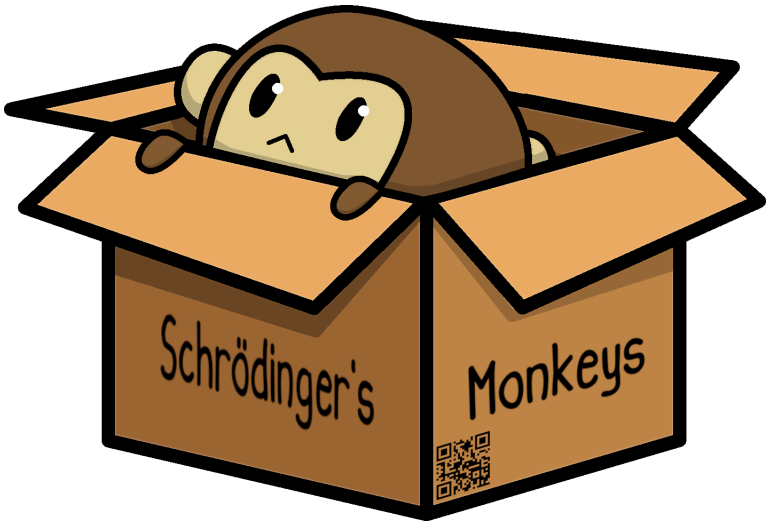
Rapport de soutenance 3

Projet Fracture



Groupe Schrödinger's Monkeys

- Toan Gaucher-Kriete
- Rapahël Leroy
- Adam Nessaibia
- Maëlys Rimbert

Juin 2021

Table des matières

1 Introduction

Fracture est un jeu vidéo de plateformes en deux dimensions, axé sur la coopération de deux joueurs évoluant dans des univers parallèles, en écran scindé. Le jeu prend place dans la Grèce antique et permet uniquement à 2 joueurs de jouer en multijoueur.

Notre équipe est soudée et arbore un nom singulier : les Singes de Schrödinger. Le logo de notre équipe se trouve sur la page de couverture de ce rapport. Vous pourriez demander ce qu'est un singe de Schrödinger ; eh bien en réalité, c'est ce que nous sommes tous. Des singes qui, dès lors qu'ils réalisent leur condition simiesque, changent d'état pour devenir humains. Nous remercions M. Ternier, grâce à qui nous avons enfin pu devenir non-singes.

Nous nous sommes tous mis d'accord sur le principe du jeu, dès le début : la mécanique principale du jeu consiste, pour les joueurs, à échanger de monde afin de surmonter les épreuves mises en place dans les niveaux. Le but est que les joueurs aient besoin l'un de l'autre pour avancer dans deux mondes distincts, d'où le nom *Fracture*.

Pour cette dernière soutenance, nous sommes fiers de vous présenter la dernière version de *Fracture*. Dans le rapport qui suit, vous en apprendrez plus sur les différentes tâches que nous avons pu réaliser au cours de ce projet, nos méthodes de travail, notre organisation, nos difficultés et nos réussites ainsi que nos progrès par rapport à ce que nous avions avancé au début du projet.

2 Avances et retards

2.1 Avancement par rapport aux objectifs du cahier des charges

En janvier 2021, notre groupe se réunissait pour décider de ce que serait *Fracture*. Il en a découlé notre cahier des charges qui présentaient les objectifs que nous nous étions fixés. Nous souhaitions réaliser un jeu de plates-formes en 2D jouable à 2 joueurs en écran scindé. Les deux joueurs devaient être amenés à coopérer d'une manière ou d'une autre pour venir à bout des différents niveaux. Ce jeu devait donc comporter :

- Plusieurs niveaux
- 2 personnages
- Plusieurs ennemis
- Plusieurs musiques
- 1 ou plusieurs boss
- Des mécaniques pour les personnages
- Un menu

Après 6 mois de développement, notre jeu a atteint tous les objectifs fixés par le cahier des charges.

Interface graphique L'interface graphique de *Fracture* comporte toutes les fonctionnalités que nécessite le jeu. Elle permet au joueur de modifier la taille de la fenêtre de jeu et de modifier les touches liées aux différentes actions réalisables par le joueur. L'interface graphique est également présente en cours de partie et permet aux joueurs de quitter la partie en cours ou, si le joueur en question est le maître de partie, de changer de niveaux. Enfin, elle rend la création d'une partie publique ou privée simple et intuitive.

Graphismes 2D Ce projet a nécessité la création de nombreux graphismes 2D. Il y a tout d'abord les graphismes des deux personnages mais aussi les graphismes du blob, des harpies, des canons et des boules de feu. Enfin, il y a également les tuiles qui ont permis la réalisation des niveaux.

Animations Les deux joueurs, leurs armes, les ennemis ainsi que certaines caméras sont animés. Afin de pouvoir animer notre jeu, il a parfois été nécessaire de créer des systèmes d'os.

Audio Pour ce qui est de l'audio de notre jeu, il comporte 3 musiques de niveaux, 1 musique de menu ainsi que plusieurs effets sonores pour accompagner certaines actions du joueur. Cette partie a elle aussi répondu aux attentes du cahier des charges.

Conception de niveau Notre jeu possède 6 niveaux dont un niveau de boss. Ils peuvent avoir des orientations différentes.

Construction des mécaniques Les joueurs ont à leur disposition une multitude de mécaniques pour venir à bout des différents niveaux. Ils sont évidemment capables de marcher, sauter et attaquer mais également de double sauter ou de se propulser en avant et d'escalader certaines parois. Il y a également la mécanique centrale de notre jeu, l'échange de position qui permet aux joueurs de passer d'un monde à l'autre afin de poursuivre l'exploration du niveau.

