

SpaceY

Rapport de Soutenance Finale



By Epit'chad

Maurin AUDENARD (Chef de Projet)

Madin AMRANE

Maxandre BERSON-LEFUEL

Théodore BECHETOILLE

Olaf PINIARSKI

Repo complet : spacey-epitchad.github.io

Site internet :

<https://spacey-epitchad.github.io/>

Cadre : Projet de 1ère année EPITA

Table des matières

1	Introduction	4
2	Vue générale sur le projet et l'équipe	5
2.1	Les membres du groupe	5
2.2	C'est quoi ce projet?	6
2.3	Répartition des tâches	7
2.4	L'Organisation de ce rapport.	9
2.5	Outils et logiciels utilisés	10
3	Le Commencement	12
3.1	Initialisation de la map	12
3.1.1	Création des premières interactions	12
3.1.2	Tilemap	12
3.2	Génération de la map	12
3.2.1	Composition basique	13
3.2.2	Génération des astéroïdes	15
3.3	Personne jouable :	15
3.4	Les Astéroïdes	16
3.4.1	Le processus de génération	16
3.4.2	Les ressources	16
3.4.3	Station de passage	17
3.5	Launcher	18
3.5.1	Instance & initialisation	18
3.6	Base de donnée locale	18
3.7	Site internet	18
4	L'Âge d'or	19
4.1	La fusée	19
4.1.1	Script d'initialisation	19
4.1.2	Le mécanisme	20
4.1.3	Détection des Astéroïdes	21
4.1.4	Avancement	22
4.1.5	Bilan actuel des fusées	22
4.2	Multijoueur	23
4.2.1	Initialisation	23
4.2.2	Game ID	23
4.2.3	Signaux	23
4.2.4	Peer-to-peer	24
4.3	Synchronisation multijoueur	24
4.4	Les lasers	24
4.4.1	Création du laser	24
4.4.2	Variables de tir	25
4.5	Lobby d'attente	25
4.5.1	Initialisation	26
4.5.2	Mini-jeu shooter	26
4.6	Interface graphique & menus	27
4.6.1	Fonctionnement des menus	27

5 L'Ère de l'inaction	28
5.1 C'est quoi l'ère de l'inaction ?	28
5.2 Aspect visuel et graphique du jeu	28
5.2.1 Les équipes :	28
5.2.2 Personnages et déplacements :	29
5.2.3 Fusée, laser et stations de passages :	29
5.2.4 Astéroïdes :	31
5.2.5 Aliens :	31
5.2.6 Emoticône :	32
5.3 Ligne d'arrivée	34
6 Destination Finale	35
6.1 Récolte et utilisation des ressources :	35
6.1.1 La récolte :	35
Joueur :	35
Développement :	35
Les problèmes rencontrés :	36
6.1.2 Les utilisations :	37
6.2 Le retour de la fusée	37
6.3 Les aliens & leur intelligence artificielle	38
6.3.1 Association d'une équipe	38
6.3.2 Identité visuelle	38
6.3.3 Pathfinding A*	39
6.3.4 Interactions	39
6.3.5 Complications	39
6.4 Tutoriel	40
6.4.1 Conception du tutoriel :	40
6.4.2 Les PNJs :	41
6.4.3 Script	42
6.5 Interface utilisateur :	42
6.6 Système Sonore	44
7 Bilan du projet :	46
8 Conclusion	48

Introduction

Nous sommes ravis de vous présenter notre projet dans le cadre de notre deuxième semestre à Epita nommé "**Space Y**".

Il s'agit d'un jeu de course et de stratégie, réalisé sur le moteur Godot, où deux équipes s'affrontent afin d'atteindre en premier la ligne d'arrivée, la lune. Les joueurs incarnent alors des astronautes qui ont pour but de faire avancer leur fusée jusqu'à cette dernière.

Ils devront surmonter différentes difficultés sous plusieurs formes qui leur empêcheront d'atteindre leur objectif.

Nous vous proposons alors d'entrer dans les coulisses du développement de notre jeu et de découvrir les différentes étapes de sa réalisation.

Vue générale sur le projet et l'équipe

2.1 Les membres du groupe

- **Madin** : Ayant grandi dans le monde du jeu vidéo, je me suis naturellement tourné vers l'informatique, une passion qui s'est renforcée grâce à ma participation à ce projet. Bien que ce projet n'ait pas été de tout repos, il m'a beaucoup appris. Mes collègues pensent que je passe trop de temps à perfectionner les couleurs, mais j'appellerais ça "travailler l'identité visuelle". C'est d'ailleurs moi qui ai pris en charge la majorité des tâches liées au visuel du jeu.
- **Maurin** : Même en étant chef de projet malgré moi (car mes camarades se sont dégonflés), j'ai adoré travailler sur SpaceY. C'est mon tout premier si gros projet, même si je développais avant. J'ai fait beaucoup de choses différentes lors de la programmation. À la fin, je me suis occupé de plein de petites parties. On m'appelait "**Terminator**". Je pense avoir eu les tâches les plus intéressantes, contrairement à d'autres. (•o•).
- **Maxandre** : Aussi appelé "l'ombre", mais pas pour les mêmes raisons que d'autres ; mon travail n'est pas visible par de l'animation ou du développement d'interactions. En effet, je me suis occupé des parties immergées du projet telles que le multijoueur. Ayant eu un rôle important dans ce projet, j'en retiens une expérience enrichissante dans le monde du développement de jeux vidéo.
- **Théodore** : Nouveau dans le domaine de l'informatique, le projet SpaceY était une expérience nouvelle pour moi. Bien que ma maîtrise en informatique soit incertaine, j'ai cependant apprécié de participer à ce projet, ce qui m'a permis d'acquérir de nouvelles compétences, notamment dans la conception des conditions de victoire du jeu. Ce fut un agréable voyage informatique passé avec de vrais amis.
- **Olaf** : Malgré un bagage technique et un niveau en développement de jeu vidéo inférieur à celui de mes collègues j'ai adoré contribuer au développement de SpaceY. De plus SpaceY m'a permis d'acquérir des compétences techniques notamment lors du développement de l'interface graphique du jeu mais également des compétences générales lors de la conceptualisation et communication de ce projet.

2.2 C'est quoi ce projet?

Lorsque notre groupe s'est réuni pour la première fois, nous avions des idées assez différentes les unes des autres. Mais notre objectif était clair : nous voulions un jeu avec une grande rejouabilité, et qui ne serait pas horrible à réaliser. Non pas par manque d'ambition ou par paresse, mais par pragmatisme. Nous ne voulions pas faire de fausses promesses lors de nos soutenances et de notre cahier des charges, vendre un jeu avec d'innombrables fonctionnalités, un nombre incalculable de mondes et de boss, mais qui au final ne pourrait pas être réalisé par manque de temps.

Notre objectif personnel était qu'à la fin de cette année et au moment de rendre notre projet, nous n'ayons pas un goût amer parce que nous n'avons pas atteint notre objectif, mais que nous pourrions présenter un jeu complet qui ressemble exactement à ce que nous avions vendu et imaginé, voire mieux.

C'est ainsi qu'est né **Space Y**, un jeu qui convenait parfaitement à ce que nous avions imaginé, un jeu en ligne avec une très grande rejouabilité grâce à sa nature de jeu multijoueur et à sa génération d'espace aléatoire qui rend chaque partie unique ! De plus, c'est un jeu avec des fonctionnalités assez simples à prendre en main, parfait pour introduire de nouveaux joueurs ou pour que des joueurs peu aguerris y prennent du plaisir, tout en gardant une dimension compétitive pour ceux qui voudraient jouer de manière plus sérieuse. Nous nous sommes alors basés sur le principe de *easy to learn, hard to master*.

Mais faire un **party game** qui peut prendre une dimension compétitive ne nous suffisait pas. Nous voulions du sang, nous voulions créer des souvenirs dignes de jeux comme *Mario Kart* où les joueurs, pour atteindre leur but, utiliseront tous les moyens mis à disposition pour empêcher l'avancement de leurs adversaires, et qu'ils ressentent de l'amertume lorsqu'ils échoueront. Car c'est dans la difficulté qu'on s'amuse le plus.

Nous allons brièvement présenter notre jeu, mais nous nous attarderons sur chaque détail de sa conception plus tard dans ce rapport. **Space Y** est un jeu de course qui se déroule dans l'espace où les joueurs doivent faire avancer leur fusée jusqu'à la ligne d'arrivée. Pour ce faire, les joueurs devront récolter différentes ressources sur les astéroïdes disposés un peu partout dans l'espace de jeu : le fer et le cristal. Le fer sert à créer des points d'accès qui doivent être posés sur les astéroïdes pour faire avancer la fusée, et les cristaux servent à tirer des lasers afin d'entraver l'avancée de leurs adversaires. Il y aura également une menace extérieure, qui ne viendra d'aucune équipe et qui n'est pas humaine : des aliens, qui auront pour simple but de détruire les vaisseaux des joueurs, ce qui leur ferait alors perdre la partie.

2.3 Répartition des tâches

Maurin :

- Map : rendu visuel, tilemap (disposition des astéroïdes), script de génération aléatoire des astéroïdes (pourcentage et taux d'apparition), gestion de la ligne d'arrivée.
- Fusée : création de la structure, script de gestion, pathfinding : détection des stations proches.
- Astéroïdes : création de la structure du nœud, script de gestion.
- Stations (waypoints) : création de la structure du nœud, script de gestion.
- Interactions des joueurs avec les stations, pose et suppression.
- Dimension visuelle : génération des différents backgrounds.
- Interface lobby : mini-jeu (shooter) avec Maxandre.
- Design des emotes avec Madin.
- Site internet : réalisation des textes.
- Joueur : barre de progression du minage et de la pose des stations.
- Paramètres sonores : barre de volumes musique et SFX (sauvegarde dans la base de données locale).
- Dimension sonore : système global de gestion des bruitages d'ambiance, sources sonores en fond, musique de jeu.
- Aliens : Attaque de la fusée à un intervalle régulier (interactions avec les ressources).

Madin :

- Interface du jeu (affichage des ressources, synchronisation entre les clients) : script de gestion, création du nœud, rendu visuel.
 - Joueur : script de gestion du joueur, animation du joueur (rotations, flammes, suspension), emotes (animations, synchronisation multijoueur, gestion des emotes d'équipes), gestion des interactions visuelles avec les entités de la carte (paralysie, minage, interface UI), déplacements du joueur.
 - Lasers : animation du laser, interactions des collisions sur les fusées.
 - Ressources : récolte de ressources, dépôt des ressources dans la fusée, conditions.
 - Fusée : système de réparation de la fusée.
 - Tutoriel : rédaction des textes et du scénario, prise des captures d'écran.
 - Dimension visuelle : assets trouvés (aliens, astéroïdes, fumée, lasers, ressources, astronautes, divers) et réalisations (emotes, NPCs (colonel, robot, petite fille, mineur)).
-

- Site internet : réalisation des textes.

