Nýr: The Last Stand

Report: First Defense

FENRIR

Maxime SOARES CORREIA
Liam Alexandre ABOUROUSSE
Anita SELVARANGAME
Mathieu DUBRULLE

March 2021

Contents

1 Introduction

1.1 Presentation of Nýr: The Last Stand

Nýr: The Last Stand is a 4 player co-op RPG game, with simple but enjoyable mechanics. The goal of the players is to visit islands created using procedural generation and to fight wave of enemies and to loot as much as they can in order to level up. Whenever they feel like it's been too long outside, they can go back to the main island: Vargfell, where they can buy new stuff, upgrade it, or spend time resting. The game is thought to be highly replayable thanks to the procedural generation and exponential difficulty.

1.2 Book of specification follow-up

Following the feedback we had after the submission of our book of specifications, we modified our schedule. You can see the modifications in red:

Tasks	**1st "Soutenance"**	**2nd "Soutenance"**	**3rd "Soutenance"**
Saves	20%	50%	100%
Character	75%	100%	100%
Items	30%	75%	100%
UI	30%	80%	100%
World / Map	35%	85%	100%
AI	40%	70%	100%
Multiplayer	50%	80%	100%
Website	30%	80%	100%

The following report will let every members of Fenrir talks individually about their experience, what has been done, what remains to be done, and what needs to be done for the next milestone.

2 Individual Tasks

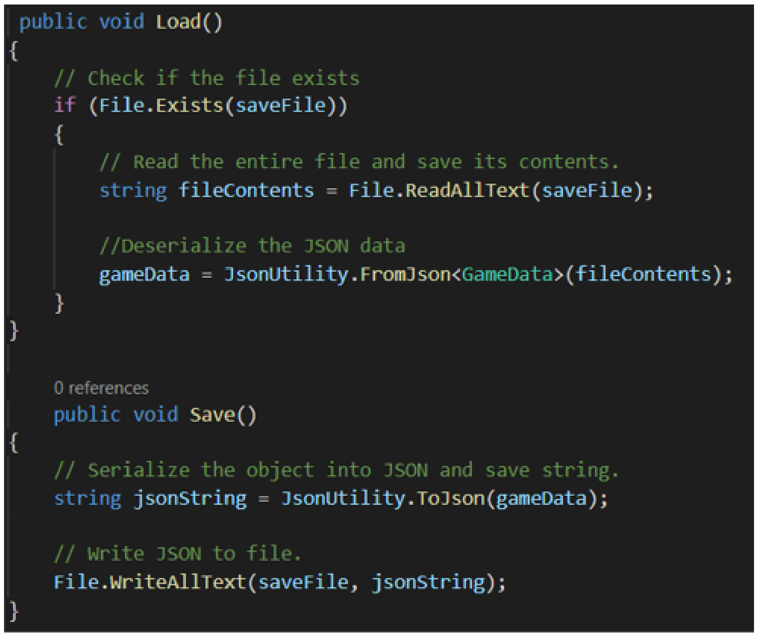
2.1 Saves (Anita)

The saves goal is to save the games characteristics when the player leaves it and to load it when he comes back. Therefore the player will find his game exactly as he left it and will not need to go through levels he already did.

What has been done:

- Player's characteristics:
  The script saving and loading the player's name and position is not implemented yet because there is nothing else to save so it was not useful to the game behaviour. However the script uses serializable data, JSONUtility

class and a class GameData I created. The class JsonUtility handles the work of serializing and deserializing data into and out of a JSON format. It uses the ToJson() and FromJson() methods for these tasks. However, in order to read data and understand the use of values, FromJson() also needs a type to know what it should be trying to read. This is where GameData comes into use with the method.



```
 public void Load()
{
    // Check if the file exists
    if (File.Exists(saveFile))
    {
        // Read the entire file and save its contents.
        string fileContents = File.ReadAllText(saveFile);

        //Deserialize the JSON data
        gameData = JsonUtility.FromJson<GameData>(fileContents);
    }
}

    0 references
    public void Save()
{
    // Serialize the object into JSON and save string.
    string jsonString = JsonUtility.ToJson(gameData);

    // Write JSON to file.
    File.WriteAllText(saveFile, jsonString);
}
```

Figure 1: Sample of the GameManager.cs script

What remains to be done:

I still need to implement this script to the game with a system on the menu to call these save and load functions. This is the part that I'm missing for the first defense but it will be done quickly.

What will be done for the next defense:

- Player's goods and characteristics:
  For the next defense the saves will be extended to all the player characteristics as the money he has, the equipment he possesses and the stats attached to him like his health and his level. All these savings are going to be done as soon as every characteristic is actually implemented in the game.
- World state:
  We will also start to work on the saving of the world state which includes

the enemies characteristic and everything the player has changed in a world. This part will work with the UI part to be implemented as for the rest since it needs a system in the menu to go in a particular world.

A little feedback:
Overall the saves is a bit behind the schedule because it is not implemented yet but this delay will be made up really soon and I'm confident the saves part will be on time for the next defense.

2.2 Map (Anita)

The map aesthetic is in low-poly viking style. I used some assets we imported as it is said in the book of specifications. The village itself was made with the *"Viking pack"* and the *"Adventure pack"* and for the sky I used another assets pack called *"Simple sky"*. This last pack has been really helpful to simulate a sky and it can be usefull in the future to implement a day/night cycle system. The goal for the first defense was to do the main island and start thinking about the procedural generating for the other islands.

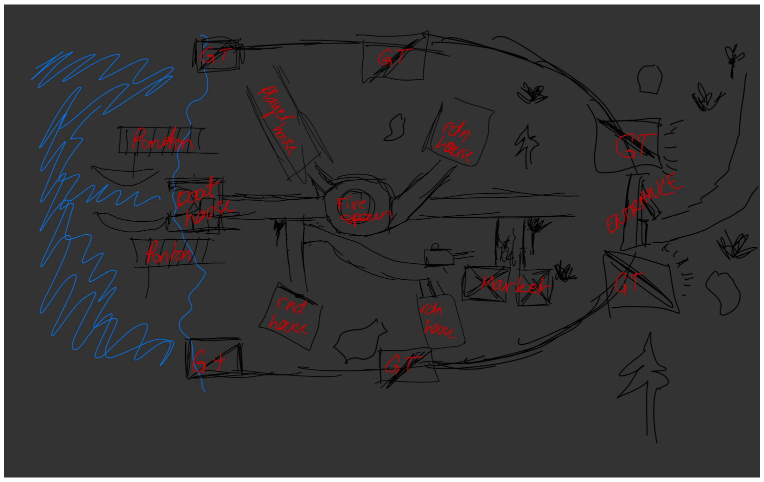


Figure 2: First sketch of *"Vargfell"*

What has been done:

- Vargfell:

  The first island which is called *"Vargfell"* is the one the player will spawn in at the beginning of the game. It's his "homeland" the player will be able to craft some equipment and buy some food there. Therefore *"Vargfell"* is very detailed and thought to be warm.

