Rapport de projet
The forust
Alpha-c

Jacobé "Way hd" Pierre

Boisson "zraulix" Brice

Guérin "Shmiti" Jean

Table des matières

1 Introduction

Voilà un beau projet d'environ six mois qui se termine maintenant. "Beau" car ce projet nous aura appris de nombreuses choses que cela soit sur notre côté humain ou bien technique. Nous nous sommes vite rendu compte ce que cela impliquait en terme de travail si nous voulions avoir un projet qualitatif et respectant au maximum ce qui avait été annoncé dans le cahier des charges.

Aucun d'entre nous ne se connaissait avant notre arrivée à EPITA, notre projet aura donc contribué à tisser des liens entre nous en apprenant à travailler ensemble. Toutefois il est nécéssaire de nuancer ces propos, au début du projet du moins avant la première soutenance même si nous travaillons tous de notre côtés, nous avions du mal à communiquer entre nous et la mise en commun de nos idées s'est faite durant la semaine de la première soutenance, ce qui ne nous a quand même pas empêcher de rendre quelque chose de convenable pour cette soutenance.

Pour gagner en efficacité, nous nous sommes efforcés de travailler mieux en groupe la deuxième soutenance et bien sûr pour cette fin de ce projet. Nous avions comme mot d'ordre la cohésion et finalement cela a était plutôt facile à mettre en place et nous a fait atteindre un rythme de croisière qui nous convient à tous, c'est je pense ce qui a fait atteindre un bon point à notre projet.

Nous pensons avec sincérité avoir atteint tout nos objectifs personnel et impératifs que nous nous étions fixés. Nous n'avons pas du faire de compromis sur les fonctionnalités à implémenter dans le jeu pour finalement livrer quelque chose de similaire à ce que nous avions envisagé dans le cahier des charges.

Sur beaucoup de points nous étions à jour ou même en avance par rapport à ce qui était prévu. Pour les autres points, comme par exemple le site web, nous avons redoublés d'effort pour compenser ce retard.

Durant ce rapport de projet nous rappellerons en quoi consiste notre projet, vous rappellerons ce qui était prévu dans le cahier des charges et les différents outils utilisés. Puis nous décrirons pour chaque aspect du jeu ce qui a été fait aux différentes soutenances dans le but d'avoir un suivi chronologique. Enfin nous aborderons le bilan personnel de chacun pour ce projet, viendra la conclusion et nous vous laisserons avec les différentes annexes dans lesquelles vous retrouverez des images détaillez de notre projet.

2 Notre projet & cahier des charges

Notre projet est un jeu de survie à la première personne réalisé à l'aide d'Unity et du C#, avec ci-dessous quelques lignes pour rappeler l'objectif :

2.1 Le principe

Comme dit plus haut notre jeu est de type survie qui se compose : d'une carte, un joueur, des ennemis et des outils pour aider le joueur. Dans le jeu, le joueur est entièrement libre, c'est donc un open world. Le joueur est donc entièrement livré à lui-même que cela soit pour ses mouvements et ses choix, rien ne lui est imposé si ce n'est les contraintes proposées par notre monde, que cela soit par les ennemis présents sur la carte ou encore les différents éléments nécéssaire pour qu'un humain puisse vivre.

2.2 Le but

Pour finir le jeu le joueur a deux possibilités : Soit survivre sur l'ile pendant 30 jours et attendre que la production de l'émission vienne le chercher en hélicoptère. Soit trouver un moyen (secret) de partir. Pour survivre le joueur doit gérer son hydratation en trouvant une source d'eau potable, son alimentation en pêchant, chassant ou cueillant. Il peut crafter des outils dans le but de simplifier sa survie. Les outils craftables sont plutôt basiques, uniquement avec des ressources ramassables. A l'exemple d'un harpon, une cabane, feu de camp ou encore un arc.

De plus il doit gérer son sommeil en choisissant le moment adapté pour aller dormir. La gestion du sommeil, de l'hydratation et de la nourriture est primordiale sous peine de voir sa barre de vie baisser, pouvant entrainer le game over (fin du jeu) si cette barre arrive à zéro. Il a cependant d'autres manières de perdre toute sa vie, effectivement après quelques jours sur l'ile des monstres apparaissent dans le but de varier et compliquer le gameplay, ceci pouvant tuer le joueur.

2.3 Reprise du cahier des charges

Ici nous récapitulons qui à fait quoi et donc ce qui est possible de faire dans notre jeu.

2.3.1 Pierre

Durant ce projet j'ai eu la chance de pouvoir toucher à beaucoup de chose, comme mettre en place une partie du système de combat à distance et au corps à corps, ainsi qu'une partie de l'IA et donc des ennemis. J'ai aussi pu prendre part au façonnage de la carte et de l'implémentation des différents modèles 3D et pour certains de leur interactions .

L'interface utilisateur était aussi une de mes tâches tout en prenant compte la gestion des différentes barres. Je suis personnellement satisfait du résultat car on retrouve tout se qui était prévu, menu pause, menu principale, options ect... J'ai aussi pu créer le système de ramassage, participer à l'implémentation de l'inventaire dans le jeu ainsi que des différentes animations et enfin du système de ramassage.

2.3.2 Brice

Durant ce projet j'ai touché à toutes les parties de unity aussi les animations pour créer les IA, que le système de NavMesh toujours pour celui-ci. Concernant l'inventaire et le système de ramassage cela m'a permis de voire les différences entre les sprites dans les canvas et les sprites tous seul, ainsi que le sound design.

J'ai aussi pratiqué la programmation orientée objet lors de l'implémentation des items. Lors de l'implémentation du système pour changer d'armes et du système de ramassage, la maitrise de la hiérarchie de Unity à été un élément essentiel. J'ai enfin appris de nombreuses choses sur les sites web lors de la création du notre

2.3.3 jean

Durant ce projet, j'ai pu travailler sur les points comme l'aspect graphique de notre jeu et le façonnage de la carte, mais également le sound design et un peu sur le site web.

2.4 Logiciels et sites utilisés

Nous allons vous lister les logiciels qui ont été utiles à la production de notre projet et à quoi ils ont servit

Moteur de jeu :

A permis de développer notre jeu et de le faire tourner.

— Unity : Il a permis de développer notre jeu et de le faire tourner.

IDE (Environnement de développement) :

A permis d'écrire et d'organiser le code en C#

— JetBrain Rider version 2019 & 2020
— Microsoft Visual Studio version 2019

Traitement d'images :

A permis de modifier, éditer et mettre en forme des images.

— Photoshop CC 2019
— Paint.net

Modélisation & Animation :

Site qui nous ont permis de trouver des animations et modèles 3D.

— Mixamo (banque d'animation et de personnages 3D)
— Skechtfab (banque de modèle 3D)
— Free3Dmodel (banque de modèle 3D)
— AssetStore de Unity (Banque de modèle 3D mais pas que)

Montage vidéo :

Utile pour monter nos vidéos des soutenaces ainsi que notre trailer.

— Final Cut pro 2019 (MacOS)
— Adobe premiere pro CC 2019

Latex :

Logitiel utilisés pour rédiger nos rapport en LaTeX.

— Texpad (MacOS) & Overleaf

3 Développement du projet

Nous attaquons ici la partie "principale" de ce rapport présentant en détail ce qui a été fait à chaque soutenances.

3.1 Pierre

3.1.1 AI

1ère soutenance :

J'ai commencé à travailler l'AI le plus tôt possible car c'est un point essentiel s'ils ont veut avoir un jeu qualitatif. Donc pour cette première soutenance, j'ai implémenté un premier monstre qui pouvait attaquer le joueur à proximité et suivre le joueur s'il était trop proche (à moins de 10 mètres). Toutefois, quelques bug sont apparus comme par exemple le fait que le zombie ne se tourne pas s'ils l'ont se place derrière lui lors du combat, problème en partie corrigé par Brice à la deuxième soutenance.

Nos monstres possèdent différentes caractéristiques et celui-ci était un peu le monstre "équilibré", c'est à dire ni trop faible, ni trop fort, j'ai respecté ce qui était marqué dans le cahier des charges, il peut donc voir le joueur à 10 mètres, il possède 75 point de vie et inflige 10 de dégâts aux joueurs (ces caractéristiques vont être modifiés aux prochaines soutenance). dont voici un exemple du résultat ci-dessous. Je l'ai de plus équipé d'une animation lorsque l'attaque se déclenche qui fait perdre des points de vie au joueur. Brice vous reparleras beaucoup plus en détails de ce monstre finalisé.

Comme vous pouvez le constater j'ai aussi implémenté une barre de vie qui se déplace bien sûr avec ce dernier pour indiquer en temps réel combien il reste de vie à l'ennemie dont le joueur doit faire face.

Nous pouvons noter que j'ai aussi créé un script qui permet de faire déplacer tout seul ce monstre d'un point A à un point B en attendant qu'un joueur se rapproche de lui.

2ème soutenance :

Commençant de plus en plus maitriser les mécanismes de l'AI dans unity dont j'ai pu largement parler durant les soutenances précédentes, j'ai pu implémenter les deux derniers monstres annoncés dans le cahier des charges, Brice à pu appliquer les même correctifs pour que les monstres soit toujours en face du joueur lors du combat. Vous pourrez les retrouver dans les annexes.

Pour rappel, voici ce qui fait globalement fonctionner un des monstres à cette soutenance :

```
1           if (Monster.remainingDistance > walkdistance)
2
3       {
4           Monster.speed = 0f;
5           anim.SetBool("Run",false);
6           anim.SetBool("Attack",false);
7       }
8       else
9       {
10
11          Monster.speed = 2f;
12          anim.SetBool("Run", true);
13          anim.SetBool("Attack", false);
14
15          if (Monster.remainingDistance < walkdistance -8 && Monster.remainingDistance !=0
16          {
17              anim.SetBool("Attack", true);
18              Monster.speed = 0f;
19
20          }
21      }
```

Si le joueur est trop loin du monstre, le monstre ne fait rien. Quand le joueur commence à se rapprocher, le monstre vient vers lui et lorsque le joueur est vraiment près le monstre déclenche l'attaque. Bien sûr ce script s'adapte en fonction des spécificités des monstres.

Concernant les deux monstres que j'ai implémentés pour cette soutenance, ils respectent les caractéristiques dans le cahier des charges (les caractéristiques d'attaque vont changer à la prochaine soutenance) , l'un est le plus faible mais le plus vif des monstres, il détecte le joueur à plus de 15 mètres, possède une attaque de 5 et une vie de 50. Et enfin le deuxième est le plus fort des trois, il respecte les caractéristiques suivantes : détecte le joueur à seulement 5 mètres, 100 points de vie et 15 d'attaques.

cela oblige le joueur à varier le gameplay en fonction des monstres.

3ème soutenance :

Pour cette dernière soutenance je me suis attelé avec Brice, à améliorer notre IA ainsi que notre système qui permet de maintenir le monstre en face de notre joueur.

Je me suis notamment occupé a rajouter des attaques au troisième monstre, le plus fort, qui est un peu le "boss" de notre jeu. Ces attaques étant des animations, je les ralenti au début pour laissez au joueur le temps d'esquiver l'attaque. Puis, grâce au Collider je peux savoir si le joueur se trouve dans les zones de frappes du monstre et infliger les dégâts nécessaires au joueur. Ces attaques se déclenchent par rapport à où se trouve le joueur par rapport au monstre, s'il est à gauche cela déclenche un coup de point à gauche par exemple.

Concernant le deuxième monstre, il possède toujours les mêmes caractéristiques, sauf que maintenant j'ai fait en sorte qu'il infecte le joueur s'il frappe ce dernier. Si le joueur ne mange pas une baie disponible sur un buisson dans les 10 secondes alors le joueur décède. Il ne possède donc plus une attaque de 5.

	Attaque 1	Attaque 2	Attaque 3
Monstre2	Infect le joueur en retirant -10 pv / seconde, tant qu'une baie n'a pas été mangé	-	-
Monstre3	Attaque d'arrivée sur le joueur : un saut avec un coup à la fin retirant -30 au joueur s'il n'esquive pas	Attaque basique : un léger coup dans lequel le monstre fait un tour sur 360 degrés et retire -15 au joueur s'il n'esquive pas	Attaque ultime tous les 3 coup basique : Gros coup de point venant du haut retirant -50 de vie au joueur s'il n'esquive pas

Les différentes caractéristiques liées à la vie et à la distance à laquelle le zombie peut voire le monstre n'ont pas changé, voici un tableau pour plus de lisibilité :

	Point de vie	Distance à laquelle le monstre détecte le joueur
Monstre2	50 point de vie	15 mètres
Monstre3	100 points de vies	7.5 mètres

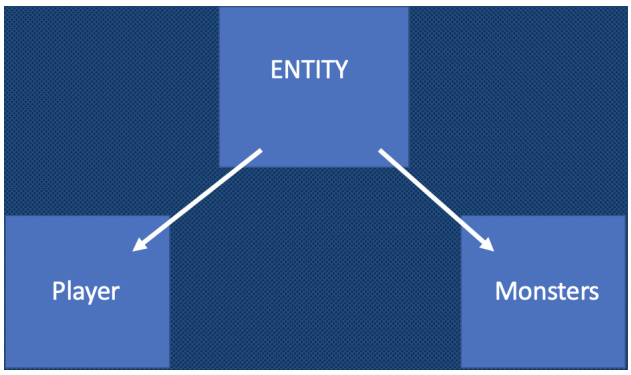
Brice vous détailleras le premier monstre.

3.1.2 Orienté objet

1ère soutenance :

Avec Brice, nous avions prit soin dès le début d'avoir un projet basé sur les mécaniques de l'orienté objet avec ce que nous avions vu en TP.

Pour cette soutenance, j'ai implémenté toutes les classes qui me serai utile pour la suite. Voici un schéma un peu plus parlant :



Donc c'est plutôt simple, comme vous pouvez le voir les entités de notre jeu comme le joueur ou encore les monstres, hérite de la super-classe "Entity".

Avoir commencé l'orienté objet dès cette soutenance nous a permis beaucoup plus flexibilité dans notre code au fur et à mesure que nous avancions dans notre projet. Il était très simple de gérer les différents attributs du joueur comme par exemple la nourriture.

J'ai pris soin de placer tous les attributs en private avec bien sûr un getter pour y accéder en dehors de la classe.

```
1         public player(float hp, float food, float sleep, float water)
2           : base (hp, 10, Mob.player) //Constructeur
3       {
4           this.food = food;
5           this.sleep = sleep;
6           this.water = water;
7           maxlife = 100;
8           capCourd = 50f;
9           gourd = 0f;
10          capbriquet = 100f;
11          briquet = 100f;
12      }
```

Voici, ci-dessus, le constructeur du joueur que j'ai implémenté pour cette soutenance. J'ai aussi crée différentes méthodes permettant par exemple : régénérer de l'eau, de la vie, de la nourriture ou encore perdre ou gagner de la vie.

Par exemple, lorsque l'un de nos monstres doit infliger des dégâts au joueur, il suffit d'appeler la méthode "TakeDamageHP" sur l'instance de notre player.

```
1         public void takedamage()
2     {
3        GameObject.Find("FPSController").GetComponent<Player>().player1.TakeDamageHP(15);
4     }
```

2ème soutenance :

Pour cette soutenance concernant l'orienté objet, je me suis occupé de gérer le temps dans la classe du joueur, car en effet pour notre projet le temps est un élément essentiel au gameplay .

```
1         public void updatetime(int hour)
2         {
3
4            float min = (20f * hour) / 24f;
5
6            float food12 = ((min * (1f / 3f)) / 20f)*100;
7
8            food = food - food12;
9
10           float water12 = ((min * (1f / 2f)) / 20f)*100f;
11
12           water = water - water12;
13
14           float dodo12 = ((min * 1f) / 20f)*100f;
15
16           sleep = sleep - dodo12;
17
18        }
```

Voici ce que j'ai implémenté concernant ce point. Cela à permis à Brice de gérer le temps dans le craft, je le rappel, dans notre jeu le temps c'est de l'argent ou plutôt l'argent c'est le temps.

En effet, lorsque Brice fait appel à cette méthode sur l'instance du joueur, il passe en paramètre la durée du craft d'un item, cette durée fait passé tant de temps dans le jeu. Il faut donc impacter les différentes barres du jeu car le joueur été censé se nourrir, se réhydrater, dormir... Et c'est donc cette méthode que j'ai implémenté pour cette soutenance qui permet de faire cela en respectant les conventions donnés dans le cahier des charges.

3ème soutenance :

Pour ma part, l'implémentation de l'orienté objet été terminée à 99% pour la 2ème soutenance. Pour celle-ci je me suis juste occupé à ajuster certaines valeurs pour équilibrer le jeu comme par exemple la capacité de la gourde que j'ai diminuée car il était trop facile de stocker des grandes quantités d'eau.

3.1.3 Combat

1ère soutenance :

Pour cette première soutenance, j'ai implémenté une partie de la mécanique des armes de corps à corps. En effet on peut déclencher l'attaque en appuyant sur le clic gauche de la souris et frapper notre chère monstre à l'aide d'animations que nous verrons par la suite. La première arme de combat que j'ai implémenté est le marteau, disponible pour cette soutenance.

