

# Projet Engi'Blocks

Rapport de dernière soutenance

\_Groupe Buildingee\_

Antoine Adam  
Alexandre Dugot  
Mohamed Ghaffour  
Nathan Labernardiere

23 juin 2021



# Table des matières

## 1 Introduction

## 2 Cahier des charges

## 3 Présentation et origine du groupe

## 4 Déroulement des soutenances

## 5 Avancements individuels sur les fonctionnalités

5.1	Personnages . . . . .	9
5.2	Gestion/design planètes . . . . .	15
5.3	Gestion du vaisseau et design . . . . .	18
5.4	Site Internet . . . . .	20
5.5	Menu principal . . . . .	23
5.6	Multijoueur . . . . .	26
5.7	Sons / effets sonores . . . . .	26
5.8	Interface carte . . . . .	28
5.9	Cinématiques . . . . .	32
5.10	Missions et histoire . . . . .	33
5.11	Gestion des sauvegardes . . . . .	35
5.12	Machines / inventions . . . . .	37

## 6 Comment jouer au jeu

## 7 Potentiels Retards

## 8 Structure du répertoire

## 9 Ressentis personnels

## 10 Conclusion

## 1 Introduction

Après un long moment pour réaliser notre jeu Engi'Blocks codé sur la plateforme Unity, le jeu est enfin terminé et prêt à être testé. La dernière fois, nous avions souligné qu'il restait de nombreuses choses à implémenter mais surtout qu'il était important de relier les diverses parties du projet entre elles, ce qui a désormais été fait.

Le vaisseau qui précédemment paraissait vide mais surtout très peu accueillant a été rempli de diverses machines fonctionnelles mais aussi de nombreux éléments décoratifs le rendant plus attrayant visuellement et surtout bien plus dans le relié au thème du jeu qui pour rappel se déroule dans l'espace à travers la galaxie.

Avant de donner un résumé global du concept de notre jeu, il est intéressant de notifier que certaines ayant été codées de telle manière ont été recodé pour certaines pour des raisons d'optimisation mais aussi de lisibilité de code.

### Résumé du jeu :

Comme dit précédemment, Engi'Blocks est un jeu se déroulant dans l'espace où le joueur qui incarne un astronaute va évoluer dans un vaisseau ayant pour but de retrouver la terre après s'être perdu. Ce dernier devra franchir plusieurs niveaux caractérisés par des missions à accomplir, des monstres à vaincre, des ressources devant être récoltées et pour finir l'amélioration de ses compétences dont son arme de combat. Tout cela étant rendu plus difficile grâce à un système de probabilité visant à ce que le joueur ne puisse pas obtenir toutes les ressources et armes les plus rares du premier coup. Pour finir la fin du jeu a été faite de façon à ce que le joueur ressente de la difficulté de part l'implémentation d'un boss final à combattre.

## 2 Cahier des charges

Le groupe n'a en rien changé en termes de membre, nous sommes toujours 4 :

- Alexandre DUGOT (chef de projet)
- Antoine ADAM
- Mohamed GHAFFOUR
- Nathan LABERNARDIERE

En termes de répartition des tâches, il n'y a pas eu de changements notables, mais nous pensons qu'il est important d'en faire un bref rappel.

Cela donne donc actuellement en termes de répartition des tâches le résultat ci-dessous :

Tâches	Antoine	Nathan	Alexandre	Mohamed
Sauvegardes	x		o	
Missions / histoires			o	x
Structure du vaisseau	o	x		
Machines / inventions	x			o
Personnages		o	x	
Gestion / design planètes		x	o	
Site Internet		o		x
Effets sonores		x		o
Cinématiques	o			x
Interface carte	o		x	
Multijoueur	x		o	

x : responsable      o : suppléant

### **3 Présentation et origine du groupe**

#### Origine du groupe BuildIngee :

Au début le groupe n'était formé que de Mohamed , puis ce dernier a proposé à Antoine et Alexandre de rejoindre le groupe car nous étions tous les trois dans la même classe et que nous avions déjà commencé à faire connaissance malgré un début d'année encore bien proche à l'époque. Par la suite Nathan a proposé à Nathan qui lui venait d'une autre classe de rejoindre le groupe. À part Mohamed nous ne connaissions pas Nathan mais rapidement nous avons passé outre toute timidité et avons commencé à travailler de manière soudée ensemble. Pour finir, ce groupe a également été formé car chacun d'entre nous avait sa propre spécialité.

#### Présentation des membres du groupe :

##### Alexandre Dugot :

Je suis quelqu'un qui de base est intéressé surtout par ce qui est en rapport avec l'intelligence artificielle. Je pense que nous avons tous vu ces vidéos de robots "intelligents" qui réussissent à interagir avec l'homme de façon étonnante. Bien qu'ici nous soyons dans le domaine des jeux vidéos , je dois dire que le jeu vidéo est une passion pour moi depuis longtemps, j'y ai passé de nombreuses heures et créer un jeu par soi même est encore quelque chose de plus exceptionnel car nous pouvons exprimer notre créativité et y implémenter des choses que nous regrettons de ne pas avoir dans les jeux auxquels on joue d'habitude. Mon choix de m'occuper des personnages est lié aux IA, je trouve cela génial de pouvoir coder des entités qui vont bouger seules avec leur propre comportement. Pour finir j'espère que vous prendrez tous du plaisir à jouer à notre jeu , bon jeu à vous !

Nathan Labernardiere :

Pouvoir créer un jeu soi-même est un rêve pour de nombreuses personnes et cela m'est possible dès la première année à EPITA. Avant de rentrer dans cette école, mes connaissances en informatique étaient très limitées et le fait qu'EPITA soit une école en informatique ouverte à tous m'a beaucoup intéressé. Mes connaissances pour débuter ce projet ne se limitaient qu'à ce que l'on avait appris lors de nos TP de programmation et je n'avais aucune idée de comment concevoir un jeu. Cette expérience était donc une chose complètement nouvelle et c'est ainsi avec joie que je me suis lancé dans cette aventure folle. Cela me permettra d'accroître considérablement mes connaissances liées à ce domaine. Par conséquent, ce projet est un excellent moyen d'apprendre tout en faisant quelque chose qui nous tient à cœur. Je suis ainsi responsable du style graphique de notre jeu ainsi que des aspects sonores de celui-ci.

Mohamed Ghaffour :

Concevoir un jeu vidéo constitue pour moi un véritable challenge étant donné le fait que je n'ai jamais eu d'initiation en la matière avant mon entrée à EPITA. Depuis toujours ma curiosité me pousse à me demander comment se crée un jeu vidéo. Avec ce projet je vais pouvoir enfin le découvrir grâce au développement de notre propre jeu ! Durant mon enfance, les jeux vidéo avaient une part importante, j'ai de nombreuses idées concernant ce projet qu'il faut maintenant développer en acquérant des connaissances à l'aide des nombreuses informations présentées sur Internet ainsi que des connaissances déjà acquises pendant le premier semestre.

Antoine Adam :

Avant de commencer ce projet, j'avais déjà créé quelques petits jeux sans moteur de jeu et en apprenant Unity, je suis aperçu de l'approche multicouche des objets, il était à la fois très simple de les paramétriser avec l'interface sans écrire la moindre ligne de code et de régler le moindre petit paramètre en C# de façon dynamique. Un autre aspect qui m'a impressionné, c'est quand je vois comment ils ont fait pour répondre à une problématique que je me suis posée avant d'utiliser un moteur de jeu. Ce projet est vraiment enrichissant.

## 4 Déroulement des soutenances

### Première soutenance :

Lors de la première soutenance nous étions arrivés avec une version encore très pauvre en termes de détail. Nous avions toutes les idées dans nos têtes par rapport à ce que nous voulions faire mais à cette époque le jeu n'était encore que le squelette de ce qu'il est aujourd'hui. Lors de cette soutenance nous avions néanmoins déjà un jeu jouable avec un personnage qui se déplaçait.

Le site internet de son côté avait déjà été créé même s'il lui manquait encore bien des éléments en raison d'un projet pas encore terminé, par conséquent nous ne pouvions pas encore mettre le téléchargement du jeu ainsi que des captures d'écran de ce dernier.

Pour finir, c'est à ce moment là que nous avons commencé à entrevoir des problèmes sur la façon d'implémenter les sauvegardes mais nous nous sommes également posé la question de l'implémentation d'un multijoueur ou non.

### Deuxième soutenance :

Au moment de passer pour la seconde soutenance, nous étions bien satisfait que lors de la première car notre jeu qui n'était qu'un squelette de code sans réel visibilité graphique sur le jeu est devenu avec plusieurs fonctionnalités nouvelles comme la carte de l'espace ou encore la gestion des sauvegardes mais aussi un aspect graphique bien plus accueillant et agréable à regarder.

Il est également important de préciser que lors de l'avancé de la première soutenance , nous étions tous collés aux divers tutoriels ou à la documentation pour pouvoir ne serait-ce qu'avancer un petit peu mais à cette période nous étions réellement satisfait car nous avons enfin après acquisition de nombreuses connaissances pu coder tout ce que nous voulions de manière plus libre sans être attachés à des modèles pour nous aider.

Malheureusement nous avons également eu quelques problèmes comme des problèmes de collision avec les personnages ou encore des problèmes sur le projet avec git de part des push qui ont créé des conflits et corrompu le projet nous obligeant donc à revenir à une ancienne version de ce dernier. C'est également ici que nous avons commencé à prendre du retard sur certaines parties comme les missions et les cinématiques.

Pour finir, cette deuxième soutenance fut un réel soulagement car nous avions enfin la quasi certitude que notre jeu sera terminé en temps et en heures de part nos capacités nouvellement acquises. Il est également à noter qu'à ce moment-là nous avons réellement pris la décision d'écartier le multijoueur du projet après que nous soyons passé à l'oral.

### Dernière soutenance :

Pour la dernière soutenance nous ne pouvons nous avancer quant à la façon dont elle va se dérouler mais nous sommes content car nous allons enfin pouvoir le jeu en lui-même et non pas en parler sans avoir le support en question pour montrer réellement ce qu'il en est sur le jeu. Nous pourrons également montrer le site internet totalement terminé avec l'interface d'installation.

## 5 Avancements individuels sur les fonctionnalités

### 5.1 Personnages :

Après 3 soutenances, le personnage et toutes ses fonctionnalités sont enfin opérationnelles.

Maintenant, nous allons faire un résumé dans l'ordre chronologique de toutes les réalisations de cette tâche depuis la 1ère soutenance à la dernière. Pour rappel le responsable de cette tâche est Alexandre Dugot accompagné de son suppléant Nathan qui s'est occupé de tous les aspects graphiques , permettant de réaliser diverses animations.

#### 1ère soutenance : (Mouvements et barre de vie)

Dans cette première partie nous parlerons de l'implémentation de l'astronaute qu'incarne le joueur ainsi que du code le faisant se déplacer mais aussi la gestion de sa vie grâce à une barre de vie.

Nous avons choisi de dessiner un petit astronaute comme personnage. Le design de ce personnage est assez classique et peut ressembler à des designs assez récurrents sur Internet. Pour animer notre personnage, il nous a fallu réaliser cinq positions différentes. Quatre de ces cinq positions correspondent au déplacement de l'astronaute sur le sol et le cinquième correspond à l'animation du saut. Ces animations sont plutôt basiques mais elles permettent quand même à notre protagoniste de se déplacer de façon fluide. Pour le moment, nous avons réalisé deux personnages, la seule différence entre eux réside dans leur colorimétrie.

Voici le rendu obtenu :

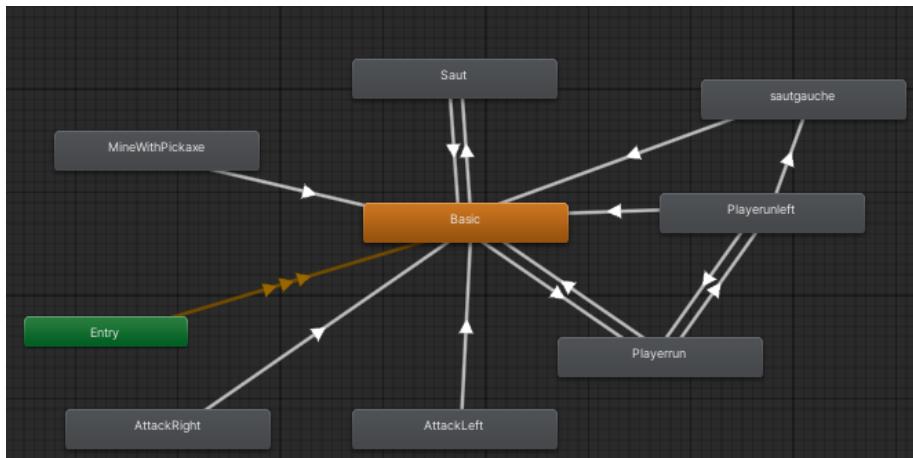


Vous voyez donc ici deux exemples d'animation qui ont été implémentés et qui sont exécutés en fonction de l'action du personnage. Pour l'animation, cela consiste à faire défiler une suite d'images très rapidement afin de pouvoir créer l'illusion d'un personnage qui court ou qui saute.

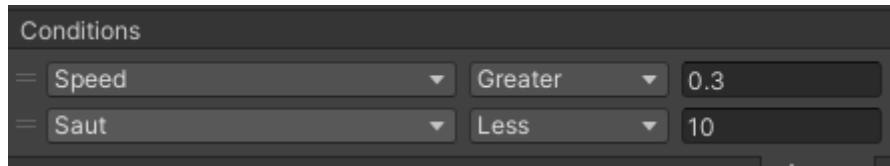
Premièrement, passons en revue la partie déplacement et saut du personnage ainsi que leur animation. Pour déplacer le joueur vers la droite et vers la gauche , nous avons défini dans le fichier MOVE.cs est une variable de type float nommé “horizontalMovement” qui définit la force du mouvement voulue , variable que nous passons ensuite dans un Vector3 qui permet de définir une nouvelle position au joueur sur l'axe X en augmentant sa coordonnée X de façon positive si la flèche droite est pressée et de façon négative si la flèche gauche est pressée.

```
Vector3 targetVelocity = new Vector2(_horizontalMovement, rb.velocity.y);
```

Pour le moment nous n'avons parlé que du déplacements dans le sens de modifier les coordonnées du joueur mais parlons maintenant des animations et de la façon dont elles sont gérées. Voici un schéma ci-dessous des diverses animations présentes.



