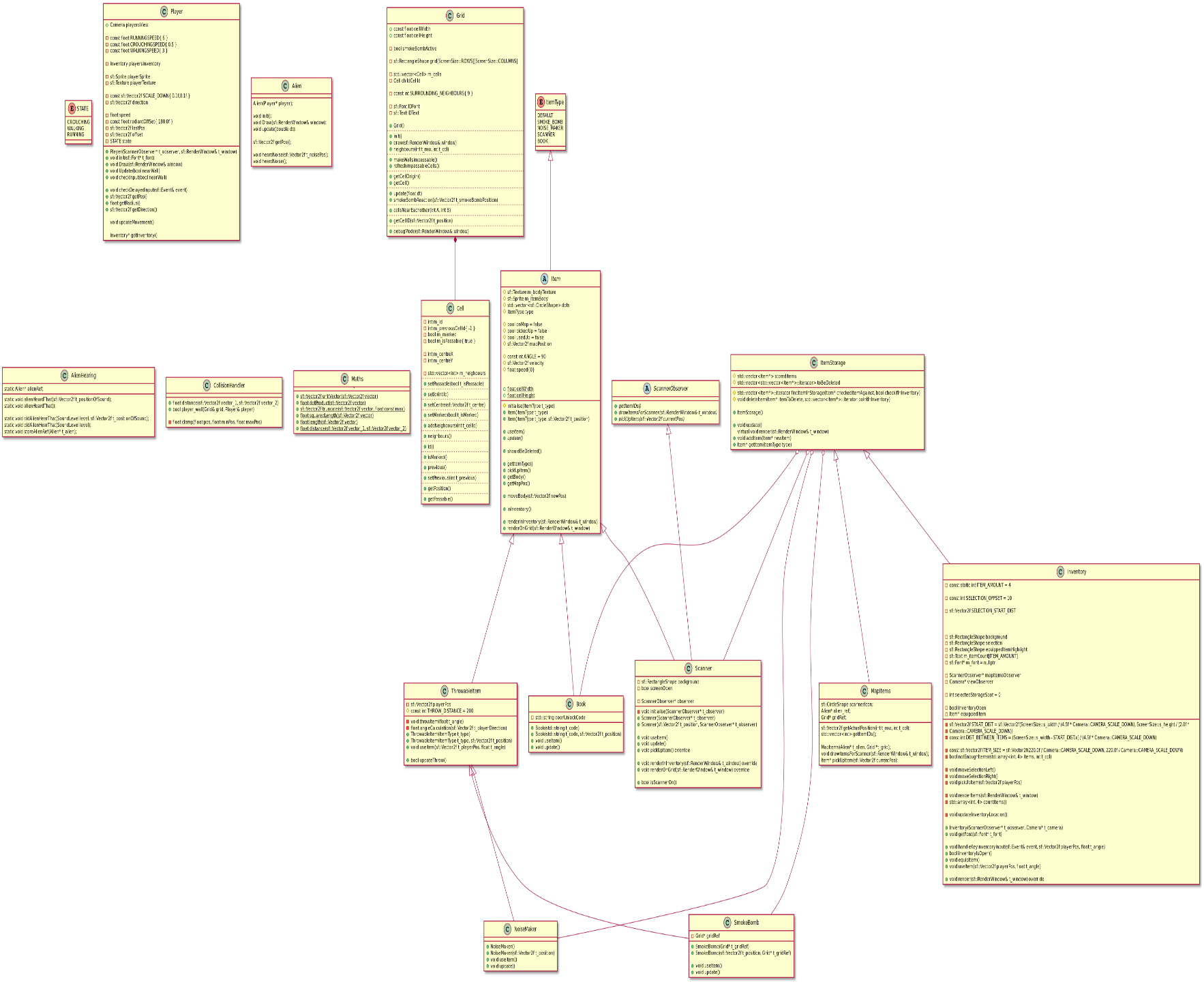
# CRC

# 

# 

# Class Diagram



**Research**

**TinyC2 was used to fine tune the collision for our game. In the second year of our course, we had experience of using TinyC2 for our previous projects and knew how to implement those functions.**

**As for YAML, we needed a way to store information and the two most popular data serialization languages are YAML and JSON. We chose YAML out of the two because, even though we have experience with both of them, we have never used JSON with SFML before.**

**Installation**

For YAML, we downloaded yaml-cpp file and a lib file from the YAML website, that contained the library needed to run YAML in our code. For example, yaml.h.

