CMP105 Coursework Report

MECH VS TANKS

Student Name: Arjun Bhatnagar

Student Id: 1904516

Git Username: arjun01bhatnagar

Introduction

The application made for this coursework is a top down shooter game which revolves around the user playing as a mech character who must reach the end of the carrier. While doing so the player encounters tanks which are roaming along the width of the carrier. The player must dodge the tanks or shoot them down while firing projectiles at it which defeats them. The carrier also has obstacles such as rocks, garbage piles and broken-down tanks. The user must use the controls provided to navigate themselves around these various obstacles as there is no way through them. The game application uses multiple screens such as the main menu screen which tells the user how to proceed. The next screen the user encounters is the controls screen which explains the user what the various controls to the game are. If the player wins the game the user is congratulated with a screen which tells them that they have won. If the player loses the game, they are met with an end screen which tells them they have lost and that they can restart the game. The user is provided with both keyboard and mouse input to help them navigate throughout the different screens and the playable level in the game itself. The user can see the animation of the character while moving it around and witness the tanks moving around which displays animation. There are a lot of collision responses in the game which will be discussed later in the report

The unique game mechanics in the game are the player can use keyboard controls to move around while also using the keyboard inputs to fire projectiles at the tanks which destroys them.

Controls

The Controls of the game are as follows

- W – To move up
- A – To move right
- S – To move down
- D – To move left
- Left Shift – To shoot projectiles
- B – To begin the game
- P – To pause the game
- Backspace – To go back
- ESC – To close the game
- Enter – To restart the game
- Mouse - To aim and rotate player

Game Screens

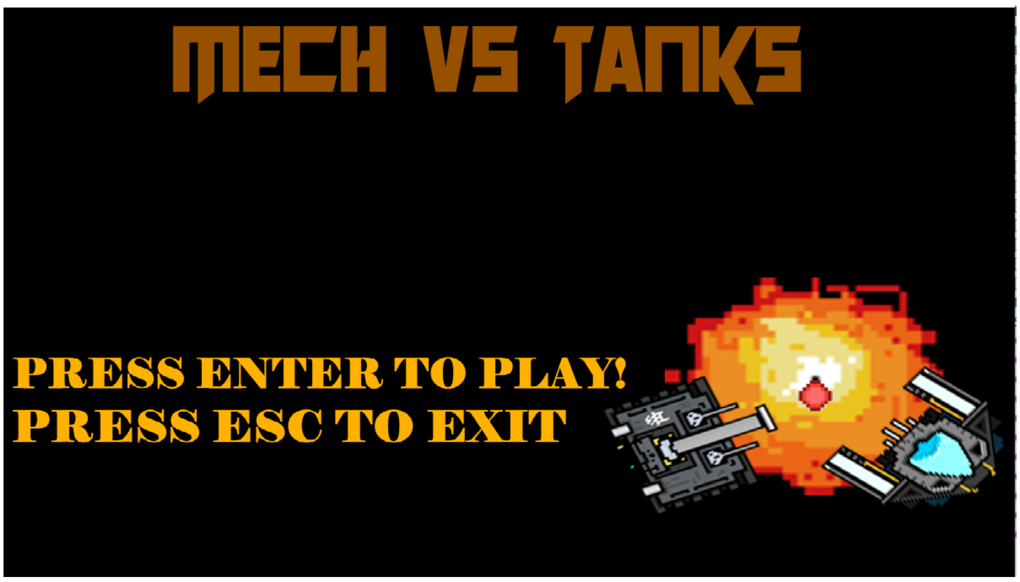
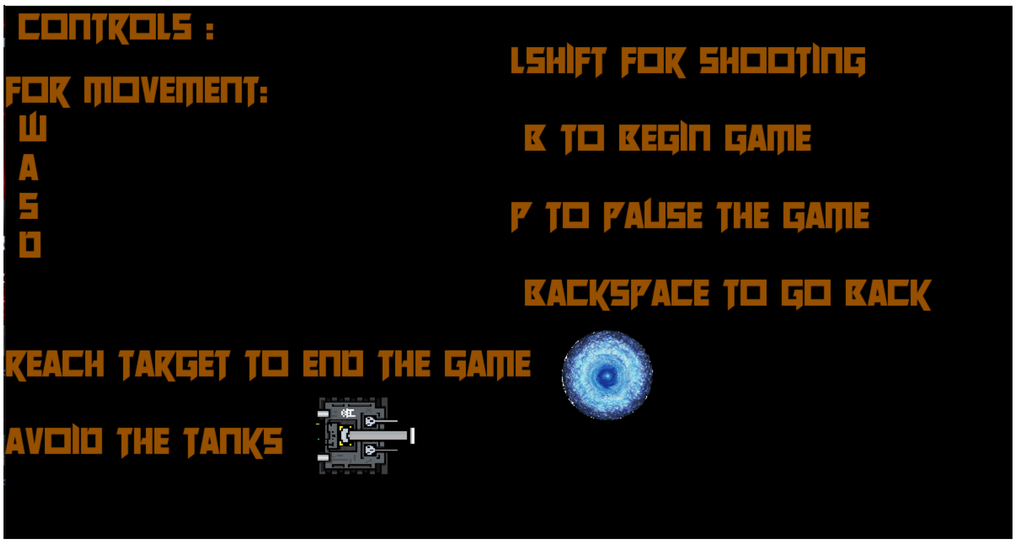


