# Rapport de soutenance #1
# The forust
## Alpha-c


Jacobé "Way hd" Pierre

Boisson "zraulix" Brice

Guérin "Shmiti" Jean

## Table des matières

# 1 Introduction

Nous avons démarré avec ce jeu assez tôt, en implémentant tout d'abord les mécanismes principale du jeu. On s'est vite rendu compte que travailler en groupe s'avérait assez difficile, mais finalement plutôt formateur. En effet chacun a des habitudes de travail différentes, tout combiner est juste une question de temps, nous avons donc appris comment collaborer même si cela est encore assez difficile.

Nous pensons que nous sommes dans les temps vis à vis de notre cahier des charges dans les fonctionnalités les plus techniques. En effet nous avons pensez que s'attaquer aux taches longues dès le début allait nous laissez plus de temps pour améliorer tous les petits détails du jeu ainsi que les tâches plus rébarbatives. C'est pour cela que nous avons essayer de travailler tous sur des tâches différentes dans le but de tout mettre en commun.

Nous avons du apprendre Unity de manière autodidacte grace à des tutos disponibles sur le site Udemy. Notre formation s'est d'abord faite en refaisant des petits jeux, comme par exemple un petit labyrinthe dans l'objectif d'avoir les bases sur Unity. Notre apprentissage du C# s'est fait avec les différents TP, même si toutefois il a fallu revoir certaines notions de C# propre à Unity, comme vous pourrez le voir dans les fonctionnalités implantées.

Nous avons commencé la mise en place de notre site web hébergé sur un VPS tournant sur ArchLinux de chez OVH, que nous allons actualiser avec la mise à disposition du cahier des charges ainsi que des rapports de soutenance.

Notre jeu possède dès à présent des fonctionnalités jouables, mais cela reste une première ébauche qui sera bien entendu améliorée par la suite pour proposer le jeu le plus complet possible et respectant au mieux notre cahier des charges. Nous pensons être prêt et avoir quelque chose de présentable pour cette soutenance.

# 2 Etat actuel

## 2.1 AI

### 2.1.1 Pierre

Je pensais que l'IA était plus difficile à implémenter dans Unity, mais finalement j'ai pris de l'avance sur ce que j'avais prévu de faire, en effet notre IA est d'ores et déjà capable d'attaquer toute seule notre joueur.

Notre IA a été implémentée à l'aide du **nav mesh agent** sur Unity, en effet à partir de là j'ai défini une cible, notre joueur, sur laquelle notre ennemi se fixe une fois que notre joueur rentre dans la zone dans laquelle il peut être détecter.

```
1         void Update()
2     {
3
4         agent.SetDestination(target.position);
5
6         if (agent.remainingDistance < 2)
7         {
8
9             ennemi.SetBool("Attack", true);
10            Debug.Log("fsdf");
11
12        }
13        else
14        {
15
16            ennemi.SetBool("Attack", false);
17        }
18    }
19
20    public void takedamageplayer()
21    {
22        GameObject.Find("FPSController").GetComponent<Player>().player1.TakeDamageHP(20);
23    }
```

*Image1*

Petite explication, à moins de 2 mètres du joueur, (cette distance est récupérée grace à "remainingDistance"), "l'agent" donc le monstre qui possède ce script va se mettre automatiquement à attaquer le joueur, et ici dans le cas de ce monstre , il infligera 20 de dégats sur les 100 points de vie du joueur. Le monstre est capable de bouger tout seul d'un point A à un point B en ligne droite, et selon les caractéristiques définies dans le cahier des charges, il se déplace vers le joueur qui rentre dans sa zone et l'attaque.

## 2.2 Combat

### 2.2.1 Pierre

Le combat contre les ennemis est implémenté, en effet quand on appuie sur le clic gauche de la souris cela déclenche l'animation de l'arme qui frappe et l'ennemi perd des points de vie. En revanche, lui aussi peut faire diminuer nos points de vie. Le combat est uniquement implémenté pour les armes de melée comme le marteau et le gourdin. Je suis plutôt en avance sur cette partie là et finalement le combat repose en grande partie sur les animations.

## 2.3 Les animations

### 2.3.1 Pierre

J'ai implémenté tout d'abord l'animation du bras qui appartien au joueur quand il marche puis l'animation du bras lorsqu'il porte un coup au monstre, tout cela à l'aide de l'animator et bien sûr à l'aide de condition en C#.

Laissez nous vous montrer en image :

