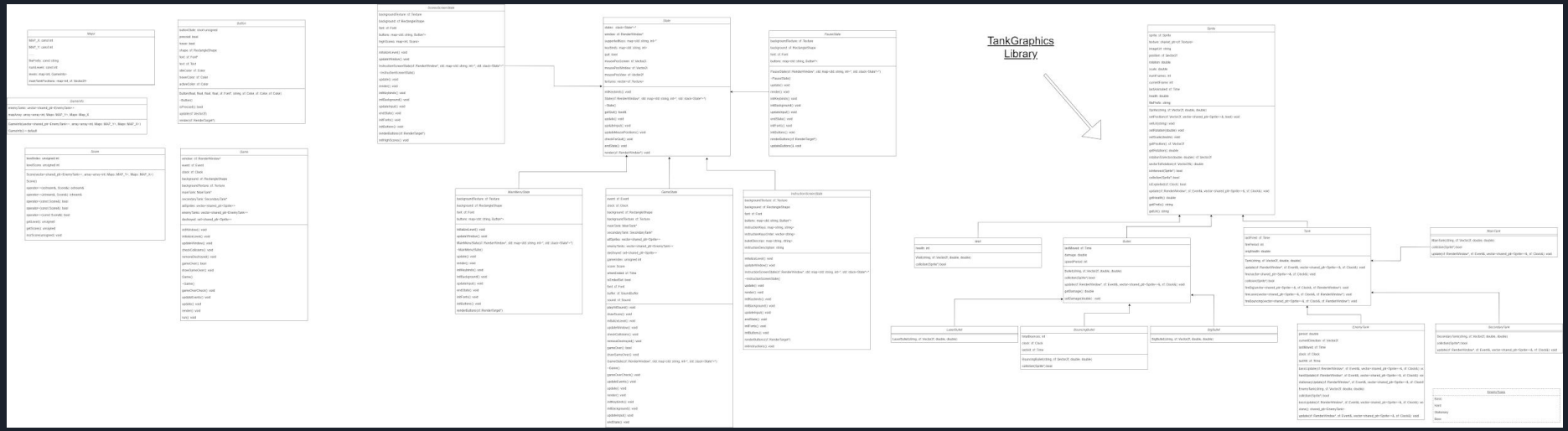
Tank Game

A game created by Yousef Helal, Zachary Iguelmamene, Larry Shields, Ben Hoang, and Qing Mei Chen

Diagram



https://drive.google.com/file/d/14VFs3_L6vCnl6uNd-MrBnJ-YYCvwkYr1/view?usp=sharing

Above is a link to the UML diagram so you can zoom in

What are you most proud of

CPU aka enemy tank

- Smart tracking and movement
- Allowed us to change the difficulty of the levels not just by adding more tanks
- Gave the game a more human component and challenge

Game Overview

After taking a few weeks to fully develop our game, we looked back on all of our work and realized that our effort as a team was what we were most proud of.

Being able to communicate and collaborate on a consistent schedule all while managing our work loads from this class as well as others really taught us what the power of cooperation could lead to.

Game Demonstration

Using:

- Windows 10, Code::Blocks 20.03
- Linux (Ubuntu 20.04.2 LTS)

Collision Detection/Sprite Storage

Sprites are stored in STL containers (depending on type)

```
vector<shared_ptr<Sprite>> allSprites;
vector<shared_ptr<EnemyTank>> enemyTanks;
set<shared_ptr<Sprite>> destroyed;
```

