DEPARTAMENTO DE ENGENHARIA INFORMATICA

DIOGO SOUSA - 55492

LOURENÇO SOARES - 54530

PEDRO RIBEIRO - 55921

SCAVENGER HUNT
GROUP ASSIGNMENT

Jogos e Simulação
GAME DESIGN DOCUMENT

NOVA School of Science and Technology | FCT NOVA
NOVA University Lisbon

Ano letivo, 2021/22

1 CONCEPT DOCUMENT

Title and Premise

Scavenger Hunt is an arcade style dungeon crawler with horror elements, where you are trying to survive for as long as possible while accumulating a high-score.

Player Motivation

The player will attempt to beat the high score in various new levels while in constant tension to run of the monster.

Unique Selling Proposition

A thrilling dungeon crawler with horror mechanics, which will cater to people who enjoy more action-styled horror games (such as *Resident Evil 4*), the high score chasing experience of rogue-likes (such as *Spelunky*). The game offers multiplayer with multiple players, increasing the difficulty since more monsters will appear.

Target Market / Target age

The target market is mainly composed of people that enjoy the thrill of trying to beat their previous record. The added horror mechanics will help make the game more interesting for other types of players. Ideally, the game will be given a T rating (according to the ESRB system) as it would contain violence and horror, but not as far as to contain actual gore or blood.

Genre

Third person arcade style dungeon crawler with horror aspects. Multiplayer consists of various number of players in the room running from multiple monsters.

Expected Hardware Requirements

We have based the following minimum requirements on the weakest hardware in our team: CPU: i7 2.2GHz; GPU: NVIDIA GeForce 555m; RAM: 4GB; Storage: 210MB;

Competitive analysis

- Hades - A fast paced hack'n'slash rogue-like dungeon crawler;
- Diablo - Dungeon crawler with strong RPG elements;
- Path of Exile - similar to *Diablo*, but free and with large a set of customizable characters;
- Dead Space - A horror game with a fully diegetic interface;
- Binding of Isaac - A body horror dungeon crawler with a quirky and mysterious story;

Goals

We intend to immerse the player in our game and make them feel anxious/fearful, dreading what is around the next corner. To accomplish this, we made the game very dark to reduce visibility. We would also have a permanently player-chasing NPC to increase tension, similar to the Tyrant in *Resident Evil 2*. Somber and dynamic music, mixed with a strong sound design, assists in increasing tension when the big monster is nearby.

2 GAME PROPOSAL

Title, Hook and Goals

Scavenger Hunt is a game that keeps the player interested by generating new experiences per level. The added horror ambience forces the player to always be aware of his surroundings, and the limited supplies will require the player to balance their ammunition against the weaker enemies or to stagger the large (unkillable) one.

Story Synopsis, Backstory and Characters

You play Skye Forager, a scavenger that usually spends her days looting spaceships and selling off what is found inside. Skye is no stranger to action, but she had the unfortunate luck of stumbling upon PATHOS-IV, a seemingly abandoned ship that turned out to be more dangerous than she could have imagined. The ship's crew was corrupted by an unrelenting biomechanical entity, and Skye seems to be its next target. One of these entities, the Hunter, is large and will seemingly stop at nothing to make Skye its next victim.

Gameplay and Main Mechanics

The player takes a third person perspective and begins by exiting their own spacecraft. PATHOS-IV is randomly generated, and incredibly dark, which results in the player having an incredibly narrow field of view due to their flashlight being one of the few sources of light. The player searches rooms, fights enemies, and scavenges for ammunition and supplies (the latter which increases the player's score). The player is also being chased by an invincible monster (the Hunter), which can, at best, be staggered by the player's weapon. In multiplayer, multiple players can join a room and run from the monsters while trying to complete the level.

Technology and Main Components

Procedural generation is used for level creation, so for each run the player will get a different experience. The multiplayer connection is created using Photon. The Hunter includes the following different states: chasing the player, patrolling, or go check on a sound. In terms of lighting, real-time lights and particles are used in some sections which will help create a creepy atmosphere. An example of this would be flickering and hanging lights and visible dust particles in the multiple rooms of the level. The materials make use of bump-maps to add an increasing level of detail.

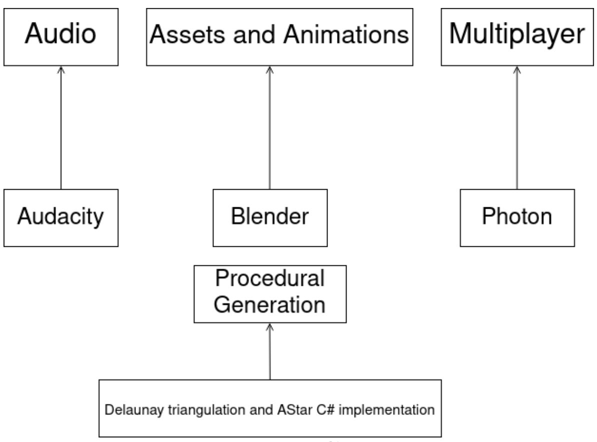


Figure 1: Main Components

Art and Audio

We find it important to have a strong visual aesthetic, so we have decided on a bright neon color palette for lighting with hard and dark shadows. Essentially, we are aiming for a combination reminiscent of *Tron*, *Far Cry 3: Blood Dragon*, *Technicolor Antichrist Box*, and *Doom 3*. The game will implement a more stylistic style of graphics as opposed to a realistic look. unfortunately, due to time constraints, we weren't able to make full use of the color palette like we intended.

