Week 1

We created a table of user requirements - actions that the user can perform. These requirements varied in priority from shall include (required) to should include (optional but would be beneficial if included).

User Requirements

	ID	Description	Priority
1	UR_FOUR_ROOMS	There must be at least 4 types of rooms in the station	Shall
2	UR_TELEPORTATION	Auber can teleport to and from teleportation pads in the station	Shall
3	UR_SPECIAL_ABILITIES	There must be at least 3 distinct special abilities within the group of infiltrators	Shall
4	UR_HEAL	Auber can teleport to the infirmary to heal	Shall
5	UR_REAL_TIME	The game must be real-time (not turn-based)	Shall
6*	UR_KEYBOARD	User can move Auber using keyboard	Shall
7	UR_VIOLENCE	Little/no graphic violence	Should
8	UR_SOUND	Minimal sound	Should
9	UR_JAVA	Written in Java	Shall

The table for user requirements has 3 columns - an ID, a brief description, and it's priority. It's priority determines the importance of each requirement manifesting in the game, with shall being the highest priority and should the lowest priority.

We also started working on the tables for the remaining requirements. This means that we were slightly ahead of our plan at this point as we had finished our user requirements.

Non Functional Requirements

	ID	Description	User Requirements	Fit Criteria
1	NFR_MOVEMENT_RESPONSE	The system is highly responsive to user keyboard inputs	UR_KEYBOARD	<0.1 seconds
2	NFR_SECURE	The system is secure and will store no data about the user		
3	NFR_OFFLINE_SECURITY	The game can be run when the user does not have internet connection		
4	NFR_HEARING	The game can be played by someone who is deaf	UR_SOUND	

Week 2

In week 2 we completed the tables for all the remaining requirements. Therefore we are on track with our plan.

Week 3

In week 3 we documented the use cases. Our use cases consisted of 5 sections: Actor, precondition, trigger, scenario and postcondition. The primary and secondary actors are the characters / in-game features that the use case relates to. Preconditions are the action involving the primary and secondary actors. The trigger is the action that occurs that starts the use case. The main success scenario is where the actor and system interact successfully, and the secondary scenario is what occurs if this interaction is unsuccessful. The success postcondition is the successful output of the use case.

Use Cases

Case: Arrest Infiltrator

Actors:

- Primary Actor: Auber
- Secondary Actor: Infiltrator

Precondition: both actors are close to each other

Trigger: user inputs arrest key

Main Success Scenario:

- the user approaches the infiltrator and holds the arrest key.
- Infiltrator stays in range whilst arrest is taking place.
- Infiltrator teleports to the brig when arrest is complete

Secondary Scenarios: :

- the user approaches the infiltrator and holds the arrest key.
- Infiltrator gets out of range before Auber completes the arrest.

Success Postcondition: Infiltrator is in the brig now. Once all infiltrators are in the brig the game is won.

We also started on the risk assessment and mitigation. A risk is a potential scenario that could affect the final result of the game . Risk mitigation involves monitoring these risks and planning actions to minimise their potential consequences. Risks can span all aspects of the game design, including potential technological issues, project issues and product issues. We scored these risks on their likelihood of occurring and the size of its potential impact, both rated from low to high.

ID	Type	Description	Likelihood	Impact	Mitigation	Owner
R1	Technology	LIBgdx not compatible with game design(can't introduce specific game feature)	Low	High	We simplify game mechanics to fit our LIBgdx capabilities	Fraser
R2	Project	One or more members of the project don't make the meeting(s)	High	Low	Fill anyone who missed out on anything post meeting and balance out workload between those who made it	Kurtas
R3	Product	The game is too difficult for the expected user	Medium	Mediu m	Provide different difficulty levels that the player can choose from	Sarah
R4	Technology	Issues with using gradle to set up LIBgdx projects	Medium	Low	Members of the team who managed to set up LIBgdx projects successfully can assist members who have issues	Finley

Week 4

In week four we started working on the Class Diagrams and we talked about which class we are having and their 3 compartments: Name, Attributes and Operations. The picture below shows the first draft of our class diagram. Which has class Entity, Auber, Aliens and Infiltrator with their attributes and their operation method. And in the first draft we didn't use the box-arrow class diagram, we just listed the class we need and their Name,attributes and operations.

Game
Entity
Properties
- Room String
- Location(x,y) Array

Methods
Auber(Inherits Entity)
Properties
- Image

Methods
- arrestInfiltrator(Infiltrator)
- teleport(newRoom)
- Move

Aliens(Inherits Entity)
Properties
- Image
- Species

Methods
- move

Infiltrator(Inherits entity)
Properties
- Image
- Arrested boolean
- NearestSystem

Methods
- destroySystem(CriticalSystem)
- findSystem()
- move

InfiltatorAbility1
Method
- UseAbility

InfiltatorAbility2
Method
- UseAbility

InfiltatorAbility3
Method
- UseAbility

We also started the website. We decided to host our website on Github Pages, and we chose this because hosting a website there is free. It has its own built-in editor which is more user-friendly and simplified than HTML and CSS, and it is a collaborative workspace allowing all members of the group to make their own edits.

