Rapport de Soutenance 1

FQTM

Mathieu Malabard
Maxence Gay
Baptiste Daumard
Téo Le Vern

SOMMAIRE

1 Reprise du cahier des charges

1.1 Résumé du jeu

Pour commencer, notre volonté est de créer un jeu vidéo alliant ingénierie, stratégie et coopération entre les joueurs où l'objectif tout au long de la partie est de coloniser Mars et de fabriquer le "graal" final, que dis-je la consécration... la poutine martienne.
En effet le but de ce jeu est d'avoir une colonie très stable et puissante, résistante aux attaques des martiens, grâce à l'aide d'armes et des nouvelles technologies que nous fabriquerons et qui évolueront tout au long de la partie.

Pour réussir à s'établir sur cette planète inhospitalière les joueurs devront explorer la carte et trouver des filons de ressources à miner pour se fabriquer de quoi se protéger. De plus, les joueurs pourront créer des machines permettant de concevoir des items de plus en plus intéressants afin d'avancer dans le jeu.

Cependant, le joueur ne sera pas seul sur cette planète inhospitalière. En effet, la base du joueur sera régulièrement la cible d'attaques de créatures hostiles attirées par la pollution de l'usine.

1.2 Avancement

Pour le moment, nous avons tous tenu notre planning. Nous nous sommes même rendu compte que certaines tâches du planning étaient bien plus rapide à faire que prévues. C'est pourquoi nous sommes plutôt confiants pour le moment quant à la tenue des délais. Toute fois, il s'avère que d'autres tâches risquent de nous prendre bien trop de temps (exemple: système de transport d'objet) et ne pourraient être effectuées que si nous avons le temps d'en faire une implémentation correcte.
L'avance que nous avons pris est illustrée sur ce schéma:

	Mathieu	Maxence	Baptiste	Téo
Fin Janvier	Apprentissage en autonomie des différents outils (Unity/GitLab/...)			
Février	Action basiques des personnages (Déplacements, intéractions, construction)	Création de prototype d'inventaire	Développement des Sprites "Personnage(s)" et "Terrain"	Mise en place du menu inventaire
	Création des bâtiments	Création des ressources	Création d'une carte prototype et sprites sols	Préparation soutenance et réglages inventaires
Mars	Création des bâtiments et arbre de technologie	Créations et implémentations des armes	implémentions des premiers éléments dans l'environnement (matériaux , sols, ...)	Création des système d'acheminement (drones, trains programmables...)
	Recherches pour premières implémentations multijoueur	Site Web	Implémentation de l'aléatoire dans la création de cartes	Recherches et premières implémentations multijoueurs et Site Web
Avril	Multijoueur	Debugage intensif	Musiques, bruitages et Débugage	Implémentation des ennemis
	Préparation deuxième soutenance et Débugage	Finalisation du site Web	Préparation deuxième soutenance	Débugage

1.3 Changement de planning

Le cahier des charges présentait un planning d'avancement. Cependant, dû à un manque de place lors de la compilation LaTeX, une partie du tableau à disparu. De plus, l'ajout de deux semaines en Juin nous a permi de réorganiser ce planning.
Avec l'avance que nous avons pris sur le multijoueur et la génération procédurale de carte, nous avons libéré beaucoup de temps en Mars. Nous allons donc pouvoir nous concentrer sur le site web. Ci-dessous le nouveau planning

	Mathieu	Maxence	Baptiste	Téo
Mars	Finalisation du multijoueur	Implémentation des outils / MAJ inventaire	Finalisation de la génération procédurale et Implémentation d'un arbre de technologie	Implémentation des outils / MAJ inventaire
	Implémentation des bâtiments	Site Web	Implémentation et design des bâtiments	Site Web
Avril	Implémentation de systèmes électriques	Implémentation des ennemis / Vie du personnage	Implémentation de systèmes électriques	Implémentation des ennemis / Vie du personnage
	Système de fabrication	Finalisation du site Web pour la deuxième soutenance	Développement de plus de bâtiments	Système de gestion de ressources
Mai	Peaufinement de l'arbre de technologie	Intéractions du personnage avec son environnement	Finalisations des graphismes et animations	Intéractions du personnage avec son environnement
	Système de sauvegarde	Musiques, bruitages	Musiques, bruitages	Système de sauvegarde
Juin	Débuggage et perfectionnement du jeu			

2 Présentation des tâches

Jusqu'à présent, chacun a travaillé de son côté sur les différentes implémentations. Dans les semaines à venir, le travail sera plus groupé, nous travaillerons plus en binômes afin d'être plus efficaces dans nos tâches respectives

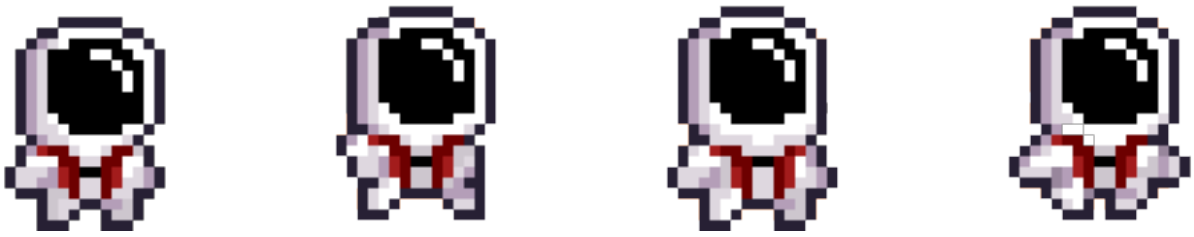
2.1 Implémentation du joueur

Pendant ce début d'année, nous avons pu développer les bases du jeu: le personnage principal que l'utilisateur contrôle.
La partie suivante résume le travail de Baptiste sur la création du personnage.
Les deux parties d'après résument le travail que Mathieu a fait sur les déplacements du personnage et le développement en cours de l'implémentation du multijoueur.
Enfin, la dernière partie résume le travail de Maxence sur la caméra

2.1.1 Sprites Personnages

Le personnage étant l'élément central du jeu, nous avons passé le plus de temps pour son design. Pour cela nous avons repris un design d'un autre personnage libre de droit sur le site OpenGameArt. Nous avons créé notre personnage en modifiant ce modèle mais en gardant sa silhouette.