Comme nous pouvons le voir, il y a beaucoup d'animations mais pour le moment nous nous occupons seulement des animations nommées “basic”, ”Playerrun” et ”Playerrunleft” pour le moment. Premièrement l'animation gris clair “basic” représente l'animation de base que le code va exécuter automatiquement au démarrage du jeu et qui se traduit par le joueur ne bougeant pas. Ensuite nous avons les deux autres animations qui ne sont activées que lorsque la condition requise est là grâce à une transition représentée par la flèche que vous pouvez voir sur l'image qui part de basic vers playerrun puis une autre vers Playerrunleft. Il y a également une flèche de retour pour que lorsque la condition n'est plus validée le joueur puisse revenir à l'état basique et éventuellement passer à une autre animation.



Ici en partant de basic, si la variable speed que l'on initialise dans la fonction Update du code MOVE.cs est supérieur à 0.3 (que la vitesse est positive) et que Saut est inférieur à 10 (c'est à dire que le joueur est au sol), on active l'animation du joueur qui court vers la droite. Ensuite 2 cas sont possibles, soit le joueur saute et nous passons sur l'animation jump ou le joueur se stoppe ou le joueur bouge à gauche (Speed <-0.3) et on active l'animation pour faire courir le joueur à gauche.

Maintenant passons à l'option saut du joueur. Premièrement nous avons créé des objets invisibles nommées “groundcheckleft” et “groundcheckright” que nous avons collés aux pieds du joueur de façon à ce que ces 2 objets suivent le joueur en permanence. Premièrement nous avons besoin de savoir si le joueur est au sol ou non en permanence pour faire en sorte que le joueur ne puisse sauter que lorsqu'il est au sol pour ne pas faire des sauts à l'infini. Pour cela nous utilisons cette ligne de code :

“isGrounded=Physics2D.OverlapArea(groundCheckLeft.position,groundCheckRight.position);” qui permet grâce au OverlapArea de regarder si les 2 objets sont en contact avec une boîte de collision quelle qu'elle soit, si oui le joueur est au sol et peut sauter sinon il ne peut pas. La variable isGrounded permet également de déclencher l'animation saut lorsque isGrounded a la valeur “true” et que le joueur appuie sur la touche espace du clavier.

Pour finir sur cette soutenance, nous avons ajouté une barre de vie gérée dans le fichier PlayerHealth.cs et Helathbar.cs dont les arguments sont les suivants:



Pour cela nous créons un slider (une barre rouge qui peut prendre une valeur entre 0 et 100 et se remplir de rouge en fonction de la valeur currentHealth dans le code. Pour finir, nous avons ajouté une méthode “TakeDamage” qui permet tout simplement de baisser ou de monter les points de vies du joueur lorsque ce dernier se fera attaquer ou qu'il se soignera. Voici une image du rendu de la barre de vie ci-dessous :



## 2ème soutenance : (Monstre)

Lors de la première soutenance, nous avions seulement abordé la partie développement du personnage qu'est l'astronaute et que le joueur contrôlera, c'est-à-dire ses déplacements et sa barre de vie. Cette fois-ci, il est question de gérer des IA qui sont en fait des monstres contre lesquels le joueur devra se défendre grâce à diverses attaques qui, elles aussi, ont été codées. Pour le moment ce ne sont que de simples IA, ces dernières peuvent seulement suivre le joueur, lui infliger des dégâts ou encore pouvoir juger à quel étage du vaisseau elles se trouvent et aviser quant au comportement à avoir pour atteindre et attaquer le joueur. Comme vous pouvez le voir ci-dessous nous avons implémenté trois types de monstres qui chacun ont leurs spécificités.



Le monstre en haut à gauche a pour comportement simple de toucher le joueur pour lui faire des dégâts et lorsque ce dernier tombe à 25% de sa vie totale, il se transforme en slime rouge qui aura une vitesse et des dégâts accrus. Pour cela il suffit juste de regarder en permanence la vie du monstre dont l'argument privé est situé dans la classe du monstre et au moment voulu changer l'image et l'animation du petit monstre. Pour le l'alien vert foncé en bas le comportement est quasiment le même au détail près qu'il peut sauter pour suivre le joueur lorsqu'il saute.

Pour pouvoir faire en sorte que le monstre suive le joueur nous avons besoin de connaître la position du joueur en permanence pour savoir si le monstre doit changer d'étage ou si ce dernier doit aller à gauche ou à droite. Pour cela voici un petit bout de code pour expliquer cela :

```
Y = GameObject.Find("perso_0");
y1 = Y.transform.position.y;
y2 = this.gameObject.transform.position.y;
```

La première ligne sert à partir du script d'un monstre de pouvoir obtenir la référence au joueur dont la référence d'objet est "perso\_0" pour pouvoir modifier ou obtenir ses paramètres. Avant cette ligne était placée dans la fonction update , c'est à dire qu'elle était exécutée en permanence mais pour des raisons d'optimisation nous avons jugé qu'il était mieux de la mettre dans la fonction start de tel sorte que la variable Y ne soit initialisée qu'une seule fois. La seconde partie permet à partir de la variable y qui représente le joueur de pouvoir obtenir la position en y du joueur. Quant à la dernière ligne, elle récupère la position en y du monstre auquel est attaché le script en question, d'où l'utilisation du "this". C'est ainsi par ce principe simple que l'on récupère les positions et que par de multiples cas nous pouvons gérer les mouvements et les rendre un minimum logique pour éviter que le monstre ne fasse que suivre bêtement le joueur sans le faire efficacement.

Lors de la dernière soutenance nous avions déjà implémenté la barre de vie du joueur présente en haut à gauche mais surtout, la méthode "Take damage" auquel on donne un argument de type Int et qui permet d'enlever un certain nombre de points de vie au joueur. Il nous a donc suffit d'utiliser cette méthode lorsque le monstre est suffisamment proche du joueur pour lui infliger des dégâts.

En début de chaque aura des capacités de base (points de vie, attaque) mais ces dernières augmentent au fur et à mesure de la partie pour en augmenter la difficulté.

Pour finir j'aimerais évoquer quelques difficultés qu'il y a pu avoir dans le développement de cette partie. Premièrement, la fonction qui gère le fait que le monstre ne peut attaquer que toutes les deux secondes devait être écrite en utilisant la méthode "System.Threading.Thread.Sleep(); \$" , or il s'est avéré que cela ne marchait pas car cette dernière bloque l'ensemble du jeu pendant 2 secondes. Nous avons donc dû utiliser la méthode "Time.time" qui va donner l'heure universelle précise en seconde, il suffit ensuite par la suite de donner la valeur Time.time à une variable et pour finir on autorise le monstre à faire des dégâts seulement quand on aura atteint Time.time + 2.

Et enfin la seconde et quant à elle liée à l'animation dont la fluidité est compliquée à gérer, d'ailleurs nous tenons à signaler qu'en l'état actuel des choses , les animations ne sont pas encore excellentes à ce moment-là du projet.

### **3ème soutenance : (Attaque du joueur et minage/BOSS)**

Dans cette partie dédiée aux personnages la majeure partie du travail était déjà terminé lors des 2 premières soutenances mais il restait néanmoins un domaine important qui était le fait que le joueur puisse attaquer les monstres avec le poing puis par la suite dans le jeu avec un sabre laser. Avant de commencer voici un aperçu du joueur lorsqu'il attaque ci-dessous :



L'attaque reprend le même principe que lorsque c'est le monstre qui attaque, quand le joueur est à côté d'un monstre (ce que nous détectons grâce à leurs coordonnées) s'il appuie sur la touche P ce dernier déclenche l'animation d'attaque et enclenchera la fonction "TakeDamage1" pour le monstre de type Slime et la fonction "TakeDamage2" de façon à lui faire prendre des dégâts en fonction de la valeur d'attaque du joueur qui est vouée à augmenter au fur et à mesure. Sur l'image ci-dessus vous pouvez observer que le joueur attaque vers la droite il y a également une animation lorsque ce dernier attaque à sa gauche.

Ensuite, il y a un autre domaine qui a été implémenté et qui est celui du minage. C'est à dire qu'à chaque niveau passé le joueur arrive sur une nouvelle planète(c'est à dire que le fond en arrière plan change) et sur chaque planète le joueur bénéficie d'un maximum de ressources à récolter, que ce soit de l'uranium, du fer et du cuivre. Pour cela, nous avons installé une station de minage en haut à droite du vaisseau où le joueur en appuyant sur B peut obtenir aléatoirement une des 3 ressources en fonction de leur disponibilité et rareté.

Nous vous donnons un petit aperçu ci-dessous avec le joueur qui lorsqu'il mine, active une animation avec une pioche et des étincelles pour donner l'illusion d'un joueur qui mine des ressources.



Il est important de notifier que le minage est la façon principale d'obtenir des ressources mais il est également possible d'en obtenir avec les récompenses dues à l'accomplissement d'une mission.

Lorsque le joueur minera, il pourra également à partir du 3ème niveau obtenir un sabre laser qui lui sera indispensable à l'accomplissement d'une des missions mais aussi pour terrasser le boss de notre dont nous allons parler dans le prochain paragraphe.

Enfin, un boss final a également été implémenté reprenant globalement le même principe que les monstres dont nous avons déjà parlé plus haut. Sauf que ce dernier n'apparaît qu'à la fin lors de l'appui sur le bouton "SUMMON THE BOSS" que le joueur doit battre après avoir accompli toutes les quêtes afin de retrouver après tant de temps sa chère planète Terre.

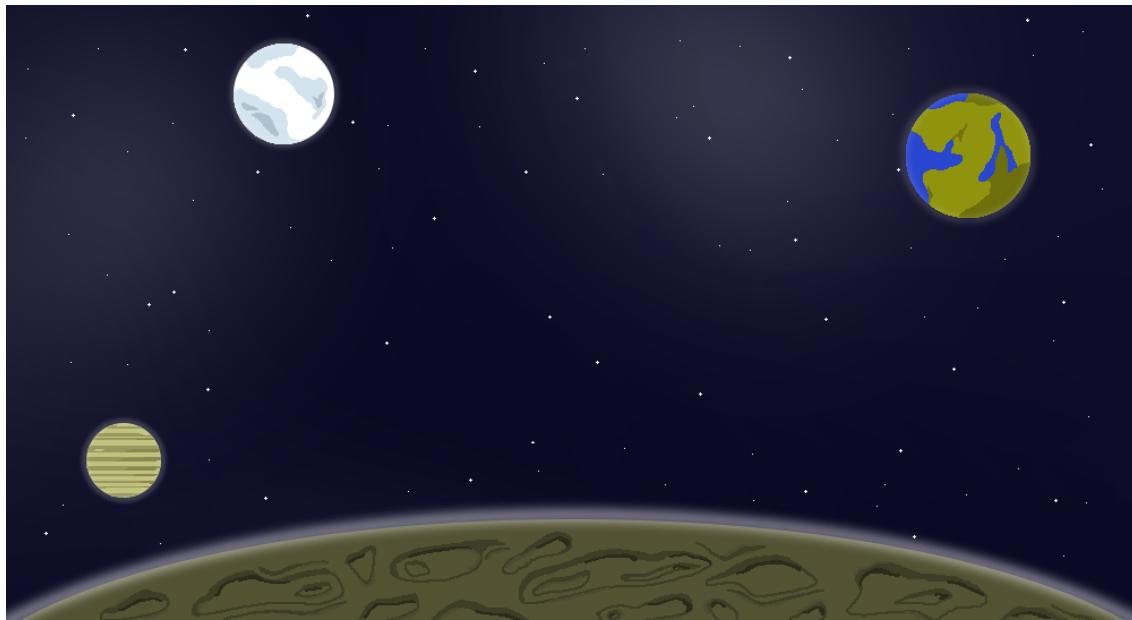
## 5.2 Gestion/Design planètes :

Cette partie consistait en la réalisation des différents décors autour du vaisseau lors de changements de planètes. À chaque reprise, nous avons dessiné sur Photoshop un fond étoilé avec une ou plusieurs planètes au loin et la planète sur laquelle nous nous situons au premier plan. Ces décors sont particulièrement longs à réaliser car le plan est grand. Ce qui est le plus compliqué dans cette tâche là est de trouver l'inspiration nécessaire et ainsi cela peut prendre parfois du temps.

