Rapport de soutenance #2
The forust
Alpha-c

Jacobé "Way hd" Pierre

Boisson "zraulix" Brice

Guérin "Shmiti" Jean

Table des matières

1 Introduction

Tout d'abord, nous pouvons dire que notre projet semble en bonne voie par rapport à nos prévisions durant le précédent rapport de soutenance ainsi que par rapport à notre cahier des charges.

L'utilisation de la collaboration Unity nous permet d'être d'autant plus efficace vis à vis de notre travail de groupe. Cette même cohésion qui avait été un problème pour nous au début de notre projet, est aujourd'hui soudé malgré le travail à distance et les mesures mise en place ces derniers temps. De plus nous avons appris à mieux répartir les tâches mais surtout pour cette soutenance, mieux relier le travail que chacun avait un peu fait de son côté pour la première soutenance, par exemple lier l'inventaire et les mécanismes de presque tous les items du jeu.

Nous avons refait l'aspect graphique du site. De plus nous affirmons avoir rattrapé notre retard vis à vis de certains points comme le sound design. Le but étant pour cette soutenance de finir d'implémenter les mécaniques principales du jeu tel que le combat, la gestion de l'inventaire et du temps.

Toutefois, nous avons rencontrés quelques problèmes à la fin de la première soutenance, plus particulièrement au niveau de l'inventaire. Grace à notre cohésion de groupe nous réussi à refaire un inventaire fonctionnel et plus adapté à notre jeu pour cette soutenance.

Au fur et à mesure de pratiquer Unity et le C#, nous avons appris beaucoup de nouvelles choses, notamment comment être mieux organisés mais aussi avoir un code plus optimisé, plus facile à lire pour nous et pour les autres.

Au delà d'être un projet pour notre formation ce projet est une véritable aventure humaine, permettant de comprendre comment chacun fonctionne et donc mieux travailler en groupe.

2 Etat actuel

2.1 L'orienté objet

2.1.1 Pierre

Implémentation de nouvelles méthodes dans la classe Player, plus particulièrement concernant la gestion du temps, propriété inhérente au joueur et essentielle dans notre jeu.

```
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      }
```

Dans notre jeu le temps c'est de l'argent ou plutôt l'argent c'est du temps, Dans le cahier des charges nous avions précisés qu'une journée durerai 30min, nous avons abaissé ce temps à 20 min. Revenons à la méthode "Updatetime()", par exemple si le joueur décide de crafter une arme qui a un coût de 8 heures, il faut que ce coût se répercute sur la faim, la santé et le sommeil.

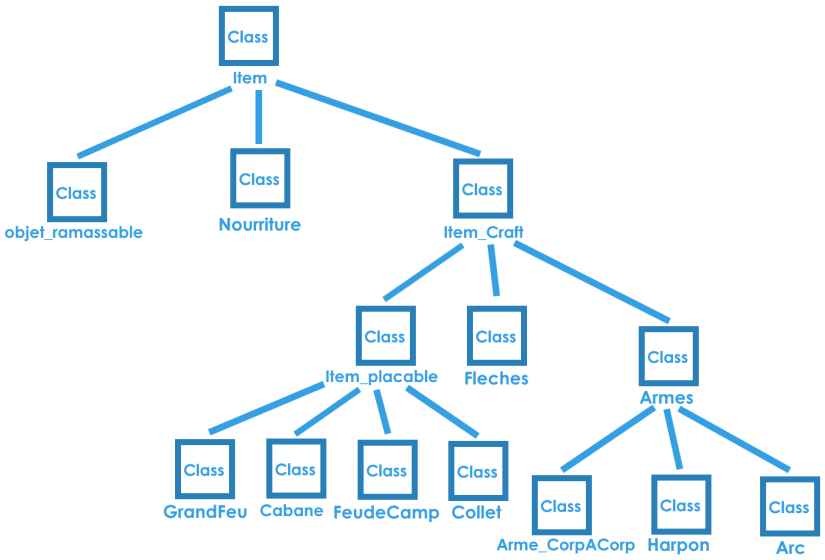
Enfaite c'est comme s'il avait passé 8 heures sans régénérer les trois éléments cités au-dessus. Cette méthode est particulièrement utile pour Brice, qui peut l'appeler pour actualiser les données du joueurs qu'il faut lors du craft d'un item. J'ai aussi fini d'implémenter toutes les méthodes permettant soit au joueur de gagner de la vie ou d'en perdre, par exemple.

Finalement avoir intégré l'orienté objet dès le début est un immense gain de temps mais en plus, rendant le code plus lisible sans réimplémenter les mêmes fonctions à chaque fois.

j'allais oublier, cette classe hérite de la classe "Entité". De plus il existe aussi la classe "Monstre" qui hérite elle aussi de "Entité". Toutefois, comme le Monstre ne peu que donner ou recevoir des dégâts, ses méthodes et propriétés sont beaucoup plus "basique".

2.1.2 Brice

Avec la fin de l'implémentation des items voici donc le hiérarchie d'héritage des items.



2.2 AI

2.2.1 Pierre

Comme précisé dans le rapport de soutenance numéro 1, deux nouveaux monstres ont été introduis, ayant chacun des spécificités différentes. Parlons du monstres 1, il est très rapides, peut détecter le joueur de très loin et vient l'attaquer quand il rentre dans sa zone. En revanche il ne possède que de hp et fait peu de dégâts joueur.

Le deuxième monstres quant à lui est le plus fort du jeu, il est donc très résistant et inflige beaucoup de dégâts. En revanche son champ de vision est plutôt limité et il est assez lent.