Son animation de marche est faite en 4 frames (images), elles suivent un principe de base de l'animation: la deuxième frame croise les bras et avance le personnage en avant, la troisième est un retour à la première mais modifiée et la dernière est un petit saut avec les bras écartés. Lorsqu'il ne bouge pas le personnage est dit en "idle", le but de l'animation est de mimer la respiration exagérée, elle reprend la première et troisième frame.
Par la suite nous créeront une animation pour miner.



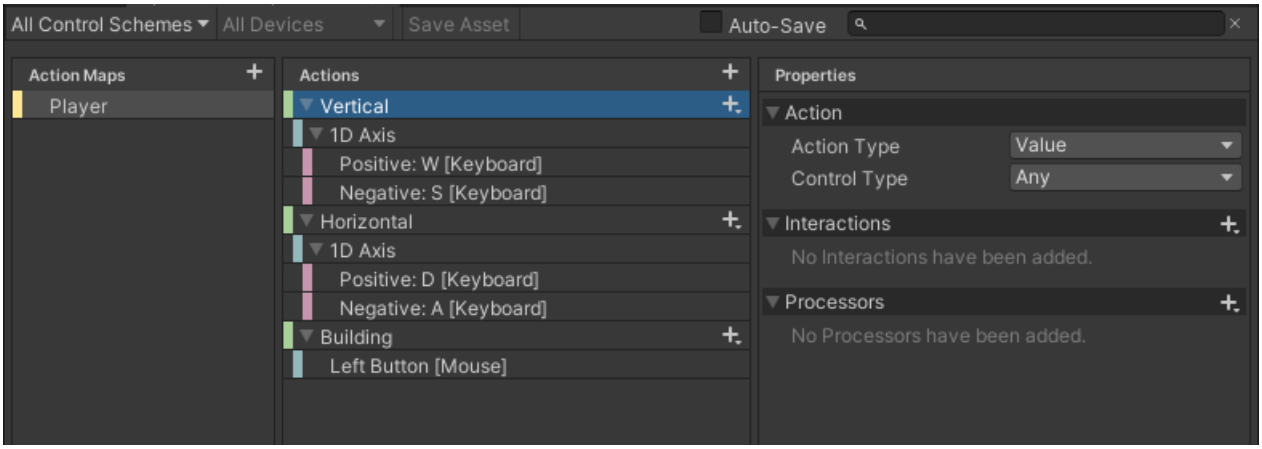
2.1.2 Mouvements

Déplacer un objet sur Unity est très simple mais il existe beaucoup de façons de le faire. En effet, On peut déplacer l'objet grâce à des vecteurs, changer ses coordonnées, utiliser un rigidbody (qui permet surtout de gérer les collisions), ou encore un character controller (un composant de Unity) qui se munit d'un rigidbody et de fonction de déplacements plus complètes qu'un simple vecteur.

Ici nous avons choisi dans un premier temps des déplacements vectoriels afin de pouvoir potentiellement gérer le cas d'une manette munie d'un joystick, permettant un déplacement non limité aux 8 directions basiques. Le fait de ne pas avoir choisi le character controller est due à notre manque d'expérience dans Unity et à notre de choix de système d'Input. En effet, le système d'Input (commandes clavier ou manette) de Unity se sépare en deux, le vieux et le

nouveau système.

Le vieux système d'Input se gére dans le code qui définit les mouvements du personnage alors que le nouveau système lui se gère à partir d'un objet, un InputAction, permettant d'avoir un code moins chargé et beaucoup plus compréhensible. De plus, l'interface de l'InputAction est très simple d'utilisation.
Nous avons donc décidé de nous concentrer sur le nouveau système d'Input plutôt que sur le character controller.



Interface de l'InputAction

L'InputAction se sépare en trois fenêtres:

- Action Maps: Un groupe de commandes qui est utilisé pour séparer les commandes des différents objets. Par exemple, pour l'ajout d'un véhicule, il faudrait créer une nouvelle Action Map pour différencier le joueur du véhicule lorsque le joueur entre dans ce véhicule.
- Actions: Un groupe d'Input pour un même action. Permet aussi de créer des 1DAxis (en rose sur l'interface) pour différencier des valeurs positives et négatives sur un axe.
  Ici, en récupérant les valeurs de l'Input "vertical" dans une variable, celle-ci va s'actualiser à -1 en appuyant sur "s", 1 sur "z" et 0 si aucune touche n'est enfoncée.
  De plus, l'InputAction, ici, a été conçu pour détecter la position de la touche et non pas sa valeur. Ainsi, quand l'utilisateur appuie sur "z" avec un clavier français ou "w" sur un clavier anglais, le résultat est le même.
- Properties: Les paramètres de lecture des Input. Par exemple, si les axes avaient pour paramètre Action type "Button" au lieu de "Value", la variable contenant l'axe prendrait constamment la valeur du dernier Input (-1 ou 1) et ne repasserait plus par 0 lorsque l'utilisateur lâche la touche. Le personnage avancerait donc constamment.

Ici, l'action "Building" ne fais pas grand chose étant donné que les bâtiments ne sont pas encore implémentés, elle ne fait qu'afficher "Built" dans la console lors d'un clique gauche de souris

2.1.3 Multijoueur

Le Multijoueur n'était supposé être implémenté qu'à partir de Mars. Cependant, nous avons vite compris que c'était une très mauvaise idée. En effet, gérer le multijoueur implique de gérer tout ce qui se rapporte au joueur: la caméra, l'inventaire...

Plus tôt le multijoueur sera implémenté, le moins de changement il y aura à faire sur des éléments pré existants. C'est pourquoi nous avons avancé son implémentation à fin février.

Notre sytème de multijoueur se fait avec un Asset nommé Mirror. Il s'agit d'un système

de multijoueur qui reprend les bases du système de Unity en les améliorant grandement. En effet, le système de base de Unity, Unet, est réputé très mauvais.
Mirror simplifie énormément le travail en ajoutant beaucoup de méthodes (tel que l'attribut "isLocalPlayer" qui renvoie true si le joueur concerné est bien le bon client) tout en gardant une syntaxe relativement similaire à Unet, permettant ainsi une migration de l'un à l'autre simple et agréable.

Aujourd'hui, notre système multijoueur est capable d'instancier deux joueurs et plus.
Cependant, des problèmes de compatibilité avec les autres objets sont déjà présents. En effet, la caméra, dont nous spécifierons le développement dans la partie suivante, est censée suivre le joueur.
Lors de l'instanciation d'un joueur, il faut aussi instancier une caméra. Lors de l'instanciation d'un deuxième joueur, il faut instancier une deuxième caméra.
Ce genre de problème de dédoublement va arriver avec chaque nouvelle implémentation et il faudra donc y remédier.

