Architecture

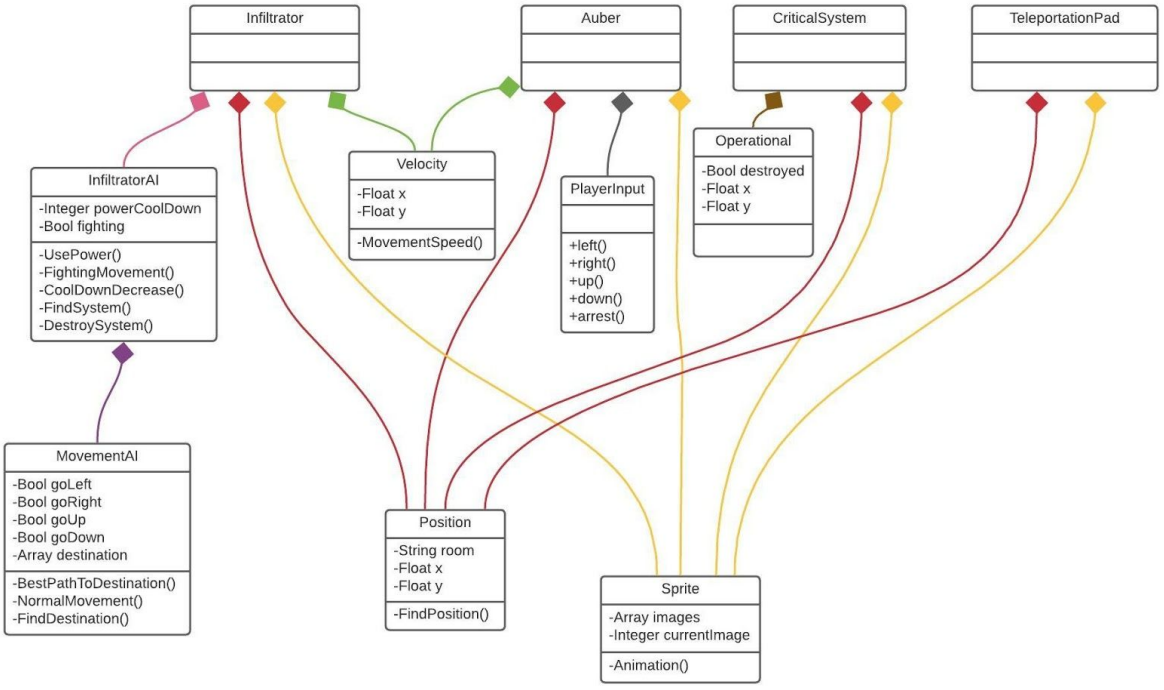
Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