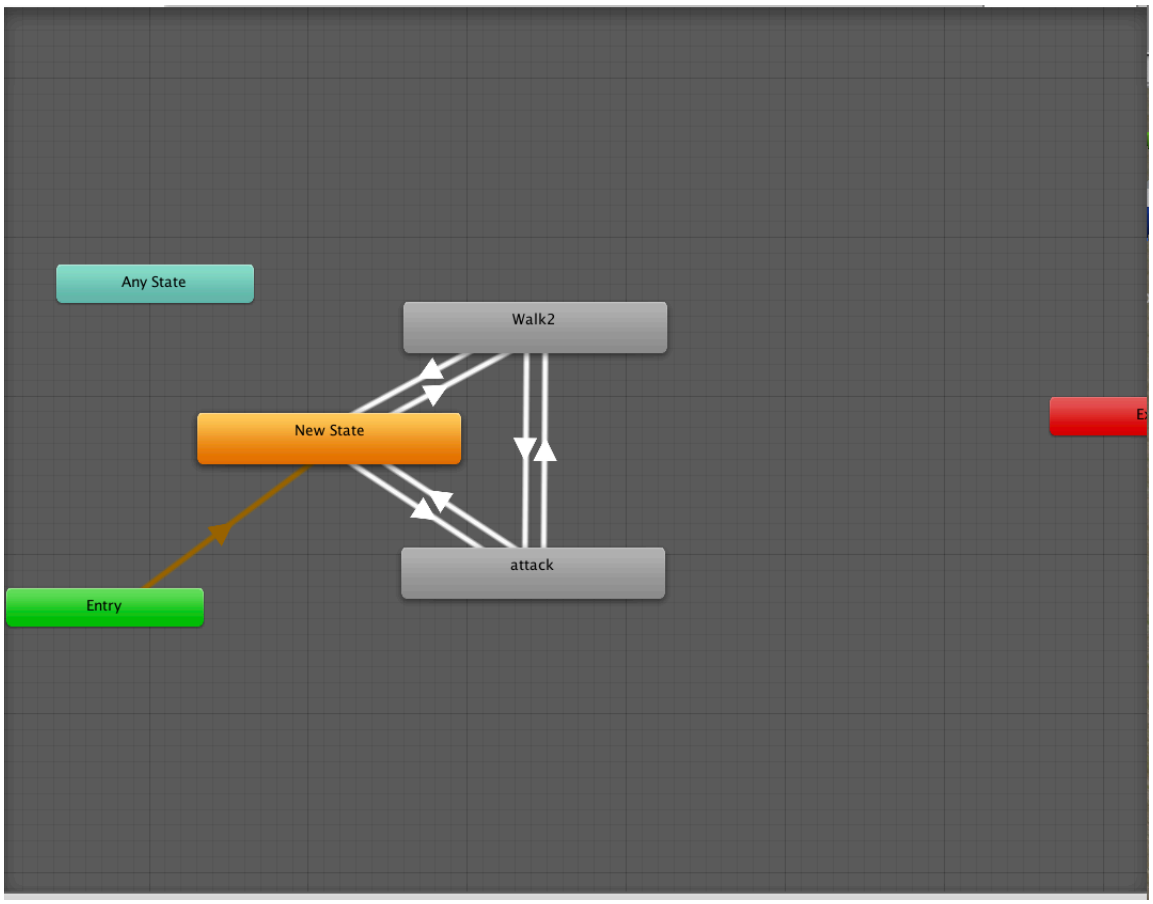



*Image 2*

Quelques explications, New State est l'état par défaut lorsqu'on lance le jeu et donc aucune animation n'est activée, les flèches blanches représentent des transitions. Sur ces flèches on peut déclencher des conditions en C# pour passer d'un état à un autre, "Walk2" est l'état sur lequel l'animation marche est déclenchée et "attack" l'animation d'attaque.

Voici ci dessous un exemple du script :

```
1     void Update()
2     {
3         if (Input.GetKeyDown(KeyCode.UpArrow))
4         {
5             pierre.SetBool("Walk", true);
6         }
7         if (Input.GetKeyUp(KeyCode.UpArrow))
8         {
9             pierre.SetBool("Walk", false);
10        }
11
12        if (Input.GetMouseButtonDown(0))
13        {
14            pierre.SetBool("Attack", true);
15
16        }
17
18        if (Input.GetMouseButtonUp(0))
19        {
20            pierre.SetBool("Attack", false);
21        }
22
23    }
```

*Image 3*

Exemple, lorsque que l'on enfonce la touche flèche du haut, la transition peut se faire et alors l'animation marche est activée. à l'inverse si on relâche on repasse à l'état initial, on peut aussi passer de l'état marche à attaque et inversement comme le montre les flèches entre les deux états. Comme le montre l'image on peut gérer ces conditions en C# l'aide de .SetBool(), avec en paramètre le nom de la condition entre deux états et si elle doit être sur vrai ou faux. Par exemple si j'appuie sur la touche tirer, ici le clic gauche, "Attack" est mis à true, tandis que si on relache "Attack" est mis à false. Car oui la touche Input.GetKey ou Input.GetMouse permettent d'obtenir les informations lorsque l'on tape au clavier ou à la souris.

Implémentation de l'arrivée sur L'ile avec une animation du joueur qui arrive en bateau. A la fin de cette animation la scène du jeu est chargée.

## 2.4 Système de viseur

Malheureusement nous n'avions pas encore pleinement connaissance des rayons dans unity, en effet ce n'est donc pas marqué dans le cahier des charges mais on a implémenté un système de viseur grace au Ray dans unity. Il sera représenté pour le joueur par une "croix verte" comme ci-dessous :

Les Ray sont en effet très pratique car ils permettent d'être plus précis pour le joueur, plus réaliste et pour nous de récolter beaucoup d'information qui nous simplifie drastiquement le code. En effet nous trouvons que les méthodes OnTriggerEnter() de Unity n'offrent pas ce niveau de précision pour ce que l'on veut faire dans notre jeu et complexifieraient notre code, nous vous avons mis un exemple de notre code ci-dessous exploitant les Rays.