2.1.4 Caméra

Pour commencer, pour la partie caméra, nous avons simplement créé un script qui va suivre le personnage. Pour les entrées nous avons:

- "Target" qui est l'objet que l'on veut suivre
- "Smooth Speed" qui est un float représentant la vitesse à laquelle la caméra suivra le joueur.
- "Offset" est un Vector3 car elle permet de décaler la caméra sur les trois axes.

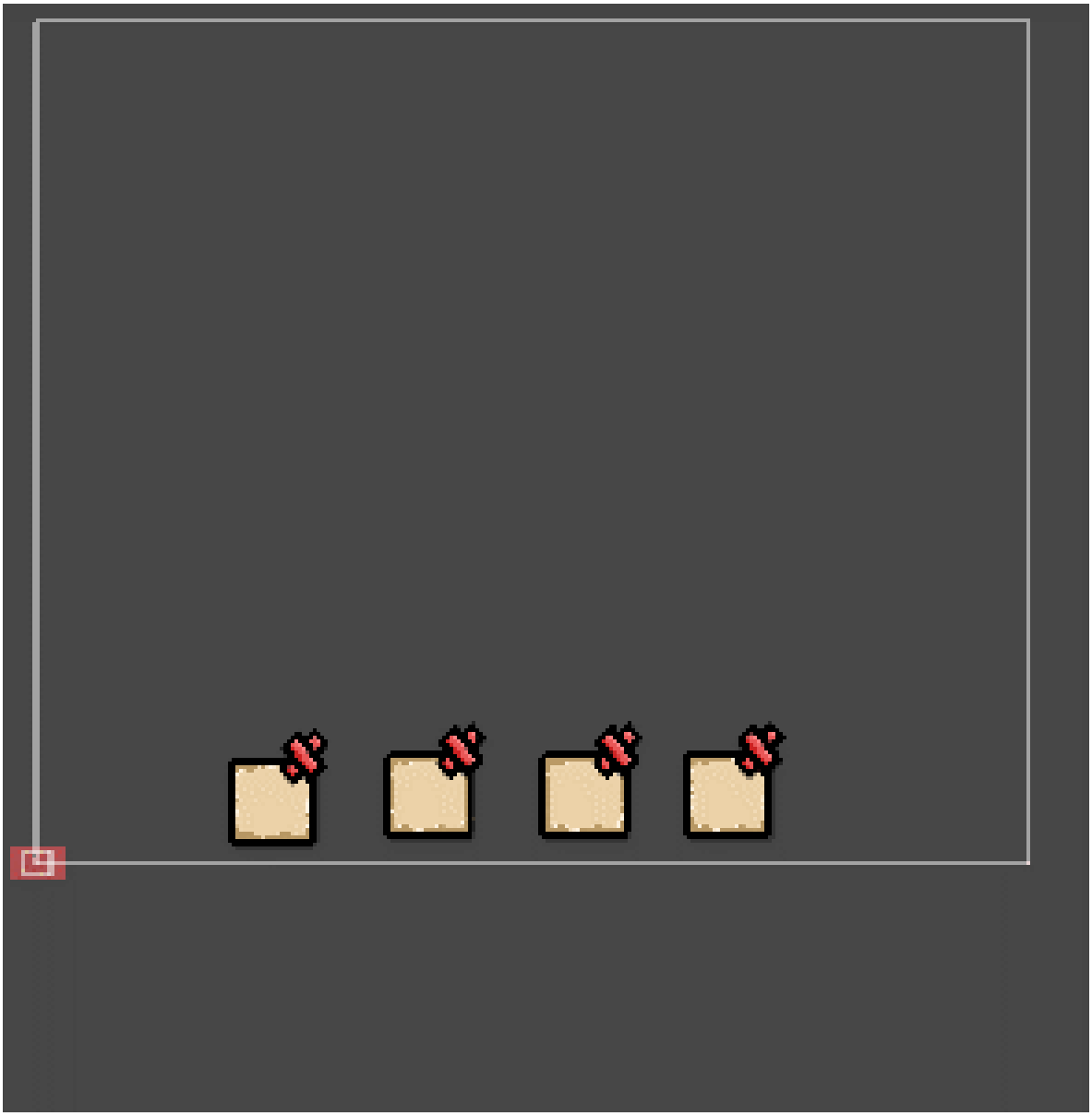
Pour cela nous avons commencé par faire le programme dans une partie nommée "fixed Update". C'est une méthode équivalente à un "Update"(méthode appelée par Unity toutes les x unités de temps) sauf que son appel se fait l'image d'après. Dans notre cas, ce sera juste après le mouvement du joueur que nous voulons bouger la caméra, d'où l'appel à cette fonction.

On commence par définir un "Vector3" qui sera égale à la position du joueur + offset.
Ensuite une variable "desired position" est créée. Elle correspond à la valeur du player. Enfin on crée une variable "smoothed position" qui correspond au Vector3.
De plus, nous utilisons un "Lerp" soit une position linéaire et cela correspond au fait d'aller de façon fluide du point actuel au point "desiredPosition" et avec une vitesse de smooth Speed.

2.2 Inventaire et items

L'inventaire est l'objet central de notre personnage. En effet, sans inventaire, aucun moyen d'accéder aux ressources et donc d'avancer dans le jeu.
Aujourd'hui, nous ne savons toujours pas exactement comment l'inventaire se comportera, soit le joueur aura un inventaire sur lui pour disposer d'objets physiquement, soit l'inventaire sera un inventaire "centralisé" où toutes les ressources créées et utilisées viendront s'ajouter ou seront prélevées directement de l'inventaire et le joueur n'aurait pas à se préoccuper de déplacer ces objets.
Le prototype d'inventaire que nous avons pour le moment correspond au premier et a été développé par Maxence.

2.2.1 Inventaire



Interface de l'UI

Le système d'inventaire marche sur la base suivante : Le joueur doit se positionner sur l'item pour que celui-ci soit récolté. (Il servira dans le cas ou l'on tuera des ennemis, ou encore dans le cas où des joueurs veulent s'échanger des objet en multijoueur).