Figure 1 This is the main menu screen


CONTROLS :

FOR MOVEMENT:
W
A
S
D

LSHIFT FOR SHOOTING

B TO BEGIN GAME

P TO PAUSE THE GAME

BACKSPACE TO GO BACK

REACH TARGET TO END THE GAME

AVOID THE TANKS

Figure 2 This screen tells the user the controls to the game

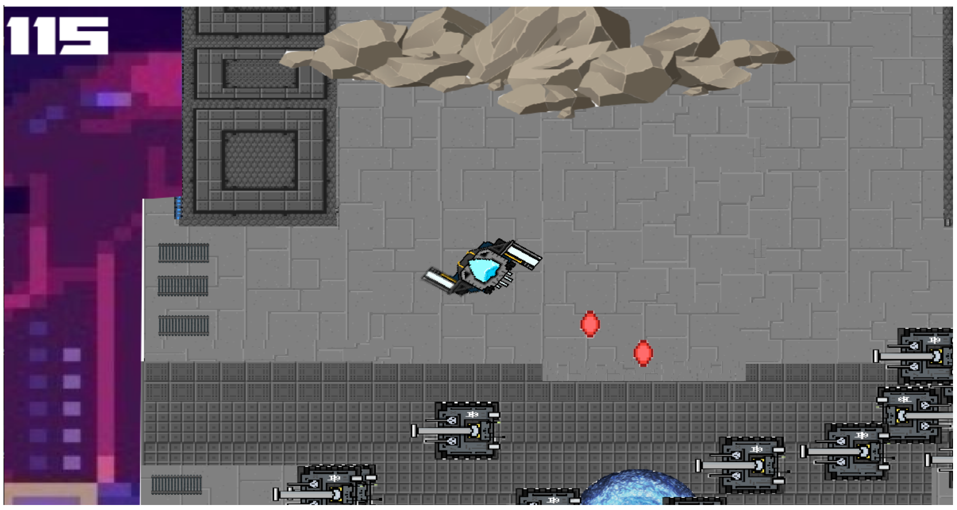
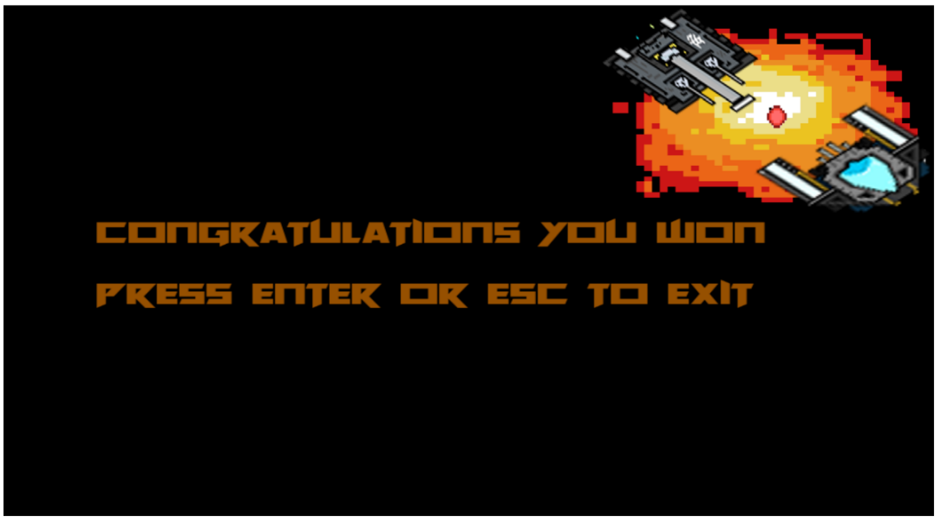