Collision detection by nested for loop

```
for (unsigned i = 0; i < vectorLength; i++) { //Checking for collisions between all other sprites
    for (unsigned j = i + 1; j < vectorLength; j++) {
        if ((allSprites[i] != allSprites[j]) && allSprites[i]->isIntersect(allSprites[j].get())) {
            bool iterCollision = allSprites[i]->collision(allSprites[j].get());
            bool jterCollision = allSprites[j]->collision(allSprites[i].get());
            bool isSound = false;
            if (jterCollision) {
                destroyed.insert(allSprites[j]);
                allSprites.erase(allSprites.begin() + j);
                vectorLength--;
                playHitSound();
                isSound = true;
            }
            if (iterCollision) {
                destroyed.insert(allSprites[i]);
                allSprites.erase(allSprites.begin() + i);
                vectorLength--;
                if (!isSound) {
                    playHitSound();
                }
                break;
            }
        }
    }
}
```

Maps / Levels / Sounds

Spawns enemy tank & declares what type of enemy it is as well as location (ex: basic tank, boss tank, etc)

```
#ifndef MAPS_CPP
#define MAPS_CPP
#include <iostream>
#include <memory>
#include <vector>
#include <string>
#include "Maps.h"
#include "EnemyTank.h"



vector<shared_ptr<EnemyTank>> Maps::mapOneEnemyTanks = { make_shared<EnemyTank>(EnemyTank(Maps::filePrefix + "enemyTank-basic-1.png", sf::Vector2f(1024.0f/1.8, 768.0f/1.3), 180, 0.4, 2.2, 50, Maps::filePrefix)) };
array<array<int, Maps::MAP_Y>, Maps::MAP_X> Maps::mapOne = {{{1,1,1,1,1,1,1,1,1,1,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,0,0,0,0,0,0,0,0,0,1 },
                                                             {1,1,1,1,1,1,1,1,1,1,1 } }};
```

```
    unsigned vectorLength = allSprites.size();
        //WE USE INDECIES HERE TO MAKE THE DELETE SYNTAX EASIER
        for (unsigned i = 0; i < vectorLength; i++) { //Checking for collisions between all other sprites
                for (unsigned j = i + 1; j < vectorLength; j++) {
                        if ((allSprites[i] != allSprites[j]) && allSprites[i]->isIntersect(allSprites[j].get())) {
                    bool iterCollision = allSprites[i]->collision(allSprites[j].get());
                    bool jterCollision = allSprites[j]->collision(allSprites[i].get());
                    bool isSound = false;
                                if (jterCollision) {
                                        destroyed.insert(allSprites[j]);
                                        allSprites.erase(allSprites.begin() + j);
                                        vectorLength--;
                                        playHitSound();
                                        isSound = true;
                                }
                                if (iterCollision) {
                            destroyed.insert(allSprites[i]);
                                        allSprites.erase(allSprites.begin() + i);
                            vectorLength--;
                            if (!isSound) {
                                playHitSound();
                            }
                                        break;
                                }
                        }
                }
        }
}
```

0’s = blank space
1 = location of wall

Sound being played given certain scenario

Health Bar

Simply a box of size 80.f by 15.f. Color red for enemy and green for ally. Set the health bar underneath the tanks by using the getPosition for x and y. When we get hit we update the health bar by taking a percentage of health left divided by the original and us that to set the new length of the bar 80.f*hpPercent for both ally and enemy tank.

```
sf::RectangleShape healthBar;
healthBar.setSize(sf::Vector2f(80.f, 15.f));
EnemyTank* enemyPtr = dynamic_cast<EnemyTank*>(this);
if (enemyPtr) {
    healthBar.setFillColor(sf::Color::Red);
} else {
    healthBar.setFillColor(sf::Color::Green);
}
healthBar.setPosition(getPosition().x - 40, getPosition().y + 40);

float hpPercent = getHealth() / origHealth;
healthBar.setSize(sf::Vector2f(80.f * hpPercent, healthBar.getSize().y));
window->draw(healthBar);
```