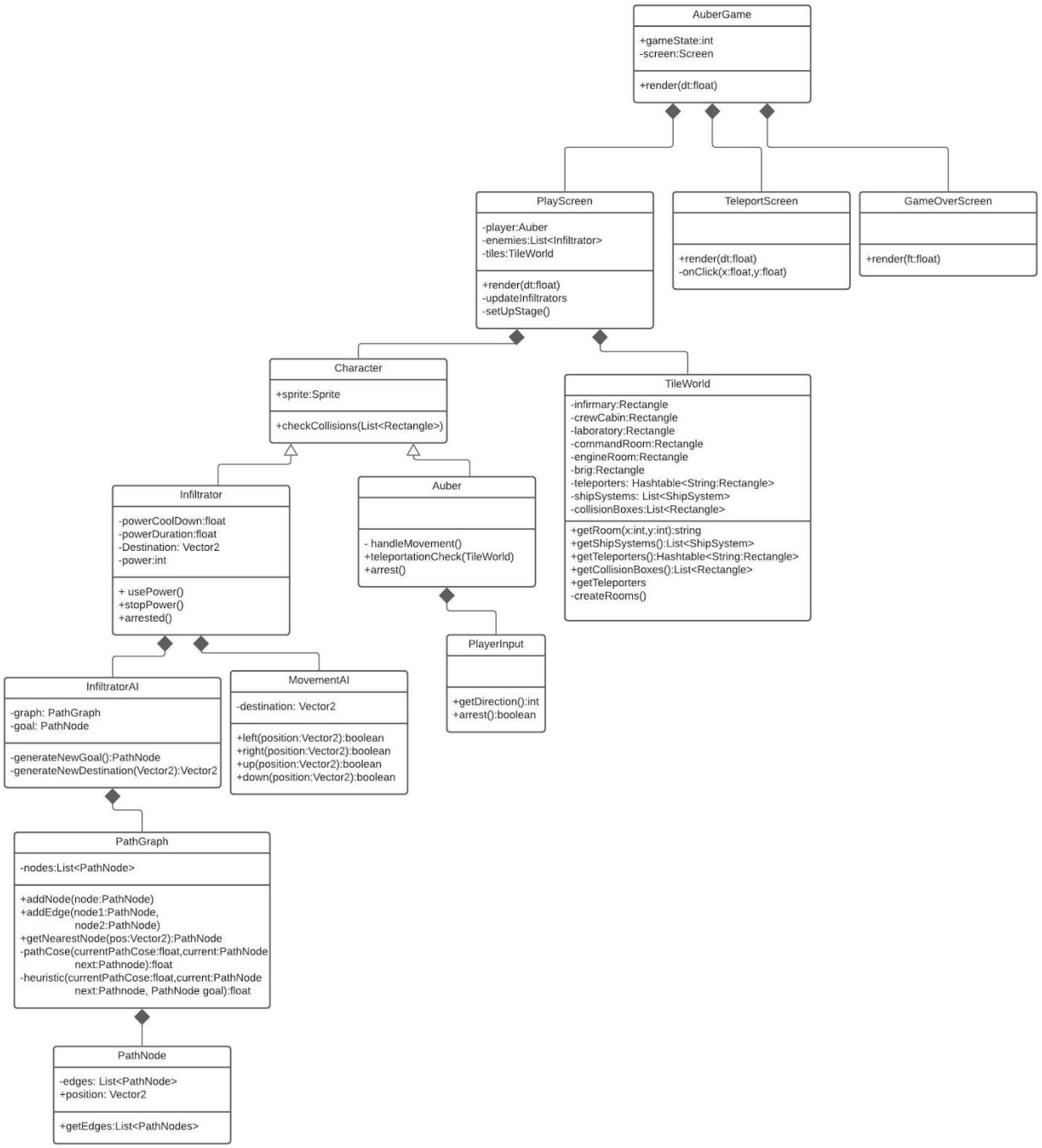
3a: Representation of architecture

We used lucidchart.com to create our class diagrams.Lucidchart itself is a software that can create all sorts of diagrams, from flow charts to class diagrams and more. We used lucidchart as using the organisation code of UML our diagrams were hard to follow, having connections grouping together making it hard to read. Lucidchart on the other hand makes easy to follow class diagrams that can be formatted to our specifications.
The language we based our class diagrams on was Java as it was the required language to code in.

<u>Abstract representation of architecture</u>



Concrete representation of the architecture



3b: Justification of architecture

Our abstract representation was based heavily on the Entity-Component-Systems pattern, favouring composition over inheritance. We chose this specific system as many current game development companies use this system as it is efficient, using less processing power, and is easier for the coder to implement. This is shown through the classes that use multiple of the same sub classes (e.g. velocity, sprite and position) however also have separate sub classes making them different. The sub classes then allow extra classes, for example a civilian class, to be implemented easier. The civilian classes can be composed of these predefined classes whilst being able to also use its own bespoke subclasses.

The abstract representation gave us a good idea of which classes needed to exist and how they would interact with each other. However when it came to implementation we found that we gravitated towards a more object oriented approach. This was due to the groups familiarity with it from our prior experiences using it. This meant that the final code uses a combination of the two ways of approaching classes, causing a difference with the concrete representation.

The concrete representation still builds on the foundation of the original abstract representation. Classes such as MovementAI, InfiltratorAI, PlayerInput etc. still have a key role in the concrete representation staying relatively the same but with more information and showing more specific methods (e.g. PathGraph, PathNode in infiltratorAI, GetDirection in PlayerInput). However there are some changes from the abstract representation, clearly shown via the connections, as now some classes inherit from a parent class. These classes are Infiltrator and Auber, as in implementation we found that it was easier to approach these two elements with an overarching parent class called Character.

The software we used in the development (ide intellij, game engine libgdx, language java) affected the concrete representation. Multiple variables were created that are specific to the game engine such as Rectangle variables like infirmary, crew cabin etc. making a rectangle of coords. This allows for collisions to be easier to code. New classes were created that did not exist in the abstract class as it is easier to program with these new classes. These classes are more specific with their variables and methods, (e.g. parameters) shown in methods getShipSystems():List<shipSystems> and getRoom(int x, int y):String. These show the format the methods would be in and the type of inputs and outputs they would have. Variable types are also specific to the language and game engine, (e.g. Rectangle, Hashtable, List etc.). This gives a better understanding of how the variables and methods would work in the context of the language and engine.

<u>Concrete Architecture and our Requirements</u>

In our concrete representation, we show how the player will interact with the game using a keyboard as per the FR_MOVEMENT and NFR_MOVEMENT_RESPONSE requirements. The player's input will be detected in the PlayerInput class and handled in Auber class in the method handleMovement() to move the auber on the screen.

For FR_ARREST (Auber can arrest enemies by holding down the space button), the detection of the space bar press is in PlayerInput class with the arrest() method. It then arrests the infiltrator using the arrest method in Auber class and the arrested method in Infiltrator.