Multijoueur *Fracture* étant un jeu uniquement jouable en multijoueur, il était essentiel qui celui-ci soit fonctionnel. Ainsi, le jeu offre deux systèmes de matchmaking différents : les parties publiques et privées. L'utilisation de Photon nous permet également de synchroniser les animations ainsi que les déplacements des joueurs et des ennemis. Enfin, l'usage des appels de procédure à distance a permis la synchronisation des tâches plus complexes tels que l'instanciation de particules ou la destruction d'ennemi.

Site internet Notre jeu est représenté sur le web sur notre site disponible ici : https://fracturegame.github.io/.

Intelligence artificielle Finalement, les intelligences artificielles de ce projet, à savoir, celle du blob, des différents types de harpies, du canon et du boss de fin, sont toutes implémentées.

2.2 Tableau comparatif avec le produit d'amusement minimal

Au cours du projet, nous avons jugé préférable de nous baser sur un produit d'amusement minimal. Voici le tableau comparant l'avancement du projet à la fin de la période de développement et les objectifs que nous avons jugé suffisant pour avoir un jeu amusant.

Tâches	Soutenance finale	Produit d'amusement minimal
Interface graphique	Menu basique, menu des contrôles, changement de niveaux en cours de partie	Menu basique
Graphismes 2D	ensemble de tuiles, 2 personnages et leur arme 3 ennemis (blob, harpie, canon et boule de feu)	ensembe de tuiles, 2 personnages, 1 ennemi
Animations	Animation de tous les ennemis, des deux personnages, tremblement de caméra	Animation d'un ennemi et des deux personnages
Audio	1 musique de menu et 3 musiques de jeu, effets sonores	1 musique de jeu
Construction de mécaniques	7 mécaniques (marche, saut, double saut, dash, escalade, changement de monde, attaque)	4 mécaniques (marche, saut, attaque, changement de monde)
Conception de niveau	6 niveaux	1 niveau
Multijoueur	Oui	Multijoueur fonctionnel
Site Web	Oui	Pas nécessaire
Intelligence artificielle	Intelligence artificielle du blob, des différents types d'harpies, du canon et du boss	Intelligence artificielle d'un ennemi

2.3 Changements de plans et abandons d'objectifs

2.3.1 Objectifs modifiés

Le nombre de niveaux Nous avons décidé de réduire le nombre de niveaux de 8 à 6. Cela nous a permis de gagner en temps.

La messagerie instantanée La messagerie instantanée a été ajoutée au jeu un peu avant la première soutenance. Bien qu'étant parfaitement fonctionnelle, elle ne semblait pas avoir un réel intérêt. Nous avons donc pris la décision de la retirer du projet.

2.3.2 Objectifs abandonnés

Le didacticiel Nous avons finalement décidé de ne pas créer de didacticiel pour notre jeu. A la place nous avons augmenté la difficulté des niveaux graduellement et ajouté un menu des contrôles afin que le joueur connaisse les touches à actionner.

Le classement des meilleurs joueurs Le classement des meilleurs joueurs n'a finalement pas été ajouté à *Fracture*. En effet, il semblait difficile de pouvoir prouver la légitimité du classement si ce dernier était stocké chez le joueur et donc modifiable. De plus, il n'était pas facile de savoir quels critères privilégier pour assigner un score à un duo. Faut-il favoriser le chronomètre ou le nombre de points de vie ? Cela reste-t-il vrai pour tous les niveaux du jeu ?

L'éditeur de niveaux Arrivé au terme de son développement, notre jeu ne possède pas d'éditeur de niveaux. Il s'agissait d'un objectif ambitieux et nous avons préféré ne pas nous attarder dessus.

3 Répartition des tâches

3.1 Toan

3.1.1 Multijoueur

Photon Unity Networking 2

C'est Toan qui s'est chargé de l'implémentation du multijoueur du jeu. Afin d'y parvenir, nous avons utilisé la version gratuite de l'asset Photon Unity Networking 2 (PUN2) de Exit Games, présent sur l'Asset Store de Unity. Ce dernier permet de connecter jusqu'à 20 joueurs dans une même partie.

Après avoir installé PUN2, il a fallu lier notre projet à Photon à l'aide d'un identifiant (AppId) pour pouvoir utiliser le Photon Cloud. Il s'agit des serveurs de Photon répartis aux quatre coins du globe. Son fonctionnement est assez simple, les clients se connectent à un premier serveur qui prend connaissance de l'AppId du client, de sa version et de la région à laquelle il souhaite se connecter. Le joueur est alors redirigé à un deuxième serveur qui se charge de lister toutes les parties en cours du jeu pour une région. Ainsi, à chaque fois qu'une partie est créée ou rejointe, le client peut être redirigé vers un serveur de jeu dans lequel il pourra finalement accéder au jeu.

Enfin, Photon est particulièrement intéressant quand il s'agit de créer un jeu avec des parties comprenant un nombre limité de joueurs, ce qui est notre cas puisque nous souhaitons réaliser un jeu jouable à 2 joueurs. Cet asset nous laisse la liberté de choisir un système de partie publique ou de parties privées joignables avec un code. Nous avons décidé d'implémenter les deux !

