CSC 340	Team Zero	Spring 2018
Software Engenering		Wesley Adams

Overview, Description and Deliverables

TeamZero Team Members

- Wesley Adams
- Dennis Kovarik
- Jonathan McKee
- Levi Butts
- Willis Doering
- Sam Backes
- JD Pessoa
- Adeshkumar Naik
- Gwyneth Kardelis
- Aidan Anderson

Client

Our clients for this project are the students enrolled in Software Engenering (CSC340) in the Fall of 2018, and Dr. Hinker, our professor. The students will use our platform to implement their own tanks and they require that that process be as easy and straight forward as possible.

Project

Elevator Pitch

Team Zero has developed a complete API platform for a turn-based version of the classic game, Tanks. The tanks can move on a grid and shoot projectiles. Developers to this platform will be able to create their own tank AI's and battle them against eachother.

Rules

Two to four tanks can play each game. Each tank will take its turn. A tank may choose any of the following moves:

1. do nothing
2. move up
3. right
4. down
5. left
6. back up
7. fire
8. turn up
9. turn right
10. turn down
11. turn left

When a tank choses to fire a projectile, the projectile will move two tiles per turn. If A tank gets hit by a projectile, It looses a life and its hit counter goes up. If the projectile reaches the end of the map without hitting a tank, the tanks miss counter goes up.
When a tank looses all of its lives, it is eliminated.
The last tank standing wins!

Purpose of the System

Abstraction - One purpose of TeamZero's implementation of Tanks is to create a platform that is simple and abstracted at the AI programmers level. While tank developers will be able to access our API funcitons, the rest of the implementation of our platform will be hidden from them.

An AI challenge - We designed our platform to be turn-based, and take place on a grid so that it can resemble an "Abstract strategy game". Other games in this category include Chess, Checkers, Go, Othello, and Mancala. The main premise is that each player needs to find a move that will put them at the greatest advantage. In other words, the players pose a series of puzzles to eachother. These types of games make great AI problems. AI strategies like "state-space search" and "minimax" could potentially be used when writing tanks for our platform.

Deliverables

The client deliverable is a complete tanks platform with the ability to program tanks using the API framework. Along with this will be delivered proper documentation on these API functions, as well as select information from this comprehensive document. Also, two sample tanks will be included in the comp folder.

User Stories and Product Backlog

Overview

The purpose of this document is to give the reader a thorouch understanding of not only the product created by TeamZero, but also of the process hehind developing the project. Once the team had a purpose for the product set out, it was a matter of getting organized and creating clear definitions of what needed to be done. In order to define actionable items for the prodect backlog, the team had to define the requirements for the project. This was defined through user stories.

User Stories

These User stories are collected through the project description and our own team's vision of this platform.
Some base requirements identified for the platform's playfield include:

- Track and manage at least two tanks
- Track and manage at least two projectiles
- Manage a turn counter
- Record projectile hits and misses
- Display the game state at each turn (i.e. watch the game being played)

These were our default User Stories and we were able to expand these into some more specific and personalized requirements.
Here are our resulting user stories:

As a User I would like to...

- Look up helpfull information in the gitlab Wiki page
- Write my own Tank based on the requirements given by the platform developers
- Dynamicaly load my tank onto the platform and battle up to 3 other tanks.
- Be able to view the status of the game at all times (Turn count, Tank health, hits, and misses).
- View the actions of the tanks each turn (Use idles to make the actions viewable to the naked eye).
- Identify the winner of the battle in a timely manner. (If the Tanks are just running in circles, the progrom shouldn't continue running indefinatly)

Product Backlog

Our team used gitlab's "Milestones" and "Issues" featurs to create a backlog for the user stories. First, these user stories were interpreted into things that needed to get done by the developers. These tasks were added to the "Milestones" tool in gitlab. From there. Devops team members created issues based on these milestones and assigned them to the appropriate team lead. The team lead then either worked on the issue or passed it off to someone else on that team. When development on that issue was completed or progress was made, the commit message identified what issue that the push is addressing. It is important to note that it took some time to refine this process and there were plenty of "sloppy" commits without propper identification to the problem it was solving. By Sprint 3 we were using this process fairly consistently.