Pour cela, lorsqu'un joueur intéragit avec un item (objet), il y a une collision entre les deux, dans un premier temps, le programme "Pickup" va lancer la partie "Void OnTriggerEnter2D" du programme qui s'active uniquement dans le cas d'une collision. Ensuite on vérifie si la collision qui opère avec l'item est bien avec un "player" avec "if(other.CompareTag"Player")".

Le programme va par la suite vérifier grâce à une liste prédéfinie si l'inventaire n'est pas plein avec une boucle "for" (qui vérifie s' il y a une place dans la liste). S'il est plein, il ne fera rien, sinon il va stocker l'élément item dans la liste, la case deviendra alors pleine et aucun item ne pourra rentrer dedans.
De plus, il faut créer pour chaque item un "button" qui sera un "prefab" (objet préfabriqué utilisé pour instancier des clones en cours de jeu). Puis on assignera ce "button" à chaque item dans son script "Pickup".

Le joueur quant à lui aura le script "Inventory" qui correspond seulement à la création de deux entrées qui sont les suivantes:

- "IsFull" qui est un booléen qui permet de savoir si le slot (emplacement) est plein ou non.
- "Slot" qui correspond au nombre de case d'inventaire, pour l'instant il n'y en a que quatre mais dans le futur nous ferons un inventaire plus grand avec en plus une possibilité de stocker les items en grand nombre sur un seul slot.

Ce script permettra donc d'assigner de créer les quatre slots qui seront soit plein, soit vide. Pour créer les cases d'inventaire nous avons donc créé des "UI image" qui sont des images qui vont se superposer au-dessus de la caméra (comme pourrait le faire un filtre photo). Ensuite, il faut créer des "UI image" pour les croix qui serviront à lâcher un item.

Pour lâcher un objet à terre, nous avons assigné un bouton sur l'image "croix" de notre UI et lorsque l'utilisateur cliquera dessus, cela détruira l'item dans la liste et créera un clone de l'item sur la carte aux coordonnées du joueur en x et en y+1 pour permettre une meilleure visualisation. Cela donnera donc le rendu suivant.

Interface de l'inventaire et des croix.

Pour faire cela nous avons commencé par faire un "button" avec comme image la croix. Dans les caractéristiques de ce button , nous y avons rajouté le "on click" qui permet au programme de s'activer lorsque l'utilisateur cliquera sur la croix. Lorsque celui-ci cliquera dessus, cela déclenchera une boucle "foreach" qui vérifiera pour chaque élément de la liste, à laquelle l'élément appartient. Quand elle aura trouvé où se situe l'objet, elle le détruira de la liste avec la commande "GameObject.Destroy(child.GameObject)".

2.2.2 Création de nouveaux objets

Les nouveaux objets à créer sont surtout des bâtiments, des armes et des ressources.

Les ressources sont le coeur du jeu, elles serviront à créer des bâtiments et des armes (qui serviront eux-même à créer plus de ressources).
Pour créer une nouvelle ressource, il faudra mettre en place le système de fabrication qui, à partir d'une certaine quantité de certaines ressources directement prélevées de l'inventaire du joueur et d'un temps donné, renvoie au joueur cette ressource dans son inventaire.
Le joueur aura donc une sorte de machine d'assemblage intégrée qui fabrique pour lui automatiquement les ressources demandées qui seront stockées dans une file d'attente.

Pour créer des ressources, le joueur aura surtout accès aux machines d'assemblage qui créeront une recette spécifiée par le joueur. Ainsi, tant que la machine sera alimentée en ressources, elle fabriquera la ressource demandée.

Les armes, quant à elles, seront utilisées par le joueur pour se défendre, lui et son usine, des attaques d'aliens. Le joueur disposera de différentes armes plus ou moins puissantes et efficaces.

2.3 Carte

La carte correspond à l'environnement avec lequel le joueur va pouvoir intéragir pour évoluer. Il est important dans un jeu bac à sable de pouvoir construire librement dessus. Ici, le joueur va surtout utiliser l'environnement pour avancer dans le jeu.
Nous voulions donner au joueur l'opportunité de pouvoir recommencer des parties sans avoir la même carte. C'est pourquoi nous avons opté pour une génération procédurale de la carte, développée par Baptiste

2.3.1 Méthode de génération procédurale

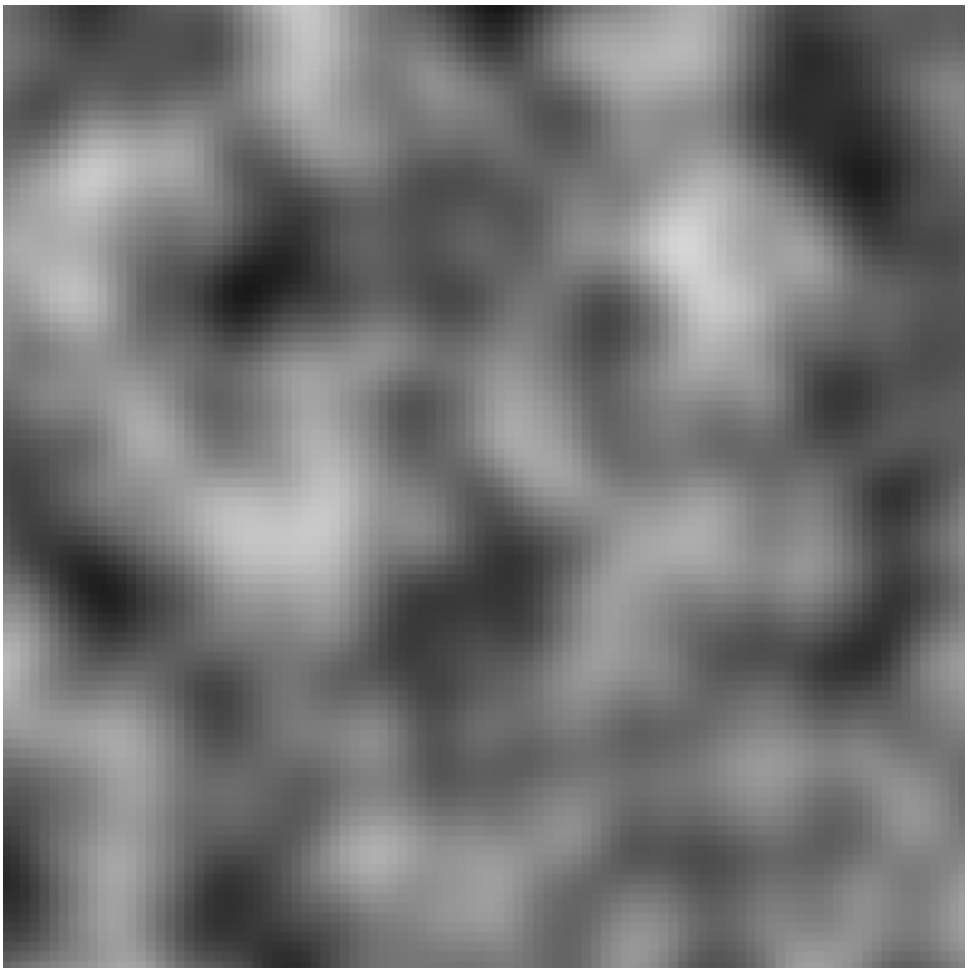
Pour la génération procédurale de ressources sur la carte du jeu, nous avons utilisé un algorithme Perlin Noise. Celui-ci nous permet de générer un carte de dimension (x,y) pseudo-aléatoirement remplie des valeurs entre 0 et 1.
C'est la méthode la plus utilisée dans les jeux vidéos, de plus le Perlin Noise est géré par Unity. Mais le rendu n'est pas très naturel, pour cela on empile plusieurs cartes pour un meilleur rendu.

