Technical Design Document (TDD)

Dawid Jerdonek and Filip Rusiecki

**1. Document outline**

**2. CRC cards**

**3. Architecture**

**4. Technology**

**5. Sprint 1 Report**

**6. Sprint 2 Report**

**7. Sprint 3 Report**

**Document Outline**

CRC cards

Architecture

Technology

Sprint 1 report

Sprint 2 report

Sprint 3 report

**CRC Cards**

CRC Cards for sprint 1

A picture containing table

Description automatically generated

CRC Cards for final project

**A picture containing diagram

Description automatically generated**

**Architecture**

Class diagram for sprint 1

Diagram, schematic

Description automatically generated

Class diagram for final project

Diagram

Description automatically generated

**Technology**

Research

Filip: For our technology we used Sfml and just worked of that, for physics I had a library from 1st and 2nd year that we created with math’s functions, for some of the collisions I have found a perfect pixel source code to collide objects by their texture. It can be found here [Click here for Website](https://github.com/SFML/SFML/wiki/Source:-Simple-Collision-Detection-for-SFML-2).

Dawid: I researched the joystick element of the SFML library. It is currently automatically included in the library and no longer needs a separate include. I have researched How this joystick works and how the different buttons are mapped, in particular to an Xbox controller which is what we used for our game

Installation

We used Sfml external library for our project, to set it up we followed the guide online which can be found here: [Click me](https://www.sfml-dev.org/tutorials/2.5/start-vc.php)

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The library can also be downloaded here : [Click me](https://www.sfml-dev.org/)

Technical Achievement

Dawid: The two main parts of the project that I found particularly challenging were getting the enemies to follow the player using our pathfinding algorithm and converting different pieces of code to work with a controller for our controller support. Focusing on the pathfinding algorithm it was difficult to figure out a way to move the enemies along the path set by pathfinding. Since after creating a path I had to find a way to reverse it and then allow the enemy to move from one cell to another in the correct order.

To get enemy to move towards the correct direction I needed to get the length between enemy and the next cell position. After that I needed to get a unit vector of this to calculate the needed velocity:

sf::Vector2f direction = m\_rangedEnemy.targetCellPos - m\_rangedEnemy.enemyRangerSprite.getPosition();

float distance = sqrtf(direction.x \* direction.x + direction.y \* direction.y); //Calculate the length between two points

m\_rangedEnemy.velocity = sf::Vector2f{ direction.x / distance , direction.y / distance } \*m\_rangedEnemy.speed; //Calculate unit vector

Once I had this velocity I was able to update the enemy position using this velocity:

m\_rangedEnemy.enemyRangerSprite.setPosition(m\_rangedEnemy.enemyRangerSprite.getPosition() + m\_rangedEnemy.velocity);

Filip: The main thing I found challenging is including the aStar into the project, trying to get the non passables to display as we were getting exception errors on a lot of aStar its self, I also found implementing of the physics a bit confusing, I really didn’t know what to add in the first place, but then I added gravity and bounciness to the ball object which I struggled on.

**Sprint 1 Report**

**Feature Design**

**Feature 1: Setup game and menu screens**

**Task 1: Setup Menu and gameloop**

**Set up the game loop with run, update and render function.**

void run();

void update(float delta);

void draw();

void Game::run()

{

srand(time(nullptr));

sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);

sf::Clock clock;

clock.restart();

while (m\_Window.isOpen())

{

sf::Event event;

while (m\_Window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

m\_Window.close();

}

timeSinceLastUpdate += clock.restart();

if (timeSinceLastUpdate > timePerFrame)

{

update();

draw();

timeSinceLastUpdate = sf::Time::Zero;

}

}

}

**Setting up the menu buttons as well as game states.**

void MainMenu::update(sf::Window& window, int& t\_gameMode)

{

m\_title.setPosition(660.0f, 10.0f);

playerSprite.armsSprite.setPosition(playerSprite.m\_playerSprite.getPosition());

else if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Left))

{

sf::Vector2i mouseLocation;

mouseLocation = sf::Mouse::getPosition(window);

if (mouseLocation.x > m\_leftOffset && mouseLocation.x < m\_leftOffset + m\_buttonWidth)

{

if (mouseLocation.y > m\_topOffset && mouseLocation.y &&

mouseLocation.y < m\_topOffset + m\_buttonHeight)

{

t\_gameMode = PLAY; //new game

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing &&

mouseLocation.y < m\_topOffset + m\_verticalSpacing + m\_buttonHeight)

{

t\_gameMode = CONTROLS; //controls

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing \* 2 &&

mouseLocation.y < m\_topOffset + m\_verticalSpacing \* 2 + m\_buttonHeight)

{

t\_gameMode = SETTINGS; //Settings

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing \* 3 &&

mouseLocation.y < m\_topOffset + m\_verticalSpacing \* 3 + m\_buttonHeight)

{

window.close(); //exit

}

}

}

}

Allowed The menu system to work in main loop by game states inside update and render(draw) functions

void Game::draw()

{

if (gameMode == MENU)

{

m\_Window.clear(sf::Color(0, 0, 0, 0));

m\_mainMenu.draw(m\_Window);

}

else if (gameMode == CONTROLS)

{

m\_Window.clear(sf::Color(0, 0, 0, 0));

m\_Window.draw(m\_controlsSprite);

m\_Window.draw(returnText);

}

else if (gameMode == SETTINGS)

{

m\_Window.clear(sf::Color(0, 0, 0, 0));

m\_settings.draw(m\_Window);

m\_Window.draw(settingsText);

m\_Window.draw(returnText);

}

else if (gameMode == PLAY)

{

//gameplay stuff

}

**Feature 2: Player and Crosshair Movement**

**Task 1: Render a player and crosshair sprite**

Set textures, origins, initial positions and scales

m\_crosshairSprite.setTexture(m\_crosshairTexture);

m\_crosshairSprite.setOrigin(258, 250);

m\_crosshairSprite.setPosition(m\_playerSprite.getPosition().x ,m\_playerSprite.getPosition().y + 100);

m\_crosshairSprite.scale(0.3f, 0.3f);

**Task 2: Setup system of handling input of keyboard, mouse**

Movement

if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) //Up

if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) //Down

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) //Right

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) //Left

Pickups

(sf::Keyboard::isKeyPressed(sf::Keyboard::E)) //Pick up

(sf::Keyboard::isKeyPressed(sf::Keyboard::F)) //Put down

Objectives

(sf::Keyboard::isKeyPressed(sf::Keyboard::Tab))

**Task 3: Create Player movement**

m\_playerSprite.move(0, -velocity); //Up

m\_playerSprite.move(0, velocity); //Down

m\_playerSprite.move(velocity, 0); //Right

m\_playerSprite.move(-velocity, 0); //Left

**Task 4: Create crosshair movement**

Set crosshair position to mouse position

sf::Vector2f mouseFloat = t\_window.mapPixelToCoords(mousePosition);

m\_crosshairSprite.setPosition(mouseFloat);

**Feature 3: Projectiles**

**Task 1: Get position of mouse on click**

Get the current position of mouse on screen when player clicks

sf::Vector2i mousePosition;

mousePosition = sf::Mouse::getPosition(t\_window); //Get mouse position

**Task 2: Create a line between player and mouse click position**

Draw a line from where the player was when projectile was shot, to the location where the mouse was clicked.

sf::Vertex begin{ playerPosition,sf::Color::Transparent };

m\_projectileLine.append(begin);

sf::Vertex end{ m\_currentProjectileLocation, sf::Color::Red };

m\_projectileLine.append(end);

**Task 3: Do all necessary calculations for calculation the travel distance and direction of projectile in 2 different cases**

Create two bools depending on if player shoots or sucks rubbish

bool m\_projectileShot = false;

bool m\_rubbishSucked = false;

Get distance and direction to apply velocity

m\_direction = playerPosition - crosshairPosition; // setup a direction variable in which the projectile will travel

m\_distance = sqrtf(m\_direction.x \* m\_direction.x + m\_direction.y \* m\_direction.y); //Calculate the length between two points

m\_unitVector = sf::Vector2f{ m\_direction.x / m\_distance , m\_direction.y / m\_distance }; //Calculate unit vector

m\_direction = m\_unitVector; //Set the direction properly

m\_distance = sqrtf(m\_direction.x \* m\_direction.x + m\_direction.y \* m\_direction.y); //Calculate the length between two points

m\_unitVector = sf::Vector2f{ m\_direction.x / m\_distance , m\_direction.y / m\_distance }; //Calculate unit vector

m\_direction = m\_unitVector; //Set the direction properly

Check length to know when the shot should be cleared

sf::Vector2f checkLength = m\_currentProjectileLocation - m\_crosshairPositionOnShot;