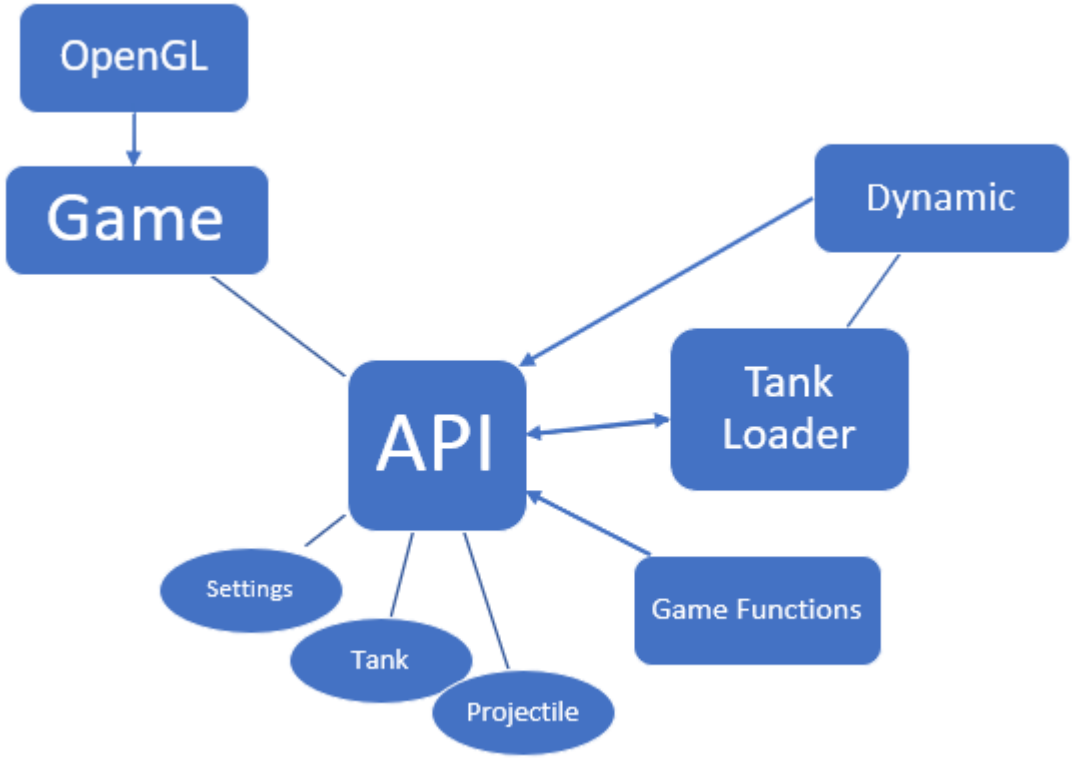
System Overview

The program is divided into 6 main classes, outside of GLUT visualization. The classes are divided into Settings, Tanks, Map, Game, GameFunctions, TankLoader, and the API.

Settings is a basic class designed to hold variables the game need to operate properly. It can be seen as the options of the program. Tank is a class for the tank object. It holds all the stats for the tank like health, hits, misses, direction, speed, and position. It also contains all the functions pertaining directly to the tanks. This includes tank moves, position and damage. Map is a class deigned for graphical interface. It does not hold the actual tanks and projectiles positions, it merely draws them when necessary. Map also has most other draw functions.

The game itself run inside class Game. This is where the turns are played out and where the game starts and ends. The GameFunctions class is where the function for each players turns are, where tanks are spawned and moved, and where projectiles are fired and destroyed. TankLoader is what loads the AI and player tanks and initializes them into the program. Consider TankLoader as an input selector. The API is a separate class entirely. It has all the functions the API developer will be able to use (public). It will also have most other functions needed to run the game (private/protected). API is a friend class of every other class in the program.