A system of dynamic audio was implemented to give an idea of volumetric sound, this system is used to change the intensity of sound based on different distances and acoustics of the room where the sound was played. This audio system is used to control all the sounds of the game (except the music), it is used for shotgun shots, footsteps of the player and to control the different sounds played by the Hunter. The Hunter (and the weaker enemies) has different sounds that play based on what it is doing (attacking, patrolling or investigating other sounds).

A Music Manager was created to play dynamic music based on the circumstances of the player, if the player gets seen, a more fast and action music is played, when a player is heard by the monster (like footsteps or shotgun shots) the audio changes to an even more tense music. The Music Manager also guarantees that the song is never interrupted until it reaches the end of the bar.

Team

- Diogo Sousa
  - AI and Particle system programmer
  - UI designer
- Lourenço Soares
  - Procedural generation programming
  - Animation and AI developer
  - Assets
- Pedro Ribeiro
  - Multiplayer implementation
  - Chair and Table assets

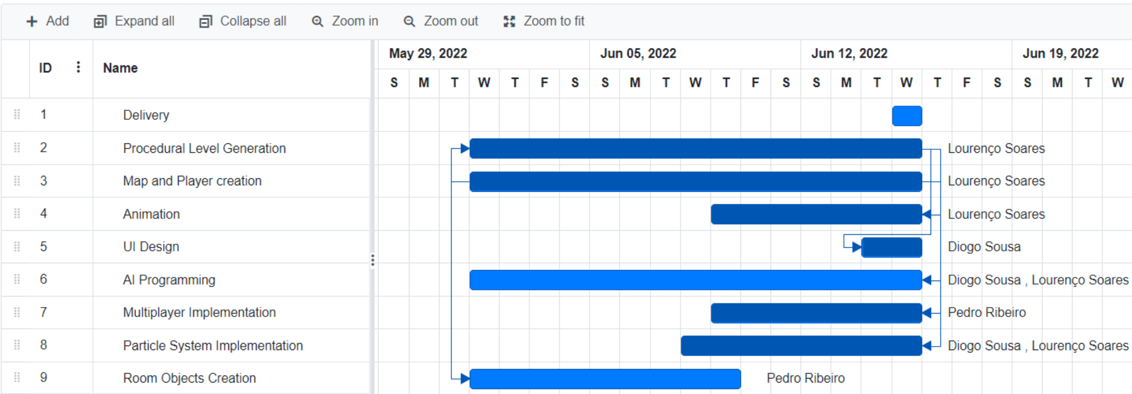


Figure 2: Gantt Chart

Risk Analysis

As is usual in game development, there is a risk associated with deadlines as the project can undergo unforeseen delays, in order to lower the chances of that happening we will follow a strict work schedule to make use of our time efficiently and to avoid any unnecessary crunches. Procedural generation also contains some risks as a more independent system can come up with some unwinnable scenarios, or lackluster scenarios where the map is too repetitive or uninteresting.

Ideally, we would be able to balance the game according to Mihalyi Csikszentmihalyi's State Of Flow to ensure the player is sufficiently engaged in the game. When considering multiplayer we will have to verify that one player does not have a significant advantage over the other, making the game unfair. Based on the deadlines there was not enough time to implement some features for that matter, these circumstances may affect the game in more negative ways. Messages to show to the player were written to make context for the user to understand the history but this was not implemented inside the final game.

The multiplayer feature is implemented as a beta, since it does not have most of the features of the single player game and cannot change level. The game is in general hard this way the user needs to have some engagement and commitment to be aware of sounds and act fast, even so this type of overall gameplay was designed to be this way, may be more frustrating for some types of players.

3 GAME INTERFACE

At the start of the game there is a Main Menu with different options. These options are: *SinglePlayer*, *Multiplayer* and *Quit* [8]. The single player button starts generating a level for a single player. When the multiplayer button is pressed a second menu is displayed. In this second menu we have options to create or join a room with the name given [9]. *Add images in annex and ref here*

While playing the game, the UI has a more *diegetic* style, the controls are showed to the player inside the game world and some of them are only shown in certain circumstances, for example reload is only shown when the player tries to fire an empty chamber [7]. The ammo information is implemented in the character model, the blue lights in the back of Skye represents the ammo that is inside the shotgun. Skye also picks shells in the world to refill her ammo. The amount of shells that Skye carries in reserve shown in her belt. When reloading the blue lights in her back start to light up as she reloads.

At the end of each level an interface is shown to the user displaying the amount of objects pilfered from the level and it's monetary value.

4 TECHNOLOGICAL IMPLEMENTATIONS

Most of the technological features implemented in the *Scavenger Hunt* were created using built-in features of Unity with the help of *C scripting*. In this section will be explained feature by feature how it was achieved.

Mandatory Components

The mandatory segments of the technological implementations were fairly forward to implement. The free moving camera when pressed u detaches from the player and is move-able through the WASD keys. The interesting viewpoints go from 1-4 (1-spawn[12], 2-exit[15],3-hunter[**??**]-4 the hunter's moving destination).Pausing the animations through the P key was implemented by freezing the time scale. And frames per second are calculated using an unscaled delta time.

