Justus Griego, Sami Lababidi, & Lucas Derr
Dr. Montgomery
CSCI 4448
5 December 2021

Project 7 Report: *Battle Tanks*

Name of Project & Team Members

Project Name: **Battle Tanks**

Team Members and Contribution:

Justus Griego - jugr2908@colorado.edu - CSCI 4448

- Event Handling & Organization of Code
- All Menus
- Destruction between tanks
- Destruction of blocks
- AI/Computer Tanks
- Assisted Level Progression
- Assisted Map creation with AI tanks
- Leaderboards input and output

Sami Lababidi - sala8822@colorado.edu - CSCI 4448

- Map class
- Levels class with .txt files for each level
- Hit detection between tanks and blocks
- Hit detection between projectiles and tanks
- Different types of blocks

Lucas Derr - lude5778@colorado.edu - CSCI 4448

- Tank movement
- Tank rotation
- Projectile spawning and movement
- Projectile hit detection
- Tank hit detection with wall
- Different tank types

Final State of System Statement

The work done for the project is currently fairly basic. The group spent a lot of time getting familiar with the Pygame package and using Object-Oriented Patterns inside of Python.

Overall design is currently the least of our worries so tanks, maps, menus, and buttons are just simple boxes and text. Once functionality of the game is complete it will be easy to go back and redesign things to look more professional.

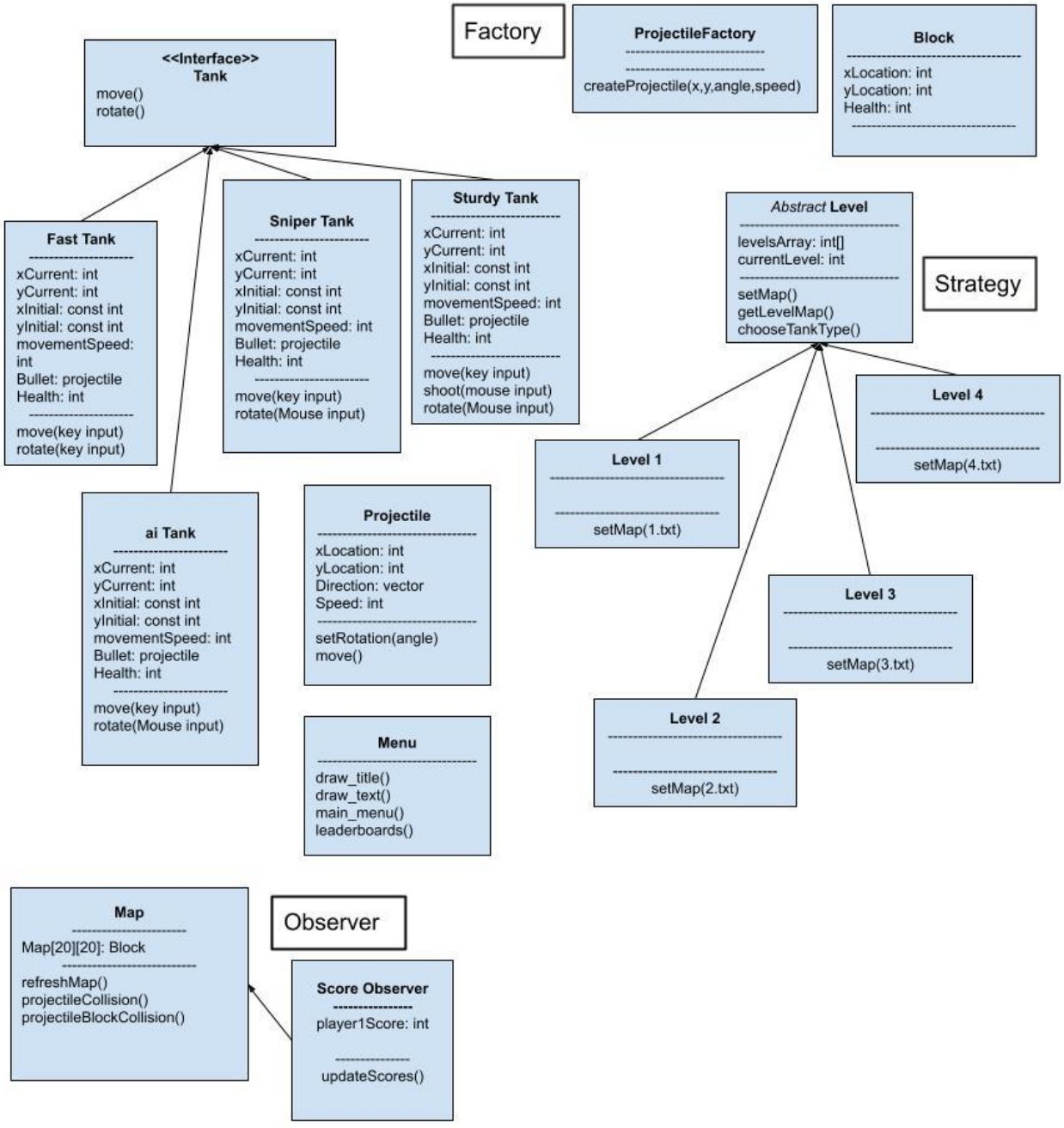
As for functionality we have the basics of a python game. When you start up the game you are greeted with a menu that allows you to select single player or leaderboards. Clicking on leaderboards shows the top 5 names and scores that are being stored inside a .csv file. When you select single player you are prompted to enter a name and select a tank. Once selected you start the gauntlet and fight other tanks. Yellow and purple blocks are indestructible while everything else isn't. Shooting an enemy grants 50 points to the player, getting shot by the enemy records your score to the leaderboards and exits to the tank selection screen. When getting past all the levels the player is greeted with a congratulations screen showing their score that is then input into the leaderboard file.