Chaque couche est appelée "Wave", chaque vague possède les propriétés suivantes: Seed (elle nous permet de compenser le bruit pour ne pas échantillonner la même zone), de Fréquence (le nombre de pic de 1 sur la carte donc plus le la fréquence est haute plus la carte est chaotique) et d'Amplitude (la taille d'une "tache") . Pour cela nous avons créé trois scripts:

- Le premier pour créer les différentes couches de nos cartes , dedans nous créons la nouvelle classe Wave.
- Le second nous permet de créer un objet des Ressources en parcourant le tableau regroupant un tableau des sprites(image) et en regardant si la ressource choisie correspond aux conditions de génération de la NoiseMap.
- Enfin un gros script, héritant des deux premiers. Il génère premièrement trois NoiseMap: une pour l'humidité, une de chaleur et une de hauteur. Leurs noms ont peu d'importance dans la génération actuelle, mais dans le futur développement de l'environnement ils auront de l'importance pour la cohérence.

Ensuite il parcourt chaque coordonnées de notre NoiseMap, à chaque fois, une fonction va parcourir une liste avec toutes les ressources, et elle va renvoyer une autre liste avec toutes les ressources qui peuvent correspondre à la ressource à afficher en fonction des trois NoiseMap générées, elles-même composées de plusieurs Wave. Pour cela elle va regarder si les trois paramètres: la moisissure, la hauteur et la chaleur de la case sont supérieures ou égales aux paramètres de la ressource testée.

Une fois la liste construite avec toutes les ressources susceptibles de correspondre à la ressource à retourner, nous allons parcourir cette même liste pour déterminer celle qui est susceptible de correspondre au mieux à la ressource à afficher, pour cela on est réalise une différence entre les paramètres de la case et les paramètres minimums. On additionne ensuite les trois différences et on le garde dans une variable, la ressource qui sera retournée est celle avec la plus petite variable.



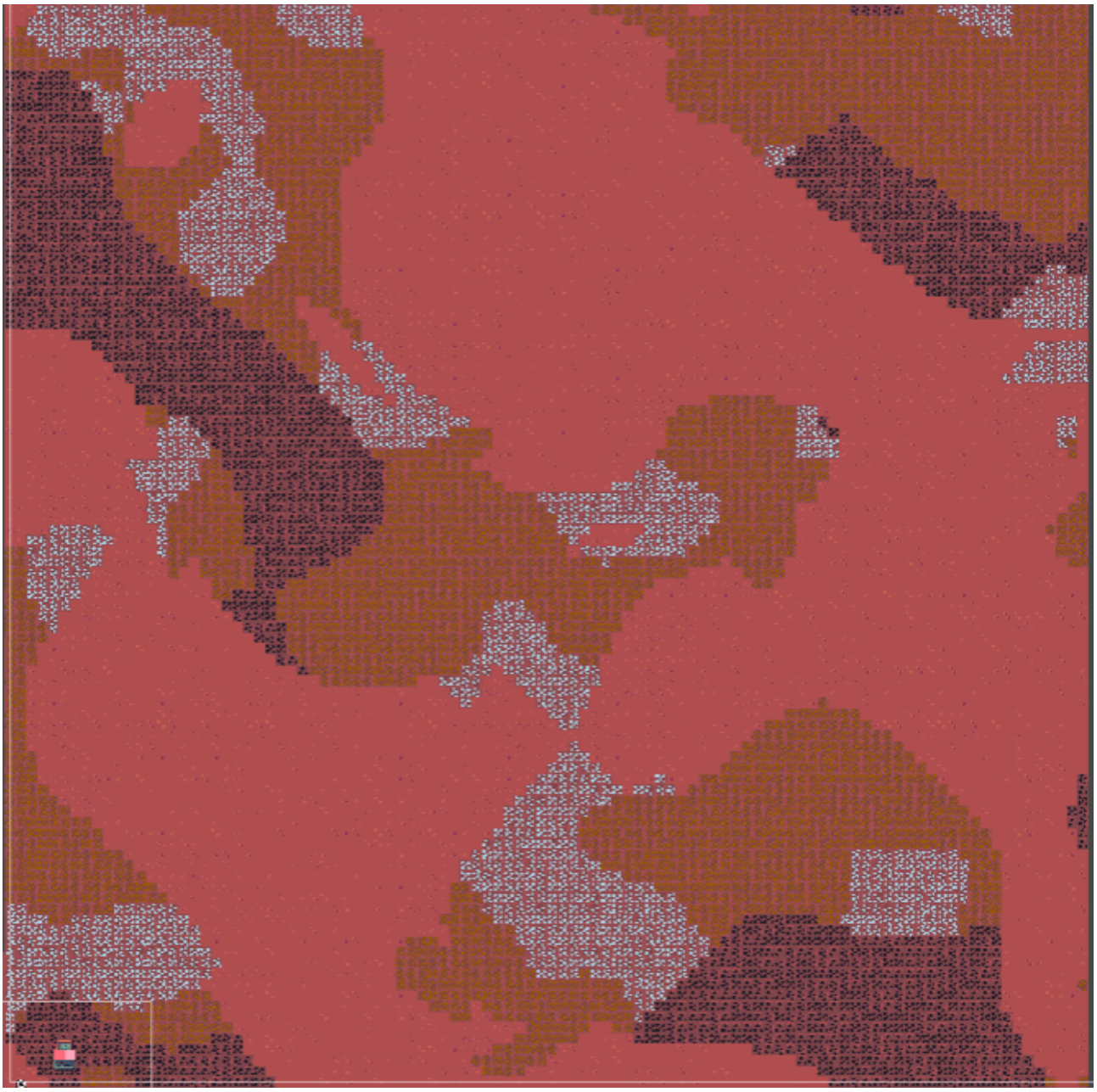
Exemple de NoiseMap avec plusieurs 'Wave'