**Task 4: Shoot a projectile on left mouse click**

Set shoot bool to true when player clicks left click and no shot is active

if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Left) && m\_projectileShot == false && m\_rubbishSucked == false && m\_ammo > 0)

{… m\_projectileShot = true;…}

**Task 5: Shoot a projectile back at player on right mouse click(used for vacuuming rubbish)**

Set rubbish sucked bool to true when player clicks right click and no shot is active

if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Right) && m\_rubbishSucked == false && m\_projectileShot == false)

{… m\_rubbishSucked = true; …}

When drawing rubbish line draw from target to player instead

sf::Vertex begin{ m\_mouseLocation,sf::Color::Transparent };

m\_rubbishLine.append(begin);

sf::Vertex end{ m\_currentProjectileLocation, sf::Color::Green };

m\_rubbishLine.append(end);

**Task 6: Clear the line of projectiles depending on distance, and collision with enemy**

Clear line if target is reached

if (sqrtf(checkLength.x \* checkLength.x + checkLength.y \* checkLength.y) <= 8.0f) //get length between two points

{

m\_rubbishLine.clear();

m\_rubbishSucked = false;

}

Clear line if enemy is hit

if (myProjectile.m\_projectileLine.getBounds().intersects(m\_enemy.enemySprite.getGlobalBounds()))

{

…

myProjectile.m\_projectileLine.clear();

myProjectile.m\_projectileShot = false;

…

}

Always clear line after some time

if (timer <= 0)

{

m\_projectileLine.clear();

m\_rubbishLine.clear();

m\_projectileShot = false;

m\_rubbishSucked = false;

}

**Task 7: Add ammo system for shooting based on rubbish sucked**

Add ammo when rubbish is sucked

if (myProjectile.m\_rubbishLine.getBounds().intersects(m\_rubbish[i].m\_rubbishSprite.getGlobalBounds()) && m\_rubbish[i].rubbishExisting == true)

{

…

myProjectile.m\_ammo++;

…

}

Take away ammo when projectile is shot

m\_ammo--;

**Feature 4: Pickups and drop zones**

**Task 1:** Allow player to pickup an item on button

if (sf::Keyboard::isKeyPressed(sf::Keyboard::E))

{

for (int i = 0; i < 3; i++)

{

m\_buildParts1[i].itemUp(); //these functions make the part keep position as player

m\_buildParts2[i].itemUp(); }

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::F))

{

for (int i = 0; i < 3; i++)

{

m\_buildParts1[i].itemDown(); // these functions make the player drop the object

m\_buildParts2[i].itemDown();

}

}}

Task 2: Create working drop zones

Created 2 of these for different pickups and dropzones

for (int i = 0; i < 3; i++)

{

if (m\_buildParts2[i].partSprite.getGlobalBounds().intersects(myPlayer.m\_playerSprite.getGlobalBounds()) && m\_buildParts2[i].PartPicked == true)

{

m\_buildParts2[i].update(myPlayer);

}

if (m\_buildParts2[i].partSprite.getGlobalBounds().intersects(m\_buildable2.m\_drop\_zone.getGlobalBounds()) && m\_buildParts2[i].PartPicked != true && m\_buildable2.dropZone\_alive == true)

{

if (m\_buildParts2[i].alive == true)

{

count\_at\_dropZone2++;

m\_buildParts2[i].alive = false;

if (count\_at\_dropZone2 == 3)

{

m\_buildable2.alive = true;

m\_buildable2.dropZone\_alive = false;

m\_objHud.gen2ObjectiveDone = true;

}

}

}

Task 3: Create a new object when all pickups are dropped in correct zone.

This turns on the finished object (generator)

if (count\_at\_dropZone2 == 3)

{

m\_buildable2.alive = true;

m\_buildable2.dropZone\_alive = false;

m\_objHud.gen2ObjectiveDone = true;

}

Task 4: create multiple drop zones and pickups.

Redid Task 2 but encountered bugs.

Task 5. Created some art assets



**Feature 5: Controller support**

**Task 1: Create a check if controller is connected and disable mouse movement if true**

Check is joystick connected

sf::Joystick::isConnected(0)

Else use keyboard and mouse

else

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard:: ...)

**Task 2: Setup all necessary input handlers for player movement and actions**

Movement

(sf::Joystick::getAxisPosition(0, sf::Joystick::X) < -20)

(sf::Joystick::getAxisPosition(0, sf::Joystick::X) > 20)

(sf::Joystick::getAxisPosition(0, sf::Joystick::Y) < -20)

(sf::Joystick::getAxisPosition(0, sf::Joystick::Y) > 20)

Pickups

(sf::Joystick::isButtonPressed(0, 2)) //Pick up

(sf::Joystick::isButtonPressed(0, 1)) //Put down

Shoot

if (sf::Joystick::getAxisPosition(0, sf::Joystick::Z) < -20 && m\_projectileShot == false && m\_rubbishSucked == false && m\_ammo > 0)

Suck Rubbish

if (sf::Joystick::getAxisPosition(0, sf::Joystick::Z) > 20 && m\_rubbishSucked == false && m\_projectileShot == false)

Check objectives

(sf::Joystick::isButtonPressed(0, 3)) //View Objectives

**Task 3: Add separate movement for crosshair with the use of right controller stick**

Move Left

(sf::Joystick::getAxisPosition(0, sf::Joystick::U) < -20)

Move Right

(sf::Joystick::getAxisPosition(0, sf::Joystick::U) > 20)

Move Up

(sf::Joystick::getAxisPosition(0, sf::Joystick::V) < -20)

Move Down

(sf::Joystick::getAxisPosition(0, sf::Joystick::V) > 20)

**Task 4: Add all functionality the same as keyboard and mouse for controller**

Game Pause

Game Exit

Unpause

Etc.

**Summary of Planned Work**

|  |  |  |
| --- | --- | --- |
| **Features and Tasks** | **Time (Hours)** | **Team Member** |
| **Feature 1: Setup game and menu screens** | **15** | **Filip** |
| Task 1: Setup all files necessary for a menu and game loop. Setup necessary h and cpp files to get the game up and running | 7 | Filip |
| Task 2: Create a menu and working game loop | 4 | Filip |
| Task 3: Create working buttons and render objects on screen. | 1 | Filip |
| Task 4: After clicking play player is taken to the game scene | 2 | Filip |
| **Feature 2: Player and crosshair movement** | **9 (4+5)** | **Filip + Dawid** |
| Task 1: Render a player and crosshair sprite | 1 | Dawid |
| Task 2: Setup system of handling input of keyboard, mouse | 2+2 | Filip + Dawid |
| Task 3: Create Player movement | 2 | Filip |
| Task 4: Create crosshair movement | 2 | Dawid |
| **Feature 3: Projectiles** | **18** | **Dawid** |
| Task 1: Get position of mouse on click | 2 | Dawid |
| Task 2: Create a line between player and mouse click position | 2 | Dawid |
| Task 3: Do all necessary calculations for calculation the travel distance and direction of projectile in 2 different cases | 6 | Dawid |
| Task 4: Shoot a projectile on left mouse click | 2 | Dawid |
| Task 5: Shoot a projectile back at player on right mouse click(used for vacuuming rubbish) | 2 | Dawid |
| Task 6: Clear the line of projectiles depending on distance, and collision with enemy | 2 | Dawid |
| Task 7: Add ammo system for shooting based on rubbish sucked | 2 | Dawid |
| **Feature 4: Pickups and drop zones** | **13** | **Filip** |
| Task 1: Allow player to pickup an item on button click | 2 | Filip |
| Task 2: Create working drop zone | 3 | Filip |
| Task 3: Create a new object when all pickups are dropped in the correct zone | 3 | Filip |
| Task 4: Create multiple drop zones and pickups | 3 | Filip |
| Task 5: Create some art assets | 2 | Filip |
| **Feature 5: Controller Support** | **8** | **Dawid** |
| Task 1: Create a check if controller is connected and disable mouse movement if true | 1 | Dawid |
| Task 2: Setup all necessary input handlers for player movement and actions | 3 | Dawid |
| Task 3: Add separate movement for crosshair with the use of right controller stick | 2 | Dawid |
| Task 4: Add all functionality the same as keyboard and mouse for controller | 2 | Dawid |

