CSC 413 Project Documentation

Spring 2021

Alejandro Zamora Ruiz

917841098

CSC 413.02

https://github.com/csc413-sp21/csc413-tankgame-zruiz95

https://github.com/csc413-sp21/csc413-secondgame-zruiz95

Table of Contents

[1 Introduction 3](#_Toc522827688)

[1.1 Project Overview 3](#_Toc522827689)

[1.2 Technical Overview 3](#_Toc522827690)

[1.3 Summary of Work Completed 3](#_Toc522827691)

[2 Development Environment 3](#_Toc522827692)

[3 How to Build/Import your Project 3](#_Toc522827693)

[4 How to Run your Project 3](#_Toc522827694)

[5 Assumption Made 3](#_Toc522827695)

[6 Implementation Discussion 3](#_Toc522827696)

[6.1 Class Diagram 3](#_Toc522827697)

[7 Project Reflection 3](#_Toc522827698)

[8 Project Conclusion/Results 3](#_Toc522827699)

# Introduction

## Project Overview

In these two projects, the goal was to implement all my knowledge of Object-Oriented Programming and develop two games. The first game that was developed was a “Tank Wars” game and the second was picked from a list of many other games, “Pyramid Panic”. Both these projects were implemented using the latest version of Java using the IntelliJ IDE.

## Technical Overview

Both the Tank Wars game and Pyramid Panic use Java files and resources bundled up into a JAR far to run a fully working game with controls, resources, and goals. Most of the games functions including character movement, map style, power up specifications were handled in dedicated java classes that were called in the main class.

## Summary of Work Completed

The goal of these projects was to use the Object-Oriented principles learned during the semester to create two game. Both games are optimized to run on practically any system using very low resources. For both games, the only provided resources were the assets for the visual effects like background, character images, and walls. All other code was created from scratch to work with the resources to create a full game. Java class files were created in both games that handled different aspects of the game from breakable walls to the way each character’s act. For both games, a game map was stored in an array which allows the user to jump into the source file for Gameworld to change the way the map looks to fit their needs.

# Development Environment

System specifications and other relevant information:

* System:
  + OS: Windows 10 20H2
  + CPU: Ryzen 7 2700x Eight-core 3.7gHz
  + GPU: Nvidia GeForce GTX 1650 4Gb
  + RAM: 16Gb 3200mHz
* IDE & Java Version
  + IntelliJ Ultimate 2020.03 Build #IU-203.7717.56
  + Java Version: 16 2021-03-16

# How to Build/Import your Project

To import the game into the IDE, just click File > Open > Open > Navigate to the game folder > Open. If it asks to open in same window or new, click New Window to avoid losing any current work. Build the project by importing the files into your desired IDE and building it using the “Build” button. It is best to use IntelliJ for this because that is the IDE that was used for development.