Plugin/Of-The-Shelf Techniques

We implemented dynamic lights and shadows, environmental maps (seen on Skye's arm and the shotgun), bump maps (almost all level geometry and items have bump maps), particles (used for the shotgun's muzzleflash, in flickering lights, and in room dust) and animator controllers (used for all the moving characters).

Procedural Generation

The level geometry is all placed in a 30x6x30 grid. A grid was chosen to simplify the writing of the algorithm. Rooms are placed at random locations in the grid, and with random sizes, and then the start and exit is placed at each far end of the level. Afterwards, a vertex is created in each midpoint of the room, and we generate a Delaunay tetrahedralization mesh to connect all the vertices. Sometimes, due to room positioning, some rooms cannot be connected, thus they are culled if they have no edges. Next, we use a minimum spanning tree to find a path where every vertex can be visited once. If this fails, we delete everything and start from the beginning again.

Once a minimum spanning tree has been created, we pick a few other edges in the mesh to create loops in the level, otherwise the player would always be encountering dead ends. Once the final mesh has been decided, we connect the rooms to each other using AStar on the grid. Doors are placed in the location where the corridor first connects to a room. Once this is done, we can start placing assets in the occupied positions of our grid.

Rooms are square so its relatively easy assign the correct assets. If a door exists on the wall location, we use a different wall model which won't block the bottom. If the room is touching the edge of our grid, we instead generate walls with windows (this is done only on the edges to ensure we don't look into corridors in the rest of the level). Corridor walls are tricky because we need to take into account which direction the wall is connected in and using the correct prefab accordingly.

Finally, every room is populated with items or monsters (using another grid, to ensure items aren't overlapping or blocking doors), the player and monster are spawned in the entrance and exit respectively, and a navmesh is generated for the level.

Procedural Audio

Since it's important to know where the monster is, our game features procedural audio. Besides our custom sound manager that allows us to perform Unity-made effects on them, pitch shift, and perform 3D audio calculations. We also wrote our own custom low pass filter using `OnAudioFilterRead` to muffle sound effects which occur in other rooms.

Multiplayer

The multiplayer was achieved using *Photon Engine* and using Photon's free cloud feature.The playable mode was divided into two parts one generated locally, and that the player's machine would be responsible for creating.And one created using photon's instantiate feature.Regarding the one created locally we'd aim to to lessen the burden on the internet seeing that the procedural generated map would be the same for both players if the seed was shared, thus the seed was created using the room name. Regarding photon we used 4 different components for the shared assets (Hunter Player) those components are: *Photon View* (added so the assets could be shown in different games), *Photon Transform view*(added so that it would be possible to see the assets moving), *Photon Animator view* (added so the animations would be shown in different machines) , *Photon Rigid Body* (added so that the assets would be collidable).

Monster AI

The Hunter's artificial intelligence has different states. Three states for when not attacking the player. These states are called: *ChasingPlayer*, *Patrolling*, *CheckingSound*. All the AI is implement using the *NavigationMeshAgent* component to find paths to some destination given.

- Patrolling
- CheckingSound
- ChasingPlayer

Patrolling

In patrolling state the script responsible for controlling the monster AI chooses a random position of a room and tells the agent to go to inside that room. If nothing changes the state of the monster he will continue walking to random rooms of the map.

CheckingSound

While in *CheckingSound* the monster goes to the position of where a sound was played. This function of the monster is called by the audio manager and the distance to the sound is checked inside the monster script. So anytime an audio is played and the monster is supposed to hear the sound this function is called. If the monster is close to place where the sound was played then he goes there to check.

ChasingPlayer

This state can happen at any moment, in every frame the monster checks if he can see the player. The monster enters this state when the player is in front of him in less than a 45 degree angle. This calculation is made using the player vector relative to the monster and a dot product between this created vector and the vector that represents the forward direction of the monster. If the player is in front of the monster then a ray cast in the direction of the player to check if there is no object hiding the player. Inside this state there is 3 more sub-states that define the combat situation called:

- Idle - when the monster is not engaging in combat with the player;
- Attacking - when the monster is close enough to the player to attack;
- Staggered - when the monster was shot and staggered by the player;