State stack

Implementing a stack of states to push and pop different parts of the game(Main Menu, GameState. new levels, etc)

```cpp
void GameState::endState()
{
    if ((gameIndex == Maps::levels.size()) && (this->allSprites.front()->getHealth() > 0)) {
        auto current = this->states->top();
        this->states->pop();
        this->states->push(new ScoresScreenState(this->window, this->supportedKeys, this->states));
        this->states->push(current);
    } else {
        auto current = this->states->top();
        this->states->pop();
        if (this->allSprites.front()->getHealth() > 0) {
            this->states->push(new GameState(this->window, this->supportedKeys, this->states, gameIndex + 1, false));
        } else {
            this->states->push(new GameState(this->window, this->supportedKeys, this->states, gameIndex, false));
        }
        this->states->push(current);
    }
    this->quit = true;
}

void GameState::updateInput()
{
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::P))
    {
        this->states->push(new PauseState(this->window, this->supportedKeys, this->states));
    } else {
        this->checkForQuit();
        this->gameOverCheck();
    }
```

```cpp
class State
{
private:

protected:
    std::stack<State*>* states; // Pointer to a stack of (stateStacks) used so eachState can access a universal state stack
```

Starting a new level

Pausing the game

Enemy Tanks/CPU

rand()

```
void EnemyTank::basicUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>&  allSprites, sf::Clock& clock) {
    firePeriod = 2000;
    double angle = (atan2((*allSprites.front()).getPosition().y - sprite.getPosition().y, (*allSprites.front()).getPosition().x - sprite.getPosition().x) * 180 / 3.14159265);
    setRotation(angle);

    float x_pos = sprite.getPosition().x;
    float y_pos = sprite.getPosition().y;
    setPosition(sf::Vector2f(x_pos + (currentDirection.x) / 10, y_pos + (currentDirection.y) / 10), allSprites, true);

    sf::Time currentTime = clock.getElapsedTime();
    if ((currentTime - lastMoved).asSeconds() > 3) {
        lastMoved = currentTime;
        currentDirection.x = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
        currentDirection.y = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
    }
    if (static_cast<int>(currentTime.asSeconds()) % 6 == 0) {
        if (x_pos > (*allSprites.front()).getPosition().x && y_pos > (*allSprites.front()).getPosition().y) {
            currentDirection.x = currentDirection.y = -1;
        }
        else if (x_pos < (*allSprites.front()).getPosition().x && y_pos >(*allSprites.front()).getPosition().y) {
            currentDirection.x = 1;
            currentDirection.y = -1;
        }
        else if (x_pos < (*allSprites.front()).getPosition().x && y_pos < (*allSprites.front()).getPosition().y) {
            currentDirection.x = currentDirection.y = 1;
        }
        else if (x_pos > (*allSprites.front()).getPosition().x && y_pos < (*allSprites.front()).getPosition().y) {
            currentDirection.x = -1;
            currentDirection.y = 1;
        }
    }

    fire(allSprites, clock, 10);
    Tank::update(window, event, allSprites, clock);
    window->draw(sprite);
}
```

Basic Enemy Tank

```
#pragma once
#include "SFML/Graphics.hpp"
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <cstdio>
#include "Sprite.h"
#include "Tank.h"
#include "Bullet.h"

using namespace std;

class EnemyTank : public Tank {
protected:
    double period;
    sf::Vector2f currentDirection;
    sf::Time lastMoved;
    sf::Time lastFire;
    sf::Time lastHit;

    void basicUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>&  allSprites, sf::Clock& clock);
    void hardUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>& allSprites, sf::Clock& clock);
    void stationaryUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>& allSprites, sf::Clock& clock);
    void bossUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>& allSprites, sf::Clock& clock);

public:
    EnemyTank(string newUrl, sf::Vector2f newPosition, double newRotation, double newScale, double newPeriod, int health, string newPref) : Tank(newUrl, newPosition, newRotation, newScale, newPref), period(newPeriod) {
    virtual void update(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>& allSprites, sf::Clock& clock) override;
    virtual bool collision(Sprite* collided) override;

    virtual shared_ptr<EnemyTank> clone() const { return make_shared<EnemyTank>(*this); }
};
```