Nous allons récapituler tous les fonds que nous utilisons dans notre jeu :

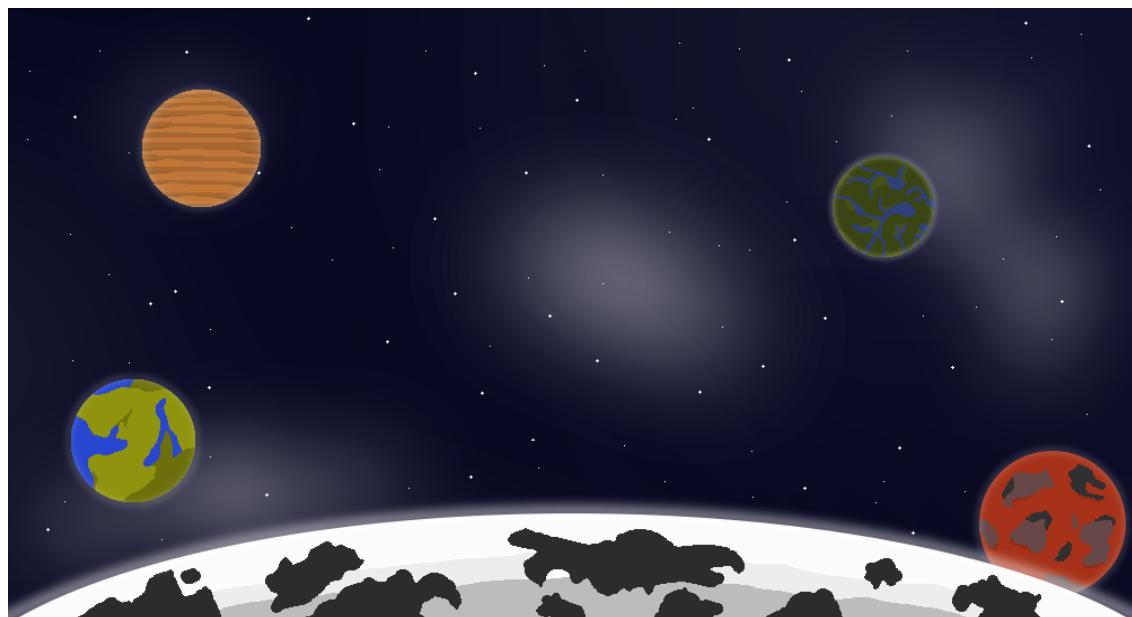
### 1ère soutenance :





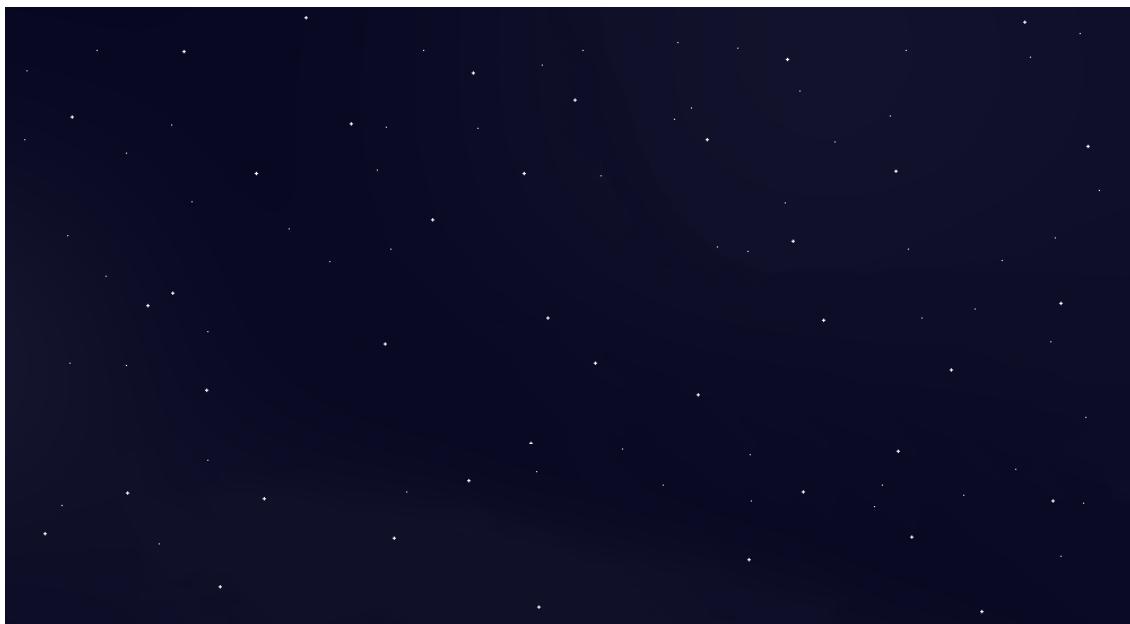
*3<sup>ème</sup> planète*

**2<sup>ème</sup> soutenance :**

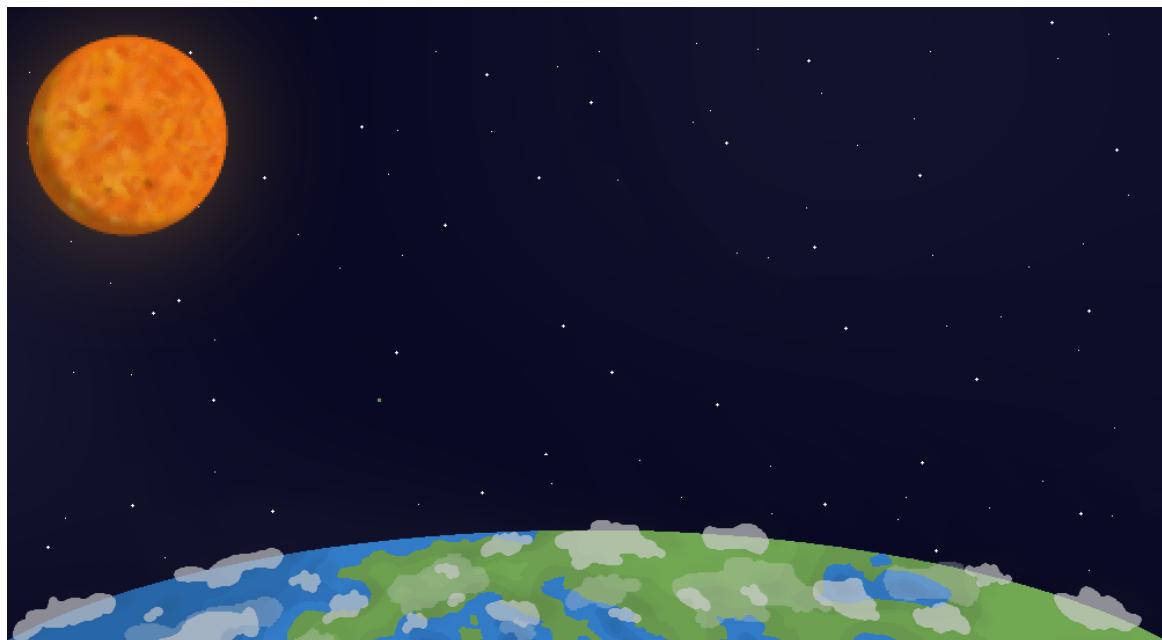


*1<sup>ère</sup> planète*

**3ème soutenance :**



*1<sup>er</sup> fond affiché au début du jeu*



*planète finale (Terre, centrée sur l'Europe)*

Nous sommes assez satisfaits de nos fonds affichés en jeu.

### 5.3 Gestion du vaisseau et design :

#### 1ère soutenance :

Au départ, nous avions décidé que la partie graphique de notre jeu soit entièrement réalisée à l'aide de Photoshop. Néanmoins, pour certains designs, nous avons repris l'aspect graphique de certains jeux. En effet, dans le cas de notre vaisseau, celui-ci peut rappeler les graphismes de Starbound.

Voici ce à quoi ressemblait notre vaisseau à la première soutenance :



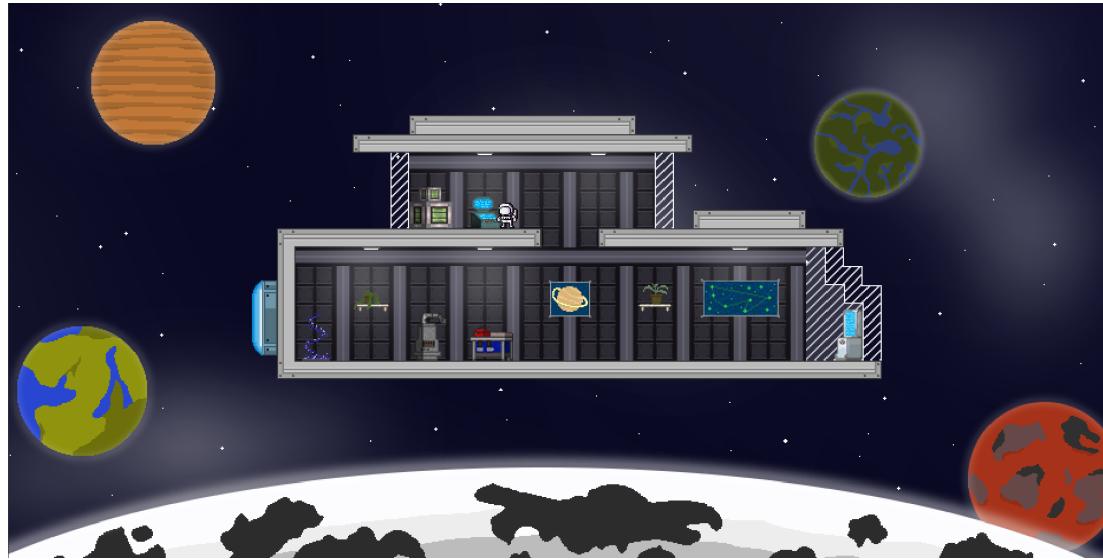
Ce vaisseau pouvait sembler simple car en effet il n'y a pas de surfaces rondes ou lisses. Cependant cela était voulu de notre part car notre objectif était de rester dans un univers que l'on pourrait qualifier de pixelisé d'où le "Blocks" dans Engi'Blocks. Pour ce vaisseau réparti sur deux étages, nous avons choisi des couleurs plutôt métalliques pour la surface extérieure du vaisseau et des couleurs relativement sombres pour en représenter l'intérieur.

#### 2ème soutenance :

\_\_\_\_\_ L'avancement de notre vaisseau au niveau du visuel n'avait pas changé depuis la précédente soutenance. En effet, nous n'avions pas implémenté de machines ou objets supplémentaires visuellement. Cependant, du côté des scripts nous avions fortement avancé pour n'avoir pratiquement qu'à ajouter l'image voulue.

De plus, du côté des dessins nous avions pris de l'avance en imaginant toutes celles que nous implémenterions de manière fonctionnelle par la suite. Pour cela, une fois de plus nous avons utilisé Photoshop et nous avons dessiné des objets décoratifs et des objets utilisables (tels qu'un poste de commande, un fourneau...). De plus, certains de ces objets utilisables possèdent une animation.

Ainsi voici le résultat que nous nous imaginions avec Photoshop (ce placement n'est pas forcément représentatif, il permet simplement d'avoir en un premier temps un aperçu de rendu) :



### 3ème soutenance :

Nous sommes enfin parvenus à ajouter plusieurs objets et machines à notre vaisseau de manière fonctionnelle. Le rendu final correspond à ce que nous espérions faire.

Ainsi, voici la liste définitive de tout ce qui se trouve dans notre vaisseau :

- Objets :
  - 2 plantes décoratives
  - 2 tableaux spatiaux
- Machines :
  - le poste de pilotage pour changer de planète une fois les missions terminées
  - la machine gérant les missions à réaliser
  - le portail pour accéder à l'étage supérieur
  - la table de fabrication permet de créer des ressources et des objets nécessaires à la progression de l'aventure
  - le fourneau électrique pour faire fondre les ressources minées
  - la station de minage qui sert à miner les ressources environnantes trouvées dans des astéroïdes
  - le réacteur qui permet de fournir de l'énergie à l'ensemble du vaisseau et machines

Ainsi, voici dans une partie “trichée” (toutes les machines ici ont été fabriquées, car au départ il n'y a pas beaucoup de machines) ce à quoi ressemble notre vaisseau :



#### 5.4 Site internet :

Le site internet constitue indirectement la vitrine de notre projet, c'est donc une étape importante. Nous allons vous présenter l'évolution de notre site Internet qui a beaucoup changé au fil de ces 3 soutenances.

#### 1ère soutenance :

À cette période là, notre site était assez simple. Le site internet est hébergé grâce à GitHub Pages et est accessible publiquement à cette adresse : <https://engi-blocks.github.io/site/>.

Pour réaliser le site nous avions choisi de le faire nous-même dans son entiereté. Pour cela, nous avions utilisé Bootstrap avec pour langage l'HTML auxquelles nous avions appliqué un design grâce aux CSS. Le site réalisé est un site web statique et non dynamique, car nous avions jugé que faire un site web statique était le plus approprié à notre situation. La prise en main au départ était assez étrange, car personne dans notre groupe n'avait encore touché à ce type de programmation. Cependant, il y a une documentation assez large sur Internet, de plus, on peut retrouver de nombreux tutoriels que ce soit sur YouTube ou sur certains sites pour prendre en main Bootstrap.

Une fois les bases acquises nous avions commencé à nous pencher sur l'esthétique de notre site. Nous cherchions à faire quelque chose d'assez épuré et agréable à l'œil. Ainsi, après plusieurs tentatives et une bannière réalisée sur Photoshop nous étions parvenus à ce résultat :



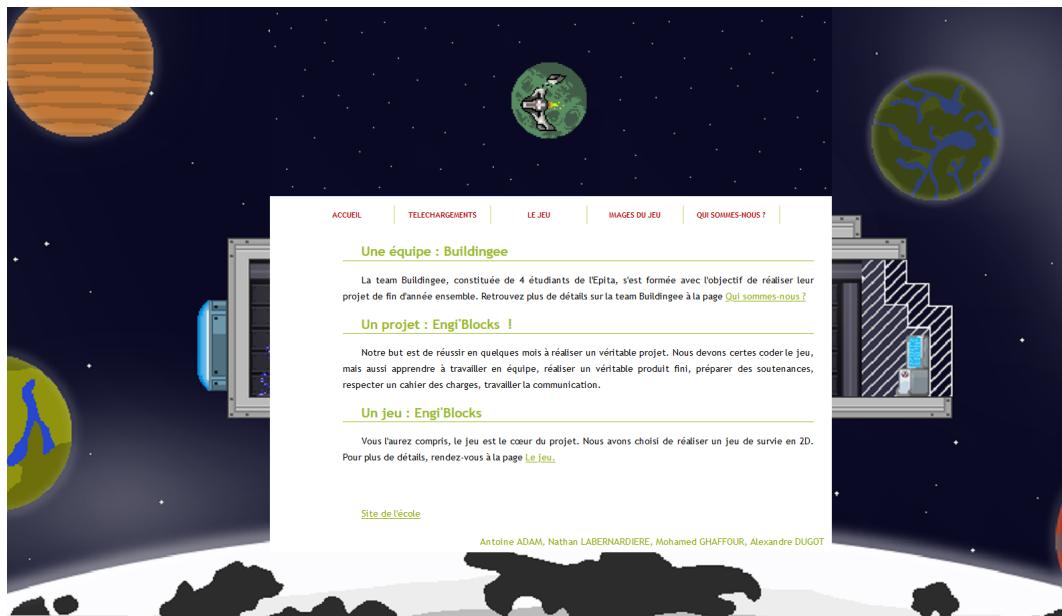
Ce thème sombre est immersif permettant de plonger la personne entrant sur notre site directement dans notre univers. De plus, nous avons décidé de placer 5 rubriques cliquables dans notre site que sont: "Accueil", "Téléchargements", "Règles du jeu", "Nouveautés" et enfin "Qui sommes-nous?". Elles n'ont pas encore été implémentées, c'est-à-dire que pour le moment elles nous redirigent uniquement vers une page blanche avec le menu du site. Nous avons déjà commencé à rédiger quelques notes afin de nourrir ce site. Des captures d'écran du jeu permettent d'illustrer la catégorie "Nouveautés" afin de montrer l'évolution du développement du projet, cette rubrique nous permettra également d'exposer les différents problèmes rencontrés au cours de ce dernier.