2.3.2 Présentation de l'environnement

L'environnement de notre jeu comportera différents terrains. Un sol basique ainsi que différentes ressources telles que du charbon, du fer et du cuivre. D'autres ressources pourront être ajoutées dans le futur.
Les différentes "tile" (cases) de la carte sont stockées sur une seule image que nous avons découpé sur Unity.

Les différentes sprites du sol de base ont été reprises sur une bibliothèque libre de droit sur OpenGameArt. Le style graphique correspondait parfaitement au style du jeu, cela nous a permis de gagner du temps.
Chaque sol et ressource possèdent différentes sprite pour créer des nuances. Comme vous pouvez le voir sur la figure ci-dessous la génération de la carte présente de grandes taches collées les une aux autres et parfois trop grandes.

L'objectif est de rendre l'environnement plus naturel en faisant en sorte que les parterres de ressources soient plus petits et le sols basique plus présent. Pour cela il nous faut ajuster les paramètres (d'amplitude de Seed et de frequency) des NoiseMap afin que les probabilités avantagent le sol basique.
Par la suite nous devrons travailler sur la collecte de ressources, elles pourront être collectées par le joueur à l'aide d'une pioche, ou à l'aide d'une foreuse.



Carte générée procéduralement de 100x100

2.4 Implémentation des bâtiments

Il nous reste pour la fin du développement à créer l'une des mécaniques les plus importantes, Les Bâtiments
Ceux-ci seront stockés dans un inventaire et ils pourront être placés par le joueur sur la carte. Seul un bâtiment sera présent dès le début du jeu, la base de départ. Elle servira de stockage, de four et de générateur d'électricité et pourra être améliorée.

Dans les bâtiments que nous comptons implémenter, il y a : des foreuses, des coffres de stockages, des transformateurs de ressources, des machines d'assemblage, des générateurs électriques (et potentiellement d'oxygène)...
Les bâtiments nécessiteront de l'électricité pour fonctionner.
Ils se placeront selon une grille, les foreuses ne pourront être placées que sur des ressources.
De plus, pour les foreuses, il faudra ajouter une variable représentant la quantité restante

de ressources dans chaque case. Ainsi, le joueur devra explorer la carte pour renouveler ses ressources.
Certains bâtiments pourront être débloqués via l'arbre de technologie.
Pour fabriquer des bâtiments, le joueur aura accès à un menu de fabrication dans son inventaire. Les fabrications se feront automatiquement si le joueur a les ressources nécessaires et le bâtiment se placera dans l'inventaire quand la fabrication sera terminée.

2.5 Menus

Les menus ont été réalisés par Téo. Il s'agit de la première image sur laquelle le joueur arrivera en lançant le jeu. Il faut donc qu'elle soit intuitive et simple d'accès

2.5.1 Menu général

Le menu "général" est la première interface sur laquelle l'utilisateur arrivera une fois le jeu lancé. Depuis celle-ci, il pourra accéder à un menu intermédiaire pour lancer une partie en solitaire ou lancer une partie en multijoueur. Il pourra aussi accéder au menu pour configurer les divers paramètres du jeu et enfin fermer l'application. Actuellement le design du menu est temporaire (voir Fig.1).

Fig.1 Menu Principal

Le menu principal est implémenté sur une scène indépendante du jeu, et n'est composé que d'objes d'UI (boutons, panel...) précédemment évoqués dans la partie de l'inventaire.
Tout d'abord, il y a un panel qui sert de fond.
Sur ce panel sont placés tous les menus et leurs UI correspondants (actuellement le système de sauvegarde de partie et le mode multijoueur n'ayant pas été réalisés les boutons Singleplayer et Multiplayer chargent juste la scène correspondante à la partie actuelle grâce à un script dédié). Ultérieurement, ils dirigeront vers des menus intermédiaires, dont un pour rejoindre une partie dans le cas du bouton multijoueur et un autre pour charger une partie ou en créer une nouvelle pour le bouton Singleplayer.

Chaque texte est un objet TextMeshPro (asset qui améliore les textes de bases de Unity) dans un objet bouton. Tous les boutons ont quelques caractéristiques communes notamment le fait que lorsque la souris les survole, le fond devient plus foncé et encore plus lorsque qu'on click dessus (voir le bouton Back sur Fig2).