EnemyTank is a public Tank
In the TankGraphics library

```
enum class EnemyTypes {
    Basic,
    Hard,
    Stationary,
    Boss
};

EnemyTypes getEnemyType(string url, string filePrefix) {
    if (url == filePrefix + "enemyTank-hard.png") {
        return EnemyTypes::Hard;
    }
    if (url == filePrefix + "enemyTank-stationary.png") {
        return EnemyTypes::Stationary;
    }
    if (url == filePrefix + "enemyTank-boss.png") {
        return EnemyTypes::Boss;
    }
    else {
        return EnemyTypes::Basic;
    }
}
```

Enum

```
void EnemyTank::update(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>& allSprites, sf::Clock& clock) {
    switch (getEnemyType(getUrl(), this->filePrefix))
    {
        case EnemyTypes::Basic:
        {
            this->basicUpdate(window, event, allSprites, clock);
            break;
        }
        case EnemyTypes::Hard:
        {
            this->hardUpdate(window, event, allSprites, clock);
            break;
        }
        case EnemyTypes::Stationary:
        {
            this->stationaryUpdate(window, event, allSprites, clock);
            break;
        }
        case EnemyTypes::Boss:
        {
            this->bossUpdate(window, event, allSprites, clock);
            break;
        }
    }
}
```

Update function

Detects enemy tank type from imageUrl using enum

```
void EnemyTank::hardUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>&  allSprites, sf::Clock& clock) {
    firePeriod = 1000;
    double angle = (atan2((*allSprites.front()).getPosition().y - sprite.getPosition().y, (*allSprites.front()).getPosition().x - sprite.getPosition().x) * 180 / 3.14159265);
    setRotation(angle);

    float x_pos = sprite.getPosition().x;
    float y_pos = sprite.getPosition().y;
    setPosition(sf::Vector2f(x_pos + (currentDirection.x) / 10, y_pos + (currentDirection.y) / 10), allSprites, true);

    sf::Time currentTime = clock.getElapsedTime();
    if ((currentTime - lastMoved).asSeconds() > 3) {
        lastMoved = currentTime;
        currentDirection.x = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
        currentDirection.y = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
    }
    if (static_cast<int>(currentTime.asSeconds()) < currentTime.asSeconds() && currentTime.asSeconds() < static_cast<int>(currentTime.asSeconds()) + 0.005 && static_cast<int>(currentTime.asSeconds()) % 4 == 0) {
        setPosition(sf::Vector2f(rand() % 700, rand() % 800), allSprites, true);
    }

    fire(allSprites, clock, 10);
    Tank::update(window, event, allSprites, clock);
    window->draw(sprite);
}
```

Hard Enemy Tank

```
void EnemyTank::bossUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>&  allSprites, sf::Clock& clock) {
    firePeriod = 100;
    double angle = (atan2((*allSprites.front()).getPosition().y - sprite.getPosition().y, (*allSprites.front()).getPosition().x - sprite.getPosition().x) * 180 / 3.14159265);
    setRotation(angle);

    float x_pos = sprite.getPosition().x;
    float y_pos = sprite.getPosition().y;
    setPosition(sf::Vector2f(x_pos + (currentDirection.x) / 10, y_pos + (currentDirection.y) / 10), allSprites, true);

    sf::Time currentTime = clock.getElapsedTime();
    if ((currentTime - lastMoved).asSeconds() > 1) {
        lastMoved = currentTime;
        currentDirection.x = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
        currentDirection.y = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / 4)) - 2;
    }
    if (static_cast<int>(currentTime.asSeconds()) < currentTime.asSeconds() && currentTime.asSeconds() < static_cast<int>(currentTime.asSeconds()) + 0.005 && (static_cast<int>(currentTime.asSeconds()) % 2 == 0)) {
        setPosition(sf::Vector2f(rand() % 700, rand() % 800), allSprites, true);
    }

    fire(allSprites, clock, 4);
    Tank::update(window, event, allSprites, clock);
    window->draw(sprite);
}
```