Etudions le code de l'un des deux monstres, ils sont assez similaire :

```
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      }
```

Donc si la distance du monstre (Monster.remainingDistance) est supérieure à la "walkdistance" il ne fait rien, il est en état de "Stand-by", distance mise à 10 unités du jeu. En revanche, en-dessous il se met à courir vers le joueur grace à son animation, puis tombé sous deux unités il se met à attaquer, toujours grace à son animation. Dans la zone où le joueur se fait détecter, le monstre va vers lui grace aux propriétés "NavMeshAgent" de Unity expliquées dans le premier rapport de soutenance. Elles permettent entre autre, de fixer un point pour le monstre donc ici notre joueur et de toujours aller vers lui tant que cette propriété est active. De plus, elle peut s'adapter aux conditions du terrain.

Durant l'animation "Attack", un appel est fait à la méthode :

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

qui permet au monstre d'infliger au joueur les dégâts qu'ils faut grace à la méthode "TakeDamageHP" du player.

Ce qui nous fait pour cette soutenance trois monstres totalement fonctionnel, avec chacun leurs propriétés indiquées dans le rapport de soutenance, permettant une vrai expérience de jeu au joueur.

2.2.2 Brice

Comme indiqué dans le dernier rapport au niveau de la section objectif, pour la deuxième soutenance, j'ai appris comment fonctionnait les IA dans Unity durant cette période. Une fois l'apprentissage finis, je l'ai tout de suite mis en pratique en ajoutant une caractéristique supplémentaire au monstre.

J'ai fait en sorte que le monstre n'attaque le joueur que lorsqu'il est devant lui, et si ce n'est pas le cas qu'il tourne sur lui même jusqu'à trouver le joueur. Bien sure cela tant que le joueur reste à proximité du monstre.

```
if (agent.remainingDistance < 2 )
      {
       Ray ray = new Ray(new Vector3(agent.transform.position.x,agent.transform.position

       if (Physics.Raycast(ray,out hit , Mathf.Infinity))
       {
          if (hit.collider.name == target.name )
          {
             ennemi.SetBool("Attack", true);
          }
          else
          {
             ennemi.SetBool("Attack", false);
             agent.transform.Rotate(Vector3.up * 10);
          }
       }
       else
       {
          Debug.Log(2);
          ennemi.SetBool("Attack", false);
          agent.transform.Rotate(Vector3.up * 10);
       }
```

Cependant cette méthode nécessitera d'être amélioré car elle ne rend pas les mouvements du monstre très naturel.

2.3 Combat

2.3.1 Pierre

Sur ce point, je suis en corrélation avec ce qui était annoncé. En effet pour cette soutenance j'ai implémenté, comme prévu, l'arc et le harpon.

En ce qui concerne l'arc, nous pouvons tirer des flèches avec un vrai système de gestion de la force avec le clic droit, plus on reste appuyer plus la force dans la flèche augmente, et quand on relâche la flèche part avec cette force (avec bien sûr une certaine limite). De plus, un viseur avec le clic droit a été introduit. En maintenant le clic droit on passe sur le viseur et donc on gagne précision. Bien entendu, la flèche inflige les dégâts qu'il faut au monstre.

```
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     }
```

Un "IEnumerator" permet de faire une pause dans le code avec "yield return new WaitForSeconds(0.1f)", tant que l'on reste appuyé sur le clique gauche, cette méthode est appelée, sauf qu'elle exécute une pause de 0.1 seconde avant d'ajouter de la puissance dans la flèche. De plus, quand la puissance ce "charge", du recule est ajouté à la flèche.

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

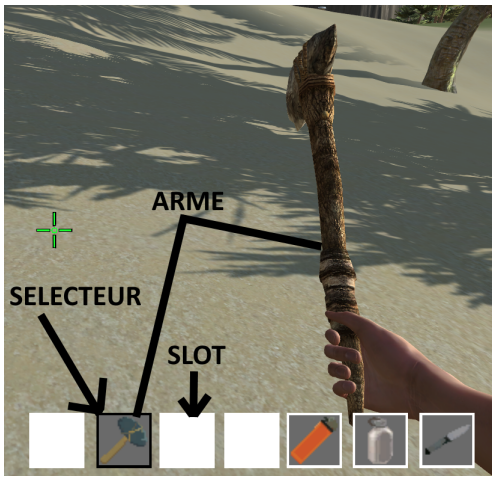
La méthode "AddForce" permet physiquement d'ajouter toute cette force dans la direction que l'on veut et donc faire partir notre flèche.

Le harpon à lui aussi été ajouté, c'est une arme à courte distance peu efficace contre les ennemis, en revanche elle est un atout majeur pour la pêche et donc la nourriture. Ceci sera expliqué plus en détail dans les sections suivantes.

Si seulement le joueur peut attaquer cela n'aurait bien sûr que peut d'interêt, c'est pourquoi comme déjà dit en parti dans la section au-dessus les monstres peuvent eux aussi combattre avec des animations différentes pour chacun. Le joueur devra donc adapter ses armes en fonctions des monstres rencontrés. C'est à dire pour un monstre ayant un champ de vision réduit l'arc pourra être privilégié, mais rien est imposé au joueur à lui d'adapter son gameplay et son approche.