The following is a diagram of this system:



Project Management

Team Member's Roles

Our team, TeamZero, consisted of 5 sub-teams. The first sub-team was Developers with Samuel Backes, Aidan Anderson and JD Pessoa. Developers were in charge of raw code generation as well as adding key features. The second sub-team was DevOps with Jonathan McKee and Adeshkumar Naik. DevOps was tasked with managing the git server and any administrative issues e.g. merging, pipeline. The third sub-team was UI with Gwyn Kardelis, Levi Butts and Will Doering. UIs responsibilities consisted of developing the visual side of the project and then integrating it into existing code. The remaining two members of the team were Dennis Kovarik, who was tasked with Quality Assurance, and Wesley Adams, who filled the role of team leader and Communications. Quality Assurances job was to oversee unit testing, schedule and conduct code reviews and make sure everything was running smoothly. Wesley Adams was tasked with decision-making, organizing meetings and resolving any issues that occurred alongside extensive documentation.

During the development of the project, a few members roles morphed so that they overlapped with multiple sub-teams. This occurred as we all discovered our own natural advantages with different jobs. Adeshkumar collaborated with Communications quite frequently to assist with large amount of paperwork required for that job. Both Dennis Kovarik and Jonathan McKee were integrated into Development as their tasks required them both to be heavily knowledgeable with the innerworkings of the program.

Project Management Approach

Our team employed agile development for this project. This meant we developed the program in three, two-week sprints with a heavy emphasis on moving quickly and accurately. Because of this, extensive testing was required for each sprint. This then resulted in weekly meetings and code reviews. Thanks to a team poll, we were able to find a time each week when most of the team could meet up. Team meetings generally entailed Wesley Adam going over the tasks completed this week as well as any task that need to be completed for the next milestone. After going through the issue backlog on git, the team would proceed to an open-floor style discussion group. Changes could be suggested, and code reports could be made by their respective sub-teams.

Alongside team meetings, there were sub-team meetings and code reviews. Sub-team meetings consisted of an entire sub-team grouping up to either discuss issues or work on the project together. Generally speaking, two or three members of other sub-teams were also present to provide feedback or answer questions. Code reviews involved Dennis Kovarik putting complex code on a projector screen and the team would go through it line by line to find any bugs. Individuals were also tasked with less complex code to review. Once a section of code was reviewed, the reviewer would contact Dennis Kovarik and the person who made the reviewed code with the results of the code review.

Sprint Schedule

For the development of Battle Tanks we used one one week sprint and three two week sprints. Sprint Zero had the following goals:

- Repo is created, permissions set, development branches need to be created.
- Stakeholders are identified and user stories are written.
- Repo wiki is started.
- Warm and breathing CI tests created.
- Pre-commit hooks / testing / formatting is decided on.
- All team members get a working development environment.
- Game Engine is designed.
- UI team creates wireframes for team approval.

By the end of Sprint Zero we had all these things completed.
Sprint One had the following goals:

- Hinker must be able to clone build and run test
- Data organization structure
- Integrating data structures with git branches
- Render the map

- Render tanks/ orientation
- Screen wrapping work
- Get going on unit tests

By the end of Sprint Two we achieved all these goals. We had implemented an ASCII display system and we could input commands for the tanks from an input file. The projectile class was written but would later be overhauled.

Sprint Three had the following goals:

- Create the API
- Implement the OpenGL display for the tanks
- Make it sparkle.

By the end of Sprint Three we had implemented an API that allowed us to program custom tanks and had implemented the OpenGL display.