We started the website a week later than anticipated however nothing at this stage depends on it so the priority was low.

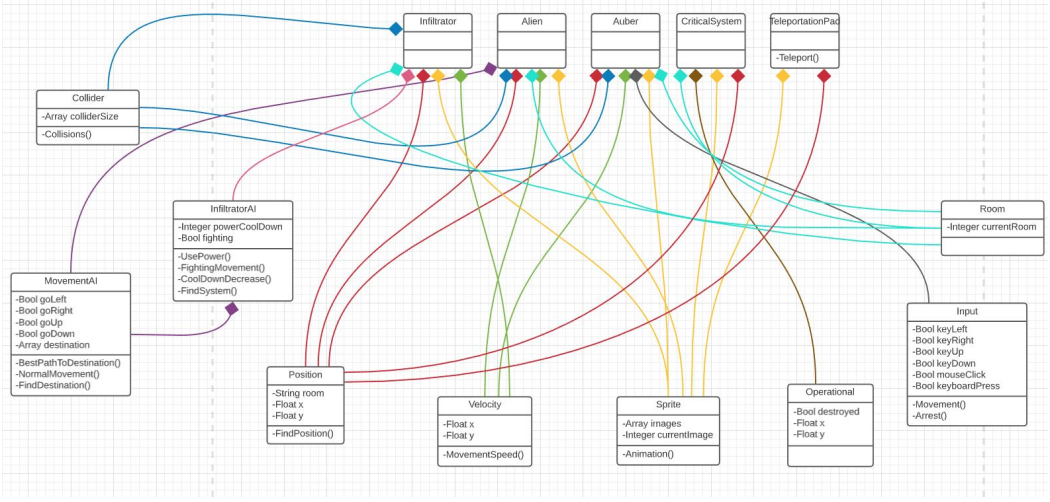
Week 5

In week five we start on the Pseudo Code which firstly gives us a general idea of what class we are going to use ,secondly what variables we are having in the class and finally we can discuss and have a general view of what function we are building within the class. The picture below shows the start of our pseudo code. We also start to think about our UGI design ideas which can let users get to know how to play our game more easily.

Movement AI

```
Def FindDestination(x,y,destination,goLeft,goRight,goUp,goDown):
        If destination[0]==x or destination[1]==y:
                destination[0]=random int (range,room area x)
                destination[1]=random int (range,room area y)
        If destination[0]>x:
                goRight=True
                goLeft=False
        Else If destination[0]<x:
                goLeft=True
                goRight=False
        If destination[1]>y:
                goUp=True
                goDown=False
        Else If destination[1]<y:
                goDown=True
                goUp=False
 Def NormalMovement(goLeft,goRight,goUp,goDown,x,y)
        If goLeft==True:
                Decrease x
        Else if goRight==True:
                Increase x
        If goUp==True:
                Increase y
        Else if goDown==True:
                Decrease y
```

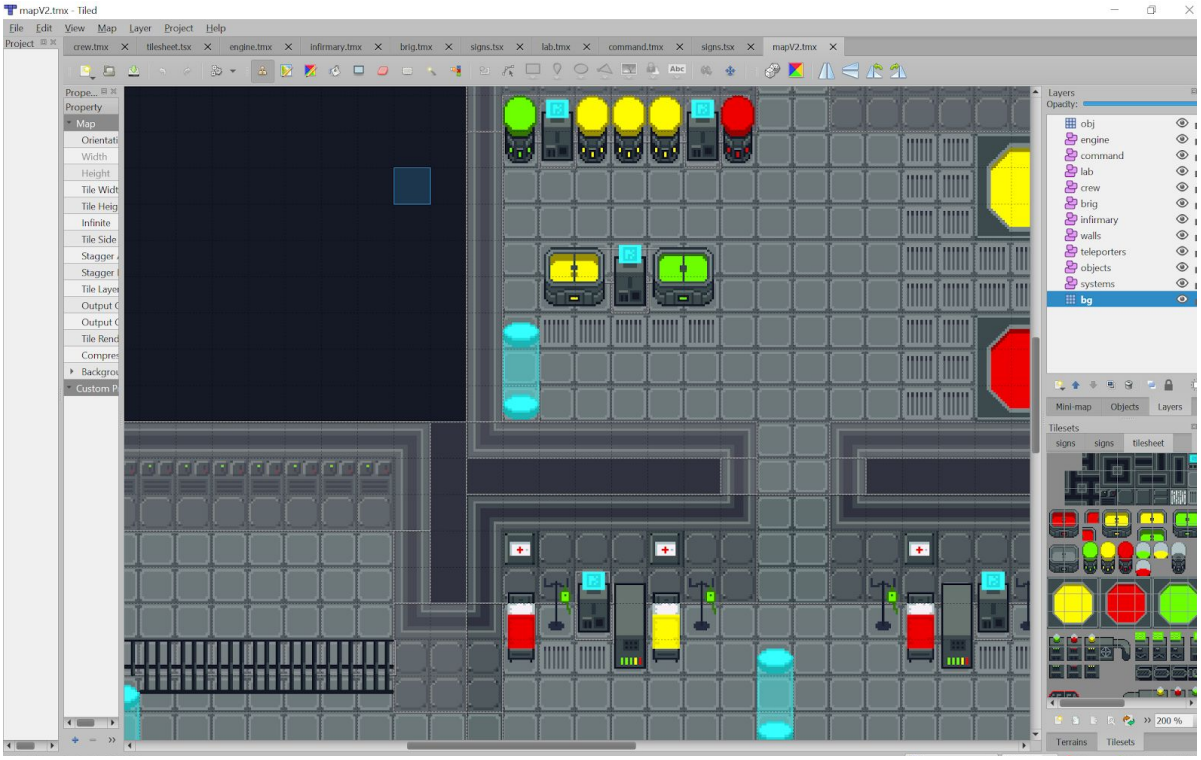
We finished the pseudo code,the general idea of the UGI design and the first version of the class diagram in week five.