### **2ème soutenance :**

Une fois les bases acquises en HTML et en CSS nous avions pris comme décision d'oublier Bootstrap et de coder entièrement notre site. Certes, l'utilisation de Bootstrap aurait pu nous faire gagner beaucoup de temps mais nous pensons que le mérite a plus d'importance lorsque le site est codé entièrement par nos soins.

Une fois les connaissances acquises nous nous étions penchés sur l'esthétique de notre site, nous avons décidé de placer 5 rubriques cliquables dans notre site qui sont : "Accueil", "Téléchargements", "Le jeu", "Images du jeu" et enfin "Qui sommes-nous ?". Toutes les rubriques sont déjà disponibles cependant la rubrique "Téléchargements" n'est pas encore finalisée. Chaque rubrique a pour but de mettre le futur joueur dans le contexte de notre jeu.

Voici ci-joint la page d'accueil du site lors de cette soutenance :



Le site Engi'blocks continue de représenter notre jeu sur la toile Internet :

- Maintenu à jour assez régulièrement : le plus important pour notre site web était qu'il reste à jour. Ainsi nous avions continué de poster des nouvelles de notre avancement sur la rubrique "Le Jeu".

- Réorganisation : le contenu du site étant de plus en plus conséquent, nous avons procédé à quelques réorganisations.

La partie Téléchargement a été réorganisée en deux parties :

– Téléchargements du jeu : Cette partie permet de télécharger la dernière version publique de Engiblocks

– Soutenances : Cette dernière partie propose notre cahier des charges ainsi que tous les documents des deux soutenances précédentes.

Les autres parties ont été moins modifiées.

### **3ème soutenance :**

Lors de cette soutenance finale le site internet était quasiment terminé, nous avons eu pour but de finaliser et de régler le moindre détail. La mission principale était d'ajouter les exécutables sur le site pour les différents systèmes d'exploitations ainsi que les rapports des différentes soutenances. Le résultat final du site est pour notre part très satisfaisant et ressemble à nos attentes, en effet il est simple et efficace comme Engi'blocks.

### **5.5 Menu principal :**

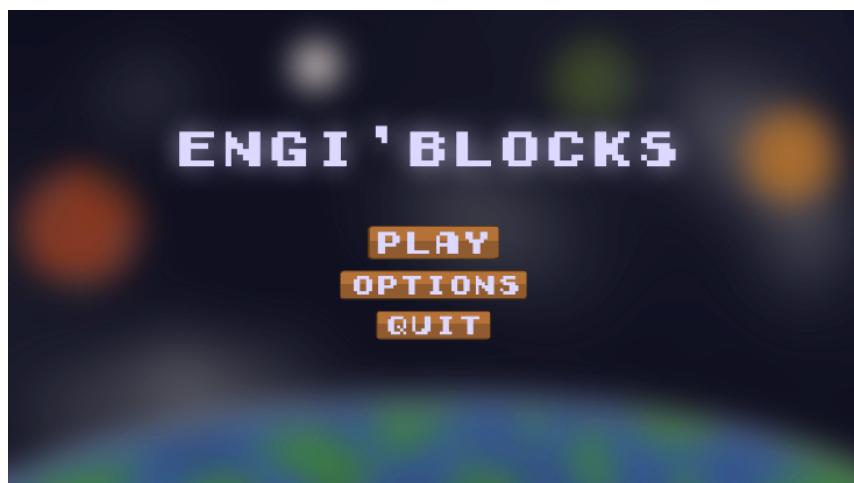
Le menu principal dans un jeu, même si celui-ci n'est pas réellement d'une grande utilité (on ne joue pas à un jeu pour son menu principal), n'est pas un élément à négliger. Nous tenions donc à soigner le plus possible celui-ci pour que lorsque le joueur arrive pour la première fois sur notre jeu, ait directement une bonne image du jeu.

### **1ère soutenance :**

Lors de cette soutenance le menu principal n'avait pas encore été implémenté car les diverses tâches que nous devions réaliser à ce moment-là ne nécessitent pas l'implémentation d'un menu principal pour être fonctionnelles.

### **2ème soutenance :**

Pour cette soutenance, il nous était obligatoire de créer un menu principal pour pouvoir avancer dans nos tâches. Voici le rendu que nous sommes parvenus à obtenir :



Cela ne peut pas se voir sur la capture d'écran, mais nous avons également décidé de rendre le titre de notre jeu dynamique pour rendre le menu principal plus vivant (le titre avance et recule lentement).

Ainsi, ce menu propose trois options cliquables par le joueur que sont :

- PLAY : Cela permet pour le moment de renvoyer le joueur sur la scène principale (là où le jeu se déroule) avec le menu des sauvegardes.

- OPTIONS : Ce bouton est accessible depuis le menu principal ainsi qu'en jeu. Une fois cliqué cela ouvre une fenêtre proposant différentes options aux joueurs. Pour des soucis d'esthétisme, le titre du jeu dynamique en fond est désactivé lorsque cette fenêtre est ouverte car sinon l'on verrait quelques lettres du titre dépasser. Ainsi, les joueurs peuvent choisir s'ils veulent jouer en fenêtré ou plein écran (le jeu sera toujours lancé automatiquement en plein écran) ou alors augmenter ou diminuer les sons et effets sonores.

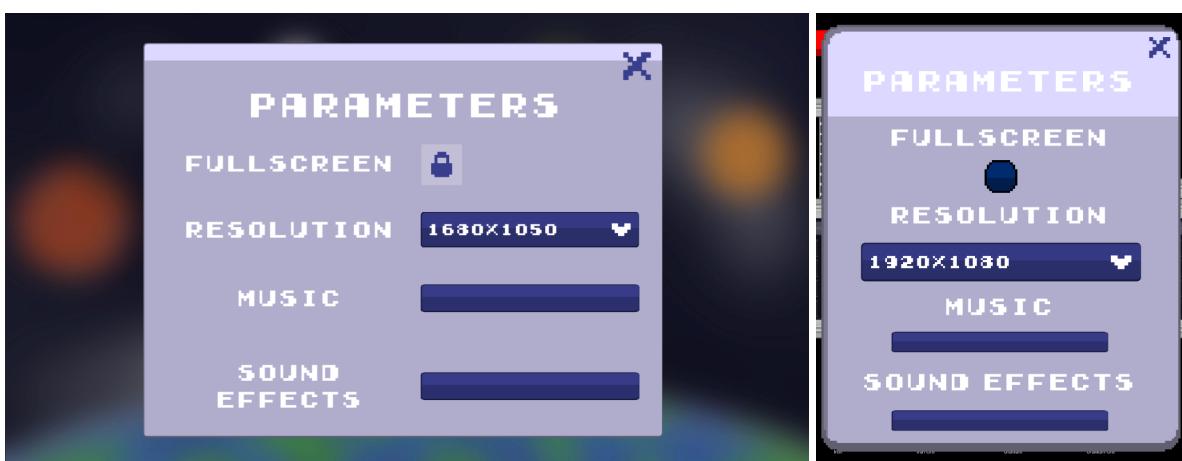
Il est également possible de choisir la résolution d'écran voulue. Cependant au départ nous avions rencontré un problème dans notre code car lorsque l'on lançait le jeu, les résolutions proposées étaient toutes en double. Pour corriger cela, nous avons cherché sur Internet et avons trouvé une ligne de code permettant de remédier à cet ennui :

```
resolutions = Screen.resolutions.Select(resolution => new Resolution { width = resolution.width, height = resolution.height }).Distinct().ToArray();
```

Cela permet de récupérer toutes les résolutions disponibles grâce au "Select". Ensuite, tous les éléments dupliqués sont supprimés grâce à l'expression "Distinct()".

Et enfin, les résolutions vont être stockées à l'aide de l'expression "ToArray()" dans notre tableau de résolutions "resolutions".

Voici ci-dessous à gauche le rendu depuis le menu principal ainsi qu'à droite le rendu en jeu :



- 
- QUIT : ce bouton permet tout simplement à la personne de fermer l'application.

Pour rendre ce menu principal le plus agréable possible pour les joueurs, nous avons choisi d'utiliser la classe “PlayerPrefs”. Cette classe va être utile dans la fenêtre où le joueur va effectuer ses réglages. Désormais les choix du joueur pour la résolution d'écran ainsi que la musique et les effets sonores seront toujours conservés entre deux scènes, ou alors à chaque fois qu'il lancera le jeu.

### **3ème soutenance :**

Pour cette dernière soutenance, aucun réel changement majeur hormis le fait qu'une scène de crédits est accessible depuis le menu principal mais aussi quand le joueur termine sa partie.

Nous avons jugé que cela était nécessaire afin de remercier les diverses personnes qui nous ont aidé, que ce soit sur des vidéos Youtube, des forums ou encore des documentations Web mais aussi afin de mettre en évidence les créateurs du jeu. Cette scène est un texte défilant donc il est impossible de tout montrer sur ce rapport mais néanmoins voici tout de même une capture d'écran :



## 5.6 Multijoueur :

Nous avons laissé la partie multijoueur dans la liste des tâches bien que nous ne l'ayons pas implémenté pour préciser de manière notre décision de ne pas le coder pour nous concentrer sur le jeu et ses fonctionnalités. Et il est important d'ajouter que la question du temps est un facteur car ayant pris du retard entre la première et la seconde soutenance sur certaines tâches, cela a fortement motivé notre choix de le supprimer du projet.

## 5.7 Sons / effets sonores :

Les sons et effets sonores sont importants dans un jeu vidéo pour l'immersion du joueur. En effet, si les sons proposés au joueur sont de mauvaises qualités ou mal choisis, le joueur se lassera. De plus, un jeu dénué de tout son est plutôt triste.

Nous avons ainsi consacré beaucoup de temps pour obtenir une bonne qualité pour l'environnement sonore de notre jeu. À noter que l'entièreté des sons et effets sonores sont des sons libres de droit que l'on a pris sur Internet, notamment à l'adresse suivante : <https://soundimage.org/>.

### 1ère soutenance :

Lors de la première soutenance nous avions prévu de ne pas encore s'occuper des musiques et effets sonores car le jeu en était à ses débuts ainsi cela n'avait pas d'importance d'essayer d'implémenter différents sons aussi tôt. Néanmoins, nous tenions à prendre de l'avance ainsi nous avions repéré des sites pour télécharger des sons plus tard.

### 2ème soutenance :

\_\_\_\_ Pour gérer tous nos sons, nous utilisons un mixeur audio contenant un groupe avec l'ensemble des sons et effets sonores et à ce groupe y est rattaché deux groupes enfants ; un pour les sons et un pour les effets sonores. En effet, il était important de créer deux groupes enfants pour différencier les musiques des effets sonores car grâce à cela le joueur a désormais plus de choix pour les réglages du volume et cela est plus agréable. On peut retrouver ces réglages là dans le menu principal ainsi que dans le jeu.

Également, nous avons utilisé un algorithme intéressant trouvé sur un forum pour ajuster plus correctement le volume du jeu.

```
audioMixer.SetFloat("Music", Mathf.Log10(volume) * 20);
```

Le fait d'utiliser la fonction log et de régler la barre de volume entre 0.0001 dB et 1 dB nous permet d'ajuster de manière plus précise le son. En effet, lorsque le joueur descendra la barre vers 0.0001 le son baissera beaucoup plus vite jusqu'à s'éteindre tandis que lorsque la barre s'approchera de 1 le son ne variera que peu.

Ici, le plus compliqué a été d'implémenter des sons. Nous ne savions pas vraiment comment cela marchait mais grâce à de la documentation sur Internet et des forums nous avons pu implémenter un fond sonore de vaisseau et des bruits de déplacements.

Pour le son ambiant du vaisseau nous utilisons un AudioClip[] et une AudioSource et nous lançons simplement le son en boucle :

```
void Start()
{
    audioSource.clip = playlist[0];
    audioSource.Play();
}
```

Pour les bruits de pas c'est différent, nous vérifions que le personnage se déplace et qu'il est au sol et si c'est le cas nous lançons le son sinon on le stoppe :

```
void Update ()
{
    if ((Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.LeftArrow)) && isGrounded)
        GetComponent<AudioSource>().UnPause();
    else
        GetComponent<AudioSource>().Pause();
}
```

### 3ème soutenance :

Beaucoup de changements ont eu lieu pour cette soutenance. En effet beaucoup de nouvelles choses ont été implémentées et ainsi il a fallu rajouter des musiques et sons qui vont avec. Tout d'abord le menu principal possède désormais une musique. Cette musique est assez calme et lente car nous voulons immerger le joueur avant même qu'il ne lance le jeu. Cela inspire une sorte de solitude et le calme de l'espace. De plus, maintenant qu'il y a des crédits, nous avons aussi ajouté une musique pour celle-ci. À l'instar de celle du menu principal, cette musique est calme et lente.