– The village:

The village was quite easy to do given that I already had the assets but it was still really time consuming. Placing every object meticulously and making sure that everything was homogeneous was the real challenge.

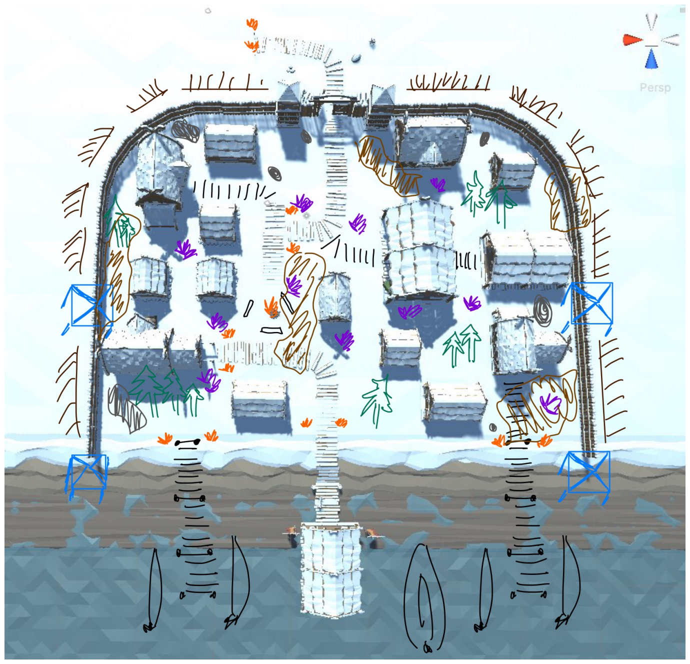


Figure 3: Sketch of the map details using what was already done.

– The environment:

The island name means ”Wolf fangs” in old Norse so I tried to make the mountains based on this shape although the assets were not auspicious to this shape. The forest surrounding the village was made with the poly brush but it created some issues like the trees were levitating and were way too inclined. So I had to shift everything by hand. The sea was simply made with the assets but we're thinking about upgrading it for the final defense.

– The lightning:

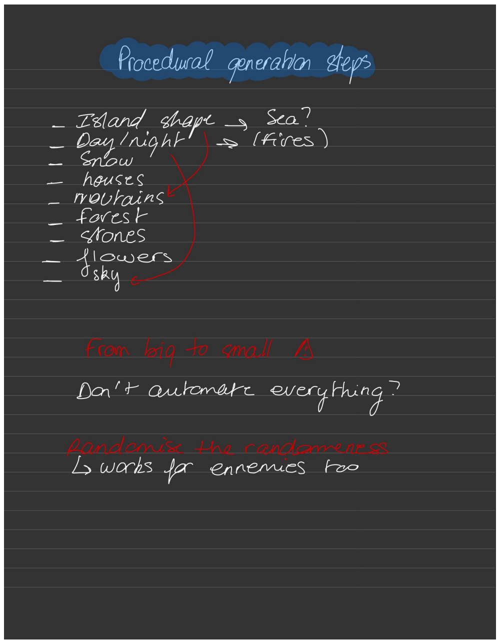
The lightning was more intricate to set up than I thought. I had to change the lightning settings of the map and put two main lights to make it look realistic. I also added several lights for the torches and the campfire to make the map look more alive.

– The NavMesh:

I used a NavMesh map to define which component of the map are walkable or not because obviously the player won't be able to walk through the objects or on things like clouds. I also made the mountains not walkable because we didn't want the map to be to vast and the goal is not to explore the village map but to explore other islands.

- Procedural generating:

The procedural generating is something that needs to be broke done in little problems. Therefore, we have already defined the different elements that are going to be on the islands created by this process.



Procedural generation steps

- Island shape → Sea?
- Day/night → (Fires)
- Snow
- houses
- moutains
- forest
- stones
- flowers
- sky

From big to small ⚠

Don't automate everything?

Randomise the randomeness
↳ works for ennemies too

Figure 4: The different elements that has been differentiate for the procedural generating.

What remains to be done:

The map is a bit ahead of the schedule the village *"Vargfell"* is almost done like planned. I only need to add some details but those details were not needed for the first defense. I also started the researches about procedural generating and prepared the terrain as planned. Thus everything that was expected for the defense is done for the map.

What will be done for the next defense:

- Vargfell:
  - Flame Shader:

    The torches and the campfire lights are done but creating the flame is a bit fastidious because I need to install the right packages and I don't master the subject yet.

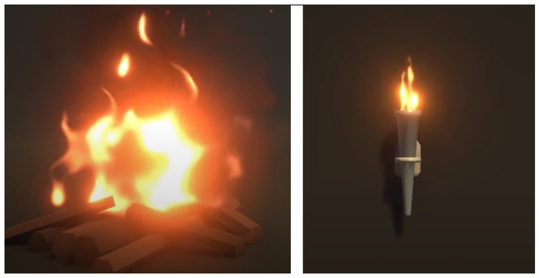


Figure 5: Screenshot of what the flames are going to look like.

  - Non-Playable characters:

    I will also add some non-playable characters by the next defense. These characters will not really take part in the game, they will by there as a "decoration" to make the village look more alive and inhabited.

- Island n°1:

  The island n°1 is going to be a "tutorial" island to help define the hierarchy of the different steps for procedural generating and to start using it for the game while the procedural generating is not ready.

- Procedural generating:

  By the next defense I hope to have the hierarchy of these element (from the biggest to the smallest), the prefabs that are going to be spawn randomly and the code that make them spawn.

A little feedback:

The only real problem encountered while working on the map was using the polybrush, I need to inform myself more on the subject. Overall making the map was a great experience even though it was really time consuming it also was really amusing and I really enjoyed working on it. I am a little bit disappointed that I didn't have time to do the flame shader before the defense but I am still in agreement with what I planned the map is done at 35-40%. We are only missing some details on Vargfell but we started to work on the procedural generating.

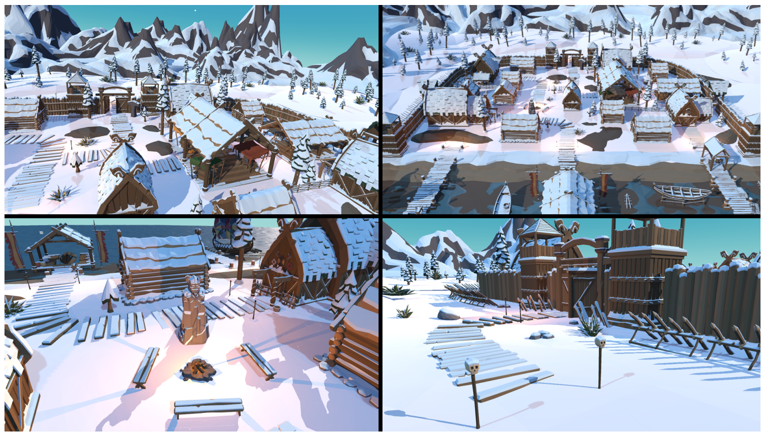


Figure 6: Screenshots of the version of *"Vargfell"* presented in the defense

2.3 Player (Maxime)

The Player part main goal was to have a functional player able to walk, to chose the best path and to reach a destination with animations. This player needed to be the frame for most of the other tasks. In fact, a Player was needed for the Multiplayer, the UI, the AI, and of course the World. This part was the first one with the website which we really focused on. It needed to be ready fast, and it was.