**Work Completed**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features and Tasks** | **Time Actual** | **Time Estimate** | **Team Member** | **Complete** |
| **Feature 1: Setup game and menu screens** | **12** | **15** | **Filip** | **Y** |
| Task 1: Setup all files necessary for a menu and game loop. Setup necessary h and cpp files to get the game up and running | 5 | 7 | Filip | Y |
| Task 2: Create a menu and working game loop | 3 | 4 | Filip | Y |
| Task 3: Create working buttons and render objects on screen. | 2 | 1 | Filip | Y |
| Task 4: After clicking play player is taken to the game scene | 2 | 2 | Filip | Y |
| **Feature 2: Player and crosshair movement** | **9 (4+5)** | **9 (3+5)** | **Filip + Dawid** | **Y** |
| Task 1: Render a player and crosshair sprite | 1 | 1 | Dawid | Y |
| Task 2: Setup system of handling input of keyboard, mouse | 2+2 | 2+3 | Filip + Dawid | Y |
| Task 3: Create Player movement | 2 | 1 | Filip | Y |
| Task 4: Create crosshair movement | 2 | 1 | Dawid | Y |
| **Feature 3: Projectiles** | **16** | **18** | **Dawid** | **Y** |
| Task 1: Get position of mouse on click | 1 | 2 | Dawid | Y |
| Task 2: Create a line between player and mouse click position | 2 | 2 | Dawid | Y |
| Task 3: Do all necessary calculations for calculation the travel distance and direction of projectile in 2 different cases | 5 | 6 | Dawid | Y |
| Task 4: Shoot a projectile on left mouse click | 2 | 2 | Dawid | Y |
| Task 5: Shoot a projectile back at player on right mouse click(used for vacuuming rubbish) | 2 | 2 | Dawid | Y |
| Task 6: Clear the line of projectiles depending on distance, and collision with enemy | 2 | 2 | Dawid | Y |
| Task 7: Add ammo system for shooting based on rubbish sucked | 2 | 2 | Dawid | Y |
| **Feature 4: Pickups and drop zones** | **13** | **13** | **Filip** | **Y** |
| Task 1: Allow player to pickup an item on button click | 4 | 3 | Filip | Y |
| Task 2: Create working drop zone | 2 | 3 | Filip | Y |
| Task 3: Create a new object when all pickups are dropped in the correct zone | 3 | 3 | Filip | Y |
| Task 4: Create multiple drop zones and pickups | 2 | 3 | Filip | Y |
| Task 5: Create some art assets | 2 | 1 | Filip | Y |
| **Feature 5: Controller Support** | **10** | **8** | **Dawid** | **Y** |
| Task 1: Create a check if controller is connected and disable mouse movement if true | 1 | 1 | Dawid | Y |
| Task 2: Setup all necessary input handlers for player movement and actions | 4 | 3 | Dawid | Y |
| Task 3: Add separate movement for crosshair with the use of right controller stick | 2 | 2 | Dawid | Y |
| Task 4: Add all functionality the same as keyboard and mouse for controller | 3 | 2 | Dawid | Y |

**Videos (With commentary)**

[Sprint 1.mp4](https://instituteoftechnol663-my.sharepoint.com/:v:/g/personal/c00247646_itcarlow_ie/EZhEDs_Cez9KiluN4RxDRJQB5TGr_qsGVW1lbr3q9kOe2w?e=XQxyLQ)

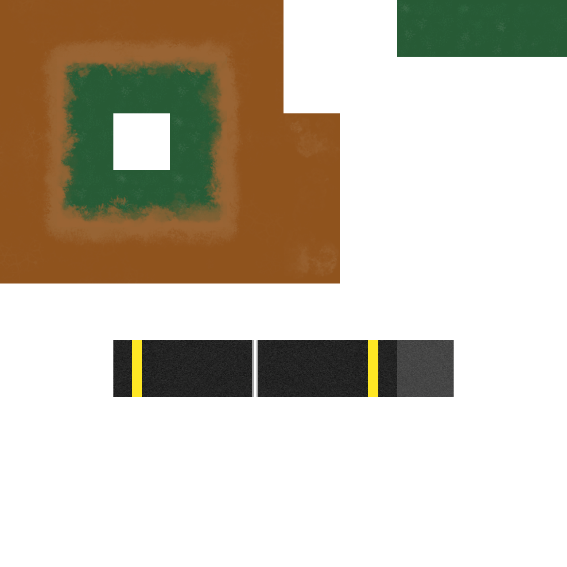
**Sprint 2 Report**

**Feature Design**

**Feature 1: Setup and add assets**

**Task 1 : Create asset for game world background**

**Made a tile set and then created world in tiled. (first version of map)**

****

**Feature 2: Create a HUD**

**Task 1: Set up a health bar minimized objectives icon and ammo count**

Health bar

m\_healthShape.setSize(sf::Vector2f(m\_health, 50));

m\_healthShape.setOutlineColor(sf::Color::Black);

m\_healthShape.setFillColor(sf::Color::Green);

m\_healthShape.setOutlineThickness(2);

in update

m\_healthShape.setPosition(armsSprite.getPosition().x - 700, armsSprite.getPosition().y + 450);

m\_healthShape.setSize(sf::Vector2f(m\_health, 50));

Minimized objectives

m\_miniPhoneSprite.setTexture(m\_phone);

m\_miniPhoneSprite.scale(0.1f, 0.1f);

m\_miniPhoneSprite.setPosition(-250, 10);

Ammo count

m\_ammoText.setFont(m\_font);

m\_ammoText.setCharacterSize(50);

m\_ammoText.setFillColor(sf::Color::Black);

m\_ammoText.setString(std::to\_string(m\_ammo));

m\_ammoText.setPosition(600,600);

m\_ammoSquare.setSize(sf::Vector2f(50, 50));

m\_ammoSquare.setOutlineColor(sf::Color::Green);

m\_ammoSquare.setFillColor(sf::Color::Cyan);

m\_ammoSquare.setOutlineThickness(1);

m\_ammoSquare.setPosition(400, 400);

**Task 2: On button click expand objectives to allow for viewing of objectives**

Check objectives

(sf::Joystick::isButtonPressed(0, 3)) //View Objectives

Keyboard

(sf::Keyboard::isKeyPressed(sf::Keyboard::Tab))

**Task 3: Update ammo if rubbish is sucked**

Update ammo when rubbish sucked

m\_ammo++;

Update ammo count on screen

m\_ammoText.setString(std::to\_string(m\_ammo));

t\_window.draw(m\_ammoText);

**Task 4: Have HUD follow the player around the map and update properly**

Update objectives position

m\_miniPhoneSprite.setPosition(t\_player.getPlayerPosition().x - 700, t\_player.getPlayerPosition().y - 300);

Update health bar position

m\_healthShape.setPosition(armsSprite.getPosition().x - 700, armsSprite.getPosition().y + 450);

m\_healthShape.setSize(sf::Vector2f(m\_health, 50));

**Feature 3: Player and Enemy Animation**

Task 1 : setup a rec for enemy and player.

void EnemyRanger::anim()

{

totalElapsed++;

if (totalElapsed > 3)

{

totalElapsed = 0;

animationframe++;

if (animationframe > 9)

{

animationframe = 0;

}

}

int col = animationframe % 9;

int row = animationframe / 1;

rectSourceSprite.height = 300;

rectSourceSprite.width = 280;

rectSourceSprite.left = col \* rectSourceSprite.width;

enemyRangerSprite.setScale(enemyRangerSprite.getScale().x, enemyRangerSprite.getScale().y);

enemyRangerSprite.setTextureRect(rectSourceSprite);

}

Task 2/3 : set up animation states for enemy and player

if (allowWalkLeft == true)

{

state = PLAYER\_WALKINGLEFT;

}

if (allowWalkRight == true)

{

state = PLAYER\_WALKINGRIGHT;

}

if (allowWalkRight == false && allowWalkLeft == false)

{

animIdle();

}

if (state == PLAYER\_WALKINGLEFT) { //when moving left

anim();

}

if (state == PLAYER\_WALKINGRIGHT) { //when moving right

anim();

}

**Task 4 : added arm to player**

sf::Vector2i mousePosition;

mousePosition = sf::Mouse::getPosition(t\_window);

armsSprite.setPosition(m\_playerSprite.getPosition());

sf::Vector2f currentArmPos = armsSprite.getPosition();

float dx = currentArmPos.x - m\_crosshairSprite.getPosition().x;

float dy = currentArmPos.y - m\_crosshairSprite.getPosition().y;

float rotationForArms = (atan2(dy, dx)) \* 180 / PI ;

armsSprite.setRotation(rotationForArms + 160);