Tout d'abord on déclare un Rayon avec sa position et sa direction, ici ce sera la position du viseur. RaycastHit, ici il s'appelle Hit, permet de récupérer toutes informations sur l'objet touché par le rayon, par exemple le nom avec la méthode Hit.Transform.name et la plus importante Hit.distance permettant de récupérer la distance de l'objet touché par le rayon par rapport à son point de départ. Avec toutes ces informations, on peut créer différentes conditions pour les différents mécanismes à implémenter pour notre jeu et qui seront détaillés dans les sections suivantes.

```
1
2         Ray ray = new Ray(Arme.transform.position, Arme.transform.TransformDirection(Vector3.forward));
3         RaycastHit hit;
4
5         if (Physics.Raycast(ray, out hit, 50))
6         {
7
8             if (hit.transform.name == "zombie@Walking" && c && hit.distance < 0.3)
9             {
10                Debug.Log("hit");
11                GameObject.Find("zombie@Walking").GetComponent<Monstre>().monstre1.TakeDamageHP(10);
12               GameObject.Find("zombie@Walking").GetComponent<Monstre>().transform.Translate(0,0,-1);
13
14
15            }
16
17
18            if (hit.transform.name == "WaterProDaytime" && f && p)
19            {
20                StartCoroutine(Trigger());
21
22
23            }
24
25            if (hit.transform.tag == "liane1" && hit.distance < 1 && p)
26            {
27                liane1 = GameObject.Find(hit.transform.name);
28                liane1.SetActive(false);
29                player1.AddInInventory(new inventory(10, inventory.item.Liane));
30            }
```

*Image 4*

## 2.5 L'orienté objet

L'orienté objet sera un des piliers principal de notre projet, c'est pourquoi nous avons décidé d'en faire une section pour vous expliciter les mécanismes de cette partie dans notre jeu. Nous avons avant tout divisé cela en deux grandes catégories, la super class Entity et la super class Item

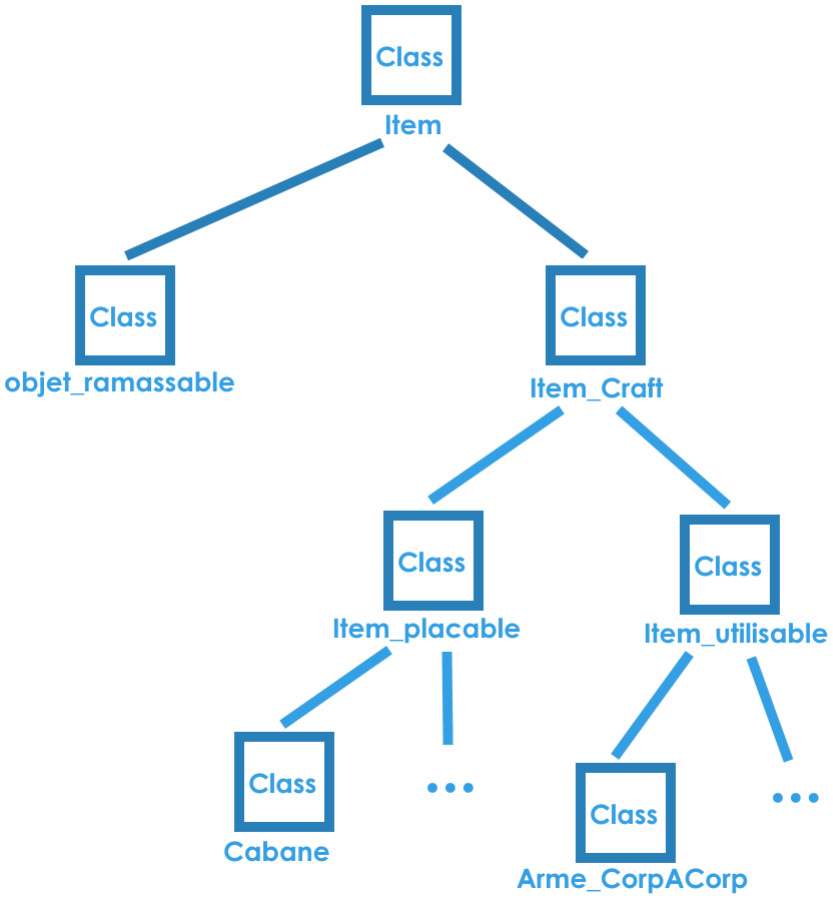
### 2.5.1 Pierre

De la class Entité hérite la class player et les classes propres au trois monstres, pour l'instant une seule class de monstres est implémentée, la class player est présente. En ce qui concerne la class Item, chaque groupe d'objet, comme défini dans le cahier des charges, à une class spécifique. Pour cette première soutenance seules les classes propres aux armes de corps à corps ainsi qu'aux objets ramassables de bases seront implémentées.

Nous allons mettre les super class en "Abstract" car nous n'aurons pas besoin de déclarer de nouveaux objets à partir de ces class. Le constructeur de la class Player prendra en paramètre les points de vie, eau, nourriture et sommeil. Ils sont tous privés avec un getter.

### 2.5.2 Brice

De la class Item héritent tous les items du jeux, que ce soit les objets ramassables sur la map ou les objets constructibles. Chaque type aura au final une class qui lui sera attribuée que ce soit les cabanes, ou encore les armes au corps à corps. Enfin ces différentes class auront des méthodes spécifiques leur permettant d'interagir avec le reste du jeu. Ces différentes méthodes sont décrites plus tard dans les sections qui les concernent (craft, ramassage des items, ...)




class.png

Les différent objets issus de ces class sont quant à eux déclarés de manière static dans un script. Cela permet d'accéder à ces objets, quelque soit l'endroit où l'on se trouve dans le jeux.