5 ANNEXES

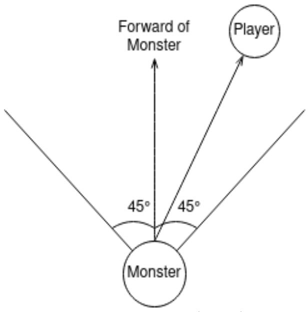


Figure 3: Monster AI Algorithm

Concept Art

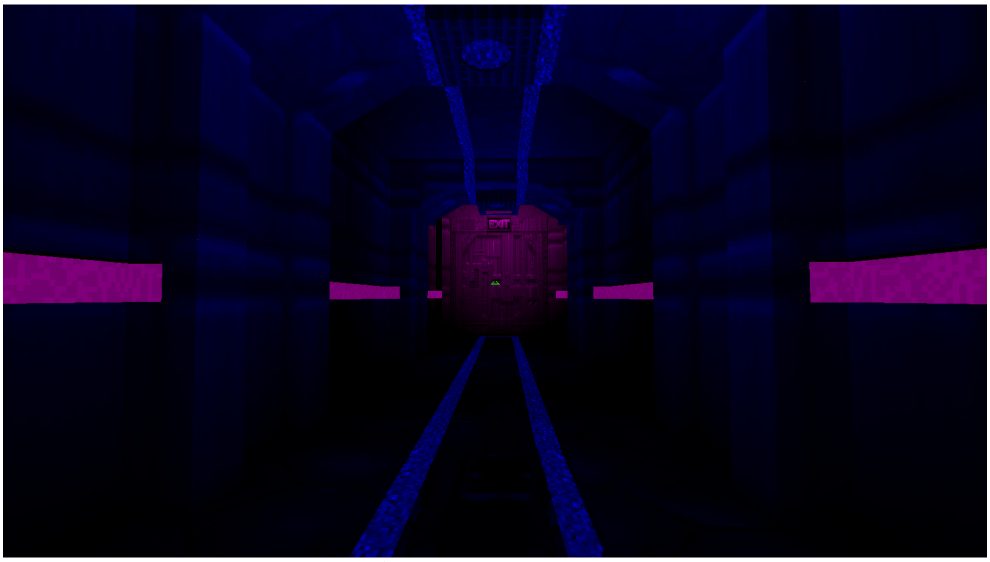


Figure 4: Initial concept art for PATHOS-IV's lighting, put together in Doom Builder

Figure 5: An image from the movie Tron, which features a strong "clean" futuristic aesthetic.