For our game we decided to use a tile map which had MapObjects. This means we could find the locations of the teleporters in order to fulfil the FR_TELEPORT_PADS requirement. The location is found in TileWorld class and stored in a hashtable, teleporters, where the key is the room name and the value being the location. This hashtable can then be checked against the Auber location in teleportationCheck(), in the Auber class, where it sees if Auber is standing on a teleportation pad. The TeleportationScreen is then shown and allows the user to pick a destination. The auber is then teleported to the destination using the hash table coordinates,

In addition we designed the implementation of FR_INFILTRATORS_AI which dictates how the infiltrator will find and destroy a system. A graph PathGraph is created to allow the infiltrator to traverse the contained PathNodes which represent either a system or a doorway to another room. This lets the infiltrator find a path from their current location to another room and a corresponding operational system.

The type of power the infiltrator has is kept in the power attribute of the Infiltrator class. We could have implemented separate classes, one for infiltrator and one for every type of power, however to avoid inheritance we decided to store it in a variable. When usePower() is called, the corresponding infiltrator power is called to effect. For FR_INVISIBILITY and FR_SHAPESHIFT the infiltrator texture is changed. For FR_HALLUCINATIONS a hallucination is caused on PlayScreen and for FR_SPEED_BOOST the speed of the infiltrator is increased in MovementAI.

In order to implement the requirement of not allowing the infiltrators to repeat their power till a cooldown period has ended in FR_SPECIAL_ABILITIES, the Infiltrator class has a powerCoolDown attribute which holds the time since the last time the power was used. This is updated using the delta time in render from PlayScreen class.

We implemented a GameOverScreen class which is displayed from AuberGame when the game state is 2 or 3 (win or lose, 1 being what the screen is whilst playing). This displays to the player if they have won or lost (FR_GAME_OVER).

The location of the 6 unique rooms as per the FR_UNIQUE_ROOMS requirements are defined in TileWorld as Rectangles. These rectangles can be used to check what room an object is in. For example it can be used to check if Auber is in the infirmary to heal as per the FR_INFIRMARY requirement.

References

Mark Jordan, 2018, Entities, Components and Systems, https://medium.com/ingeniouslysimple/entities-components-and-systems-89c31464240d

Implementation

Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

6b: Unimplemented features

We failed to implement our UR_CITIZENS requirement which stated we needed non hostile civilians in the game. This was due to time constraints however this does not affect the gameplay. This was an optional requirement that we assigned the priority “may” to. Because of this we could not implement FR_SHAPESHIFTING the way we wanted to. Instead we shapeshifted the infiltrator into an object from the ship to disguise it.

When starting up the game, the user is shown how they can move using WASD and arrest using space. They are also told they need to protect the systems and what the systems look like. An image of the map of the spaceship is also shown. This partly fulfils NFR_OPERABILITY which states “Any user with any gaming skill level can learn to play the game within the first session”. However I think we could have added more to fully implement this by adding a tutorial mode with pop ups giving the player hints. However this specific feature was never a requirement.

Our FR_MOVEMENT requirement was successfully fully implemented however could be improved as collisions could cause the player movement to become slightly jumpy. This never resulted in the player getting completely stuck, but would take the player half a second to move away from the collision box to start normal movement.

Resources
For our project we used a mix of open source assets and some we made.

“Arks”,Unknown date, Dino Characters, https://arks.itch.io/dino-characters

“DeadEnd-lmz”,Unkown date, Industrial RPG pixel Art
https://deadend-lmz.itch.io/industrial-rpg-pixel-art

Method Selection and Planning

Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

4a: Justification of Software engineering methods

Software engineering methods

In terms of planning we used an agile method, more specifically following a scrum approach. Each development in the project was divided into sprints, getting together for a daily meeting reporting what each person had accomplished and what the plan for the rest of the day would be.

We decided on this approach as we wanted to stay flexible around the changing requirements of the project. There were constantly changing factors due to new game ideas we'd brainstormed or new requirements we faced after our multiple customer meetings. For example we decided to not use graphic violence after our client stated the game was to be played on open day. Another significant change was our decision to switch from Box2D to Scene 2D, all while still programming the game. However due to our agile method planning this was a seamless transition and did not take a heavy toll on time management.

Trello

Trello was an extremely helpful tool when it came to integrating the scrum approach into our project. We created a board for each specific task (architecture, risk assessment, etc) where we split the page into 3 sections consisting of what to do, what we're doing, and what we've done. This just aided in staying organised with each daily task and making sure no one was working on the same job.

GitHub

A few of our members had used GitHub before and were fond of the site. We ultimately decided on using it as it was perfect for the situation we currently find ourselves in. As there was no possibility of meeting up GitHub was ideal for remote collaboration as we all coded from separate locations.

Another aspect that attracted us to using GitHub was its security. We'd all have versions of the code and if anything was to go wrong or astray we could very simply revert back to old versions of our code or use someone else's branch, a possible problem that we had noted down in our risk assessment, mitigated through the use of GitHub.