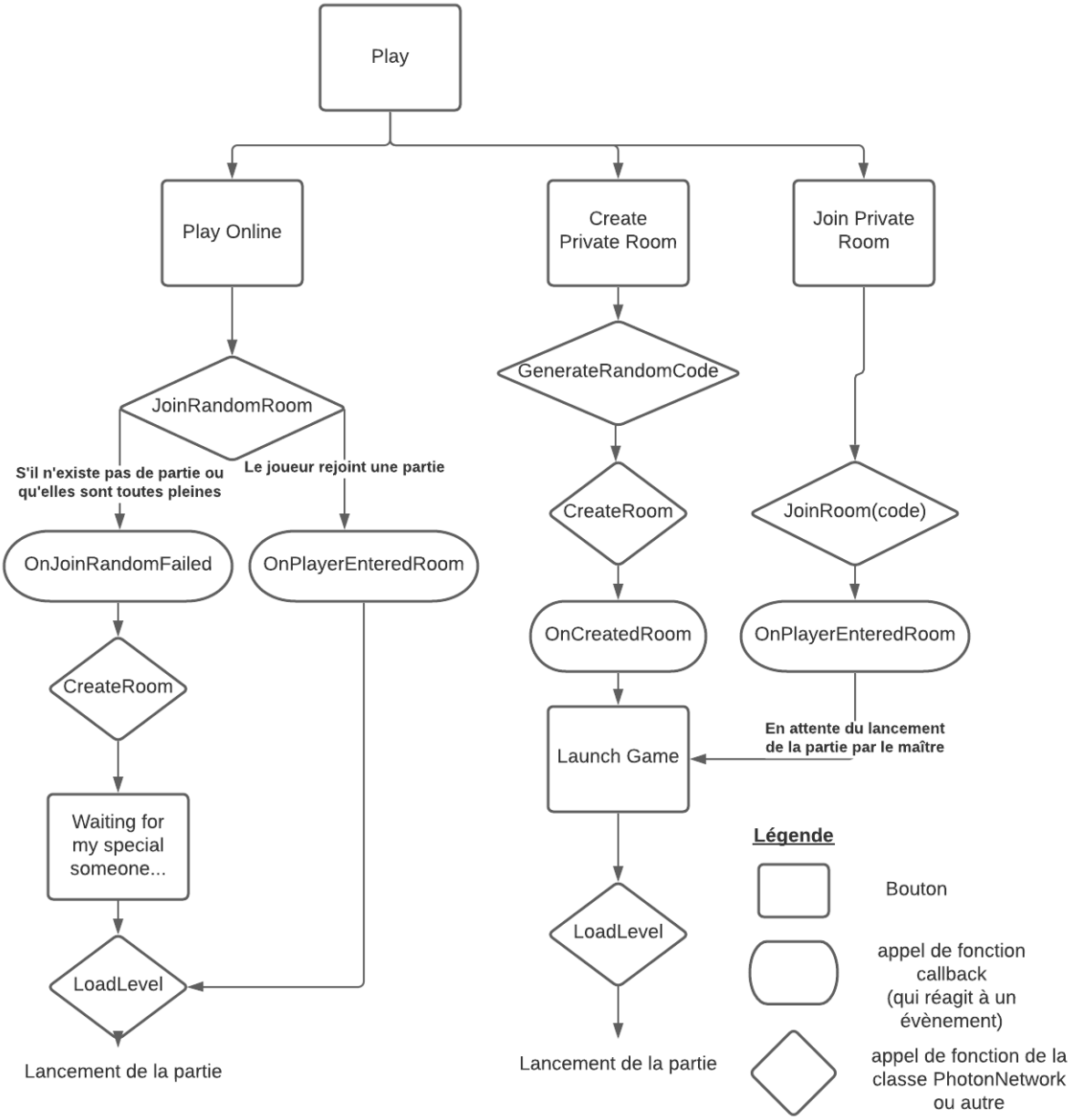
Systèmes de matchmaking

Il existe deux systèmes de matchmaking :

- les parties publiques
- les parties privées

Partie publique Dans une partie publique, s'il existe déjà une partie avec une place de libre, le joueur la rejoint et la partie se lance immédiatement. Au contraire, s'il n'existe aucun partie ou si toutes les parties sont déjà pleines, le joueur crée sa partie et attend son coéquipier.

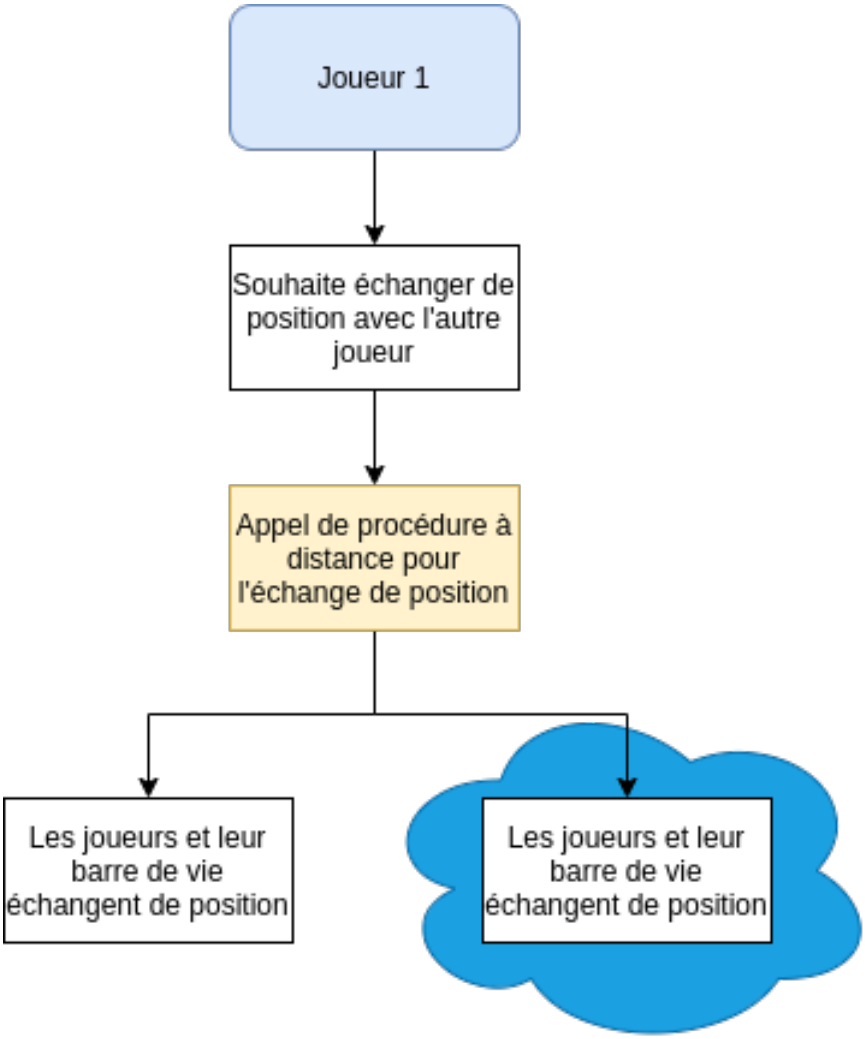
Partie privée Afin de rejoindre une partie privée, il faut connaître le code généré par le créateur de la partie.