For tinyC2, we imported the header file from our previous project, allowing us to call on it quickly.

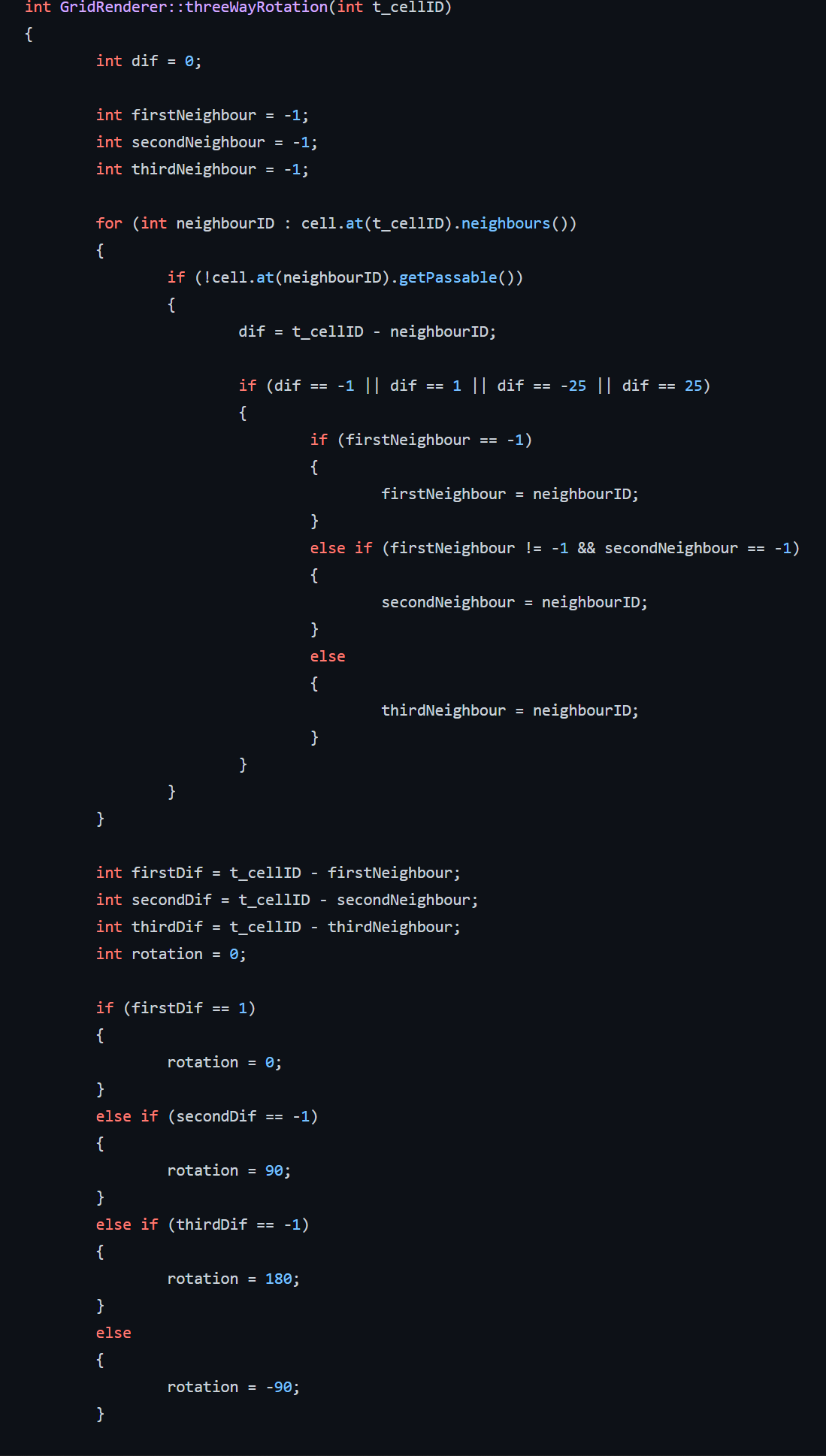
**Technical achievement**

Caroline

For me, the most challenging part of the project was the Level Editor. Not only did I have to make a new screen for it, where every click could change out the grid looked, but I also had to connect a save system to this grid. There were a lot of issues when saving and loading into the Level Editor screen, sometimes the items or walls didn’t get placed correctly for what seemed like no reason, so on and so forth. Once that was done, having to make that saved level be loaded later on as the player’s custom level caused some hiccups as well, so I would say all the code surrounding the saving and level Editor caused a lot of headaches for me. For example, below is my save changes function, which broke so many times. Items getting duplicated, getting lost, not saving, saving several players or aliens when there should just be one. I got it all to work in the end, but it took a significant chunk of time. 



Danial

The most challenging part of the project for me was the rendering of the walls using the grid. Writing a program to know which walls to use based is harder than I expected as we need to rotate the wall so that it does not look out of place. The way I solve the issue would be by getting the number of impassable neighbors or walls around the current cell and decide which wall to use based on that. Then we can use the differences between the IDs of the neighbors and the current cell to find out the rotation of the wall. For example, below is the code to find the rotation to use for the wall that looks like a T. 

# Sprint Report 1

## Feature Design

### Feature 1: Player

**Task 1 : Render a rectangle**

Make a rectangle that temporarily represents the player, and display it on the screen.

* void Player::render(sf::RenderWindow)
  + Draw a coloured rectangle on the screen.
  + Have it in the Player class.

**Task 2 : Read player input to move the rectangle**.

Read the WASD keys from the keyboard, and then move the rectangle in the direction corresponding to that key.

* void Player::handleKeyInput()
  + Use the KeyHandler class to check which keys are being pressed, and move the player appropriately.

**Task 3 : Player different movement modes.**

Have an enum class that represents three different states the player will be in - walking, running and sneaking.

* Enum class movementSpeeds  
  {  
   Idle,  
   Sneaking,

Walking,

Running  
   
}

* + Change the player's movement speed, depending on which state the enum is in.