Figure 3 Level Screen


CONGRATULATIONS YOU WON

PRESS ENTER OR ESC TO EXIT

Figure 4 Screen displayed when player reaches aim and defeats the tanks

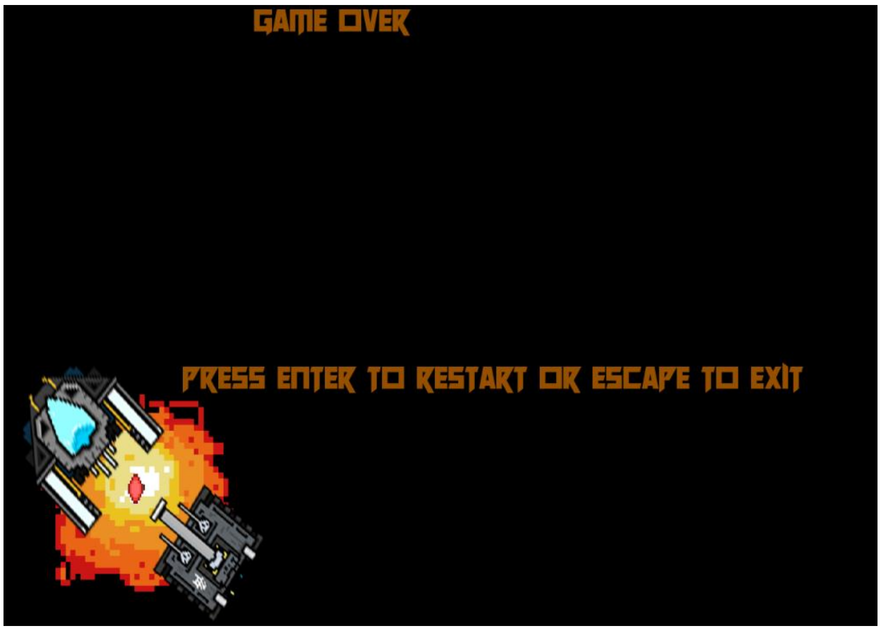


Figure 5 This screen is displayed when user loses the game

Input

The input class provided in the framework helped to build the application because it provided an easy method to implement different types of controls to the game. The different types of implementations of the input class were useful for defining the controls of the game. The functions such as isKeyDown and setKeyUp are helpful in telling the compiler what the user wants to input, and the character moves accordingly. The W A S D controls which help in the movement of the character are made using the input class. The other controls such as navigating through the menu like pressing backspace, enter and escape are all implemented using the input class. The input class also gives us functionality to implement all the before mentioned navigation controls. For example:

```
if (input->isKeyDown(sf::Keyboard::W))
{
    velocity.y = -350;
}
else if (input->isKeyDown(sf::Keyboard::A))
{
    velocity.x = -350;
}
else if (input->isKeyDown(sf::Keyboard::S))
{
    velocity.y = 350;
}
else if (input->isKeyDown(sf::Keyboard::D))
{
    velocity.x = 350;
}
```