Two features that did not get implemented are multiple types of AI tanks and multiplayer. The reason for not having multiple types of tanks is because when we realized we were short on time, this feature wasn't needed for a complete game. The reason for not having multiplayer was we struggled to figure out how to use two mouse inputs and ultimately decided this was one of our last priorities.

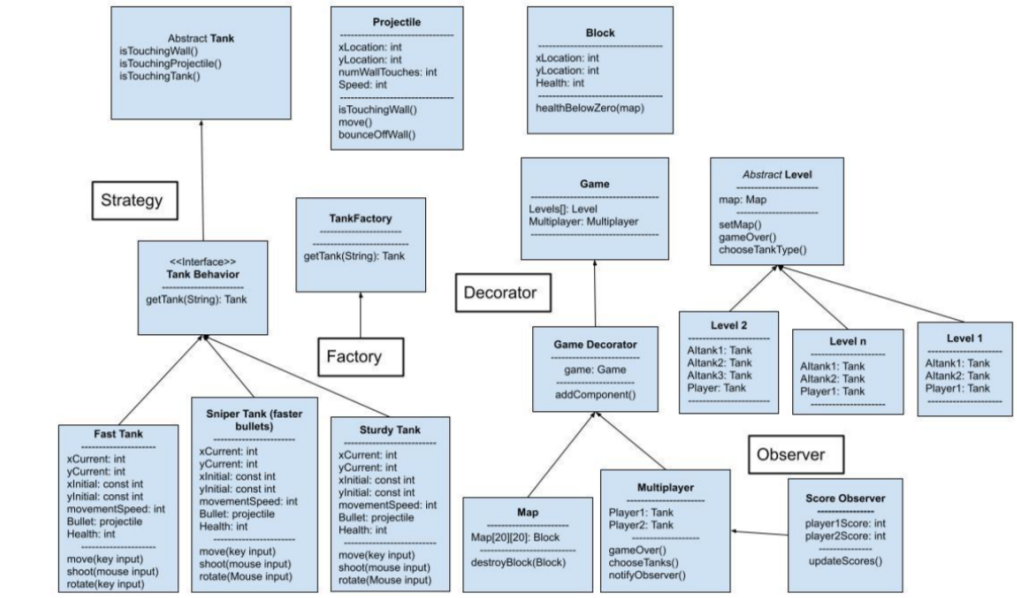
Overall I'm very satisfied with our final product. Although it's quite basic with limited functionality the use of our Object-Oriented techniques will make it very easy to edit, modify, adjust, and add new features.