We then decided to make this version of the class diagram more abstract with less details to fit with our abstract architecture.

Week 6

In week six we start to implement the rooms and Auber. We found the assets we wanted to use for our game, created a map using Tiled and began implementing the movement of Auber as well as collisions.




```java
@Override
protected void handleMovement() {
    //Left movement
    if(PlayerInput.getDirection()==1){
        Vector2 position = movementSystem.left();
        setPosition(position.x,position.y);
        if (facingRight==true){
            sprite.flip( x: true, y: false);
            facingRight=false;
        }
    }
    //Right movement
    if(PlayerInput.getDirection()==2){
        Vector2 position = movementSystem.right();
        setPosition(position.x,position.y);
        if (facingRight==false){
            sprite.flip( x: true, y: false);
            facingRight=true;
        }
    }
    //Up movement
    if(PlayerInput.getDirection()==3){
        Vector2 position = movementSystem.up();
        setPosition(position.x,position.y);
    }
    //Down movement
    if(PlayerInput.getDirection()==4){
        Vector2 position = movementSystem.down();
        setPosition(position.x,position.y);
    }
}
```

```java
public class TileWorld {
    private Hashtable<String,Rectangle> teleporters;
    private ArrayList<ShipSystem> shipSystems;
    private ArrayList<Rectangle> collisionBoxes;


    private Rectangle infirmary;
    private Rectangle brig;
    private Rectangle crew;
    private Rectangle command;
    private Rectangle laboratory;
    private Rectangle engine;
    private int scale;


    public TileWorld(PlayScreen screen){
        /* Creates all objects that the sprites can interactive with
        *@param screen the main game screen*/

        TiledMap map= screen.getMap();
        this.scale=AuberGame.ZOOM;
        createRooms(map);
```

Week 7

As the implementation of Auber took longer than expected, we began programming the infiltrators and the systems. We found we needed to add a few more procedures than we originally thought when designing the system.

```java
public class ShipSystem {
    private float x;
    private float y;
    private int state;
    private String room;
    private PathGraph graph;

    public ShipSystem(float x,float y, String room, PathGraph graph){
        this.x=x;
        this.y=y;
        this.room=room;
        this.state=0;
        this.graph = graph;
    }
    public void setState(int state){
        //state 0= operational, state 1=under attack, state 2= not operational
        this.state=state;
    }
    public int getState() { return state; }
    public String getRoom() { return room; }
    public Vector2 getPosition() { return new Vector2(x,y); }
```

Week 8

In week 7 we began testing the testing phase of development. First we started with developer testing - where we devised test cases that together gave us a level of confidence that our game is working. These test cases were prioritised by how likely the bug was to occur if plate by a regular user. More common bugs would include [BUGS E.G. GOING THROUGH WALLS, GETTING STUCK, COLLISION ERRORS]. We used unit testing - writing pieces of code that test [E.G. TESTING WALL COLLISIONS WITH CORNERS, BEING 1 PIXEL OFF TELEPORTER, DOOR, ETC.]. These were fast, easy to write, and easy to control.We tried to fix as many bugs as possible and reduce the likelihood of others before deploying our software.

How the plan evolved

The plan was followed through well upto coding which took a bit longer than expected, particularly Auber. However we had predicted this and therefore left a few days in our plan free before deployment. We started our website a week or so after we had planned however this did not affect the rest of our progress.