Flow chart des systèmes de matchmaking

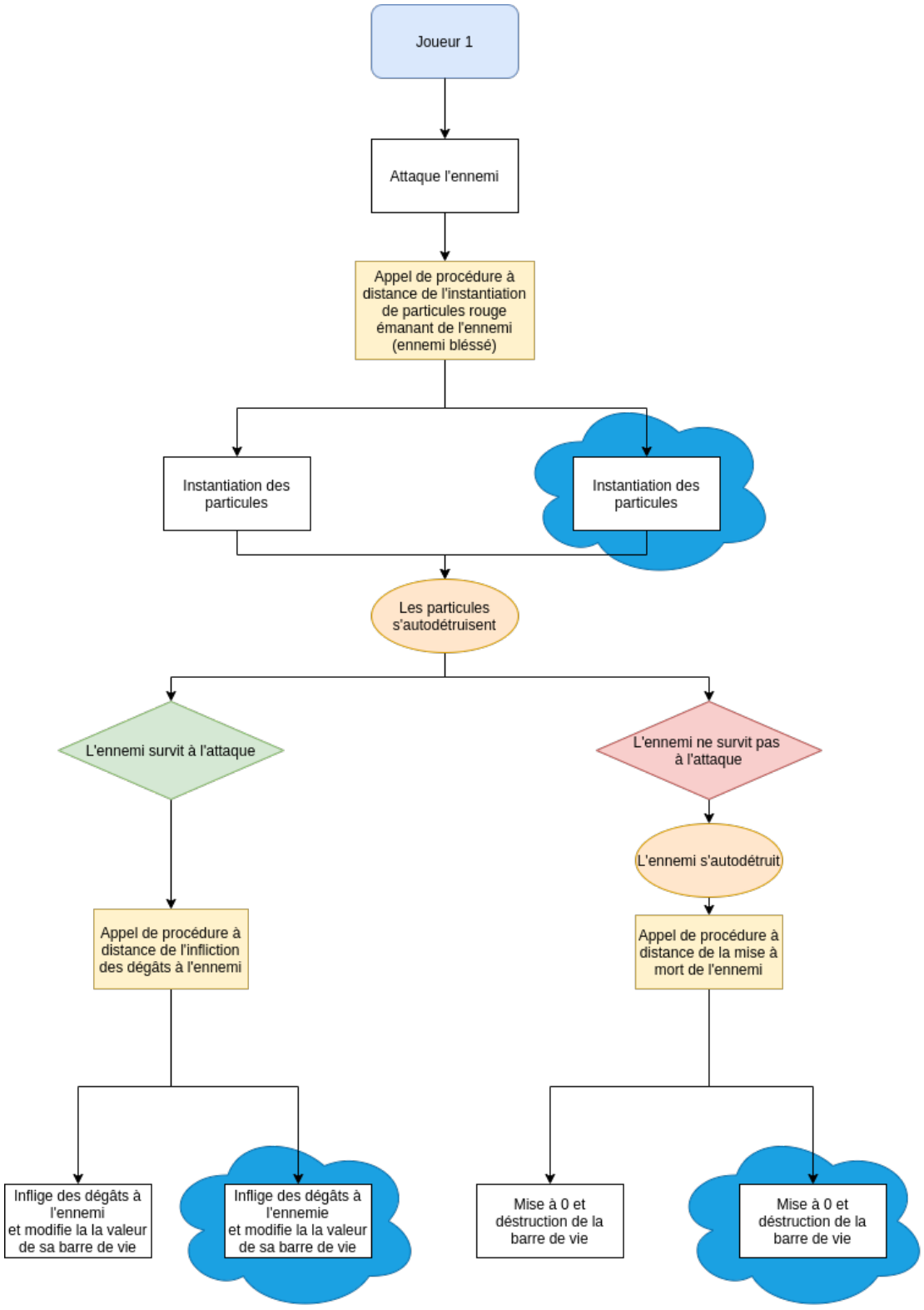
Appels de procédure à distance (RPC)