Concernant en jeu, tout d'abord il y a eu un changement du fond sonore du vaisseau. Celui-ci a été changé par un autre de meilleure qualité et qui tourne en boucle sans que l'on entende de coupure. Nous avons dû ajouter des effets sonores lorsque le personnage donne un coup de poing ainsi que lorsqu'il mine. Pour l'effet de minage ainsi que le coup de poing, nous utilisons simplement la méthode "PlayOneShot()" à chaque fois que l'animation est lancée.

Lorsque le joueur ouvre la carte, les missions ainsi que l'interface des fabrications, un petit son est joué grâce à la méthode “GetComponent< AudioSource >().Play(0);”.

Évidemment, nous ne détaillerons pas complètement en entier cependant certains effets sonores ont des conditions supplémentaires pour pouvoir être joués. Par exemple, lorsque le menu pause est activé, le temps est arrêté et donc les déplacements du joueur aussi. Ainsi, lorsque ce menu est activé, le script où le son des pas est géré est désactivé.

En définitive, nous sommes plutôt satisfaits de l'ambiance sonore que nous avons pu produire pour notre jeu et nous espérons que les personnes qui essayeront notre jeu seront du même avis.

### 5.8 Interface carte :

Le but de cette partie est d'obtenir une carte que l'on peut ouvrir depuis le poste de pilotage à l'avant. À partir de cette dernière, il y aura possibilité d'accéder aux autres systèmes stellaires qui seront déblocables au fur et à mesure en ayant accompli les missions.

#### 1ère soutenance:

Pour le moment tout n'est pas encore implémenté, pour l'instant le joueur peut accéder à la carte grâce à la touche “M” quand il est dans la “boxcollider2D” du poste de pilotage, cette touche permet d'obtenir une image au centre de l'écran représentant la carte. Pour quitter la carte ce dernier peut appuyer sur la touche “ESCAPE”. Pour le moment il n'y a que cela qui est implémenté, car nous attendons d'avoir terminé les missions étant nécessaires aux conditions de la carte pour pouvoir commencer à y ajouter les divers systèmes. Pour afficher la carte à des moments voulus nous avons utilisé la commande **InteractUI.enabled = false;**

où le interactUI correspond au tag du fichier image/texte à afficher et ensuite le enabled qui étant faux ou vrai affiche ou non la carte ou le texte.

Pour détecter lorsque le joueur appuie sur une touche en question nous avons utilisé la commande **Input.GetKeyDown(KeyCode.example)** qui permet de détecter quand le joueur appuie sur la touche en question et renvoie un booléen.

Voici donc un rapide aperçu ci-dessous :



Pour finir il est important de comprendre qu'à ce niveau là, la carte n'était pas fonctionnelle et pouvait seulement être ouverte afin de voir une image.

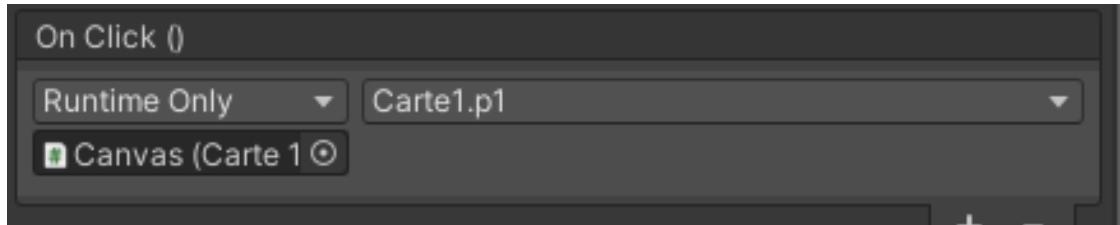
### **2ème soutenance :**

Dans le précédent rapport un poste de pilotage avait déjà été implémenté permettant lorsque nous sommes dans un certain rayon autour de celui-ci ainsi qu'en appuyant sur la touche M de pouvoir l'ouvrir. À ce moment-là, cette dernière n'était qu'un écran avec étoiles sans utilité. Mais désormais cette carte a enfin l'utilité prévue, lorsque l'on ouvre la carte, nous pouvons observer quatre planètes à l'écran ainsi qu'un bouton en dessous de chacune d'elle comme vous pouvez le voir ci-dessous :



Chacun de ces boutons permettent au joueur lorsque ce dernier en aura rempli les conditions de pouvoir passer au niveau suivant. Pour gérer ces 4 boutons nous avons utilisé un seul script avec à chaque une méthode reliée à chacun de ses boutons et qui dès que le bouton est activé par le joueur s'exécute.

Chaque bouton est ce que l'on appelle un GameObject sur unity, ces derniers ont donc besoin d'être reliés de la façon suivante :



Le Canvas(Carte 1) représente le code auquel on veut lier notre bouton et le Carte1.p1 représente la méthode “p1” que l'on veut exécuter lorsque l'on appuie sur le bouton en question. Pour finir lorsque le bouton est enclenché , nous activons l'apparition de 2 monstres qui à chaque nouvel appuie deviennent de plus de plus en plus fort.

Ci-dessous, vous trouverez un exemple de code exécuté lorsque l'on appuie sur le bouton tout à gauche de l'image présentée au-dessus.

```
public void p1()
{
    if (cp1)
    {
        monstre1.SetActive(true);
        monstre2.SetActive(true);
        cp1 = false;
    }
    else
    {
    }
}
```

Ici, lorsque la condition liée au mission est vérifiée grâce à la variable booléenne “cp1” , nous activons la mise en marche des 2 monstres en utilisant la commande “GameObject.SetActive(true);” qui permet d'activer le GameObject en question.

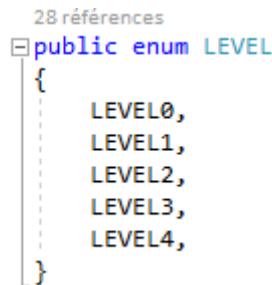
Comme vous pouvez le remarquer, lorsque l'on rentre dans le “if” nous remettons de suite cp1 à l'état “false” pour que le joueur ne puisse relancer plusieurs fois le niveau et ainsi pouvoir profiter des divers avantages qu'offre la validation d'un niveau de façon illimitée.

Pour finir il y a également un “else” qui lui est vide pour le moment est qui à terme quand toute les missions seront implémentées aura pour instruction si les conditions ne sont pas validées , d'envoyer un message au joueur lui disant les missions qu'il n'a pas achevé.

### **3ème soutenance :**

Actuellement, la carte dans ses aspects graphiques et fonctionnelle n'ont pas eu de changements hormis des changements tel qu'un changement de rendu du bouton graphiquement.

Cependant lorsque l'on appuie sur les différents niveau , cela ne fait plus simplement apparaître des monstres comme avant mais c'est aussi là que grâce à un enum appelé “LEVEL” dans le code Carte1.cs que nous savons après que le joueur ait changé de level à quel level il se trouve.



```
28 références
public enum LEVEL
{
    LEVEL0,
    LEVEL1,
    LEVEL2,
    LEVEL3,
    LEVEL4,
}
```

Cela nous permet donc notamment lors des sauvegardes de pouvoir aisément enregistrer le niveau de progression du joueur étant donné que la simple définition de la variable statique level de type LEVEL permet d'initialiser toute ayant un rapport de près ou de loin avec le changement de niveau.

Par exemple, cela permet de changer le fond derrière le vaisseau simulant un changement de planète pour le joueur ou encore les missions qui à chaque niveau changent mais aussi les disponibilités en ressources de chaque planète qui sont actualisé à chaque changement de niveau.

En résumé, il y a eu peu de changements sur la carte mais un changement qui malgré sa courte longueur permet désormais de lier plusieurs parties de notre projet.

## 5.9 Cinématiques :

### **1ère soutenance :**

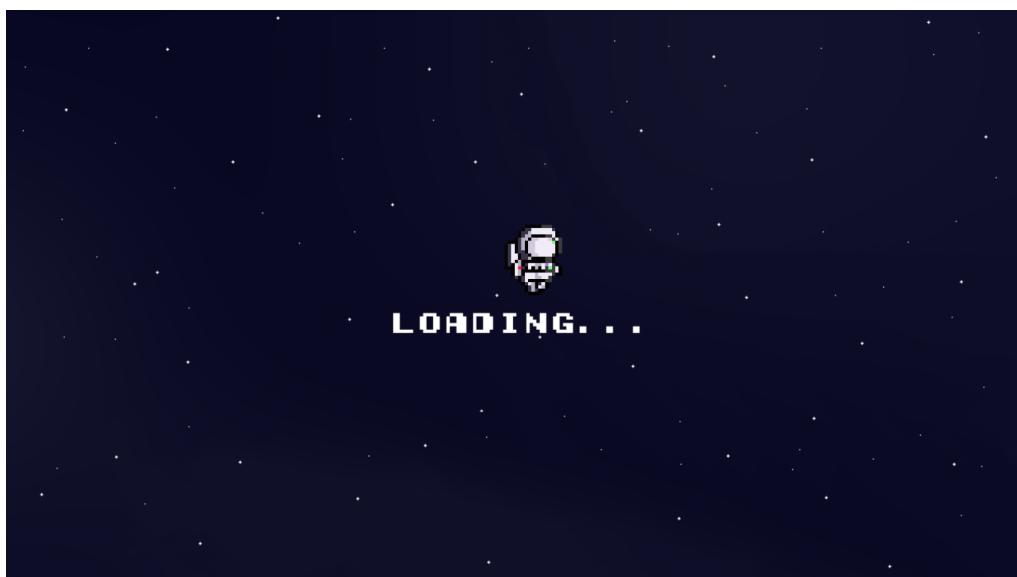
Lors de la première soutenance, nous n'avions encore rien effectué pour cette tâche, car nous comptions la commencer pour les futures soutenances. En ce qui concerne la façon de les réaliser, nous hésitions encore entre deux méthodes. Soit, nous enregistrerons une vidéo pour la faire défiler aux moments voulus soit, nous créerons une séquence d'images à partir d'Unity. À cette période là, nous envisagions également d'enregistrer des voix off à écouter lors de ces cinématiques.

### **2ème soutenance :**

Malheureusement, les cinématiques n'ont pas été une priorité pour cette soutenance. Nous avions pourtant prévu de commencer cette tâche malheureusement nous avons jugé plus raisonnable de finir correctement les autres tâches à réaliser ainsi d'attendre que le jeu soit entièrement implémenté pour savoir en détails où ajouter des cinématiques/transitions.

### **3ème soutenance :**

Le jeu étant enfin entièrement développé, nous avons pu implémenter quelque chose qui se rapproche plus d'une transition qu'une cinématique (à l'origine nous envisagions de faire défiler une longue cinématique pour introduire le jeu). Nous avons donc créé une petite animation avec écrit "loading..." et notre personnage qui marche par dessus. En voici une capture d'écran :



Cette transition est affichée lorsque le joueur lance le jeu ainsi que lorsque le joueur change de planète grâce au poste de pilotage. Cette transition étant courte elle ne joue aucune mélodie lors du passage de menu principal au jeu. Lors d'un changement de planète, on entend juste le son du moteur du vaisseau. De plus, lorsque l'animation est jouée à ce moment-là, la carte est désactivée, les déplacements sont désactivés, on ne peut donc pas entendre le joueur se déplacer et il ne peut pas non plus bouger.

Cette transition très basique fait quand même son effet car jusque-là les changements d'environnement étaient très brutaux.

### 5.10 Missions et histoires :

La partie Missions et Histoires constitue une étape importante du projet, en effet selon l'originalité et la qualité de ces dernières le joueur sera plus au moins en immersion. Il nous a semblé évident que nous devions d'abord avoir en tête l'histoire du jeu, pour ensuite pouvoir enfin imaginer des missions qui donneront envie au joueur de progresser.

#### 1ère soutenance :

À la date de cette première soutenance de nombreuses missions ont vu le jour et l'histoire du jeu commence réellement à prendre forme. Cependant, pour implémenter ces missions, il est préférable d'attendre la version définitive des machines et de la carte, en effet ces dernières peuvent encore évoluer et donc donner place à d'autres missions ou à sens inverse mener à une incohérence.

Nous avons tout de même commencé à réfléchir à un schéma qui nous aidera à implémenter les missions. Si toutes les conditions ont été validées, le joueur pourra évoluer d'un niveau et atteindre une nouvelle planète.

Pour finir, il est important de préciser qu'à ce moment-là les missions n'étaient que du pseudo code ou encore de simples schéma sur papier avec leur conditions de validation. Globalement il faut dire que rien n'avait encore été codée en C# pour le moment pour les missions.

#### 2ème soutenance :

Comme ce sera dit dans la partie liée au retard, nous avons certains retards sur la partie des mission sur cette 2ème soutenance. La seule chose qui a été faite sont toutes les interfaces graphiques (Titre de l'interface, design de l'interface, bouton pour obtenir les récompenses....).

### **3ème soutenance :**

Enfin, lors de cette dernière période de travail , les missions ont pu être entièrement terminées. Ces dernières sont au nombre de 4 par niveau soit 16 au total car il y a 4 niveaux.

Les missions sont de 2 types différents, celles nécessitant de construire une machine et celles dont une collecte d'objets est obligatoire à leur accomplissement. Pour détecter leur accomplissement et éviter une perte de performance, nous actualisons l'accomplissement des missions que lorsque le joueur ouvre l'interface des missions en haut à gauche de notre vaisseau. Avant de parler de leur fonctionnement, voyons ensemble une capture d'écran de ce que cela donne afin de pouvoir expliquer tout cela de manière plus assidue.



