which is used find where they will jump to before the program is ran. There is a BinaryOperation interface that I wanted to use for BopCode to make that class a little cleaner looking but I ran out of time for now.

7 Class Diagram



8 Class Descriptions

8.1 Observable

This is an interface that has methods to manipulate the observers in various ways. Methods attachObserver and detachObserver control what objects can be observed or not. The notifyObservers method tells all the observers to update and renderObservers() renders all the observers.

8.2 TankMain

This class implements the observable method and thus, this class if responsible for handling all the observers. This class is essentially the "core" of the game in that all the screens use this class for running their individual screens. There are variables batch, font, shaperenderer that help draw and render text, sprites, and shapes that are used in this game. Then there are two lists, observers and collidables, which are used for handling the objects in those lists respectfully. For example, checkCollision checks the collisions for all the collidables given a certain game object. The create() method is called when first running the game and I use this method to instantiate all the variables we need for the game and sets the screen to the title screen. The dispose method is called when the game stops running. The resetMap() method is called when the game ends and the player wants to restart. It just creates a new map so that the old map is not used for the next game.

8.3 Observer

This is an interface for anything that can be observed in our game. This has the methods draw, update, and render which draws, updates, and renders the object respectfully. At the time of writing this, there is another interface called Collidable which I might combine into just this interface if I have the time.

8.4 Collidable

This is an interface for anything that is collidable in our game. It has a checkCollision method that checks the collision given an object. If I have time, I will combine this interface with the observer interface so this class may not have to exist.

8.5 Game Object

This is an abstract class that represents an object in our game. The constructor takes a position in the game, width, height, and a texture. The objects contain a texture which is then used to be able to make a sprite using the sprite library with libgdx. A sprite is essentially a texture that you can move, rotate, and manipulate depending on what you're trying to do. The position and size taken is used to create a rectangle that represents the hitbox of that object. There are two draw methods, one for the sprite and the other for the hitbox, and these are used to draw those in game. There is a dispose method which is called when we no longer need an object. There is a stop() method which is called when we no longer want are object to keep updating.

8.6 Movable

This is an abstract class that represents objects that can be moved. It extends Game Object and takes an angle in its constructor. It has methods for moving and rotating and a check border function for checking the world bounds.

8.7 Tank

A tank extends movable, and it now additionally takes a controller for checking inputs. This is our player class and where the user can interact with the game. Tanks have bullets where they can shoot and be shot by them. Tanks also can activate a shield with a powerup and increase their bullet damage. Tanks also have a health bar where it changes based on the players health

8.8 Bullet

A bullet extends movable and only moves forward when instantiated. They have a damage variable that can be set if so desired.

8.9 Powerup

The powerup abstract class extends game object and it has an execute method along with its own draw and render functions.

8.10 ShieldPowerup

This class extends Powerup and it activates a tank's shield if collided by a tank. This shield protects the tank from bullets for a short time

8.11 HealthPowerup

The health powerup gives the user maximum health.

8.12 BulletPowerup

The bullet powerup doubles the tank's damage from their bullets and also turns the tank a different color to indicate the new damage output.

8.13 MapHandler

For this project, I used a tiled map which is a program where you can create 2d levels and import them into a libgdx game. This class creates the map used for the game and establishes the bounds for the map. There are 30 pixels per tile in our game. The measurements I use for the entire program is in game unit tiles and not pixels. This class also has a checkWall() function which checks if a given object collides with a tile in one of the "Walls" layers.

8.14 Controller

This class is used as the players controller for the game. You can customize it by its constructor which you can set what up, right, down, left, and shoot are.

8.15 Game Constants

I didn't really use this class aside from establishing the initial screen size for the game. This will likely be omitted in future updates.

8.16 Screens

There are multiple screen classes in this project which are set depending on the state of the game. There is a title, game, and end screen. Each screen takes the game in its constructor and has three functions that are called in the background with libgdx. The show() function is called when the game is first switched to that screen. The render() function is called each frame the game is rendering at. The hide() function is called when the screen is finished and moves to another screen.

8.17 Resources

The resources class contains all the textures used for the game.

8.18 Desktop Launcher

This class is used to run the TankMain class and is part of the libgdx functionality.

9 Self Reflection

At first, this project was really daunting and intimidating. Compared to the other projects, we were given almost no starter code and the idea is so much bigger. It was especially difficult for me because I decided to challenge myself and learn the tools of libgdx and what they have to offer. It was difficult at points because most of the project videos were not helpful to my situation because I was using different tools. Throughout the whole development process, I was googling so many things about how to do certain things with libgdx. The first couple weeks consisted of reading documents and forums and watching videos to learn what libgdx has to offer an how I can apply it to my tank game. I was really frustrated in the beginning because I was very confused about how everything worked but eventually it just started to click. That was the most difficult part of the project for me but now I know that I can research tools on my own to better my arsenal for future projects. I am proud to say that I created a game from scratch with essentially no help from other classmates because I chose to use different things.

There are areas I can improve for this project. One area is that the cameras show the out of bounds area to the player. Another area is that I can implement quality of life things like respawns, sound, effects, but I ran out of time unfortunately. The code could use some cleaning up as well along with more comments and explanations, but I am just glad I “finished”.

10 Project Conclusion/Results

I made sure to do all the requirements listed in milestone two. I wish I had more time to make the collisions better because they are a little iffy along with just making the game look and feel better.

In conclusion, this has been my favorite project ever in school and I had a lot of fun creating it despite the frustrations. This class in general was great and I learned so much about developing and it was very informational. I felt as though I learned a year’s worth of information and skills in the time frame of one summer. Thank you for the great class professor!