Drive/Docs

Google drive was one of the first tools we decided on and was a very simple decision among the group. Everyone is very familiar with it, especially when it comes to group projects. We knew it was important to stick what has worked for us in the past and so Google Drive was dropped into the conversation immediately.

An alternative we had considered was OneDrive, however as students we have unlimited storage space on Google Drive, and we were more familiar with Drive.

It's easily accessible to everyone in the group and a good way of tracking which tasks have been completed or need to be assigned, which helps keep everyone motivated to work as you know your fellow team members can see who has or hasn't been keeping up with the tasks.

We knew it would be particularly useful at the beginning of the project where the focus is on documentation as we would split up in pairs to work on requirements and google drive and docs allows multiple users to work on the same document.

Intellij

In terms of alternative tools we had considered, the most significant decision came down to choosing between Eclipse and Intellij. Ultimately we came to the conclusion that Intellij was the best choice for our project. We had members of the group who had used one or both so we had a team discussion on the positives and negatives of the two choices.
Whilst both have their advantages the majority voted for Intellij. Fraser gave us the strongest argument as he had attempted to create a game recently, specifically with Intellij and found that it had been much easier to learn the finer details of it if you had never used it before, in comparison to Eclipse which had a steeper learning curve.
We had already discussed the importance of time management and staying as efficient as possible, which led to us picking Intellij in the hope it saved us time and stress, due its user friendly interface.

4b: Team organisation

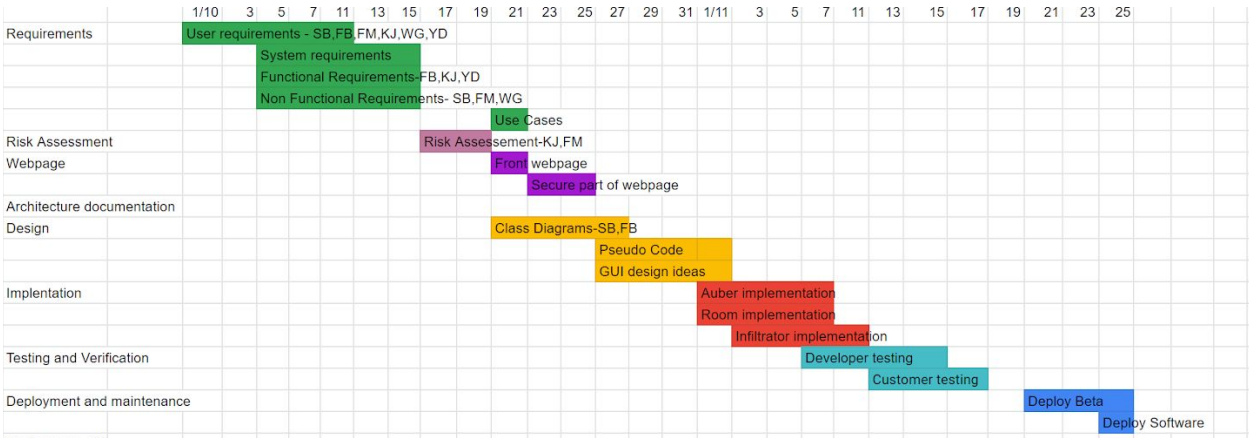
Our group has set up a workspace on Trello, an online collaborative teamwork platform. We have created separate rooms for each deliverable, within which there are 3 sections: to-do, doing and done. This approach is appropriate for the team as we are unable to meet in person due to the current COVID-19 lockdown situation, and it keeps all team members up to date with what tasks have and haven't been completed yet.

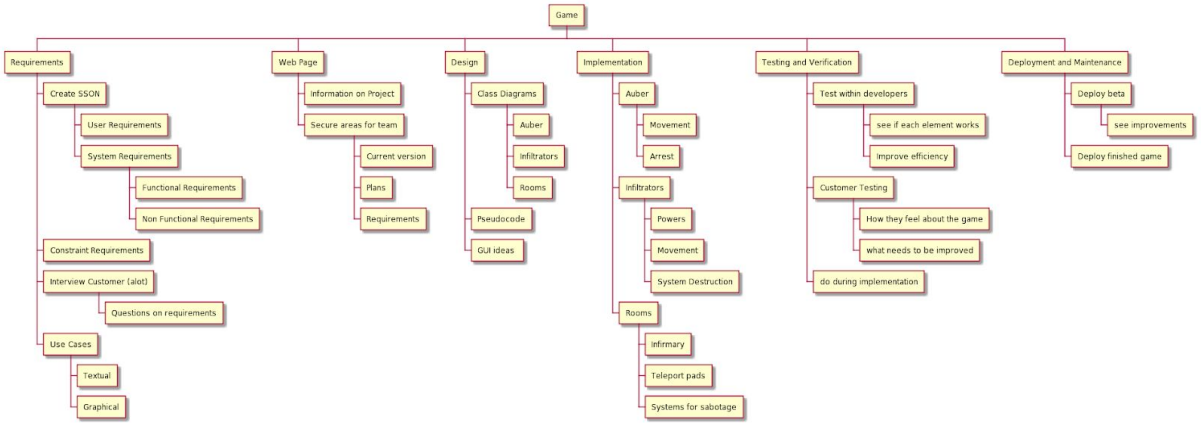
We have also set up a Google Drive, to which each member can upload work they have done to a Google Docs file. This is where all of our work was uploaded, and from there it was added to the Github Pages