# How to Run your Project

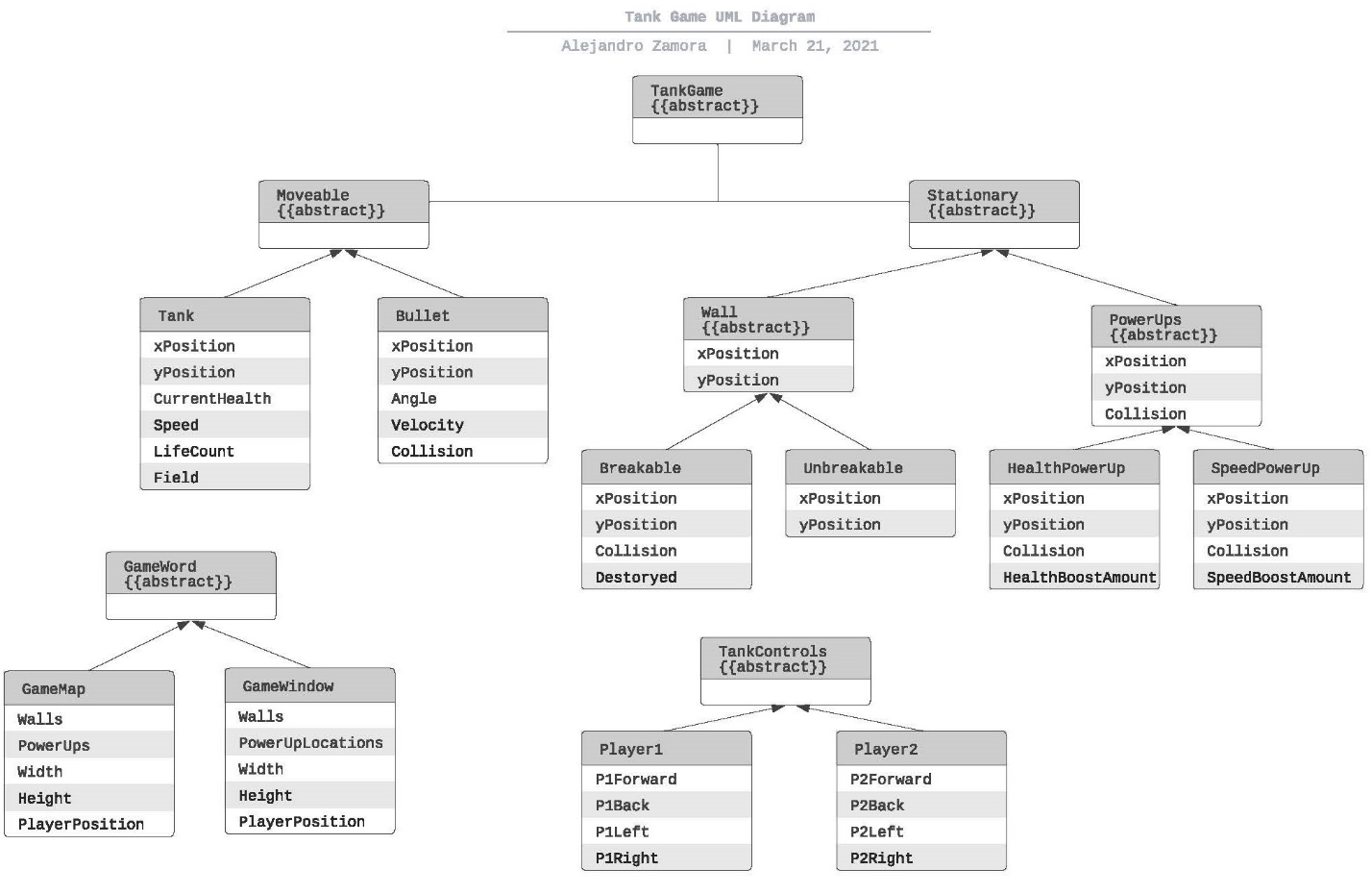
Once the project has been built, you may run the project by using the “Run” feature which will launch the game in the IDE. Alternatively, you can also bundle the app into a JAR file by going to File > Project Structure > Artifacts > Click the Plus and select JAR and from the menu select “From Modules and Dependencies” > Locate the Main class > Click OK. Once the JAR file is ready, click OK once more and then navigate to Build > Build Artifacts. This will create the JAR file you can run outside of the IDE.