2.3.2 Brice

Comme prévu durant cette période j'ai finis d'implémenter les items d'armes pour gérer les armes en tant que ressources c'est à dire les craft. Je me suis aussi occupé avec l'aide de Pierre du système pour choisir l'arme que l'on veut utiliser c'est à dire la mettre dans les slots depuis l'inventaire, dans le jeu choisir le slot que l'on veut utiliser et enfin activer l'arme correspondante.



2.4 Item

2.4.1 Pierre

Ici je parlerai de l'implémentation des mécaniques des différents items dans le gameplay.

Premièrement, j'ai implémenté un briquet (ou allume feu quelques désaccord a propos de ça entre nous :)) et donc le feu de camp qui va avec dans la mécanique. Comme prévu, le briquet est disponible dès le début de partie pour le joueur. Il dispose d'un réservoir s'écoulant à chaque utilisation et ne pouvant se régénérer.

```
1       IEnumerator ok()
2   {
3       firecamp.SetActive(true);
4       g = false;
5       yield return new WaitForSeconds(10f);
6
7
8       firecamp.SetActive(false);
9       g = true;
```

```
10      }
11
12      IEnumerator capacity()
13      {
14          current = current - 10;
15           f = false;
16          yield return new WaitForSeconds(1f);
17           f = true;
18
19      }
20
21      // Update is called once per frame
22      void Update()
23      {
24          briquet.fillAmount = current / vie;
25          if (Input.GetMouseButton(0))
26          {
27
28              if (f && current >0)
29              {
30                  briquet.enabled = true;
31                  fire.SetActive(true);
32                  StartCoroutine(pause());
33              }
34              p = true;
35          }
36
37          if (Input.GetMouseButtonUp(0))
38          {
39              briquet.enabled = false;
40              fire.SetActive(false);
41              p = false;
42          }
43      }
44
45      private void OnTriggerStay(Collider other)
46      {
47          if (other.tag == "feu" && p && g && current >= 0)
48          {
49              StartCoroutine(ok());
50
51          }
52
53      }
```

Je pense que la mécanique est assez complète et mérite d'être détaillée ici. La capacité du briquet est de 100 au départ et diminue de 10 par seconde d'utilisation. C'est la méthode "capacity()" qui permet de faire cela, même méthode que l'arc pour l'ajout de la force, on fait une pause de 1 se-

conde dans le code puis on diminue de 10. Pour rappel le briquet permet d'allumer le feu, qui lui permet de faire cuire notre nourriture, cela permet de faire le lien entre l'implémentation "inventaire" de Brice et la mienne dans le jeu.

Les booléens, f et g permette de savoir quand est-ce que je peux rappeler les méthodes "ok()" et "capacity()", donc à la fin de la pause dans le code. Le booléen p permet de différencier l'appuie sur le clic gauche sur la souris et lorsque qu'on relâche.

Dans la méthode update on actualise la capacité du réservoir, celle que le joueur peut voir sous forme d'image grace à "briquet.fillAmount = current / vie", qui permet de diminuer la taille de l'image et donc du réservoir. Elle permet aussi de gérer les input, c'est à dire actualiser le booléen p en fonction du clic.

Pour finir, la méthode OnTriggerStay permet de savoir quand le joueur rentre dans le "field" du feu de camp et ainsi allumer le dit feu. Et bien sûr je n'oublie pas de vérifier la capacité du réservoir.

Comme expliqué dans la partie combat (2.3) et la suivante j'ai implémenté une partie de la pêche avec le harpon pour que Brice puisse ajouter le script qui permet l'ajout dans l'inventaire.

2.4.2 Brice

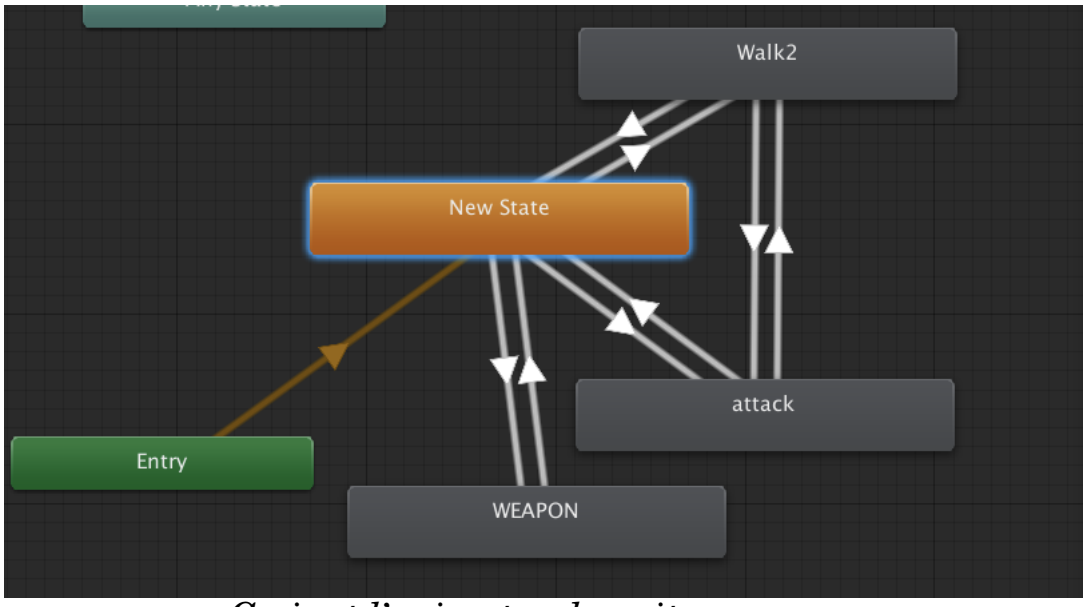
Durant cette période j'ai tout d'abord finis d'implémenter tout les items au niveau de l'inventaire. J'ai entre autre ajouté les armes, la nourriture, les items plaçable. Une fois ces items implémentés j'ai commencé à m'attaquer à la partie utilisation de ses items dans le jeu avec le poisson, et le lapin que l'on peut désormais faire cuire et mange à l'aide du feu.

```
1  private void OnTriggerStay(Collider other)
2      {
3          int indice = position du selectionneur de slot
4          if (indice < 4)
5          {
6              Items itemHand = item selectionne
7              if (itemHand is Nourriture && Torch.activeSelf)
8              {
9                  if (itemHand.name == "poisson" && InventoryObject.poisson.own > 0)
10                 {
11                     affichage du message
12                     if (Input.GetKeyDown(KeyCode.E))
13                     {
14                         actualisation des valeur de sommeil, eau, ... en fonction du temps
15                         InventoryObject.baton.own -= prix de cuisson
16                         Augmentation de la nourriture en fonction de l'apport du poisson
17                         InventoryObject.poisson.own--;
18                     }
19                 }
20                 else if (itemHand.name == "lapin" && InventoryObject.lapin.own > 0)
21                 {
22                     affichage du message
23                     if (Input.GetKeyDown(KeyCode.E))
24                     {
25                         actualisation des valeur de sommeil, eau, ... en fonction du temps
26                         InventoryObject.baton.own -= prix de cuisson
27                         Augmentation de la nourriture en fonction de l'apport du poisson
28                         InventoryObject.lapin.own--;
29                     }
30                 }
31                 else
32                 {
33                     retire le message
34                 }
35             }
36             else
37             {
38                 retire le message
39             }
40         }
41     }
```