Premièrement sur chaque niveau il y aura affiché la planète correspondante comme vous pouvez le voir en bas de l'image ci-dessus. Les 4 planètes correspondant aux 4 niveaux sont en réalité toute les 4 superposées aux même endroit, par conséquent pour afficher la bonne planète, il suffit d'activer seulement celle que l'on veut activer et de désactiver les 3 autres grâce à la commande "GameObject.SetActive(true/false)". Pour calibrer correctement cela , il nous suffit de regarder la variable "level" dans la classe "Carte1" de type LEVEL qui correspond à l'enum donc nous avons déjà parlé dans la partie liée à l'interface de la carte plus haut dans le rapport.

Pour les textes décrivant les 4 missions par niveau, ceci sont également implémentés en fonction de la valeur de "level" pour cela il suffit de modifier l'élément texte à chaque fois grâce à la commande "GameObject.GetComponent<Text>.text = "message".

Ensuite, chaque mission possède un bouton “CLAIM” qui permet au joueur de pouvoir récupérer sa récompense lorsque la mission est complétée. Comme vous l’avez sans doute remarqué sur l’image ci-dessus, ces derniers sont verts ou rouge , ceci correspondant en vert à une quête complétée et en rouge une non complétée. Pour cela en fonction de la condition nécessaire à la mission, je change simplement la couleur de la façon suivante “Button.GetComponent<Image>.color = new Color(r,g,b)” avec r,g et b des entier naturel entre 0 et 255. Après avoir cliqué sur le bouton claim, seulement quand il est vert, le joueur pourra obtenir sa récompense. Bien sûr grâce à une variable de type “bool” , nous faisons en sorte que le joueur ne puisse appuyer qu’une seule fois sur le bouton afin de lui éviter d’accumuler les récompenses, ce qui casserait toute la synergie de notre jeu.

Sachez aussi que l’accomplissement des 4 missions d’un niveau est obligatoire pour le passage au prochain niveau. Le joueur pourra essayer d’aller cliquer sur la carte mais cela n’aura aucun effet s’il n’a pas accompli les tâches demandées.

Pour finir, le dernier niveau est un niveau particulier car il y a deux autres quêtes “Kill the Boss” et “Return To Earth”. La mission permettant de tuer le boss de notre jeu est seulement accessible après avoir réalisé les 4 quêtes habituelles. La quête “Return To Earth” quant à elle est accessible après avoir tué le boss et marque tout simplement lorsque l’on appuie dessus la fin du jeu et va mener directement au crédit du jeu ensuite le menu principal.

### 5.11 Gestion des sauvegardes :

Les trois soutenances pour les items, machines et sauvegardes sont détaillées en une seule partie pour faciliter la compréhension.

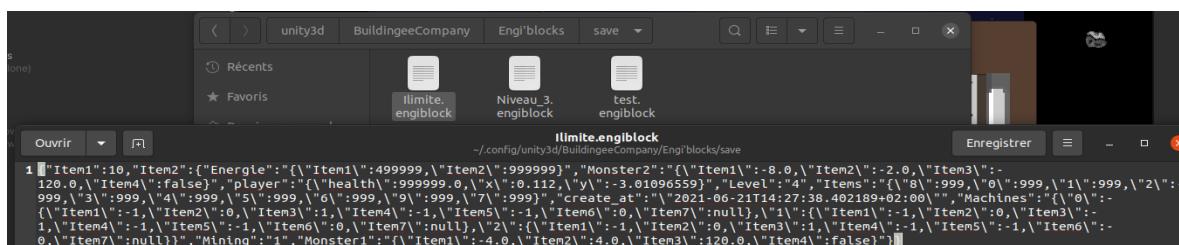
Les sauvegardes ont évolué toute au long du projet et actuellement elle fonctionne comme ceci :

Enregistrement :

- on utilise une fonction statique pour pouvoir l’utiliser partout et simplement dans le code, elle est exécutée au lancement de la scène principale, partout où il y a besoin des sauvegardes, et elle s’occupe d’ajouter les lambdas (fonction anonyme en argument à la fonction) à une liste dans la classe “Save”,  
les arguments :

- la clé : une chaîne de caractère permettant de cibler les données
- le lambda “save” : il retourne les données à sauvegarder
- le lambda “load” : il est appelé quand une sauvegarde est utilisée pour modifier les valeurs avec celle passée en paramètre du lambda
- le type : il permet au sérialiseur de garantir le typage de la valeur sauvegardée
- le lambda “newGame” : il est appelé si une nouvelle partie est créée

- Ensuite, si l'utilisateur clique sur un bouton de sauvegarde dans le menu pause, une boucle va parcourir la liste des éléments à sauvegarder et sérialiser les données récupérées par le lambda “save” pour les mettre dans un dictionnaire avec comme clé la clé vu ci-dessus, et comme valeur la chaîne de caractère donnée par le sérialiseur.
- Enfin, le dictionnaire est sérialisé à son tour avec le numéro de version codé en dur, la chaîne de caractère résultant est enregistrée dans un dossier dont le chemin est donné par Unity pour garantir le multiplateforme.





Sauvegarde :

Cette partie est plus compliquée, elle est un peu vulgarisée :

- Quand l'utilisateur arrive sur le menu de sauvegarde, une fonction scanne tous les fichiers dans le répertoire de sauvegardes et désérialise les données (l'inverse des deux derniers points vu ci-dessus), si la version de la sauvegarde est différente de celle du jeu ou si des valeurs provoquent une erreur, le fichier est supprimé, toutes les valeurs sont ensuite stockées dans un dictionnaire.
- Certaines valeurs obtenues sont affichées pour donner des détails sur les parties, comme le niveau d'avancement, les dates et le nom de la sauvegarde.
- Enfin, si l'utilisateur charge une partie, une fonction va exécuter tous les lambdas “load” avec les valeurs de la sauvegarde ciblée précédemment désérialisées.

### 5.12 Machines/Inventions :

Item :

Le terme anglais “item” correspond à “objet” en français, mais pour éviter des risques de confusion avec l’orienté objet en programmation, le terme anglais sera utilisé.  
Pour des raisons similaires, “scriptableObject” sera utilisé à la place de “objet scriptable”.

Les scriptableObject sont des instances de classe de configuration où les variables publics ont été modifiées par Unity pour améliorer la productivité sur des objets (au sens programmation) similaires. Unity propose une interface de modification agréable à utiliser pour inscrire les valeurs.

Les items ont radicalement changé entre la première et seconde soutenance avec un peu de modification pour la troisième.

Tous types d'item possèdent des informations constantes et ils sont donc stockés dans des scriptableObjects. Les constantes :

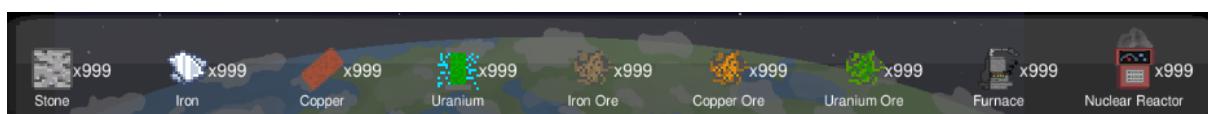
- identifiant unique (“id”)
- l'image
- le nom
- le nombre d'item au lancement d'une nouvelle partie
- la description

La classe item stock :

- la quantité
- le type (ci-dessus)
- une liste de vues (“List<ItemView>”)
- une liste de lambdas, pour stocker des fonctions qui seront exécutées à chaque changement de quantité (c'est une optimisation)

“ItemView” est une classe qui permet d'afficher l'item auquel elle est rattachée en générant dynamiquement plusieurs gameObject. L'intérêt de cette classe est de pouvoir afficher à plusieurs endroits différents le même item tout en gardant son nombre actualisé.

Le joueur possède un inventaire correspondant à la classe “InventoryShip” où est stocké un dictionnaire liant l'id à l'instance de son item dans l'inventaire. Toutes les méthodes sont statiques pour permettre de manipuler l'inventaire dans toutes les parties du code, cette classe possède aussi un système d'événement pour déterminer si un item de l'inventaire a changé de quantité toujours en utilisant les lambdas.



## Machines :

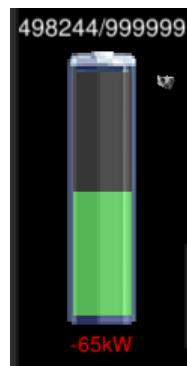
### Gestion de l'énergie électrique :

Pour simplifier les machines, les actualisations sont effectuées toutes les secondes, cela permet aussi d'avoir une correspondance entre la puissance en Watt et l'énergie en Joule ( $W=J/s$  avec  $s=1 \Leftrightarrow W=J$ ) chaque seconde, la somme des puissances est ajouté à l'énergie. Il n'y a qu'un seul stockage qui est commun à tout le vaisseau avec une capacité limitée à 10 000 ( $kJ=kW/s$ ). Les machines consomment ou produisent de l'électricité de différentes manière selon leur niveau :

- Lorsque la machine est inactive
- Quand elle travaille selon la recette

Toutes les valeurs de consommation sont stockées dans les `scriptableObjects` des machines. À chaque actualisation, les machines vont ajouter leur puissance à l'énergie totale. Si l'énergie vient à manquer, la machine arrête le processus en cours, ce qui rendrait probablement impossible la fin du jeu si le joueur ne peut plus produire de l'électricité.

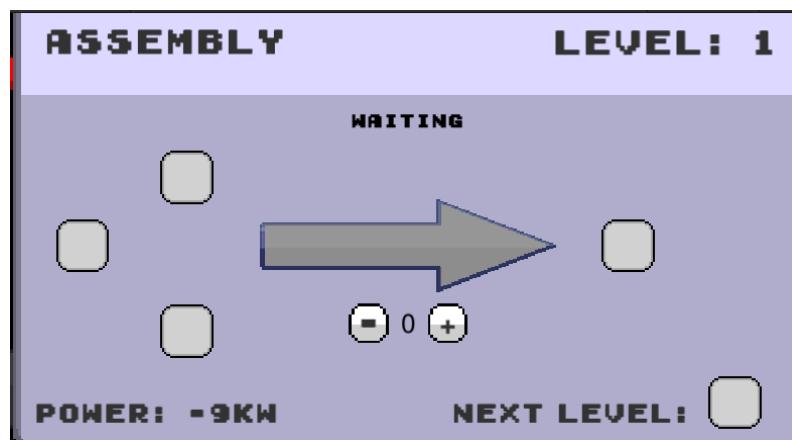
Le joueur dispose d'une jauge d'énergie électrique avec une indication sur sa capacité, sa contenance et la puissance totale.



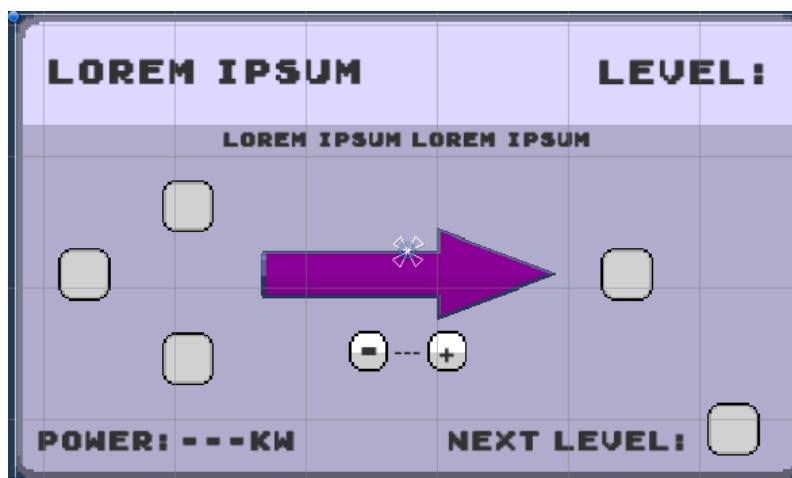
“Préfabs” :

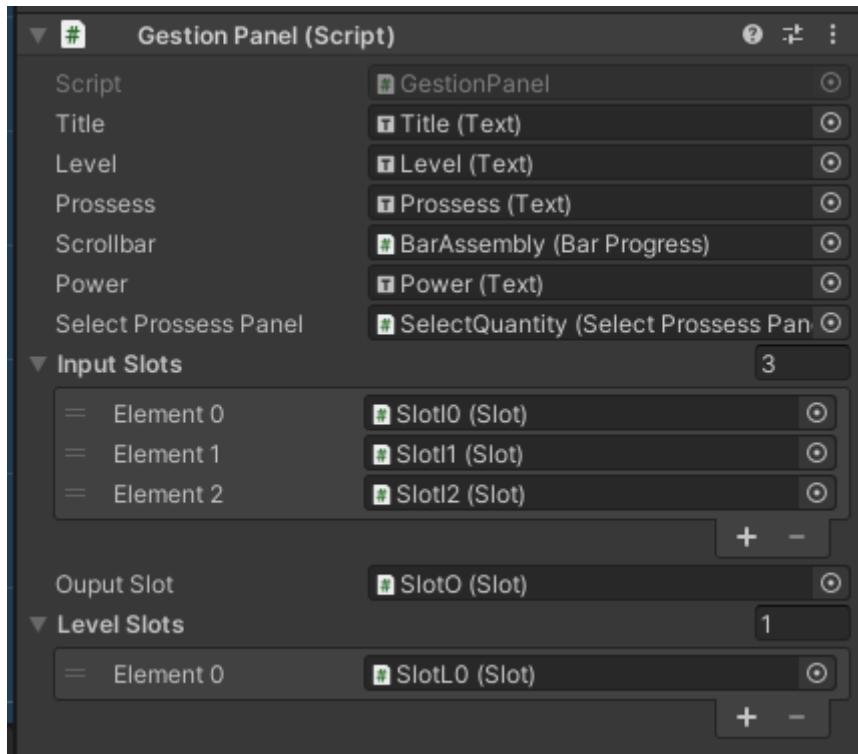
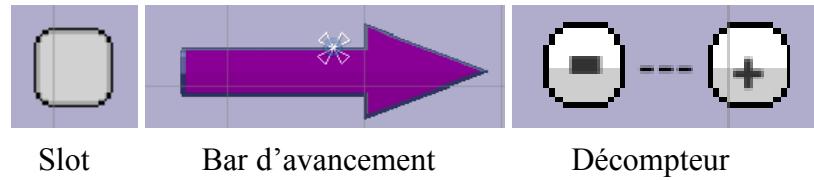
Pour améliorer la productivité, les panels des machines sont des préfabs contenant des sous-éléments communs à toutes les machines. Pour créer un panel, il suffit juste de glisser et déposer ces sous-préfabs dans un nouveau préfab et d'y ajouter un connecteur (un script en C#) qui permet de lier tous les sous-éléments. La classe machine pourra interagir avec le panel par l'intermédiaire de celui-ci, le sous-élément possédant des composants de code leur donnent une logique d'Interface Homme Machine.