Demonstrations

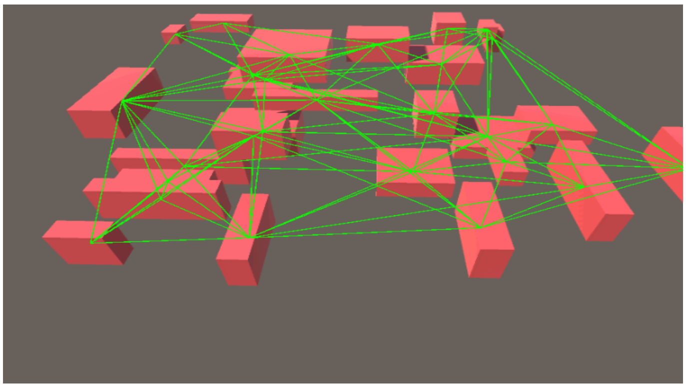


Figure 6: Delaunay Triangulation example

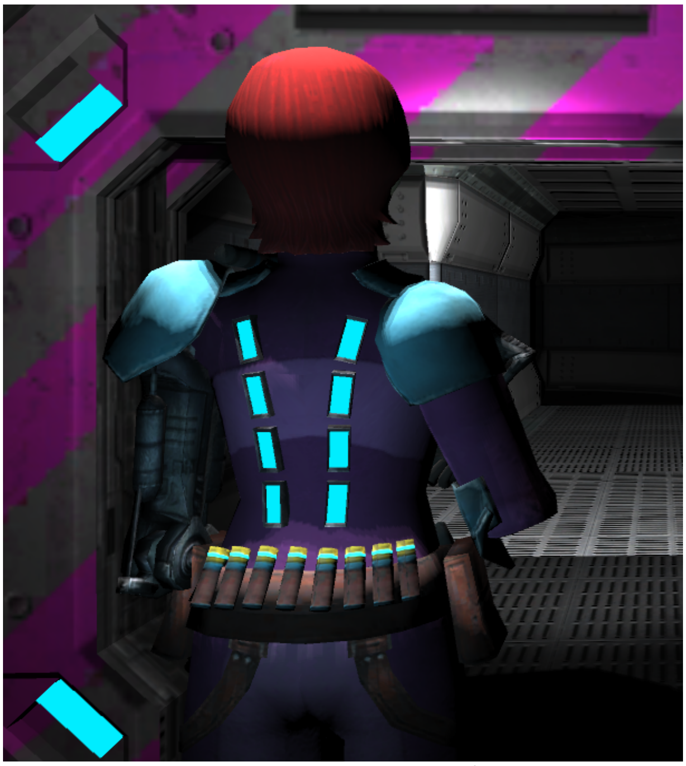


Figure 7: Diegetic Interface