We have set deadlines for each deliverable and their respective subsections using a Gantt chart. This is suitable for the project as time management allows us to keep on track with deadlines and not fall behind.

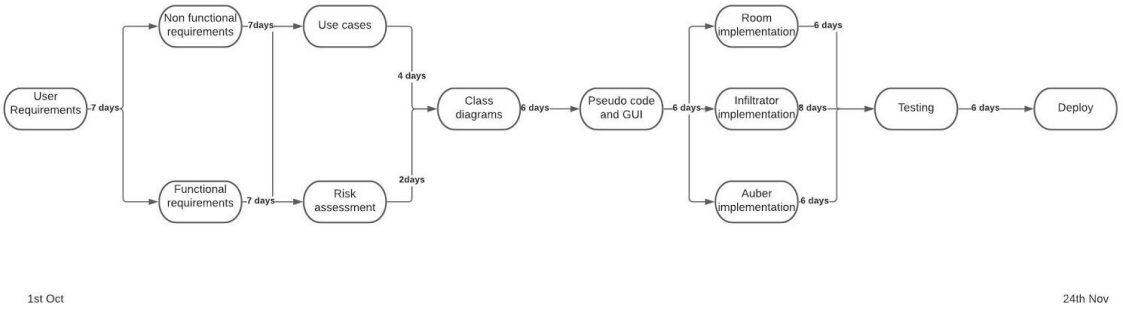
For each small task, roles are assigned during our group meetings, and each group member can assign themselves any desired roles by annotating the Gantt chart with their initials. Letting group members to assign themselves tasks allows for work to be allotted without the need for a group meeting, and can increase productivity if each group member is working on a task they want to complete.

4c: Project plan





<u>Our critical path</u>



Our critical path shows the estimated minimum possible amount of time. From this we can deduce the project will take at least 44 days by adding the paths together (taking the larger from the parallel path). This optimistic estimation suggests we may finish on the 13th of November which gives us 11 days extra time if anything goes wrong.

<u>Our plan</u>

1/10/20-16/10/20

In the first week, we will discuss with our stakeholder their needs. This task priority is very important as every other task depends on it. Our user requirements from this meeting should be finished on the 11th of October. From our stakeholders needs we will decide our game requirements which the implementation depends on.

17/10/20-20/10/20
When we finish the requirement we will be looking at the risk assessment. We will identify the risks that are likely to threaten the project and we will analyze the consequence of it. In addition we will discuss how to minimise or avoid the causes of the risks and throughout the project we keep monitoring the risks to see if the risks are happening. This is medium priority as if something goes wrong in our project we will be better equipped to deal with it.

21/10/20-26/10/20
After we finish the Risk Assessment, we will start doing the Use case and the web page.The web page has a low priority at this stage as nothing depends on it and can be done any time during the project.

21/10/20-28/10/20
Simultaneously we will begin the class diagram design. Our class diagram design is high priority as the implementation depends on it. If our class diagram is not detailed enough or badly engineered this will affect the outcome of our game. For the class diagram we will design the classes we need, their attributes and methods as well as the relationship between classes.

27/10/20-2/11/20
We will then begin the pseudo code which will give us a general idea of what to code in the implementation state. We will also begin to design the GUI which may inform us on extra things we need to implement for our game to be played easily. The pseudo code is medium priority as the code depends on it however it is not essential to write the code.

1/11/20-12/11/20
We will then start coding, starting with Auber and the rooms. After that has been tested we will begin coding the infiltrators. The infiltrators depend upon the room as it needs a destination for the AI (the ship systems)

7-11/20-18/11/20
We will then begin the test cycle in unison with our coding, testing what we have already done before moving on. This is a high priority and the rest of the code is dependent on it as we cannot keep writing code until it has been tested.

21/11/20-24/11/20
We have left a gap between testing and deployment to allow for room if any part of this project requires more time. By the 23rd we should have completed the entire project.

Requirements

Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

2a: Introduction

In our first meetings we read through the assessment briefing and discussed what our stakeholder's needs might be. We then had our first customer meeting where we discussed their requirements for the game and we asked our user what they required from the game. We then negotiated what we would implement and what extra features we could add that the user may not have originally asked for.