What has been done:

- Player Movement:

The first choice here was to chose how will the player will move through the world. First, the group wanted a more classic system, using the well-known "WASD" (or "ZQSD" with french keyboards), but after appropriate consideration, it wasn't suitable with what we were looking to do with the game. Instead, we decided to go for a point-and-click system. I began my research really early and had found multiple ways to succeed. At first, I used a not really optimized locomotion system. Every time the player decided to go somewhere, it created an object and whenever the player reached it, it destroyed the object.

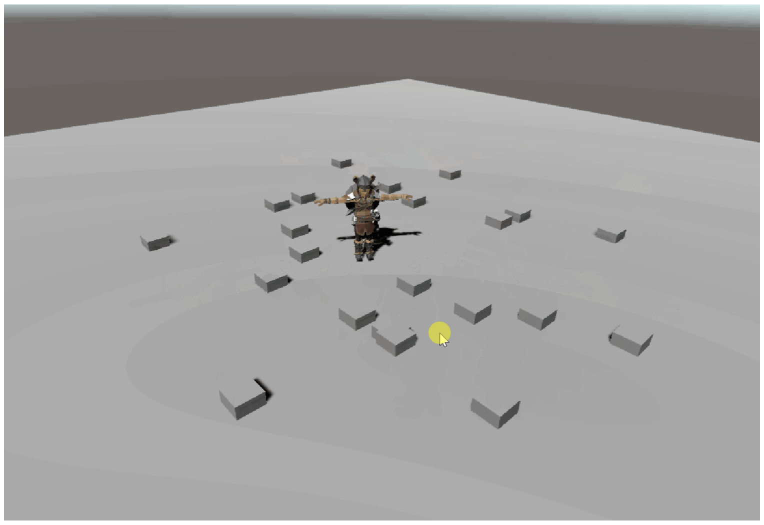


Figure 7: Screenshot from an early build of the player movement, using pointers.

Finally, I decided to use an built-in system from Unity: Navigation Mesh. After a bit of research, I finally managed to create a smart movement system. If the user clicks on a A point, the character will, alone, finds the quickest route to get to its destination. Using Navigation Mesh, we can easily define ”Walkable” and ”Not Walkable” areas.

Following this system, I created a Target lock system. While left mouse button click allows the user to chose a destination for the player to reach, the right mouse button click allows the user to focus a targetable object (i.e. an enemy, a container, or an interactable). This way, the user don’t have to click everywhere in order to follow a moving target for instance. Instead, the user just have to right click on the target, and the character will travel to its destination and follow it until it reaches it.

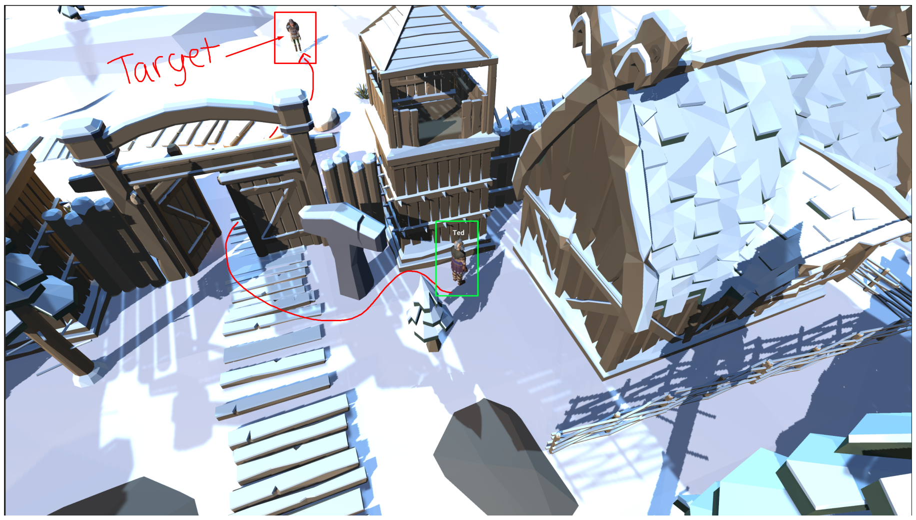


Figure 8: A quick schema to explain how the smart movement works. Ted aka the Player (green box) will choose the quickest route (red line) to get to the target (red box).

In order to have something else than just a character T-posing, I had to animate it. Again, with a bit of research, I finally managed to get something which looked exactly the way I wanted. I downloaded animations from mixamo.com, and then I used a bit of maths to get a float between 0 and 1 representing the current speed of the player, updating at each frame. Using Unity Animation Blend Tree, I managed to play animations depending on the speed.

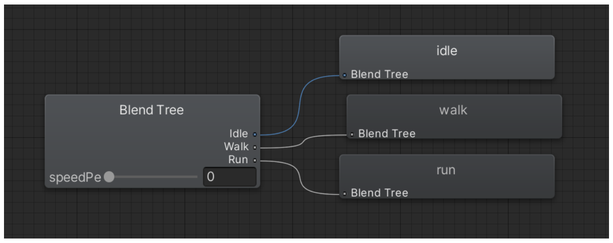
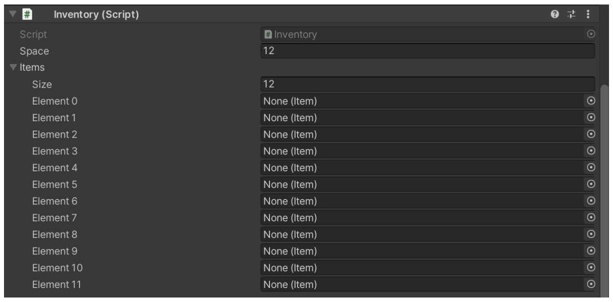


Figure 9: Screenshot of the blend tree. Depending on the float speedPercent, animations are blended to look more realistic. For example, when at 0.75, it blends walking and run together to look like a real movement.

- Inventory:

I also worked on making a working inventory, just to have an idea of what it will looks like in the game. This mechanic is closely linked to the UI part, which I also worked on. Every player has an instance of an inventory, which was primordial since it's a multiplayer game. I had to make sure that players didn't share the same inventory. The inventory is also very modular and we can easily modify the slots in it.