**Feature 4: Setup camera to follow player**

**Task 1 : setup a new view**

void ResizeView(const sf::RenderWindow& window, sf::View& view, float &t\_trueHeight, float &t\_trueWidth, sf::View & extraGameView)

{

float aspectRatio = float(window.getSize().x / float(window.getSize().y));

view.setSize(VIEW\_HEIGHT \* aspectRatio, VIEW\_HEIGHT);

t\_trueHeight = (VIEW\_HEIGHT);

t\_trueWidth = (VIEW\_HEIGHT \* aspectRatio);

extraGameView.setSize(t\_trueWidth, t\_trueHeight);

extraGameView.setCenter(t\_trueWidth /2, t\_trueHeight /2);

};

**Task 2/3 : spawn player in the center of the screen and have camera look at player**

view.setCenter(sf::Vector2f(myPlayer.getPlayerPosition().x + myPlayer.m\_playerSprite.getLocalBounds().width

\* myPlayer.m\_playerSprite.getScale().x / 100.0f,

myPlayer.getPlayerPosition().y + myPlayer.m\_playerSprite.getLocalBounds().height \* myPlayer.m\_playerSprite.getScale().y / 2.0f));

m\_Window.setView(view);

**Feature 5: Add Audio**

Task 1 : Add temp and final audio

Created audio for sucking rubbish and also added menu song.

m\_sound.setBuffer(m\_buffer);

m\_sound.setLoop(true);

m\_sound.play();

m\_sound.setVolume(20.0f);

Task 2 : Run audio on certain actions

Added audio in certain actions similar to the method below

if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Right) && m\_rubbishSucked ==

suckSound.play();

}

**Feature 6: Controller mode updates**

**Task 1: Update crosshair movement with controller using player position and rotation**

Add rotation around player (This was changed to normal crosshair movement in the end)

m\_crosshairTransform.rotate(-2, m\_playerSprite.getPosition());

Move crosshair when player moves

if (sf::Joystick::getAxisPosition(0, sf::Joystick::Y) > 20)

{

m\_playerSprite.move(0, velocity);

m\_crosshairSprite.move(0, velocity);

}

**Task 2: Rotate and move arms towards crosshair and have the crosshair update correctly**

Set the arms Rotation while controller used

sf::Vector2f currentArmPos = armsSprite.getPosition();

float dx = currentArmPos.x - m\_crosshairSprite.getPosition().x;

float dy = currentArmPos.y - m\_crosshairSprite.getPosition().y;

float rotationForArms = (atan2(dy, dx)) \* 180 / PI ;

armsSprite.setRotation(rotationForArms + 160);

**Feature 7: Add controls screen**

**Task 1: Take player to correct screen when they click the button in main menu or use controller**

Menu controller support

if (sf::Joystick::isConnected(0))

{

if (sf::Joystick::isButtonPressed(0, 0)) //A

{

t\_gameMode = PLAY; //New Game

}

if (sf::Joystick::isButtonPressed(0, 2)) //X

{

t\_gameMode = CONTROLS; //Controls

}

if (sf::Joystick::isButtonPressed(0, 3)) //Y

{

t\_gameMode = SETTINGS; //Settings

}

if (sf::Joystick::isButtonPressed(0, 6)) //Back

{

window.close(); //Exit

}

}

Keyboard

else if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Left))

{

sf::Vector2i mouseLocation;

mouseLocation = sf::Mouse::getPosition(window);

if (mouseLocation.x > m\_leftOffset && mouseLocation.x < m\_leftOffset + m\_buttonWidth)

{

if (mouseLocation.y > m\_topOffset && mouseLocation.y && mouseLocation.y < m\_topOffset + m\_buttonHeight)

{

t\_gameMode = PLAY; //new game

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing && mouseLocation.y < m\_topOffset + m\_verticalSpacing + m\_buttonHeight)

{

t\_gameMode = CONTROLS; //controls

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing \* 2 && mouseLocation.y < m\_topOffset + m\_verticalSpacing \* 2 + m\_buttonHeight)

{

t\_gameMode = SETTINGS; //Settings

}

if (mouseLocation.y > m\_topOffset + m\_verticalSpacing \* 3 && mouseLocation.y < m\_topOffset + m\_verticalSpacing \* 3 + m\_buttonHeight)

{

window.close(); //exit

}

}

}

**Task 2: Change what text displays on controls screen depending on if controller is connected and add textures**

Add textures for controls on controller

Diagram, schematic

Description automatically generated

Add textures for controls on keyboard

Diagram

Description automatically generated

Change display depending on input device

if (sf::Joystick::isConnected(0))

{

returnText.setString("Press B on the Conroller to Return");

}

else

{

returnText.setString("Press ESCAPE on the Keyboard to Return");

}

settingsText.setString("Press P to Turn Music Off \nPress O to Turn Music On");

if (sf::Joystick::isConnected(0))

{

settingsText.setString("Press X to Turn Music Off \nPress Y to Turn Music On");

}

if (sf::Joystick::isConnected(0))

{

if (!m\_controlsTexture.loadFromFile("Assets/MenuAssets/Controls.png"))

{

std::cout << "error with font file file";

}

m\_controlsSprite.setTexture(m\_controlsTexture);

m\_controlsSprite.setOrigin(600, 0);

m\_controlsSprite.setScale(1.5, 1.1);

m\_controlsSprite.setPosition(1980.0f / 2.0f, 0);

}

else

{

if (!m\_controlsTexture.loadFromFile("Assets/MenuAssets/ControlsKey.png"))

{

std::cout << "error with font file file";

}

m\_controlsSprite.setTexture(m\_controlsTexture);

m\_controlsSprite.setOrigin(600, 0);

m\_controlsSprite.setScale(1.5, 1.5);

m\_controlsSprite.setPosition(1980.0f / 2.0f, 200);

}

**Summary of Planned Work**

|  |  |  |
| --- | --- | --- |
| **Features and Tasks** | **Time (Hours)** | **Team Member** |
| **Feature 1: Setup and add assets** | **13 (9 + 4)** | **Filip + Dawid** |
| Task 1: Create asset for game world background | 7 | Filip |
| Task 2: Create and add temporary assets | 2 | Dawid |
| Task 3: Apply complete and temporary assets to game | 2+2 | Filip + Dawid |
| **Feature 2: Create a HUD** | **15** | **Dawid** |
| Task 1: Set up a health bar minimized objectives icon and ammo count | 7 | Dawid |
| Task 2: On button click expand objectives to allow for viewing of objectives | 2 | Dawid |
| Task 3: Update ammo if rubbish is sucked | 2 | Dawid |
| Task 4: Have HUD follow the player around the map and update properly | 3 | Dawid |
| **Feature 3: Player and Enemy Animation** | **13** | **Filip** |
| Task 1: Set up rect for enemy and player | 4 | Filip |
| Task 2: Setup animation states for enemy and player | 3 | Filip |
| Task 3: Change animation states while player and enemy move | 4 | Filip |
| Task 4: Added arm to player | 2 | Filip |
| **Feature 4: Set up Camera to follow player** | **9** | **Filip** |
| Task 1: Set up a new view | 1 | Filip |
| Task 2: Spawn player in centre of screen and have camera look at player | 4 | Filip |
| Task 3: Have camera follow the player | 4 | Filip |
| **Feature 5: Add Audio** | **3** | **Filip** |
| Task 1: Add some temp and final audio | 1 | Filip |
| Task 2: Run audio on certain actions | 2 | Filip |
| **Feature 6: Controller mode updates** | 8 | **Dawid** |
| Task 1: Update crosshair movement with controller using player position and rotation | 5 | Dawid |
| Task 2: Rotate and move arms towards crosshair | 3 | Dawid |
| **Feature 7: Add controls screen** | 4 | **Dawid** |
| Task 1: Take player to controls screen when they click the controls button in main menu | 2 | Dawid |
| Task 2: Change what text displays on controls screen depending on if controller is connected | 2 | Dawid |