Ce qui donne en jeu :



Le préfab :





“Connecteur” (interface entre la classe Machine et le préfab panel)

“Slots” :

Ce sous-élément des panels est développé à part parce qu'il est assez complexe.  
Il peut être soit une entrée ou une sortie d'item, interactif ou non.

Une entrée possède un menu dont les items sont des vues de l'inventaire du vaisseau, les vues à l'intérieur sont sélectionnées automatiquement à partir des recettes du scriptableObject de la machine. Le menu est utilisé par le joueur pour sélectionner un item d'une recette.

Une sortie est un item stocké dans le slot qui n'est pas dans l'inventaire du vaisseau, il est récupérable en cliquant dessus et sera transféré dans l'inventaire,

Les clics sur un slot sont détectés par un bouton invisible, en désactivant l'interactivité du slot, le menu ou la récupération sera désactivé et le slot permettra juste d'afficher et de détecter les clics dessus.



ScriptableObjects :

le “scriptableObjects” des machines permet de délocaliser toutes les constantes liées à une machine précise, pour pouvoir créer une classe la plus générique possible.

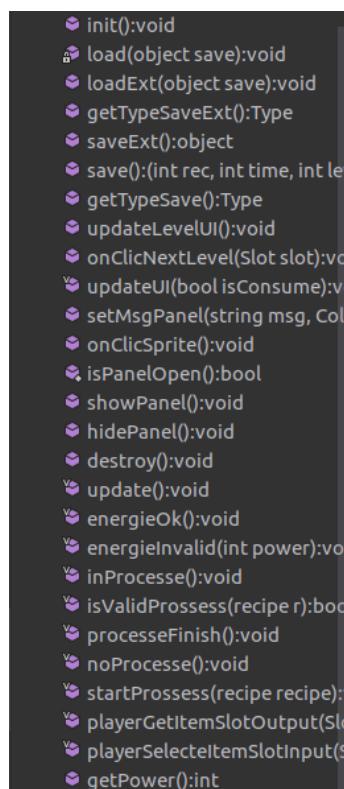
Les constantes présentent à l'intérieur sont :

- un identifiant unique
- l'item lié à la machine
- les niveaux :
  - une liste d'item pour passer le niveau
  - l'énergie requise pour passer le niveau
  - la puissance quand la machine est à l'arrêt
- les recettes :
  - la liste des items nécessaire pour faire la recette
  - l'item produit
  - le temps de production
  - la puissance lors du traitement
  - le niveau de la machine nécessaire
- le nom de la machine
- l'image de la machine dans le vaisseau
- la position et l'agrandissement
- le type de la classe qui va gérer la machine (c'est soit la classe Machine ou une sous classe de celle-ci)
- le préfab du panel

## Fonctionnement interne :

Les instances de machine sont générées automatiquement via la classe “Activator” du C# en utilisant le nom de la classe à utiliser dans le scriptableObjects.

La classe Machine est découpée en une trentaine de sous-fonctions “virtual” pour pouvoir les écrire dans des sous-classes comme celle du générateur pour le réacteur. Chaque aspect peut être modifié sans devoir tout réécrire, elle utilise le polymorphisme de la POO.



## 6 Comment jouer au jeu

Dans cette partie, l'entièreté des fonctionnalités du jeu sera expliquée pour permettre à toute personne n'ayant jamais joué à ce jeu de pouvoir avancer et même terminer le jeu sans trop en dire non plus car il est important de garder certaines surprises dans un jeu.

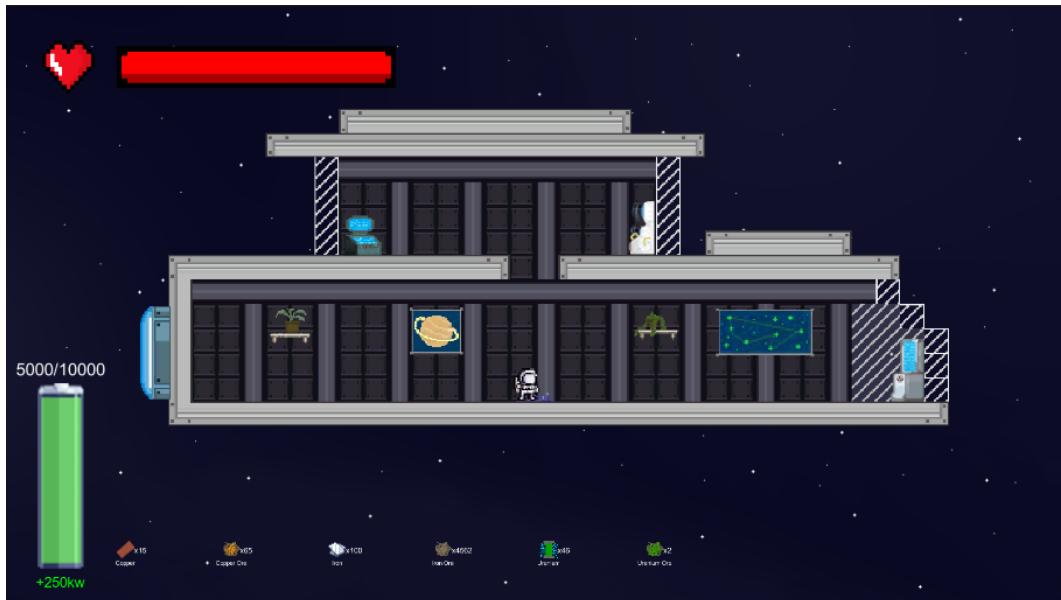
Premièrement, passons rapidement en revu le menu principal étant donné que ce dernier est déjà présenté dans les tâches personnelles de ce rapport. Lorsque le joueur lance le jeu, il se retrouve directement sur le menu principal qui est composé de 4 boutons. Le premier permettant de lancer une partie, le second permet d'accéder aux options, le 3ème d'accéder aux crédits et enfin le boutons pour quitter.

Intéressons nous au premier bouton permettant de lancer une nouvelle partie. Lorsque vous appuierez sur ce bouton, vous obtiendrez la fenêtre ci-dessous :



Vous pouvez voir ici divers boutons dont le bouton load qui permet de charger une des sauvegardes ci-dessus ou encore le save over qui permet d'écraser une sauvegarde, delete pour supprimer les sauvegardes afin que vous ne vous retrouviez pas avec une montagne de sauvegardes inutiles pour beaucoup. Et enfin le bouton NEW GAME qui va désormais nous intéresser.

Maintenant, après avoir appuyé sur ce bouton vous allez vous retrouver dans la situation suivante dont nous vous donnons une capture d'écran ci-dessous :



Sur cette image, vous pouvez voir diverses choses, pour commencer, vous pouvez voir votre barre de vie en haut à gauche représentée par la barre rouge et le cœur. En bas à gauche vous pouvez observer la barre d'énergie en vert avec sa capacité ainsi que son débit d'utilisation. Pour la barre, nous reviendrons plus en détail sur cette dernière lorsque nous parlerons des machines.

Maintenant, passons aux six icônes que vous pouvez voir en bas à droite de la barre d'énergie. Ces derniers représentent les six ressources du jeu (le cuivre en lingot et mineraï et la même chose pour le fer et l'uranium). Ces derniers seront utiles pour réussir les quêtes mais aussi pour construire et faire fonctionner les machines. On peut voir à la droite de chacun des objets leur nombre et en dessous leur nom.

Maintenant passons au personnage, comme dit précédemment le joueur possède sa barre de vie en haut à gauche mais surtout divers contrôles.

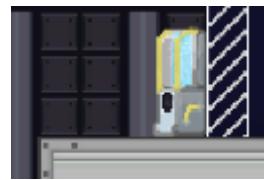
Pour commencer le joueur peut aller à droite et à gauche grâce aux flèches directionnelles droite et gauche. Lorsque le joueur se trouve près d'un monstre, ce dernier peut attaquer en utilisant la touche P. Pour finir, le joueur peut utiliser le portail violet en bas au milieu du vaisseau avec la touche "U" pour se diriger vers l'étage. Pour finir les autres touches nécessaires à des actions vous seront données en jeu.

Nous allons désormais passer au déroulement du jeu, quand vous serez arrivé dans la partie vous serez automatiquement mis au niveau 0, pour commencer il vous suffit d'aller tout à droite du vaisseau en bas pour ouvrir l'écran de carte.



Après avoir appuyer sur la touche M vous allez observer les diverses planètes représentant chacune un niveau, du niveau 1 au niveau 4. Pour activer un niveau, il vous suffit d'appuyer sur le bouton situé en dessous de la planète. Lorsque vous aurez fait cela , soyez réactif car à chaque lancement de niveau , des monstres seront à vos trousses.

A partir du niveau 1 quand vous aurez les monstres, il vous sera désormais possible d'aller miner en utilisant la touche B sur la station de minage ci-dessous.



Une fois à côté de la station, il vous sera possible de miner et d'obtenir du cuivre , du fer et de l'uranium en fonction des planètes.

Maintenant passons en revue la partie la plus importante et qui vous permettra d'avancer dans le jeu, c'est-à-dire les missions. Pour ceci, il vous faudra ouvrir la machine ci-dessous avec la touche T.



Après avoir appuyé sur la touche "T" vous pouvez désormais observer l'interface de mission. Chaque niveau sera constitué de 4 quêtes avec un bouton claim à la droite de chacune d'entre elle qui permettra au joueur de pouvoir récupérer sa récompense mais aussi de savoir si sa mission est complétée, pour cela il suffit de voir si le bouton est vert ou rouge, le rouge représentant une mission non validée et le vert une mission validée.

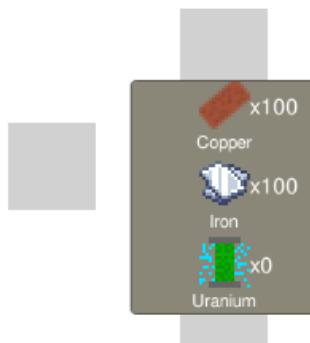
Pour les 3 premiers niveaux, tel est le schéma que vous aurez à suivre à chaque fois. Cependant, le dernier niveau est particulier car il y a 2 autres boutons, l'un permettant de faire apparaître le boss du jeu qui est en fait un grand monstre faisant bien plus de dégâts que ses prédecesseurs et qui ne peut être appelé que si toutes les missions autre que celle de tuer le boss sont accomplies.

Enfin dès que le boss sera vaincu et les tâches donc terminées il sera possible d'appuyer sur le bouton "Return To Earth" qui est tout simplement la dernière action que vous ferez dans le jeu car c'est le bouton de fin de jeu qui vous amènera directement aux crédits.

Pour finir, nous allons nous intéresser aux machines qui sont au nombre de 3 et qui ne seront pas disponibles en début de partie, il vous faudra les construire. La première appelée "Assembly Table" qui permettra de construire les machines suivantes, la seconde appelée "furnace" qui vous permettra de transformer en lingot les minerais que vous minerez et enfin le réacteur qui sera nécessaire à l'alimentation des autres machines. Il vous sera fourni une certaine quantité d'énergie au début du jeu pour faire fonctionner les autres machines le temps d'obtenir le réacteur. Voici l'exemple de l'interface d'une machine ci-dessous, en l'occurrence l' "assembly table" car celle-ci est la plus dure à utiliser de toutes les machines que vous pourrez voir au fil du jeu.



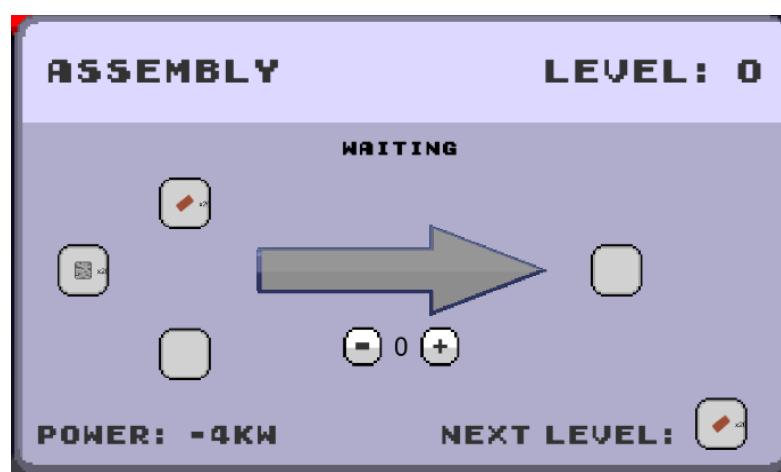
Comme vous le voyez, à partir d'éléments présents à gauche, nous pouvons obtenir diverses choses à droite et notamment des machines. La barre de progression représente le temps qu'il faudra pour que la fabrication ait lieu. Pour finir, les boutons + et - permettent quant à eux de pouvoir choisir le nombre d'objets que l'on souhaite créer à la suite. Pour choisir l'objet à sélectionner, il suffira de cliquer sur une case à gauche et par la suite l'objet que l'on veut y insérer. Voici un aperçu du procédé de choix d'objet ci-dessus. Pour information, les machines sont accessibles par un simple clic gauche sur cette dernière.