Boss Enemy Tank

```
void EnemyTank::stationaryUpdate(sf::RenderWindow* window, sf::Event& event, vector<shared_ptr<Sprite>>&  allSprites, sf::Clock& clock) {
    firePeriod = 100;
    double angle = getRotation() + 1;
    setRotation(angle);

    fire(allSprites, clock, 10);
    Tank::update(window, event, allSprites, clock);
    window->draw(sprite);
}
```

Stationary/spinning Enemy Tank

Requirements

- Polymorphism
- Namespaces
- Your own library
- Exception handling
- STL containers
- C++ 11/14/17/20 features
- Style guidelines (written and oral)

Polymorphism

UML diagrams shows that we are inheriting from multiple classes, namely the Sprite class where we do many of our positioning and movements of the tank which is then translated to other parts of the project to be used. This stopped us from writing code over and over for something that is very basic.

Namespaces

Maps.h allowed us to create maps easily by having our own namespace Maps especially when we are creating multiple levels we needed it to be easy and so Maps.h was chosen to be made a namespace. We wanted to avoid name clashes and keep maps specific to maps.

tankgraphics

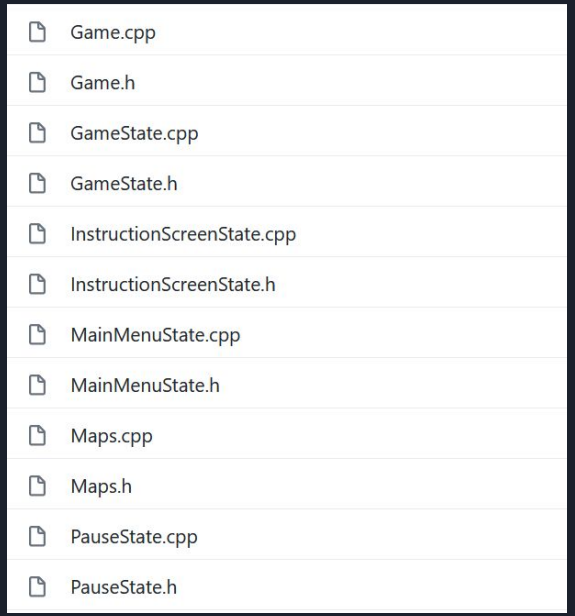
```
#ifndef MAPS_H
#define MAPS_H
#include <iostream>
#include <memory>
#include <vector>
#include <string>
#include <array>
#include "EnemyTank.h"

struct GameInfo; //Forward Decleration

namespace Maps {
    static const string filePrefix = "../data/";
    static const int numLevels = 2;

    static const int MAP_X = 20;
    static const int MAP_Y = 18;


    extern vector<shared_ptr<EnemyTank>> mapOneEnemyTanks;
    extern array<array<int, MAP_Y>, MAP_X> mapOne;

    extern vector<shared_ptr<EnemyTank>> mapTwoEnemyTanks;
    extern array<array<int, MAP_Y>, MAP_X> mapTwo;

    extern map<int, GameInfo> levels;
}


struct GameInfo {
    vector<shared_ptr<EnemyTank>> enemyTanks;
    array<array<int, Maps::MAP_Y>, Maps::MAP_X> mapArray;

    GameInfo(vector<shared_ptr<EnemyTank>> e, array<array<int, Maps::MAP_Y>, Maps::MAP_X> m) : enemyTanks(e), mapArray(m) {}
    GameInfo() = default;
};
#endif
```

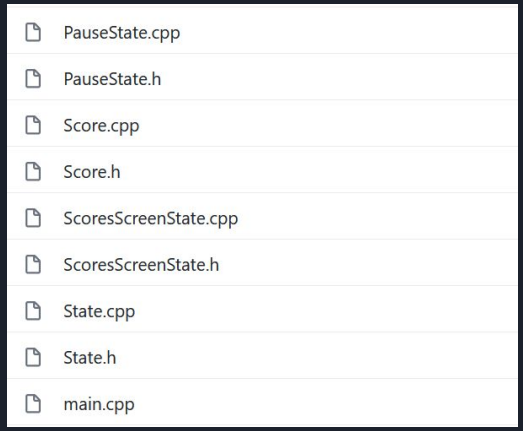
App(Game Engine)

Everything related to State/utilities we combined into the App Project.