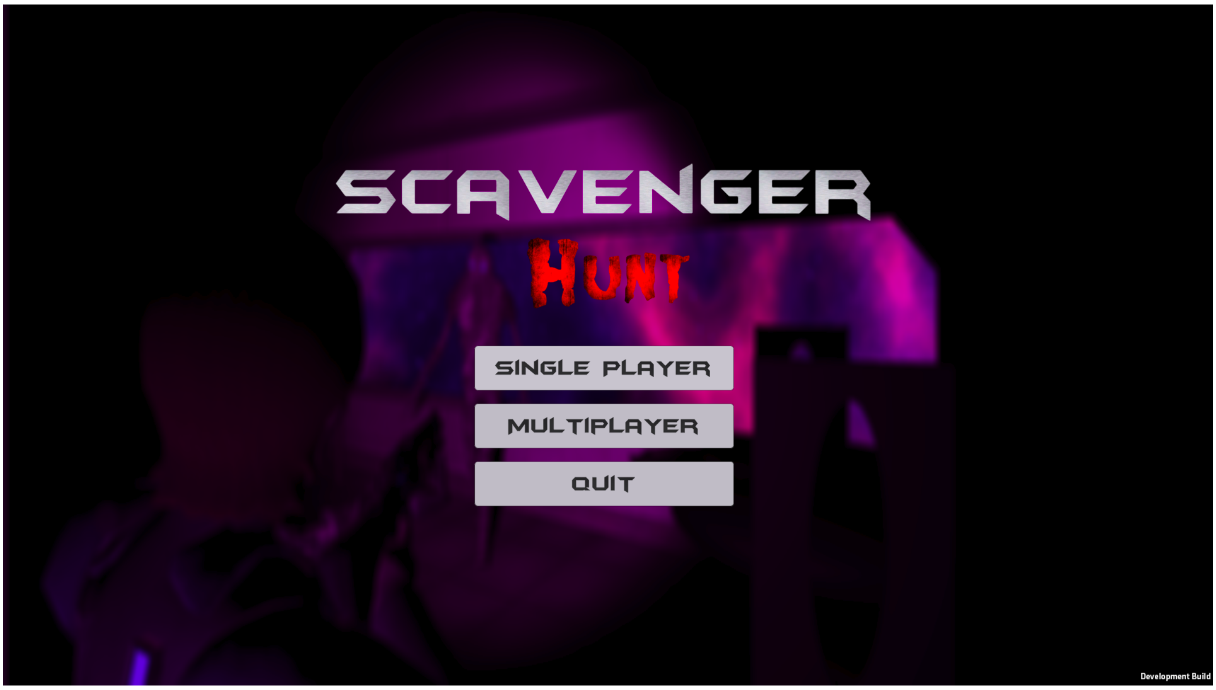


Figure 8: Main Menu

Figure 9: Multiplayer Menu

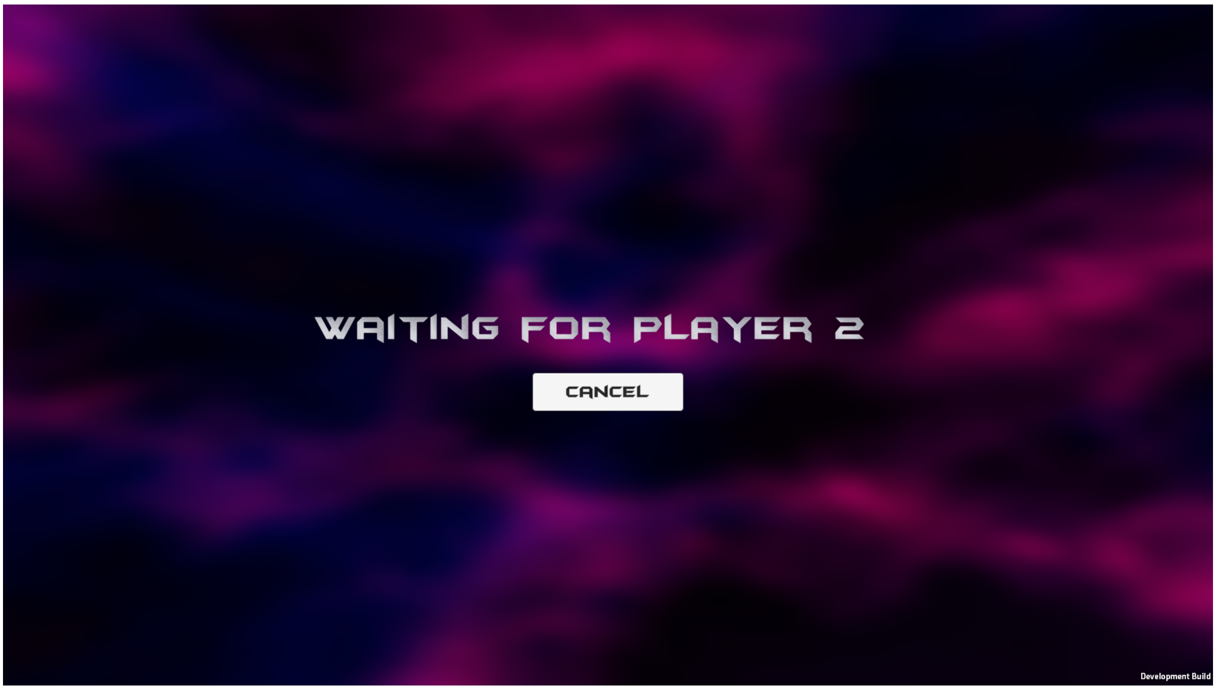
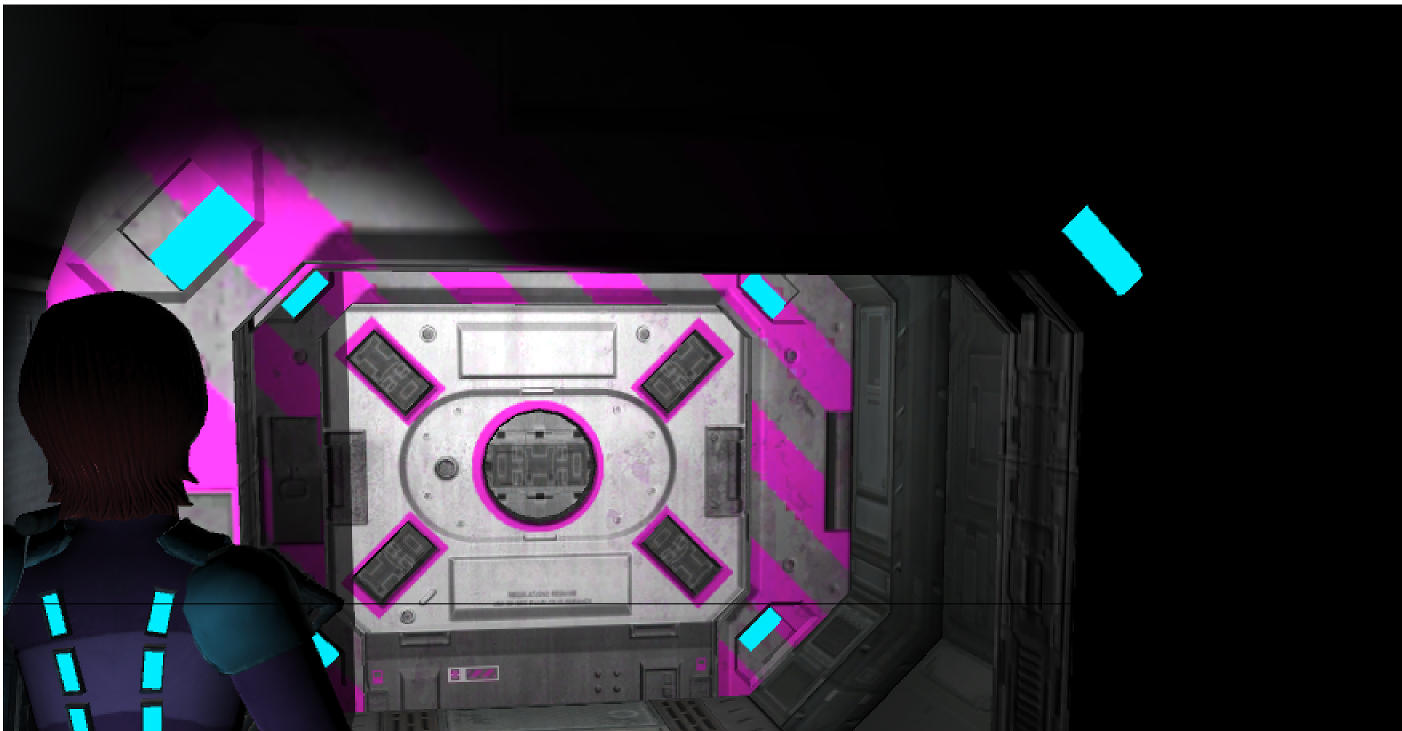