Inventory (Script)

Script	Inventory
Space	12
Items	
Size	12
Element 0	None (Item)
Element 1	None (Item)
Element 2	None (Item)
Element 3	None (Item)
Element 4	None (Item)
Element 5	None (Item)
Element 6	None (Item)
Element 7	None (Item)
Element 8	None (Item)
Element 9	None (Item)
Element 10	None (Item)
Element 11	None (Item)

Figure 10: Screenshot of the Inventory script inspector. The "Space" field can be easily modified to any value. Basically, it's an array containing items, so it's easily accessible and flexible.

The inventory is displayed in-game (see UI) by pressing the corresponding hotkey, using Unity new Input Manager. This way, it's easy to make possible modifiable inputs. Whenever an Item is picked up (see Items), it will appear in the inventory and disappear from the scene. If the user clicks on the item, it will use it whatever it does (healing, restoring stamina, equipping sword, whatever.).

What remains to be done:

• Overall:
From what I planned when we first wrote the book of specifications, nothing remains to be done. Generally, I'm pretty happy with the work done on that part.

What needs to be done:

• Player combat:
Now that AI and Stats are implemented, and that Player Movement is fully operational, we need to create a combat system. It will handle different style of weapons: Sword, Axe, Shield, maybe spear, and even Bow with projectile physics, with corresponding animations.

• Combat tweaks:
If the Player Combat system is on the right track and if I'm satisfied enough with how it looks, I will definitely add spells to spice up the game. Those spells will cost stamina, and will have different behaviors depending on what weapon the player is using. I will also love to implement a skill tree to unlock those spells.

A little feedback:

Working on that part was a really fun experience. I learned a lot, and even if it was kind of hard during the first weeks, I managed to be at ease with my working environment (Unity and Visual Studio 2019). Animations were a pain at the beginning, but at the end, I'm again at ease with it now. The feeling of doing something tangible and be able to see what I was doing in real time was what helped me to keep the rhythm and to start working very early on.

2.4 UI (Maxime)

Even if UI wasn't an important part for the first defense (about 30%), and since I was ahead of the schedule for the player part, I allocated a lot of my time into it. User Interface is the whole graphical part allowing the user to interact with our game, it includes: Menu, Health bar, Stamina, Inventory, Gamertags, various displays, effects, etc. This part was mainly done using Unity UI and Photoshop CC 2021.

What has been done:

- Main Menu:

The main menu is basically where the player can press "PLAY" to start a game, "OPTIONS" to change the settings of his game, or "QUIT" if he wants to leave us. I've gone through a lot of designs before having a menu that didn't feel wrong. Here is the first design:

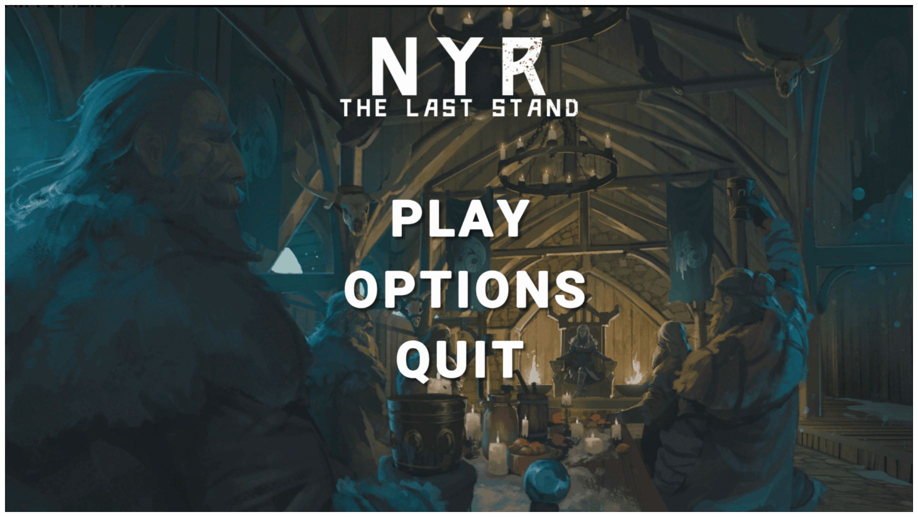


Figure 11: Screenshot of the very first menu. The "PLAY" button spawned the player on a test scene, and didn't include Multiplayer. The animations on the buttons were very fat, and didn't look that well.

Unfortunately, this first menu couldn't handle multiplayer because of its structure. Moreover, the texts fields which I were using were low-res one and weren't fitting on a bigger screen (i.e 4K resolution). I needed to find something else, which we could use on any resolutions. I start designing a menu on Photoshop and then I transposed it on Unity UI. I also decided to go for more visible animations, which felt, again, more user-friendly. Finally, this second design looked like this:

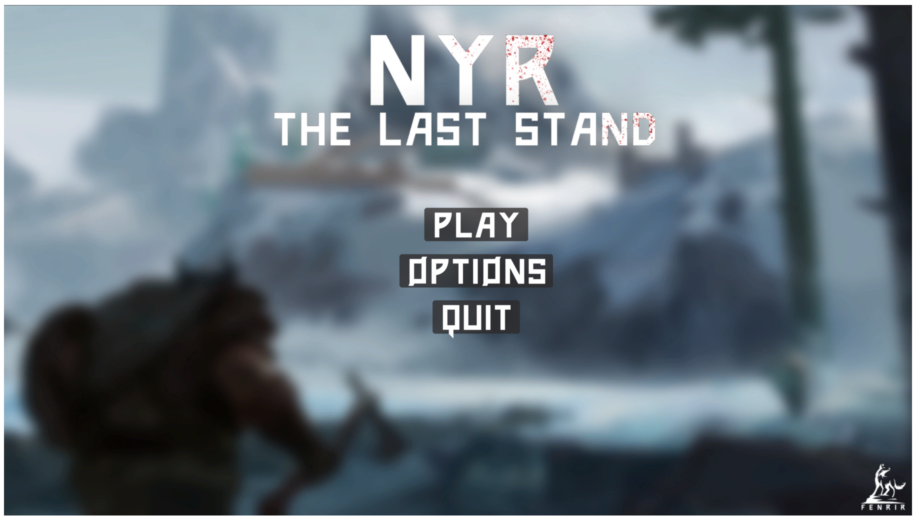


Figure 12: Screenshot of the second menu in 4K resolution.

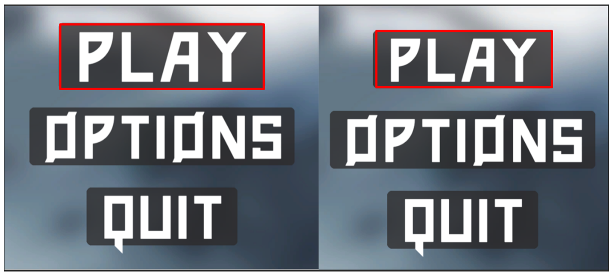


Figure 13: Screenshots showing how the animation works. When hovering with your mouse, the button scales up to be more visible, and kind of bounce.