We represented our requirements using a Functional, Nonfunctional and User Requirement table. This allowed us to clearly group our requirements and relate them to one another. Our User requirements table has a column for priority which allows us to focus on the things the user thinks is essential before moving on to less essential requirements. Our Functional and Nonfunctional requirement tables have a column for their related user requirement.

A System Requirement is description of how the system will deliver on the needs of the users, detailed descriptions of functionality, services and constraints.
The Functional Requirements are the things that a system must do and an action that the system has to take if it is to provide useful functionality for its user.

Functional requirements can be grouped into:

- Transformation:
    - Required response to a condition/event
    - For example FR_MOVEMENT and FR_ARREST
- Invariant:
    - Properties that must always hold
    - For example FR_SPECIAL_ABILITIES and FR_UNIQUE_ROOMS
- Failures:
    - Forbidden/permissible transformations
    - For example FR_GAME_OVER because you cannot win or lose until you arrest 3 infiltrators or the infiltrators destroy 15 systems.

For the Non-functional Requirement, there are 13 attribute which are Security, Reliability/Availability, Timing, Precision, Constraint, Maintainability, Documentation, Resilience, Integrability, Scalability, Operability, Auditability, Accessibility, Usability. For example 'The system is highly responsive to user keyboard inputs' from NFR_MOVEMENT_RESPONSE is a precision constraint.

<u>SSON</u>
A single-player game, in which the user tries to arrest the infiltrators before they destroy a critical number of key systems of the station.

2b: Statement of requirements

User Requirements

ID	Name	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 between teleportation pads in the station, each room has a teleportation pad	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 heal when and only when in the infirmary	Shall
5	UR_REAL_TIME	The game must be real-time (not turn-based)	Shall
6	UR_KEYBOARD	User can move Auber using the arrow keys and/or the WASD keys	Shall
7	UR_VIOLENCE	No graphic violence	Should
8	UR_SOUND	Minimal sound in terms of music and necessity to have sound on	Should
9	UR_JAVA	Written in Java only	Shall
10	UR_ARREST	Auber can arrest infiltrator	Shall
11	UR_FAIL	If user lets the infiltrators destroy the systems, they must lose	Shall
12	UR_SUCCEED	User can win the game by arresting infiltrators	Shall
13	UR_INFILTRATORS	There must be 8 infiltrators	Shall
14	UR_CITIZENS	There must be non-hostile characters in the game	May
15	UR_SYSTEMS	The systems can be destroyed by the infiltrators	Shall
16	UR_ACCESSIBILITY	The game must be accessible and easy to play	Shall

Functional Requirements

	ID	Description	User Requirements
1	FR_MOVEMENT	When user presses movement key Auber moves in appropriate direction (WASD and arrow keys)	UR_KEYBOARD
2	FR_ARREST	Auber can arrest enemies by holding down the space button.	UR_ARREST
3	FR_TELEPORT_PADS	Auber can only teleport on teleport pad to any other room	UR_TELEPORTATION
4	FR_SPECIAL_ABILITIES	Infiltrators have powers and cannot repeat their power till a cooldown period has ended	UR_SPECIAL_ABILITIES
4a	FR_INVISIBILITY	The infiltrator can go invisible	UR_SPECIAL_ABILITIES
4b	FR_HALLUCINATIONS	The infiltrator can cause Auber to have hallucinations until they enter the infirmary	UR_SPECIAL_ABILITIES
4c	FR_SHAPESHIFTING	The infiltrator can shapeshift into a civilian	UR_SPECIAL_ABILITIES
4d	FR_SPEED_BOOST	The infiltrator can move at an increased speed	UR_SPECIAL_ABILITIES
5	FR_UNIQUE_ROOMS	System has 6 unique rooms: Command room, infirmary, laboratory, crew cabin, engine room and the brig.	UR_FOUR_ROOMS
6	FR_SABOTAGE	Infiltrator can sabotage system when next to it	UR_SABOTAGE
7	FR_INFIRMARY	When auber is in infirmary they heal and the hallucination effect wears off	UR_HEAL
8	FR_GAME_OVER	System generates “Game Over” message when you lose	UR_FAIL
9	FR_INFILTRATORS_AI	Infiltrators must be able to utilise an AI to move on the screen and destroy systems	UR_INFILTRATORS
10	FR_SYSTEMS_DESTROYED	The player can see how many systems have been destroyed	UR_SYSTEMS

Non Functional Requirements

	ID	Description	User Requirements	Fit Criteria
1	NFR_MOVEMENT_RESPONSE	The system is highly responsive to user keyboard inputs	UR_KEYBOARD	<0.05 seconds
2	NFR_OFFLINE_SECURITY	The game can be run when the user does not have internet connection	UR_ACCESSIBILITY	No internet
3	NFR_HEARING	The game can be played by someone who is deaf	UR_SOUND	N/A
4	NFR_ENGLISH	All messages shown to the user must be in one language and minimal jargon	UR_ACCESSIBILITY	All in the English language
5	NFR_OPERATING_SYSTEM	The system must be able to run on specified operating systems	UR_ACCESSIBILITY	Linux, MacOS, Windows
6	NFR_OPERABILITY	Any user with any gaming skill level can learn to play the game	UR_ACCESSIBILITY	Within the first game session