2.5 Animations

2.5.1 Pierre

L'animation du harpon a été implémentée pour cette soutenance, c'est une animation qui est déclenchée via le clic gauche de la souris et fait partir le pic du harpon sur une courte distance



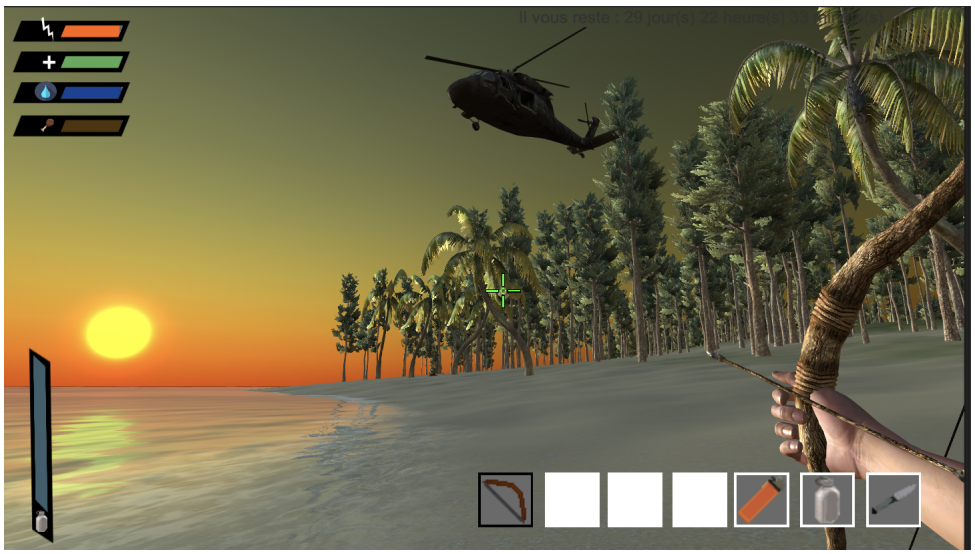
Ceci est l'animator de unity

Pour rappel, les rectangles gris représentent des animations, les flèches blanches des transitions pouvant être controlées à l'aide de booléens et donc du C# et le rectangle orange l'état par défaut dans lequel il ne se passe rien. Les deux autres animations ont déjà été présentée à la soutenance précédente, en revanche la nouvelle animation "WEAPON" représente celle du harpon. Pour rappel a proximité de certains points d'eau, le joueur pourra récupérer des poissons avec 1/4 de chance d'en attraper un, Brice en parlera plus en détail dans sa partie.

En ce qui concerne les animations des monstres, ce son les seuls que je n'ai pas créée de moi même. En revanche comme évoqué dans la partie AI (2.1.1), je me suis occupé à l'aide du C# et de l'animator Unity, de relier les animations et transitions entre elles, c'est-à-dire passer de l'état "stand-by" $\leftrightarrow$ courir vers le joueur $\leftrightarrow$ l'attaquer. Chaque monstres à désormais une animation unique pour courir et attaquer ainsi que dans l'état où il ne fait

rien.

Comme prévu pour cette soutenance, j'ai implémenté l'animation de fin lorsqu'il gagne la partie. En effet, le joueur aura 30s pour se rendre sur la plage à fin de monter dans l'hélicoptère qui l'évacuera de l'Ile.



Evacuation

Voici un petit rendu de ce que cela donne, avec en prime un petit couché de soleil sur notre Ile dont je reparlerai après.Une fois la partie gagné un hélicoptère arrive, il atterrit, le joueur dans un rayon de 1 mètre de l'hélicoptère devra appuyer sur E, ce qui le fera monter dans l'hélicoptère. Ensuite l'hélicoptère décolle de manière autonome, avec une dernière vue sur l'ile pour le joueur.