2ème soutenance :

j'ai beaucoup avancé dans cette partie en implémentant les armes à distances comme l'arc ou le harpon pour cette soutenance. La mécanique de l'arc que j'ai mise en place est particulièrement intéressante car elle permet de gérer la force que le joueur met dans la souris. Voici ci-dessous ce que j'ai réalisé :

```
1         IEnumerator addforce()
2     {
3        f = false;
4        yield return new WaitForSeconds(0.1f);
5        puissance = puissance + 100;
6        recule += 10;
7      GetComponent<Rigidbody>().AddForce(transform.TransformDirection(Vector3.down*recule))
8        f = true;
9     }
```

J'ai déjà eu l'occasion de parler longuement des IEenumerator durant les précédents rapports de soutenances, mais petit rappel ils permettent

de faire une "pause" dans le code avec *yield return new WaitForSeconds(0.1f) ;*, qui ici s'arrête 0.1s sur cette ligne avant d'aller à la suite du code. Ce qui permet lorsque l'on maintient le clique gauche enfoncé d'ajouter de la force toute les 0.1s. Puis lorsque l'on relâche le clic gauche :

```
1 GetComponent<Rigidbody>().AddForce(transform.TransformDirection(Vector3.up*puissance));
```

Puissance, c'est la variable qui a emmagasinée la force que le joueur a ajoutée en maintenant le clic gauche et vector3.up la direction, ici vers l'avant. Et avec ça on utilise la propriété "AddForce" qui comme son nom l'indique permet d'ajouter une force.

J'ai aussi implémenté un harpon, en revanche le rendu ne nous convenait pas et nous avons donc décidé le ré-implémenter pour la dernière soutenance.

3ème soutenance :

Pour cette dernière soutenance, je me suis occupé de la ré-implémentation du Harpon. C'est n'est donc plus une arme comme l'arc qui tire à distance mais bien une très longue armes au corps à corps. Le harpon se décline en deux version amélioré et basique et j'ai donc implémenté les deux, le harpon amélioré faisant plus de dégâts aux monstres et donnant plus de poissons à la pêche.

De plus pour les armes au corps à corps que cela soit le marteau ou le gourdin, j'ai réalisé de nouvelles animations pour ne pas que le jeu soit trop monotone, d'ailleurs j'ai aussi implémenté le gourdin pour cette soutenance qui a une mécanique similaire au marteau, même si cette arme est moins puissante que le marteau comme dit dans le cahier des charges. En effet ces animations varient un coup donné par le joueur (déclenchée à l'aide du clic gauche) sur deux. J'ai aussi corrigé des bugs, par exemple on pouvait mettre des dégâts avec le briquet, ce qui n'est pas correct et j'ai donc résolu cela.

De plus certains problèmes ont commencés à se présenter par exemple, si le joueur cliquait très rapidement sur sa souris ou encore qu'il restait enfoncé dessus cela infligeait des dégâts de manière continue et beaucoup trop rapide au monstre, ce qui rendait le monstre très simple a tuer peu importe l'arme. Pour remédier à cela j'ai crée un petit timer avec les corroutines, expliqués au dessus, de 1 seconde dans le but que le joueur ne puisse infliger des dégâts qu'une seul fois par seconde, ceci est synchronisé avec les animations qui dure une 1 seconde.

De plus, pour les soutenances précédentes nous préférions nous assurer que nos mécaniques de toutes les armes marchent le mieux possible pour

commencer à réellement implémenter les caractéristiques, notamment les dégâts. Pour cela, avec Brice, nous avons exploité le potentiel de l'orienté objet. Premièrement la classe "Entity" dont hérite la classe player possède un attribut "Dammage" correspondant aux dommages que cette entité va infliger, mais il faut alors les actualiser en fonction de l'arme, j'ai alors crée une méthode qui permet d'actualiser les dammages et Brice vous expliquera comment il gère cette méthode dans le script de changement d'armes. C'est d'ailleurs avec cela que l'on a corrigé le problème du briquet car ses "Dammages" sont à 0, il n'inflige donc plus de dégâts.

Pour finir, je me suis occupé du système de munition des flèches dont je reparle dans le craft. Si le joueur à au moins une flèche dans l'inventaire je l'affiche dans la main. Sinon je la désactive. S'il il tire une flèche et qu'il en a encore au moins une dans l'inventaire, une flèche invisible est placée par défaut dans la main dans le but d'enregistrer la position de la flèche tirée qu'il faut remettre dans la main. Je vérifie dans l'inventaire de Brice grâce à ses méthodes le stock de flèches. Je la rajoute en conséquence dans la main, s'il en a plus elle disparait jusqu'à qu'il en rachète une.

3.1.4 Site web

1ère soutenance :

Concernant le site web pour la première soutenance, je me suis occupé de mettre en place un VPS (Virtual private server) loué chez OVH, que cela soit au niveau de la sécurité ou de l'administration.

2ème soutenance

Correction de fautes d'orthographes sur les site, création d'un premier trailer.

3ème soutenance :

J'avais personnellement prit un petit retard sur ce que j'avais initialement prévu de faire sur le site web. Toutefois, je me suis rapidement mis au travail et j'ai commencé à maitriser les outils web, tel que HTML 5, CSS 3 ou encore Java script et j'ai crée une page de présentation de notre projet ainsi qu'une page concernant les médias. Voici la page de présentation de notre projet :

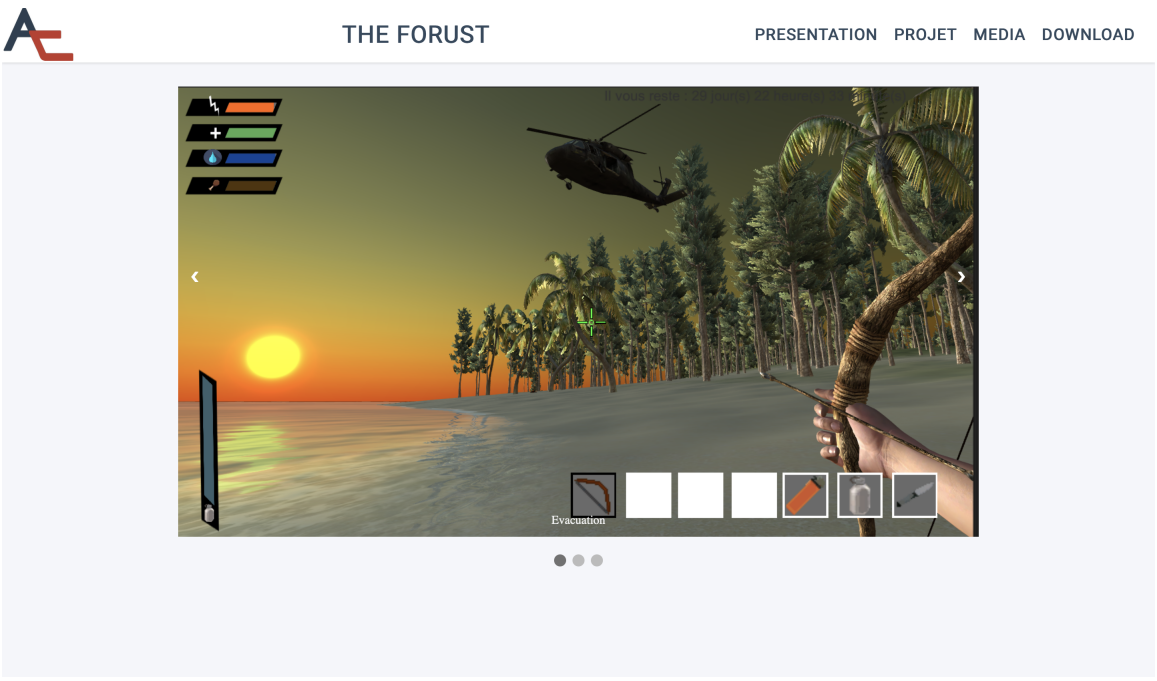
Ce qu'il est possible de faire avec cette page : On peut cliquer sur, par exemple, "Présentation du projet" et lorsque l'on clic, avec une animation que j'ai réalisé à l'aide du JavaScrit et du CSS, les information concernant "Présentation du projet" vont s'afficher de manière progressive de haut en bas pour donner un côté plus "smooth" et jolie. Il en va de même pour la présentation des membres du groupe. En ce qui concerne le JavaScript, il m'est utile pour détecter le clic de la souris et d'afficher les différents blocks.

Finalement le CSS pour le fond, l'HTML pour la forme et le java script pour lier les deux.

Sinon, l'intérêt pour le visiteur ? Il peut sélectionner les informations à afficher ou non et accéder directement au contenu qui l'intéresse.

Pour cette soutenance je me suis occupé de créer une page média avec les mêmes outils que précédemment.

Je pense qu'il est important pour notre visiteur d'avoir un aperçu de notre projet rapidement, pour pourquoi pas télécharger notre jeu. L'objectif ici si je peux me permettre, est de lui mettre l'eau à la bouche à travers le visuel du jeu. Voici à quoi elle ressemble :



Tout d'abord, c'est une page qui va regrouper les différentes vidéos et photos de notre jeu si le visiteur veut un aperçu de ce dernier.

Donc le visiteur peut se déplacer de deux façon différentes pour voir les images ou vidéos, soit avec les petites flèches sur les côtes de l'image ou bien les points en-dessous de l'image. La mise en forme avec avec le CSS et HTML est plutôt simple, juste bien placer les flèches et les points ainsi que bien dimensionner les images sur la page.

Pour le Java script c'est lui qui va s'occuper d'activer et de désactiver les images qu'il faut afficher ou non. Les images, du moins les blocks contenant les images, sont stockés dans un tableau. Il suffit alors d'allez activer l'image que l'on souhaite à l'aide de son indice dans le tableau et désactiver les autres, tout en prenant en compte les limites du tableau.

En plus de cela je me suis occupé de réaliser un trailer pour notre jeu à l'aide du logitiel Adobe Premiere pro. Il montre rapidement ce qu'il est possible de faire dans le jeu et un aperçu de la map au travers de jolie un plan et d'un (somptueux) montage vidéo (sans ironie).

3.1.5 Interface utilisateur

1ère soutenance :

Pour la première soutenance, j'ai implémenté un menu d'accueil avec un bouton play et quit, donc assez basique. J'ai également mis en place un menu pause, qui comme son nom l'indique permet de faire pause. Le joueur peut soit reprendre le jeu ou bien quitter le jeu. J'ai mis en place les quatre barres relatives à sa survie que le joueur peut voir.

Je savais déjà que ce n'était pas le résultat que je voulais et comme vous allez pouvoir le voir, ceci à beaucoup évolué pour la deuxième soutenance. Toutefois ce que j'avais fait avait le mérite d'être fonctionnel dès la première soutenance et les différentes mécaniques n'ont pas énormément évoluées en elles-mêmes.

2ème soutenance :

C'est avec des idées pleins la tête que je me remettais en route pour la deuxième soutenance dans le but d'améliorer tout cela.

Pour ce faire j'ai utilisé Photoshop ainsi que les icônes que Brice à réalisé en pixel art comme par exemple le poulet à côté de la barre de nourriture (la barre brune). Dans la partie dédiée à ces barres j'expliquerai en détails comment elles se sont mises en place. Sinon avec les deux images ci-dessous, vous pouvez voir que les barres sont beaucoup plus jolies et plus lisibles que des quarts de cercle de couleurs. La barre de la gourde s'est aussi harmonisée avec le reste tout comme la barre affichant la capacité du briquet.

Sur les nouvelles barres un dégradé a été appliqué. Par exemple pour la barre de vie elle démarrera au vert puis fini au rouge si le nombre de point de vie restant au joueur est critique. il en va de même pour les autres barres. Si la couleur restait tout le temps pareil le regard du joueur aurait tendance à oublier les différentes barres. Ce changement visuel oblige le joueur, du moins nous l'espérons à porter son regard sur les différentes barres du jeu.

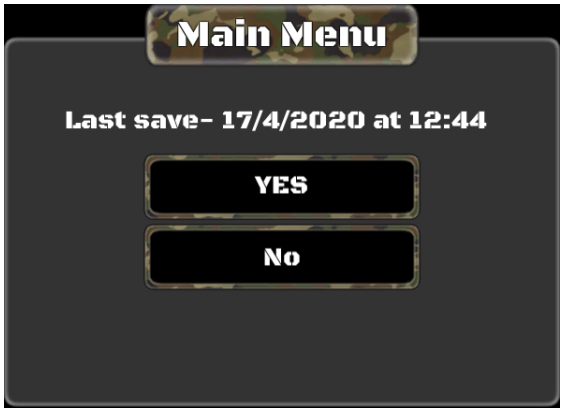
FIGURE 1 – 1ère soutenance

FIGURE 2 – 2ème soutenance

Comme vous pouvez le constater avec ces deux images comparant le deux soutenances un bon en avant a été fait dans la clarté et dans le design. Je suis plutôt fière de ce que j'ai fait.

Le deuxième gros morceau auquel je me suis attaqué est le système de save. Pour cette soutenance, en appuyant sur le bouton "Save" disponible dans le menu pause le joueur peut enregistrer toutes les informations relatives à ce dernier, la vie, sa position, le temps déjà passé ect...

Lorsqu'il revient sur le jeu et qu'il appuie sur play, deux choix lui sont alors proposés si une sauvegarde existe, la date de la dernière save est affichée, il peut cliquer sur "Yes" s'il veut utiliser cette save ou appuyer sur non s'il veut relancer une partie. Comme une image est toujours plus parlante que 1000 mots Voici une image un peu plus parlante de ce que j'ai fait pour cette soutenance :



Dans cet exemple une sauvegarde a bien été effectuée, le jeu lui propose naturellement de relancer cette sauvegarde. Ce dernier peut refuser en appuyant sur "No". La date est une indication supplémentaire pour que le joueur puisse au mieux se repérer. Les données sont enregistrer à l'aide des playerspref :

```
1       PlayerPrefs.SetFloat("x",x);
```

La string est le nom que l'on donne à la donnée que j'enregistre. Pour la récupérer il me suffit de remplacer set par get avec seulement le nom de cette donnée. Il faut toujours vérifier qu'elle existe avec une des méthodes du namespace playerpref avant de faire cela.

3ème soutenance :

Pour cette dernière je me suis attaqué au menu option dans notre jeu.

Ce bouton est accessible dans le menu pause ou sur l'écran d'accueil du jeu, lorsque l'on clique dessus une liste d'options s'affichent dans lesquelles on retrouve notamment : la qualité du jeu (très faible, faible, medium, élevé, très élevé et pour finir ultra pour les personnes possèdant des machines aux performances élevées), la screen-size (la taille de l'écran) : une liste de taille lui est alors proposées lorsqu'il clique dessus parmi celle proposée par son écran, et enfin le volume global du jeu. Voici le rendu final :

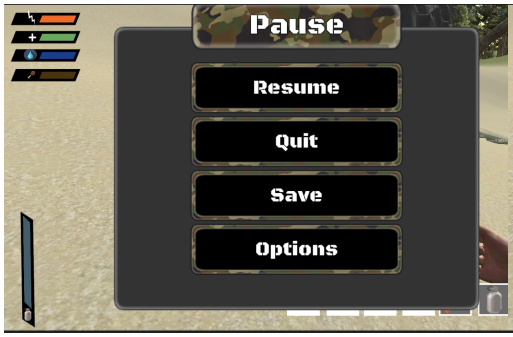


FIGURE 3 – Menu pause

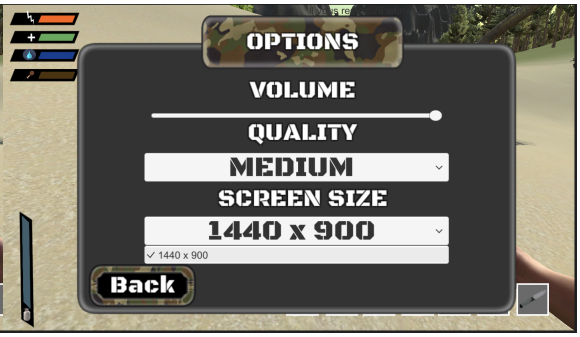


FIGURE 4 – Options

Concernant cette soutenance, le système de sauvegarde a fini d'être implémenté. Nous pouvons sauvegarder en plus des choses possibles à la deuxième soutenance : Les objets qui ont été posés sur la map, et en général les objets craftés sauvegardés réapparaîtront dans l'inventaire.

3.1.6 Animation

1ère soutenance :

Les animations font partie intégrantes de notre jeu, et pour cette première soutenance j'ai crée une animation qui fait bouger le bras de notre joueur lorsque ce dernier marche. De plus j'ai crée la première animations lorsque le joueur frappait avec l'arme de corps à corps (le marteau).

En ce qui concerne les animations du premier monstre j'ai implémenté dans le jeu son animation de déplacement ainsi que son animation lorsque ce dernier frappe le joueur.