**Work Completed**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features and Tasks** | **Time Actual** | **Time Estimate** | **Team Member** | **Complete** |
| **Feature 1: Setup and add new complete and temporary art assets** | **12(8+4)** | **13 (9 + 4)** | **Filip + Dawid** | **Y** |
| Task 1: Create asset for game world background | 6 | 7 | Filip | Y |
| Task 2: Create and add temporary assets | 2 | 2 | Dawid | Y |
| Task 3: Apply complete and temporary assets to game | 2+2 | 2+2 | Filip + Dawid | Y |
| **Feature 2: Create a HUD** | **16** | **15** | **Dawid** | **Y** |
| Task 1: Set up a health bar minimized objectives icon and ammo count | 7 | 7 | Dawid | Y |
| Task 2: On button click expand objectives to allow for viewing of objectives | 2 | 2 | Dawid | Y |
| Task 3: Update ammo if rubbish is sucked | 2 | 2 | Dawid | Y |
| Task 4: Have HUD follow the player around the map and update properly | 4 | 3 | Dawid | Y |
| **Feature 3: Player and Enemy Animation** | **13** | **13** | **Filip** | **Y** |
| Task 1: Set up rect for enemy and player | 4 | 4 | Filip | Y |
| Task 2: Setup animation states for enemy and player | 4 | 3 | Filip | Y |
| Task 3: Change animation states while player and enemy move | 4 | 4 | Filip | Y |
| Task 4: Added arm to player | 1 | 2 | Filip | Y |
| **Feature 4: Set up Camera to follow player** | **9** | **9** | **Filip** | **Y** |
| Task 1: Set up a new view | 1 | 1 | Filip | Y |
| Task 2: Spawn player in centre of screen and have camera look at player | 3 | 4 | Filip | Y |
| Task 3: Have camera follow the player | 5 | 4 | Filip | Y |
| **Feature 5: Add Audio** | **2** | **3** | **Filip** | **Y** |
| Task 1: Add temp and final audio | 1 | 1 | Filip | Y |
| Task 2: Run audio on certain actions | 1 | 2 | Filip | Y |
| **Feature 6: Controller mode updates** | **9** | **8** | **Dawid** | **Y** |
| Task 1: Update crosshair movement with controller using player position and rotation | 5 | 5 | Dawid | Y |
| Task 2: Rotate and move arms towards crosshair and have the crosshair update correctly | 4 | 3 | Dawid | Y |
| **Feature 7: Add controls screen** | 4 | 4 | **Dawid + Filip** | **Y** |
| Task 1: Take player to controls screen when they click the controls button in main menu | 1 | 2 | Filip | Y |
| Task 2: Change what text displays on controls screen depending on if controller is connected + Texture | 3 | 2 | Dawid | Y |

**Videos**

[**Sprint2.mp4**](https://instituteoftechnol663-my.sharepoint.com/personal/c00247646_itcarlow_ie/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fc00247646%5Fitcarlow%5Fie%2FDocuments%2F3rd%20Year%2FProject3%5FDawidJerdonek%5FFilipRusiecki%2FSprint%202%2Emp4&parent=%2Fpersonal%2Fc00247646%5Fitcarlow%5Fie%2FDocuments%2F3rd%20Year%2FProject3%5FDawidJerdonek%5FFilipRusiecki)

**Sprint 3 Report**

**Feature Design**

**Feature 1: A Star pathfinding**

**Task 1: Grid auto-generated using neighbours algorithm**

Setup neighbours algorithm

for (int direction = 0; direction < 9; direction++) {

if (direction == 4) continue; // Skip 4, this is ourself.

int n\_row = t\_row + ((direction % 3) - 1); // Neighbor row

int n\_col = t\_col + ((direction / 3) - 1); // Neighbor column

// Check the bounds:

if (n\_row >= 0 && n\_row < m\_maxRows && n\_col >= 0 && n\_col < m\_maxCols) {

// A valid neighbor:

//std::cout << "Neighbor: " << n\_row << "," << n\_col << " - " << t\_current << std::endl;

t\_cells.at(t\_current).addNeighbour(n\_row + (n\_col \* 40));

// add the cell id

if (direction == 0 || direction == 2 || direction == 6 || direction == 8)

{

int diagId = t\_cells.at(n\_row + (n\_col \* 40)).m\_id;

t\_cells.at(t\_current).m\_diagonalList.push\_back(t\_cells.at(n\_row + (n\_col \* 40)).m\_id);

}

}

}

Have 1600 cells

for (int i = 0; i < 1600; i++) // 40 \* 40 = 1600

{

int posY = i / 40;

int posX = i % 40;

neighbours(posX, posY, m\_cellsArray, i);

}

**Task 2: AI traversing game map with A Star**

Calculate path for enemies

aStar(start, end);

int i = 0;

int index = goal->m\_id;

// std::cout << m\_cellsArray.at(index).m\_previous->m\_id << std::endl;

m\_starPathShape[i].setPosition(m\_cellsArray.at(index).m\_shape.getPosition());

if (m\_myEnemy.m\_pathFound.empty() == true)

{

m\_myEnemy.m\_pathFound.push\_back(index);

while (m\_cellsArray.at(index).m\_previous != nullptr)

{

//std::cout << m\_cellsArray.at(index).m\_previous->m\_id << std::endl;

m\_myEnemy.m\_pathFound.push\_back(m\_cellsArray.at(index).m\_previous->m\_id);

m\_starPathShape[i].setPosition(m\_cellsArray.at(index).m\_shape.getPosition());

index = m\_cellsArray.at(index).m\_previous->m\_id;

i++;

}

}

m\_myEnemy.enemyMoving = true;

Move enemies along the path

if (m\_myEnemy.enemyNextPosFound == false)

{

m\_myEnemy.targetCellPos = m\_cellsArray.at(m\_myEnemy.m\_pathFound.back()).m\_shape.getPosition();

sf::Vector2f direction = m\_myEnemy.targetCellPos - m\_myEnemy.enemySprite.getPosition();

float distance = sqrtf(direction.x \* direction.x + direction.y \* direction.y); //Calculate the length between two points

m\_myEnemy.velocity = sf::Vector2f{ direction.x / distance , direction.y / distance } \* m\_myEnemy.speed; //Calculate unit vector

m\_myEnemy.enemyNextPosFound = true;

}

m\_myEnemy.enemySprite.setPosition(m\_myEnemy.enemySprite.getPosition() + m\_myEnemy.velocity);

auto position = m\_myEnemy.enemySprite.getPosition();

if (m\_myEnemy.velocity.x < 0)

{

m\_myEnemy.enemySprite.setScale(-0.6f, 0.6f);

}

else { m\_myEnemy.enemySprite.setScale(0.6f, 0.6f); }

sf::Vector2f length = m\_myEnemy.targetCellPos - m\_myEnemy.enemySprite.getPosition();

if (sqrtf(length.x \* length.x + length.y \* length.y) <= 4)

{

m\_myEnemy.enemyNextPosFound = false;

m\_myEnemy.m\_pathFound.pop\_back();

}

if (m\_myEnemy.enemyAlive == false)

{

m\_myEnemy.enemyNextPosFound = false;

m\_myEnemy.m\_pathFound.pop\_back();

m\_myEnemy.m\_pathFound.clear();

reset();

m\_myEnemy.enemyAlive = true;

}

**Task 3: Debug mode showing impassable areas**

int h = 0;

int h = 0;

//Houses

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myhouse.m\_houseSprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

m\_notTraversal[h].setPosition((m\_cellsArray.at(j).m\_shape.getPosition()));

m\_cellsArray.at(j).m\_isPassable = false;

h++;

}

}

}

//Factories

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myFactory.m\_factorySprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{ m\_notTraversal[h].setPosition((m\_cellsArray.at(j).m\_shape.getPosition()));

m\_cellsArray.at(j).m\_isPassable = false;

h++;

}

}

}

//Puddles

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myPuddles.m\_puddleSprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

m\_cellsArray.at(j).m\_isPuddle = true;

h++;

}

}

}

//Walls

for (int i = 0; i < 24; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myWalls.m\_wallSprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

m\_cellsArray.at(j).m\_isWall = true;

h++;

}

}

}

//Generators

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_buildable.m\_drop\_zone.getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

m\_notTraversal[h].setPosition((m\_cellsArray.at(j).m\_shape.getPosition()));

m\_cellsArray.at(j).m\_isPassable = false;

h++;

}

}

**Task 4: Debug mode showing optimal A Star path**

void Grid::findPath(Cell\* start, Cell\* end, Cell\*& goal)

{

reset();

aStar(start, end);

int i = 0;

int index = goal->m\_id;

m\_starPathShape[i].setPosition(m\_cellsArray.at(index).m\_shape.getPosition());

if (m\_myEnemy.m\_pathFound.empty() == true)

{

m\_myEnemy.m\_pathFound.push\_back(index);

while (m\_cellsArray.at(index).m\_previous != nullptr)

{

m\_myEnemy.m\_pathFound.push\_back(m\_cellsArray.at(index).m\_previous->m\_id);

m\_starPathShape[i].setPosition(m\_cellsArray.at(index).m\_shape.getPosition());

index = m\_cellsArray.at(index).m\_previous->m\_id;

i++;