As said previously, I also needed to be able to create a multiplayer game on this menu, and even join one. To do so, I created a new menu which is enabled whenever the user clicks on ”PLAY”. This menu allows the user to input an Username which needs to be at least 3 character long. When the username is valid, the ”CREATE” and ”JOIN” appears on the screen, allowing the player to chose what he wants to do from here.

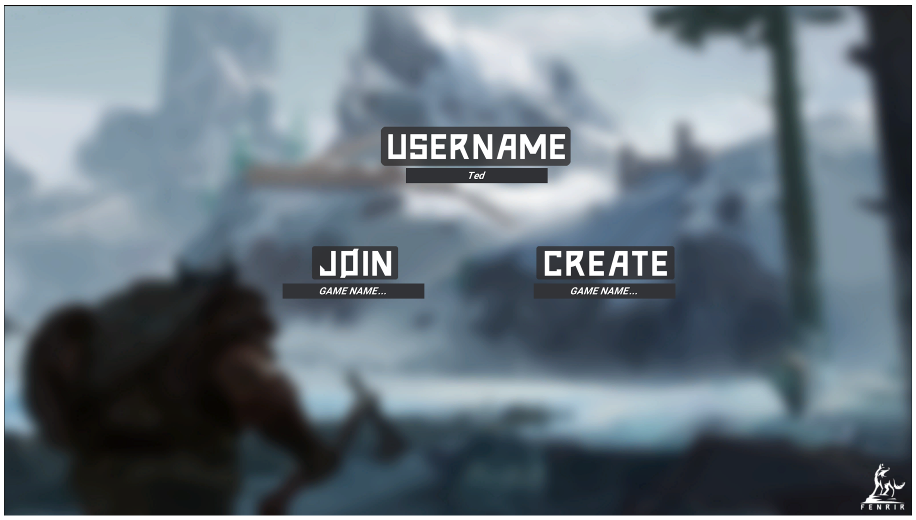


Figure 14: Screenshot of the Connect menu. When the username is valid (Ted in the example), the user can "CREATE" or "JOIN" a Multiplayer game.

For the needs of the project, this menu will be upgraded and a new design implementing our newest functionalities is already being worked out.

- Loading Screen:

To ensure that every players load correctly the same, I had to make an asynchronous loading. Basically, it means that if Player 1 has finished loading, he can already start playing, while Player 2 3 are still loading the map. To make this kind of loading was mandatory for the implementation with Photon, else, it will have caused a lot of issues. The "Scene Loader" prefab was made to be easily applicable on any scenes. It was a pain to make at first, but now we can reuse it whenever and wherever we want. The "Scene Loader" allow a lot of customisation, i.e the name of the scene which is loading, tips at the bottom of the screen, a loading bar, audio, etc.

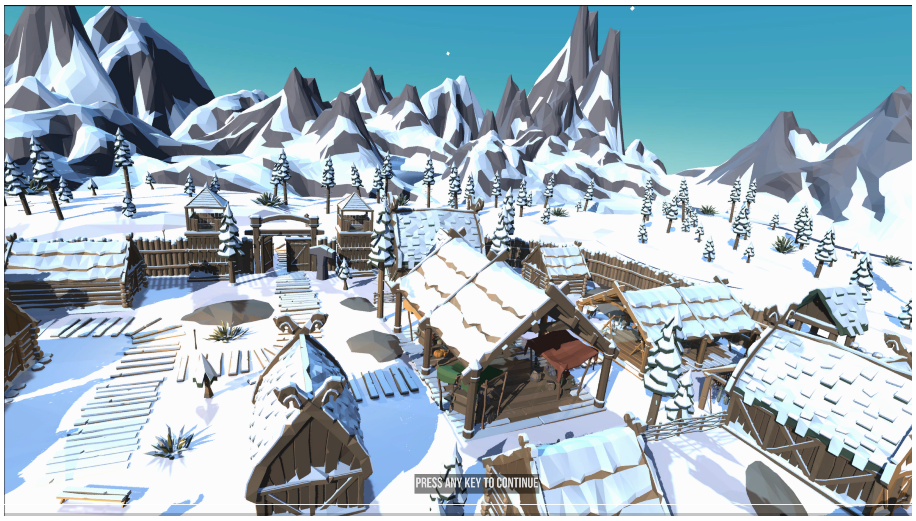


Figure 15: Screenshot of the loading screen. When the loading is done, a "PRESS ANY KEY TO CONTINUE" appears at the bottom of the screen, indicating the players what to do.

To finish with this part, I also implemented a "Spawn Menu" which makes our lives a lot easier when developing. In fact, the "Spawn Menu" appears when the loading is done and a key pressed. A big "SPAWN" button is now present, and whenever a Player clicks on it, it instantiate a PlayerPrefab in the scene, syncing it with other instances of Photon (basically with the other players).

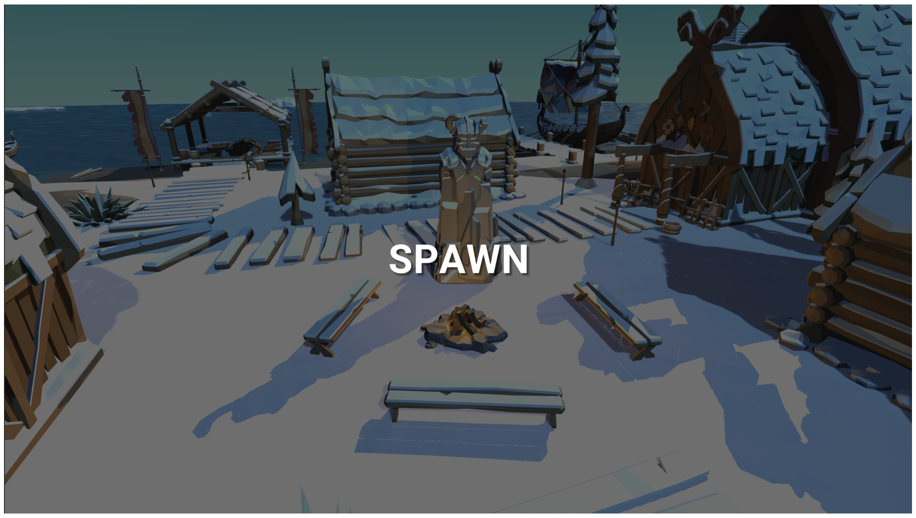


Figure 16: Screenshot of the Spawn Menu.

- Player UI:

Not much work were done on this part since it weren't necessary for what was planned. The only thing done was for clarity and it's the gamertag. Basically, it's just a Text floating above the player's head displaying his chosen username, with a specific color. Whenever the player rotates the camera, the Gamertag follows it in order to always face the Player camera.

