CSC 413 Project Documentation

Fall 2019

Lukas Pettersson

916329042

CSC 413.01

https://github.com/csc413-01-spring2020/csc413-tankgame-LukasPettersson

Table of Contents

1 Introduction

1.1 Project Overview

This project implements an action tank game using a JFrame. The game is a 2 player split screen tank shooter. Where each player is represented on each half of the screen by a tank. The goal is to use good aim and tactics to shoot down your opponents tank and bring their health down to zero. While doing this, random objects will spawn around the map that can be helpful in your quest to defeat your opponent. The game has both breakable and unbreakable walls as well as custom item spawning locations which are easily implemented with your own map as detailed in the summary below. The game also has an inventory system where you can pick up and store power ups that are found around the map to use at an opportunistic moment.

1.2 Technical Overview

Setup:

The game implements a graphical user interface using JPanel. I create BufferedImage objects and draw them to a Graphics2d object which is then painted onto the screen using JPanels repaint function. Each object that is part of the screen is a gameObject which is the top level abstraction for all objects that are on the screen. Since a lot of the game objects have different behaviors many are stored in the World class. Walls, Items, and spawn Point data are all stored in ArrayList objects which are initialized in the World Object.

We initialize the map and the objects by reading from a tab separated integer list called map3.txt, maps are 141x141 units, where each unit represents 20 pixels on the screen. There is a 20x20 unit buffer around the play area making the total play area 101x101 units or 2020x2020 pixels. The integer value of each unit in this 2 dimensional map represent which object is going to be in that location. The unbreakable(9) and breakable(2) walls are stored in an ArrayList of walls and the custom spawnpoint(2) coordinates are stored in a 2 dimensional ArrayList.

When the game starts, the initial setup creates a new Jpanel World object, init is then called to initialize the JFrame and other objects. The two players are initialized with their spawn points, angle, and sprite name input as string. The sprite name is the key to a HashMap that fetches the BufferedImage for the correct object using class loaders. In addition I also initialize an EventManager object that is used to spawn different items around the given spawn locations.

The two Controls classes are also initialized and implements a UserControlledCharacter class. The controls class uses keyListener to pick up characters that are pushed on the keyboard. These clicks then toggle Boolean values in the UserControlledCharacter class.

Then the rest of the setup refers to setting up and configuring the JFrame. Implementing keyListeners for the jFrame, sets the size, make it non resizable, set the location, set default close operation, and lastly set it visible.

Runloop:

This is where the program will be run from. 144 times a second the while loop repeats which updates the position of the tanks, as well as listens for and resolves key press events. Resolves collisions that are caused by the updating of the tanks or projectiles and resolves item spawning events. Then checks if the win condition has been met. Which prints who the winner is and resets the game. Together all of these systems work independently of each other, and are tied together in the World class.

1.3 Summary of Work Completed

gameObject Abstraction:

While creating the GameObject abstraction I tried to focus using data fields stored at the correct abstraction level, prioritizing readability and understandability of the code. Each level of abstraction is described below

- The top level gameObject has the necessary data fields x and y for calculating the position of the object, a BufferedImage for knowing which asset Image is tied to the object, an integer angle, which refers to the angle of the object. And a Rectangle object that is used to determine hitboxes for the objects.
    - The Movable class extends gameObject and implements a velocity in the x and y direction called vx and vy, speed R, and rotation speed, as well as the default angle that the object spawns in with.
        - The projectile class implements the Movable class and implements its own update and check border functions.
    - The Character class extends Movable and implements hit points and default hit points for the tank, as well as member functions to manipulate the hit points of the object.
        - UserControlledCharacter extends character and functionality for the buttons and stores the projectiles and inventory data. This class is also implements its own update and check borders function that have to be different from the projectile class implementation because the movement is driven by the user, and it also has a different check border function because tanks and projectiles have different behaviors upon collision with the border.
        - This class needs to have different update and check border functions from

- The Stationary class extends gameObject and exists to give non-moving objects their own abstraction since they do not need to implement heavy update or moving functions.
    - Item extends Stationary and implements the max number of items that are available, as well as an item type depicting what an item does when the player decides to use it. Due to time constraints, there is only one item implemented which is a max shield item, however the abstraction is built so that other buffs to the character can be made without much tweaking of the source code.
    - The Wall class extends Stationary and is used to describe walls, it has a Boolean isBreakable which dictates if the wall can break or not, as well as a lifeCount, which specifies how much life a wall has. The reason that breakable and unbreakable walls are the same object is so that they can easily be stored in the same place and operated on the same. With only one check to see if it is breakable and take the right action in that case.

Player Abstraction:

The Player object exists to consolidate all the data of a single user, this includes the life count of that user, their UserControlledCharacter and their inventory. As well as member functions to interact with these objects. All of the functions that work on any aspect of the player should go through this class to fetch that object first. In Hindsight the GUI should be part of this class as well. However it is currently implemented in the World class. The GUI fetches data from the player object and displays it on the screen. Using Graphics2d objects, the gui function is called in the paintComponent function inside the World class.