}

}

m\_myEnemy.enemyMoving = true;

**Task 5: Weighted areas, and index on each cell**

Create weighted areas (Puddles and walls)

if (mychild->m\_isPuddle == true)

{

weightOfArc = 1.5f;//Add child to the pq

}

if (mychild->m\_isWall == true)

{

weightOfArc = 10.0f;//Add child to the pq

}

distToChild = (weightOfArc + pq.top()->m\_pathCost);

Check if particular cell is a puddle or wall

//Puddles

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myPuddles.m\_puddleSprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

//m\_notTraversal[h].setPosition((m\_cellsArray.at(j).m\_shape.getPosition()));

//m\_notTraversal[h].setFillColor(sf::Color::Yellow);

m\_cellsArray.at(j).m\_isPuddle = true;

h++;

}

}

}

//Walls

for (int i = 0; i < 24; i++)

{

for (int j = 0; j < m\_cellsArray.size(); j++)

{

if (m\_myWalls.m\_wallSprites[i].getGlobalBounds().intersects(m\_cellsArray.at(j).m\_shape.getGlobalBounds()))

{

//m\_notTraversal[h].setPosition((m\_cellsArray.at(j).m\_shape.getPosition()));

//m\_notTraversal[h].setFillColor(sf::Color::Yellow);

m\_cellsArray.at(j).m\_isWall = true;

h++;

}

}

}

Index all cells

for (int row = 0; row < m\_maxRows; row++)

{

for (int i = 0; i < m\_maxCols; i++)

{

Cell cell(cellPositions, count);

cellPositions.x += 5120 / 40; // adjusting cell position on screen - x axis

if (cellPositions.x >= 5120)

{

cellPositions.x = 0;

cellPositions.y += 5120 / 40; // adjusting cell position on screen - y axis

}

count++;

m\_cellId[count].setFont(m\_font);

m\_cellId[count].setCharacterSize(26);

m\_cellId[count].setFillColor(sf::Color::White);

m\_cellId[count].setString(std::to\_string(count));

m\_cellId[count].setPosition(cellPositions);

//m\_starPathShape[count].setPosition(100, 100);

m\_cellsArray.push\_back(cell);// pushing back the cell

}

**Feature 2: Visual tutorial, hints and progress**

Task 1: Set up rubbish to lead to generator parts

Added position to all rubbish example below

m\_rubbish[0].randomRubbish = rand() % 13 + 1;

m\_rubbish[0].m\_rubbishSprite.setPosition(3000, 100);

m\_rubbish[1].randomRubbish = rand() % 13 + 1;

m\_rubbish[1].m\_rubbishSprite.setPosition(80, 900);

m\_rubbish[2].randomRubbish = rand() % 13 + 1;

m\_rubbish[2].m\_rubbishSprite.setPosition(300, 1700);

m\_rubbish[3].randomRubbish = rand() % 13 + 1;

m\_rubbish[3].m\_rubbishSprite.setPosition(120, 2400);

m\_rubbish[4].randomRubbish = rand() % 13 + 1;

m\_rubbish[4].m\_rubbishSprite.setPosition(400, 3200);

Task 2: Set up text at beginning of game as hints on next actions

if (displayTut == true)

{

if (allowReset == true)

{

delta = tutClock.restart().asSeconds();

allowReset = false;

}

tutorialText.setPosition(myPlayer.m\_playerSprite.getPosition().x - 450.0f, myPlayer.m\_playerSprite.getPosition().y-200.0f);

tutElapsed = tutClock.getElapsedTime();

tutorialText.setString("Use right click or left trigger to suck in rubbish and gain ammo");

if (tutElapsed.asSeconds() >= 10.0f)

{

tutorialText.setString("Use left click or right trigger to shoot rubbish at thugs");

}

if (tutElapsed.asSeconds() >= 20.0f)

{

tutorialText.setString("Follow rubbish to find Generator parts \n Place them at Generators in top right of map ");

}

if (tutElapsed.asSeconds() >= 30.0f)

{

tutorialText.setString(" ");

displayTut = false;

}

}

Task 3: Change game colour depending on objectives complete

//setting the allowence of the gamestage to change once 2 gens are complete <---------

if (m\_buildable1.alive == true && m\_buildable2.alive == true)

{

firstStageComplete = true;

map.gameStage = 1;

if (m\_houses.loadSecondStageHousesOnce == false) //setting second stage houses

{

m\_houses.secondStageHouses();

}

if (m\_factories.loadSecondStageFactoriesOnce == false) //setting second stage factories

{

m\_factories.secondStageFactories();

}

if (myPlayer.loadSecondStagePlayerOnce == false) //setting second stage player

{

myPlayer.secondStagePlayer();

}

if (m\_puddles.loadSecondStagePuddleOnce == false) //setting second stage puddle

{

m\_puddles.secondStagePuddle();

}

if (m\_enemy.loadSecondStageEnemyOnce == false) //setting second stage enemy

{

m\_enemy.secondStageEnemy();

}

if (m\_enemyRanger.loadSecondStageEnemyOnce == false) //setting second stage enemy2

{

m\_enemyRanger.secondStageEnemy();

}

if (m\_buildable1.loadSecondStageOnce == false) //setting second stage gen1

{

m\_buildable1.secondStage();

}

if (m\_buildable2.loadSecondStageOnce == false) //setting second stage gen2

{

m\_buildable2.secondStage();

}

}

if (m\_objHud.m\_objectiveComplete1 == true && m\_objHud.m\_objectiveComplete2 == true)

{

map.gameStage = 2;

if (m\_houses.loadThirdStageHousesOnce == false) //setting second stage houses

{

m\_houses.thirdStageHouses();

}

if (m\_factories.loadThirdStageFactoriesOnce == false) //setting second stage factories

{

m\_factories.thirdStageFactories();

}

if (myPlayer.loadThirdStagePlayerOnce == false) //setting second stage player

{

myPlayer.thirdStagePlayer();

}

if (m\_puddles.loadThirdStagePuddleOnce == false) //setting second stage puddle

{

m\_puddles.thirdStagePuddle();

}

if (m\_enemy.loadThirdStageEnemyOnce == false) //setting second stage enemy

{

m\_enemy.thirdStageEnemy();

}

if (m\_enemyRanger.loadThirdStageEnemyOnce == false) //setting second stage enemy2

{

m\_enemyRanger.thirdStageEnemy();

}

if (m\_buildable1.loadThirdStageOnce == false) //setting second stage gen1

{

m\_buildable1.thirdStage();

}

if (m\_buildable2.loadThirdStageOnce == false) //setting second stage gen2

{

m\_buildable2.thirdStage();

}

gamewon = true;

}

**Feature 3: Enemy functionality**

**Task 1: Have enemy one damage player when the collide**

if (Collision::PixelPerfectTest(myPlayer.m\_playerSprite, m\_enemy.enemySprite))

{

myPlayer.m\_health--;

m\_particleSystem.Initialise(myPlayer.getPlayerPosition());

}

if (myPlayer.m\_health <= 0)

{

myPlayer.m\_health = 0;

gameLost = true;

}

**Task 2: Set up projectile for enemy two**

Set up shot target as player

m\_direction = playerPosition - enemyPosition;

Shoot projectile at player

if (m\_projectileShot == false)

{

timer = 100;

playerPosition = t\_player.m\_playerSprite.getPosition();

enemyPosition = enemyRangerSprite.getPosition();

//std::cout << m\_mouseLocation.x << m\_mouseLocation.y << std::endl;

m\_direction = playerPosition - enemyPosition; // setup a direction variable in which the projectile will travel

m\_distance = sqrtf(m\_direction.x \* m\_direction.x + m\_direction.y \* m\_direction.y); //Calculate the length between two points

m\_unitVector = sf::Vector2f{ m\_direction.x / m\_distance , m\_direction.y / m\_distance }; //Calculate unit vector

m\_direction = m\_unitVector; //Set the direction properly

m\_distance = sqrtf(m\_direction.x \* m\_direction.x + m\_direction.y \* m\_direction.y); //Calculate the length between two points

m\_unitVector = sf::Vector2f{ m\_direction.x / m\_distance , m\_direction.y / m\_distance }; //Calculate unit vector

m\_direction = m\_unitVector; //Set the direction properly

m\_velocityVector = m\_direction \* m\_projectileSpeed;

m\_currentProjectileLocation = enemyPosition;

m\_projectileSprite.setPosition(enemyPosition);

m\_crosshairPositionOnShot = playerPosition;

m\_projectileShot = true;

}

if (m\_projectileShot == true)