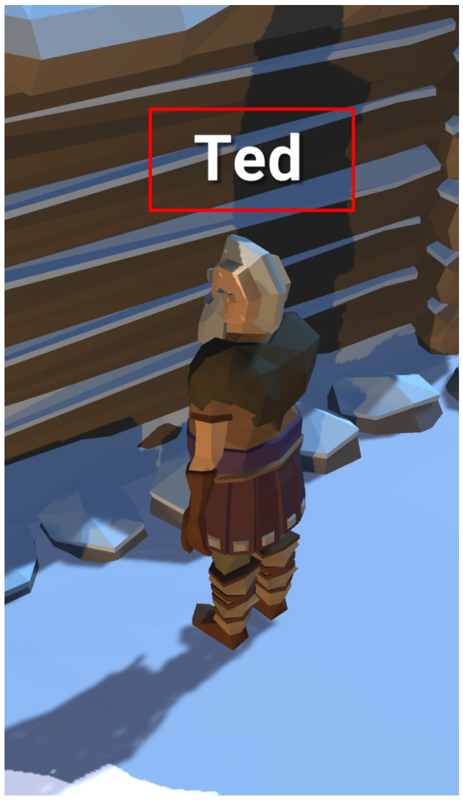


Figure 17: Screenshot of the Gamertag.

The other thing which is pretty important for the behavior of the game is the Inventory. I've done a simple inventory tab on Photoshop which is probably going to get modified for the next defenses. Here is a glimpse:

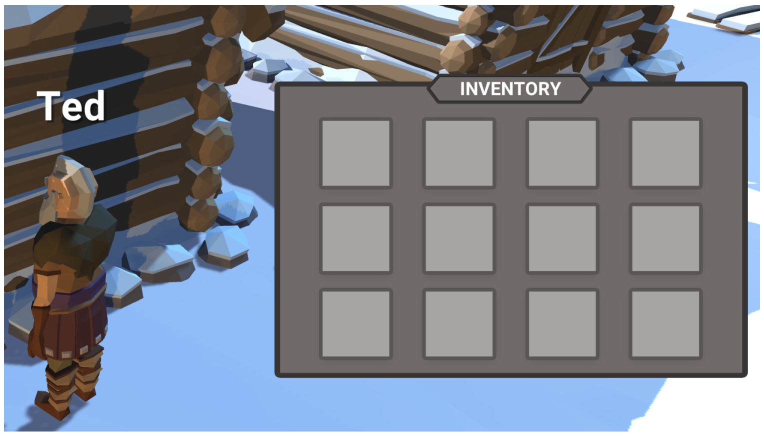


Figure 18: Screenshot of the Inventory.

What remains to be done:

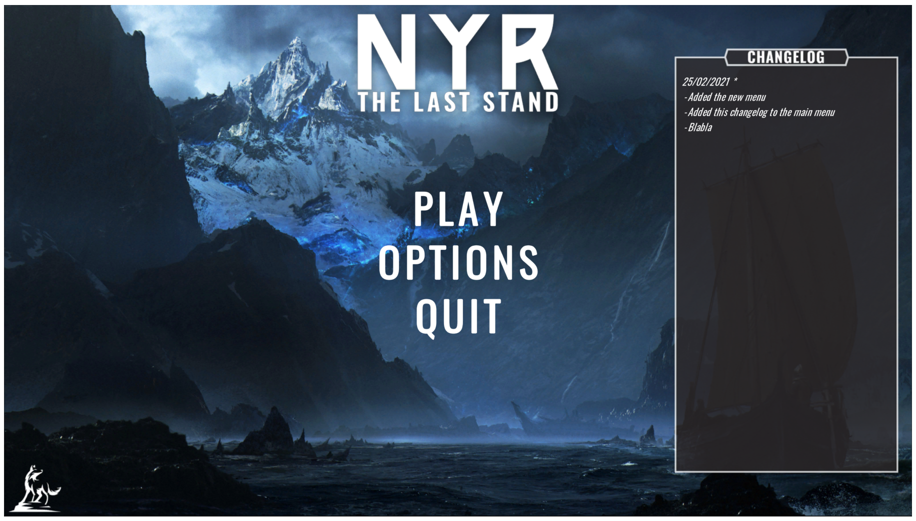
- Overall:

I've done a lot more than what I was supposed to do, so I will say nothing remains to be done to achieve the goal I set myself.

What needs to be done:

- Main menu:

The actual main menu is great for what it's in the actual version of the game but it will soon become obsolete because of the Saves system and the future Multiplayer. This is why I need to review the design again, in order to make sure it suit the game. I've already start working on a menu looking more professional, and I added a changelog to it, which will basically allows the player to know what has been updated recently.



25/02/2021 *
-Added the new menu
-Added this changelog to the main menu
-Blabla

Figure 19: Screenshot of the third menu design. It includes the changelog and a brand new logo.

- Player UI:

I need to modify a bit the gamertag in order to see it better on white surfaces (i.e snow) by adding a simple black outline. In the near future, I will have to add a display for the health bar, and the stamina for the player. A simple Character menu which will allow the Player to see what he has currently equipped and his various stats (i.e health, stamina, armor, damage). If I have the time, I will probably add better displays to see what target we're locking, or even displaying the other players stats in your lobby, RPG-style.

A little feedback:

I had a lot of fun working on this part too. This part allowed me to totally unleash my creativity, and me, as a RPG player, couldn't be more eager to work on creating what seems like the perfect interface for me. Again, I had a lot of trouble getting things right on screen during the first weeks but I managed to get a pretty good result.

2.5 Items (Liam)

We have implemented an object system into Nyr. Most of its elaboration is based around C# Scripts. Yet the graphics are still temporary. To develop this part, we worked closely with the inventory and multiplayer systems. In fact, they are intrinsically linked. In short, every time one of the player picks up an item it is stocked in the inventory. Then, from the inventory, the player can equip the item in one of his equipment slots. Once it's equipped it will modify his stats.

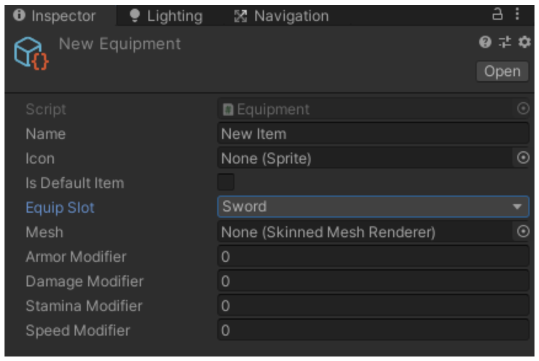
What has been done:

- Picking Up Items:

Picking up items was made instantaneous thanks to a C# Script called ItemPickup.cs that says whether or not the item can be picked up the player. If the item contains this script, then the player will be able to store it inside his inventory.

- Equipment:

For the first defence, we focused on the equipment side of items which meant working on the inventory side of things and the statistics. We decided that there will be 4 statistics that can be modified by pieces of equipment. These 4 stats are Armor, Damage, Stamina and Speed. For now, every equipment piece will be able to fill one of these 7 Slots : Sword / Shield / Necklace / Right ring / Left Ring / Armor Rune / Damage Rune.
In order to create faster pieces of equipment, we created a new piece in the asset menu called "Equipment" that load a fresh piece of equipment with all the components we need to modify by hand.