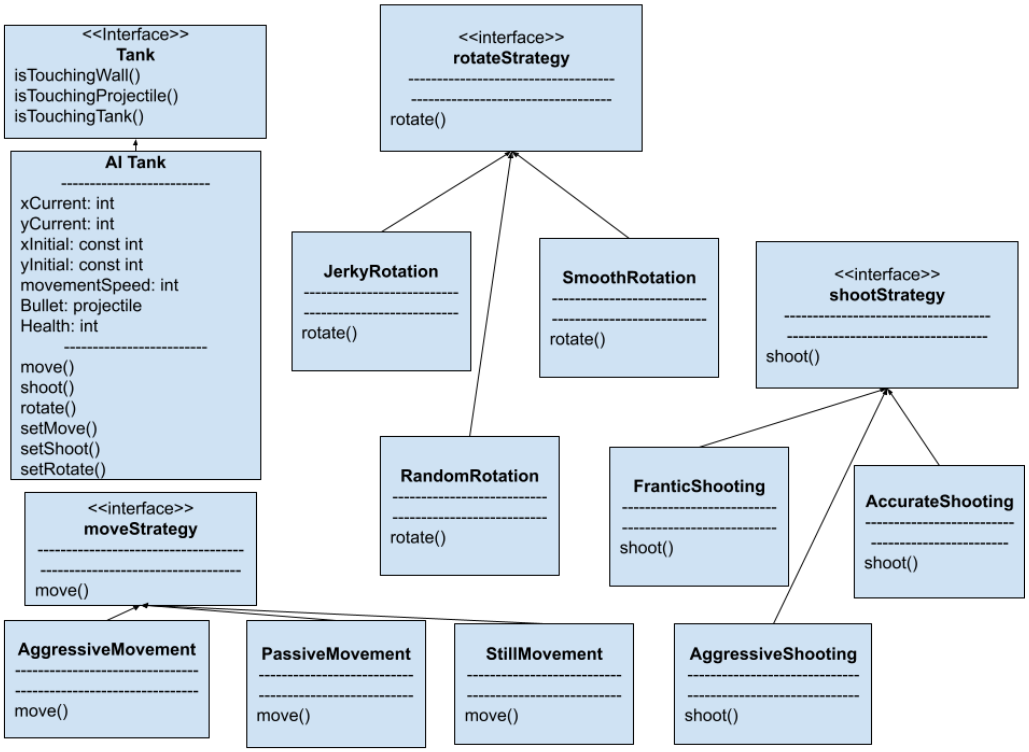
Final Class Diagram and Comparison Statement

Final Class Diagram



Project 5 Class Diagram





Analysis of Class Diagrams

Like many coding projects, our initial design for the project was slightly ambitious when compared to what was accomplished, however, we still managed to implement all the main features of our initial design. As seen in the final class diagram, the tank classes are derived from the tank interface which define the only tank functions: move() and rotate(). The biggest changes from our initial diagram to the final one are: lack of AI capability, omission of the decorator pattern and game class, and overall simplification of object-oriented design features. The reason for the lack of AI capability was simply time. Getting all the core features took priority over AI, and in order to get a functioning demo, our AI tanks were limited in capability as compared to what was described in the initial class diagram. The reason for the lack of a game class with a decorator pattern for maps was because it simply was not necessary. Our levels were loaded in with .txt files and stored in an array, which was accessed in the game loop. There was no need to waste time creating a game decorator pattern when other essential features needed to be implemented. Finally, the reason for simplification of object-oriented design patterns is because of our use of Pygame. Pygame is a python package with many built in functions and classes designed for game development. Trying to adhere to object-oriented principles while using Pygame was at times difficult because Pygame allows for simple code to generate functionality. Despite this, we still managed three design patterns in our final demo, and would have had one more if we had more time to implement AI tanks as intended. Overall, the functionality of the code took priority over adhering to the initial class diagram, which resulted in a satisfying final product that was fun to create and is easy to use.

Third-Party code vs. Original Code Statement

We didn't use very much code from online. In regards to pygame many other projects were not object-oriented thus making it more difficult to understand how we were going to develop our game specifically but we did reference some other general pygame projects to get a good understanding of how the package works. Of course we used a multitude of resources like stack overflow and Google searches to get past annoying errors and bugs.

Statement on the OOAD Process for your Overall Semester Project

Using the OOAD process and concepts made it easier for the team to work together and helped us stay more organized. It also made the process smoother since we already had our class and UML diagrams so when we started coding we had a reference of the design to go back to. These diagrams also helped us delegate tasks between team members. Even though we made some changes from the original plan, these changes did not change the structure of the classes significantly.