Maxandre :

- Aliens : Pathfinding (A*), apparition sur la map, script de gestion.
- Map : rendu visuel, bordures et collisions, apparition des entités, tilemap (disposition des astéroïdes), script d'apparition des astéroïdes.
- Ligne d'arrivée : script de gestion des conditions de victoire.
- Joueur : synchronisation des entités en multijoueur (affichage du pseudo, indicateur d'utilisation, synchronisation des caméras).
- Fusée : création de la structure, script de gestion (instanciation des fusées, gestion des équipes).
- Lasers : création de la structure du nœud, script de gestion (collisions, synchronisation des instances chez les clients), interactions des collisions sur les aliens.
- Interface graphique : menus et animations (page d'accueil, paramètres, création d'une partie).
- Interface lobby : partage des données de connexion, affichage des joueurs connectés, mini-jeu (shooter).
- Multijoueur : création du serveur, partage de l'adresse IP, chiffrement de l'adresse IP, création des instances du client, signaux multijoueurs (connexions, déconnexions).
- Tutoriel : création de la structure du nœud et script de gestion, prise des captures d'écran.
- Launcher : instantiation de la base de données locale.
- Base de données locale : sauvegarde des variables de paramétrage (pseudo, sons, etc.) en JSON.
- Dimension visuelle : assets trouvés (boutons, polices d'écriture, curseur, divers).
- Site internet : développement du site, réalisation des textes.

Théodore :

- Ligne d'arrivée : affichage du message de fin, script de gestion des conditions de victoire avec Maxandre.

Olaf :

- Interface graphique : création de menus incompatibles avec le jeu, repris par Maxandre.
-

2.4 L'Organisation de ce rapport.

Ce rapport est divisé en quatre grandes étapes énumérées dans l'ordre chronologique :

1. Le Commencement :Le commencement est là première étape de notre développement, elle s'étend sur la majeure partie du mois de décembre. À ce moment-là, aucune partie ou fonctionnalité n'était reliée à une autre. Ce n'étaient que des fonctionnalités individuelles, rien qui ne ressemblait à un jeu.
2. L'âge d'or : L'âge d'or est sûrement la partie la plus productive qu'ait pu voir SpaceY, elle s'est déroulée de la fin du mois de décembre jusqu'au début du B3 à la fin du mois de janvier. Nous étions en vacances et c'est à ce moment que les pièces commençaient à s'assembler ensemble.
3. L'ère de l'inaction : C'est la période où nous avons été le moins productifs, s'étendant du mois de février à la mi-avril. C'est un peu un désert de l'avancement.
4. La destination finale : La destination finale est, comme son nom l'indique, la partie finale du développement, démarrant de la fin de l'ère de l'inaction jusqu'à notre soutenance finale. C'est le moment où beaucoup de choses sont terminées ou déboguées.

Chaque étape, séparée par les différents chapitres, contient ce qui a été fait lors de cette étape afin de donner une dimension temporelle à ce rapport.

2.5 Outils et logiciels utilisés

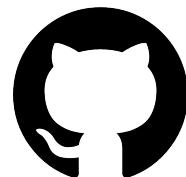
Tout au long de l'année, notre groupe de projet a utilisé divers outils pour développer notre jeu vidéo, chacun ayant une utilité spécifique et essentielle à notre progression.

Discord a été notre principal moyen de communication.

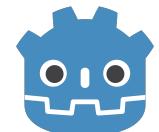
Nous l'avons utilisé non seulement pour discuter en temps réel, mais aussi pour transférer des ressources graphiques et sonores. Les appels vocaux nous ont permis de collaborer plus efficacement, tandis que les discussions de groupe ont facilité l'entraide et la résolution rapide des problèmes techniques ou créatifs.



GitHub a été indispensable pour le stockage et la gestion de notre projet. Grâce à GitHub, nous avons pu suivre les modifications apportées par chaque membre et collaborer de manière synchronisée. Nous avons beaucoup utilisé les branches afin de programmer en même temps mais chacun sur sa version, ce qui a été vital pour le projet. Cela a également assuré la sauvegarde sécurisée de tous les fichiers du projet, nous permettant de revenir à des versions précédentes si nécessaire.



Godot a été choisi comme moteur de jeu en raison de sa flexibilité et de sa gratuité. Il nous a permis de développer et d'implémenter les mécaniques de notre jeu avec une grande efficacité. Godot est aussi un moteur de jeu très léger, ce qui est très pratique lorsque nous n'avons pas des PC particulièrement puissants, nous permettant de faire le développement dans de bonnes conditions.



Pour l'édition de code, nous avons utilisé **Visual Studio Code (VSCode)** et **Rider**. VSCode est apprécié pour sa légèreté, sa rapidité et ses nombreuses extensions qui facilitent le développement en Godot. Rider, quant à lui, est un environnement de développement intégré (IDE) robuste, particulièrement utile pour ceux d'entre nous qui préfèrent un outil tout-en-un avec des fonctionnalités avancées de refactoring et de débogage.



Procreate a été notre outil de choix pour créer certains sprites, comme les émoticônes et les personnages du tutoriel. Sa simplicité d'utilisation et ses capacités de dessin numérique en ont fait un choix idéal pour nous qui ne sommes pas graphistes,



permettant la création d'assets de bonne qualité en très peu de temps.

Enfin, nous avons utilisé plusieurs **sites** de génération et de traitement d'assets pour compléter nos ressources graphiques et sonores. Ces sites nous ont aidés à obtenir des éléments visuels et audio supplémentaires, souvent gratuits ou sous licence libre, ce qui a enrichi notre jeu sans nécessiter de création entièrement originale pour chaque asset. “

Le Commencement

3.1 Initialisation de la map

La map est un élément clé du jeu que Maurin développera en détail. De mon côté, je me suis attardé sur bordures de cette map ainsi que l'apparition des entités, de la tilemap et du script d'apparition des astéroïdes.

3.1.1 Création des premières interactions

Les bordures de la map sont des Area2D sur lesquelles nous avons attaché un nœud de collision. Ce dernier permet aux joueurs de ne pas se perdre en dehors de l'écran.

Le script de gestion des entités permet de faire apparaître une fusée pour chaque équipe sur laquelle des tags sont associés afin d'en discerner l'équipe. Ce script permet aussi de faire apparaître les joueurs (autant de petits astronautes que de joueurs connectés !) mais aussi l'apparition à intervalle régulier des aliens. Cette partie fut relativement simple si on ne compte pas la synchronisation des entités joueurs. Cette partie sera abordée plus tard dans la réalisation du multijoueur.

3.1.2 Tilemap

La tilemap fut problématique pour le début. Nous ne trouvions pas de moyen de faire apparaître des astéroïdes. Pour résoudre cela, nous avions au préalable disposé tous les astéroïdes que nous avions supprimés au fur et à mesure.

3.2 Génération de la map

Dans SpaceY, la map est la pierre angulaire du jeu. C'est d'ailleurs pour cela que nous avons décidé de la faire en tout premier.

En effet, la faire plus tard ralentirait le développement car, comme tout prend vie sur la carte, sans elle, nous n'aurions pas pu faire toutes les interactions entre les différentes parties du jeu.

Pour l'instant, la map est constituée des trois éléments suivants :

1. Le départ et l'arrivée.
2. Les limites de la map, constituées par des "colliders".
3. Les astéroïdes qui permettent la récolte des ressources ainsi que les mouvements de la fusée.

3.2.1 Composition basique

Le départ se positionne tout à gauche de la map, et l'arrivée est quant à elle positionnée tout à droite, au même endroit que l'objectif, la Lune.

Nous avons choisi ce placement, car c'est pour nous le plus naturel puisqu'il suit le sens du temps, qui est très souvent représenté de gauche à droite. Ainsi, les joueurs ont l'impression d'avancer à la fois dans leur objectif et dans le temps, ce qui est une ressource précieuse dans une course.

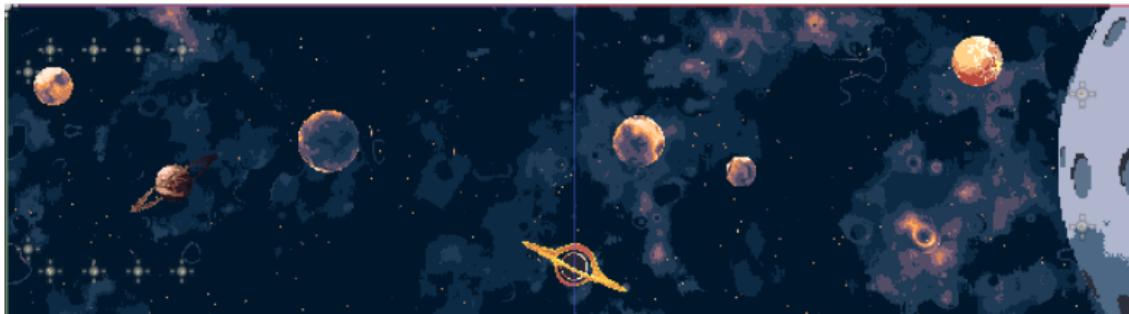


FIGURE 3.1 – Départ au début de la partie

Afin que rien ne sorte de la map, nous avons installé des colliders, des limites, qui bloquent les joueurs et les aliens trop aventureux.

Ces limites physiques sont constituées d'objets faits pour, dans Godot, ce sont les StaticBody2D.

Les StaticBody2D sont des corps, des objets physiques qui sont immuables pour les autres composantes du jeu. Ainsi, les joueurs ne pourront pas se perdre et les aliens ne pourront pas s'enfuir.

Pour que les lasers ne consomment pas trop de ressources, nous avons décidé de mettre en place un système afin de les faire disparaître dès lors qu'ils ne sont plus utiles.

Pour ce faire, nous avons mis en place une Area2D qui s'étend sur toute la zone jouable. Ainsi, lorsque les lasers sortent de cette zone, ils s'autodétruisent. Bien que ces lasers ne réservent pas beaucoup de stockage dans la mémoire, nous tenions à les gérer proprement pour que le jeu soit le moins gourmand en ressources possible.

Voici un schéma représentant la disposition de ces limites

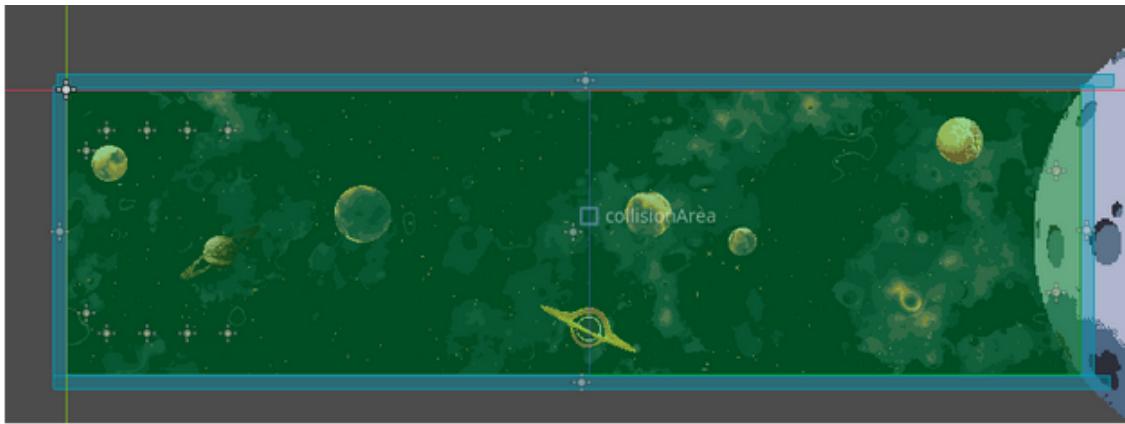


FIGURE 3.2 – Départ au début de la partie
En bleu : colliders, En vert : Area2D.