De plus j'ai crée l'animation de la cinématique lorsque le joueur arrive sur l'ile.

Mais il fallait absolument diversifier les animations du joueur et du monstre pour les prochaines soutenances.

2ème soutenance :

Comme précisé dans la section AI, deux nouveau monstres ont été implémentés pour cette deuxième soutenance. Donc ces derniers possèdent 3 animations : une pour l'état où ils ne font rien (Animations qu'il les fait regarder dans toutes les directions.), l'animation de marche lorsqu'ils se di-

rigent vers le joueur et enfin l'animation d'attaque.

Même si elles servent à la même chose, toutes les animations sont différentes entre les monstres. J'allais oublié de préciser, ces animations sont trouvées sur le site "Mixamo" qui est une banque d'animations, je me charge de les implémenter dans le jeu avec le C# et unity. En effet il est nécéssaire de les déclencher au bon moment.

De plus, j'ai implémenté l'animation qui vient chercher le joueur en hélicoptère lorsque ce dernier gagne la partie. J'ai donc implémenté un script, qui permet au joueur lorsqu'il s'approche de l'hélicoptère de monter dedans. Cela grâce aux rayons qui seront expliqués après, ils me permettent de repérer l'hélicoptère au sol.

J'ai essayé de faire l'animation du harpon or elle était moche (appelons un chat un chat). J'ai donc eu immédiatement pour objectif une fois cette soutenance fini de la refaire proprement.

3ème soutenance :

Les animations de notre jeu manquaient de diversité, je me suis donc occupé de remédier, avec Brice, à cela. Premièrement, j'ai rajouté une animation supplémentaire lorsque le joueur frappe avec une arme de corps à corps.

```
1           void Update()
2     {
3
4
5        if (Input.GetMouseButtonDown(0))
6        {
7           Debug.Log(i);
8
9           if (i % 2 == 0)
10          {
11
12             pierre.SetBool("Attack", true);
13          }
14          else
15          {
16             pierre.SetBool("Attack2", true);
17          }
18
19       }
20    }
21
22
23    public void returndefaultstate()
```

```
24      {
25          pierre.SetBool("Attack", false);
26          pierre.SetBool("Attack2", false);
27          i++;
28      }
```

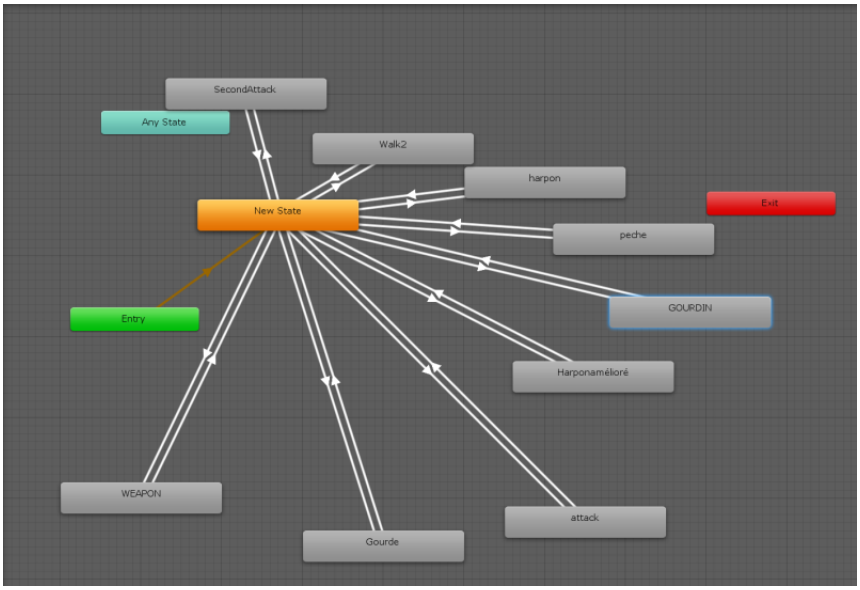
Voici comment tout ceci fonctionne, comme on peut le voir ces animations s'activent un clic sur deux. A la fin de des animations, *"returndefaultstate()"* est appelé et vient incrémenter la valeur "i" pour passer à l'animation suivante tout en s'assurant de désactiver les animations inutilisées.

J'ai aussi implémenté différentes animations d'attaques trouvées sur le site "Mixamo" pour le dernier monstre, le plus fort. Une attaque très puissante qui est un saut sur le le joueur lorsque celui ci s'approche pour la première fois de lui, puis une animation d'attaque légère qui est un coup de point sur 360 puis enfin pour terminer une attaque ultime infligeant tout de même beaucoup de dégâts. Il a fallu que gère le faite que le joueur puisse esquiver l'attaque en ralentissant le début des animations, puis créer une zone spécifique pour que le joueur puisse recevoir les dégâts s'il s'y trouve au moment du coup, cela signifie que le joueur n'a pas réussi à esquiver le coup. Le combat devient donc vraiment intéressant avec ces différents pattern.

Comme annoncé précédemment, j'ai entièrement refait l'animation du harpon, même plus que ça, cela ce distingue maintenant en deux animations distinctes : l'harpon en tant qu'arme et l'harpon en tant qu'outil de pêche.

Lorsque le joueur vise un plan d'eau dans lequel il est possible de pêcher, l'animation de pêche est déclenchée cette animation fait lancer le harpon, c'est grâce aux rayons dans unity dont je reparlerai par la suite . Dans tous les autres cas l'animation fait comporter l'harpon un peu comme une lance. Tout ceci au clic gauche de la souris bien sûr.

Pour récapituler, voici toutes les animations relative au joueur :



Pour rappel, les flèches blanches représentent des transitions sur lesquelles peuvent être posé des conditions contrôlées à l'aide du C# permettant de passer d'une animation à une autre.

Donc ici il y a les deux animations du harpon, les deux animations de l'attaque des armes au corps à corps et l'animation de marche. On peut aussi noter une animation supplémentaire pour le gourdin et du harpon amélioré.

3.1.7 Gestion des différentes barres et du temps

1ère soutenance :

Pour cette première soutenance j'ai implémenté les bases de la mécanique de la gestion des barres. Notamment le fait de raccourcir ou de s'agrandir en fonction que l'on perd de la vie ou bien que l'on en regagne. Voici ce qui gère cela de la première à la dernière soutenance :

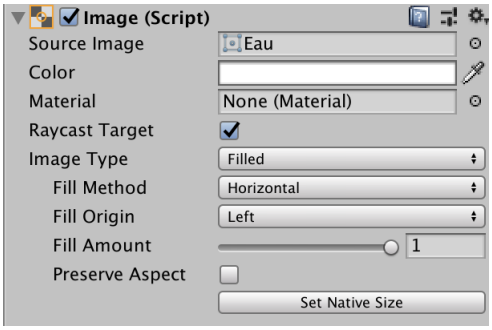
```
1           private void Update()//here uptade one per frame
2     {
3         eau.fillAmount = player1.Water / player1.Max;
4         food.fillAmount = player1.Food / player1.Max;
5         dodo.fillAmount = player1.Sleep / player1.Max;
```

```
6         life.fillAmount = player1.Hp / player1.Max;
7
8     }
```

Encore une fois, nous essayons au maximum d'exploiter l'orienté objet, qui permet de faire cela simplement. En effet les attributs du player sont très souvent entrain d'évoluer grâce aux différentes méthode implémenté dans la classe. Et avec ces quatre lignes ci-dessus nous n'avons plus jamais besoin de s'occuper de l'actualisation des barres de vie.

En effet cette méthode update est appelée une fois par image affichée, par exemple 30 images par seconde = 30 appel à cette fonction. Elle vérifie directement dans les attributs du joueur et actualise la taille des barres en fonctions du maximum.

Voici comment sont représentées les barres dans Unity :



Pour ce faire, j'ai passé les barres en type "Filled", c'est à dire en mode remplissage. La propriété "Fillamount" remplit l'image en fonction de sa taille maximum. Comme vous pouvez le voir ce remplissage ce fait de manière horizontal pour rendre le tout jolie. Et c'est donc sur cette propriété "Fillamount" que je vais jouer. Les différentes barres diminuent au fur et à mesure du temps qui passe. Toutefois elles n'été pas calibrée sur l'écoulement du temps annoncé dans le cahier des charges.

2ème soutenance :

J'ai dédié une partie sur cela dans la section affichage mais j'ai refait le design global des barres de la première à la deuxième soutenance.

Comme expliqué dans le cahier des charges et dans nos rapports, le temps est une mécanique importante de notre jeu qui doit être dépensé judicieusement, c'est une sorte de monnaie qui impact les différentes barres.

Le temps s'écoule, les différentes barres doivent en être impactées. En effet, pour rappel dans notre jeu 24h durent 20 min temps réel.

De plus, dans une journée de jeu on perd automatiquement 1/3 de notre barre de nourriture et 1/2 de notre barre d'eau et entièrement la barre de sommeil. En dessous d'un certain seuil des barres de nourriture et d'eau, la barre de vie se met à diminuer elle aussi. Je me suis donc occupé de tout cela dans le jeu. J'ai donc réalisé ce script, ici pour la nourriture :

```
1        IEnumerator food()
2    {
3       while (_player.player1.Food >= 0)
4       {
5          yield return new WaitForSeconds(1f);
6          if (_player.player1.Food > 50)
7          {
8             _player.player1.UpHP(0.0004081f*100f);
9             _player.player1.TakeDamageFood(0.00027f*100f);
10         }
11         else if (_player.player1.Food >= 20 &&_player.player1.Food <= 50)
12         {
13            _player.player1.TakeDamageFood(0.00027f*100f);
14            _player.player1.TakeDamageHP(0.00081f*100f);
15         }
16         else if(_player.player1.Food <= 20)
17         { _player.player1.TakeDamageFood(0.00027f*100f);
18            _player.player1.TakeDamageHP(0.00122f*100f);
19         }
20
21     }
```

Au dessus de 50 point (sur 100) de la barre de nourriture, le joueur peut régénérer sa vie. En revanche, en coordination avec le cahier des charges, le joueur perd 1/3 de cette bar par journée de jeu, c'est à dire 20 min. Il faut donc faire la conversion pour une seconde pour que cette barre s'écoule continuellement et non brutalement. cela fait 1,37 point retiré par heure dans le temps du jeu. Tombé sous 50 points de cette barre cela passe à 4 points, et en dessous de 20 cela passe à 6. Cela permet de rajouter de la pression sur le joueur. Encore une fois, se sont les corroutines qui permettent de faire passer une seconde.

C'est un mécanisme similaire qui est appliqué aux autres barres avec les caractéristiques annoncé dans le cahier des charges. Voici donc globalement le travail que j'ai réalisé pour cette première soutenance concernant les différentes barres.

3ème soutenance :

J'avais pratiquement terminé le travail de cette partie, il s'est toutefois posé une question d'équilibrage dans le but de rendre le jeu jouable. En effet, par exemple la barre de vie ne se régénérait pas assez rapidement par rapport aux dégâts une fois tombé sous un certain seuil.

Nous allons évoquer l'implémentation de tous les items de notre jeu autre que ceux dédiés au combat car déjà abordé dans la section "Combat".

3.1.8 Items

1ère soutenance :

Pour cette première soutenance concernant les items, je me suis occupé d'implanter la gourde, qui permet au joueur de transporter de l'eau et de pouvoir la boire quand il le désire. Il n'y a qu'une seul source d'eau potable sur la carte à lui de la trouver. Dans notre jeu la gourde est représentée par une barre, comme vous pouvez le voire sur la figure 2 de la section "Interface utilisateur", qui peut se remplir ou à l'inverse se vider lorsque le joueur décide de boire dedans pour régénérer sa barre d'eau.

Cette dernière fait partie par défaut du joueur, je me suis donc chargé de l'intégrer à la classe du joueur.

```
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       }
```

Voici ce que j'ai réalisé lorsque le joueur s'approche de ce point d'eau et qu'il appuie sur la touche d'interaction "E", pour savoir si le joueur est proche du point d'eau je me sers des rayons. C'est une méthode de la classe player.

Tout d'abord je m'assure que la capacité actuelle de la gourde n'est pas au maximum, cette capacité est de 50. Sinon je lui donne 10 dans la gourde, comme expliqué plus haut, la barre représentant la capacité de la gourde s'actualise immédiatement. Après cela il doit attendre 1 seconde dans le

but d'éviter les bugs, avant de pouvoir remplir de l'eau dans sa gourde. Le booléen "f" me permet de vérifier cela, après 1 seconde écoulée à l'aide de la corroutine il passe à "true" et le joueur peut reboire. L'implémentation de cet objet été donc terminé et totalement fonctionnelle pour cette soutenance.

2ème soutenance :

J'ai implémenté pour cette soutenance un item disponible dès le début du jeu pour le joueur et qui permet au joueur d'allumer le feu permettant de cuire la nourriture, le briquet.

Ce dernier possède un réservoir qui diminue à chaque utilisation lorsque l'on possède le briquet dans la main. Etant donné par défaut je l'ai lui aussi intégré à la classe player. Comme la gourde, je lui ai mis un attribut indiquant sa capacité actuelle, et bien sûr un attribut indiquant sa capacité maximun. J'ai rendu possible son activation à l'aide du clic gauche de la souris. Sur une capacité de 100 initial, il perd 10 seconde par seconde où le clic est enfoncé. Pour rajouter une difficulté supplémentaire, il est impossible au joueur de régénérer cette capacité au cour de la partie. Le joueur devra gérer intelligemment son utilisation. En ce qui concerne le script que j'ai réalisé en voici une partie :

```
1   IEnumerator capacity()
2    {
3       current = current - 10;
4        f = false;
5       yield return new WaitForSeconds(1f);
6        f = true;
7    }
8
9    void Update()
10   {
11      if (Input.GetMouseButton(0))
12      {
13         if (f && current >0)
14         {
15            briquet.enabled = true;
16            fire.SetActive(true);
17            StartCoroutine(pause());
18         }
19         p = true;
20      }
21
22      if (Input.GetMouseButtonUp(0))
23      {
24         briquet.enabled = false;
```

```
25              fire.SetActive(false);
26              p = false;
27          }
28      }
```

Pour rappel, la méthode "Update()" est celle appelée par chaque images que génèrent notre ordinateur. En possession du briquet, une image du réservoir briquet s'affiche au clic gauche de la souris, cette image est bien entendu une barre qui diminue au fur et à mesure de son utilisation. Finalement le mécanisme est plutôt similaire à la gourde. Le booléen f permet de vérifier que une seconde d'utilisation s'est écoulée et que l'on peut à nouveau diminuer la capacité du briquet. Lors du clic, si la capacité le permet, on affiche la flamme du briquet.

Cette barre est actualisée de la même manière que les autres barres. Avec tout cela, en collaboration avec Brice, nous nous sommes occupés du feu de camp qui je le rappel, permet la cuisson de notre nourriture comme le lapin et le poisson.

Je me suis pour ma part occupé de la bonne activation du feu de camp. Il possède un "collider", une sorte de "boite" autours de l'objet possédant certaines propriétés dont une qui m'est plutôt très utile, la possibilité de détecter quelque chose qui pénètre dedans, notamment le joueur. Il me suffit de vérifier que la flamme du briquet est allumée pour allumer le feu de camp. Je vérifie dans tous les cas à l'allumage de flamme du briquet que sa capacité permet d'allumer cette flamme.

J'ai aussi commencé la mécanique du sommeil à l'approche d'une cabane, peut importe le niveau de celle-ci. En effet pour le sommeil, j'ai fait en sorte que si le joueur s'approche d'une cabane construite un texte apparait lui proposant de dormir en lui régénérant sa barre de sommeil tout en impactant ses autres barres comme la nourriture et l'eau. Toutefois il faut que je rajoute un voile noir pour simuler le sommeil.

Voici donc le travail que j'ai réalisé concernant cette partie pour cette soutenance.

3ème soutenance :

Pour cette dernière soutenance, j'ai implémenté le grand feu de camp qui a partir du moment de sa construction donne une chance au joueur supplémentaire de quitter l'ile. Depuis le début du projet notamment dans le cahier des charges, ont vous parlé d'un moyen secondaire de quitter l'ile. A tout moment de la partie, un avions peut passer dans le ciel.

S'il le joueur construit le grand feu camp et qu'il l'allume à l'aide du briquet lorsque l'avion passe dans le ciel, il a une chance sur deux de ce faire évacuer de l'ile. L'avion reste deux minutes dans le ciel en faisait le tour de l'ile avant de définitivement disparaitre. Si la chance du joueur est bonne alors la partie s'arrête. J'ai aussi rajouté le voile noir de la cabane pour simuler le sommeil.

J'ai aussi implémenté toutes les mécaniques qui se passent durant le sommeil du joueur, c'est à dire faire réapparaitre aléatoirement tous les items présents sur la carte à des points que j'ai fixés. Le bois, les feuilles, les pierres, lianes et baie sur les buissons apparaissent de manière aléatoire sur la moitié des points disponible.