We can assume that the user has high enough computer specifications to run our game as well as a means to input data such as a keyboard and mouse. We also assume that the screen resolution is 2560x1440 (however we will still implement resize).
An associated risk was assuming that the user had previous gaming skill level and would understand the base of any video game. Although this is a small risk we still mitigated it with the addition of a tutorial.
An associated risk involves the Infiltrators AI. As the AI could get trapped on areas making it impossible for the player to finish the game. This is a large risk when making an AI as stopping the player from playing is the worst scenario. However it can be mitigated we can give specific nodes so that the AI does not go off track and moves to an area they would get stuck.

References

Unknown Author, 2014, Software Requirements Specification, http://graduatestudents.ucmerced.edu/tshea2/files/SoftwareRequirementsSpecification_0.pdf

Risk Assessment and Mitigation

Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

5A: Justification of Risk Format

We have taken various risks that may occur and given them a likelihood that ranges between low to medium to high. The impact is in reference to how much of a problem this would actually be in the larger picture of the project as a whole. The rating for impact is also low to medium to high. With each risk we note down how we would minimise the impact instead of outright preventing it.
The way we have decided to approach the risk assessment is through a table of values showing, Type, Description, Likelihood, impact and how to mitigate said risk. We found that this was the easiest way to represent the risks in a moderate amount of detail that would be easy to understand.
The risks can be referenced through ID and each risk has its own row making them easy to follow and differentiate. Due to it being in a table format the level of detail isn't high which is a good thing as it does not over complicate the risk itself making it hard to read. This is shown in the description box and further emphasised in the mitigation box as the mitigation provides to the point ways of preventing said risk. The likelihood and impact columns help us to determine which ones to look out for and how likely they will be, looking out for high impact and expecting high likelihood risks.

5B: Risk Assessment Table

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	Medium	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
R5	Project	Problems regarding time. We may struggle to finish intended tasks within the time limit	High	Medium	Focus on finishing the key tasks that have the highest priority	Yupeng
R6	Project	Yupeng's 7 hour time difference will cause problems with communication and organisation	High	Low	When possible try to find reasonable tasks and assign work for him after meetings.	Will
R7	Product	One or more of the infiltrators' abilities is confusing for a player who has just started the game	Low	Medium	Brief description of each ability is available in the game menu	Fraser

R8	Product	The ship is too difficult to navigate(due to size of map or room layout) for user	Medium	Mediu m	A map with the names of each room and location of Auber is always available to user	Kurtas
R9	Technology	LIBgdx runs poorly on low end hardware	Low	High	Test the product on low end hardware and make graphical and gameplay tradeoffs to improve performance.	Finley
R10	Product	The game takes too long to finish/ complete.	Medium	High	User has the ability to change settings (speed, number of infiltrators, etc)	Sarah
R11	Product	The characters/ setting look unappealing to the user	Medium	Low	Prioritise on making the presentation clear and easy to understand rather than the aesthetic of it	Finley
R12	Product	Sound/music is irritating and/or unnecessary in the overall design of the game	Low	Low	Users can decrease the volume of the game or mute the sound all together.	Will
R13	Technology	Our main source of communication (discord) may go down due to faulty servers	Medium	Low	Have a second form of communication (eg snapchat, email, facebook)	Yupeng

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.