Le réceptacle des astéroïdes est l'un des éléments de la map qui a été le plus complexe à choisir.

En effet, nous avons dû nous y reprendre à plusieurs fois. Tout d'abord, nous voulions constituer un Array rempli de types d'astéroïdes, ce qui a été fait.

Cependant, après s'être renseignés sur les algorithmes de pathfinding destinés aux IA du jeu, nous nous sommes rendu compte qu'il était plus simple d'utiliser un objet Godot : une "TileMap". Les TileMap présentent beaucoup d'avantages par rapport aux Array pour notre utilisation.

En effet, elle est facile à mettre en place, facile d'accès, a une taille infinie et des réglages faciles à modifier.

Cependant, elle présente aussi des désavantages : les éléments de la TileMap sont forcément sur la grille, alors que dans le jeu, nous voudrions que les astéroïdes soient certes disposés en grille, mais pas parfaitement afin d'avoir un sentiment plus naturel face aux placements des astéroïdes.

Aussi, les actions possibles sur les éléments des TileMap ne sont pas aussi simples que celles des Array. C'est aussi le cas du parcours des TileMap, qui est très restreint. Cela a pour effet de complexifier l'accès aux éléments de la TileMap, ce qui peut être handicapant lors du développement.

3.2.2 Génération des astéroïdes

La génération des astéroïdes est complètement aléatoire. Cela lui permet d'être simple et complètement impartiale, ce qui nous paraissait important pour un jeu de course comme SpaceY, qui se concentre sur l'accessibilité.

3.3 Personne jouable :

Évidemment pour contrôler un personnage on a besoin qu'il existe.

C'est le premier nœud qui a été début le personnage n'était là que pour faire des tests, il s'agissait simplement d'une scène avec un sprite d'astronaute qu'on a ensuite fait bouger. Mais à ce niveau là le personnage n'avait rien de spécial et il n'avait pas vocation à l'être.

Son but était d'être un vaisseau pour les différentes interactions du jeu que nous développerons au fur et à mesure de ce rapport de projet (la récolte de ressource, le tir de laser...).

3.4 Les Astéroïdes

Dans cette partie, nous nous intéresserons aux astéroïdes et à leur rôle crucial dans notre jeu, notamment dans l'avancement de la fusée et sa défense.

3.4.1 Le processus de génération

La génération des astéroïdes se fait en plusieurs étapes :

1. Une seed est choisie. Comme dit précédemment, elle permet d'avoir le même aléatoire, et donc les mêmes astéroïdes entre les différents joueurs.
2. Chaque astéroïde tire un chiffre aléatoire. Ce chiffre déterminera les ressources contenues dans l'astéroïde en question et ce, pour tous les astéroïdes possibles.
3. Un autre chiffre aléatoire détermine son apparence afin qu'il y ait une variété dans l'environnement de jeu.
4. Le choix des astéroïdes pour constituer l'environnement de jeu est quant à lui donné à la génération de la map.

3.4.2 Les ressources

Comme dit à l'étape 2, afin de déterminer quelles sont les ressources contenues dans l'astéroïde, une valeur aléatoire entre 0 et 100 est tirée.

Il y a différentes possibilités d'astéroïdes selon la valeur tirée :

1. Entre 0 et 29 : L'astéroïde ne possède aucune ressource (gris).
2. Entre 30 et 59 : L'astéroïde possède du fer (orange rougeâtre).
3. Entre 60 et 100 : L'astéroïde possède du fer et du cristal (bleu).

- Mais à quoi servent donc ces ressources ?

Et bien, il s'agit de la clé de la victoire d'une équipe. Le fer sert à créer des stations de passage qui permettent de faire avancer les fusées. Il s'agit du seul moyen pour faire avancer ces dernières. Donc, sans fer, il est impossible de gagner et c'est pour cela que c'est une ressource abondante.

Le cristal, lui, permet de lancer un laser destructeur lorsque le joueur est positionné autour de sa fusée, mais nous entrerons dans les détails de ce que peut faire le laser plus tard dans une partie dédiée.

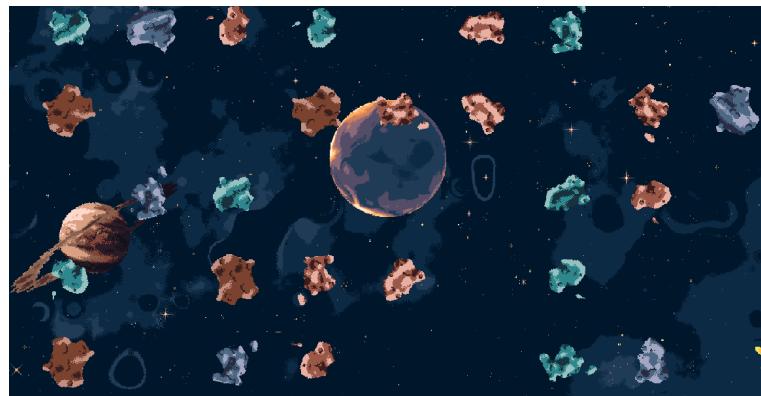


FIGURE 3.3 – Échantillon des astéroïdes possibles

3.4.3 Station de passage

La relation entre les astéroïdes et les stations de passage ne se limite pas aux matériaux. En effet, nous sommes dans l'espace, et les seules plateformes à disposition sont... les astéroïdes ! Une fois que le joueur a récupéré suffisamment de fer pour construire une station, il doit se positionner près d'un astéroïde et être dans une certaine zone (une "AreaBody2D"). Ensuite, en maintenant une touche, il pourra lancer la construction de la station.

Pour ce faire, nous avons utilisé un chronomètre (un "Timer" sur Godot) et une barre de chargement qui se remplit au fur et à mesure que le chronomètre avance, représentant ainsi l'avancement de la construction. Une fois que la barre de chargement est remplie, la station est construite. L'action inverse est également possible : appuyer sur la même touche pendant que la jauge monte à côté d'une station permet de la détruire.

À ce stade du développement, la pose des stations est gratuite car les ressources ne sont pas encore implémentées. Mais elles le seront plus tard.



FIGURE 3.4 – La barre de chargement



FIGURE 3.5 – Une station créée sur un astéroïde

3.5 Launcher

Le launcher du jeu est une scène qui n'est pas visible par le joueur. Lorsque SpaceY est lancé, le joueur est directement redirigé vers le menu principal.

3.5.1 Instance & initialisation

Mais ce qu'il ne voit pas, c'est cette scène de launcher. Elle est utilisée afin de créer la base de donnée locale si c'est la première utilisation du jeu par le joueur. Elle est aussi utilisée pour charger le curseur en forme de cible et instancier les paramètres définis au préalable par le joueur, tel que le volume sonore.

3.6 Base de donnée locale

La base de donnée locale alors instanciée va récupérer un fichier JSON contenant les informations nécessaires au jeu (paramètres comme expliqué ci-dessus) telles que le pseudo du joueur et le volume sonore. Les fonctions de chargement et de sauvegarde servent, comme leur nom l'indique, à convertir et enregistrer les dictionnaires C en format JSON dans un fichier sur l'ordinateur du joueur, et à aller chercher ces données JSON et les convertir en dictionnaires C.

3.7 Site internet

Nous avons réalisé le site internet afin d'avoir une page vitrine pour présenter le projet SpaceY. Il est réalisé entièrement en HTML et CSS, mais utilise Bootstrap pour la partie responsive sur téléphone par exemple. J'ai aussi participé à la rédaction des textes du site.

L'Âge d'or

4.1 La fusée

Dans une course de fusées, il faut des fusées !
Et des fusées contrôlables de préférence...

Voici un début de partie classique de SpaceY :



FIGURE 4.1 – Départ au début de la partie

4.1.1 Script d'initialisation

Dans SpaceY, les fusées apparaissent au départ qui se situe tout à droite. Chaque équipe a sa propre fusée. Ainsi, l'appartenance de la fusée à une équipe est signifiée par sa couleur.

Les fusées sont très espacées pour que les équipes ne se rencontrent pas tout de suite. Cela permet de prendre en main les touches et de s'organiser lors du début de la partie. L'organisation étant très importante dans SpaceY, nous devions permettre de la mettre en place en laissant du temps aux joueurs.

Le script de gestion de la fusée permet d'instancier une fusée sur la map (par le biais du script de la map), mais aussi de faire correspondre la fusée à l'équipe concernée. Pour ce faire, la fusée reçoit un tag qui indique le numéro de l'équipe, ainsi la fusée change d'elle-même son apparence en fonction de son tag.

4.1.2 Le mécanisme

Une fusée c'est bien, mais c'est encore mieux quand elle avance, surtout dans une course ! Afin de faire avancer la fusée, les joueurs peuvent poser des "Waypoints" (aussi appelés des stations de passage) sur n'importe quel astéroïde libre, qui construisent le futur itinéraire de la fusée.

Cependant, afin que la fusée se dirige vers un Waypoint, ce dernier ne doit pas être posé très loin. En effet, la fusée détecte les Waypoints qui se trouvent dans un certain périmètre autour d'elle. Cela oblige les joueurs à s'organiser pour aller chercher des ressources qui permettent de poser les Waypoints aux bons endroits, afin de faire le meilleur itinéraire.

Afin que les joueurs puissent se représenter la portée de la fusée, nous avons mis en place des pointeurs sur les astéroïdes atteignables. Les pointeurs d'une fusée ne sont visibles que pour les joueurs possédant la fusée, cela est une façon simple qui permet d'aider à la planification de l'itinéraire.



FIGURE 4.2 – Visuel des pointeurs

4.1.3 Détection des Astéroïdes

Pour ce faire, nous avons mis en place une Area2D centrée sur la fusée en permanence et une autre qui est centrée sur le Waypoint. La première est la zone de détection de la fusée et la seconde est la zone d'accessibilité au Waypoint. Ainsi, un Waypoint est considéré comme atteignable par la fusée lorsque ces deux zones se chevauchent.