Script	Equipment
Name	New Item
Icon	None (Sprite)
Is Default Item	
Equip Slot	Sword
Mesh	None (Skinned Mesh Renderer)
Armor Modifier	0
Damage Modifier	0
Stamina Modifier	0
Speed Modifier	0

Figure 20: Screenshot of freshly created piece of equipment

- Statistics:

Statistics and Equipment are two complementary part of the work. When launching a fresh instance of the game; the player spawns with base stats. It's supposed base stats are 100 Health, 100 Stamina, 1 speed, 1 Damage and 0 Armor. For this first defence we only managed to go as far as creating those stats and modifying the stats for armor and damage. Armor will have a direct

impact on health. For example, if the player takes 10 damage but is equipped with a rune of protection that gives him 5 armor, he will only lose 5 health. We subtract the damage received by the amount of armor the layer is equipped with.
Yet, in this part there were two problems. "What if the players has more armor than the damage given ?" and "What if his health is inferior or equal to 0 ?" To resolve those problems we had to create a moment where the players dies (for now it's only signaled in the unity console). And if the damage is inferior to the amount of armor, the player could have received Hp since subtracting a negative number is a positive sum. We then had to create a stopping case to prevent this problem.

What remains to be done:

We need to implement a progression system based on experience you can collect from accomplishing tasks and killing enemies. Through the experience and progression, we must implement stats that evolve proportionally as the player progresses in the adventure.

What will be done for the next defense:

It's capital that by the next defense I can present models for my items, and that the sword and shield are shown on the player model while the game is played. It would be great if stats evolve at the same rate as the player.

2.6 Multiplayer (Liam)

At first, the multiplayer we wanted to implement into Nyr was supposed to be built around the asset called Mirror. It created a local server on a machine and other machines could connect and join. Yet, there was a major problem: the server was hosted directly by a player and not by a third-party server. In fact, this meant that the player had to open the ports of his IP so other players could connect. They could also use Hamachi to emulate a LAN. We found that really annoying because it was combined with optimisation issues. The result wasn't intuitive and that's why we worked on the multiplayer using the brand new Photon Engine. The main product they propose is PUN. A reliable multiplayer integration specifically designed for Unity. Compared to Mirror, PUN is more straightforward in it's way to create and join multiplayer games because all of them are located on the Photon cloud which guarantees low latency and no punch-trough issues.
What has been done:

Players can create a room that they give a unique name by clicking on the button "Create" on the menu and the next scene is asynchronously loaded.

Then, other players can join by entering the name of the room previously created under the button "Join" on the menu. For every player spawning, he has his own player prefab with all the scripts needed and it's own camera that the players can individually control.

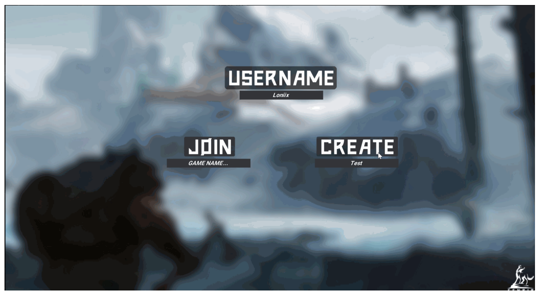


Figure 21: Screenshot of the connection menu.

What remains to be done:

We are satisfied with what have been done during those first months. There is still a lot to do, but we are on time.

What needs to be done:

For the next defense, we plan on adding a "server navigator", allowing the users to see which games are currently on, and how many players joined that game. It is way more convenient than entering a game name in order to join since mistakes are easily made, and that the name input is case-sensitive. We also plan on making a simple but efficient chat, so players can communicate when playing together.

2.7 AI (Mathieu)

What has been done:

- Added the detection range
- Added the movement
- Added the facing behavior (the AI always faces the player)
- Added the "focus closest player" behavior
- Optimized this behavior :

To detect and move toward a player, we use the "GameObject.FindObjectWithTag()" method which is very expensive, this method browse the list of all the objects on the scene and returns an array with all the objects which have the "Tag" placed in parameters. In a first version, this method was called once per AI and per frame, and we changed the code so it is called once per frame in a different script where all the AI on the scene collect their data from.

- Added a model and animations

What remains to be done:

- implement a statistic system
- implement a health system
- implement fight behavior
- implement animations for the fight
- create different classes of enemies
- create a "Boss" class

What needs to be done:

- implement a statistic system
- implement a health system
- implement fight behavior
- implement animations for the fight

Commentary:

We might be a bit late on the AI compared to what was planned on the book of specifications (40 percent). This is due to the fact that the AI has to be developed at the same rate as the Character and the UI because the AI interact with the players, and we didn't thought about it when we redacted the book of specifications. However we are satisfied with the actual advance of the AI and we are confident for the realisation of the next tasks.

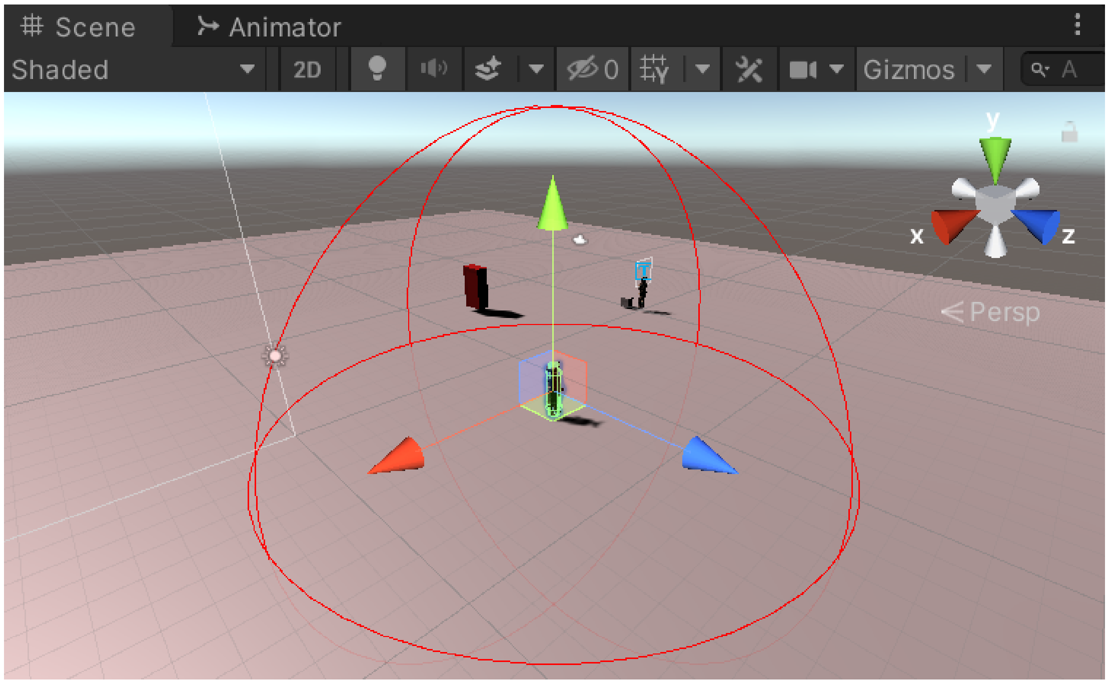


Figure 22: Detection zone

A little feedback:

There is a lot of resources on the internet and specifically on YouTube for the AI. By following tutorials, i manage to create a basic AI easily (which can move and has a detection range). However i encountered a major problem : How is it possible to access to the player object if there is none on the scene ? (the player are instantiated when they connect to the room). After some researches, i found the GameObject.FindObjectWithTag() which helped me a lot to access to the player object. I chose to give to the AI a logical behavior : it follows the closest player in his range of detection. Then Maxime helped me with the Animation of the model. It glitched at first as i kept applying the root motion to the animation but i fixed this.