En ce qui concerne le code que j'ai implémenté pour réaliser cela, petit rappel :

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

Je déclare un rayon qui est positionné sur le joueur, lorsqu'il heurt un objet, c'est la variable "hit" qui s'occupe de récupérer toutes les informations nécéssaires.

En ce qui concerne notre animation :

```
1  if (hit.transform.tag == "Helico" && hit.distance < 1 && p)
2   {
3   GameObject.Find("FPSController").GetComponent<FirstPersonController>().enabled = false;
4              transform.position = new Vector3(484.9f,3348.53f,634.5f);
5              transform.rotation = Quaternion.Euler(0,30.256f, 0);
6              transform.parent = heli;
7    }
```

Premièrement, grace à la variable hit au-dessus, on vérifie si ce que l'on a touché avec le rayon est bien un hélicoptère, ensuite si la distance est bien inférieur à un mètre et enfin si l'on appuie sur la touche E.

Ensuite on positionne notre joueur dans l'hélicoptère et enfin on passe le joueur en enfant de l'hélicoptère dans le but que celui-ci parte bien avec lui, grace à la méthode "transfrom.parent".

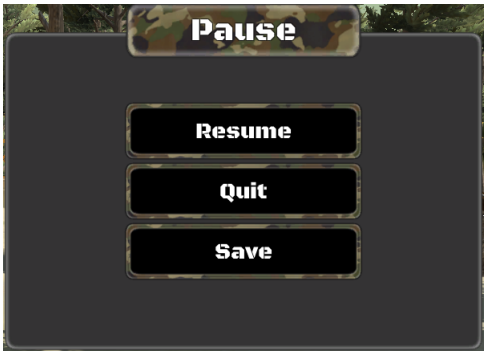
De plus, l'animation du coup de marteau donné par le joueur est désormais moins "robotique" et plus fluide.

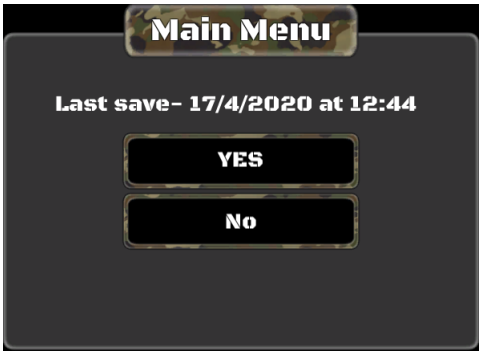
2.6 Interface

2.6.1 Pierre

Pour cette soutenance je me suis occupé en partit du système de sauvegarde et de l'implémenter dans l'interface. Donc en résumé des fonctionnalités possible que j'ai mise en place : si l'on a pas de sauvegarde en cours cela lancera la partie lorsque l'on appuiera sur "Play", sinon une fenêtre proposera au joueur de lancer sa dernière sauvegarde avec la date de celle-ci ou alors de lancer une nouvelle partie.

Pour pouvoir sauvegarder à tout moment j'ai implémenté un bouton save dans le menu pause.





Donc à gauche le menu pause avec le bouton save et à droite le menu qui apparaitra au joueur si il a sauvegardé une partie avant. Si le joueur appuie sur "yes" il reprend sa dernière partie et "no" il en relance une nouvelle.

Donc moi je me suis occupé de sauvegarder les données relative au joueur, c'est-à-dire les points de santé, de nourriture ect... ainsi que la où en est le joueur dans le temps. Quelques explications :

```
1          public void save()
2      {
3          GameObject player = GameObject.Find("FPSController");
4          string a = "Last␣save-␣" + DateTime.Now.Day + "/" + DateTime.Now.Month + "/" + Date
5                  DateTime.Now.Hour + ":" + DateTime.Now.Minute;
6
7          //POSITION DU JOUEUR
8
9          float x = player.transform.position.x;
10         float y = player.transform.position.y;
11         float z = player.transform.position.z;
12
13         //CARACTERISTIQUE DU JOUEUR
14
15         float food = player.GetComponent<Player>().player1.Food;
16         PlayerPrefs.SetFloat("x",x);
17         PlayerPrefs.SetFloat("y",y);
18         PlayerPrefs.SetFloat("z",z);
19         PlayerPrefs.SetFloat("food",food );
20         PlayerPrefs.SetString("date",a);
21     }
```

Je conçois que ce code est plutôt répétitif c'est pourquoi je n'ai pas tout écrit ici mais globalement c'est ce qui se cache sous le bouton "save", c'est pourquoi nous étudierons une refonte plus efficace de ce système pour la dernière soutenance. Sinon sur Unity, la sauvegarde des données en C# ce

fait à l'aide de *"PlayerPrefs"*, donc ici *SetFloat* permet d'enregistrer la valeur en lui associant une *key* qui est une string par exemple "x", si je veux récupérer cette valeur j'appelle la méthode *GetFloat* en lui passant en paramètre la key.

Il existe d'autres méthodes intéressantes associés aux *PlayerPrefs*, par exemple vérifier si une donnée existe ou non, dans mon cas cela veut dire qu'une sauvegarde existe et que je dois la charger. Si le joueur veut recommencer une nouvelle partie, alors j'efface les données existantes à l'aide d'une méthode qui permet de tout faire en une fois.