Ajout de la mécanique d'échange de position à l'aide de RPC



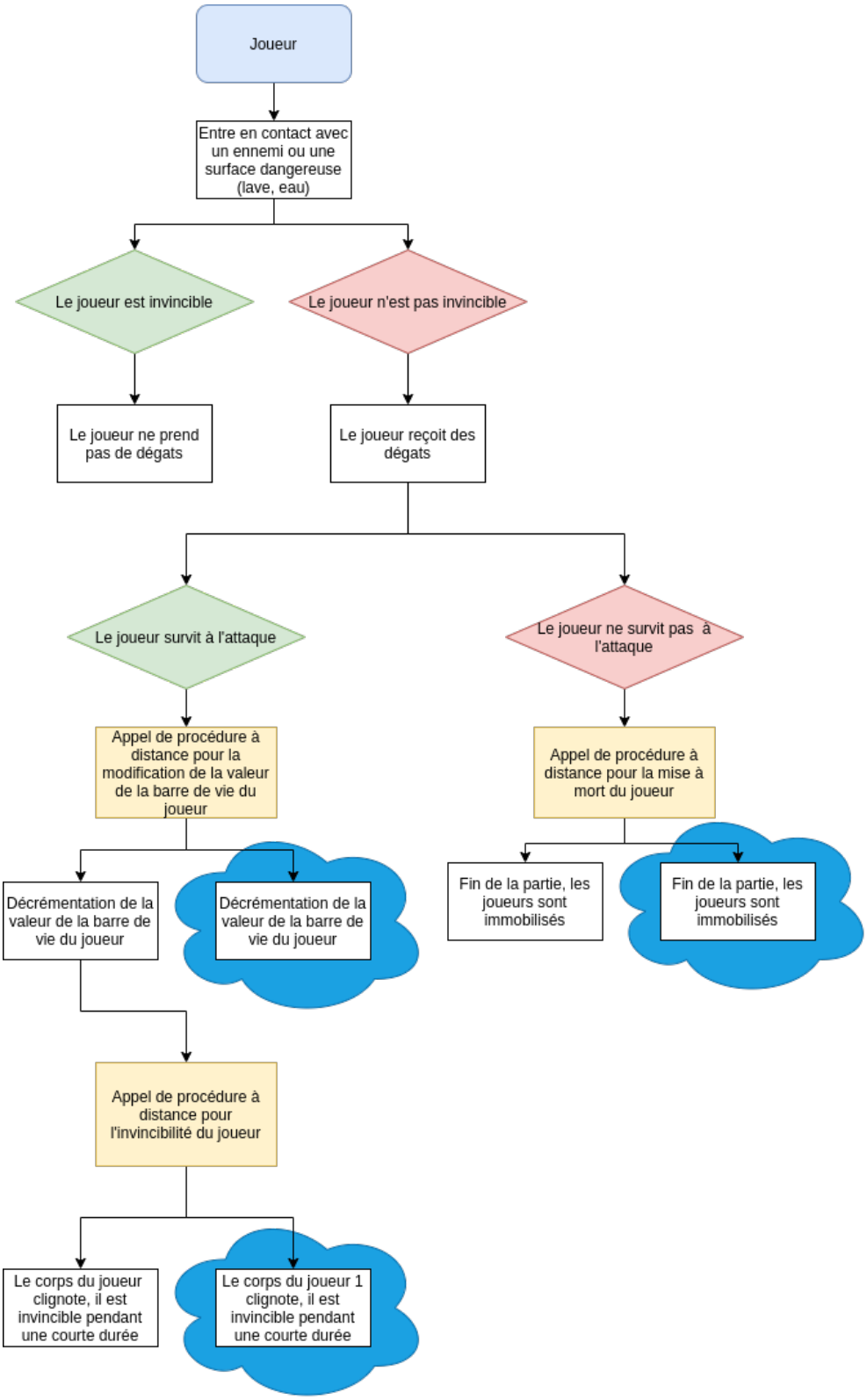
Flow chart du fonctionnement de la mécanique d'échange de position

Synchronisation des dégâts infligés aux ennemis via des appels de procédure à distance.



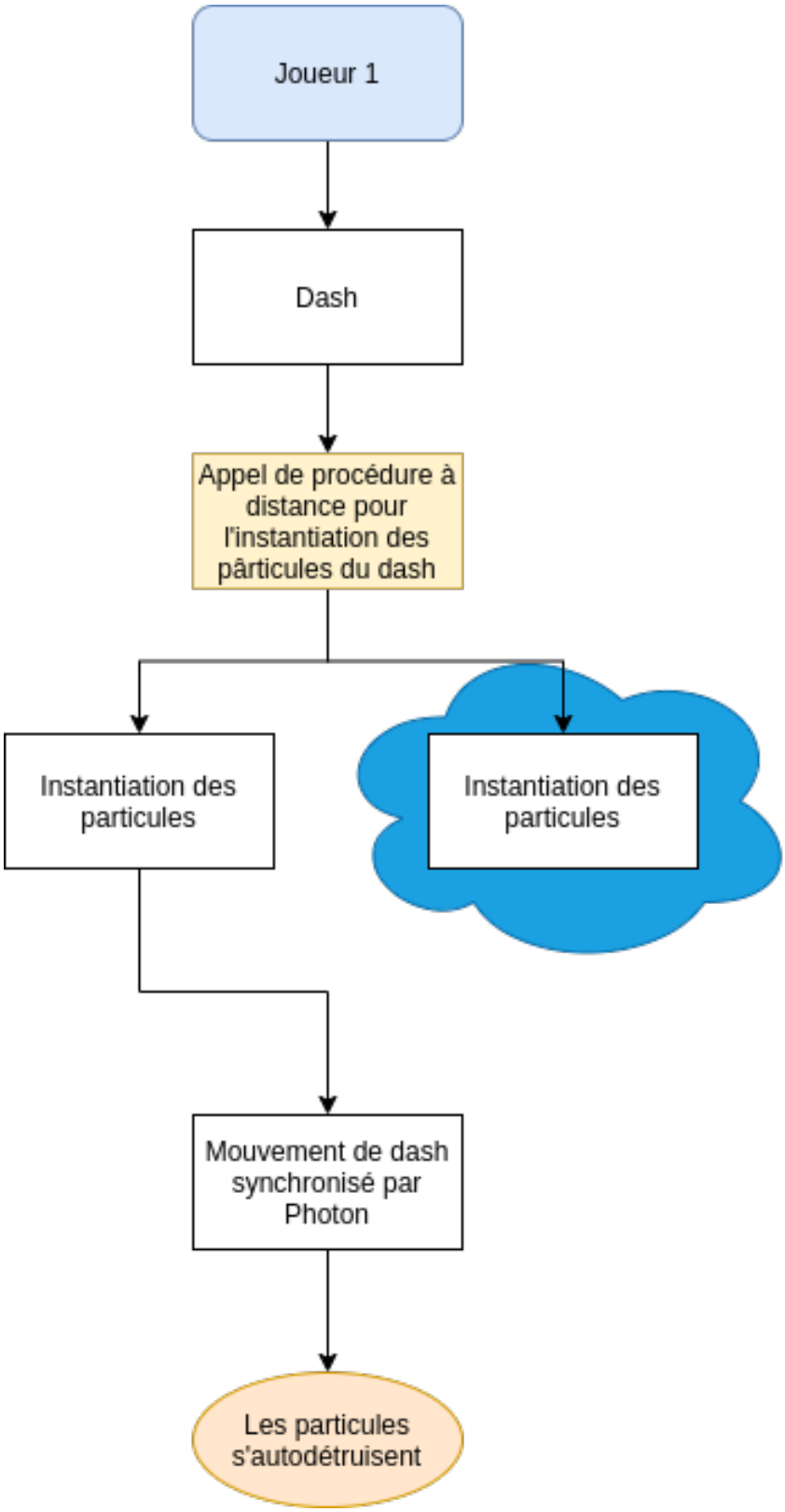
Flow chart de la synchronisation des dégâts infligés aux ennemis