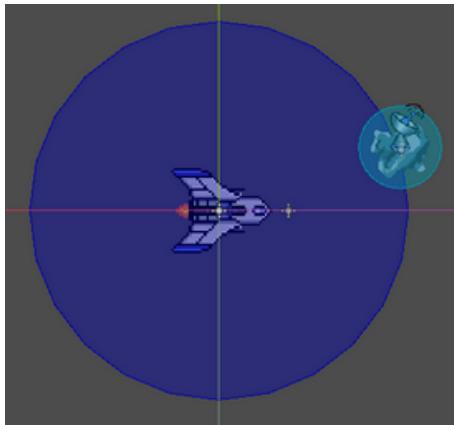


FIGURE 4.3 – Astéroïde invisible pour la fusée

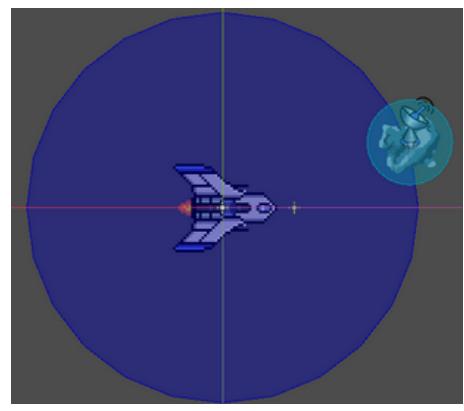


FIGURE 4.4 – Astéroïde dans le champ de détection

Le fonctionnement derrière est très simple :

1. Lorsque deux Area2D entrent en contact, la zone de la fusée émet un signal :

```
_on_checkpoint_area_entered()
```

2. Ce signal déclenche une fonction de la fusée qui prend en paramètre l'Area2D qui vient d'entrer (area) :

```
public void _on_checkpoint_area_entered/Area2D area) {
    if (area.IsInGroup($"Rocket{equipNumber}")
        && !area.IsInGroup($"AlreadyUsedStation"))
    {
        area.AddToGroup($"AlreadyUsedStation");
        if (!gotTarget)
        {
            _navigationAgent.TargetPosition = area.GlobalPosition;
            LookAt(area.GlobalPosition);
            gotTarget = true;
        }
        else
            checkpoints.Add(area);
    }
}
```

La fusée vérifie que la Station qui vient de rentrer est bien une station de son équipe grâce à la condition suivante :

```
area.IsInGroup($"Rocket{equipNumber}")
```

Et si la station entrante est bien dans son équipe, la fusée fait une dernière vérification : comme les stations ne peuvent pas être utilisées plusieurs fois, pour que la fusée s'y rende, il faut que la station ne soit pas dans le groupe "AlreadyUsedStation" et ceci est vérifié comme ceci :

```
!area.IsInGroup($"AlreadyUsedStation") )
```

Ainsi, si ces deux conditions sont remplies, la station entrante est dite "disponible". Donc si la fusée n'a pas déjà une destination, elle ira vers cette station disponible, sinon elle l'ajoute dans sa liste de stations à rejoindre.

4.1.4 Avancement

Pour faire avancer la fusée, nous utilisons une fonctionnalité inhérente à Godot : les "NavigationAgent". Ces nœuds permettent de récupérer simplement le vecteur à appliquer afin de se rapprocher de la cible. Ainsi, nous avons juste à appliquer ce vecteur sur la fusée pour faire son déplacement. Afin qu'on n'ait pas l'impression que la fusée glisse vers la cible, nous avons fait en sorte qu'elle pointe toujours vers sa cible, comme le ferait une vraie fusée ! Pour le faire, nous utilisons la fonction "LookAt()".

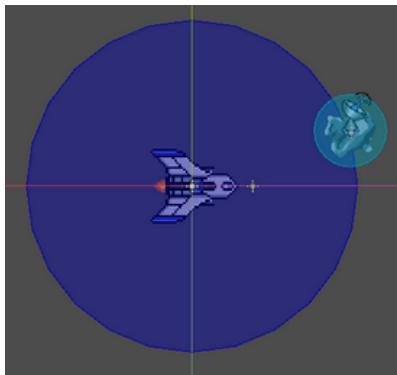


FIGURE 4.5 – Sans LookAt()

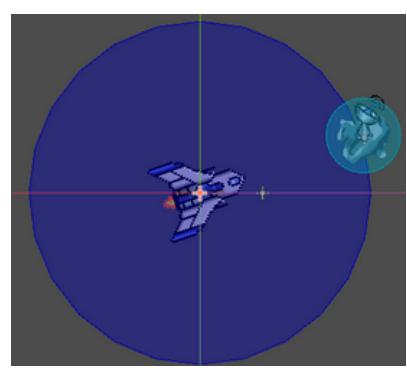


FIGURE 4.6 – Avec LookAt()

4.1.5 Bilan actuel des fusées

À ce stade du développement, la fusée est presque entièrement fonctionnelle. Il ne manque que les interactions avec les ressources, comme la transformation du fer brut en lingot de fer, ainsi que l'interaction avec le laser. Ce dernier, plus tard, fera perdre des points de vie à la fusée. Si elle n'a plus de points de vie, l'équipe adverse gagne, ce qui constitue l'un des moyens de remporter la partie.

4.2 Multijoueur

Le multijoueur fut une des parties les plus complexes à développer sur SpaceY. C'est la première fois que nous touchions au réseau et au partage de données sur un réseau local.

4.2.1 Initialisation

Pour ce faire, nous avons commencé par créer un serveur lorsqu'un joueur décide de créer une partie. Le port 136 est alors utilisé pour partager la connexion. Afin de rejoindre une partie, l'host doit partager son adresse IP, mais cette dernière est chiffrée en game ID (soit numéro d'identification de la partie). La syntaxe utilisée est la suivante :

```
(uint) IPAddress.NetworkToHostOrder((int) IPAddress.Parse(ip).Address);
```

Il s'agit simplement d'une conversion de l'IP en une suite de chiffres. De la même façon qu'elle est cryptée, elle est décryptée de la façon suivante :

```
IPAddress.Parse(ip.ToString()).ToString();
```

Cela permet de retrouver la forme originale de l'adresse.

4.2.2 Game ID

Le game ID est affiché sur l'écran de l'host, il peut alors la partager avec ses amis. De plus, nous avons ajouté une fonctionnalité permettant aux joueurs souhaitant rejoindre la partie de presser les touches CTRL+ V afin de rejoindre la partie avec l'adresse IP enregistrée dans le presse papier. Il est important de noter que l'utilisateur n'a pas besoin de placer son curseur dans le champ de texte pour réaliser cette commande.

Une fois que les joueurs ont rejoint, ils deviennent alors des clients (ou peers) connectés à l'ordinateur de l'host.

4.2.3 Signaux

Afin de recevoir les informations nécessaires, des signaux sont envoyés automatiquement à l'host et aux joueurs connectés dans la partie dès que les événements suivants se produisent :

- Connexion d'un client : affichage de l'ID du joueur dans la console
 - Déconnexion d'un client : Si l'utilisateur est l'host, le serveur est déconnecté et les clients sont envoyés vers le menu d'accueil. S'il s'agit d'un joueur, il est déconnecté, et les autres joueurs reçoivent l'information qu'ils doivent supprimer son ID de leur base de données locale. De plus, l'astronaute du joueur est supprimé de la map.
-

- Connexion au serveur : Seul l'host reçoit l'information qu'un joueur a pu se connecter, il reçoit aussi son ID. L'host envoie alors l'information à tous les joueurs connectés dans la partie. Le joueur est envoyé dans le lobby d'attente.
- Erreur de connexion au serveur : Affichage de l'erreur dans la console ainsi qu'un message visuel apparaît sur l'écran du joueur concerné.

4.2.4 Peer-to-peer

Etant donné que nous utilisons un réseau local peer to peer pour jouer à SpaceY, nous avons décidé d'envoyer les informations des joueurs à chaque rafraîchissement d'image sur le réseau. Nous avons fait en sorte de ne transmettre que les informations nécessaires afin de ne pas surcharger le réseau. De ce fait, le jeu est très fluide.

4.3 Synchronisation multijoueur

Le nombre de joueurs est déterminé par le nombre de clients connectés au serveur. Lorsque la partie est lancée, le même nombre d'astronautes apparaît sur la carte, répartis dans des équipes différentes. Pour synchroniser la position des astronautes entre les clients, nous avons utilisé un nœud MultiplayerSyncronizer.

Ce dernier permet de stocker chez tous les clients le vecteur de position d'un joueur, de ce fait, d'actualiser en temps réel la position chez chaque machine. Les pseudos des joueurs apparaissent de la même manière et sont enregistrés dans le même nœud.

Le script permet de vérifier quel joueur est celui contrôlé par le joueur local de la machine. C'est-à-dire que le script vérifie que l'identifiant du joueur est le même que celui de la machine. Si c'est le cas, un indicateur vert apparaît sur le joueur concerné, et la caméra se synchronise sur la vue du joueur.

4.4 Les lasers

Les lasers étaient une partie amusante à aborder. En effet, elle reprenait différentes parties vues au préalable telles que la synchronisation multijoueur des astronautes, ou les collisions entre les entités. C'est pourquoi nous l'avons réalisé rapidement.

4.4.1 Création du laser

Pour commencer, il s'agit d'un nœud CharacterBody2D auquel est rattaché un nœud d'animation 2D qui permet de jouer l'animation en mouvement du laser. Il est instancié lorsqu'un joueur clique sur la touche concernée.



FIGURE 4.7 – Image du laser

4.4.2 Variables de tir

À partir de ce moment, chaque joueur reçoit l'information qu'une personne a tiré. Ils reçoivent notamment les coordonnées vers lesquels doivent se diriger les lasers ainsi que la position de départ. Ainsi, les lasers sont gérés individuellement sur chaque client pour éviter les problèmes de synchronisation. Lorsqu'un laser entre en collision avec un alien, celui-ci disparaît. Les autres interactions ont été développées par un autre membre de notre équipe.

4.5 Lobby d'attente

Le lobby d'attente fut une tâche complexe, en effet, c'est à ce moment que les informations d'initialisation du multijoueur apparaissent.