**Task 4: Creating main game loop.**

Set up the game loop, and implement the player loop into that.

* void Game::processGameEvents()
  + Player::handleKeyInput()
  + Handle all inputs by the player, and apply them to the right classes
* void Game::update()
  + Player::update(double dt)
  + Handle updating each object in game

### Feature 2: Items

**Task 1: Item parent class**

Make a parent class that will hold most characteristics of the items in the game.

* class Item()  
  {  
   sf::Sprite;  
  }
  + Parent class will save on repeated code, and make the inventory system cleaner.

**Task 2: Make children classes**

Have several smaller classes to make up different kinds of items.

* class smokeBomb : public Item
* class scanner : public Item
* Each child class will differ slightly, for example, the smoke bomb will affect the alien, but the scanner doesn’t.

**Task 3: Picking up items**

Have the player be able to pick up an item if they are facing it, and press the left mouse button.

* Item::getPickedUp()
* Player::handleKeyInput()

**Task 4: Using items**

Have the player be able to use an item after they have picked it up.

* Player::IsKeyPressed() / Player::IsKeyHeldDown
* Item::Trigger();
* Until inventory is done, the player can only carry and use one item at a time.

### Feature 3: Inventory

**Task 1: Store several items**

Have the inventory store the items the player picks up.

* Class Inventory()  
  {  
  std::vector<Item> inventory;  
  }

**Task 2: Make it visible**

Have the player see the inventory when they hit I

* Inventory::handleKeyInput()
* Inventory::render()

**Task 3: Make inventory manager (order, stock)**

Have the inventory organised, and be able to tell the player if they have multiples of different items.

* Inventory::sort()
  + Sort could organise it better for display.

### Feature 4: Grid

**Task 1: Make a square the size of the player, and make an array of those squares.**

* The squares, or cells, will be around the size of the player

**Task 2: Have the cells know which ones will be impassable to the player / alien**

* This is so we can change the movement of those characters depending on the cell

**Task 3: Have the cells know which cells are their neighbours.**

* Good for the A\* pathfinding and level editors.

### Feature 5: Alien

**Task 1: Make the alien be able to move to a point given to it.**

* This will be the start of its patrolling system.

**Task 2: Set up the randomised patrolling points**

* The alien goes to the same 4 points, but once it reaches one point, the next one it goes to will be random.