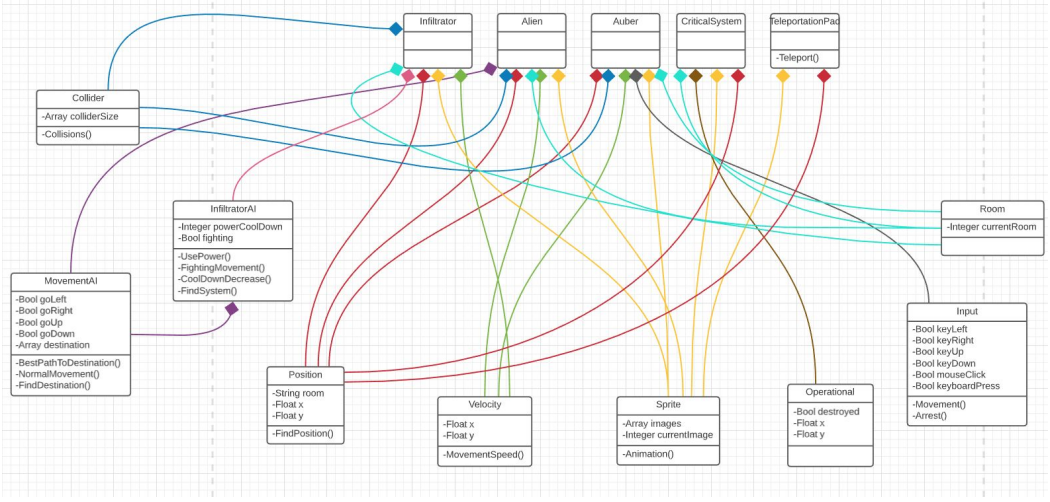
Week 5

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
```

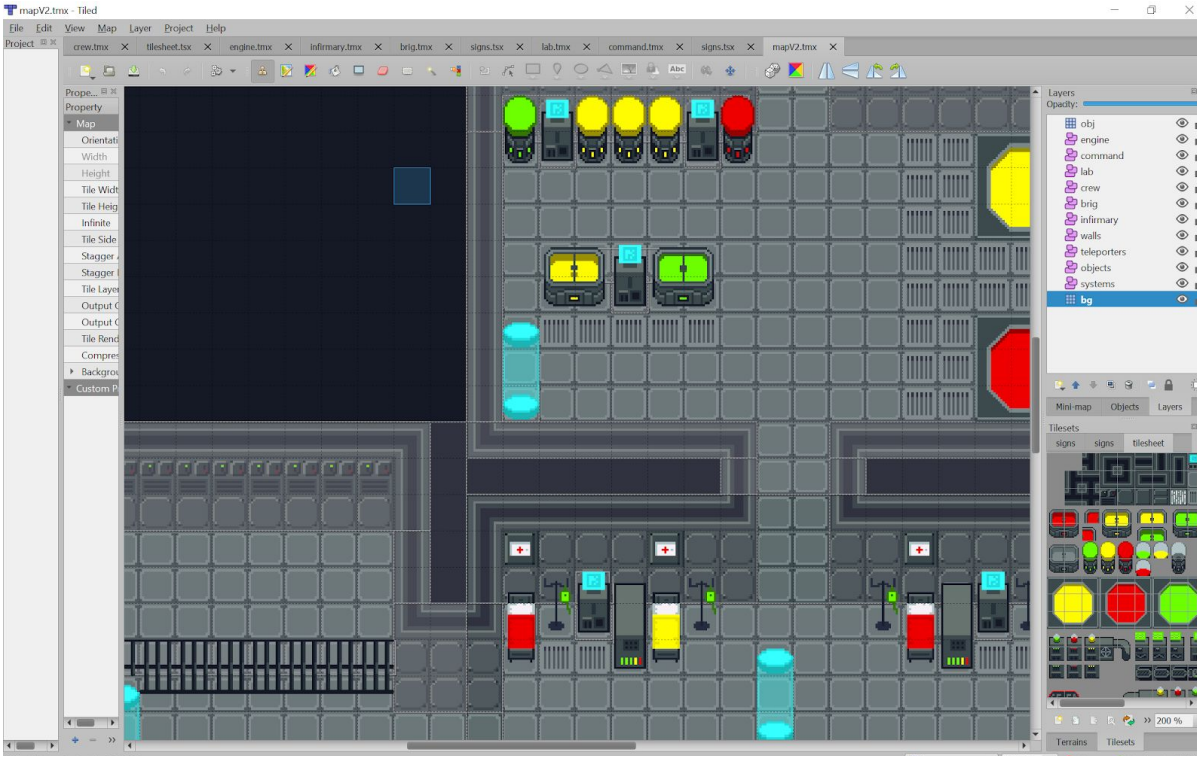
We finished the pseudo code,the general idea of the UGI design and the first version of the class diagram in week five.



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.

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.

Minimal Postcondition: NONE

Case: Destroy System

Actors:

- Primary Actor: Infiltrator
- Secondary Actor: System

Precondition: Infiltrator is within range of an undamaged system

Trigger: AI decides to destroy the system

Main Success Scenario:

- The infiltrator stays still for the duration of the destruction
- The system breaks

Secondary Scenarios: :

- The infiltrator is interrupted, i.e. they move or are arrested, during the duration of the destruction
- The system remains unbroken

Success Postcondition: The system is broken, which causes the relevant effect to happen to the ship, if all of the systems are broken then the game ends in a loss

Minimal Postcondition: NONE IDK

Case: Teleport

Actors:

- Primary Actor: Auber
- Secondary Actor: Teleport Pad

Precondition: Auber is on the teleport pad

Trigger: user inputs teleport key

Main Success Scenario:

- the user approaches the teleport pad and presses the teleport key.
- User decides which teleport pad he wants to go to on the menu that pops up
- User teleports to the pad he chooses

Secondary Scenarios: :

- the user approaches the teleport pad
- User goes off menu and decides to stay in the current room they are in

Success Postcondition: Auber teleports to various rooms to stand a better chance of repairing systems or arresting infiltrators

Minimal Postcondition: NONE IDK