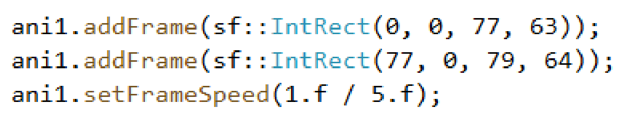
Figure 6 This figure is a code snippet from the Character class which shows how the functions from the input class are being used to implement movement

Sprite Work

The sprite work used in the application is used for at least four places in the coursework. These places are:

- The Character (Mech)
- The Opponent (Tank)
- The Projectile
- The aim

For the character of the user (Mech), the sprites were taken from an online source (mentioned later in the report). These sprites were ideal for the character usage as they displayed movement of the character easily because of their hand movement. To animate the character sprite, two frames of the character sprites were used and they were animated using the animation class. Notably the addFrame function which helps in representing a single frame and then two frames can be added to make the character seem like it's moving where in reality only the images are being changed depending upon the time set by another function from the animation class called the setFrameTime function.



```
ani1.addFrame(sf::IntRect(0, 0, 77, 63));
ani1.addFrame(sf::IntRect(77, 0, 79, 64));
ani1.setFrameSpeed(1.f / 5.f);
```

Figure 7 This figure shows how the animation is being done using the functions from animation.cpp class provided in the framework

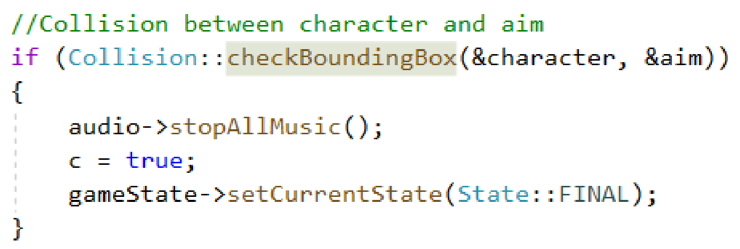
The other sprite work being is using the similar techniques for animating the tanks(NPC). The Projectile moving from the character to attack the tanks is not animated but it is user controlled which is shot by the user whenever the press LShift. There are other sprites being used as well in the game. The aim/destination for the player is made using a sprite. The different obstacles used like the broken-down tanks, the rocks and the garbage lying around are also part of the sprite work

Collision Detection and Response

The collision detection done in the coursework is:

- Between Character (Mech) and Opponent (Tank)
- Between Character and Destination
- Between Character and Obstacles
- Between Character and Walls
- Between Opponent and Projectiles