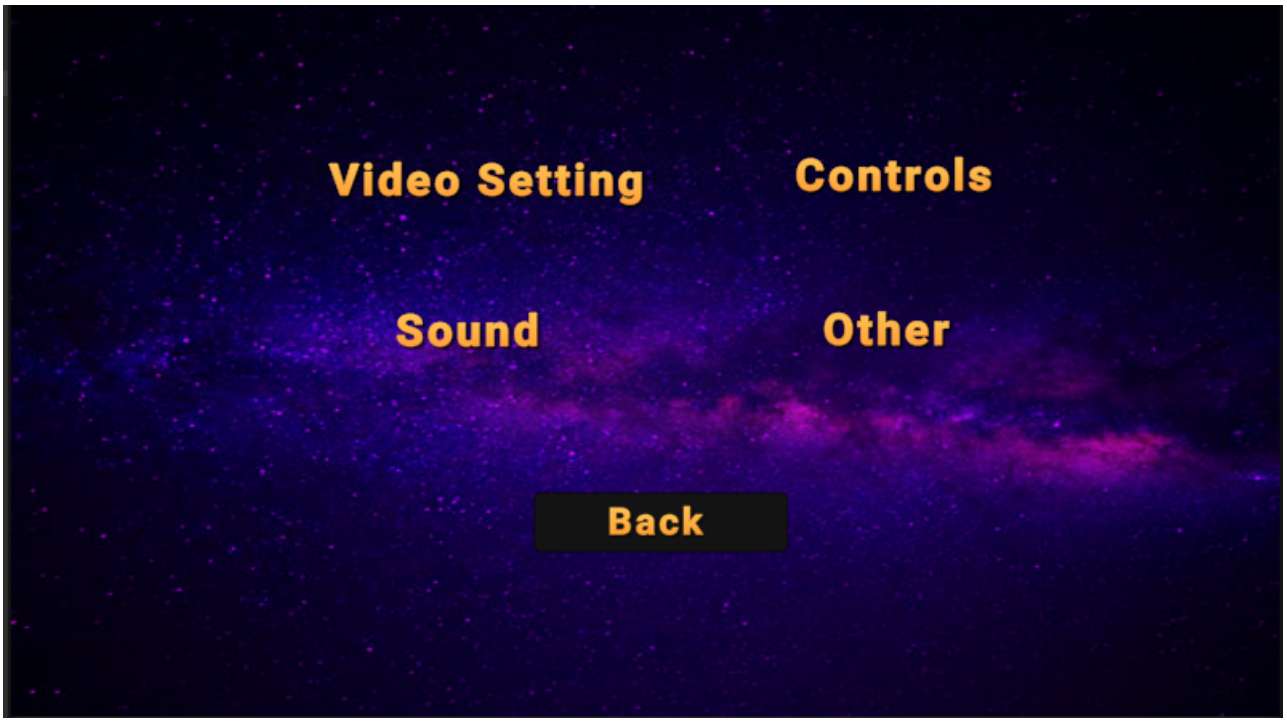


Fig.2 Menu de configuration

Comme tous les menus fonctionnent sur les mêmes principes, je vais expliquer en détail le menu d'option. Lorsqu'on sélectionne le bouton "Options" sur le menu principal (Fig.1) l'UI de celui-ci va être désactivé (en modifiant la propriété GameObject.SetActive à false) tandis que l'UI du menu "Options" (Fig.2) va lui s'afficher (GameObject.SetActive à true). Il n'y a aucun changement de scène.

Nous souhaitons laisser un maximum de liberté et de configurations possibles à l'utilisateur. Pour cela, nous aimerions implémenter diverses options telles que :

- Sound: réglages son général et par section (volume monstres, machines, environnement ...)
- Vidéo Setting: Mode fenêtré, plein écran ...
- Controls : Pouvoir ré-attribuer les touches aux actions (modification du système d'input)
- Other : Un menu libre où l'on pourra ajouter des idées au cours du projet

2.5.2 Menu du joueur

Les menus présentés précédemment sont plutôt propre au jeu en général, avec la configuration de l'environnement du jeu. Cependant, il y a aussi des menus qui sont présents dans la partie directement.

L'interface de gestion des ressources, permettra de tenir informé le joueur sur la quantité qu'il possède de chaque ressource. Il y aura aussi un arbre de technologie où le joueur pourra débloquer au fil de sa progression diverses améliorations, de nouveaux bâtiments... L'aventure du joueur se fera donc étape par étape, restreinte par les branches non débloqués de l'arbre. Le menu où le joueur va pouvoir créer de nouveaux bâtiments pour améliorer sa base (cela sera

sûrement un système de type catalogue). Et enfin, un menu "échap" qui permettra d'accéder au menu de configuration, sauvegarde et quitter. Contrairement au mode multijoueur, en solitaire ce menu mettra le jeu en pause.

2.6 Implémentation de l'IA

Comme dans la plupart des jeux, il y aura des entités qui seront les ennemis du joueur et qui auront pour but de ralentir sa progression voire de faire perdre la partie si le joueur ne réagit pas. Notre jeu se déroulant sur une planète autre que la Terre, il y aura des aliens pour incarner le rôle des monstres hostiles.

Ces monstres apparaîtront sous formes de vagues qui seront déclenchées selon certains facteurs. Il est évident que plus le joueur progressera dans le jeu, plus ses bâtiments défensifs et ses armes seront puissants donc la difficulté des vagues d'attaque sera augmentée. Cela se traduit par l'augmentation des points de vie de chaque monstres, de leurs dégâts et de leurs nombres.

Nous souhaitons implémenter différents types de monstres qui auront chacun un comportement spécifique:

- Basique: Monstre avec des attributs moyens qui ciblera le bâtiment le plus proche
- Rapide: Monstre avec peu de points de vie mais avec une vitesse de déplacement élevée
- Mastodonte : Peu de dégât, beaucoup de points de vie, lent, cible les défenses
- Kamikaze : Très fragile, gros dégâts (explose), cible les bâtiments défensifs
- Boss : Monstre final qui permettra de mettre fin à la partie

Et chaque type de monstre (sauf le boss) aura une version "mutante" où les attributs de celui ci seront augmentés. Bien sûr, le nombre et la variété des monstres dépendront du temps que nous aurons.

3 Réalisation

3.1 Problèmes rencontrés

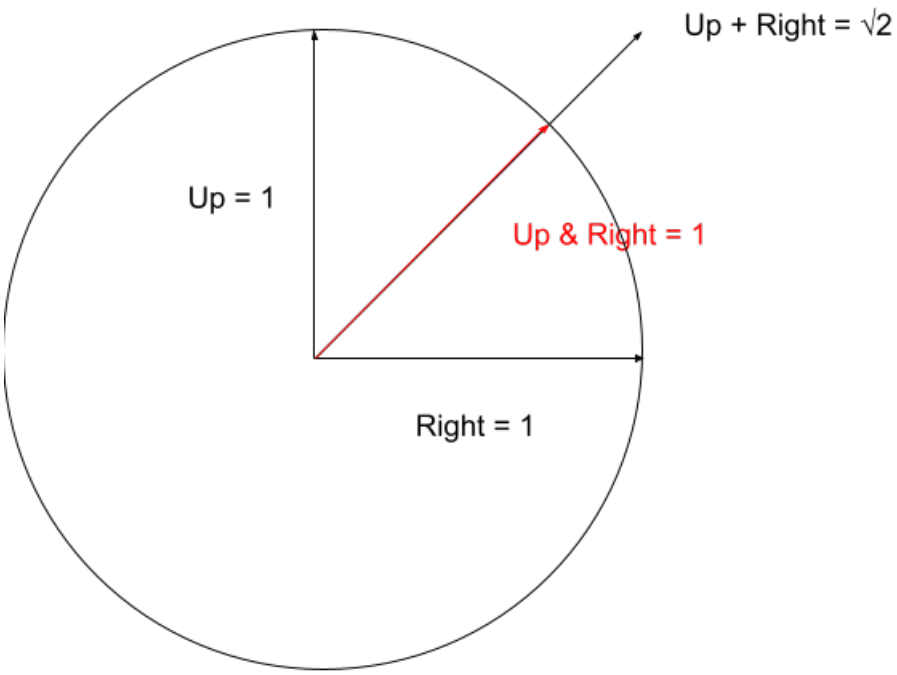
Le dépôt git nous a posé beaucoup de problèmes au début du projet. Des fichiers se créaient à chaque fois que Unity se lançait malgré le .gitignore, des conflits apparaissaient à chaque push/pull...