## 2.6 Gestion des différentes barres

### 2.6.1 Pierre

Je pense que je suis plutôt en avance sur ce point. En effet avec la classe player dont nous avons parlé juste au dessus et toutes les méthodes qui lui ont été implémentées comme par exemple : UpWater(), qui permet au joueur de régénérer sa barre d'eau et par la même occasion ses points d'eau. Bien sûr ceci est dupliqué sur les autres barres comme la faim, le sommeil et la soif. Pour cette première soutenance les barres peuvent s'actualiser en temps réel en fonctions des différents point de vie, eau, santé et sommeil.

Le visuel des barres va changer dans le but de les rendre plus esthétiques, mais en l'état elles sont fonctionnelles, implémentées sous forme de 1/4 de cercle pour chaque barre. Pour cela je me suis servi du composant Canvas sur Unity. Ensuite j'ai ajouté des images pour chaque barres, ici les images sont des sprites, puis passer en type "filled" et en "Fill method Horizontal" , c'est à dire que si nous jouons sur ce paramètre, l'image "disparait" de manière horizontal, petit exemple ci-desous :

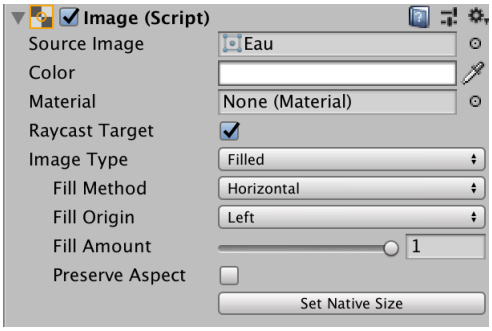



*Image 5*

Ici c'est la barre de l'eau, au départ elle est totalement remplie elle est donc à 100% et ainsi Fill Amount est à 1, grace au script C# nous allons jouer sur ce paramètre lorsque le joueur perd des points d'eau. Finalement elle est reliée à l'objet déclaré appelé "Player1" qui sera notre joueur. Maintenant laissez moi vous détaillez comment j'ai mis cela en place en C#, tout ceci marche pour toutes les barres à l'heure actuelle.

```
1           private void Update()//here uptade one per frame
2    {
3
4
5        pierre.SetActive(k);
6
7        #region Bar de vie
```

```
8
9           eau.fillAmount = player1.Water / player1.Max;
10          food.fillAmount = player1.Food / player1.Max;
11          dodo.fillAmount = player1.Sleep / player1.Max;
12          life.fillAmount = player1.Hp / player1.Max;
13
14          #endregion
15      }
```

Tout d'abord, les variables eau, food, dodo et life sont de type image et donc stock les informations de leurs images respectives et en faisant une division de float, on actualise la barre Fill Amount présentée dans l'image 5, la méthode update permet de faire cela à chaque Frames.

De plus, pour cette soutenance, nous pourrons dès à présent boire de l'eau dans certain plan d'eau implémenté sur la map et gerer son eau avec la gourde qui est implémenté, nous reparlons de cela après.

## 2.7 Système de ramassage

### 2.7.1 Brice

Le système de ramassage fonctionne en collaboration avec le viseur cité plus tôt, grâce au Raycast on peux savoir si l'on vise un objet et quel objet est visé il suffit alors d'appeler une méthode de la Class objet_ramassable, get() qui ajoute 1 à la variable comptant le nombre d'item de cette catégorie.

```
1  public void get()
2      {
3          if (own <= max_own)
4          {
5              own += 1;
6          }
7      }
```

*methode issue de la class Item_ramassable*

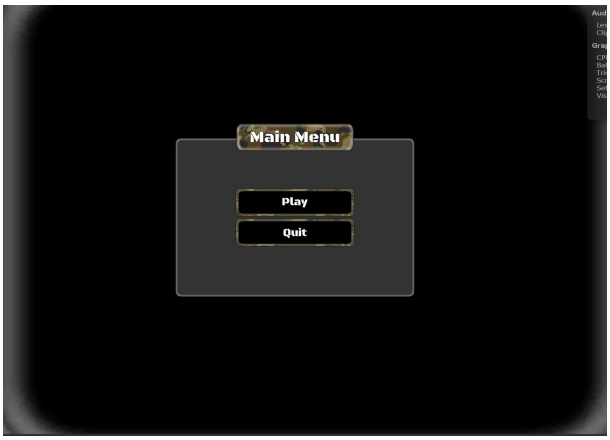
### 2.7.2 Pierre

Je me suis occupé de la mise en place du viseur comme expliqué dans la section 2.3, pour faire simple les objets que l'on peut ramasser possèdent des tags, quand l'objet que frappe le rayon avec ce tag match à une certaine distance alors on ramasse l'objet.

## 2.8 L'interface

### 2.8.1 Pierre

Pour cette soutenance, l'écran d'accueil est implémenté mais pas le système de sauvegarde, en tous cas il ne sera pas fonctionnel ni présentable, toutefois à partir de cet écran de chargement nous pouvons charger le jeu et le quitter, de plus un petit écran de chargement le temps que la carte se charge avec une barre de défilement est implanté. Tout ceci grâce à l'importation du Module UI de Unity en C# voici ci-dessous un exemple de l'écran d'accueil :




Je ne reparlerai pas dans cette section des barres de vie car une section leur est dédiées, en revanche je vais parler de la barre de vie du monstre, en effet j'ai décidé de mettre la barre directement au dessus de lui dynamiquement comme le montre l'image ci-dessous :

Donc l'idée c'est de réduire la largeur de la barre de vie à chaque coup donné par le joueur, cette taille est représentée par un vecteur ayant des coordonnées X & Y, bien sûr ce monstre est représenté par un objet monstre dans le code.

```
        void Update()
    {
        vie.size = new Vector2(((monstre1.Hp)*10)/75, 10f);

        if (monstre1.Hp <= 0)
        {
            GameObject.Find("zombie@Walking").SetActive(false);
        }

    }
```

Donc j'actualise la coordonnée en X en fonction de la vie du monstre pour réduire la largeur de la barre.