Voici un exemple sur les bâtons de ce que j'ai réalisé :

```
1   foreach (Transform baton1 in baton.transform)
2       {
3          stockbaton[k] = baton1.gameObject;
4
5          k++;
6       }
7    public void spawn()
8     {
9
10         int k = 0;
11
12         foreach (Transform baton1 in baton.transform)
13      {
14
15         baton1.gameObject.SetActive(false);
16         k++;
17      }
18      while (checkifactiv(stockbaton)<baton1)
19      {
20         stockbaton[Random.Range(0, baton.transform.childCount)].SetActive(true);
21      }
22    }
23
24    public int checkifactiv(GameObject[] Stock, bool baie = false)
25     {
26           int k = 0;
27           foreach (GameObject item in stock)
28           {
29              if (item.transform.GetChild(0).gameObject.activeSelf)
30              {
31                 k++;
32              }
33           }
34
35      }
```

Premièrement je remplie un tableau de tous les bâtons, puis pour ne pas avoir de problèmes je les désactives tous à l'appel de la fonction spawn (celle qui est appelée lors du sommeil), ensuite j'ai une fonction "checkifactiv" qui permet de vérifier combien on été activé, de bâtons dans cet exemple. Tant que la moitié n'a pas été activée, on réactive dans le jeu un bâton de manière aléatoire. Cette condition dans la boucle while permet d'être sure d'avoir toujours le bon nombre de bâtons activés.

Il y a juste pour les lapins et collet que cela diffère en effet à l'appel de "spawn", si un collet est posé, il a une chance sur quatre qu'un lapin réapparaisse dedans pendant la nuit.

3.1.9 Mécanique de gameplay/Ramassage

Ici les mécaniques de gameplay concerne toutes les fonctionnalités disponible pour le joueur et utile à sa survie, autre que le craft et le combat qui mérite leur partie dédiés.

1ère soutenance :

Pour cette première soutenance j'ai réalisé le système de ramassage, en apprenant l'utilisation des rayons dans Unity essentiel à notre jeu vu le nombre de fois où je les ai mentionnés auparavant. C'est donc ici que vient leur explication.

Les rayons dans unity est une sorte de segment qui a donc un début (le point sur lequel on le place donc très souvent notre joueur), une longueur et une direction. Dès que ces rayons vont venir frapper un objet nous allons obtenir une grande quantité d'information sur cet objet qui vient d'être touché comme par exemple son nom, la distance par rapport au point de départ du rayon.

Vous devenez commencer à comprendre pourquoi ils nous sont si utiles, rien que avec ces deux informations nous pouvons faire énormément de chose dans notre jeu comme vous avez pu le constater, notamment le système de ramassage. Jean s'est occupé sur notre carte de disposer plein d'items ramassable comme des batons, lianes, pierre ect... Je me suis occupé à placer un viseur sur l'écran du joueur, le rayon démarre sur le viseur et il suit toujours sa direction, notamment lorsque le joueur bouge la souris.

Voici ce fameux viseur que j'ai mis en place que le joueur voit au milieu de son écran pour l'aider à réaliser toute sorte d'action, frapper les monstres, ramasser par terre.

Concernant le ramassage, il me reste juste à vérifier si ce que le rayon tappe est bien est bien un item ramassable et que la distance est inférieur à 2 mètres pour être sure que le joueur est à proximité de cet item. Si tel est le cas et que le joueur appuie sur E la touche d'interaction, j'appel les méthodes que Brice a créee qui permettent d'ajouter dans l'inventaire.

Voici le rayon que j'ai crée :

```
1  Ray ray = new Ray(Arme.transform.position,
2  Arme.transform.TransformDirection(Vector3.forward*10));
3        RaycastHit hit;
```

Comme vous pouvez le voir il a une direction, vers l'avant par rapport à la croix verte du viseur et une distance de 10 mètre. La variable hit s'occupe de récolter les information de ce que vient d'heurter le rayon. C'est grâce à elle que je peux faire toute mes vérifications.

2ème soutenance :

Pour cette deuxième soutenance je me suis occupé à implémenter ces deux mécaniques : un viseur et un arc.

Pour ce qui est du viseur sur l'arc une caméra est placée sur l'avant de l'arc. La visé s'active à l'aide du maintient sur le clic droit de la souris. Les différentes caméras dans Unity possèdent un ordre de priorité, sur lequel je vais venir jouer en C#. En effet lors du maintient du clic gauche de la souris la caméra sur le devant de l'arc devient prioritaire sinon c'est la caméra normal placée sur le joueur. Il y'a toute fois fallu que je modifie la

croix verte pour l'adapter à ce système de visé de l'arc.

Ensuite j'ai écrit l'algorithme qui a permis à Brice de mettre en place le changement d'arme :

```
1       Entier current = 0;
2       Entier previous = 0;
3
4       A chaque image
5       {
6
7       previous = current
8
9       si la molette de la souris bouge :
10
11      current +- 1 selon le sens de la souris, modulo la taille du tableau qui
12      stock nos armes.
13
14      si previous est different de current :
15
16      Appeler des fonctions qui vont venir desactiver et activer les armes qu'il faut
17      }
```

Pour rappel les armes sur l'écran du joueur sont stockées dans des slots dans lesquelles il peut naviguer à l'aide de la molette de sa souris. Current et previous me permettent de savoir si la souris à bouger et de faire le nécéssaire tout en prenant en compte les dimensions du tableau.

3ème soutenance :

J'avais personnellement fini mon travail pour cette partie, je me suis concentré à chercher des bugs dans le but de les corriger. Toutefois, ces recherches se sont révélées infructueuses.

3.1.10 Inventaire

1ère soutenance :

Pour cette première soutenance, je me suis occupé de lier la première version de l'inventaire que Brice a développé au joueur, en effet il faut bien le rendre disponible en jeu sous pression d'un bouton et s'assurer au maximum qu'il n'y est pas au bug majeur. Pour cela le joueur pouvait ouvrir l'inventaire en appuyant sur "A" et le fermer par une pression sur cette même touche. De plus en temps normal le curseur de la souris est verrouillé et non visible, je me suis donc occupé de le déverrouiller et de le rendre visible

à l'ouverture de l'inventaire.

2ème soutenance :

J'ai refait exactement la même chose qu'à la première soutenance avec le nouvel inventaire de Brice en canvas qu'il vous expliquera bien plus en détails.

3ème soutenance :

Pour cette dernière soutenance j'ai corrigé un bug que j'ai constaté à l'ouverture de l'inventaire, il y avait une interférence entre le joueur et ses items avec l'inventaire. Je m'explique si par exemple on avait l'arc dans les mains et que l'ont désirés acheter une flèches dans l'inventaire, on appuie sur le clic gauche pour la crafter, sauf que en même temps la flèche été tirée, ce qui est un peu dommage pour le joueur.

La raison ? Les script gérant les armes été toujours activées à l'ouverture de l'inventaire, je me suis donc occupé de bien les désactiver.

3.1.11 Craft

1ère soutenance :

Implémentation des items In-Game, notamment la gourde et le marteau. Comme déjà parlé dans la section "Mécanique de de gameplay/Ramassage", je me suis occupé d'ajouter les ressources ramassées dans l'inventaire qui vont servir au craft.

2ème soutenance :

J'ai continué l'implémentation des Items du craft comme le briquet et gérer l'allumage du feu camp. Et bien sûr de l'arc mais également de l'implémentation de la première version du harpon.

3ème soutenance :

Pour cette dernière soutenance je me suis occupé de gérer le craft des flèches et donc le système de munition. En effet, pour donner ou redonner une flèche dans la main du joueur il faut que je m'assure qu'il les a bien craftées dans l'inventaire et faire disparaitre la flèche de sa main lorsqu'il n'en a plus.

J'ai terminé l'implémentation de toutes les armes : nouvelle version du harpon, harpon amélioré, gourdin pour ainsi dire mais également du grand feu camp donnant au joueur une chance supplémentaire de gagner la partie. D'ailleurs j'entend par le mot "implémentation", toutes les fonctionnalités annoncées dans le cahier de charge.

3.1.12 Aspect graphique

1ère soutenance :

Implémentation du modèle 3D trouvé du 1er 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, J'ai également aidé Jean dans le façonnage la map, j'ai crée une cascade et une rivière, ces le point d'eau sous la cascade dans lequel le joueur peut boire. J'ai également ajouté des textures, notamment celle de sable au bord des points d'eau.

Vous pourrez retrouvez toutes les images légendés dans la partie annexe. Implémentation du modèle 3D du marteau trouvé sur l'asset store unity.

2ème soutenance :

Pour cette deuxième soutenance, j'ai ajouté le modèle du petit et grand feu de camp trouvé sur l'asset store Unity et le modèle du premier harpon trouvé sur le site GrabCAD. Et bien sûr les deux autres monstres trouvé également sur le site de mixamo.

Pour rendre le jeu plus réaliste j'ai ajouté un cycle jour/nuit avec un magnifique couché de soleil que vous pourrez admirer dans les annexes !

3ème soutenance :

Pour cette dernière soutenance, j'ai donné beaucoup plus de relief à la carte en ajoutant, des petits creux, vallées et collines. J'ai crée un espace un peu marécageux qui exploite ce nouveau relief. Notre carte n'était pas très verdoyante, j'ai donc remédié à cela pour cette soutenance en ajoutant des différents types de fleurs et herbes sur la carte notamment dans la partie marécageuse. Tout en prenant soin de faire des zones denses, d'autres moins en variant les différents types de végétaux. Je vous laisse regarder le travail accomplie dans les annexes.

3.2 Brice

3.2.1 Gestion des Items

Pour ma part, je me suis occupé de l'implémentation des items en tant qu'objet, c'est-à-dire que j'ai implémenté les items au niveau du code.

Dans ma démarche pour implémenter les items, j'ai commencé par une phase de réflection quant à la meilleur technique à adopter. J'ai assez rapidement décidé d'implémenter une superclass Item de la quelle toutes les class des items hériterai. J'ai alors donné à cette class 4 attributs fondamentaux des items : le nom, le nombre d'item que l'on possède, le nombre maximum d'items de ce type que l'on peut posséder et l'icone représentant l'item.

```
1    public int own;
2    public int max_own;
3    public string name;
4    public Sprite icone;
5    public int ielt;
6
7    public Items(int own, int max_own, string name, int ielt, Sprite icone)
8    {
9       this.own = own;
10      this.max_own = max_own;
11      this.name = name;
12      this.icone = icone;
13      this.ielt = ielt;
14   }
```

Vient ensuite l'implémentation des class héritant directement de la class Item. Pour cela j'ai cherché à regrouper les items dans des catégories, 3 catégories ce sont alors dégagés. Les objets ramassables (items ramassables), ce sont les items que l'on trouve naturellement sur l'île et qui servent uniquement pour construire d'autres items. Deuxièmement, la nourriture, les items appartenant à la catégorie nourriture sont des items que l'on trouve en réalisant des actions particulières, tel que pécher ou chasser. Ces items sont uniquement destiné à être mangé par le joueur, dans le but de remonter sa barre de faim. La dernière catégorie sont les items que l'on peut construire (les items craftables) à partir des ressources (les items ramassables).

Concernant les objets ramassables, je ne leur ai pas donné d'attribut en plus dans la nouvelle class, car les attributs de la class Items suffisent.

```
1    public Item_rammassable(int own, int max_own, string name, int ielt, Sprite icone) :
2    base(own, max_own, name, ielt, icone)
3    {
4
5    }
```

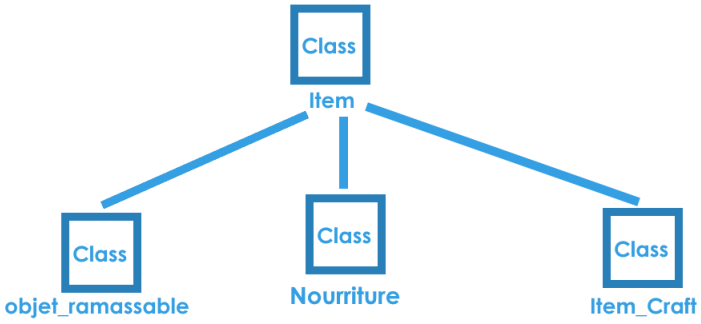
La class nourriture quant à elle a nécessité des attributs en plus. Un premier pour définir si l'objet nécessite d'être cuit pour être mangé et un autre définissant l'apport en nourriture au joueur.

```
1     public int foodGain;
2     public bool cook;
3
4     public Nourriture(bool cook, int foodGain, int own, int max_own, string name,
5     int ielt, Sprite icone) : base(own, max_own, name, ielt, icone)
6     {
7        this.foodGain = foodGain;
8        this.cook = cook;
9     }
```

Quant à la class des items craftables, j'ai du aussi y ajouter deux attributs : le coût de construction en ressource et en temps.

```
1     public int[] Craft_Cost = new int[4];
2     public int CraftTime;
3     public Item_Craft(int[] Craft_Cost, int CraftTime, int own, int max_own, string name,
4     int ielt, Sprite icone) : base(own, max_own, name, ielt, icone)
5     {
6      this.Craft_Cost = Craft_Cost;
7      this.CraftTime = CraftTime;
8     }
```

Voici l'arbre des class à cette étape :

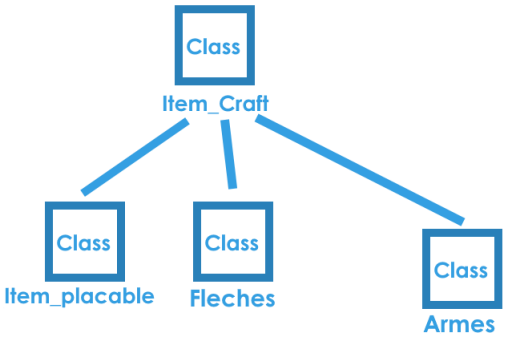


Ils n'y a aucune class qui hérite de la class nourriture et de la class items rammassables, car ces deux class définissent entièrement les items qu'elles représentent et il n'y a plus de distinction suffisante au seins des éléments définies par ces class pour nécessiter d'autres class.
La class item craft possède elle au contraire de nombreuses class héritant d'elle, car ils existes encore de nombreux type d'items craftables différents.

La class item craft se sépare en 3 branches. Les items placables, il s'agit des items que l'on peut placer sur la carte dans le but d'effectuer une action.

Par example la cabane qui sert à dormir, ou le collet qui sert a chasser. Les consommables, sont les items qui sont consommés pour faire fonctionner quelque chose. Les armes, il s'agit des items dont le joueur se servira pour combattre les monstre.
La class item placable, possède un nouvel attribut, "place" qui indique si l'objet à été placé sur la map.
La class consommable est un peu particulière car elle n'a pas été définie, les flèche étant les seul consommable du jeu. Il a donc été préférable de définir directement la class flèche héritant de la class item craft. Cette class ne possède pas d'attribut supplémentaire car elle est déjà parfaitement définie par ceux présent dans la class item craft.
La class armes possède deux nouveaux attributs : premièrement la distance à laquelle l'arme fait des dégâts et le nombre de dégât qu'elle inflige.

A cette étape voici le schéma des class des items craft :



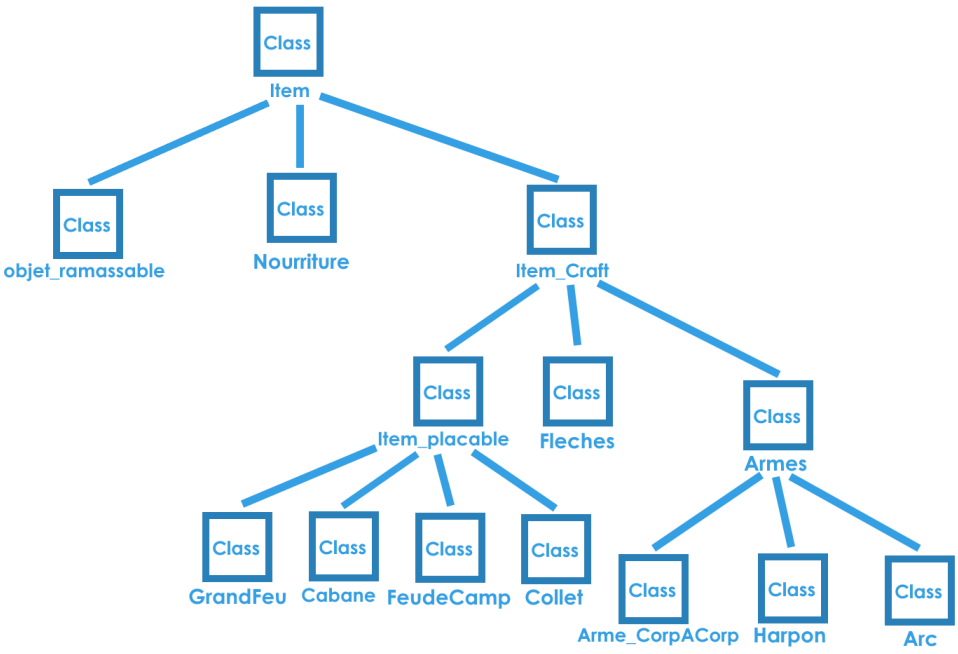
Des class item placable et item craft hérite directement les class des items en eux mêmes.
Pour la class item placable : le grand feu, les cabanes, le feu de camp, le collet.
Pour la classe armes : les armes au corps à corps, les harpons et l'arc.