The state class is the backbone of the game and enables it to run through its 5 different states or stages (MainMenu, Game, Pause, Score, Instructions)

This library primarily relies on the std::stack container to push or pop the game's states through Game.cpp.



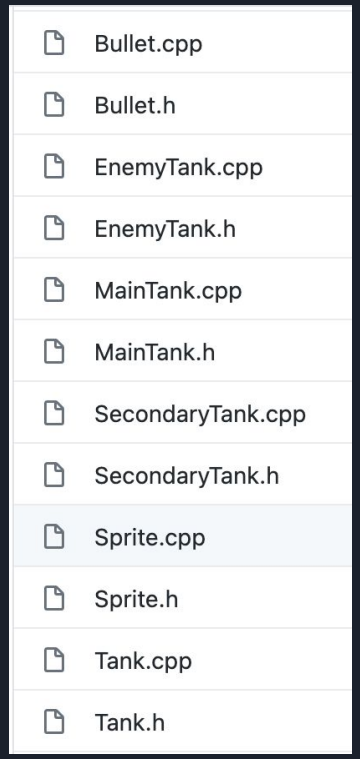


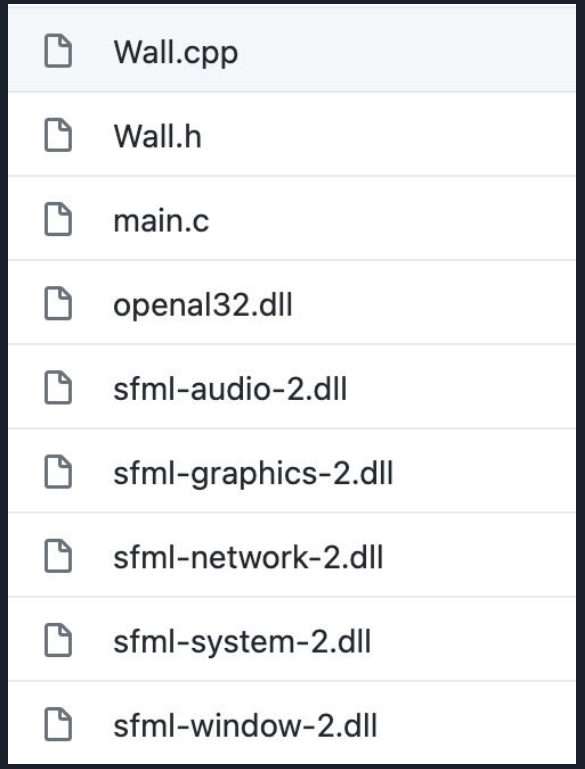
Own libraries: TankGraphics Library

All that is related to Sprite we packaged into the TankGraphics Library.

This library provides the bullet tracking, collision, maps, tanks, and enemy tanks with secondary tanks.

There is a lot that can be build with this library that heavily rely on vectors and movement.