{

sf::Vector2f checkLength = m\_currentProjectileLocation - m\_crosshairPositionOnShot; //check length

//std::cout << timer << std::endl;

m\_currentProjectileLocation += m\_velocityVector; //Add velocity to the current position of projectile tip

m\_projectileLine.clear(); //Ensure the previous shot is cleared

m\_projectileSprite.setPosition(m\_currentProjectileLocation);

//Draw the line

sf::Vertex begin{ enemyPosition,sf::Color::Transparent };

m\_projectileLine.append(begin);

sf::Vertex end{ m\_currentProjectileLocation, sf::Color::Red };

m\_projectileLine.append(end);

timer--;

//if (sqrtf(checkLength.x \* checkLength.x + checkLength.y \* checkLength.y) <= 8.0f) //get length between two points

//{

// m\_projectileLine.clear();

// m\_projectileShot = false;

//}

}

Set up delay and distance threshold for enemy shot

sf::Vector2f checkLength = m\_enemyRanger.enemyRangerSprite.getPosition() - myPlayer.m\_playerSprite.getPosition(); //check length

if (sqrtf(checkLength.x \* checkLength.x + checkLength.y \* checkLength.y) <= 700.0f) //get length between two points

{

elapsed = clock.getElapsedTime();

if (elapsed.asSeconds() >= 1.0f)

{

//Shoot Projectile

m\_enemyRanger.shoot(myPlayer);

}

if (elapsed.asSeconds() >= 2.0f)

{

clock.restart();

m\_enemyRanger.m\_projectileLine.clear();

m\_enemyRanger.m\_projectileShot = false;

m\_enemyRanger.m\_projectileSprite.setPosition(m\_enemyRanger.enemyRangerSprite.getPosition());

}

}

else

{

m\_enemyRanger.m\_projectileLine.clear();

m\_enemyRanger.m\_projectileShot = false;

m\_enemyRanger.m\_projectileSprite.setPosition(m\_enemyRanger.enemyRangerSprite.getPosition());

}

**Task 3: Have enemy two damage player when projectile hits player**

if (m\_enemyRanger.m\_projectileLine.getBounds().intersects(myPlayer.m\_playerSprite.getGlobalBounds()))

{

myPlayer.m\_health -= 30;

m\_particleSystem.Initialise(myPlayer.getPlayerPosition());

m\_enemyRanger.m\_projectileLine.clear();

m\_enemyRanger.m\_projectileShot = false;

}

**Task 4: Respawn enemies when they are killed**

if (myProjectile.m\_projectileLine.getBounds().intersects(m\_enemy.enemySprite.getGlobalBounds()))

{

…

m\_enemy.enemySprite.setPosition(900.0f, 3500.0f);

…

}

if (myProjectile.m\_projectileLine.getBounds().intersects(m\_enemyRanger.enemyRangerSprite.getGlobalBounds()))

{

…

m\_enemyRanger.enemyRangerSprite.setPosition(1200.0f, 3500.0f);

…

}

**Feature 4: Integrate Gamestates**

**Task 1: Set up all objectives**

**Task 2: Update objectives on task list**

**Task 3: Create a win state if all tasks are complete**

**Task 5: Lose state if player dies to enemies**

**Task 6: Finish Pause screen**

**Feature 5: Physics**

Task 1: Create a ball with physics

if (rubbish.getPosition().y + rubbish.getRadius() > trueHeight)

{

velocity.y \*= -0.7;

rubbish.setPosition(rubbish.getPosition().x, trueHeight - rubbish.getRadius());

}

else if (rubbish.getPosition().y - rubbish.getRadius() < 0)

{

velocity.y \*= -0.7;

rubbish.setPosition(rubbish.getPosition().x, 0 + rubbish.getRadius());

}

if (rubbish.getPosition().x + rubbish.getRadius() > trueWidth)

{

velocity.x \*= -0.7;

rubbish.setPosition(trueWidth - rubbish.getRadius(), rubbish.getPosition().y);

}

else if (rubbish.getPosition().x - rubbish.getRadius() < 0)

{

velocity.x \*= -0.7;

rubbish.setPosition(0 + rubbish.getRadius(), rubbish.getPosition().y);

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

playingStateofExtragame = drawingLine;

}

if (playingStateofExtragame == drawingLine)

{

if (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

sf::Vector2f localPosition = (sf::Vector2f)sf::Mouse::getPosition(m\_Window);

moveForce = localPosition - rubbish.getPosition();

if (thor::length(moveForce + velocity) < rubbishMaxSpeed)

{

velocity += moveForce;

}

else {

thor::setLength(moveForce, rubbishMaxSpeed);

velocity += moveForce;

}

if (thor::length(velocity) < rubbishMaxSpeed)

{

thor::setLength(velocity, rubbishMaxSpeed);

}

playingStateofExtragame = Jump;

gravity.y = 9.8 \* gravityScale;

}

}

if (playingStateofExtragame == Jump)

{

playingStateofExtragame = drawingLine;

}

velocity = velocity + (gravity \* timeSinceLastUpdate.asSeconds());

rubbish.move(velocity.x \* timeSinceLastUpdate.asSeconds(), velocity.y \* timeSinceLastUpdate.asSeconds());

if (rubbish.getPosition().y > 600 - rubbish.getRadius())

{

gravity.y -= 1.0;

velocity.y -= 1.0;

}

sf::Vector2i localPosition = sf::Mouse::getPosition(m\_Window);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(localPosition.x, localPosition.y),sf::Color::Magenta),

sf::Vertex(sf::Vector2f(rubbish.getPosition().x,

rubbish.getPosition().y),sf::Color::Transparent)

};

**Feature 6: Add Final Assets to game**

**Task 1: Update all assets to final and ensure they work**

Update all assets to final assets

Scale and reposition if needed

**Feature 7: Add collisions for player to game**

**Task 1: Create collisions method**

for (int index = 0; index < 10; index++)

{

//left sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_houses.m\_houseSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().x < m\_houses.m\_houseSprites[index].getPosition().x)

{

myPlayer.velocity = 0;

right = true;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A) || sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::S))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))

{

right = true;

}

else

{

myPlayer.velocity = 5;

right = false;

}

}

}

//right sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_houses.m\_houseSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().x > m\_houses.m\_houseSprites[index].getPosition().x + 290)

{

myPlayer.velocity = 0;

left = true;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D) || sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::S))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))

{

left = true;

}

else

{

myPlayer.velocity = 5;

left = false;

}

}

}

//top of sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_houses.m\_houseSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().y < m\_houses.m\_houseSprites[index].getPosition().y)

{

down = true;

myPlayer.velocity = 0;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::A) || sf::Keyboard::isKeyPressed(sf::Keyboard::D))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::S))

{

down = true;

}

else

{

myPlayer.velocity = 5;

down = false;

}

}

}

//bottom sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_houses.m\_houseSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().y > m\_houses.m\_houseSprites[index].getPosition().y + 300)

{

//std::cout << " bottom collision" << std::endl;

up = true;

myPlayer.velocity = 0;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::S) || sf::Keyboard::isKeyPressed(sf::Keyboard::A) || sf::Keyboard::isKeyPressed(sf::Keyboard::D))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))

{

up = true;

}

else

{

myPlayer.velocity = 5;

up = false;

}

}

}

else if (up == false && down == false && left == false && right == false) {

myPlayer.velocity = 5;

}

**Task 2: Create collisions for houses**

myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_houses.m\_houseSprites[index].getGlobalBounds())

**Task 3: Create collisions for factories**

Apply code to factories

**Task 4: Create collisions for walls**

Apply code to walls

Set up code for rotated walls

for (int index = 20; index < 24; index++)

{

//left sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_walls.m\_wallSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().x < m\_walls.m\_wallSprites[index].getPosition().x)

{

std::cout << " left collision" << std::endl;

myPlayer.velocity = 0;

right = true;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A) || sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::S))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))

{

right = true;

}

else

{

myPlayer.velocity = 5;

right = false;

}

}

}

//right sprite

if (myPlayer.m\_playerSprite.getGlobalBounds().intersects(m\_walls.m\_wallSprites[index].getGlobalBounds())

&& myPlayer.m\_playerSprite.getPosition().x > m\_walls.m\_wallSprites[index].getPosition().x + 64)

{

myPlayer.velocity = 0;

left = true;

std::cout << " right collision" << std::endl;

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D) || sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::S))

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))

{

left = true;

}

else

{

myPlayer.velocity = 5;

left = false;

}

}

}

else if (up == false && down == false && left == false && right == false) {

myPlayer.velocity = 5;