Ces différentes class se contentent d'ajouter les attributs spécifiques a ces items, on peut citer la vitesse de pêche pour le harpon, car il sert aussi à pécher, ou le nombre de nuits où l'on est protégé dans la cabane, car elle sert à dormir.

Les différentes méthodes ajouté dans les class sont créé en fonction du besoin, au moment de l'implémentation d'une fonctionnalité dans le jeu. Par exemple la méthode get dans la class item ramassable, qui permet de récupérer un item par terre, a été implémenté lors de l'ajout du système de ramassage.

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

On obtiens finalement le schéma des class suivant en assemblant toutes les parties :



Les différents objets créés à partir de ces class, ont été créés dans le script InventoryObject de manière static dans le but d'y avoir un accès facile depuis n'importe où dans le projet, de même toujours dans le but de faciliter l'accès aux différentes donné plusieurs listes et tableaux, ont été

créés dans le but de regrouper les données. Il y a par exemple une liste regroupant les items craft, un tableau contenant les ressources disponibles, ...

Remarque de fin du projet :

Les attributs on été imaginés au début du projet en fonction de ce que l'on aurait besoins, en incluant le maximum de cas possibles dans le but d'éviter d'en rajouter plus. La tache étant fastidieuse car il faut rajouter l'attribut partout en suivant l'héritage. Le résultat de cela est que certains attributs ajoutés en début de projet ne sont finalement pas utilisés car une autre manière de remplir leurs fonctions a été trouvé au cours du projet. Il y a par exemple l'attribut "place" dans les objet plaçable qui est remplacé par le fait de regarder si le game object correspondant est activé.

Bien que beaucoup de cas est été prévu à la base, il a tout de même fallut rajouter certains attributs tel que l'attribut "ielt", qui permet d'accéder au model 3D de l'item.

Répartition sur les différentes soutenances :

Cette tache a été réalisée pour la soutenance 1 et la soutenance 2. Avant la soutenance 1 a été réalisé la structure du système de class avec les différentes class principales ainsi qu'une ou deux class d'item en eux même dans le but de tester le système d'implémentation. Il a donc suffit de compléter le travail commencé pour la soutenance 1 en implémentant tous les items en eux même pour la deuxième soutenance.

3.2.2 Inventaire, Système de Craft Gestion des Slot

Dans cette partie je vais vous expliquer les différentes étapes de la création de l'inventaire, du système de craft ainsi que de la gestion des slots. Pour cela je vais commencé par vous expliquer comment l'inventaire est implémenté pour vous expliquer l'évolution qu'il a subit au cours du projet.

Implémentation :

Petite remarque pour éviter les confusions : il y a deux choses que j'appelle inventaire : l'ensemble général qui contient le menu de craft, les slots et la partie centrale, et il y a aussi la partie centrale que j'appelle inventaire. Pour éviter cette confusion j'appellerai donc le premier menu de l'inventaire et le second inventaire tout court.

Avant de commencer à vous expliquer comment il a été implémenté, je vais commencé par vous expliquer comment fonctionne le menu de l'inventaire pour que vous en ayez une meilleur idée.

Le menu de l'inventaire est composé de 3 parties principales : la partie inventaire, les slots, le menu de craft.

La partie inventaire est la partie principale du menu de l'inventaire, c'est à cet endroit que tous les items sont et à travers lequel on peut interagir avec les items, les sélectionner ou les déplacer. Il nous donne aussi des informations : le nombre d'item que l'on possède, si l'on possède l'item (en vert), si l'on peut le crafter (en orange), ou pas (en rouge).

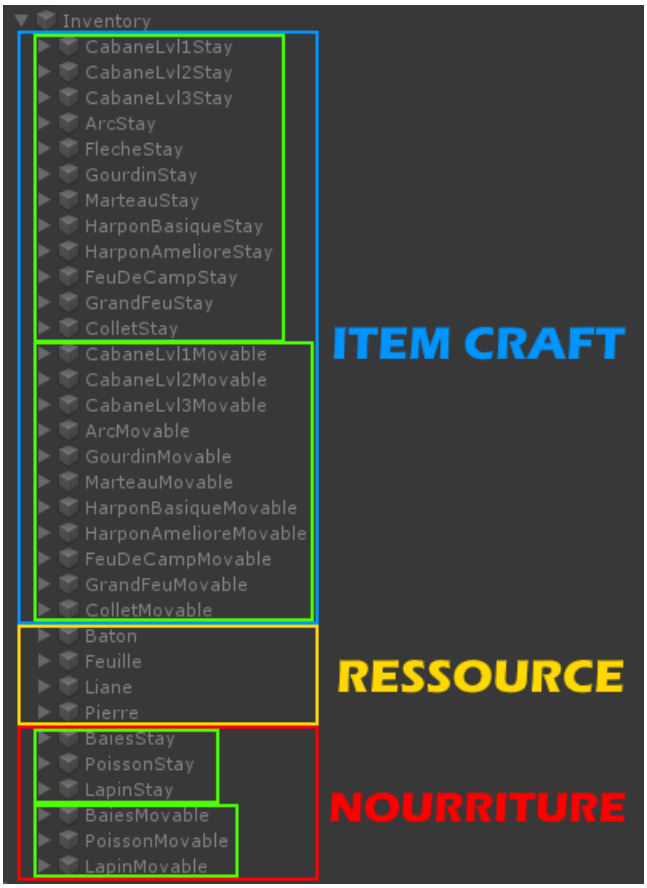
Le menu de craft permet de crafter l'item sélectionné dans l'inventaire à l'aide d'un bouton, il nous donne aussi des indication visuelle quant à la quantité de ressources nécessaire pour construire l'item, ou en nous affichant l'item sélectionné.

Les slots permettent de rendre accessible en jeu, les items qui y sont glissé depuis l'inventaire.

Maintenant passons à l'explication de l'implémentation, pour cela commençons par regarder la hiérarchie du menu de l'inventaire dans unity.

Nous retrouvons bien ici les 3 éléments du menu de l'inventaire.

Voyons plus en détail la hiérarchie de la partie inventaire du menu.



Dans cette hiérarchie on peut déjà voir plusieurs truc intéressant, on voit déjà les 3 types principaux d'items qui ne se mélange pas, bien que les mélanger ne changerait rien. Il est quand même plus clair de les séparer étant donner qu'ils sont implémentés de manières différentes. Une deuxième chose que l'on peut remarquer, c'est que pour les items nourritures et craftables, ils sont présent en double et pas les ressources, la aussi cela vient de la manière dont ils sont implémentés.
Pour comprendre cela il faut connaître les 4 chose que l'on peut faire dans l'inventaire : survoler l'item (but seulement esthétique), sélectionner l'item, afficher des informations et mettre l'item dans les slots (déplacer l'item). Or ces 4 actions ne sont pas nécessaires pour tous les 3 types d'item. Les ressource ne servent que comme composant du craft elle n'ont donc pas besoins d'être sélectionné, car la sélection permet de crafter l'item, or ces items ne sont pas construit par le joueur mais ramassés sur la map. Elles n'ont pas non plus besoin d'être déplacés car elles ne sont pas utilisables par le joueur directement. L'inventaire ne doit donc que signaler le survole et afficher les

informations.
Pour la nourriture en plus d'afficher les informations et de signaler le survole il faut aussi pouvoir les déplacer, car le joueur peut les utiliser dans le jeu.
Enfin concernant les items craftable, il nécessite les 4 actions, en effet le joueur peut les utiliser dans le jeu et en plus de cela il faut qu'il soit sélectionnables pour ouvrir le menu de craft dans le but de les construire.

Détaillons maintenant le script des 4 actions possibles :

1) survole :

```
1  private bool over;
2
3  public void Update()
4  {
5     if (!GameObject.Find("Inventory").GetComponent<Inventory>().select)
6     {
7        if (over && !drag)
8        {
9           GetComponent<Image>().color = new Color(0, 1, 0, 1);
10       }
11       else
12       {
13          GetComponent<Image>().color = new Color(1, 1, 1, 1);
14       }
15    }
16    txt.text = objet.own + "/" + objet.max_own;
17 }
18
19 public void OnPointerEnter(PointerEventData eventData)
20 {
21    over = true;
22 }
23
24 public void OnPointerExit(PointerEventData eventData)
25 {
26    over = false;
27 }
```

On définis une variable over qui passe à true quand la souris arrive sur l'image de l'item (le gameobject) grâce à la fonction OnPointerEnter, et qui passe à false quand la souris en sort grâce à la fonction OnPointerExit. Quand over est à true on met la couleur des contours en jaune, sinon on la met en blanc (on ne fait pas cela si l'item est sélectionné).

2) information :

```
1  public int indice;
2  public Item_Craft objet;
3
4  public void Start()
5  {
6     objet = InventoryObject.Item_Craft[indice];
7     ColorUpdate();
8     txt = GetComponentInChildren<Text>();
9  }
10 public void Update()
11 {
12    txt.text = objet.own + "/" + objet.max_own;
13 }
14
15 public void ColorUpdate()
16 {
17    objet = InventoryObject.Item_Craft[indice];
18    if (objet.own > 0)
19    {
20       GetComponentsInChildren<Image>()[1].color = new Color(0.43f,0.74f,0.31f,1);
21    }
22    else if (objet.Craftable(InventoryObject.ressource()))
23    {
24       GetComponentsInChildren<Image>()[1].color = new Color(1,0.50f,0.16f,1);
25    }
26    else
27    {
28       GetComponentsInChildren<Image>()[1].color = new Color(1,0.16f,0.24f,1);
29    }
30 }
```

Pour afficher les informations il faut déjà savoir de quel item on parle, pour cela on récupère l'item à l'aide d'un indice fournis dans l'éditeur unity. Grâce à cet indice on récupère l'item correspondant dans une liste définis dans la class InventoryObject (définis plus tôt dans la section item). On met alors à jour dans l'affichage le nombre d'occurrence de l'item que l'on possède, dans la fonction update.
De plus on appelle la fonction ColorUpdate au début du jeu et après chaque craft pour actualiser les couleurs nous indiquant si l'on peut crafter un item ou pas. Cette fonction regarde si l'on possède l'item ou si l'on a de quoi le construire, et définis la couleur appropriée en fonction.

3) mouvement :

Comme on l'a vu plus tôt dans la hiérarchie, il y a pour les item que l'on peut déplacer deux éléments. Cela vient du fait qu'il y a un élément qui est la pour être déplacé et un qui ne bouge pas.

```
1  public void OnBeginDrag(PointerEventData eventData)
2  {
3     if (InPlaceUp(Place1))
4     {
5        Slot.Slot1 = null;
6     }
7     else if (InPlaceUp(Place2))
8     {
9        Slot.Slot2 = null;
10    }
11    else if (InPlaceUp(Place3))
12    {
13       Slot.Slot3 = null;
14    }
15    else if (InPlaceUp(Place4))
16    {
17       Slot.Slot4 = null;
18    }
19 }
20
21 public void OnDrag(PointerEventData eventData)
22 {
23    if (objet.own > 0)
24    {
25       if (InPlace(Place1))
26       {
27          transform.position = Place1.transform.position;
28          drag = false;
29       }
30       else if(InPlace(Place2))
31       {
32          transform.position = Place2.transform.position;
33          drag = false;
34       }
35       else if(InPlace(Place3))
36       {
37          transform.position = Place3.transform.position;
38          drag = false;
39       }
40       else if(InPlace(Place4))
41       {
42       transform.position = Place4.transform.position;
43       drag = false;
44       }
```

```
45          else if (Math.Abs(Input.mousePosition.x - initialposition.x) < 25 &&
46                Math.Abs(Input.mousePosition.y - initialposition.y) < 25)
47          {
48             transform.position = initialposition;
49             drag = false;
50          }
51          else
52          {
53             transform.position = Input.mousePosition;
54             drag = true;
55             transform.SetSiblingIndex(100);
56          }
57       }
58       else
59       {
60          transform.position = initialposition;
61       }
62    }
63
64    public void OnEndDrag(PointerEventData eventData)
65    {
66       if (objet.own > 0)
67       {
68          drag = false;
69          transform.SetSiblingIndex(siblingindex);
70          if (InPlaceUp(Place1))
71          {
72             DefSlot(ref Slot.Slot1);
73          }
74          else if (InPlaceUp(Place2))
75          {
76             DefSlot(ref Slot.Slot2);
77          }
78          else if (InPlaceUp(Place3))
79          {
80             DefSlot(ref Slot.Slot3);
81          }
82          else if (InPlaceUp(Place4))
83          {
84             DefSlot(ref Slot.Slot4);
85          }
86          else
87          {
88             transform.position = initialposition;
89          }
90       }
91    }
92
93    private bool InPlace(Transform Place)
94    {
```

```
95      return Math.Abs(Input.mousePosition.x - Place.transform.position.x) < 25 &&
96         Math.Abs(Input.mousePosition.y - Place.transform.position.y) < 25;
97  }
98
99  private bool InPlaceUp(Transform Place)
100 {
101     return Mathf.Abs(transform.position.x - Place.position.x) <= 0.5f &&
102          Mathf.Abs(transform.position.y - Place.position.y) <= 0.5f;
103 }
104
105 private void DefSlot(ref GameObject Slot)
106 {
107     if (Slot != null && Slot != this.gameObject)
108     {
109         try
110         {
111             Slot.transform.position = Slot.GetComponent<ItemsCraftMovable>().
112             initialposition;
113         }
114         catch (Exception e)
115         {
116             Slot.transform.position = Slot.GetComponent<NourritureMovable>().
117             initialposition;
118         }
119     }
120     Slot = this.gameObject;
121 }
```

Comme vous pouvez le voire le script du mouvement est plutôt long. Je vais donc expliquer le script chronologiquement à l'exécution de celui-ci.

Le script commence par l'exécution de la fonction OnBeginDrag, au moment ou l'on clique sur l'item pour le déplacer. Le but de cette fonction est de dire à l'éventuel slot qui contiendrait l'item qu'il ne le contient plus car il est maintenant en mouvement.

La fonction exécute ensuite est la fonction OnDrag. Elle est exécutée pendant tout le mouvement de l'item. L'objectif principal de cette fonction est de faire suivre la souris par l'item. Elle permet aussi de mettre l'item au premier plan. Cependant il y a plusieurs cas ou l'item ne doit plus suivre la souris, s'il est proche de sa position de départ, ou s'il est proche de la position d'un slot (ceci est testé à l'aide de la fonction InPlace, où le paramètre PlaceX représente la position d'un slot). De plus l'item ne doit pas pouvoir bouger si l'on ne le possède pas.

L'exécution du script se termine par la fonction OnEndDrag, cette fonction se charge de dire au slot quel item il contienne si l'item est présent à la fin du déplacement.

4) sélection :

```
1  public void OnPointerUp(PointerEventData eventData)
2  {
3     GameObject.Find("Inventory").GetComponent<Inventory>().Gselect = gameObject;
4     GameObject.Find("Inventory").GetComponent<Inventory>().select = true;
5     GameObject.Find("Inventory").GetComponent<Inventory>().craftitem = objet;
6  }
7
8  void Update()
9  {
10    if (select && Input.GetMouseButtonUp(0))
11    {
12       if (Gselect == old_Gselect && !craft_enable)
13       {
14          select = false;
15          old_Gselect = null;
16       }
17       else if (old_Gselect != null)
18       {
19          old_Gselect.GetComponent<Image>().color = new Color(1,1,1,1);
20       }
21       craft_enable = false;
22       craft.GetComponent<DisplayCraft>().enabled = false;
23       craft.SetActive(false);
24    }
25    else if (select)
26    {
27       old_Gselect = Gselect;
28       Gselect.GetComponent<Image>().color = new Color(0,0,1,1);
29    }
30
31    if (select)
32    {
33       craft.SetActive(true);
34       craft.GetComponent<DisplayCraft>().enabled = true;
35    }
36    else
37    {
38       craft.GetComponent<DisplayCraft>().enabled = false;
39       craft.SetActive(false);
40    }
41 }
```

La sélection est réparti dans deux script, un premier sur l'item qui passe les informations de l'item nécessaire au deuxième. Ces informations sont : qui est l'objet représentant l'item (gameobject), indique qu'un objet est sélectionné et renseigne l'item dont il s'agit (quels item de la class InventoryObject).

Le deuxième script se sert de ces informations pour indiquer visuelle-

ment quel item est sélectionné en changeant la couleur du contour du gameobject récupéré précédemment. Il ouvre enfin le menu de craft en activant le gameobject correspondant (vous avez pu voir plutôt dans la hiérarchie que le menu de craft était de base désactivé), et en lui indiquent l'item dont il doit s'occuper.

Vous avez sans doute remarqué qu'il n'y a pas que le menu de craft qui était désactivé dans la hiérarchie, mais aussi l'inventaire. Celui-ci est activé par le joueur dans le jeu quand il ouvre l'inventaire, une mécanique implémenté par mon camarade Pierre.