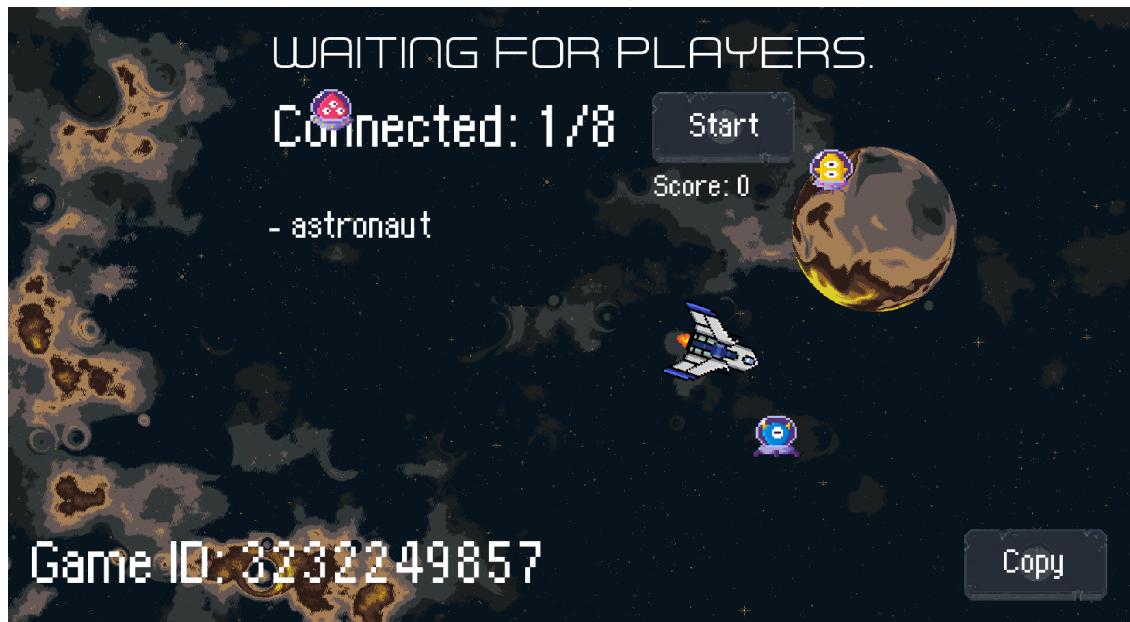


FIGURE 4.8 – Départ au début de la partie

4.5.1 Initialisation

Tout d'abord, l'hôte crée un serveur ou ouvrant son port et en partageant son adresse IP. Il est alors considéré comme "host" et obtient le numéro d'identification 1 (ID). A chaque nouvelle connexion, il reçoit l'ID du nouveau joueur, le stock dans une base de donnée locale et l'envoie à tous les joueurs déjà connectés afin que chaque joueur garde en mémoire les clients connectés au serveur. Une fois ces informations transmises, le pseudo du nouveau joueur apparaît sur les écrans de tous les joueurs.

4.5.2 Mini-jeu shooter

Dans ce lobby, nous avons ajouté un mini-jeu afin de faire patienter les joueurs connectés. Il s'agit d'un shooter dans lequel les joueurs contrôlent une fusée qui tire sur des aliens, le score est inscrit en haut de l'écran. Dans ce mini-jeu, les données sont uniquement locales, et il n'apporte aucune dimension au jeu principal.

4.6 Interface graphique & menus

L'interface graphique est une partie sur laquelle nous ne souhaitions pas passer autant de temps qu'au final. En effet, un membre de l'équipe a tout d'abord réalisé un menu qui n'était pas adéquat, et dans le mauvais langage de programmation. C'est pour cela que la tâche a dû être effectuée par une autre personne.

4.6.1 Fonctionnement des menus

on a tout d'abord créé les différentes scènes au besoin comme le menu principal, les paramètres, le menu de lancement et le lobby d'attente. Dans chacune des scènes, une animation avec le nom de la scène est présente afin de dynamiser le rendu visuel.

Le texte à une petite animation qui le fait flotter sur l'écran. Des noeuds de type TextureButton ont été utilisés afin de créer des boutons personnalisés ce qui a permis de permettre et fluidifier le déplacement entre les différents menus.

L'Ère de l'inaction

5.1 C'est quoi l'ère de l'inaction ? :

Après avoir passé un mois et demi assez chargé en terme de travail sur le projet en début d'année, il s'en est suivi une période s'étalant de Février à Avril environ où nous avons pas été très actifs par manque de temps (Le B3 ayant été un bimestre assez chargé), et par un léger manque de motivation concernant le début du mois d'Avril.

Nous profitons alors de ce petit trou dans le développement du projet pour vous présenter certaines choses qui ne correspondent pas à une période chronologique précise, ou certains moment clefs de la conception de notre jeu.

5.2 Aspect visuel et graphique du jeu

Dans cette partie nous allons nous attarder sur le côté visuel du jeu qui ne peut pas être traité autre part sans divaguer (on ne parlera pas de l'interface utilisateur ici), les différents choix graphiques et artistiques et leurs impacts sur le jeu et le joueur. Cette partie est à part car elle est assez différente des autres car elle n'a pas été faite à un moment spécifique mais a été implémentée au fur et à mesure de l'avancement du jeu.

5.2.1 Les équipes :

Étant que notre jeu est un jeu où deux équipes s'affrontent, il faut bien un moyen pour les joueurs de distinguer de quel équipe ils font parti. Nous avons alors choisi un duo de couleur très populaire pour représenter la dualité dans le monde du jeu vidéo : le rouge et le bleu. Ce sont deux couleurs très présente dans l'imaginaire collectif pour représenter une opposition, comme en politique par exemple, nous les retrouvons aussi régulièrement dans la culture populaire pour exprimer une opposition (par exemple : Star Wars), mais aussi et surtout dans les jeux vidéos comme dit précédemment, ce sont souvent les couleurs du joueur 1 et du joueur 2 (par exemple : les jeux Super Smash bros, la série des "Naruto Ultimate Ninja Storm", Clash Royale...).

Ces deux couleurs ont donc été utilisées à chaque fois qu'il est important de savoir à quel équipe un certain élément appartient, à savoir les joueurs, les fusées et les stations (comme dans la photo ci-dessus), cela permet d'éviter les confusions et de ne pas tirer de laser sur son coéquipier par erreur, ou de ne pas se tromper de fusée lors de la construction de station par exemple, ce qui permet de grandement fluidifier la partie du point de vu des joueurs et de se concentrer uniquement sur la victoire.

En ce qui concerne le choix de la couleur, nous faisons simplement un test sur l'équipe du joueurs et de la fusée et nous chargeons le sprite accordé



FIGURE 5.1 – Aperçu des deux équipes différentes lors d'une partie.

5.2.2 Personnages et déplacements :

Pour un jeu qui tourne autour d'une course dans l'espace, quoi de mieux que des astronautes pour représenter les personnages jouables ? Nous avons opté pour un design assez simpliste et mignon afin de coller à l'esprit du jeu, où il ne faut pas vraiment se prendre la tête (même si jouer de façon compétitive est, à mon avis, plus amusant).

Cependant, nous ne voulions pas que notre personnage soit une simple image collée qui bouge, ce qui donnerait une impression de jeu inachevé. Il était essentiel de le rendre plus vivant et dynamique. Nous évoquerons la majorité des animations en temps voulu (comme le minage, les émotes...), ici nous allons nous attarder sur le déplacement du personnage.

Étant dans l'espace, nous ne pouvions pas utiliser de simples pas. Nous avons donc opté pour un propulseur à l'arrière du personnage, dont l'opacité augmente puis diminue pour rendre l'animation plus dynamique. De plus, nous avons orienté le personnage vers la direction dans laquelle il avance pour rester cohérent.

La gestion de l'opacité a été réalisée grâce à la propriété "Modulate" des objets de Godot. Quant à la direction, nous avons utilisé la rotation des nœuds de Godot pour la contrôler.

5.2.3 Fusée, laser et stations de passages :

En ce qui concerne les fusées, nous avons opté pour un design assez conventionnel (créé par Thunder246 et publié sur le subreddit "Pixel Art"), à l'exception de la couleur. Nous avons changé la couleur verte de base pour l'accorder aux

couleurs des équipes, comme montré dans la section "Les équipes".

Pour le laser que les joueurs peuvent tirer, nous avons choisi une animation plus détaillée que le reste des éléments du jeu, car nous voulions qu'il se démarque. Les lasers ont la même couleur, quelle que soit l'équipe qui les tire, car ils ont le même effet indépendamment de l'équipe du joueur. Nous n'avons donc vu aucun intérêt à indiquer quelle équipe a lancé un laser (même si cela aurait pu créer des querelles entre les joueurs, ce qui aurait pu être drôle, mais pas nécessaire).



FIGURE 5.2 – Une frame du laser tiré.

Enfin, concernant les points d'accès, nous avons opté pour un design rappelant les vieilles antennes réseaux et paraboles. Ce visuel explicite souligne leur utilité : permettre à la fusée d'avancer, rappelant les moyens utilisés pour communiquer avec les avions, par exemple. Comme pour les joueurs et les fusées, nous avons utilisé le bleu et le rouge pour indiquer quelle équipe la station appartient.



FIGURE 5.3 – Les deux modèles de stations différentes posées sur des astéroïdes.

5.2.4 Astéroïdes :

Les graphismes des astéroïdes représentent un aspect primordial du jeu puisque ces derniers doivent pouvoir être différenciés par les joueurs selon leur type tout en correspondant au visuel général de SpaceY. Ainsi les trois différents types d'astéroïdes sont différenciés par trois différentes couleurs :

- Les astéroïdes gris sont des astéroïdes qui ne possèdent pas de ressources. C'est la couleur des astéroïdes dans l'imaginaire collectif.
- Les astéroïdes oranges rougeâtres représentent les astéroïdes possédant du fer. Il s'agit de la couleur du minerai de fer avant d'être traité.
- Les astéroïdes bleus représentent les astéroïdes possédant des cristaux. Il s'agit de la couleur du cristal dans l'imaginaire collectif et avec laquelle il est le plus souvent représenté dans les jeux vidéo.

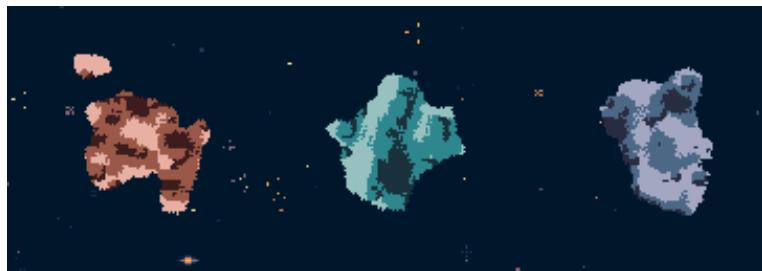


FIGURE 5.4 – Les trois types d'astéroïdes.

Ainsi, les astéroïdes peuvent être facilement distingués grâce à ces trois couleurs très distinctes et significatives pour chaque ressources. De plus, nous souhaitions que la taille et la forme soient différentes pour chaque astéroïde afin de rendre l'environnement du jeu plus vivant et éviter un effet de clonage où chaque élément se ressemble.

5.2.5 Aliens :

Le jeu représente des humains dans l'espace. Nous nous sommes donc dit que les ennemis les plus intuitifs qu'ils pourraient avoir seraient une forme de vie extraterrestre hostile. Alors, nous avons simplement opté des aliens en soucoupe volante, un cliché assez répandu dans la culture populaire, et qui donc parlerait à tout le monde, permettant de facilement se rendre compte de la menace qu'ils représentent pour l'avancée du joueur



FIGURE 5.5 – Deux des différents types d’aliens.

5.2.6 Emoticône :

Cette partie sera plus détaillée que les autres de l’aspect graphique car le côté technique sera aussi abordé, ça sera assez précis chronologiquement car le design des emotes a été fait pendant “l’ère de l’inaction”.

Toujours dans l’esprit multijoueur du jeu, nous voulions mettre un moyen de communication entre les joueurs intégrer au jeu, ce qui leur permettrait de transmettre des émotions dans le jeu (de la joie, de la tristesse, pouvoir provoquer...). C’est un principe qu’on retrouve souvent dans les jeux multijoueurs de ce style (le meilleur exemple est Super Smash Bros où chaque personnage a des “taunt” personnalisées).