Synchronisation des dégâts infligés aux joueurs via des appels de procédure à distance.



Flow chart de la synchronisation des dégâts infligés aux joueurs

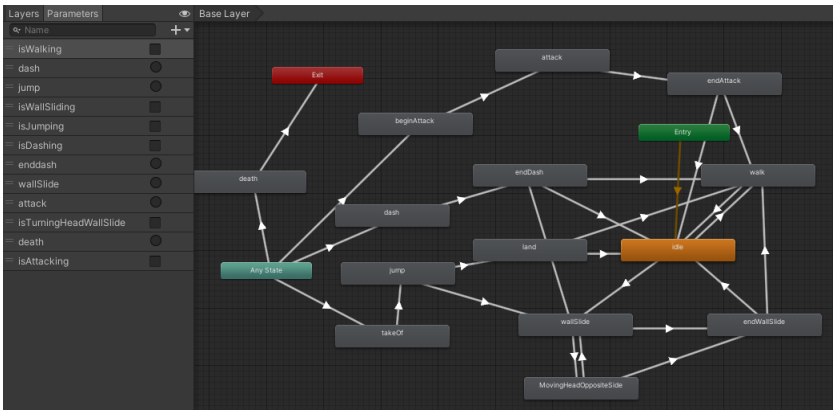
Ajout de la mécanique de propulsion en avant à l'aide de RPC



Flow chart du fonctionnement de la mécanique de propulsion en avant

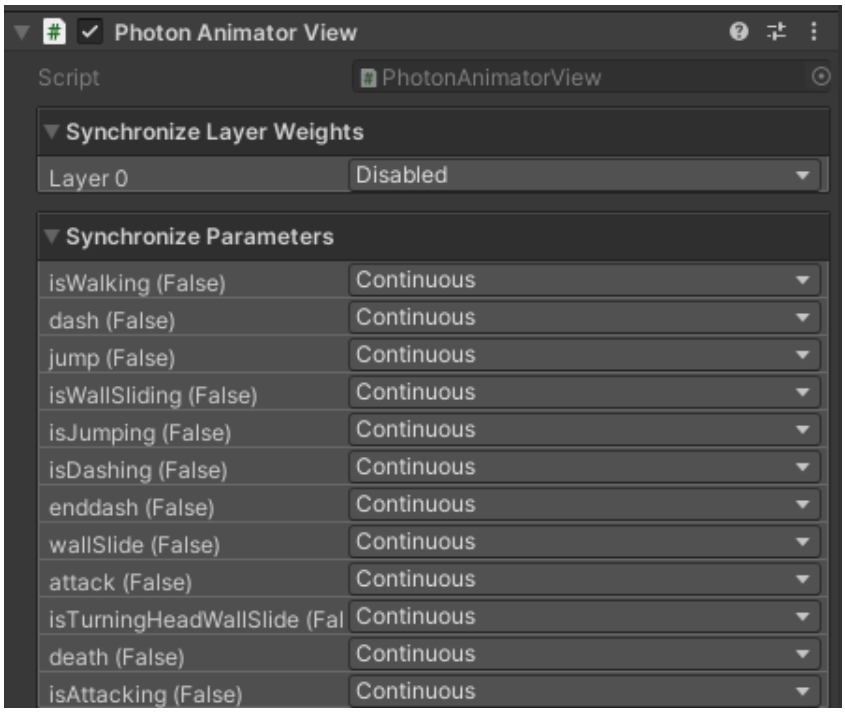
Synchronisation des animations

Une fois les animations créées il a fallu les déclencher au bon moment dans le code, mais aussi les synchroniser entre les deux joueurs. Il faut donc synchroniser les paramètres utilisés dans le régulateur d'animations.



Régulateur d'animations avec ses paramètres sur la gauche

Pour accomplir cela, rien de plus simple, il suffit d'ajouter un composant Photon-View à l'objet qui contient le régulateur d'animations et de demander à synchroniser tous les paramètres.