World Abstraction:

The world class holds main the run loop and the setup that is required by the system. It initializes the world from an Integer map, creates the two players for the game, initializes all the classes and containers that are necessary for the game and runs the games run loop.

The SpriteData class is put here because it holds initialization data for all the objects in the game, stored as a HashMap of strings and BufferedImages used for the objects. The EventManager is called every run loop and is responsible for creating items at the preset locations that are stored in the World class. The collision detector is also called once every game loop and checks if any object is collided with another. It calls functions for each type of collision that happens, since each object has a different collision outcome I decided to use separate classes for resolving different types of collisions.

2 Development Environment

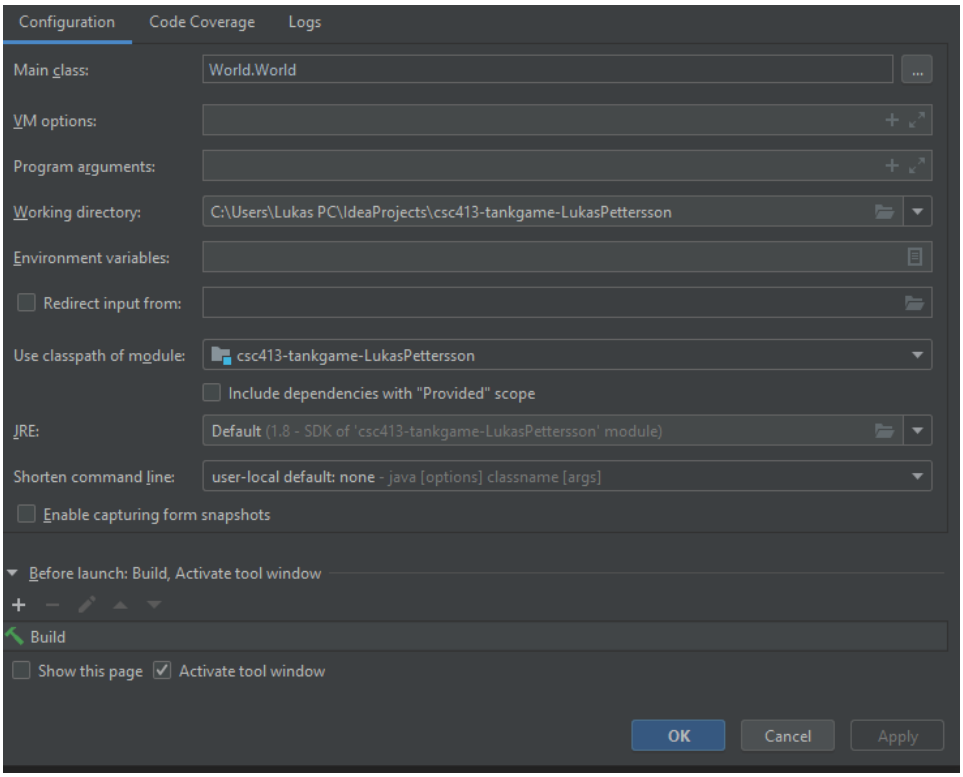
Java version: 13.0.1

IDE: IntelliJ IDEA community edition

3 How to Build/Import your Project

BUILD AND COMPILE PROJECT

1) Clone the repository from Github down to your own machine.
2) Open the IntelliJ project by clicking on the project in the Github project folder.
3) Choose the 'World' configuration by choosing Run > Edit Configurations.
4) The settings for the world configuration should be as follows:



5) Click Run > Run ‘World’

6) The project should now build with the World configuration.

4 How to Run your Project

RUN GAME FROM JAR

1) To run the game from the Jar file, clone the repository to your local machine

2) Using terminal navigate to /csc413-tankgame-LukasPettersson/jar/

3) Run the command ‘Java -jar ./csc413-tankgame-LukasPettersson.jar’