Figure 10: Waiting Screen

Figure 11: Player Character



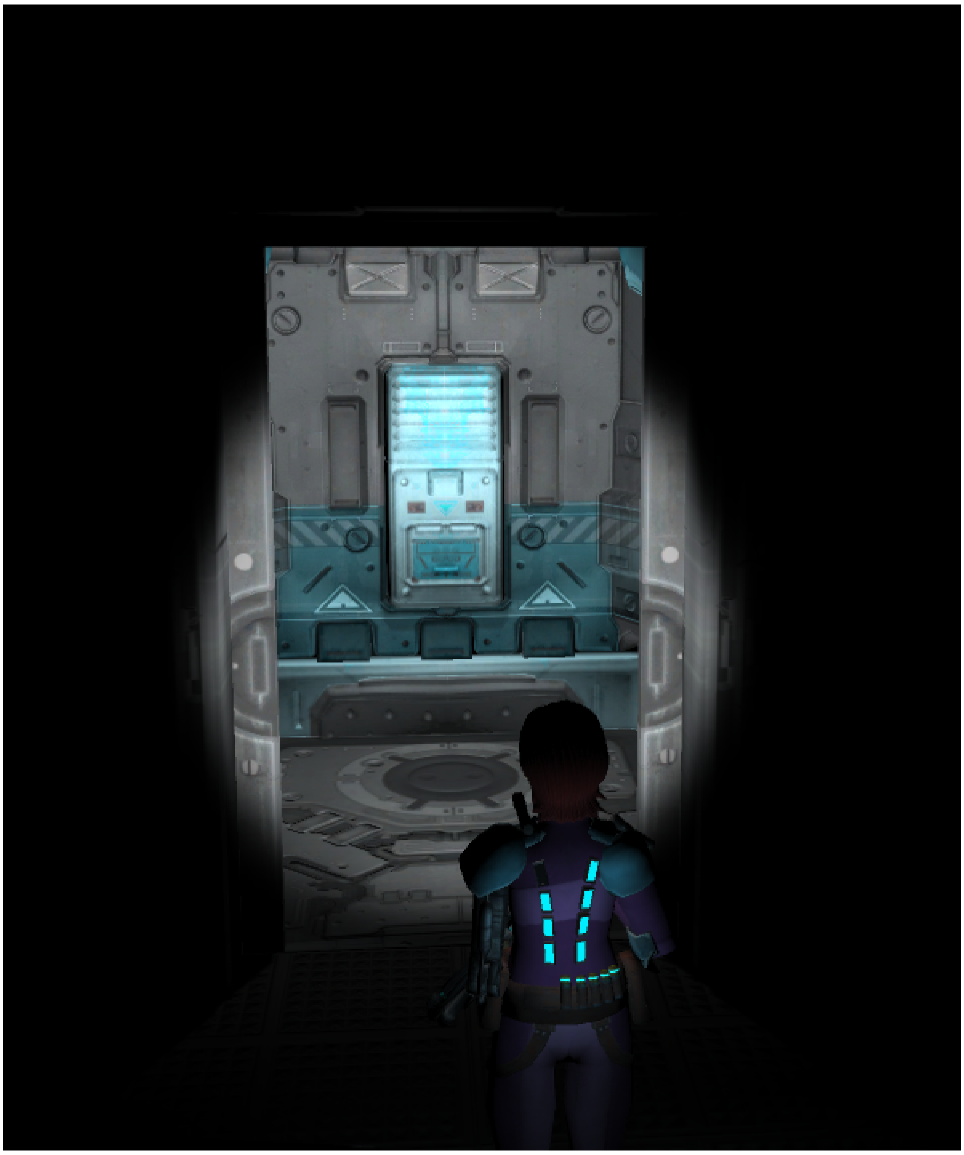


Figure 13: Elevator

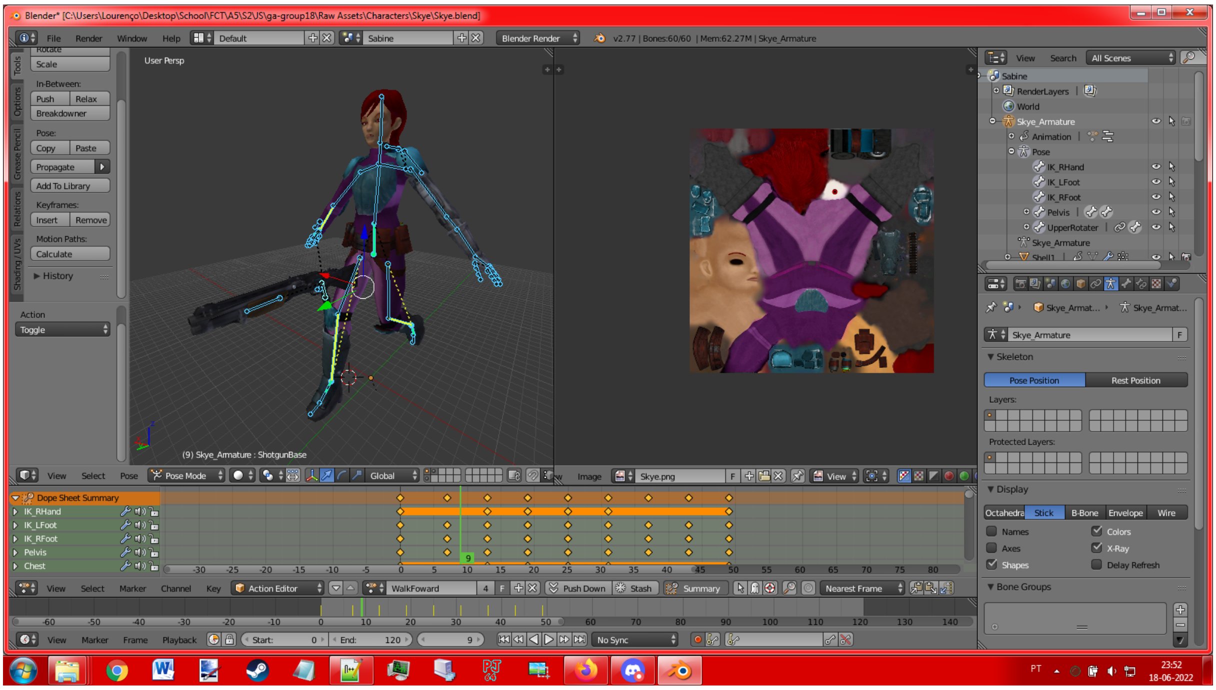


Figure 14: Player Model

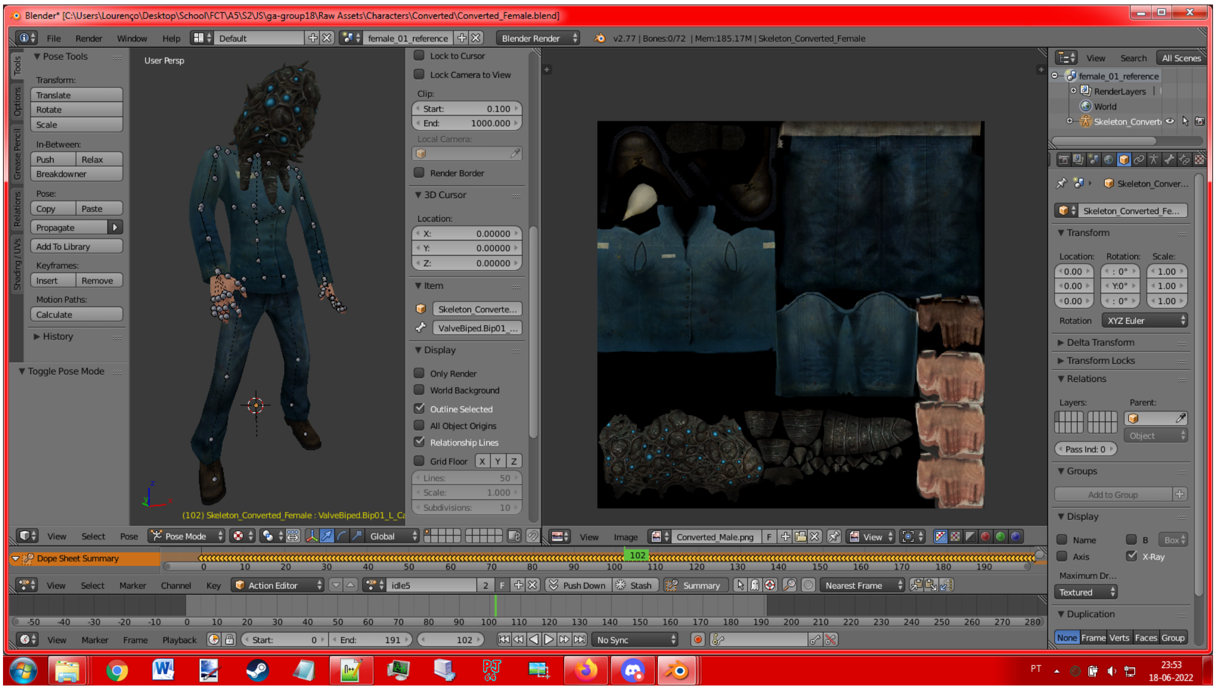
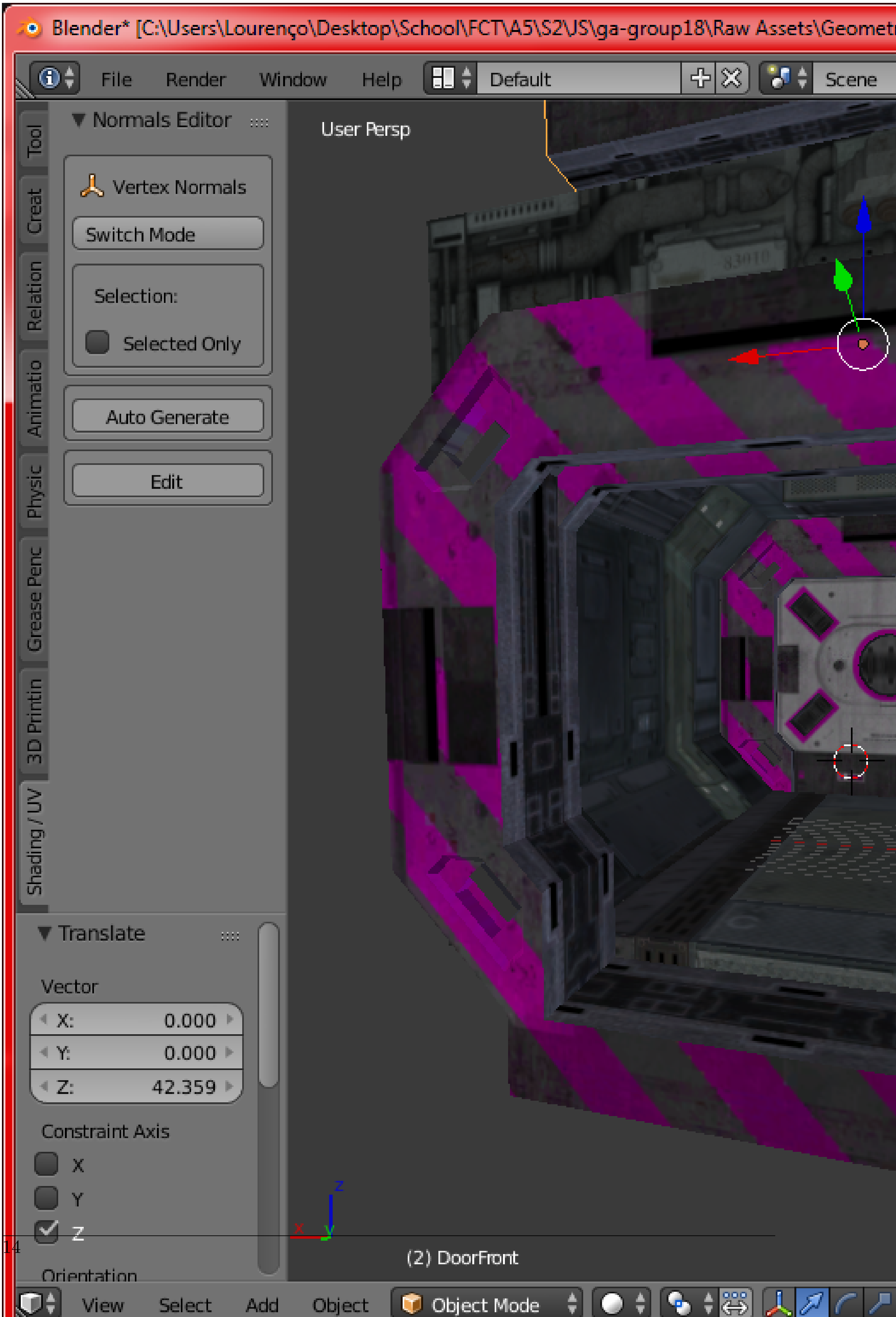


Figure 15: Converted crewmember Model


Blender* [C:\Users\Lourenço\Desktop\School\FCT\A5\S2\JS\ga-group18\Raw Assets\Geomet
File
Render
Window
Help
Default
Scene
Tool
Creat
Relation
Animatio
Physic
Grease Penc
3D Printin
Shading / UV
Normals Editor
Vertex Normals
Switch Mode
Selection:
Selected Only
Auto Generate
Edit
User Persp
Translate
Vector
X: 0.000
Y: 0.000
Z: 42.359
Constraint Axis
X
Y
Z
Orientation
(2) DoorFront
View
Select
Add
Object
Object Mode