Exception Handling

We used try catches in the score to check the score.bin file and throw an error so the program does not crash. Then there is a catch to catch the error. This allows the program to continue without the scores crashing the entire program.

```
void ScoresScreenState::initHighScores() {
    for (int i = 1; i <= Maps::numLevels; i++) {
        highScores[i] = Score(i, 0);
    }
    try {
        ifstream fin("scores.bin", ios::binary);
        if (!fin) {
           throw string("Could not open file: scores.bin");
        }
        string input;
        Score score;
        while (getline(fin, input)) {
            istringstream sout(input);
            sout >> score;
            unsigned level = score.getLevel();
            if (score > highScores[level]) {
                highScores[level] = score;
            }
        }

    } catch (string err) {
        ofstream fout("scores.bin", ios::binary);
        if (!fout) {
            cerr << "Could not open file: scores.bin" << endl;
            exit(-2);
        }
        this->initHighScores();
    }
}
```

STL Containers

- Vector
  - Used in our library to store the location of objects ie: bullets, and tanks
- Set
  - Used to hold destructed sprites(and animate them)
- Map
  - Used to create the level. This allowed us to easily create level just by changing some 0s and 1s.
- Stack
  - We used stack in the State.h to store all the states the game allows.

C++ 11/14/17/20 features

- Initializer lists for constructors

- Making use of the auto keyword when dealing with iterators

- Making use of the override keyword when dealing with inheritance and virtual functions

- Making use of shared_ptr and make_shared when initializing addresses to points in memory

- Making use of the default keyword when dealing with class definitions

- Making use of the to_string()

- Using initializer lists when creating objects of certain types(vectors, etc.)

Style Guidelines I

1. All member function definitions should be placed outside of the class in the appropriate header file. The exceptions to this, are constructor initializer lists, and “one-liners”

2. Variables and functions should make use of camelCase

3. Class and Namespace names should make use of PascalCase

4. Global variables(including constants) should not be used

5. Class member variables should be declared either private or protected. To facilitate this requirement, getters and setters should be provided if there is a need

6. Function and Class beginning braces should be on the same line as prototype and class name, respectively

7. All if statements should include opening and closing brackets(regardless of whether their content is only one line long

8. C-Style casting should not be used, and should instead be traded for static_cast, dynamic_cast, etc

Style Guidelines II

9. C header files should not be made use of, and their C++ counterparts should be used instead.

10. All data written by files should be written in binary mode

11. smart_pointer and unique_pointer should be made use of as much as possible.

12. Any image displayed on the screen using SFML, should inherit from the Sprite class, which provides a number of methods for making dealing with sf::Sprite easier

13. If a class member function, or function otherwise do not alter any data values, they should be declared as const as often as possible

14. Header files should always have include guards to protect against redefinition.

15. The auto keyword should be used when dealing with iterators, as to avoid lengthy names

16. If you initialize memory on the heap, you are responsible for cleaning it up

17. Declaring two or more classes in the same header file should only be done if one inherits from the other, and one of the classes definitions is relatively short

Style Guidelines III

18. Using this to access member variables is up to the person coding, and should depend on if the writer thinks the code will be more readable with its addition
19. Adding a file prefix to a file name should only be done using Maps::filePrefix or Sprite::filePrefix if inside a class that inherits from Sprite
20. If a particular part of a constructor is fairly long, an init…() member function should be created, and that should instead be called in the constructor.

Summary

- Originally, we had seven members on our team and had two of them drop later in the course. Due to this, we had to figure out a way to redistribute the work evenly throughout the active members.
- At the start of the project, we came up with a plan to meet at least twice a week at a specific time. Each meeting was a checkup for us team members to make sure that we were all on pace to complete our given assignments on time.
- We learned how to implement SFML into a game and how to collaborate on GitHub.
- Communication: As team members dropped or as others group members were preoccupied it is good to make sure that clear communication is made to reallocate work.
- Online resources and tutorials. People all around use SFML and so finding resources that were outside the classroom was very crucial to helping us build what we wanted and learn new way of building our game.

Summary II

- Documentation is key
  - Do it first later rather than later. As the code increases in size it becomes harder to tell what does what.
- We are most proud of our CPU development.
- The project was a good idea.
- The project was worth the time as it allowed for us to implement knowledge acquired in class in a high level application.
- Working in a group and depending on others was fun and a good learning experience.
  - We allocated the work according to skill level so everyone was challenged in some way.