}

}

**Feature 8: Update controller support for all new features**

**Task 1: Update controller support**

Hold Y to look at objectives

if (sf::Joystick::isButtonPressed(0, 3)) //Y

{

m\_objHud.m\_viewObjectives = true;

}

Press Back to quit

if (sf::Joystick::isConnected(0))

{

if (sf::Joystick::isButtonPressed(0, 6)) //Back

{

m\_Window.close();

}

}

Pause or Play Audio in menu

if (sf::Joystick::isButtonPressed(0, 2)) //X

{

m\_settings.turnSoundOff();

m\_mainMenu.m\_sound.stop();

}

if (sf::Joystick::isButtonPressed(0, 3)) //Y

{

m\_settings.turnSoundOn();

m\_mainMenu.m\_sound.play();

}

Pause Screen

if (sf::Joystick::isConnected(0))

{

if (sf::Joystick::isButtonPressed(0, 0)) //A

{

paused = false;

}

}

**Feature 9: Add new audio**

**Task 1: Record audio clips**

**Task 2: Implement audio into game**

**Feature 10: Final Bug and Glitch Fixes**

**Task 1: Fix glitches and bugs and refine gameplay**

**Summary of Planned Work**

|  |  |  |
| --- | --- | --- |
| **Features and Tasks** | **Time (Hours)** | **Team Member** |
| **Feature 1: A Star Pathfinding** | **16(10+6)** | **Dawid + Filip** |
| Task 1: Grid auto-generated using neighbours algorithm | 4 | Dawid |
| Task 2: AI traversing game map with A Star | 4 | Dawid |
| Task 3: Debug mode showing impassable areas | 2 | Filip |
| Task 4: Debug mode showing optimal A Star path | 4 | Filip |
| Task 5: Weighted areas, and index on each cell | 2 | Dawid |
| **Feature 2: Visual tutorial, hints and progress** | **7** | **Filip** |
| Task 1: Set up rubbish to lead to generator parts | 2 | Filip |
| Task 2: Set up text at beginning of game as hints on next actions | 2 | Filip |
| Task 3: Change game colour depending on objectives complete | 3 | Filip |
| **Feature 3: Finish enemy functionality** | **5** | **Dawid** |
| Task 1: Have enemy one damage player when the collide | 1 | Dawid |
| Task 2: Set up projectile for enemy two | 2 | Dawid |
| Task 3: Have enemy two damage player when projectile hits player | 1 | Dawid |
| Task 4: Respawn enemies when they are killed | 1 | Dawid |
| **Feature 4: Integrate Gamestates** | **5** | **Dawid** |
| Task 1: Set up all objectives | 2 | Dawid |
| Task 2: Update objectives on task list | 1 | Dawid |
| Task 3: Create a win state if all tasks are complete | 1 | Dawid |
| Task 5: Lose state if player dies to enemies | 1 | Dawid |
| Task 6: Finish Pause screen | 2 | Filip |
| **Feature 5: Add Physics** | **6** | **Filip** |
| Task 1: Create a ball with physics | 5 | Filip |
| Task 2: Allow player to access mini game in certain area on button press | .5 | Filip |
| Task 3: Add a goal and win to physics mini game | .5 | Filip |
| **Feature 6: Add Final Assets to game** | 4(1+3) | **Dawid + Filip** |
| Task 1: Update all assets to final and ensure they work | 1+3 | Dawid + Filip |
| **Feature 7: Add collisions for player to game** | **9(4+5)** | **Filip + Dawid** |
| Task 1: Create collisions method | 6(4+2) | Filip + Dawid |
| Task 2: Create collisions for houses | 1 | Dawid |
| Task 3: Create collisions for factories | 1 | Dawid |
| Task 4: Create collisions for walls | 1 | Dawid |
| **Feature 8: Update controller support for all new features** | 2 | Dawid |
| Task 1: Update controller support | 2 | Dawid |
| **Feature 9: Add new audio** | **4(2+2)** | **Dawid + Filip** |
| Task 1: Record audio clips | 2(1+1) | Dawid + Filip |
| Task 2: Implement audio into game | 2(1+1) | Dawid + Filip |
| **Feature 10: Final Bug and Glitch Fixes** | **2(1+1)** | **Dawid + Filip** |
| Task 1: Fix glitches and bugs and refine gameplay | 1+1 | Dawid + Filip |

**Work Completed**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features and Tasks** | **Time Actual** | **Time Estimate** | **Team Member** | **Complete** |
| **Feature 1: A Star Pathfinding** | **17(10+5)** | **16(10+6)** | **Dawid + Filip** | **Y** |
| Task 1: Grid auto-generated using neighbours algorithm | 3 | 4 | Dawid | Y |
| Task 2: AI traversing game map with A Star | 5 | 4 | Dawid | Y |
| Task 3: Debug mode showing impassable areas | 2 | 2 | Filip | Y |
| Task 4: Debug mode showing optimal A Star path | 3 | 4 | Filip | Y |
| Task 5: Weighted areas, and index on each cell | 2 | 2 | Dawid | Y |
| **Feature 2: Visual tutorial, hints and progress** | **9** | **7** | **Filip** | **Y** |
| Task 1: Set up rubbish to lead to generator parts | 3 | 2 | Filip | Y |
| Task 2: Set up text at beginning of game as hints on next actions | 2 | 2 | Filip | Y |
| Task 3: Change game colour depending on objectives complete | 4 | 3 | Filip | Y |
| **Feature 3: Finish enemy functionality** | **6** | **5** | **Dawid** | **Y** |
| Task 1: Have enemy one damage player when the collide | 1 | 1 | Dawid | Y |
| Task 2: Set up projectile for enemy two | 3 | 2 | Dawid | Y |
| Task 3: Have enemy two damage player when projectile hits player | 1 | 1 | Dawid | Y |
| Task 4: Respawn enemies when they are killed | 1 | 1 | Dawid | Y |
| **Feature 4: Integrate Gamestates** | **8(5+3)** | **6(5+1)** | **Dawid + Filip** | **Y** |
| Task 1: Set up all objectives | 2 | 2 | Dawid | Y |
| Task 2: Update objectives on task list | 1 | 1 | Dawid | Y |
| Task 3: Create a win state if all tasks are complete | 1+1 | 1 | Dawid + Filip | Y |
| Task 5: Lose state if player dies to enemies | 1 | 1 | Dawid | Y |
| Task 6: Finish Pause screen | 2 | 1 | Filip | Y |
| **Feature 5: Add Physics** | **5** | **6** | **Filip** | **Y** |
| Task 1: Create a ball with physics | 3 | 4 | Filip | Y |
| Task 2: Allow player to access mini game in certain area on button press | 1 | 1 | Filip | Y |
| Task 3: Add a goal and win to physics mini game | 1 | 1 | Filip | Y |
| **Feature 6: Add Final Assets to game** | **4(2+2)** | **4(1+3)** | **Dawid + Filip** | **Y** |
| Task 1: Update all assets to final and ensure they work | 2+2 | 4(1+3) | Dawid + Filip | Y |
| **Feature 7: Add collisions for player to game** | **10(4+6)** | **9(4+5)** | **Filip + Dawid** | **Y** |
| Task 1: Create collisions method | 6(4+2) | 4(2+2) | Filip + Dawid | Y |
| Task 2: Create collisions for houses | 1 | 1 | Dawid | Y |
| Task 3: Create collisions for factories | 1 | 1 | Dawid | Y |
| Task 4: Create collisions for walls | 2 | 1 | Dawid | Y |
| **Feature 8: Update controller support for all new features** | **5** | **2** | **Dawid** | Y |
| Task 1: Update controller support | 5 | 2 | Dawid | Y |
| **Feature 9: Add new audio** | **1h 30min** | **4(2+2)** | **Dawid + Filip** | **N** |
| Task 1: Record audio clips | 30min Filip | 2(1+1) | Dawid + Filip | N |
| Task 2: Implement audio into game | 1 Filip | 2(1+1) | Dawid + Filip | N |
| **Feature 10: Final Bug and Glitch Fixes** | **6(4+2)** | **2(1+1)** | **Dawid + Filip** | **Y** |
| Task 1: Fix glitches and bugs and refine gameplay | 6(4+2) | 1+1 | Dawid + Filip | Y |

**Videos (With commentary)**

[**Sprint3.mp4**](https://instituteoftechnol663-my.sharepoint.com/:v:/g/personal/c00247646_itcarlow_ie/ESifa7IfGY1DkAwK7YlDxdwBKhTQAYD9jnH8WoTlyy54kA?e=bPWuRR)