Un menu pause à également été implémenté, en effet l'espace UI d'unity propose des boutons qui peuvent exécuter une méthode C# lorsque l'on clique dessus, c'est ce qui a été exploité ici, regardez ce script :

```
        void Update()
    {

        if (Input.GetKeyDown(KeyCode.Escape))
        {
            p = !p;
            if (p)
            {
                GameObject.Find("FPSController").GetComponent<FirstPersonController>().enabled = false;
                Time.timeScale = 0;
                image1.SetActive(true);
                Cursor.visible = true;
                Cursor.lockState = CursorLockMode.None;


            }
            else
            {
                GameObject.Find("FPSController").GetComponent<FirstPersonController>().enabled = true;
                Time.timeScale = 1;
                image1.SetActive(false);
                Cursor.visible = false;
                Cursor.lockState = CursorLockMode.Locked;

            }
        }

    }

    public void resume()
    {
```

```
32          Time.timeScale = 1;
33          GameObject.Find("FPSController").GetComponent<FirstPersonController>().enabled = true;
34          image1.SetActive(false);
35          Cursor.visible = false;
36          Cursor.lockState = CursorLockMode.Locked;
37
38      }
39
40      public void exit()
41      {
42          SceneManager.LoadScene(0);
43      }
44  }
```

Donc ici on peut faire pause en appuyant sur escape et revenir en jeu en appuyant sur cette même touche, c'est le booléen P qui s'en charge pour différencier les deux cas. Lorsque l'on fait pause, on fige le temps d'où "Time.timeScale=0", on rend le curseur visible de la souris en mode libre de façon à pouvoir cliquer sur les différents boutons. La fonction resume est en quelque sorte attribuer au bouton Resume du menu Pause et exit au bouton exit qui déclenche leur fonctions attribué lorsque l'on clic dessus.

## 2.9 Site web

### 2.9.1 Pierre

Configuration de l'administration et de la sécurité du vps.

### 2.9.2 Brice

Pour ma part, je me suis dans un premier temps occupé de la configuration du serveur pour accueillir le site web, c'est à dire : installation de apache pour faire fonctionner le site, redirection du nom de domaine vers notre serveur et notre site, création et configuration de la base de donnée mySQL et enfin mise en place du ftp.

Dans un second temps j'ai créé un code HTML et css de base pour le site.

## 2.10 Commandes

### 2.10.1 Pierre

Pour cette soutenance, la plupart des commandes sont d'ores et déjà implémentées, en effet on peut interagir avec les objets avec la touche E, frapper les ennemis avec le clique gauche de la souris, se déplacer à l'aide des flèches directionnelles ou des touches ZQSD.

Les méthodes :

```
1 Input.GetKeyDown
2 Input.GetKeyUp
3 Input.GetMouse
```

Récupérant respectivement : lorsque l'on frappe la touche, la remontée ainsi que les clics sur la souris. Ces méthodes m'ont ainsi bien aidé pour faire les différentes taches expliquées plus haut.

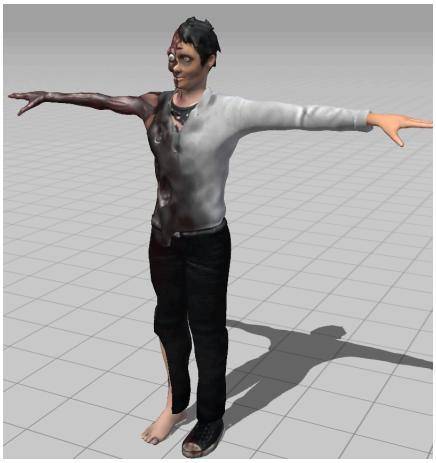
### 2.10.2 Brice

L'inventaire implémenté est quant à lui contrôlable à la souris pour effectuer les différentes actions tel que craft(construire) un nouvelle item (objet) ou déplacer les différents éléments. L'inventaire quant à lui est ouvrable à l'aide de la touche A.

## 2.11 Les graphismes

### 2.11.1 Pierre

Implémentation du modèle 3D trouvé du monstre trouvé sur le site Mixamo sur lequel sont disponible de nombreux modèle 3D et animations. Mise en place de modèles 3D de végétaux pour décorer la map, aide à Jean pour le modelage du terrain et la mise en place de texture.



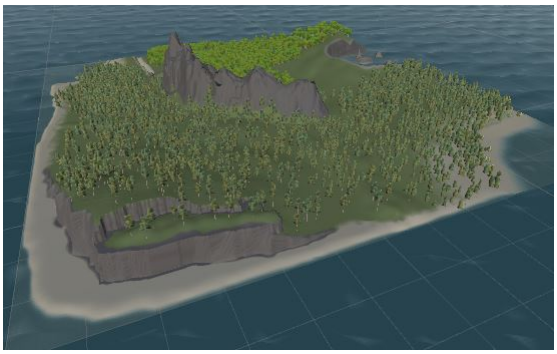
Voici le modèle 3D du monstre trouvé sur Mixamo. De plus j'ai implémenté les modèles 3D des armes de corp à corp et de la cabane tous trouvé sur l'asset store gratuit d'Unity.