**Task 3: Make it so the alien has a vision cone.**

* Have it so that it is able to react to when it spots the player in front of them.

**Task 4: Make the alien be able to hear.**

* No sound is implemented yet, so it will just be button presses
* Makes the alien deviate from its original path

## Summary of planned work

|  |  |  |
| --- | --- | --- |
| Feature and tasks | Time(Hours) | Team member |
| **Feature 1: Player** | 6 |  |
| **Task 1 : Render a rectangle** | 1 | Danial |
| **Task 2 : Read player input to move the rectangle**. | 1 | Danial |
| **Task 3 : Player different movement modes.** | 2 | Danial |
| **Task 4: Creating main game loop.** | 2 | Danial |
| **Feature 2: Items** | 9 |  |
| **Task 1: Item parent class** | 3 | Caroline |
| **Task 2: Make children classes** | 2 | Caroline |
| **Task 3: Picking up items** | 1 | Caroline |
| **Task 4: Using items** | 3 | Caroline |
| **Feature 3: Inventory** | 6 |  |
| **Task 1: Store several items** | 3 | Caroline |
| **Task 2: Make it visible** | 1 | Caroline |
| **Task 3: Make inventory manager (order, stock)** | 2 | Caroline |
| **Feature 4: Grid** | 5 |  |
| **Task 1: Make a square the size of the player, and make an array of those squares.** | 1 | Danial |
| **Task 2: Have the cells know which ones will be impassable to the player / alien** | 2 | Danial |
| **Task 3: Have the cells know which cells are their neighbours.** | 2 | Danial |
| **Feature 5: Alien** | 7 |  |
| **Task 1: Make the alien be able to move to a point given to it.** | 2 | Danial |
| **Task 2: Set up the randomised patrolling points** | 1 | Danial |
| **Task 3: Make it so the alien has a vision cone.** | 2 | Danial |
| **Task 4: Make the alien be able to hear** | 2 | Caroline |
| **Total** | 33 |  |

## Work Completed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature and tasks | Time Estimate | Time Actual | Team member | Complete |
| **Feature 1: Player movement** | 6 | 6.5 |  |  |
| **Task 1 : Render a rectangle** | 1 | 0.5 | Danial | YES |
| **Task 2 : Read player input to move the rectangle**. | 1 | 3 | Danial | YES |
| **Task 3 : Player different movement modes.** | 2 | 1 | Danial | YES |
| **Task 4: Creating main game loop.** | 2 | 2 | Danial | YES |
| **Feature 2: Items** | 9 | 18 |  |  |
| **Task 1: Item parent class** | 3 | 5 | Caroline | Yes |
| **Task 2: Make children classes** | 2 | 8 | Caroline | Yes |
| **Task 3: Picking up items** | 1 | 2 | Caroline | Yes |
| **Task 4: Using items** | 3 | 3 | Caroline | Yes |
| **Feature 3: Inventory** | 6 | 11 |  |  |
| **Task 1: Store several items** | 3 | 5 | Caroline | YES |
| **Task 2: Make it visible** | 1 | 2 | Caroline | YES |
| **Task 3: Make inventory manager (order, stock)** | 2 | 4 | Caroline | YES |
| **Feature 4: Grid** | 5 | 14 |  |  |
| **Task 1: Make a square the size of the player, and make an array of those squares.** | 1 | 3 | Danial | YES |
| **Task 2: Have the cells know which ones will be impassable to the player / alien** | 2 | 7 | Danial | YES |
| **Task 3: Have the cells know which cells are their neighbours.** | 2 | 4 | Danial | YES |
| **Feature 5: Alien** | 7 | 14 |  |  |
| **Task 1: Make the alien be able to move to a point given to it.** | 2 | 4 | Danial | YES |
| **Task 2: Set up the randomised patrolling points** | 1 | 2 | Danial | YES |
| **Task 3: Make it so the alien has a vision cone.** | 2 | 4 | Danial | YES |
| **Task 4: Make the alien be able to hear** | 2 | 4 | Caroline | YES |
|  | 33 |  |  |  |

# Sprint Report 2

### Player Mission

**Task 1: Display mission**

Show the text of the display on the top right (temporarily, it will go into the main map later)

* MissionSystem::render()

**Task 2: Mission order**

Organise the mission text into an order to be displayed to the player one after another.