Passons maintenant au menu de craft, ce système est nettement plus simple que le précédant. Après avoir sélectionné un item le menu de craft s'ouvre en affichant des informations sur l'item. Nous avons alors la possibilité de construire l'item en appuyant sur le bouton "CRAFT". Ce qui appel un script procédant à cette construction.

Regardons maintenant comment fonctionne le système de craft en détail. La première chose essentiel à faire pour le craft c'est de connaître l'item qui doit être crafté pour cela il faut récupérer l'item qui à été donné lors de la sélection. Cette information est récupéré lors de l'activation du système de craft, grâce à la fonction OnEnable. Grâce à cela on sait quel item doit être crafté, on peuxt donc afficher les informations visuelles.

```
1 private void OnEnable()
2 {
3     item = GameObject.Find("Inventory").GetComponent<Inventory>().craftitem;
4     icone.GetComponent<Image>().sprite = item.icone;
5     text.text = item.name + "\n" + "\n" + "Vous␣possdez␣:␣" + item.own +"␣/␣" +
6         item.max_own + "\n" + "\n" + "Ressource␣:" + "\n" + "baton␣:␣" +
```

```
7           InventoryObject.ressource()[0] + "␣/␣" + item.Craft_Cost[0] + "\n" +
8           "feuille␣:␣" + InventoryObject.ressource()[1] + "␣/␣" + item.Craft_Cost[1]
9              + "\n" + "pierre␣:␣" + InventoryObject.ressource()[2] + "␣/␣" +
10                item.Craft_Cost[2] + "\n" + "liane␣:␣" + InventoryObject.ressource()[3]
11                   + "␣/␣" + item.Craft_Cost[3];
12  }
```

Passons maintenant à la partie du craft en lui même, c'est-à-dire ce qui se passe après l'appui sur le bouton du craft.

```
1   public void OnPointerDown(PointerEventData eventData)
2   {
3      GameObject.Find("Inventory").GetComponent<Inventory>().craft_enable = true;
4      item = GameObject.Find("Craft").GetComponent<DisplayCraft>().item;
5      craft(InventoryObject.ressource(), item);
6      ItemsCraftMovable[] invetoriesImv =
7         GameObject.Find("Inventory").GetComponentsInChildren<ItemsCraftMovable>();
8      ItemsCraftStay[] invetoriesIst =
9         GameObject.Find("Inventory").GetComponentsInChildren<ItemsCraftStay>();
10     foreach (var elt in invetoriesImv)
11     {
12        elt.ColorUpdate();
13     }
14     foreach (var elt in invetoriesIst)
15     {
16        elt.ColorUpdate();
17     }
18     }
```

Après avoir récupéré à son tour l'item à craft "item = GameObject.Find("Craft").GetComponent<DisplayCraft>().item ;" la fonction gérant le craft est lancé.

```
1   public void craft(int[] ressource, Item_Craft item)
2   {
3      if (Craftable(ressource, item.Craft_Cost))
4      {
5         item.own += 1;
6         RemoveComponant(ressource, item.Craft_Cost);
7         GameObject.Find("FPSController").GetComponent<Player>()
8            .player1.updatetime(item.CraftTime);
9         GameObject.Find("UI").GetComponent<sound>().Sound1();
10     }
11     else
12     {
13        GameObject.Find("UI").GetComponent<sound>().Sound2();
14     }
15  }
```

Cette fonction commence par vérifier que le craft est possible.

```
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      if (item.own >= item.max_own)
10     {
11         valide = false;
12     }
13     return valide;
14 }
```

Pour cela elle vérifie d'abord que le joueur à les ressource pour construire cet item et qu'il possède la place pour le stocker.
Dans le cas où les conditions sont validé un item est donné au joueur "item.own += 1", puis les composant nécessaire à sa construction sont retiré de sont inventaire "RemoveComponant(ressource, item.Craft_Cost)".

```
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  }
```

Le temps est ensuite actualisé pour prendre en compte le temps de construction, et enfin le son de réussite du craft est joué. Si les condition ne sont pas valide seul le son d'échec du craft est joué.

La dernière étape est d'actualiser les couleur des item car il y a eu un changement au niveau des ressources.

Pour fermer le menu de craft il suffit ensuite de cliquer à côté du menu.
La dernière partie du menu de l'inventaire est la partie slot, celle ci est toute simple quand un item est déposé dans un des slot, une variable déclaré de manière static. slot prend la valeur de cet item. Celle-ci est alors accessible depuis partout dans le projet pour connaître les items présent dans les slots.

```
1  public static GameObject Slot1;
2  public static GameObject Slot2;
3  public static GameObject Slot3;
4  public static GameObject Slot4;
```

Répartition sur les différente soutenance :

soutenance 1 : Réalisation des éléments principaux du menu de l'inventaire, le déplacement des items, la sélection des items, le menu de craft.

soutenance 2 : Adaptation de l'inventaire au canvas (précédemment réalisé avec des sprite), ajout d'indication visuelle dans l'inventaire et le menu de craft, fin de l'implémetation des slots, ajouts de tous les items ainsi que des sprites les symbolisant, mise en place dans le jeu et résolution des bugs.

soutenance 3 : Ajout d'indications visuelles et résolution de bugs.

3.2.3 Sélecteur d'item

Le sélecteur d'item permet au joueur de choisir l'item qu'il veut utiliser parmi ceux présent dans les slots, il a été implémenté avec Pierre.

Le joueur peut choisir entre les quatre item dans les slots, la gourde et le briquet. Pour changer d'item il fait soit tourner la molette vers le bas pour avoir l'item précédant ou vers le haut pour avoir l'item suivant. Quand la molette tourne elle incrémente un indice qui varie entre 0 et 5, les valeurs 4 et 5 correspondent respectivement au briquet et à la gourde. Pour obtenir l'item des quatre valeur précédant, on récupère la valeur d'un tableau à l'indice actuel. Ce tableau Contient les item placé dans les slots. Il nous suffit alors d'activer les différente chose allant avec l'item correspondant. Par exemple le gameobject pour le modèle 3D qui est le fils à l'indice "ielt" (attribut des items évoqué plus tôt) de la hiérarchie, ou alors pour les armes au corps à corps par exemple, l'animation des coups. Il ne faut pas oublier aussi de désactiver ceux de l'item précédant.

```
1  void Update()
2  {
3     previous = selectweapon1;
4     if (Input.GetAxis("Mouse ScrollWheel")>0f)
5     {
6        if (selectweapon1 <= 0)
7        {
8           selectweapon1 = 5;
9        }
10       else
```

```
11          {
12              selectweapon1--;
13          }
14          GameObject.Find("Slot").GetComponentsInChildren<Image>()[3*previous].color
15          = new Color(1,1,1,1);
16          GameObject.Find("Slot").GetComponentsInChildren<Image>()[3*selectweapon1].color
17          = new Color(0,0,0,1);
18      }
19      if (Input.GetAxis("Mouse ScrollWheel") < 0f)
20      {
21          selectweapon1++;
22          selectweapon1 %= 6;
23          GameObject.Find("Slot").GetComponentsInChildren<Image>()[3*previous].color
24          = new Color(1,1,1,1);
25          GameObject.Find("Slot").GetComponentsInChildren<Image>()[3*selectweapon1].color
26          = new Color(0,0,0,1);
27      }
28      if (selectweapon1 != previous)
29      {
30          deselect();
31          select();
32      }
33  }
34
35  public void select()
36  {
37      GameObject.Find("FPSController").GetComponent<Player>().player1.SetDamage(0);
38      if (selectweapon1 == 4)
39      {
40          transform.GetChild(20).gameObject.SetActive(true);
41      }
42      else if (selectweapon1 == 5)
43      {
44          transform.GetChild(21).gameObject.SetActive(true);
45          gourde.SetActive(true);
46          GetComponent<Animator>().enabled = true;
47      }
48      else
49      {
50          Items item = Slot.items[selectweapon1];
51          if (item is Armes)
52          {
53              if (item is ArmesCorpaCorp)
54              {
55                  GetComponent<Animator>().enabled = true;
56                  GetComponent<AnimationArme>().enabled = true;
57                  transform.GetChild(item.ielt).gameObject.SetActive(true);
58              }
59              else if (item is Harpons)
60              {
```

```
61              GetComponent<Animator>().enabled = true;
62              transform.GetChild(item.ielt).gameObject.SetActive(true);
63          }
64          else if (item is Arc)
65          {
66              transform.GetChild(item.ielt).gameObject.SetActive(true);
67          }
68          Armes it = (Armes) item;
69          GameObject.Find("FPSController").GetComponent<Player>().player1.SetDamage(it.dam
70      }
71      else if (item is Item_placable)
72      {
73          if (item is Cabane)
74          {
75              transform.GetChild(19).gameObject.SetActive(false);
76              transform.GetChild(item.ielt).gameObject.SetActive(true);
77          }
78          else
79          {
80              transform.GetChild(item.ielt).gameObject.SetActive(true);
81          }
82      }
83      else if (item is Nourriture)
84      {
85          transform.GetChild(item.ielt).gameObject.SetActive(true);
86      }
87  }
88  previous = selectweapon1;
89 }
```

Répartition sur les différentes soutenances :

soutenance 2 : Création du système et ajout des premiers items.
soutenance 3 : Ajout de tous les items.

3.2.4 IA

Dans la création des IA je me suis occupé de deux choses, premièrement j'ai retravaillé le zombie de base implémenté par Pierre pour la première soutenance en lui donnant des paternes de combat et de déplacements, deuxièmement, avec Pierre nous avons créé les système de spawn et de déplacement général des zombies.

Zombie Basique :

Je me suis occupé d'améliorer le zombie basique implémenté par Pierre pour la première soutenance. Je lui ai créé des paternes, dans le but de le rendre plus intéressant.
Quand le joueur est à distance du zombie, le zombie marche en direction d'un point (script de déplacement expliqué par la suite), puis quand le joueur entre dans une sphère autour du zombie le script des paternes de combat est activé.

```
1  private void OnTriggerEnter(Collider other)
2  {
3     if (other.tag == "player")
4     {
5        patern.enabled = true;
6        deplacement.enabled = false;
7     }
8  }
9
10 private void OnTriggerExit(Collider other)
11 {
12    if (other.tag == "player")
13    {
14       patern.enabled = false;
15       delplacement.enabled = true;
16    }
17 }
```

Script réalisé par Pierre pour la première soutenance

La partie paterne fonctionne de la manière suivante, le zombie regarde si le joueur est dans son champ de vision, si il n'y est pas (caché par quelque chose : arbre, colline, ...), le zombie se met en position d'attente et active l'animation "idle", sinon le script de combat est activé.

```
1  private void Update()
2  {
3     if (inFOV(transform, player, new Vector3(player.position.x, player.position.y + 1,
4     player.position.z), new Vector3(transform.position.x, transform.position.y + 1.5f, tra
5     {
6        attaque.enabled = true;
7     }
8     else
9     {
10       ennemi.SetBool("idle",true);
```

```
11          attaque.enabled = false;
12      }
13  }
```

Le script champs de vision test si le joueur est visible par le zombie, pour cela le zombie récupère tous les gameobject à proximité de lui, ensuite il envoi un raycast vers ces items et s'il touche le player c'est qu'il n'y a rien entre les deux et qu'il le voit.

```
1   public static bool inFOV(Transform checkingObject, Transform target,
2      Vector3 playerTete, Vector3 IATete, float maxAngle, float maxRadius)
3   {
4      Collider[] overlaps = new Collider[30];
5      int count = Physics.OverlapSphereNonAlloc(checkingObject.position,
6         maxRadius, overlaps);
7      for (int i = 0; i < count; i++)
8      {
9         if (overlaps[i] != null)
10        {
11           Vector3 directionBetween =
12               (target.position - checkingObject.position).normalized;
13          directionBetween.y *= 0;
14          float angle = Vector3.Angle(checkingObject.forward, directionBetween);
15          Ray ray1 = new Ray(IATete, target.position - IATete);
16          Ray ray2 = new Ray(IATete, playerTete - IATete);
17          RaycastHit hit;
18          if (Physics.Raycast(ray1, out hit, maxRadius))
19          {
20            if (hit.transform == target)
21            {
22              return true;
23            }
24          }
25          if (Physics.Raycast(ray2, out hit, maxRadius))
26          {
27            if (hit.transform == target)
28            {
29              return true;
30            }
31           }
32        }
33      }
34     return false;
35    }
```

Détaillons maintenant le script d'attaque. Premièrement le zombie se dirige vers le joueur. Le zombie s'arrête quand il est en face du joueur et à porté pour le taper (représenté par un box collider). A ceux moment il lance en alterné ses deux coups basique, si le coup touche (si le joueur est toujours présent dans le box collider au moment de l'impact), le joueur prend des dégâts, le zombie enchaîne alors avec un gros coup. Dans le cas où le premier coups est manqué ou après le gros coup (qu'il touche le joueur ou pas), le zombie attend avec l'animation "idle" puis le script recommence.

Spawn et déplacement des zombies :

Cette partie à été réalisé avec Pierre.

Je vais maintenant vous expliquer comment fonctionne le spawn et le déplacement général des zombies. Les zombies spawn au début de la nuit, il leur est alors donné une destination aléatoire, une fois cette destination atteinte ils se dirigent tous vers la plage du spawn du joueur. Quand il ne reste plus que 3h (du jeu) de nuit ils repartent vers leur spawn.
De plus le nombre de zombie augmente au fur et à mesure des nuits.

```
1  private void Update()
2  {
3     attaque.enabled = false;
4     int min = GameObject.Find("Hud").GetComponent<CompteaR>().min;
5     if (min % 1440 < 180 || min % 1440 > 480)
6     {
7        agent.SetDestination(DestSpawn.position);
8     }
9     else if (firstGet)
10    {
11       agent.SetDestination(DestPlage.position);
12    }
13    else if (Vector3.Distance(agent.transform.position, DestRand.position) < 3f)
14    {
15       firstGet = true;
16    }
17    else
18    {
19       agent.SetDestination(DestRand.position);
20       firstGet = false;
21    }
22 }
```

Répartition sur les différentes soutenances :

soutenance 2 : apprentissage de l'IA dans unity

soutenance 3 : retravaille du zombie basique et création du script de spawn.

	Attaque 1	Attaque 2	Attaque 3
Monstre1	Attaque d'arrivée sur le joueur : Le monstre alterne entre un coup de poing droit et un gauche, qui inflige 5 points de dégâts	Lorsque la première attaque touche le joueur : Le zombie envoi un gros coup balayant tous l'espace devant lui, cette attaque inflige 10 points de dégâts	-

Les différentes caractéristiques liées à la vie et à la distance à laquelle le zombie peut voire le joueur ont changé, voici donc un tableau regroupant les nouvelles valeurs :

	Point de vie	Distance à laquelle le monstre détecte le joueur
Monstre1	75 points de vie	15 mètres

Remarque :

Le zombie basique est un peu lent il a donc du mal à allé vers la plage ce qui limite les rencontres avec le joueur.

Plus le nombre de zombies augmente plus ils sont lent à réagir. Ce qui fait que nos zombies mettent parfois plusieurs seconde à réagir quand ils sont nombreux.

3.2.5 Sound Design

Pour ma part je me suis seulement occupé du sound design du menu de craft, en effet c'est Jean qui à fait la plus grosse partie du travail dans ce domaine. J'ai pour cela créé un script à côté de l'inventaire dans lequel il y a une fonction qui déclenche le son pour un craft réussis et une pour un craft qui ne marche pas. J'ai utilisé un script externe, car le fait de déclencher le son depuis le script de craft le faisait se couper avant sa fin.

```
1  private AudioSource craftNoise;
2  public AudioClip craftWork;
3  public AudioClip craftNotWork;
4
5  void Start()
6  {
7     craftNoise = GetComponent<AudioSource>();
8  }
9
10 public void Sound1()
11 {
12    craftNoise.PlayOneShot(craftWork);
13 }
14
15 public void Sound2()
16 {
17    craftNoise.PlayOneShot(craftNotWork);
18 }
```

Répartition sur les différentes soutenances :

Cette tâche à été réalisé pour la deuxième soutenance et amélioré pour la troisième.

3.2.6 Système de ramassage

Pour ma part je me suis occupé du ramassage des baies. Ce système marche de la manière suivante, lorsque le joueur entre dans le collider autour des baies le jeu lui propose d'appuyer sur E pour ramasser les baies. Au moment où le joueur appui sur E, une baies est ajoutée à l'inventaire du joueur.

```
1  private void OnTriggerStay(Collider other)
2  {
3     if (other.tag == "player")
4     {
5        if (baies.activeSelf)
6        {
7           pressE.SetActive(true);
8           if (Input.GetKeyDown(KeyCode.E))
```

```
9           {
10              if (InventoryObject.baies.own < InventoryObject.baies.max_own)
11              {
12                  GameObject.Find("FPSController").GetComponent<Player>().easy
13                      .PlayOneShot(GameObject.Find("FPSController")
14                          .GetComponent<Player>().Ramasser);
15                  InventoryObject.baies.own++;
16                  baies.SetActive(false);
17              }
18          }
19      }
20      else
21      {
22          pressE.SetActive(false);
23      }
24  }
25  }
26
27  private void OnTriggerExit(Collider other)
28  {
29      pressE.SetActive(false);
30  }
```

Répartition sur les différentes soutenances :

Cette tâche à été réalisé pour la troisième soutenance.