### 2.11.2 Jean

Je me suis occupé de la conception du terrain, je pense avoir pris une bonne avance sur ce qui était prévu dans le cahier des charges. La carte est dès à présent jouable à quelques détails près(ex : La map n'a pas encore de limite donc le joueur peut tomber dans le vide).

Description rapide de la carte :

La carte est une île composée de plusieurs lieux notables, une chaine de montagne au centre (elle permet de limiter la mobilité du joueur sur la carte et le pousser à explorer en lui forçant à la contourner), une crique, des falaises (pour rendre l'environnement moins linéaires et plus sympathique à découvrir), des plages (essentiel pour une île), des forêts, Une cascade et une rivière (constitue un point d'eau en plus d'enrichir l'environnement).

rendu 1.JPG 


rendu 3.JPG

## 2.12 Gestion inventaire

### 2.12.1 Pierre

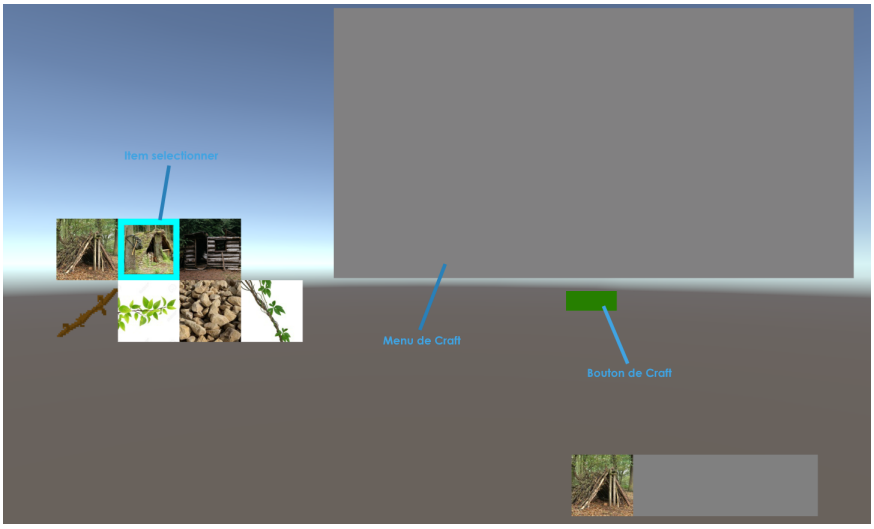
Pour ma part je me suis occupé de son intégration dans le jeu et le gameplay, par exemple le fait de pouvoir l'ouvrir ainsi que que le fermé. De plus, je me suis occupé de l'ajout des items dans l'inventaire au moment moment, en bonne quantité et au bon endroit.

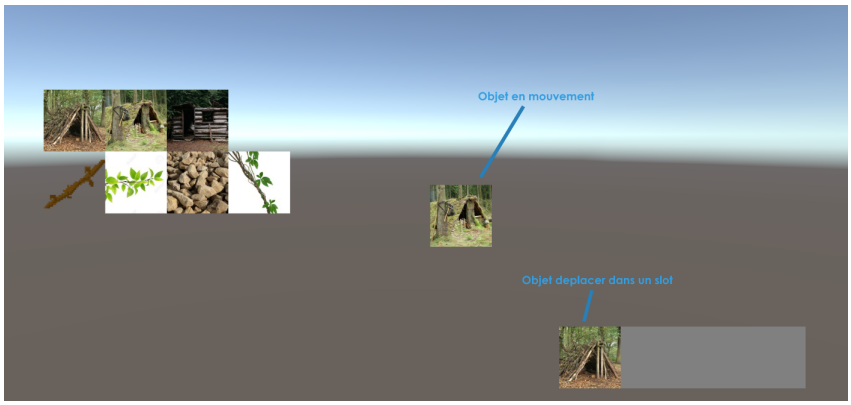
### 2.12.2 Brice

Je me suis occupé de l'implémentation de l'inventaire en lui même c'est à dire aussi bien au niveau de l'interface qu'au niveau du code pour la gestion des items/craft.

J'ai commencé dans un premier temps par créer toute la partie interface de l'inventaire :

Le système qui permet de bouger un item, le système qui permet de sélectionner un item, le menu de craft ... Cependant c'est une partie qui va devoir être refaite car ayant utilisé des sprites à la place de canvas l'inventaire a du mal a s'intégrer dans un environnement 3D.




2.png

Ensuite j'ai réalisé toute la partie gestion des Items en script, cette partie est déjà expliquée dans la partie gestion des items, enfin j'ai réalisé les scripts pour

gérer le craft d'item, je détaille cela dans la prochaine section qui est dédiée au craft. Il me reste cependant à gérer la partie slot du joueur qui permet au joueur d'utiliser un item hors de l'inventaire.