* MissionSystem::std::array<string> missions;
  + This is also where we decide what the missions will be.

**Task 3: Update mission**

This will detect when to change the text in the top right to a new objective.

* MissionSystem::update(double dt);

**Task 4: Complete all tasks**

Have the code be able to tell when all tasks are complete, and be ready for when the win screen is coded.

* MissionSystem::end()
* For now, display some text saying you win.

### UI ( Menu Only )

**Task 1: Game states**

Set up the different game states

* Have different game states

**Task 2: Menu layout**

Have a very basic looking menu with buttons you can mouse over.

* The menu should have 3 different buttons ( Play , Level Editor , Exit )

**Task 3: Going to game through buttons**

Being able to click on the buttons, to go to different game states

* The button should show the different condition if the player hover over the button or not
* The button should show the different condition if the player click on it
* The game states should only change on mouse button release not click and if the player was in the button they click on, if the player click and move out of the button , nothing happen

### Level Editor

**Task 1: Read from a yaml file**

Make a class that can read and load from a file and pass it to game:

* Read from a file

**Task 2: Have information used by existing classes**

Example: Player and alien know where they should start, where the wall and items be on the grid

* Initialize for each class
* Initiate the levelLoader, then passing the information to the other classes

**Task 3: Allow player to edit / create a level of their own in level Editor**

Collab the grid class with the levelEditor

* When player goes to levelEditor , they will see a small version of the grid and the menu for all the items on the left side
* If a player clicks on a cell which is passable , it become impassable and vice versa
* Allow player to put down player character , alien and items

**Task 4: Save the changes the player made into a file**

Saving the new level into a yaml file that can be load in

* Once player leave the player the levelEditor , an option will pop up asking if the player would want to save the changes they made or just leave and the level will be deleted.
* The position of the items and characters like the player and Alien will also be saved into a yaml file after player click on the save option
* Let the player load their level in the main game and play with it

### Animation

**Task 1: create a struct frame**

The struct would have the rect of the sprite sheet and the duration of the animation:

* The struct should be in the same header as the animation as only the animation class use it

**Task 2: Create an animation class**

The animation class would handle all things related to animation

* Have an addFramefunction to add the rect to the list of sprites for the animation
* The duration will increase if more frame is added

**Task 3: Update to loop through the list of sprites**

Loop through the list to create an animation

* If the animation has a loop , the update will always loop before the last frame so that the animation will keep on looping
* If the animation does not have a loop , the animation will be done once and it should stop at the last frame

### Grid Renderer

**Task 1: create a Grid renderer class**

The class would handle the rendering of the level:

* This include the walls and ground , basically the layout of the level

**Task 2: Create a function that know which walls to use for the current cell**

The function should know the following:

* Check how many impassable neighbors the current cell has
* The number of the impassable neighbors around the current cell will indicate which wall is used there

**Task 3: Create a function to know which way should the wall be facing**

The function should take in the id of the current cells and the ids of the neighbor cells

* The differences between the id should indicate the way the wall should be facing
* This depends on what type of wall is being placed

**Task 4:Edit some of the walls so that it looks like its connected to each other**

The walls will require some editing to make it look smooth

### Pathfinding

**Task 1: Write the arcs and nodes into a text file**

This should be done automatically every time the game is run:

* This is so it works even if the layout of the level is changed

**Task 2: The aStar should have a starting cell and ending cell**

The starting cell should be the cell the alien is currently on and the ending cell should be the target cell:

* Only call the pathfinding once when the target cell is reached

**Task 3: Create a debug mode that shows the path the alien will take**

We can draw a line from the starting cell to the cell the alien is going for

**Task 4: The pathfinder should act dynamically to the level**

The player can throw bomb which might require the alien to take on another path.