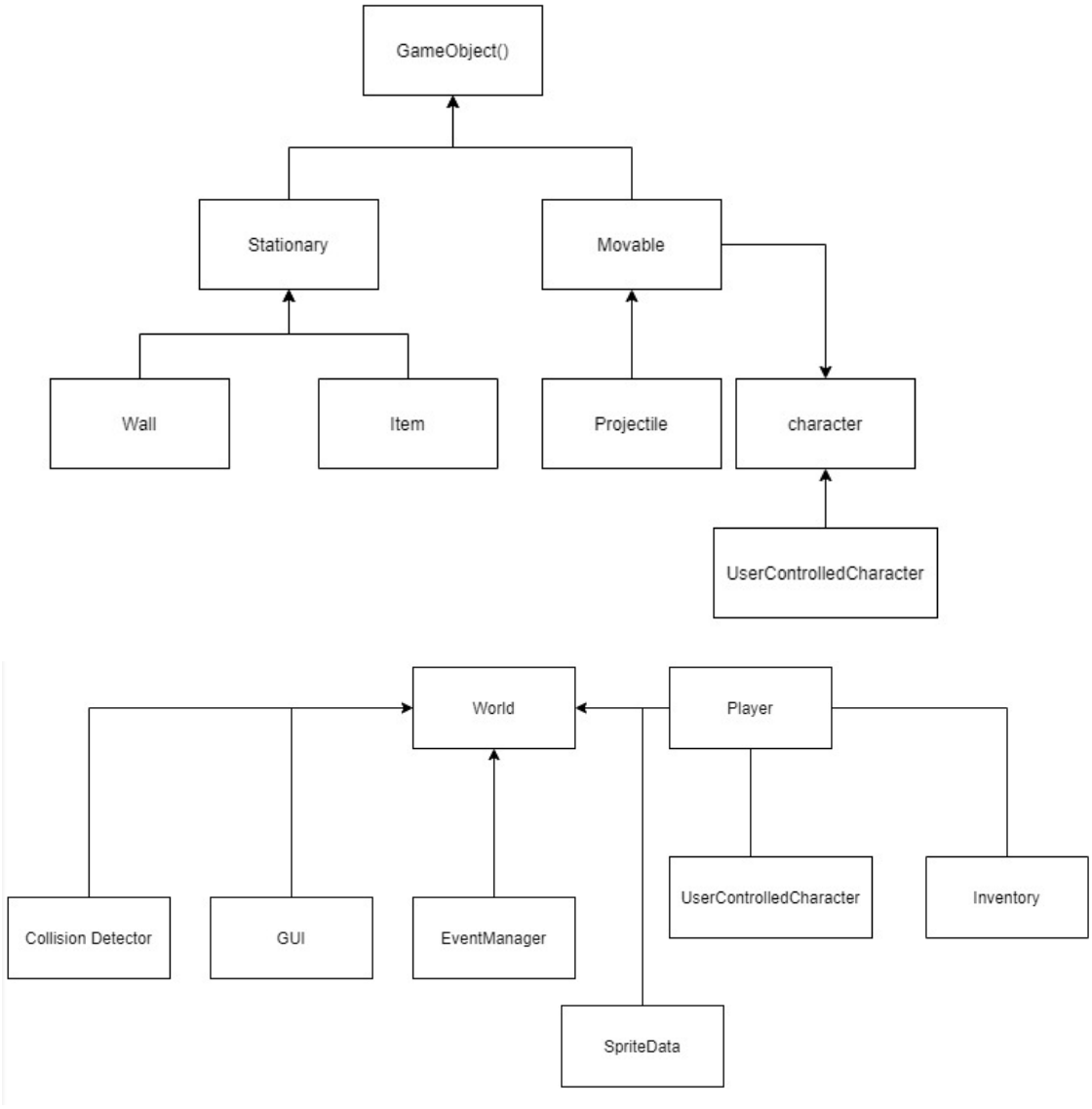
4) The game should launch after running this command.

5 Assumption Made

-The map has a 400 pixel buffer on each side to avoid throwing raster exceptions due to trying to draw outside of the map.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

Going into this project I had a lot of ideas of what I wanted to do and was very ambitious about the goals that I wanted to achieve with the time that I was given. I feel as if I let myself down a bit with what I was able to accomplish vs what I wanted to accomplish. I had the idea that the tanks would pick up different types of items (both ranged and non-ranged weapons) that they would be able to fight monsters with. I wanted to create NPC's that had their own ability to move and fight against the user-controlled characters. Due to the current situation I was unable to complete all the

goals that I had set for myself to realize a product that I know I could have done better given more time. In the end I fulfilled all the requirements for the project, however, I wish I could have gotten more interesting items and non-player character implemented as well. Even though I wanted to implement more features, I would like to highlight the features that were implemented, There is an Event Manager that is able to spawn items given that there are spawn locations detailed on the map, this feature implements a little bit of luck to the game, given that the spawns happen randomly, this would also spawn different types of items given that more items were implemented in the item class. Another feature that I am proud of is the player inventory, here items are stored to be used later. I thought it would be a neat idea to have the items be removed in the order that you pick them up because it gives the game strategy where you might want to pick up items in a different order, giving you a competitive advantage. Features I would like to implement to this would be showing the exact items that are in your inventory, as well as add more types of powerups.

8 Project Conclusion/Results

I loved working on this project and for the most part it went smoothly, I started off with a good abstraction that made it easy to implement new features. Working with the code that was given to me in the tankRotationExample class was very helpful and I was able to implement the desired parts in the right places. I have learned that I do not like working with JPanel and User interfaces. Even after this project it feels daunting thinking about having to implement my own GUI. The game uses my own graphics for everything but the tanks, which were provided to us. I did not have time to set a win condition and program a restart screen to the game. I would like to do that as well when I have more time to put on this project. I managed game objects well and removed items quickly when they collided

with the games bounding box or ran off the screen. I could have done a better job on the World abstraction because that class carries a lot of data and could potentially be broken down more.