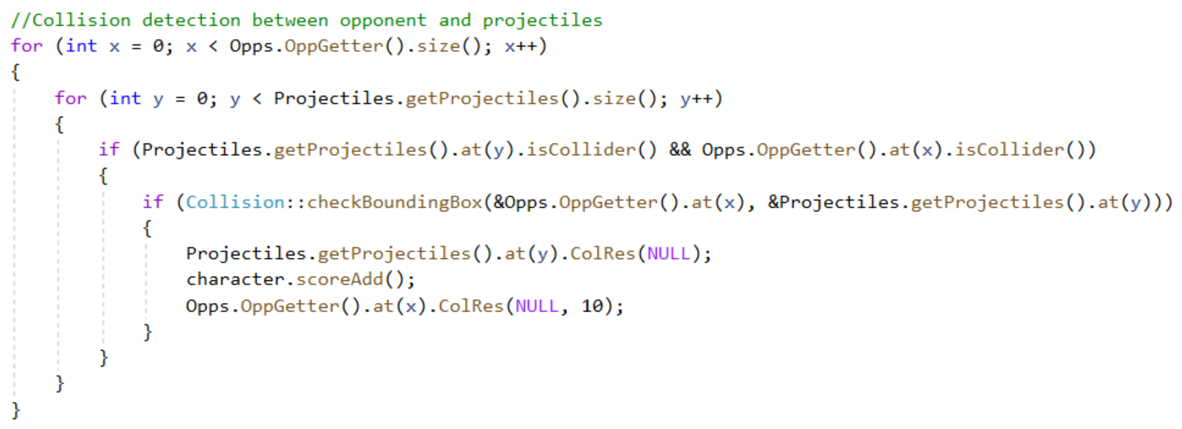
After collision occurs between the user's character and the tank, the user dies and it's game over. The collision type used is AABB collision detection. Using the Collision class provided in the framework, it helps the program to determine if the box of the character has touched the box of an enemy tank, then it plays a death sound and turns the state to a game over state where the user can restart or exit the game. Similarly if the character collides with the box of the aim set at the end of the carrier by destroying all tanks or dodging them, then the player wins the game and is greeted with a win screen from where the player can restart or exit the game. Similar is the collision detection system between the obstacles and the character. When the character collides with the obstacle, it can't move further.



```
//Collision between character and aim
if (Collision::checkBoundingBox(&character, &aim))
{
    audio->stopAllMusic();
    c = true;
    gameState->setCurrentState(State::FINAL);
}
```

Figure 8 This figure shows how the program responds if the collision occurs between the destination and the character

The character when it collides with wall results in it being stopped from moving further so the character stays within the bounds of the carrier. When the projectile is fired from the character and it hits the tank, the tank loses all it's health and it is destroyed.

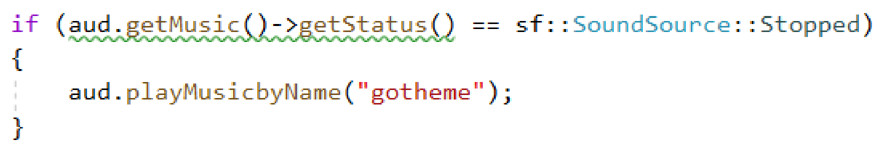


```
//Collision detection between opponent and projectiles
for (int x = 0; x < Opps.OppGetter().size(); x++)
{
    for (int y = 0; y < Projectiles.getProjectiles().size(); y++)
    {
        if (Projectiles.getProjectiles().at(y).isCollider() && Opps.OppGetter().at(x).isCollider())
        {
            if (Collision::checkBoundingBox(&Opps.OppGetter().at(x), &Projectiles.getProjectiles().at(y)))
            {
                Projectiles.getProjectiles().at(y).ColRes(NULL);
                character.scoreAdd();
                Opps.OppGetter().at(x).ColRes(NULL, 10);
            }
        }
    }
}
```

Figure 9 Collision Detection between the projectile and tank

Audio

The audio work used is the coursework is used at certain places in the game. Notably, when the level starts, the user will hear the level theme which keeps on playing throughout the whole game. The user will hear the projectile being fired from the character when they press the LShift button. If the user meets a tank, they lose the game at which point they hear a death scream and a level end music is played. If the player reaches the destination, they win the game and a level complete or a win sound is played. The audio work was implemented in the coursework with the help of the AudioManager class provided in the framework. The different functions in the class helped to implement all the audio in the coursework.