3.2.7 Modèle 3D

Comme tous les membres du groupe j'ai participé à la recherche d'objet 3D lorsque l'on en avait besoin. Je peux par exemple citer les pierres autour du feu.

3.2.8 Implémentation des items (gameplay)

Je me suis occupé de l'implémentation de nombreux items dans le jeu, en plus de la partie class. Je peux citer la pose du feu de camp, la cuisson sur le feu de camp, la pose du grand feu, les baies, les buissons, les cabanes, les collets.
Beaucoup de ces items on des script assez ressemblant pour interagir avec. Premièrement, une méthode OnTrigger pour détecter quand le joueur approche de l'item, sans suit alors plusieurs test, comme savoir si le joueur à un item dans la main (pour les items plaçable, ou la cuisson par exemple), si le joueur possède un item, ou au contraire si il n'en à pas trop dans le cas ou l'on récupère un item. Si les conditions sont vérifié alors on affiche un message comme quoi le joueur peut faire une action en appuyant sur une touche. Si le joueur appui sur cette touche l'item exécute l'action demandé.

Voici le script de ramassage du lapin dans le collet pour illustré cela.

```
1  private void OnTriggerStay(Collider other)
2  {
3     if (Bunny.active)
4     {
5        int indice = GameObject.Find("hand_right_prefab")
6           .GetComponent<Selectedweapon>().selectweapon1;
7        if (indice < 4)
8        {
9           Items itemHand =
10          Slot.items[GameObject.Find("hand_right_prefab")
11             .GetComponent<Selectedweapon>().selectweapon1];
12          if (InventoryObject.lapin.own <InventoryObject.lapin.max_own)
13          {
14             pressE.SetActive(true);
15             if (Input.GetKeyDown(KeyCode.E))
16             {
17                InventoryObject.lapin.own++;
18                Bunny.SetActive(false);
19                pressE.SetActive(false);
20             }
21          }
22          else
23          {
24             pressE.SetActive(false);
25          }
26       }
27    }
28    else
29    {
30       pressE.SetActive(false);
31    }
32 }
33
34 private void OnTriggerExit(Collider other)
35 {
36    pressE.SetActive(false);
37 }
```

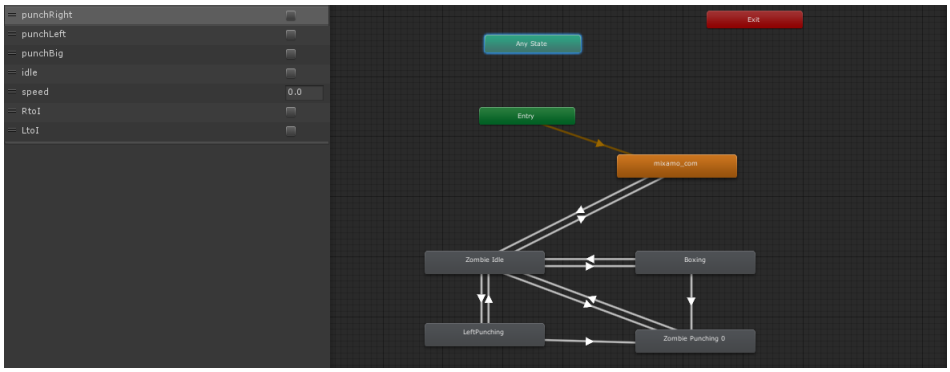
Répartition sur les différentes soutenances :

Cette tâche à été réalisé en continue durant les 3 périodes.

3.2.9 Animation

Pour ma part je me suis occupé de l'animation du zombie basique, pour cela j'ai géré 5 animations : la marche (mise par Pierre), le coup de poing droit (mis par Pierre), le coup de poing gauche, le grand coup et l'animation d'attente ("idle").

Je me charge donc dans le script d'attaque du zombie de bien enchaîner les animation pour obtenir quelque chose de cohérent. Pour cela j'ai du par exemple ralentir les animation de coup de poing pour rendre l'attaque esquivable par le joueur.



Répartition sur les différentes soutenances :

Cette tâche à été réalisé pour la troisième soutenance.

3.2.10 Interface

Du côté de l'interface je me suis occupé de tout ce qui concerne le menu de l'inventaire, pour cela j'ai entre autre désigné tous les logo des items.

J'ai aussi réalisé un système pour indiquer le moment de la journée au joueur, il s'agit d'un soleil et d'une lune (en fonction du jour ou de la nuit), qui bouge de gauche à droite pour indiquer l'heure.

```
1  private void Start()
2  {
3     ecart = start.position.x - end.position.x;
4  }
5
6  private void Update()
7  {
8     min = GameObject.Find("Hud").GetComponent<CompteaR>().min%1440;
9     if (min < 480f)
10    {
11       nuit.SetActive(true);
12       lune.SetActive(true);
13       soleil.SetActive(false);
14       ciel.SetActive(false);
15       lune.transform.position = new Vector3(end.transform.position.x +
16          ecart*(min/480f),lune.transform.position.y,lune.transform.position.z);
17    }
18    else
19    {
20       nuit.SetActive(false);
21       lune.SetActive(false);
22       soleil.SetActive(true);
23       ciel.SetActive(true);
24       soleil.transform.position = new Vector3(end.transform.position.x +
25          ecart*((min - 480f)/960f),soleil.transform.position.y,
26             soleil.transform.position.z);
27    }
28 }
```

Répartition sur les différentes soutenances :

Les logos des items ont été réalisé pour la deuxième soutenance, et le système pour indiquer le moment de la journée au joueur à été réalisé pour la troisième.

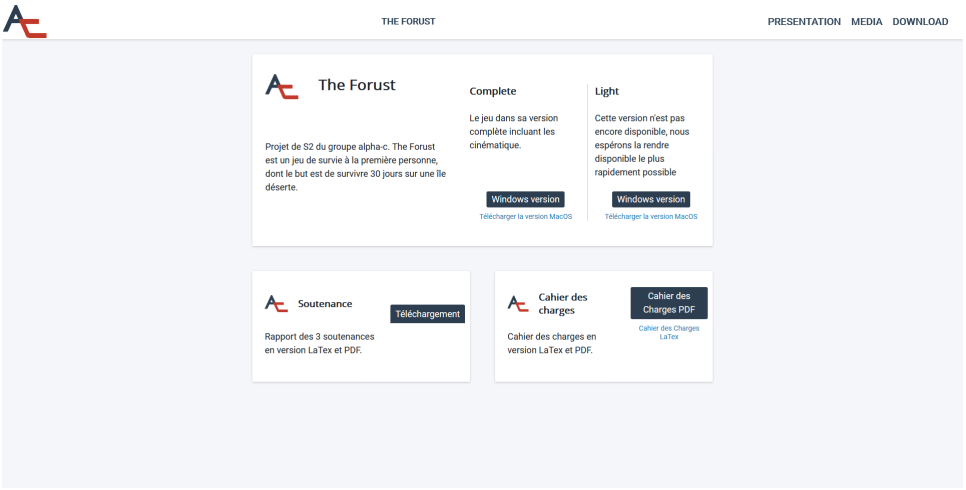
3.2.11 Site Web

Dans la création du site web je me suis occupé de 3 choses. La mise en place du serveur, pour cela j'ai installé apache2, php, et ai configuré le ftp pour que mes camarade puissent se connecter aussi sur le serveur. J'ai ensuite redirigé le nom de domaine vers le serveur et ai configurer apache

pour affiché notre page web. La deuxième chose que j'ai faîtes à été de créer l'aspect général du site j'ai pour cela créé la feuille de style principal, ainsi que la bar de navigation du site. Elle est par ailleurs équipé d'un petit script javascript qui la fait réduire de taille quand on défile sur la page.

```
1  window.onscroll = function () {
2        if (document.documentElement.scrollTop > 60) {
3              document.getElementById("tete").style.position = "fixed";
4              document.getElementById("tete").style.height = "50px";
5        }
6        else{
7              document.getElementById("tete").style.position = "fixed";
8              document.getElementById("tete").style.height = "70px";
9        }
10 }
```

La dernière chose dont je me suis occupé à été la page de téléchargement.



Répartition sur les différentes soutenances :

soutenance 1 : configuration du serveur, mise en ligne du site web, création de la feuille de style, et du début de la barre de navigation.

soutenance 2 : fin de la création de la barre de navigation et début de la page de téléchargement.

soutenance 3 : fin de la page de téléchargement.

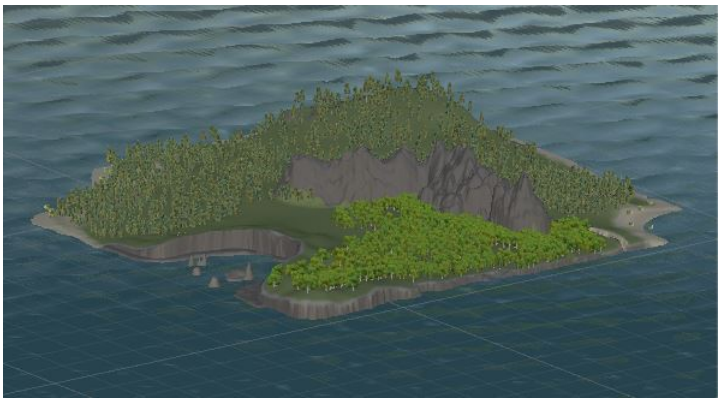
3.3 Jean

3.3.1 Aspect graphique

1ère soutenance :

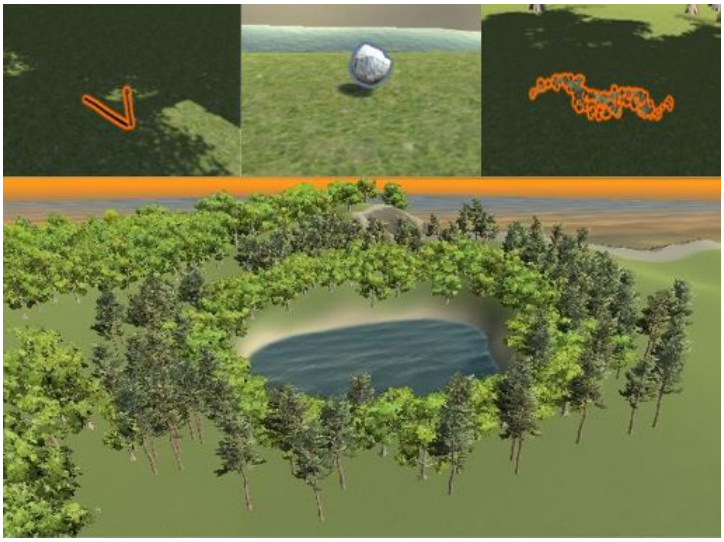
En ce qui concerne la création et le design de la carte, pour la première soutenance, j'ai pu modéliser une bonne partie de l'île ou le jeu allait se dérouler. J'avais commencé par créer le terrain de l'île, puis gérer les formes de la carte comme les montagnes au centre, les falaises ou encore les courbures du sol et tout cela grâce aux outils de création de terrain d'Unity.

J'ai ensuite appliqué les textures correspondantes pour donner un aspect plus réaliste aux montagnes, plages, forêts, plaines et également pour les délimiter. Enfin, pour terminer avant la première soutenance, j'ai ajouté la végétation en créant les forêts et ajouter de l'eau autour de l'île et de ce fait obtenir cet aspect général de la carte comme on peut le voir sur l'image.

rendu 2.jpg 

2ème soutenance soutenance :

Ensuite, pour la deuxième soutenance, le travail apporté sur la map était très différent de celui de la première soutenance. Cette fois-ci, je me suis occupé d'ajouter les éléments qui pourront être ramassé(bois, pierre, ...) par le joueur afin d'être utilisés lors de crafts ou autres. Il a donc fallu chercher des assets qui correspondent et les répartir un peu partout sur la carte pour que le joueur puisse les trouver. En plus de cela, on a également ajouté un lac sur l'île qui pourra aussi servir de point d'eau pour le joueur.



3ème soutenance :

Pour la dernière soutenance, j'ai réalisé un travail pour améliorer les textures des montagnes et du sol, les rendre plus réalistes, j'ai également continué a placé des objets ramassables sur la carte, des buissons, revoir le nombre d'objets déjà placés lors de deuxième soutenance et en rajouter.

Il y a également eu quelques changements notables sur la map avec la création d'une zone un peu marécageuse et une révision des formes du sol pour rendre le terrain un peu plus sauvage. On a également ajouté de l'herbe au sol pour améliorer l'environnement de jeu.

3.3.2 Sound design

2ème soutenance soutenance :

J'ai commencé le sound design pour la deuxième soutenance, et je l'ai aussi quasiment fini pour cette soutenance. L'essentiel du travail consistait à rechercher des sons qui correspondent à une animation ou à une ambiance dans le jeu (ex : animation d'attaque, musique de victoire). Pour ce faire j'ai consulté plusieurs banques de sons en ligne pour trouver des bruitages et musiques.

La plupart du temps les bruitages nécessitaient d'être retouchés pour correspondre correctement à l'animation ou l'ambiance voulu. J'ai donc utilisé un site pour découper les sons ou les superposer afin de les retravailler. On peut voir sur l'image qui suit une liste des sons et musiques trouvés, les sites où je les ai trouvés (pour les bruitages) et les auteurs des musiques.

armes		monstre 1	
	gourdin		bruit d'attaque (1 par paterne)
	arc		bruit de mort
	marteau		bruit de déplacement
	harpon	monstre 2	
environnement			bruit d'attaque (1 par paterne)
	eau		bruit de mort
	feu		bruit de déplacement
	vent	monstre 3	
ambiance			bruit d'attaque (1 par paterne)
	musique de combat		bruit de mort
	musique calme/routine		bruit de déplacement
	musique normale/défaut(également menu)	https://www.sound-fishing.net/	
	musique victoire		
	musique défaite		
personnage		https://www.auboutdufil.com/index.php?mood=angry	
	pas dans l'eau		
	pas sur sable		
	pas dans plaine/forêt	https://www.musicscreen.be/musique-libre-de-droit-action1.html	
	craft(tout craft confondus)		
	respiration		
	manger	https://www.musicscreen.be/musique-libre-de-droit-action1.html	
	boire		
	ramasser		
	douleur/dégat	musiques:Tristan Lohengrin - Bittersweet Season/Tai Chi - Valtteri Kujala/Lajos Baetzel-The Journey before	
	mort		

3ème soutenance soutenance :

Ensuite, pour la troisième soutenance je n'ai qu'eu a rajouté quelques sons qui manquaient. Notamment des bruits effrayants pour les monstres en mouvement.

3.3.3 Site web

Pour le site web, je me suis occupé du texte d'accueil du site web dans lequel je présente le groupe et rapidement le projet.

4 Récapitulatif final des items :

Voici un résumé final des objets et de toutes les caractéristiques après équilibrage par rapport à ce qui a été annoncé dans le cahier des charges

4.1 Les objets plaçables :

Cabane :
La cabane est un objet essentiel a acquérir en debut de partie car elle permet de dormir. Une obligation sous peine de mourir.

Nom	Cout	Temps	Spécifications
Cabane de niveau 1	30 batons 5 feuilles 5 lianes	1 heures	permet de dormir
Cabane de niveau 2	70 batons 20 feuilles 20 lianes 10 pierres	8 heures	permet de dormir protèges deux nuit contre les monstres
Cabane de niveau 3	100 batons 40 feuilles 40 lianes 30 pierres	2x12 heures	permet de dormir protèges cinq nuit contre les monstres

Feu :
Le feu de camp permet de faire cuire ses lapins et ses poissons pour les rendre comestible.

Nom	Cout	Temps	Spécifications
Feu de camp	10 batons	30 minutes	permet de cuire la nourriture
Grand feu	50 batons	2 heures	Avoir une chance supplémentaire d'être évacué par l'avion.

Chasse :
Le collet est plaçable près des terrier de lapin, ils sont limite a trois et on une chance sur 4 d'attraper un lapin par jour.

Nom	Cout	Temps	Spécifications	vitesse de chasse
Collet	1 lianes	1 heures	permet de chasser limité à 3 collets	0,25 lapin / Nuit du joueur

4.2 Armes

Armes aux corps à corps :

Les armes au corps a corps permettent de faire des dégât aux mobs. Elles attaquent a plus faible distance que les autres armes mais fond plus de dégât.

Nom	Cout	Temps	Spécifications	dégâts	distance(m)
Gourdin	2 batons	instantanée	attaque les monstres au corps à corps	8	1
Marteau	1 batons 1 lianes 1 pierres	2 heures	attaque les monstres au corps à corps	20	1

Armes à distances :

Les armes a distance permettent d'attaquer les monstres sans que le joueur ne se mette en danger.