Flow chart du fonctionnement de la mécanique de propulsion en avant

La valeur *Continuous* implique que ces paramètres seront synchronisés à une très haute cadence, ce qui est exactement ce que nous souhaitons.

3.1.2 Site Web

Toan s'est vu attribuer la réalisation du site web. Notre site est consultable à l'adresse : https://fracturegame.github.io. Il a été réalisé à l'aide du générateur de sites statiques HUGO et du thème min_night.

Le site possède 6 pages :

- L'accueil, qui présente le projet
- Une page de téléchargements du jeu ainsi que ses instructions d'installation
- La présentation des membres du groupe
- L'historique de notre projet avec les liens permettant de télécharger le cahier des charges et les rapports à venir
- La chronologie de réalisation du projet avec les liens des outils utilisés
- Une version web de notre jeu

Le site possède également un en-tête qui sert de barre de navigation et un bas de page. Il s'adapte à la taille de l'écran sur lequel il s'affiche afin que tout le monde puisse le consulter sans gêne.

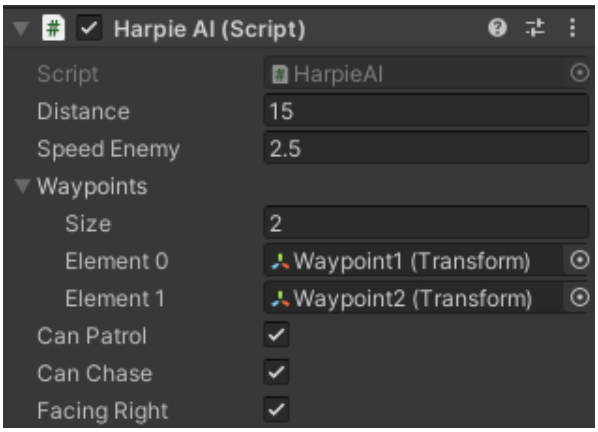
Le gain de temps apporté par l'utilisation d'un thème a pu profiter au reste du site puisque davantage de soin a pu être apporté à la création des pages en HTML et CSS. Il a tout de même fallu adapter le thème qui est pensé pour être utilisé en tant que blog.

3.1.3 Intelligence Artificielle

Harpie

La harpie est le deuxième ennemi de notre jeu et possède une intelligence artificielle plus poussée que le blob. Tout comme ce dernier, elle est capable de "patrouiller" entre deux points mais peut également partir à la poursuite d'un joueur se trouvant proche d'elle. Si la harpie parvient à toucher le joueur, elle lui infligera des dégâts. A l'inverse, si la distance entre le joueur et la harpie devient trop grande, la harpie fera demi-tour en direction de son point de départ et poursuivra sa patrouille. La harpie peut également passer à travers les murs.

De plus, il est possible de modifier son comportement en quelques clics depuis l'éditeur Unity comme le montre la capture d'écran qui suit :



Options responsables du comportement d'une harpie

En effet, si l'on souhaite, par exemple, que la harpie soit statique et chasseuse, il suffit de décocher la valeur *Can Patrol*. Il est également possible de choisir l'orientation avec laquelle la harpie débutera la partie.

Notre jeu étant divisé en deux mondes, il a fallu imposer certaines règles à la harpie afin qu'elle ne suive que le joueur présent dans son monde même si le joueur de l'autre monde est à une distance suffisamment petite pour être attaqué.

Enfin, la harpie réagit aux attaques des joueurs différemment du blob puisqu'elle est propulsée en arrière à chaque coup qu'elle encaisse. Elle peut tout de même être éliminé si un joueur lui saute dessus.

Harpie Bombardier

La harpie bombardier est une variante de la harpie basique. De fait, il s'agit d'une harpie capable de patrouiller mais pas de partir à la poursuite des joueurs. A la place, elle *bombarde* des particules en direction du sol qui jouent le rôle de mini-bombes et qui explosent au contact du sol ou d'un joueur.

De plus, ce genre de harpie est bien plus imposant que les harpies classiques et ne peut être vaincu par les joueurs. Le but est donc de dépasser la harpie pendant qu'elle recharge ses bombes afin de ne pas encaisser de dégâts.

Harpie Transporteuse

La deuxième variante de notre harpie classique se nomme la *harpie transporteuse*. Elle a été créée spécialement pour le niveau de fin et a pour objectif d'aider le boss de fin à s'enfuir. Afin d'accomplir cette mission, les harpies transporteuses sont munies d'un câble qui, une fois activé et relié au boss, permet à ce dernier de s'envoler avec elles. Elles ne peuvent donc ni patrouiller, ni poursuivre le joueur et attendent simplement que le boss de fin se joigne à elles.

Ce genre de harpie ne représente donc pas une menace directe aux joueurs mais doit tout de même être éliminé afin de faire chuter le boss.

Harpie Libre

Tout comme la harpie transporteuse, la harpie libre a aussi été spécialement implémentée pour le niveau de boss du jeu. En effet, puisque ce niveau regroupe les deux joueurs dans le même monde, il nous fallait une harpie capable de chasser n'importe lequel d'entre eux. Elle est donc plus *libre* que la harpie originale.

Siphon à feu grégeois

Durant cette dernière période de développement, nous avons aussi réalisé un troisième ennemi. Il s'agit d'un siphon à feu grégeois et peut être ramener à un canon qui propulse des boules de feu en direction d'un joueur proche de lui, à intervalles de temps réguliers.

Si le joueur est suffisamment proche du canon, ce dernier va modifier sa rotation afin de suivre son déplacement puis tirer une boule de feu en sa direction. Si la boule de feu touche le joueur, elle lui infligera des dégâts. Lorsque la boule de feu entre en collision avec un joueur ou le décor, elle est détruite et instancie quelques particules rouges pour simuler son explosion.