Lorsque nous voulions montrer des expressions faciales, on a décidé de faire apparaître des visage en néon pour accentuer le côté futuriste. On a alors décidé d’implémenter 7 emotes qui se lance chacune lorsqu’on appuie sur les touche de 1 à 7 :

- **Coucou** : Appuyer sur 1 lance une animation où l’astronaute fait un coucou, parfait pour saluer en début de partie.
- **Cœur** : Appuyer sur 2 fait apparaître un cœur holographique sur le scaphandre du joueur, qui grandit jusqu’à disparaître (une de mes préférées, signée Madin).
- **Clin d’œil** : Appuyer sur 3 fait apparaître un visage néon sur le scaphandre du joueur, qui fait un clin d’œil.
- **Tourne** : Appuyer sur 4 fait tourner le joueur sur lui-même.
- **Bien fait!** : Afficher un “Bien fait” aurait été trop long et pas assez fluide, nous avons donc opté pour un mot plus court et familier : “Cheh!”. Appuyer sur 5 fait défiler ce mot en néon sur le scaphandre du personnage.

- **Gg <3** : Appuyer sur 6 fait défiler un “Gg” suivi d’un cœur rouge pour l’équipe bleue et un “Gg” suivi d’un cœur bleu pour l’équipe rouge.
- **Larme** : Appuyer sur 7 fait apparaître un visage néon en train de pleurer sur le scaphandre du joueur.

Voici quelques exemples (évidemment ce sont des images et les voir animées est plus intéressant).

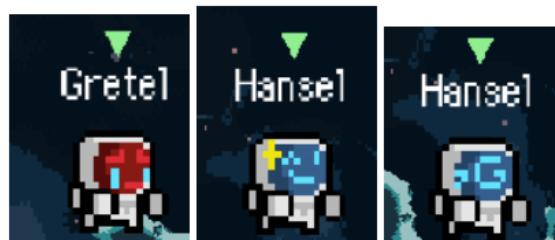


FIGURE 5.6 – Quelques emotes disponibles dans SpaceY.

Concernant la partie technique : Après avoir fini de les dessiner , ce qui était déjà assez long, nous avons dû les implémenter et cela n'a pas été de tout repos. Pour les implémenter nous avons utilisé des AnimatedSprite2D, une structure de Godot, que nous ne rendons visible que lorsqu'une des touches est appuyée, sinon elle est invisible et l'animation ne joue pas. Cependant nous avons fait face à plusieurs problèmes en ce qui concerne l'implémentation multijoueur.

Tout d'abord, elles ne se lançaient que sur un seul PC et s'affichaient sur les 2 joueurs, ce qui était dû à des problèmes de RPC entre les 2 joueurs, la machine ne savait pas lequel des deux joueurs lançait l'animation, nous l'avons donc réglé en lançant la fonction en RPC.

Une fois ce problème réglé, elles se lançaient effectivement sur un seul joueur, mais uniquement sur le pc du joueur qui lance l'animation, cela était dû au fait que l'animation n'était pas synchronisé entre les joueurs, et toutes les informations n'étaient pas partagées (en particulier les propriétés “frames” et “visible”) qui rendaient l'animation invisibles et ne les lançait pas en multijoueur. Ce qui a été assez simple à régler une fois le problème trouvé.

Dans l'ensemble, la création et l'implémentation de ces émotes était très amusante et agréable personnellement (Madin) une de mes parties préférées du développement du jeu.

5.3 Ligne d'arrivée

La ligne d'arrivée est un constitué d'un astéroïde et d'un nouveau noeud signifié par un drapeau.

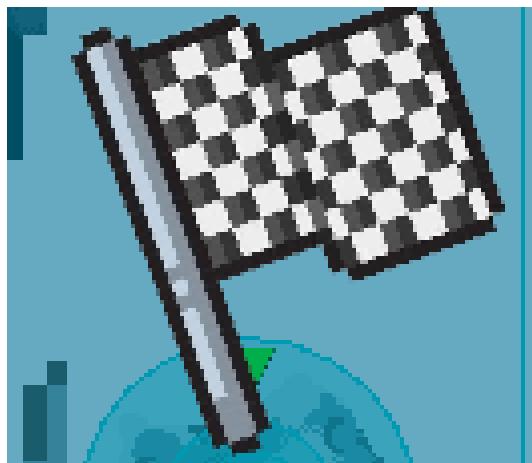


FIGURE 5.7 – Aperçu des deux équipes différentes lors d'une partie.

En effet son taux d'apparition est de 100%.
Le script de gestion détecte simplement la collision avec une fusée. Dès qu'une fusée rentre en collision avec ces astéroïdes le message de fin est personnalisé et envoyé aux différents joueurs.

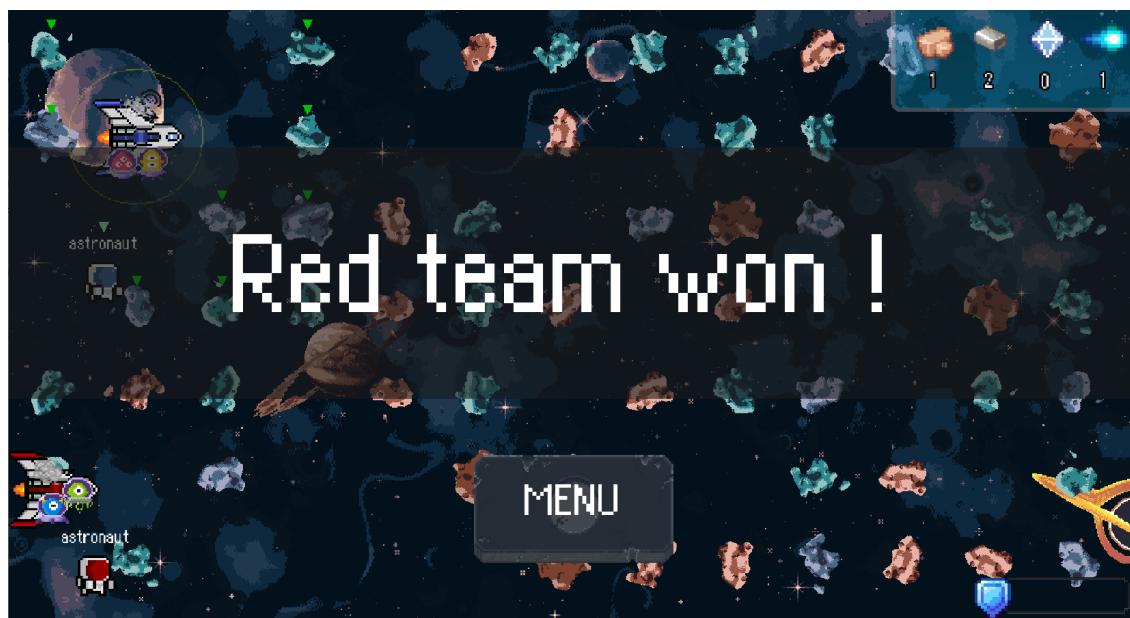


FIGURE 5.8 – Exemple d'écran de fin.

Destination Finale

6.1 Récolte et utilisation des ressources :

Nous l'avons évoqué plus tôt, les astéroïdes possèdent un nombre aléatoire de ressources, cela a été implémenté lors de la création de la map. Cependant, les ressources sont créées pour être récoltées par les joueurs, et c'est là qu'intervient une mécanique avec un nom judicieusement trouvé "la récolte de ressource", qui, elle, a été implémenté plus tard.

Comme vous l'aurez deviné, ça sera le thème de cette partie.

6.1.1 La récolte :

Joueur :

Tout d'abord, du côté du joueur, il doit se mettre prêt d'un astéroïde et maintenir la touche "espace" pendant un certain temps, une animation de minage se lance et une barre de chargement apparaît, à la fin de ce temps (lorsque la barre de chargement se remplit entièrement) l'astéroïde devient gris montrant qu'il ne contient plus de ressource, et les ressources apparaissent dans l'inventaire du joueur

Développement :

Pour ce qui est de l'implémentation de cette mécanique :

- Dans un premier temps, lorsque le joueur appuie sur la touche "espace" on vérifie qu'il soit à proximité d'au moins un astéroïde grâce à la liste "asteroidReachable" qui est la liste des astéroïdes atteignables par le joueur.



FIGURE 6.1 – Le personnage pendant qu'il mine



FIGURE 6.2 – Après avoir fini de miner

- Ensuite on lance la méthode "MiningRessource()" du joueur qui lance le chronomètre, l'animation et la barre de chargement et empêche le joueur

de bouger tant qu'il effectue l'action. Si le joueur lâche la touche "espace", il est remis à ses paramètres de base et ne récupère pas de matériaux

- Enfin une fois ce chargement fini on récupère les ressources de l'astéroïde à l'index 0 de la liste avec les méthodes "GetRessource()" et "GiveRessource()" des astéroïdes qui mettent toute ses ressources à 0, change son visuel et renvoie la quantité de ses ressources, on utilise alors le résultat pour l'additionner aux ressources du joueur dans l'inventaire. Voici le code des 2 méthodes :

```
public void GiveRessource()
{
    quantityOfIron = 0;
    quantityOfCristal = 0;
    ChangeSprite ("normal");
}

public (int, int) GetRessources()
{
    int iron = quantityOfIron;
    int cristal = quantityOfCristal;
    GiveRessource();
    return (iron, cristal);
}
```

Les problèmes rencontrés :

Pour ne pas changer, la majorité des problèmes rencontrés dans cette partie étaient liés avec l'implémentation du multijoueur. Au tout début, le fait d'appuyer sur "espace" lançait l'action chez tous les joueurs. Dès qu'un joueur était à proximité d'un astéroïde, il suffisait que n'importe quel joueur déclenche l'action pour qu'il se mette à miner. De plus, l'action ne se lançait que sur une seule machine.

Ces problèmes sont assez similaires aux problèmes des émoticônes, la solution ne sera donc pas particulièrement développée car nous avons procédé de la même manière.

La fonction qui se joue à la fin du chronomètre "MiningTimerTimeout()" est lancée en RPC, ce qui permet de synchroniser les astéroïdes (ressources et sprite). Nous récupérons également l'identifiant du joueur qui lance l'action et le comparons à celui de la machine pour que la méthode ne se lance pas chez tous les joueurs.

6.1.2 Les utilisations :

Les ressources représentent la clé de la victoire dans Space Y, sans elle il est impossible d'avancer ou d'empêcher l'ennemie d'avancer, nous allons donc passer en revue les différentes utilisation de ces dernières :

Les cristaux : Nous commençons par eux bien que ce soit la ressource la plus rare, car ils ne servent qu'à une seule chose : tirer des lasers (dont nous avons développé les nombreuses utilités précédemment). Lorsqu'un joueur rentre dans la zone autour de sa fusée, il dépose automatiquement ses cristaux, qui peuvent être utilisés par toute l'équipe.

Le fer : Une ressource brute, très commune dans les astéroïdes, qu'on retrouve dans les astéroïdes oranges et bleus. Mais inutile sous cette forme, son but est d'être ramené à la fusée pour être raffiné puis transformé en lingot de fer (3 fer = 1 lingot).