Development

Development Environment and Setup

The Team Zero uses specific development platform. The current development platform the Team Zero Software uses Linux and OpenGL. The Communications team of Team Zero has put up instructions for extra tools and setups. The breakdown of our environment, along with these instructions can be found in the new developer document

Source Control - Branches

Initially when we started off we had 5 main Branches :

Master - That was primarily our release branch that was always protected and nobody could push to it unless there was a merge from one of the existing branches to the Master

Devel - This branch was primarily used by the developers to push their classes(Tank/Projectile/GameClass/Ga to Gitlab. This branch was technically our working branch and any bugs fixes or feature enhancements were pushed to this branch. Most of other branches were branched out of devel for the other teams like QA and UI to work on. After the branches were merged into Devel by Jon and then we had a working branch ready for a code review. Only after a code review and other fixes, we merged the devel branch into master.

Dev.ops.testing - This branch was mainly to write most of the Devops Test. Most of the coverage and catch tests were written to this branch which was latter integrated and ran on any commit by anyone to Gitlab. Later on during the last phase of the development due to multiple merge issues and other conflicts we decided to make dev.ops.testing our main working branch.

UI - This branch was mainly for the UI team to work on. The UI team always branched out of devel which was our working branch and wrote the Map functions, Open GL functions/ Glut functions. After the UI team was done with their features for the sprint Jon had this branch merge into Devel.

QA - This branch was primarily for the QA team to break our working directory. In this branch we had it branched out of Devel and had the QA team break things and find bugs. After the bugs and other features were fixed in the code reviews, This branch was deleted.

Apart from that we had multiple branches created during development cycle just in case we were messed up on the merges which Jon later reverted/cherry picked and got things fixed. This was to Ensure that we didn't lose our old content. During the development cycle developers did run into issues after certain commits were reverted and they weren't upto date with current branch, but these were fixed as the development cycle came to an end.

Testing

Team Zero developed the software Tanks through (mostly) using Test Driven Design. This was achieved through using unit tests, integration tests, and system tests. There were two certification cycles for testing the product. The first was completed after the modules for the gameSetting, map, and tank modules were completed and tested through unit tests. This certification cycle ended with a code review. The second certification ended on 10/14/18. Aditional testing information can be found in TeamZero_Test_Plan.pdf.

Unit tests - Unit tests where designed to test one feature of each module each module individually and independently. Unit tests were required to cover (as much as possible) all relevant possibilities and expected results for each module. Catch.hpp was the tool used to drive unit testing. Unit tests for each module were stored in separate files mainly found in the catch folder. After the majority of unit testing was complete, we moved on to integration tests.

Integration tests - Integration tests were mainly used to test functions in the gameFunctions files, and they automated whenever possible. These tests would include construction objects from several different modules and expecting output from the result of several different modules. In addition, the QA team constructed a separate game loop and helper functions to test the interaction between the different modules and produce a ascii version of the game.

System tests - System tests were performed at the end of the second certification cycle. All system tests were done manually and the results of the test were determined by inspection. To test the platform, The QA team designed an individual tank class that was dynamically loaded onto the platform. These tank classes were modified in various either move randomly on the platform or to move in certain ways to test the platform itself.

Code Reviews

Regarding the code reviews, there were 2 of them during the software development process. The first one was performed after the first certification cycle after the individual modules where finished, and the last code review was performed in the middle of the second certification cycle. The code reviews started with a general discussion of the software itself. Then each member of the team would be assigned to review a file that they were unfamiliar with. Issue tracking would done on GitLab using the Issues feature. Team members would use this to report bugs, feature requests, or request for change. Each request would include the file where the issue could be found and a description of the issue. If the issue was a bug, then the issue would also include details on how to replicate the bug.
Code review breakdowns are also located in this directory (Documentation).

Developer Usage

Assuming there are at least two tank AIs in the comp folder (two sample tanks are given in the deliverables), and the makefile is edited to load those tanks, the following instructions will run the game:

```
make
```

```
./battletanks [-map ''default.map''][-maxturn numTurns]
   [-ui (ascii|opengl)][-idle 1000]
```

API Documentation

All necessary API Documentation can be found in Tank_API_Document.pdf