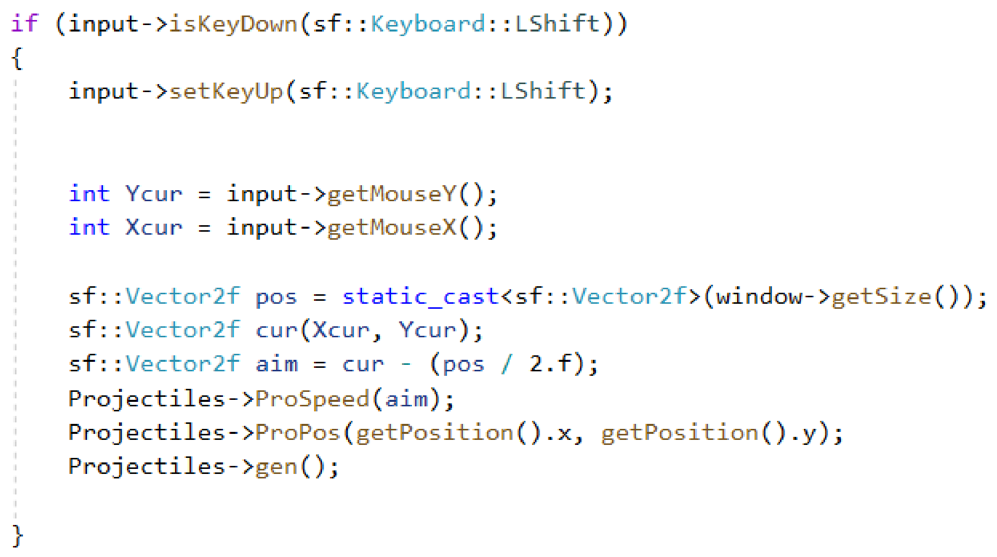
```
if (aud.getMusic()->getStatus() == sf::SoundSource::Stopped)
{
    aud.playMusicbyName("gotheme");
}
```

Figure 10 This figure shows the implementation of audiomanager class

Game Logic and Unique Mechanics

The game has a simple logic. To win the user must either defeat all the enemies and reach the aim or dodge all the enemies and reach the end. To implement this the character had to move first. To move the character the input of the keyboard, notably the W A S D keys was used. If the W key was pressed, then the character will move in the negative y direction. The similar logic was applied to the other keys as well depending on their placement on the keyboard.

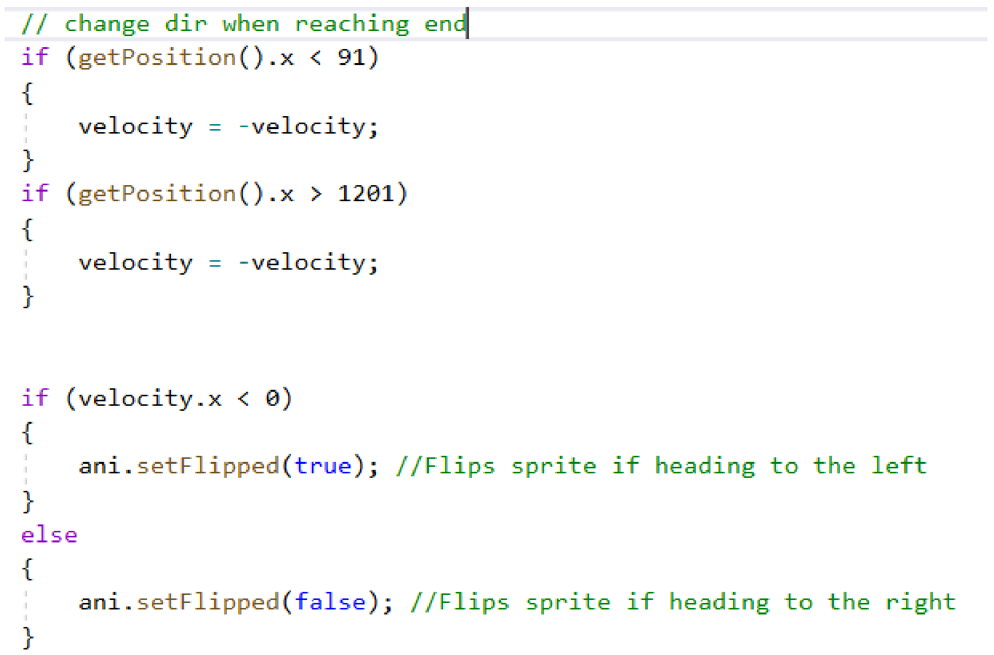
The unique mechanic used for firing a projectile was done using the mouse and the keyboard both. Whenever the LShift key was pressed, the program took an input of the position of the mouse cursor with the x and y co-ordinate. After the function had that information, it generated a projectile from the character to the cursor of the mouse which resulted in it looking like a projectile was being fired by the character.



```
if (input->isKeyDown(sf::Keyboard::LShift))
{
    input->setKeyUp(sf::Keyboard::LShift);


    int Ycur = input->getMouseY();
    int Xcur = input->getMouseX();

    sf::Vector2f pos = static_cast<sf::Vector2f>(window->getSize());
    sf::Vector2f cur(Xcur, Ycur);
    sf::Vector2f aim = cur - (pos / 2.f);
    Projectiles->ProSpeed(aim);
    Projectiles->ProPos(getPosition().x, getPosition().y);
    Projectiles->gen();

}
```