## 2.13 Craft

### 2.13.1 Brice

Pour implémenter le craft j'ai dans un premier temps cherché à récupérer deux information :

La quantité de ressource disponible pour le craft :

Pour cela j'ai implémenté une méthode dans ma Class InventoryObject (Class dans laquelle les items sont déclaré statiquement), cette méthode me retourne un tableau contenant le nombre de chaque item nécessaire au craft.

```
1     public static int[] ressource()
2     {
3         int[] ressource = new int[]{baton.own,feuille.own,pierre.own,liane.own};
4         return ressource;
5     }
```

Les information sur l'item a craft :

```
1         item = GameObject.Find("Inventory").GetComponent<Inventory>().craftitem;
```

Cette fonction permet de récupérer les informations de l'objet sélectionné depuis la Class Inventory.

Une fois ces informations disponibles j'ai implémenté le craft. Le caft se déroule en 3 étapes :

Premièrement l'appel de la fonction craft quand le bouton est pressé :

```
1     private void OnMouseDown()
2     {
3         GameObject.Find("Inventory").GetComponent<Inventory>().craft_enable = true;
4         item = GameObject.Find("Craft").GetComponent<DisplayCraft>().item;
5         craft(InventoryObject.ressource(),item);
6
7     }
8
9     public void craft(int[] ressource, Item_Craft item)
10    {
11        if (Craftable(ressource, item.Craft_Cost))
12        {
13            item.own += 1;
14            RemoveComponant(ressource, item.Craft_Cost);
15        }
16    }
```

Quand le bouton est pressé la fonction craft est lancée, cette fonction appelle la deuxième étape.

Vérification que l'on a tous les items nécessaires au craft :

```
1   private bool Craftable(int[] ressource, int[] needed_resource)
2   {
3       int len = ressource.Length;
4       bool valide = true;
5       for (int i = 0; i < len && valide; i++)
6       {
7           valide = ressource[i] >= needed_resource[i];
8       }
9       return valide;
10  }
```

Cette fonction retourne true si la quantité de chaque ressource nécessaire est inférieure ou égale à la quantité de ressources disponibles.

La troisième étape consiste à retrancher au ressource la quantité de ressource utilisé pour le craft.

```
1   private void RemoveComponant(int[] ressource, int[] needed_resource)
2   {
3       InventoryObject.baton.own -= needed_resource[0];
4       InventoryObject.feuille.own -= needed_resource[1];
5       InventoryObject.pierre.own -= needed_resource[2];
6       InventoryObject.liane.own -= needed_resource[3];
7   }
```

## 2.14 Items

Ici nous parlerons de l'implémentation des mécaniques des différents items dans le gameplay.

### 2.14.1 Pierre

Personnellement je me suis occupé de l'implémentation du gameplay de la gourde et donc de la gestion de l'eau. je rappelle que la gourde est donnée par défaut au joueur, elle permet de stocker de l'eau. Comme elle est donnée par défaut, elle fait partie intégrante de la classe Player. Je tiens à préciser avant tout qu'elle est représentée et gérée sur l'écran du joueur de la même manière que les barres de vie.

Tout d'abord je défini une nouvelle méthode dans la classe Player, qui est un IEenumerator, un IEenumerator sert à faire "une pause" dans le code lorsque le programme l'éxécute

```
1   public IEnumerator addgourd()
2       {
3           if (gourd <= CapGourd)
```

```
4           {
5               gourd += 10;
6
7               GameObject.Find("FPSController").GetComponent<Player>().f = false;
8               yield return new WaitForSeconds(120f);
9               GameObject.Find("FPSController").GetComponent<Player>().f = true;
10          }
11          else
12          {
13              yield return new WaitForSeconds(120f);
14
15          }
16      }
```

Donc si c'est inférieur à la capacité de la gourde on ajoute, on ajoute la quantité de 10 dans la gourde. Le booléen f me sert à vérifier si oui ou non je peux ajouter, il faudra attendre 120 secondes pour que le booléen f repasse à true et donc que je puisse reboire de l'eau. C'est ce que l'on appelle une "Corroutine".

Comme d'habitude je me sers de mon viseur pour savoir si je suis au bord d'un cour d'eau.

```
1       if (hit.transform.tag =="Lake" && p && f && hit.distance < 2)
2           {
3               StartCoroutine(player1.addgourd());
4           }
```

Le booléen p me sert à faire la différence lorsque j'appuie sur la touche E ou lorsque je la relache, ceci se trouve dans Update() et dans Physics.Raycast. Si toutes ces conditions sont vérifiées alors je déclenche ma Corroutine et j'ajoute de l'eau dans ma gourde. Je peux alors boire de l'eau en appuyant sur F ce qui régénère cette barre, mais réduit logiquement la quantité dans ma gourde.

### 2.14.2 Brice

Quant à moi je m'occupé de tous les items disponible dans l'inventaire dont les méthodes seront implémentées au fur et a mesure que l'on en aura besoin.

# 3 Et après ?

Nous pensons que le projet est sur une bonne voie, en revanche certaine personne du groupe vont devoir rattraper se qu'ils n'ont pas fait pour cette première soutenance dans le but que tout le monde code et participe au projet.

## 3.1 AI

### 3.1.1 Pierre

En se qui concerne l'AI pour ma part, il s'agit de l'implémenter sur les deux monstres restant mais comme c'est déjà fait pour un ça ira vite sur les deux autres. Je pense que le gros de mon travaillant restant est de corriger tous les bugs que je pourrai constater du à l'IA, qui je pense seront nombreux une fois la carte complètement fini, faire en sorte qu'elle ne rentre pas en collision avec des objets par exemple. Au vue de l'avancement sur ce point pourquoi pas ne pas implémenter un quatrième monstre qui pourrait attaquer à distance.