2.6.2 Brice

Pour ma part je me suis occupé de designé tout les logos des items proprement en 2D. J'ai aussi implémenté de manière propre l'interface de l'inventaire, de selection des slots ainsi que du menu de craft en ajoutant des indications visuelle tel que le nombre d'item disponible, ou mettre en vert les items qu'on possède en orange ce qu'on peut craft et en rouge ce que l'on ne peut pas craft. Pour rendre tout cela plus adaptable au différent utilisateur jouant au jeu j'ai réimplémenté tout cela dans des Canvas, un outil proposé par Unity pour adapté automatiquement l'interface à l'utilisateur.

2.7 Gestion des différentes barres et du temps

2.7.1 Pierre

Le sommeil et le temps sont deux éléments intiment liés, avec l'aide de Brice j'ai implémenté la mécanique du sommeil, en effet le joueur doit se présenter devant une cabane, appuyer sur E ce qui le fera dormir et passer 8h en appelant la méthode *updatetime()* du player dont je parle dans la section (2.1) orienté objet. Maintenant il doit faire attention aux valeurs des différentes barres s'il veut aller dormir.

Pour cette soutenance le temps à un réel impact en continu sur les différentes barres, et le rythme de la décroissance de ces barres est reprise de notre cahier des charges. Par exemple le joueur perd 1/3 de sa barre de nourriture par journée (représentant 20 min dans le jeu).

```
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          else
21          {
22             //
23          }
24       }
25    }
```

Avec cet IEnumerator on chercher à dininuer notre de barre de food au fur et à mesure du temps, ainsi comme on perd 1/3 de notre barre en 20 min, avec un simple produit en croix on retrouve la valeur que l'on veut pour une seconde. Donc ici pour la nourriture, tant que la quantité de nourriture du joueur excède 50, le joueur perd normalement en quantité de nourriture et peu régénérer sa barre de vie. Ensuite, il perd de la vie entre 20 et 50 de nourriture et la même quantité de nourriture, sous 20 de nourriture cette barre commence à diminuer sévèrement et la vie aussi.

Avec des valeurs légèrement différentes, le mécanisme de la barre d'eau est similaire. Toutefois pour le sommeil, s'il ne dort pas pendant 24 heures (donc 20 min en jeu) sa barre se retrouve à zéro et il perd de la vie. Si une des deux autres barres venait à tomber à zéro (nourriture et eau) alors il perd la partie.

2.8 Mécaniques de gameplay

2.8.1 Pierre & Brice

J'ai implémenté pour cette soutenance deux principaux points à notre jeu, premièrement l'algorithme du changement d'armes avec la molette de la souris :

```
1  public class Selectedweapon : MonoBehaviour
2  {
3     public int selectweapon1 =0;
4     void Start()
5     {
6        select();
7     }
8
9     void Update()
10    {
11       int previous = selectweapon1;
12       if (Input.GetAxis("Mouse ScrollWheel")>0f)
13       {
14          if (selectweapon1 >= transform.childCount - 1)
15          {
16             selectweapon1 = 0;
17          }
18          else
19          {
20             selectweapon1++;
21          }
22       }
23
24       if (Input.GetAxis("Mouse ScrollWheel") < 0f)
25       {
26          if (selectweapon1 <= 0)
27          {
28             selectweapon1 = transform.childCount - 1;
29          }
30          else
31          {
32             selectweapon1--;
33          }
34       }
35
36       if (selectweapon1 != previous)
37       {
38          select();
39       }
40
41    }
42
```

```
43      public void select()
44    {
45       if (selectweapon1 == 4)
46       {
47          transform.GetChild(20).gameObject.SetActive(true);
48       }
49       else
50       {
51          Items item = Slot.items[selectweapon1];
52          if (item is Armes)
53          {
54             if (item is ArmesCorpaCorp)
55             {
56                GetComponent<Animator>().enabled = true;
57                transform.GetChild(item.ielt).gameObject.SetActive(true);
58             }
59
60             if (item is Arc)
61             {
62                transform.GetChild(item.ielt).gameObject.SetActive(true);
63             }
64          }
65       }
66    }
67 }
```

Quelques explications, les objets fils d'un parent sont représentés sous forme de tableau, donc on fait un première appel dès le début pour activer donc le premier fils du parent (le parent est ici la main de notre joueur), "selectweapon1" représente l'arme que l'on souhaite sélectionner, si l'on touche la molette pour changer d'arme la position dans les slots d'items est enregistrée dans "previous", (on vérifie aussi lorsque l'on arrive aux extrémités du tableau pour repartir en haut ou en bas du tableau).

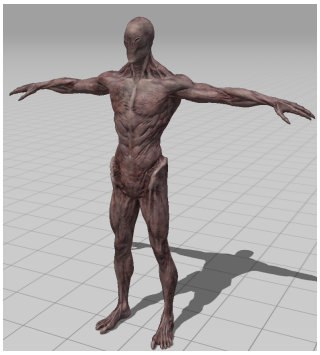
De la, on incrémente ou décrémente "selectedweapon1" en fonction du mouvement de la molette. Si "previous" est différent de "selectedweapon1" cela veut dire que l'arme a changé. Donc on appel la fonction "select" qui va se charger d'activer l'arme que l'on souhaite grace à un élément un indice déclaré dans la classe de l'arme. Pour faire simple, les armes sont fils de l'objet "main droite", les fils d'un objets sont représentés en C# par un tableau et donc on va activer l'indice de celui que l'on veut.

La deuxième fonctionnalité est l'implémentation d'un viseur. Cela consiste en une caméra placé sur le bout l'arc avec un viseur plus précis. Tout ceci s'active lorsque l'on maintient le clic droit enfoncé. Ce clic change le viseur et la priorité entre les caméras du jeu. Lors du clic, la camera sur l'avant de l'arc devient la principale et lorsque l'on relâche il se passe l'inverse.