Maintenant que nous avons vu qu'il était possible de faire des machines grâce à “l’assembly table”, voyons maintenant la façon dont nous les fabriquons.

Premièrement, voyons la fabrication du four électrique, ci-dessous vous pouvez voir en haut à gauche “LEVEL: 0” , ce qui signifie que la machine est au level 0. C'est le niveau minimum mais suffisant pour obtenir le four , ce qui n'est pas le cas du réacteur.

Ensuite comme vous pouvez le voir il faut 20 de cuivre et 20 de pierre pour le fabriquer puis ensuite cliquer une seule fois sur le bouton “+”. Pour finir vous avez en bas à gauche la consommation d'énergie par seconde.



Maintenant pour le réacteur il est nécessaire d'augmenter le niveau de l'assembly table pour le fabriquer en cliquant en bas à droite, ce qui coûte 20 de cuivre. Pour ce qui est des éléments requis, il suffit de remplacer les 20 de pierres par 20 de fer et d'ajouter 5 d'uranium sur le dernier emplacement.

Avant d'en terminer avec les règles, sachez que certaines choses n'ont pas été dites ici-même pour laisser un temps soi peu de surprise au joueur. Bien sûr, ces “choses” qui n'ont pas été dites seront expliquées dans le jeu au moment voulu. Sachez également qu'une mauvaise gestion de l'énergie menant à un état où vous ne pouvez plus en produire ou que vous n'en stockez plus vous oblige à recharger une ancienne sauvegarde si existante ou alors de recommencer une partie complète. Par ailleurs, chaque machine ne doit être construite qu'une seule fois, dans le cas contraire cela pourrait compromettre la partie aussi.

## 7 Potentiels Retards

Au début du projet nous avions fourni un cahier des charges résumant les diverses idées que nous avions pour notre jeu mais aussi le temps que nous avions prévu pour leur réalisation. Certes, ce dernier est au final plutôt bien suivi dans sa globalité mais certaines choses ont changé. Par exemple, le multijoueur qui devait être une tâche parmi les autres a fini par être exclu du projet. Mais nous avons également eu des problèmes sur les diverses soutenances sur notre planning de réalisation que nous allons vous rappeler ci-dessous en fonction des 3 soutenances.

### **1ère soutenance :**

Par rapport aux prévisions présentées dans le cahier des charges, nous pouvons noter quelques retards dans les domaines suivants :

- La gestion du multijoueur
- La gestion des missions

Nous pensions à la base implémenter le début du multijoueur de notre projet avant la première soutenance, mais finalement nous nous sommes confortés dans l'idée qu'il valait mieux faire fonctionner le reste, c'est-à dire la partie solo avant d'étudier la question d'un potentiel mode multijoueur. Pour les missions, nous préférions commencer ces dernières lorsque le système de carte et les machines seront opérationnelles. Malgré cela les idées de bases sont déjà là, notamment pour les missions dont nous avons déjà une claire idée de la façon de les faire et surtout en quoi elles vont consister et ce qu'elles vont apporter dans le contexte du jeu.

Cependant, malgré certains retards, nous avons pris beaucoup d'avance dans certains domaines dont les suivants en particulier :

- Gestions des sauvegardes
- Gestion des personnages
- Le site internet

Le site internet ne devait être créé que pour la seconde soutenance en grande partie, mais finalement nous avons trouvé le temps de le faire, mais ce dernier n'est pas encore terminé. Pour ce qui est du personnage, l'implémentation de la barre de vie n'était pas prévue pour tout de suite, mais nous l'avons finalement fait ainsi qu'une barre d'oxygène qui quand elle est vide fait baisser la vie du personnage. Bien entendu, cela n'est qu'une partie de ce qui devra être fait pour finaliser le jeu.

### **2ème soutenance :**

Par rapport aux prévisions présentées dans le cahier des charges, nous pouvons noter quelques retards dans les domaines suivants :

- La gestion des cinématiques
- La gestion des missions
- La gestion du multijoueur

Nous pensions à la base implémenter le début du multijoueur de notre projet avant la deuxième soutenance, mais finalement nous nous sommes confortés dans l'idée qu'il valait mieux faire fonctionner le reste, c'est-à dire la partie solo avant d'étudier la question d'un potentiel mode multijoueur. Pour les missions, nous avons réalisé les missions en pseudo-code. C'est-à-dire que toutes les conditions sont là, mais qu'il manque encore l'implémentation de ces dernières due à des problèmes. Notamment le fait que nous avons un bug qui fait que la machine gérant les missions ne s'affiche pas ou encore que pour une raison inconnue nous n'arrivons pas à obtenir le nombre d'item dans le code de la machine. Et pour finir, les cinématiques n'étant qu'un plus dans le jeu, nous avons pensé que ces dernières devraient plutôt être implémentées à la fin lorsque le reste sera complètement opérationnel.

Cependant, malgré certains retards, nous avons pris beaucoup d'avance dans d'autres domaines dont les suivants en particulier :

- Personnages
- Structure du vaisseau
- Interface carte

### **3ème soutenance :**

Comme on peut le voir sur le résumé des 2 dernières soutenances, quelques retards peuvent être visibles. Par conséquent nous avons dû rattraper plusieurs de ces et nous trouvons que nous avons globalement réussi même si certaines choses notamment sur l'aspect graphique du jeu n'ont pas pu être implémentées comme une vidéo de présentation de notre monde ou pouvoir se rendre réellement sur d'autres planètes. Hormis le multijoueur qui de son côté a été simplement supprimé du projet il n'y a pas de vrai retard sur les fonctionnalités en elles-mêmes.

Pourtant, nous sommes tout de même déçus car nous aurions voulu implémenter plus de missions et de machines. Il y a plusieurs causes à cette déception.

Premièrement, les missions ont été plutôt longues à mettre en place car il y a avait de nombreux éléments graphiques à gérer mais aussi des idées de missions à trouver et aussi à gérer leur accomplissement et leur détection. Au-delà de la longueur de leur implémentation, nous avons aussi été confrontés à des bugs, comme la gestion des objets nécessaires aux missions dont les “id” que nous utilisons pour détecter de quel objet il s’agit qui était mal dans le désordre. Par conséquent, tous les codes ont dû être modifiés pour rectifier le tir. Le second problème est également un problème de rédaction de code. Le second bug venait d’un problème de scène qui ne change pas lorsque l’on exécute la commande car celles-ci n’avait en fait pas été ajouté aux “builsettings”, ce qui fait qu’après le lancement du jeu cette dernière ne s’activait jamais. Ce problème était simple à régler mais il nous a fallu du temps pour comprendre d’où le problème venait.

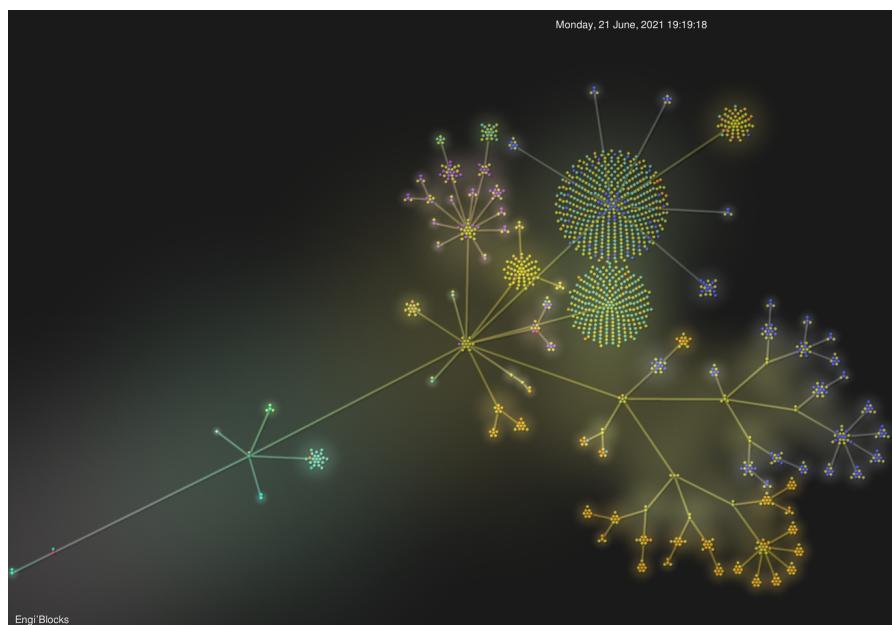
Ensuite, quand il fut temps de commencer la conception des diverses classes nécessaire à l’implémentation de nos machines, cela a créé bien des bugs dans les sauvegardes mais aussi dans la scène principale de notre jeu.

Le second problème lié aux machines provient de la façon dont nous avons voulu concevoir le code. Le but était d’implémenter tout cela de façon dynamique, notamment pour une question de performance mais l’implémentation dynamique est un procédé très long , par conséquent cela nous a pris une grande partie de notre temps.

Nous savons qu’il y a des manières de le faire plus rapidement grâce à unity mais nous n’avions pas le temps d’apprendre ces dernières là où nous pouvions le faire en C sharp avec nos propres connaissances acquises au fil de l’année.

## 8 Structure du répertoire

La structure du répertoire Git, c'est-à-dire du projet, est représentée par cet arbre ci-dessous. Cette visualisation a été générée à l'aide de l'outil Gource à partir de l'historique de modification Git. Chaque branche représente un dossier, chaque élément représente un fichier et chaque couleur représente un type de fichier.



## **9 Ressentis personnels**

### Antoine Adam :

Le projet a beaucoup évolué sur la dernière soutenance, j'avais sous-estimé le travail pour créer des machines à partir de “ScriptableObject”. Il me manquait surtout du temps que j'ai fini par avoir quand les cours se sont terminés. J'ai beaucoup travaillé durant la dernière semaine sans savoir si j'allais pouvoir finir à temps, mais heureusement, j'ai réussi. J'ai perdu du temps à refaire ce que j'avais fait en début d'année parce que je n'avais pas vraiment de repère sur ce qu'il me faudrait des mois plus tard. Je suis fier de ce qu'on a accompli dans ce jeu.

### Alexandre Dugot :

Nous en sommes enfin à la fin de ce projet qui a commencé en décembre. Je m'attendais à ce que ce soit long et plutôt long et difficile mais sûrement pas à ce qu'il y ait autant de bug à résoudre. Cela montre donc que le métier de codeur n'est pas seulement de coder des fonctionnalités mais qu'il faut un temps considérable pour déboguer les divers programmes créés. Malgré cela, étant quelqu'un ce projet est un avantage dans quasiment tous ses aspects, que ce soit le fait de créer son propre jeu au lieu de jouer en permanence à d'autre jeu mais aussi d'apprendre le Csharp d'une manière différente de ce que l'on peut faire en tp de programmation dans un gros projet de manière je trouve, plus amusante.

### Mohamed Ghaffour :

Je dois avouer que lors de la préparation de ce projet j'ai rencontré des problèmes dont j'ignorais la connaissance. En effet j'ai compris qu'il était parfois préférable de se détacher de son idée de départ et de lire les documentations afin de progresser. Ce projet a été pour moi une réelle opportunité de progresser, en effet je sens une réelle différence sur la compréhension et la manipulation de l'orienté objet en C#. J'ai également appris à manipuler l'HTML et le CSS, ce qui a été pour moi une grande découverte.

Au-delà de la programmation, ce projet m'a énormément aidé sur l'aspect communication et esprit d'équipe. En effet, le groupe a connu des hauts et des bas et je pense que la communication est l'une des règles d'or lors d'un travail de groupe.

Nathan Labernardiere :

Cette expérience a été pour moi très positive. Au départ j'appréhendais un peu ce projet car en début d'année j'avais très peu d'expérience en informatique et nous n'avions pas encore réalisé beaucoup de TP en C#. C'est pourquoi j'ai rapidement voulu me renseigner, me documenter avec ce que l'on peut trouver sur Internet et j'ai pu trouver beaucoup de ressources pour me guider. J'ai par la suite commencé à prendre de l'aisance sur ce projet et par la même occasion commencé à rencontrer les premiers bugs et problèmes à affronter. Pour certaines tâches je ne m'imaginais pas du tout que cela pourrait être aussi dur et qu'il fallait consacrer beaucoup de temps pour obtenir un résultat satisfaisant. Malgré tout cela, je prenais beaucoup de plaisir à m'investir dans ce travail. Ainsi, au final je suis satisfait de notre travail fourni, de ce que nous avons réussi à produire et aussi des liens que nous avons réussi à souder pour former une bonne équipe. Je n'hésiterai pas à recommencer une expérience similaire !

## 10 Conclusion

Premièrement, nous sommes heureux de nous dire que nous sommes passés d'un jeu encore très superficiel à un jeu totalement terminé dont les bugs sont peu nombreux et essentiellement graphiques.

Après de longues heures à coder ce jeu, cela est vraiment réjouissant de le voir arriver à son terme. Nous avons tous compris à travers ce jeu à quel point le métier de codeur dans le jeu vidéo peut être long et éprouvant mais surtout que ces derniers passent la majorité de leur temps à régler des bugs plutôt qu'à coder le jeu en lui-même. Malgré cela, étant ce projet est un avantage dans quasiment tous ses aspects, que ce soit le fait de créer son propre jeu au lieu de jouer en permanence à d'autre jeu mais aussi d'apprendre le Csharp d'une manière différente de ce que l'on peut faire en tp de programmation dans un gros projet de manière plus amusante nous trouvons.

Pour finir nous sommes conscients que notre façon de coder n'est pas encore très poussée pour un jeu vidéo mais cela nous a donné de nombreuses bases, que ce soit pour des projets personnels ou liés à EPITA. Enfin, nous pensons que ce genre de projet est très utile dans le sens où il nous projette dans un des nombreux univers de la programmation, nous donnant possiblement des pistes sur ce que nous aimerais faire à l'avenir.