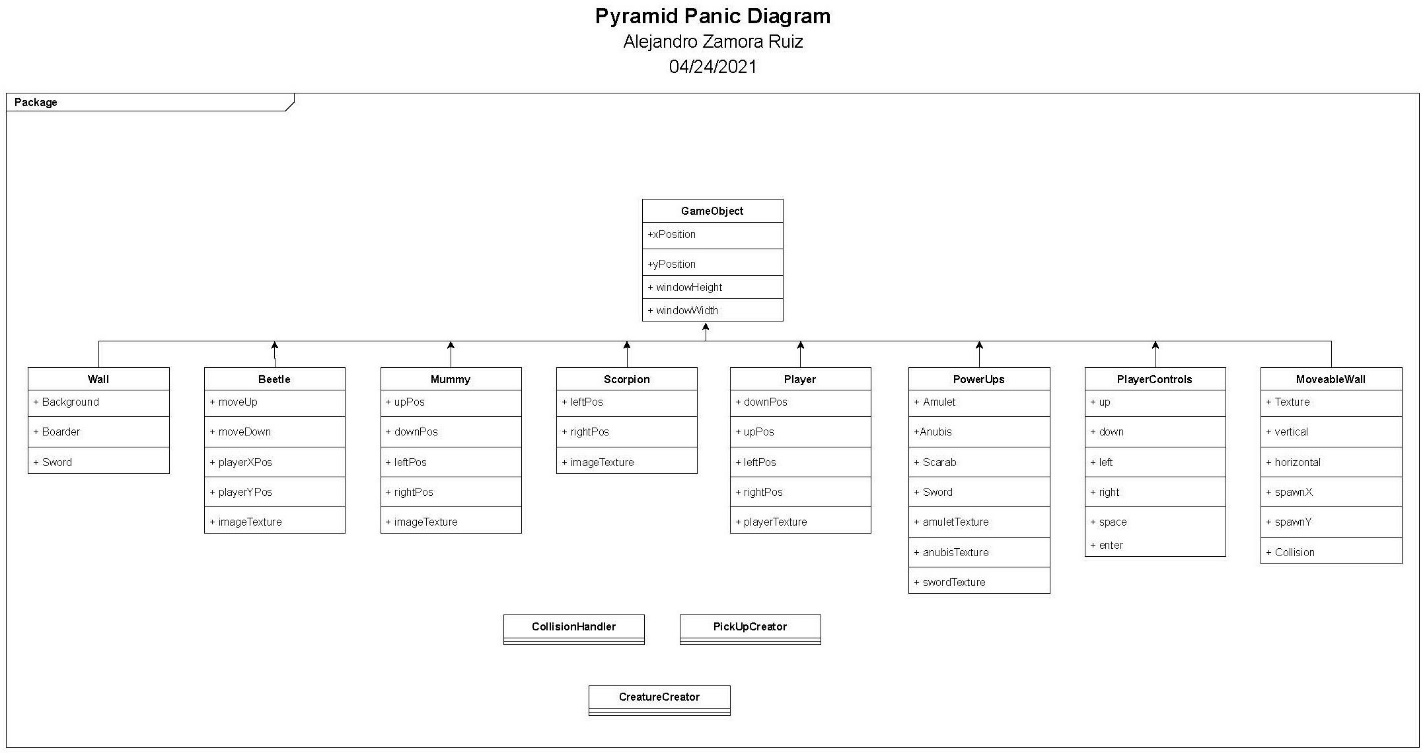
# Assumption Made

At the start of the project, I felt confident that the work was going to be difficult, and even though it was, through some trial and error I managed to get the games to work. Some assumptions I made before I started the games had to do with how the player characters will interact with the worlds. When it came to the Tank game, I assumed I would be able to create multiple levels for the player or that I could change the spawn point after every death. Even though it is possible, I did not have enough time to implement that. For the Pyramid Panic game, I would have loved to implement a feature that would allow the player to respawn in the same spot they died in much like most triple-A titles today. For both games, I assumed that I could create multiple levels for the player to play I but that was too difficult for me at the time. Both games were fun to create and although they don’t have those features, it is something I could implement in the future and the games are fun on their own as they stand.

# Implementation Discussion

## Class Diagram

Tank Wars Diagram:

Pyramid Panic Diagram: 

# Project Reflection

As stated in the Assumptions made, even though both games were hard to develop while maintaining proper OOP principles, like other projects in this class, helped develop my skills further as it put everything I learned in the class to the test. It was a very engaging set of projects that are left with plenty of room to expand as I develop my skills further by adding features to it that were omitted from the final presentation. Eventually I will further develop the projects to have multiple levels, features most games have like respawning in the same place the player character died and music.

# Project Conclusion/Results

In conclusion, both games are complete and playable on almost any hardware. They each have the basics needed to have fun from powerups to enemy characters that reduce player life when interacted with. The Tank Wars game has two players that can be controlled using the keyboard independently, a mini map and power ups both players can use. Pyramid Panic has enemy NPCs that affect the player when interacted with, pickups and powerups along with a sword that starts a countdown timer that ends the game when either the player makes it to the spawn point or the timer runs out of points.