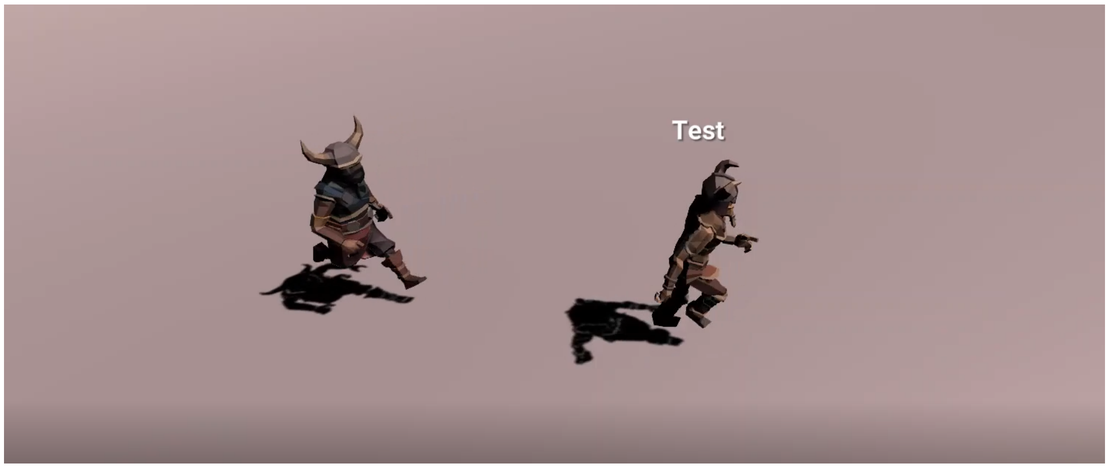


Figure 23: AI chasing the player

2.8 Website (Mathieu)

What has been done:

- The three page and their links at the top of the website
- Assets added
- Customization of the character font
- Screenshots of the game
- Links to the resources added

What remains to be done:

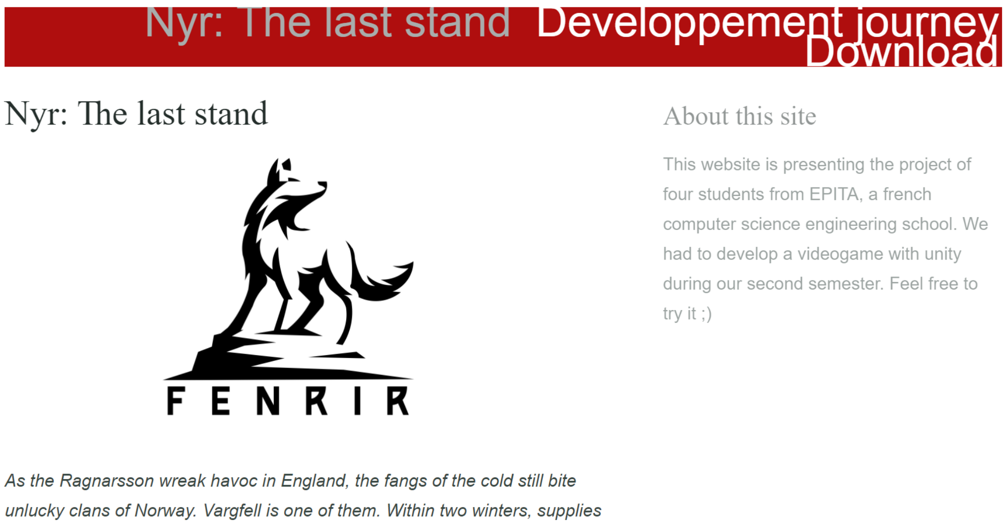
- Modification of the header so the pages links fit
- Modification of the header
- Host the site on GitHub
- Complete the information on the game and add the different links
- Change the screenshots in function of the advance of the game

What needs to be done:

- Update of the information and screenshots of the game
- Hosting the website
- Modification of the header

Commentary:

I think that I'm really in advance on this part, the last thing which could take much time is the modification of the header. As I'm developing the website with Jekyll, it will be easy to host the website on GitHub, and the updates of the information about the game on the website are really easy to do. It is said in the book of specifications that the website had to be finished at 30 percent for the first defense, i would say it is finished at around 60 percent. Also a small precision, for the moment the website is hosted locally, we wish to host it on GitHub when the work done will be satisfying enough to have something presentable to show, that is why the website will be hosted on GitHub for the next presentation.



Nyr: The last stand

About this site

This website is presenting the project of four students from EPITA, a french computer science engineering school. We had to develop a videogame with unity during our second semester. Feel free to try it ;)

As the Ragnarsson wreak havoc in England, the fangs of the cold still bite unlucky clans of Norway. Vargfell is one of them. Within two winters, supplies

Figure 24: Screenshot from the website

A little feedback:

What took me the most time is the setup of my work environment. I had to install and configure WSL Ubuntu, then install and configure Jekyll and finally install VS code and his extension for WSL. Then i started working on the website. The text/images/links integration part was quite easy as I'm using the Markdown language to write text on the pages. However, it took me time to understand fully how the software works in his page management. Also as I don't have a previous experience in web development, i chose to use an asset. The link of the GitHub page of this asset is on the 3rd page of the website. I made a selection of good looking assets and then showed my selection to my team. We agreed on one of them, however i quickly understood that this one was really dense and didn't have documentation, and it was the same for most of the assets i chose in first place until i find the asset "Alembic". It is quite simple but aesthetic and it has a GitHub page with documentation (short but it was enough for me). Then i had to look for the customization of the website (which is not completely done for now). I'm not feeling completely comfortable with CSS right now, but I'm working on it. Also the fact that the assets also include Flex Box CSS doesn't help me to understand the asset's CSS code.

3 Conclusion

Globally we are quite satisfied with our achievements. We have a good group dynamic and we communicate well. Overall, we are in agreement with what we planned in the book of specifications. For the moment, the game works fine, there is still a lot to do but we are confident and motivated for the next tasks.

List of Figures