Nom	Cout	Temps	Spécifications	dégats	distance (m)
Arc	1 batons 1 lianes	1 heure	attaque les monstres à distance Nécéssite des fléches	10	40
Fléches	1 baton	2 heures	Utilisable avec l'arc	none	none

Armes / outils de pêches :

Les harpons on deux utilités ils permettent de pêcher et d'attaquer les monstre avec une plus longue porté que les armes aux corps a corps.

Nom	Cout	Temps	Spécifications	dégats	distance (m)	vitesse de pêches
Harpon basique	1 batons	1 heure	Permet : pêcher/attaquer	8	2	1 poisson / h
Harpon Amélioré	1 baton 1 liane 1 couteau	1 heures	Permet : pêcher/attaquer	18	2	2 poisson / h

4.3 Consommables

- Poisson : Nécessite d'être cuit (Redonne 20 points de la barre de nourriture sur 100).

- Lapin : Nécessite d'être cuit (Redonne 30 points de la barre de nourriture sur 100).

- Baies : Ce mange cru redonne 2 point de la barre de vie sur 100 et soigne du poison du monstre 2 qui peut

5 Bilan personnel

Section dédiée à comment nous avons vécue le projet et ce qu'il la pu nous apporter.

5.1 Pierre

Nous voilà à la fin de se projet et je ne m'en réjouis pas au contraire ! J'ai beaucoup aimé travaillé dessus avec mon groupe. En accord avec les autres membres du groupe, j'ai particulièrement prit à coeur mon rôle de chef du groupe en planifiant ce que chacun devait réaliser.

Je n'ai malheureusement pas eu la chance de faire des projets de ce genre jusqu'à présent qui sont, je le pense sincèrement, un atout pour notre vie futur d'ingénieur. En effet, plus tard nous ne travaillerons rarement seul sur des projets et travailler avec les autres n'est pas quelque chose d'inné, cela s'apprend au fur et à mesure du temps que nous travaillons avec les autres. Nous avons personnellement tous des habitudes de travail différentes, des plages horaires différentes...Ce projet m'a permis d'être plus à l'écoute des autres membres et moins entêté dans mes idées.

Depuis toujours, j'aime faire les choses seul avec le moins d'aide possible. Cela peut paraitre bizarre, mais travailler avec les autres n'a jamais été facile pour moi. Je ne veux pas paraitre prétentieux mais j'ai une petite tendance à prendre les devants sans penser aux autres. J'ai rapidement compris que si je faisais cela pour ce projet je n'irais pas loin et que cela ne m'apporterai rien. J'ai vite vu que l'en travaillant efficacement en équipe nous devenons rapidement plus efficace.

Du côté technique, ce projet m'a grandement appris à maitriser le C# et surtout l'orienté objet que l'on a vu en TP et que j'ai pu mettre en pratique. J'ai bien sûr appris à maitriser Unity pour créer le jeu. Autour du projet, pour créer des pages du site web j'ai du me mettre au java script, html et css. Trois choses dont je n'avais touché auparavant tout comme la mise en place d'un VPS. J'ai pu mettre en oeuvre ce que je savais déjà sur Photoshop pour retoucher ou créer des images comme notre logo et la cover de notre jeu. J'ai également mis à profit mes compétences sur Adobe première pro et final cut pro pour créer les vidéos de nos soutenances et le trailer.

Pour finir, j'ai appris à maitriser le LaTeX pour rédiger des rapports qui sera un atout pour la suite.

Ce fut globalement une très grande expérience pour moi que je suis prêt à remettre pour la suite.

5.2 Jean

Le projet m'a permis pour commencer de découvrir Unity, une partie de ses fonctionnalités et j'ai notamment beaucoup apprécié créer la map sous Unity et me former sur le logiciel. Le projet ma également permis d'en découvrir plus sur le déroulement d'un projet informatique. Je retiendrai particulièrement l'importance qu'il faut accorder au cahier des charges en début de projet car c'est lui qui va nous guider tout le long de ce dernier, faire attention à ne pas le surcharger pour éviter de devoir retirer des choses ou d'avoir des retards et ne pas oublier des éléments qui manquerai plus tard.

Ce projet m'aura fait découvrir un peu plus le travail en équipe, comment s'organiser pour travailler et gérer les éventuels problèmes (ex : problème de push sur la collab Unity que l'on a pu avoir et qui nous a demandé plus de rigueur sur nos push). Pour finir, ce projet m'aura permis d'en découvrir un peu plus sur la réalisation d'un jeu vidéo, sujet qui m'avait un peu toujours intriguée et sur lequel j'ai pu obtenir des réponses.

5.3 Brice

Le projet est maintenant sur ça fin, après 6 mois de travaille sur notre jeu. Bien que tous n'aura pas été toujours facile, surtout pour la motivation, je suis vraiment content d'avoir réussi à remplir les objectifs, fixés il y a 6 mois dans un cahier des charges qui pouvait paraître flou et rempli de chose compliqué à réaliser.

Une des choses sur la quelle, nous à bien fait travailler le projet est la collaboration avec ses collègue. En effet déjà dès le début du projet nous avons été confronté à ce problème, car bien que la communication était bonne, chacun travaillais de son côté se qui a rendu le projet difficile à rassembler pour la première soutenance. Le problème à été résolu pour la deuxième, car on ne voulait pas répéter l'erreur, en travaillant tous sur la collaboration unity.

Cependant un deuxième évènement est venu mettre notre travaille d'équipe à l'épreuve, le confinement. Pour le coup on à plutôt bien réussi à gérer cela, en étant en appelle sur discord dès qu'une autre personnes travaillait en même temps que nous.

Le projet nous a aussi apporté un nouveau lot de compétences tel que la création d'un site web et la gestion du serveur pour l'héberger. La maîtrise de unity et une approche de ce à quoi ressemble la création d'un jeu vidéo. Mais plus important que cela le projet nous à permis de mettre en pratique nos compétences, ce qui nous à fait nous rendre compte, que l'on était déjà capable de faire beaucoup de chose.

Pour conclure se rapport je tenait à dire que je suis particulièrement fière de ce que notre groupe a accompli, car malgré les moments de baisse de motivation nous avons réussi à produire un travaille constant pour mener à bien le projet.

6 Conclusion

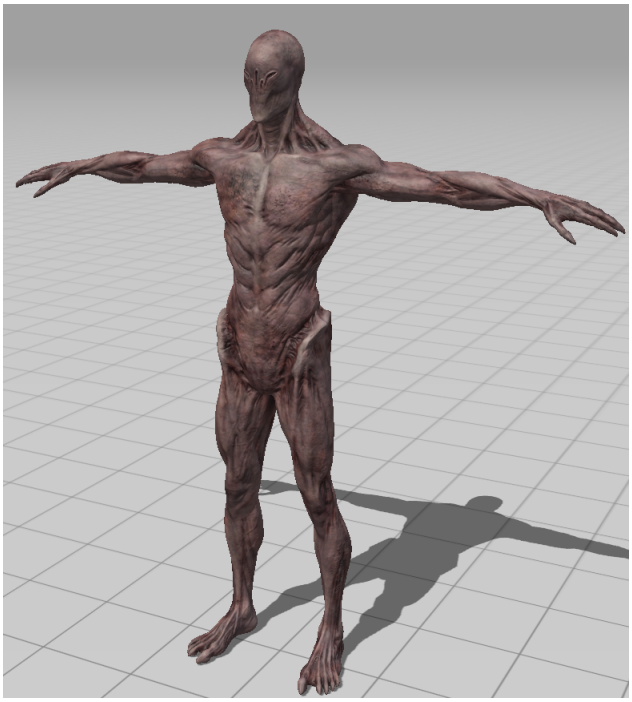
En conclusion, le jeu est terminé, on a réussi à faire ce que l'on voulait et tout ce qu'on avait prévu de faire. On est très content et fier de la version finale du jeu que l'on a créée et d'avoir su respecter les délais et les tâches que l'on s'était fixés. Nous avons déjà tous détaillés dans nos bilans personnels ce que nous a apporté le projet et il en ressort que cette "aventure" aura été une expérience enrichissante pour chacun de nous et ce à tous les niveaux.

Nous pouvons aussi souligner les aspects bénéfiques qu'a eu le projet sur la cohésion du groupe, nous avons tous appris à mieux nous connaitre durant les divers moments de travail en équipe et cela nous a permis de nous renforcer mutuellement. Enfin, on consacre les dernières lignes de ce rapport pour remercier EPITA qui nous a donné l'opportunité de réaliser un tel projet ainsi que toute l'équipe enseignante pour leur suivi durant tout son déroulement.

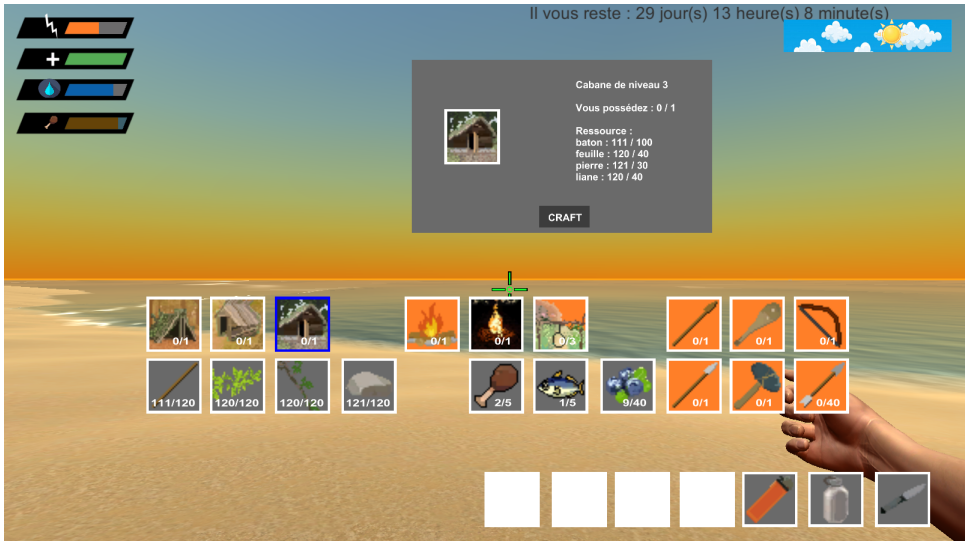
7 Annexes

Monstre 3 (le boss)

Monstre 1



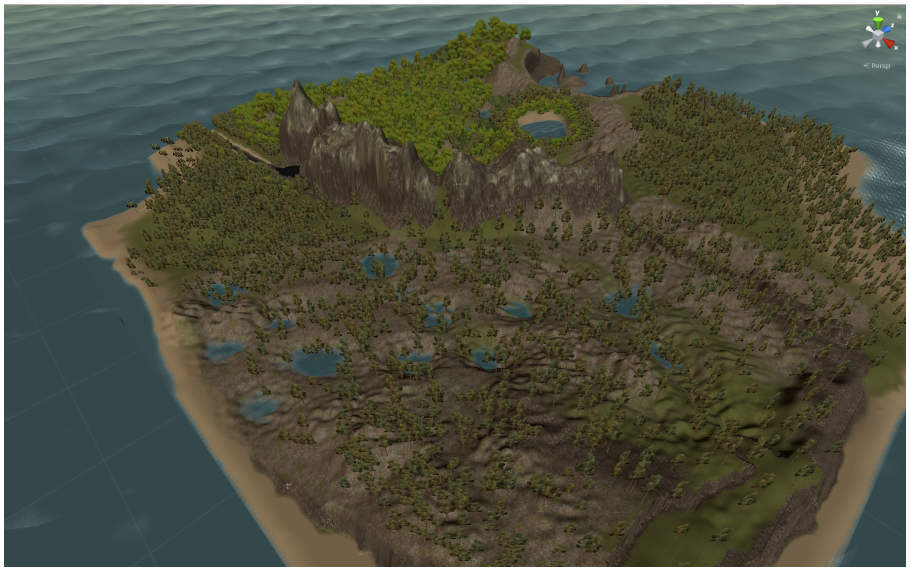
Monstre 3 (monstre qui infecte)



Inventaire

Baie / Buisson

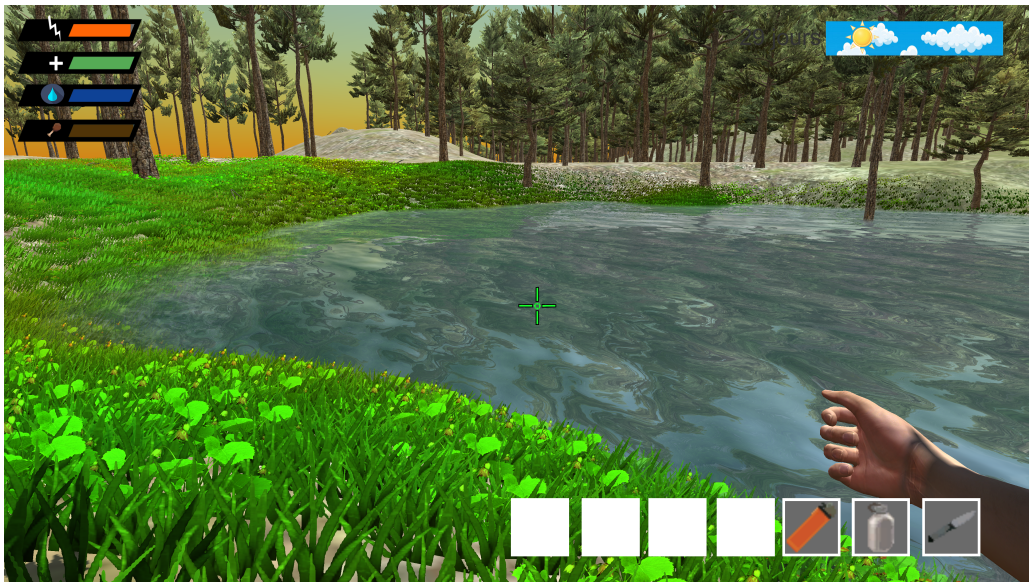
Source d'eau potable du jeu



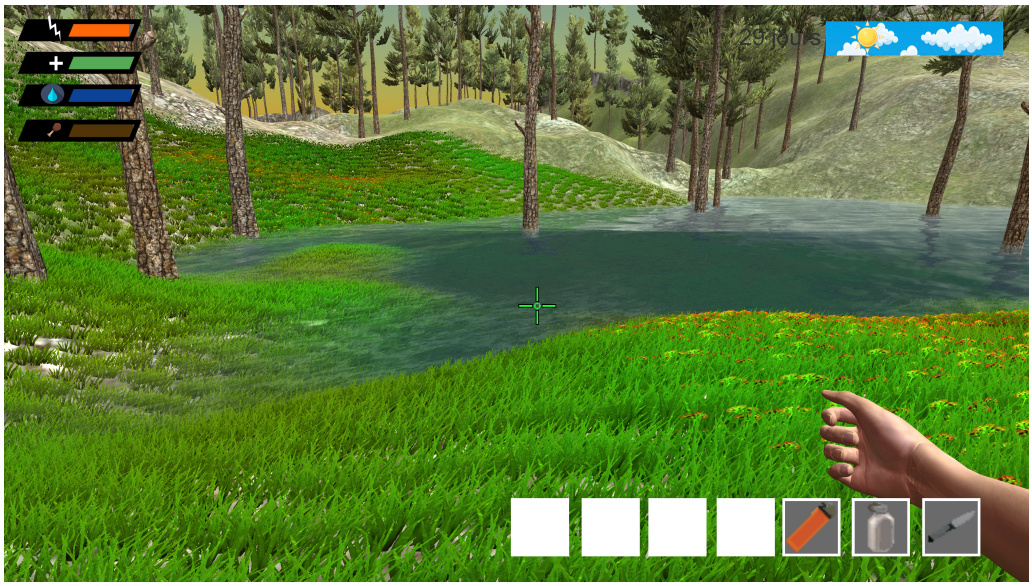
Rendu final de la carte

point d'eau Peche

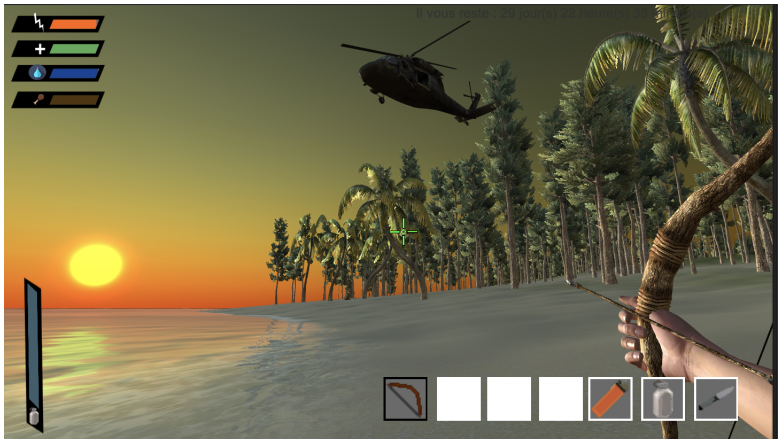
Végétation



Marécage



Marécage



Victoire