Figure 11 This figure shows the above description in code

For the movement of the tanks, their sprites were generated using a loop. Whenever they reached the end of the width of the carrier, their sprite was flipped, and the direction of their velocity was changed as well. This made it seem like they were travelling from left to right and vice versa to block the path of the user.



```
// change dir when reaching end
if (getPosition().x < 91)
{
    velocity = -velocity;
}
if (getPosition().x > 1201)
{
    velocity = -velocity;
}


if (velocity.x < 0)
{
    ani.setFlipped(true); //Flips sprite if heading to the left
}
else
{
    ani.setFlipped(false); //Flips sprite if heading to the right
}
```

Figure 12 This figure shows how the tank sprites were flipped

Conclusion

This coursework gave the perfect mix of art and maths implemented together to make a functional video game application. Using the framework taught to us in the lectures I was able to build a top down shooter game. There were many challenges while making this application. I had to familiarise myself with the framework which resulted in me finding more and more about how even the simplest games require a lot a time and effort. While making the application I didn't think that it would be such a daunting task. I was satisfied with how the game turned out but I did realise after finishing the game that there could've been lots of improvements in the application. I could've made the enemy tanks more functional by making them fire some missiles at the character. I could've made the obstacles more interactive if they dealt damage to the character instead of just blocking the path of the character. I have learnt a great deal from doing this coursework. I could've added some more information in the display regarding the progress of the user. I have learnt how to mage code and write it in a more efficient way. I have learnt how to use different functions to my advantage by exploring the framework.

References

Sounds:

Character death: char_death.ogg-

https://freesound.org/people/scorpion67890/sounds/396806/

Level Fail: level_fail.ogg-

https://freesound.org/people/davidbain/sounds/135831/

Level theme: level_theme.ogg-

https://freesound.org/people/Sirkoto51/sounds/352599/

Level_win: level_win.ogg-

https://freesound.org/people/LittleRobotSoundFactory/sounds/274180/

Projectile: projectile_hit.ogg-

https://freesound.org/people/LittleRobotSoundFactory/sounds/274180/

Sprites:

Character Sprite : https://tandinos.itch.io/top-down-mech-tank-64x64

Tank sprite : https://tandinos.itch.io/top-down-mech-tank-64x64

Broken tank : https://gta.fandom.com/wiki/Tank?file=Tank-GTA2.png

Garbage : https://www.pikpng.com/pngvi/iRbRRww_890-x-359-5-transparent-background-rubbish-png/

Explosion : https://toppng.com/show_download/244692/explosion-pixel-art/large

Rocks: https://novocom.top/view/4a515b-transparent-background-png-transparent-rock-clipart/