### 3.1.2 Brice

Pour ma part il s'agira d'en apprendre plus sur le fonctionnement des IA dans unity dans le but d'aider Pierre dans la gestion des IA d'ici à la deuxième soutenance.

## 3.2 Animation

### 3.2.1 Pierre

Premièrement apporter une amélioration aux animations déjà implémentés dans le but des les rendent certainement plus réaliste, "moins robotique". Ensuite mettre en place les animations concernant les armes à distance comme l'arc ou l'harpon. Faire les différentes animations du personnage qui réussi le jeu, qui trouve un moyen de sortir de l'Ile ou qui gagne contre le temps, en effet il sera évacué donc faire l'animation de l'évacuation. Faire les animations du personnage qui pêche.

## 3.3 Interface

### 3.3.1 Pierre

Même si nous ne sommes pas graphiste on peut toujours améliorer l'aspect visuel. Le menu pause sera implémenté et fonctionnel, de plus comme déjà dit, le visuel des barres de vie sera amélioré.

### 3.3.2 Brice

Une fois les mécanismes de l'inventaire termine, il sera nécessaire de lui donner une interface claire, pour cela il faudra une harmonisation des différents visuel.

## 3.4 Gestion des différentes barres

### 3.4.1 Pierre

Implémentation de la pêche pour la deuxième soutenance et de la chasse ainsi que du feu pour faire cuire le tout, donc ceci concerne la barre de l'eau et de faim. Ainsi que l'implémentation du sommeil. Correction éventuel de bugs.

## 3.5 Les graphismes

### 3.5.1 Pierre

Pour la deuxième soutenance, implémentation des deux autres modèles de monstres, rajout de décors dans la map comme plus de végétations, d'arbre mort. Implémentation des modèles 3D des armes à distances et des améliorations de la cabane. Ajout du grand feu.

### 3.5.2 Jean

Le gros du travail restant sera plus concentré sur des détails dans le but d'améliorer le réalisme de la carte ou dans le choix de meileurs textures ou assets ; Pour l'instant nous n'avons utiliser en majorité que les assets de bases d'Unity ce qui nous limite rapidement, par exemple une forêt est constituée d'arbes qui ont tous 3 branches sciées ce qui est questionnable sur une île déserte.

## 3.6 Combat

### 3.6.1 Pierre

Qui dit armes à distances dit nouveau système de combat implémenté, pour l'arc gérer la trajectoire des flèches avec le viseur et la physique de unity. Il en va de même pour le harpon. Implémenter la visée avec le clic droit de la souris.

### 3.6.2 Brice

L'implémentation des différents types d'armes ainsi que leurs mécanismes.

## 3.7 Item

### 3.7.1 Brice

L'implémentation de tous les items restant devra être fait le plus rapidement possible.

## 3.8 Inventaire

### 3.8.1 Brice

La principal tâche qu'il reste à faire sur l'inventaire est la réimplémentation sous forme de canvas. Dans le but que l'inventaire soit mieux intégré au jeu. Il faudra aussi ajouté quelque indication visuel tel que le nombre d'item disponible.

### 3.8.2 Pierre

Pour ma part je vais m'occuper de relier l'inventaire au player pour qu'il puisse être accessible en jeu et aider Brice à gérer les différents items dans celui-ci.

## 3.9 Craft

### 3.9.1 Brice

Le système de craft étant plutôt bien avance il ne reste qu'a clarifier les differents éléments en ajoutant des indications visuel tel que le nombre d'item disponible, le nombre d'item nécessaire, ...
Ainsi que le coût en temps du craft.

### 3.9.2 Pierre

Je vais aider Brice en rendant le craft accessible in-game et en l'aidant à implémenter les différents Item dans le code et bien sûr avoir accès en jeu à se que l'on a craft.

# 4 Conclusion

Malgré un début difficile, nous avons cependant atteint nos premiers objectifs fixé pour cette soutenance, cependant s'ils l'ont veut continuer dans cette voie et atteindre nos objectifs pour la deuxième soutenance, il ne faut pas se relâcher et persévérer à améliorer notre travail d'équipe. Nous avons décidé d'apporter des modifications dans l'avancement de notre projet pour la deuxième soutenance.

| | Pierre | Jean | Brice |
|---|---|---|---|
| Commandes | 70% | 70% | |
| Gestion inventaire | 90% | - | 90% |
| **Implémentation des items :** | - | - | - |
| Cabane/feu/chasse | - | 65% | 65% |
| Armes corps à corps/distance/pêches | 65% | - | 65% |
| Outils/consommables | 65% | 65% | - |
| Système de ramassage | - | 95% | 95% |
| Système de craft | 95% | | 95% |
| **Gestion des différentes barres :** | - | - | - |
| Santé | 85% | - | 85% |
| Soif | - | 85% | 85% |
| Faim | 85% | - | 85% |
| Sommeil | 85% | 85% | - |
| **Combat :** | - | - | - |
| Gameplay du joueur | 60% | - | 60% |
| Gameplay des monstres | 60% | - | 60% |
| Physique | 80% | | 80% |
| IA | 50% | | 50% |
| **Graphismes :** | - | - | - |
| Animation | 70% | - | 70% |
| Modèle 3d | - | 70% | 70% |
| Aspect graphique général | 70% | 70% | - |
| Sounds design | - | 40% | 40% |
| Interface | 90% | - | 90% |
| Site Web | 70% | 70% | 70% |