Les lingots : Ils représentent la ressource avec le plus d'utilité du jeu :

- Créer une station de passage coûte 2 lingots.
- Détruire une station, alliée ou ennemie, coûte également 2 lingots.
- Réparer sa fusée lorsque ses points de vie sont inférieurs à 5 coûte 1 lingot.

L'implantation a été plutôt simple car les ressources étaient gérées dans un dictionnaire qui représentait l'inventaire des joueurs, alors il suffisait de faire des tests sur les valeurs nécessaires et de soustraire le coup à chaque action.

6.2 Le retour de la fusée

Maintenant que les Ressources sont fin prêtes, l'une de leurs utilisations est la réparation de la fusée ! En effet la fusée n'est pas invincible car elle possède des points de vie limités. Ainsi si la fusée n'a plus de points de vie alors elle est détruite et l'équipe adverse gagne.

La fusée peut perdre des points de vie dans deux cas de figure :

- Si un aliens tape la fusée (ce point sera plus détaillé dans la prochaine section concernant les aliens).
- Si l'équipe adverse tire un laser sur la fusée (une fusée ne peut pas se tirer dessus).

Dans ces deux cas de figure la fusée perd un point de vie par aliens/laser qui vient à l'atteindre.

La fusée possède au maximum dix points de vie et selon le montant de points de vie restant de la fusée, celle-ci a deux états :

- **L'état "normal".** Cet état se manifeste quand la fusée possède 5 points de vie ou plus. Dans cet état la fusée fonctionne normalement, elle peut avancer et n'a pas d'apparence particulière.
- **L'état "endommagée".** Cet état se manifeste quant à lui quand la fusée possède moins de 5 points de vie . dans cet état la fusée fume pour indiquer visuellement son état au joueurs, et elle se fige sur place jusqu'à ce qu'elle soit réparée.



FIGURE 6.3 – fusée dans l'état "normal"



FIGURE 6.4 – Fusée dans l'état "endommagée" (la fumé est animée)

Pour réparer la fusée la procédure est très simple : tout d'abord il faut récupérer 1 lingot de fer, puis s'approcher de la fusée et appuyer sur la touche [R] le temps qu'une jauge se remplisse. Une fois fait, la fusée remonte à 10 points de vie sur 10 et elle se remet à fonctionner.

6.3 Les aliens & leur intelligence artificielle

Les aliens consistent en une entité entièrement gérée par l'ordinateur qui va apporter une difficulté dans la progression des joueurs. Pour leur réalisation, il s'agit d'un nœud de type CharacterBody2D sur lequel sont accrochées plusieurs dépendances. L'alien est doté d'un nœud de collision (Area2D et CollisionShape2D) afin de ne pas pouvoir passer au travers des murs.

6.3.1 Association d'une équipe

Premièrement, l'alien est associé à une équipe. Il doit traquer la fusée de l'équipe concernée. Pour ce faire, un tag y est associé, il s'agit du chiffre 1 ou 2. En fonction de ce chiffre, son lieu d'apparition est défini dans le script de gestion de la map.

6.3.2 Identité visuelle

Lors de l'apparition de l'alien, une identité visuelle est définie de manière aléatoire. Pour ce faire, son nœud Sprite2D cherchera aléatoirement une image parmi 4 disponibles.

6.3.3 Pathfinding A*

La partie la plus importante et la plus intéressante est encore à venir. Afin de traquer la fusée d'une équipe, les aliens utilisent un algorithme de pathfinding nommé A* (soit "A Star").

Pour ce faire, un nœud `NavigationAgent2D` est utilisé afin de calculer le meilleur itinéraire à l'aide de cet algorithme. Du côté du script, un timer est déclenché toutes les 0.5 secondes afin de rafraîchir en mémoire la position de la fusée. Plusieurs autres fonctions constituent l'algorithme de sélection des positions vers lesquelles se déplacer.

6.3.4 Interactions

L'interaction du laser qui entre en collision avec l'alien est codée dans ce même script. Dès qu'un laser touche un alien, cet alien se détruit instantanément en produisant un effet sonore.

Par contre l'interaction avec la fusée, le fait de la faire perdre des points de vie à intervalle régulier, est lui complètement fait dans la fusée,

6.3.5 Complications

Ce fut compliqué de développer cette partie de pathfinding, en effet, nous sommes restés plusieurs semaines avec un algorithme défaillant qui ne faisait se déplacer l'alien qu'en direction de la fusée, sans esquiver les astéroïdes, de ce fait, il restait bloqué dans le décor.

Nous avons aussi dû modifier la génération des astéroïdes afin que les aliens puisse naviguer correctement. En effet nous avons modifié la `Tilemap` qui contient tout les astéroïdes afin d'avoir une case entre tous les astéroïdes pour signifier que l'alien peut passer entre.

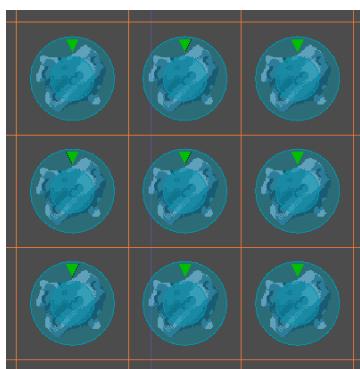


FIGURE 6.5 – TileMap avant la modification

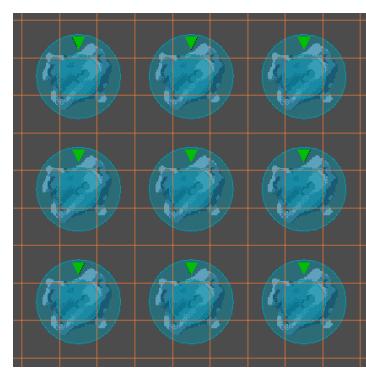


FIGURE 6.6 – TileMap après la modification

6.4 Tutoriel

Il serait inutile d'expliquer pourquoi un tutoriel est important, voire indispensable, dans un jeu vidéo. Il faut une référence pour guider les nouveaux joueurs avant qu'ils prennent leurs marques. La majorité des jeux vidéo l'intègre de façon diégétique, comme dans les jeux Pokemon où c'est un personnage du jeu plus expérimenté qui apprend les base à notre personnage au début du jeu.

6.4.1 Conception du tutoriel :

En ce qui concerne Space Y, qui est un jeu uniquement multijoueur, il est difficile de créer une phase de tutoriel au début du jeu car il n'y pas de début de jeu à proprement parler. Nous l'avons donc mis dans les menus du jeu dans la section "help" pour apprendre les différentes touches et mécaniques aux joueurs.

Au départ, nous voulions créer un tutoriel interactif et scénarisé où des personnages donnent des indications au joueur pour lui apprendre les différentes touches et actions. Cependant, par manque de temps, cela n'a pas été possible.

Nous avons alors opté pour un autre type de tutoriel. L'essence de la première idée a été conservée, car il s'agit toujours de différents personnages non jouables qui expliquent au joueur l'objectif de sa mission (comment gagner la partie). La différence réside dans l'exécution. Au lieu de scénariser cela comme un terrain d'entraînement, nous avons mis en scène un cours se déroulant sur une télévision, où les PNJs dictent au personnage quoi faire.



FIGURE 6.7 – Capture d'écran de la première image du tutoriel

6.4.2 Les PNJs :

- **Le colonel** : Il s'agit du chef, celui qui donne les ordres et confie les missions. Il est sévère et impartial mais reste bienveillant envers le personnage. C'est lui qui dicte les différentes étapes de la mission et explique au joueur son objectif.



FIGURE 6.8 – Icône de "LE COLONEL".

- **Le mineur** : Son nom est assez explicite, c'est lui qui apprend au personnage comment miner et raffiner les différentes ressources. Il a été pensé comme un personnage calme qui ne divague pas trop.



FIGURE 6.9 – Icône du mineur.

- **L'ingénieur** : Il se présente comme "The waypoint guy" en anglais et insiste sur le fait qu'il n'est pas un robot (est-ce vrai pourtant ? Qui sait ?). Il parle de façon très informelle et a l'air pressé de partir. C'est lui qui apprend au joueur les différentes utilités des lingots (construction et destruction de points d'accès et réparation de la fusée).

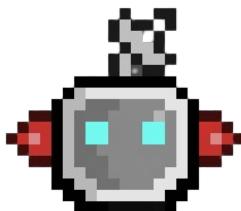


FIGURE 6.10 – Icône de l'ingénieur.

- **La petite fille** : Il s'agit de la fille du colonel. On ne sait pas vraiment pourquoi elle est ici, mais elle déteste les aliens et l'équipe adverse et adore les lasers. C'est d'ailleurs elle qui apprend au joueur comment tirer un laser et ses différentes utilités. Elle est très impulsive et dynamique.



FIGURE 6.11 – Icône de la petite fille



FIGURE 6.12 – Icône de la petit fille parlant de laser.

6.4.3 Script

Le script de gestion du tutoriel sert à gérer les cas de changement d'étape du tuto.

Soit, à chaque fois que l'utilisateur va presser le bouton avant ou arrière. C'est aussi à ce moment que j'ai commencé à prendre des captures d'écran pour illustrer le tutoriel.

6.5 Interface utilisateur :

Dans la majorité des jeux vidéo, si ce n'est tous, l'interface utilisateur (abrégée UI) est cruciale pour rendre l'expérience utilisateur plus agréable. Dans notre cas, ce qui doit être affiché en jeu inclut le nombre de ressources du joueur et le nombre de lasers qui peuvent encore être tirés, en haut à droite, ainsi que la barre de vie de la fusée, en bas à droite de l'écran.

Pour implémenter cette UI, nous avons créé la classe `levelui` qui affiche les informations d'un joueur en particulier. Il s'agit d'un nœud `CanvasLayer`, ce qui signifie qu'elle s'affiche de manière constante sur l'écran, et d'une `TextureProgressBar` pour la barre de PV de la fusée.