2.9 Aspect graphique

2.9.1 Pierre

Avec l'aide de Jean j'ai implémenté dans le jeu pour cette soutenance un grand nombre des modèles 3D trouvés sur l'asset store de Unity et sur le site Mixamo.



Voici les modèles deux nouveaux monstres trouvés sur Mixamo, que j'ai mis en place dans le jeu (expliqué dans la partie animation 2.5). De plus nous pouvons voir aussi dans la partie animation (2.5) l'hélicoptère, l'arc et le briquet trouvés sur l'asset store unity, le harpon à lui été trouvé sur le site de modélisation "GrabCAD". Avec Brice nous avons implémentés tous les modèles 3D concernant les ressources comme la nourriture.

Dans le but de rendre le jeu plus réaliste, j'ai réalisé un cycle jour/nuit caller sur une journée de 20 min, comme en témoigne ce (magnifique) couché de soleil dans la section animation (2.5).

Les différentes barres plus arrondies et plus "smooth".

2.9.2 Jean

Au niveau des graphismes, la map était déjà bien avancée, il y a donc eu peu de changements à faire. On a ajouté un lac au milieu de la carte avec de la végétation autour pour enrichir l'environnement. J'ai également cherché à améliorer les textures, en trouver de meilleurs qualités ou d'autre pour augmenter la diversité de la carte. Cependant bien qu'ayant trouvé plusieurs textures qui pourrait faire l'affaire je n'ai pas encore réussi à bien les exploiter pour obtenir une amélioration notable de la map. Je les garde donc de côté pour la prochaine soutenance.

2.10 Craft

2.10.1 Brice

Durant cette période j'ai pour ainsi dire terminé le système de craft, j'ai rajouté une commande implémenté par pierre qui permet de faire passer le temps quand on craft en fonction du cout en temps du craft. J'ai aussi réglé quelques bugs trouvé dans le système de craft tels que le fait de limiter le craft au nombre maximun d'item craftable par item.

2.10.2 Pierre

Implémentation des fonctionnalités des items in-game.

2.11 Inventaire

2.11.1 Brice

J'ai presque finis l'implémentation du système d'inventaire durant cette période. J'ai comme dit plutôt commencé par réimplémenter l'inventaire à l'aide du système de canvas, puis y ai mis tout les items, j'ai ensuite rendu les items mis dans les slots disponible dans le jeu et je me suis attaqué à la résolution de certains bug, bien qu'il en reste encore quelques un tel que une duplication d'item.

2.11.2 Pierre

Je me suis occupé de tout l'ajout des différents items trouvés au sol dans l'inventaire à l'aide des rayons dans Unity et des classes implémentés par Brice. De plus je me suis occupé de lier l'inventaire "2D" de Brice dans le jeu et le rendre compatible avec les différentes classes déjà implémentées, donc j'ai couplé l'inventaire à notre joueur pour le rendre accessible in-game.

2.12 Système de ramassage/ramassage

2.12.1 Pierre

J'avais en très grande partie terminé le système de ramassage pour la première soutenance. Pour cette soutenance je l'ai juste rendu un peu plus tolérant pour que le joueur ai moins de mal à ramasser les items au sols.

2.12.2 Jean

J'ai réalisé l'ajout et le placement sur la map des objets qui vont pouvoir être ramassés par le joueur c'est-à-dire les pierres, bâtons et lianes.

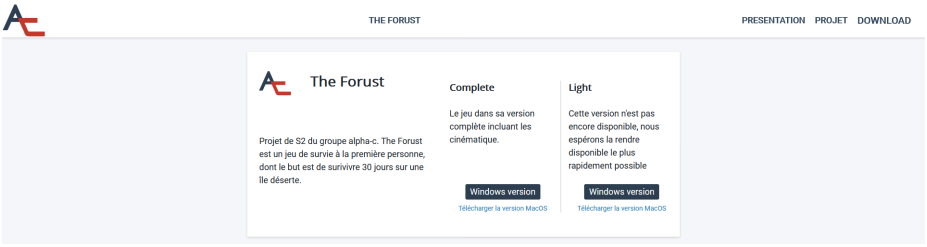
2.13 Site web

2.13.1 Pierre

Pour cette soutenance j'ai implémenté la page présentation du site, étant comme son nom l'indique la présentation de notre groupe, le site web est vraiment le seul point où je pense avoir du retard pour cette soutenance.

2.13.2 Brice

Pour cette soutenance je me suis attaqué à rendre le site web plus jolie, j'ai pour cela implémenté une barre de navigation qui bouge quand on scroll la page, et ai commencé à rendre plus jolie la page de téléchargement bien qu'elle ne soit pas finis.



2.14 Sound design

2.14.1 Jean

Le sound design à bien avancé et a rattrapé le retard qu'il avait pu prendre pour la première soutenance. J'ai fait la recherche des bruitages et musiques que l'on va pouvoir retrouver dans le jeu. Pour les bruitages j'ai consulté des banques de sons en ligne et réalisé des découpes pour obtenir des bruitages qui convenait pour l'ambiance sonore du jeu. Pour les musiques, il a été plus difficile de les trouver, car contrairement aux bruitages où le choix peut être arbitraire (ex : boire de l'eau), le choix des musiques lui doit en revanche correspondre à l'ambiance du jeu et à l'idée que l'on veut transmettre (victoire/ combat...). Il est donc plus compliqué de trouver la bonne musique sans la faire (ce qui m'était impossible). J'ai donc dû écouter de nombreuses musiques libres de droit sur divers sites pour trouver les bonnes musiques. Après avoir réuni les sons on a pu les implémenter dans le jeu.