* Call the pathfinding again incase an event like this happen.
* If the level change , the alien will call pathfinding again

## Summary of planned work

|  |  |  |
| --- | --- | --- |
| Feature and tasks | Time(Hours) | Team member |
| **Feature 1: Player Mission** | 6 |  |
| **Task 1: Display mission** | 1 | Caroline |
| **Task 2: Mission order** | 1 | Caroline |
| **Task 3: Update mission** | 2 | Caroline |
| **Task 4: Complete all tasks** | 2 | Caroline |
| **Feature 2: Menu** | 4 |  |
| **Task 1: Game states** | 1 | Danial |
| **Task 2: Menu layout** | 1 | Danial |
| **Task 3: Going to game through buttons** | 2 | Danial |
| **Feature 3: LevelEditor** | 11 |  |
| **Task 1: Read from a yaml file** | 2 | Caroline |
| **Task 2: Have information used by existing classes** | 3 | Caroline |
| **Task 3: Allow player to edit / create a level of their own in levelEditor** | 3 | Caroline |
| **Task 4: Save the changes the player made into a file** | 3 | Caroline |
| **Feature 4: Animation** | 6 |  |
| **Task 1: create a struct frame** | 1 | Danial |
| **Task 2: Create an animation class** | 3 | Danial |
| **Task 3: Update to loop through the list of sprites** | 2 | Danial |
| **Feature 5: Grid Renderer** | 11 |  |
| **Task 1: create a Grid renderer class** | 1 | Danial |
| **Task 2: Create a function that know which walls to use for the current cell** | 3 | Danial |
| **Task 3: Create a function to know which way should the wall be facing** | 3 | Danial |
| **Task 3: Edit some of the walls so that it looks like its connected to each other** | 1 | Danial |
| **Feature 4: Pathfinding** | 6 |  |
| **Task 1: Write the arcs and nodes into a text file**  **The pathfinder should act dynamically to the level** | 2 | Danial |
| **Task 2: The aStar should have a starting cell and ending cell** | 4 | Danial and Caroline |
| **Task 3: Create a debug mode that shows the path the al**  **ien will take** | 2 | Danial |
| **Task 3: The pathfinder should act dynamically to the level** | 4 | Caroline |

## Work Completed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature and tasks | Time Estimate | Time Actual | Team member | Complete |
| **Feature 1: Player Mission** | 6 | - |  |  |
| **Task 1: Display mission** | 1 | - | Caroline | NO |
| **Task 2: Mission order** | 1 | - | Caroline | NO |
| **Task 3: Update mission** | 2 | - | Caroline | NO |
| **Task 4: Complete all tasks** | 2 | - | Caroline | NO |
| **Feature 2: Menu** | 4 | 7 |  |  |
| **Task 1: Game states** | 1 | 1 | Danial | YES |
| **Task 2: Menu layout** | 1 | 2 | Danial | YES |
| **Task 3: Going to game through buttons** | 2 | 4 | Danial | YES |
| **Feature 3: LevelEditor** | 11 | 16 |  |  |
| **Task 1: Read from a yaml file** | 2 | 2 | Caroline | YES |
| **Task 2: Have information used by existing classes** | 3 | 4 | Caroline | YES |
| **Task 3: Allow player to edit / create a level of their own in levelEditor** | 3 | 5 | Caroline | YES |
| **Task 4: Save the changes the player made into a file** | 3 | 5 | Caroline | YES |
| **Feature 4: Animation** | 6 | 8 |  |  |
| **Task 1: create a struct frame** | 1 | 1 | Danial | YES |
| **Task 2: Create an animation class** | 3 | 3 | Danial | YES |
| **Task 3: Update to loop through the list of sprites** | 2 | 4 | Danial | YES |
| **Feature 5: Grid Renderer** | 8 | 11 |  |  |
| **Task 1: create a Grid renderer class** | 1 | 1 | Danial | YES |
| **Task 2: Create a function that know which walls to use for the current cell** | 3 | 4 | Danial | YES |
| **Task 3: Create a function to know which way should the wall be facing** | 3 | 5 | Danial | YES |
| **Task 3: Edit some of the walls so that it looks like its connected to each other** | 1 | 1 | Danial | YES |
| **Feature 4: Pathfinding** | 10 | 8 |  |  |
| **Task 1: Write the arcs and nodes into a text file**  **The pathfinder should act dynamically to the level** | 2 | 2 | Danial | YES |
| **Task 2: The aStar should have a starting cell and ending cell** | 4 | 3 | Danial and Caroline | YES |
| **Task 3: Create a debug mode that shows the path the alien will take** | 2 | 2 | Danial | YES |
| **Task 3: The pathfinder should act dynamically to the level** | 4 | 1 | Caroline | YES |