Dans un premier temps, nous nous attarderons sur l'affichage des différentes ressources : Les icônes représentent, dans l'ordre : le fer brut, les lingots de fer, les cristaux du joueur, et le nombre de lasers que la fusée peut tirer. Nous avons utilisé des `TextureRectangle` pour afficher les différentes icônes (fer, lingot, cristal et laser) et des `Label` pour afficher la valeur de chaque ressource. Le texte du `Label` est relié aux valeurs de chaque ressource dans l'inventaire du joueur, à l'exception des lasers qui ne sont pas dans l'inventaire du joueur mais une propriété de la fusée. On accède au joueur grâce à la propriété `player` du `levelui` et, en accédant à la fusée de son équipe, on affiche la quantité de lasers

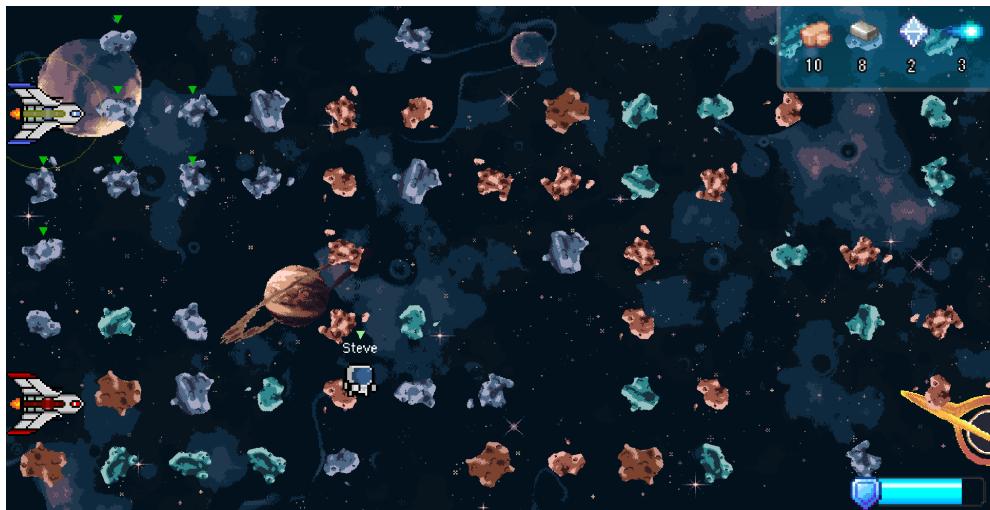


FIGURE 6.13 – Une capture d'écran qui montre une partie de SpaceY.



FIGURE 6.14 – L'affichage de l'inventaire du joueur.

qui peuvent être tirés.

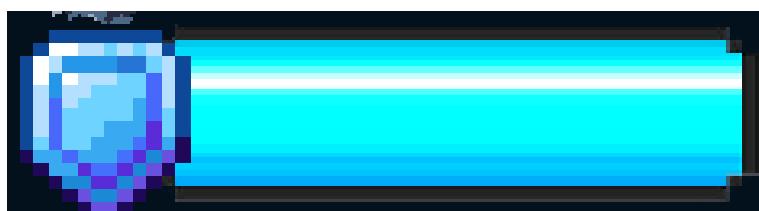


FIGURE 6.15 – Barre de vie complète.

Dans un second temps, parlons de la barre de vie :
 L'icône des points de vie est représentée par un bouclier pour symboliser la défense, tandis que la barre de vie utilise une texture au style futuriste pour correspondre au reste du jeu. Il s'agit d'une TextureProgressBar dont le pourcentage est relié aux PV de la fusée (nous y accédons de la même manière que pour le nombre de lasers). Cette barre de vie change de couleur lorsque la fusée passe en dessous de 5 PV pour indiquer qu'elle ne peut plus avancer, afin d'éviter toute

confusion pour les joueurs sur l'état de la fusée (bien qu'il y ait également de la fumée qui s'échappe de la fusée).



FIGURE 6.16 – Barre de vie inférieur à 5PV.

En ce qui concerne les difficultés rencontrées, elles provenaient de la synchronisation entre la bonne UI et le joueur, afin de ne pas afficher les valeurs de chaque joueur, ce qui n'aurait été ni lisible, ni pratique.

Une première idée a été de redéfinir l'UI de chaque joueur lorsqu'il rejoint la partie. Le problème était que seul l'UI du dernier joueur qui rejoignait la partie s'affichait, car elle était redéfinie à chaque nouveau joueur.

Pour régler ce problème, nous avons associé une UI à chaque joueur, et avons rendu les UI des autres joueurs invisibles. Ainsi, le seul UI visible est celui de l'astronaute que le joueur contrôle. Le seul inconvénient de cette solution est qu'elle n'est pas très optimisée, car elle revient à générer les UI de tous les joueurs présents sur chaque machine, pour qu'un seul d'entre eux soit visible pour chaque joueur au final.

6.6 Système Sonore

L'aspect sonore est souvent un aspect oublié des jeux, on ne le voit pas, souvent c'est assez discret et on ne le remarque presque plus. Cependant cette ambiance sonore est très importante pour tout les jeux, elle permet de mettre le joueurs dans une certaine émotion.

Par exemple lors d'une scène triste il est préférable de mettre une musique lente et triste, une musique trop énergétique pourrait sortir le joueurs du moment ce qu'on ne veux surtout pas.

C'est pour cela que SpaceY devait avoir des musiques et des bruitages. Pour ce faire nous avons fait un noeud, "SoundSource", qui est mis dans les noeuds "autoload". Les noeuds autoload sont des noeuds qui sont appelés quelque soit la scène cela est très pratique lorsqu'on veut accéder à un noeud à n'importe quel moment.

Dans SoundSource s'y trouvait deux fonctions principales : PlayMusic et PlaySfx. La première permet de lancer de la musique en continue, pour ce faire elle dis-

pose d'une librairie de quelque musiques et en choisie une aléatoirement. La seconde permet de lancer des bruitages. Cette fonction à été utiliser un peu partout dans le code, car beaucoup d'interactions possèdent leur propre son qui les permet de les rendre unique et reconnaissable n'importe où sur le terrain de jeu.

Par exemple : lorsqu'une vague d'aliens apparaît, un son particulier est émis et ce dernier est audible sur tout le terrain de jeu, donc quelque soit la position du joueurs, il est au courant qu'une nouvelle vague arrive.

Pour qu'il n'y est qu'une fonction pour tout les bruitages, il a fallu ruser :

```
public static void PlaySfx(string name)
{
    foreach (var s in sfx)
        //sfx est la librairie de bruitage
    {
        if (s.Name == name)
        {
            ((AudioStreamPlayer)s).Play();
            return;
        }
    }
    throw new ArgumentException("Sound not found");
}
```

Ce code est assez simple : il prend en entré une chaîne de caractères et le compare avec tout les sons disponibles, or tout les sons on un nom différents, donc si le son correspond alors il est joué, s'il ne correspond à aucun son dans la librairie alors le code renvoie une erreur ArgumentException.

Bilan du projet :

- **Personnage** : Toutes les fonctionnalités, actions et interactions prévues sont présentes, et certaines ont même été rajoutées comme les emotes.
- **GUI** : Nous avons réussi à implémenter une interface utilisateur complète, que ce soit dans les menus ou dans la partie, et qui correspond à exactement ce que nous avions prévu.
- **Alien** : Les aliens sont complètement finis, il y a plusieurs visuels d'aliens pour la diversité et le pathfinding est opérationnel.
- **Map** : Les entités ne peuvent pas s'échapper, les astéroïdes apparaissent aléatoirement, les promesses faites ont été tenues mais nous sommes déçus de n'avoir pas pu rajouter d'autres types de générations.
- **Conditions de Victoire** : Nous avons fini 100% des conditions de victoire prévues à la base, à savoir atteindre la ligne d'arrivée. Nous avons même pu rajouter une autre condition avec les points de vie de la fusée qui n'était pas prévu initialement
- **Site Internet** :
 - **Fusée** : Les fusées sont complètes, toutes les interactions sont présentes et fonctionnelles. La navigation marche du feu de dieu.
 - **Construction des points d'accès** : La construction des points d'accès est complètement finie et permet d'aller jusqu'au bout du jeu.
 - **Astéroïdes** : Les astéroïdes ont également été complètement finis, leur génération fonctionne sans aucun bug.
 - **Laser** : Le laser a été complété à 100%, il est aussi puissant que nous l'avions imaginé et il ne présente aucun bug.
 - **Ressources et Récolte** : La fonctionnalité marche correctement et ne cause aucun problème, elle a été finie à 100%.
 - **Assets graphiques** : Il ne manque aucun asset graphique, ils ont tous été implémentés et nous conviennent parfaitement.

- **Assets sonores** : Tous les assets ont été trouvés et sont tous raccord, leur utilisation est omniprésente dans le code.
- **Réseau multijoueur** : Le réseau multijoueurs est globalement au point, la synchronisation des actions lors de la partie marche très bien. Cependant la déconnexion n'est pas fini car nous n'arrivions pas à trouver le bon moyen.

Conclusion

Au final tout à été fait, Space Y est vraiment devenu ce qu'on imaginait en début d'année, voire même plus que ce que nous avions prévu ! Il est néanmoins loin d'être parfait, notamment sur la déconnexion des joueurs qui n'est pas au point.

Mais nous sommes fières de ce que nous avons pu accomplir lors de cette année, nous avons appris beaucoup de chose, que cela soit dans l'aspect humain comme technique.

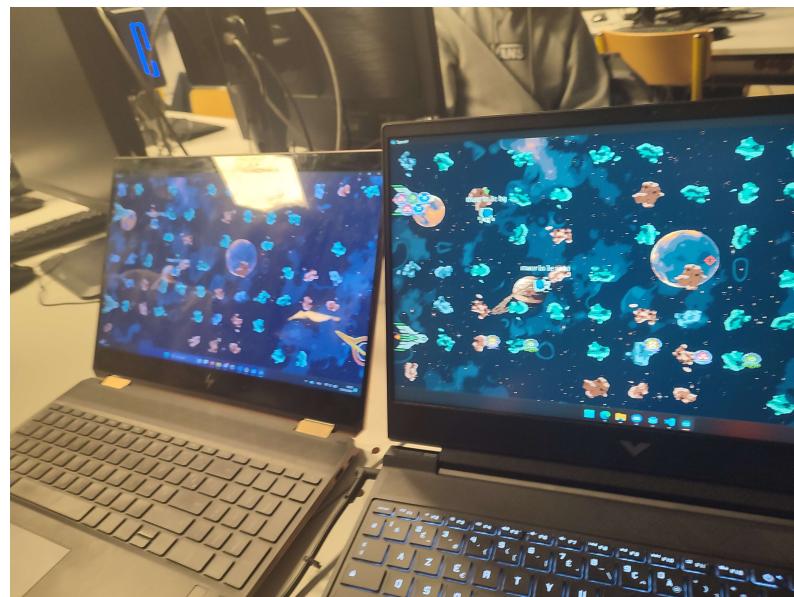


FIGURE 8.1 – La première partie de SpaceY en multijoueurs !

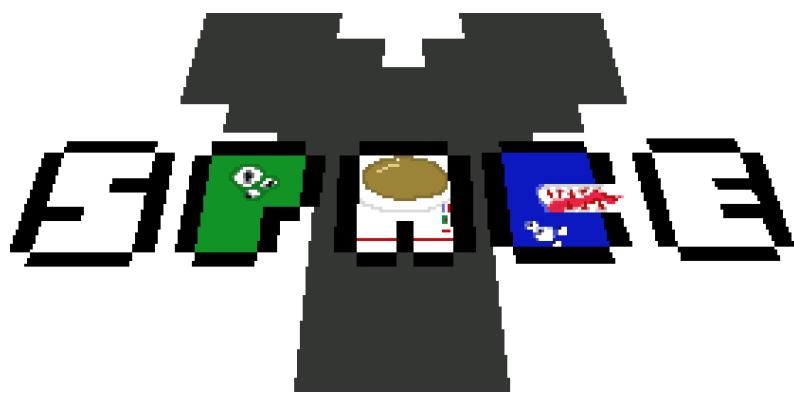


FIGURE 8.2 – Le tout premier logo