2.14.2 Brice

Pour ma part je me suis occupé de rajouter un son lorsque le craft est réussi. Un son pour symbolisé l'échec du craft sera peut être nécessaire pour que le joueur ai une meilleur compréhension de cet échec.

3 Et après ?

3.1 AI

3.1.1 Pierre & Brice

Maintenant que nos AI sont implémentée et fonctionnel, en collaboration avec Brice, nous contons les améliorer au maximum que nous pouvons, dans le but de rendre l'expérience de jeu plus compliquer, plus réaliste, plus que ça c'est un véritable défi pour nous, mais aussi et surtout qu'elles possèdent le moins de bug possible.

3.2 Ramassage

3.2.1 Jean

Je pense que la tâche est quasiment terminée, il faudra juste veiller à ce que la quantité d'objet implémenté n'entrave pas l'expérience de jeu (ex : Il est trop difficile de trouver ces objets car il n'y en a pas assez ou au contraire trop facile car il y en a trop) et la corriger si besoin.

3.3 Combat

3.3.1 Pierre

Même si les mécanismes principales sont aujourd'hui au point et que toutes les armes disponibles sont implantées , nous pensons avoir assez d'avance sur différents point du jeu pour essentiellement concentrer nos effort sur l'IA et le système de combat car cela peut faire prendre une autre dimension à notre jeu, l'objectif est que cela soit le plus fun et plus réaliste que possible. Pour le combat, le rendre le moins linéaire que possible et donc diversifier les attaques des armes.

3.3.2 Brice

L'objectif pour la prochaine soutenance sera donc d'implémenté toute les armes proprement, en réimplémentant par exemple l'harpon qui ne nous satisfait pas.

3.4 Sound design

3.4.1 Jean guerin

Il reste encore à trouver quelques bruitages et à les implémenter dans le jeu mais au vu de la quantité déjà trouver je pense être dans les temps vis-à-vis du cahier des charges. Sur les 38 bruitages/musiques annoncés dans le cahier des charges, il n'en reste plus que 8 à trouver.

3.4.2 Brice

Nous espérons d'ici la fin avoir un sound design agréable à écouté et donnant une ambiance dans le jeu pour cela il va nous falloir rajouté de nouveau son et changer quelque son comme les son de pas actuellement répétitif.

3.5 Item

3.5.1 Pierre

Corriger d'éventuel bugs, améliorer la flamme du briquet ainsi que l'affichage de son réservoir.

3.6 Animations

3.6.1 Pierre

Je vais revoir l'animation du harpon qui ne me satisfait pas entièrement, de plus de nouvelles animation seront ajoutées dans le but d'améliorer le combat, plus particulièrement au niveau des armes de corp à corp.

3.7 Brice

Finir d'implémenter les items dans la partie game de manière à ce qu'il soit tous utilisable tel que les cabane de niveau 2 et 3 ou encore le collet.

3.8 Interface

3.8.1 Pierre

Ajout d'un menu paramètre dans le menu pause ou d'accueil, permettant entre autre de régler la luminosité ou le volume du jeu. De plus je porterais à l'étude d'un système de save en JSON, permettant de sauvegarder plus de données en seul coup

3.9 Site web

3.9.1 Pierre

Le seul point sur lequel j'ai du retard par rapport à mes prévisions même si le site est déjà accessible dans un navigateur. Donc je me chargerais de faire les pages de présentation de notre projet et de nous même.

3.9.2 Brice

Pour ma part il va falloir que je finisse d'implémenté la page de Downloads et de rendre le site plus harmonieux dans son ensemble.

3.10 Craft & inventaire

3.10.1 Pierre Brice

Nous porterons une attention particulière à finaliser l'intégration de celui dans le jeu donc à chercher tous les potentiels bug..

3.11 Aspect graphique

3.11.1 Pierre

Implémenter avec Jean d'autre modèles de végétaux, varier en termes d'arbre et de relief les différentes parties de la carte. Nous allons probablement rajouter d'autres points d'eau et rivières, ceci est toujours en cours de débat car nous ne voulons pas que le joueur est trop d'accès à l'eau.

3.11.2 Jean

Comme le laissait sous-entendre le paragraphe sur les graphismes, le gros du travail restant pour la désign de la carte réside dans l'amélioration des textures et la diversification de celles-ci.

4 Conclusion

Pour conclure, notre projet à désormais atteint sa vitesse de croisière, et a passé un réel cap en terme de jouabilité, de gameplay ainsi que de visuel par rapport à la première soutenance. Nous savons désormais travailler en groupe et exploitant au mieux le versioning de fichier intégré à Unity. Pour la soutenance finale nous devrions pas avoir de soucis à boucler notre projet et remplir nos ambitions initial.

Toutefois, nous devons rattraper un léger retard sur notre site internet, même si la page de téléchargement est totalement fonctionnelle. Enfin, par rapport à la première soutenance, nous pouvons dire que travailler efficacement groupe plutôt que faire de notre côté puis tout rassemblez nous a permis d'accélérer l'implémentation de tout ce que nous avions prévu. Cette aventure humaine nous apprend à mesurer l'ampleur d'un tel projet et de répondre au mieux à ses contraintes, de plus elle renforce notre esprit de groupe qui est un atout majeur pour la formation d'ingénieur.