Il nous a fallu du temps pour résoudre ce problème car non seulement nous ne pouvions pas commencer le projet convenablement, mais en plus, du fait de notre inexpérience avec git, nous ne pouvions pas partager le travail correctement avec le groupe.
Finalement le problème devait venir du fait que le ".gitignore" a été ajouté après le dossier de Unity. Il a suffit de supprimer les dossiers cachés du dépôts pour que les fichiers gênants ne soient plus considérés par git.

Un problème basique rencontré a été le déplacement du joueur. Celui ci se fait par vecteur en deux dimensions. Le script récupère une valeur x et y pour le vecteur (1 ou -1) ce qui donne soit le vecteur $\overrightarrow{0}$, soit le vecteur (0,1) soit (1,0) soit (1,1) (sans compter ceux avec des valeurs

négatives).

Ce vecteur représente le mouvement du personnage. Pour avoir la vitesse du personnage, on multiplie la valeur du vecteur par un nombre. Le problème ici se pose quant au vecteur (1,1).
En effet, sa valeur est plus grande que les vecteurs (0,1) et (1,0) ce qui fait que le personnage se déplace plus rapidement en diagonale.



Il nous fallait donc un vecteur dont la norme est constamment 1. Heureusement Unity possède une méthode qui fait cela: "Vector2.normalize", une méthode qui garde l'orientation du vecteur tout en normalisant sa norme à 1, nous permettant de rendre le code plus clair.

La génération procédurale de carte n'est pas tout à fait terminée. En effet, elle n'est pas aléatoire, lors du lancement du jeu, le jeu calcule et génère cette carte, mais la même à chaque fois. Nous somme actuellement en train de corriger ce problème.

Un problème majeur mais qui ne nécessite que plus de documentation sur Mirror est le fait que, le composant qui se charge de créer et gérer le serveur de jeu, le NetworkManager, n'instancie qu'un joueur et rien d'autre. Or, la caméra n'est pas un fils du joueur, c'est un objet à part qui prend comme cible le joueur.
Pour le moment, nous avons défini la caméra comme objet fils du joueur afin d'avoir une caméra qui suit le joueur mais un objet fils d'un autre partage ses coordonnées ce qui fait que notre caméra n'a plus cet effet de décalage.
Pour résoudre ce problème, il faudra se documenter un peu plus en profondeur sur Mirror et sur le networking.

3.2 Ressenti global sur le projet

Ce projet est sans doute celui qui pour nous quatre, va nous prendre le plus de temps et qui va nous demander une rigueur assez constante pour mener à bien nos objectifs. Nous avons su comprendre cela rapidement et nous avons commencé dès le début à nous documenter énormément sur les différentes plateformes que nous allions utiliser par la suite.

Malgré un début où nous étions très efficace, la période de partiels nous a fortement ralenti et nous avons baissé le rythme de travail pendant 1 à 2 semaines. Mais, juste avant les vacances, nous avons réussi à nous remettre dans le projet et énormément avancer de grandes parties du jeu telles que le multijoueur qui n'était pas prévu avant Mars, l'inventaire qui était assez complexe ou encore la génération de carte aléatoire qui ne devait être implantée que fin Mars.

Bien que nous n'en soyons encore qu'au début de notre jeu, nous avons acquis de multiples connaissances sur la création de jeux vidéo mais également sur la gestion du temps et du travail en équipe.
Ce projet nous demande un travail régulier et rigoureux mais c'est beaucoup de fierté qui en découle, tant au niveau personnel qu'au niveau collectif.

Nous sommes assez confiants sur la suite du projet. En effet, nous avons bien avancé le projet car nous pensions que cela nous demanderait bien plus de travail. Cependant, nous allons rentrer dans la partie complexe du projet avec des parties spécifiques à notre jeu où les tutoriels Unity sur Youtube ne suffiront plus.

4 Conclusion

Durant ces 2 mois, nous avons déjà réussi à implémenter les bases du jeu:

- Un personnage qui se déplace avec son animation
- Une caméra qui suit ses mouvements de manière fluide
- Une carte pseudo-aléatoire avec la génération de minerai
- Un inventaire fonctionnel et pratique
- Un multijoueur pseudo-fonctionnel
- Design des ressources de bases et du personnage

Cela peut sembler assez léger mais nous avons dû, pendant quelques semaines, nous acclimater avec les différentes plateformes à notre disposition telles que GitKraken, Unity...
Mais, il nous reste encore de nombreuses tâches à accomplir avant que notre jeu puisse ressembler à ce que nous souhaitons. Ces tâches sont les suivantes :

- L'implémentations des divers bâtiments/items
- La création du Site Web
- L'implémentations des ennemis (IA)
- La création de l'arbre des technologies
- La mise en place des bruitages et des musiques
- Finalisation du multijoueur et de l'inventaire

Avec ce nouveau planning ajusté, nous repartons avec de meilleures bases pour cette deuxième période de projet. A la fin de cette période, nous devrions avoir fini l'implémentation générale des bâtiments, des ennemies, des armes et de l'arbre de technologie. A partir de cela, le jeu devrait être presque opérationnel dans le sens où il sera possible de "jouer" au jeu puisque beaucoup d'éléments de gameplay auront été implantés. Le jeu sera donc plus intéressant à présenter.

Comme nous disions dans le cahier des charges en citant Oscar Wilde, nous visions la Lune pour atterrir au moins dans les étoiles.
Maintenant visons Mars pour peut-être atterrir sur la Lune.