Enfin, puisque notre jeu possède deux mondes distincts, il a fallu programmer le siphon pour qu'il ne vise que le joueur présent dans son monde, même quand le joueur visé utilise la mécanique d'échange.

Boule de feu grégeois

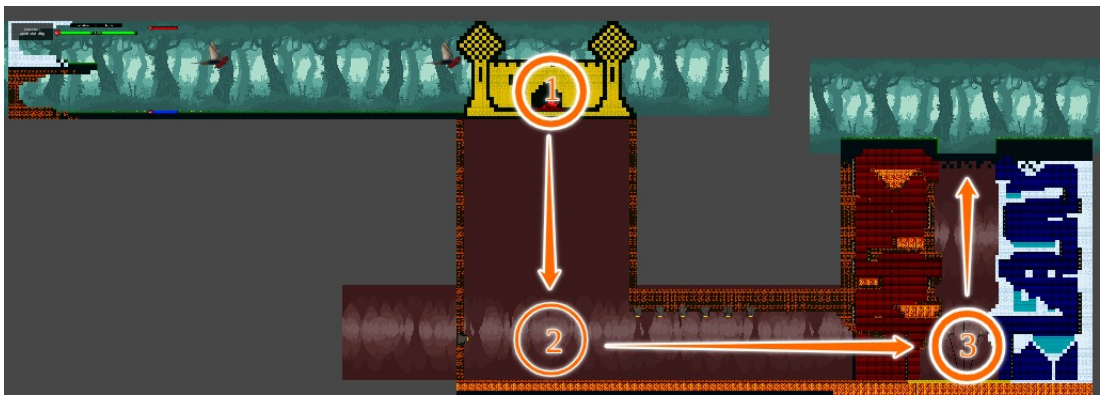
La boule de feu grégeois se contente de suivre la direction que lui a indiqué le canon et est détruite lorsqu'elle entre en collision.

Siphon à feu grégeois libre

Tout comme la harpie libre, le siphon à feu grégeois libre peut viser n'importe lequel des deux joueurs. Il est utilisé dans le niveau de fin et a une taille augmentée ainsi qu'une force de poussée plus importante que le siphon basique.

Roi Blob

L'intelligence artificielle du roi blob consiste en 3 phases :



Les différentes phases de l'IA du roi Blob dans le niveau de fin

- **Phase 1**

  Le roi Blob attend dans son château qu'un des deux joueurs franchisse le seuil de sa demeure. Il se sent alors en danger et, pendant que plusieurs harpies libres accourent pour le protéger, saute à deux reprises afin de détruire son château et de s'évader par le souterrain. A chaque saut, des tuiles du château sont détruites et des débris surgissent du point d'atterrissage du boss. De plus, si l'un des joueurs se trouve sous le boss lorsqu'il atterit, le joueur en question est instantanément mis en échec.

- **Phase 2**

  Le roi Blob entraîne les joueurs dans sa chute. S'ensuit une course-poursuite dans le souterrain qui contient de nombreux siphons à feu grégeois libres. Le blob continue sa course jusqu'à arriver à son *point d'extraction*.

- **Phase 3**

  Il s'agit d'un point à partir duquel des harpies transporteuses peuvent l'aider à s'échapper en l'élevant en direction de la sortie du souterrain. Le blob s'arrête donc à cet endroit et se laisse porter. Si, au cours de son ascension, les joueurs parviennent à éliminer les harpies transporteuses, le roi blob s'écrasera alors au sol, le détruisant par la même occasion. C'est la lave qui se trouvait en dessous du sol qui mettra fin à son existence.

  Au contraire, si les joueurs ne parviennent pas à vaincre les harpies transporteuses, le blob s'échappera du souterrain et les joueurs auront alors perdu.

3.1.4 Niveau de fin

Le niveau de fin est le lieu de l'affrontement entre les deux joueurs et le boss, à savoir, le roi Blob. Il s'agit du seul niveau dans lequel les deux joueurs se retrouvent dans le même monde. Ce niveau peut être divisé en 4 parties :

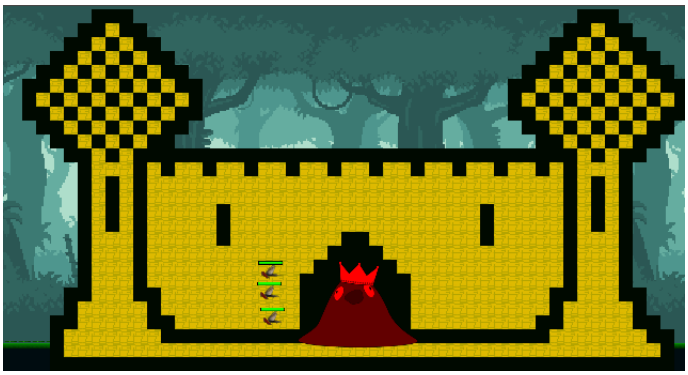
- **Partie 1**

  Les deux joueurs apparaissent pour la première fois dans le même monde. En haut à gauche de l'écran, un texte indique que leur objectif est de rejoindre le boss. Cependant, deux harpies bombardier se trouvent sur leur chemin. Les joueurs devront donc profiter des moments où les harpies rechargent leurs bombes pour avancer jusqu'à la partie 2.

Première partie du niveau de fin

- **Partie 2**

  Une fois rentrés dans le château du blob, les joueurs sont attaqués par plusieurs harpies tandis que le roi s'éloigne du combat. Ce dernier détruit alors sa demeure en deux sauts pour s'enfuir dans son souterrain. Les joueurs tombent avec lui. A noter que le roi détruit des tuiles de son château à chaque saut et peut mettre en échec un joueur s'il lui atterrit dessus.

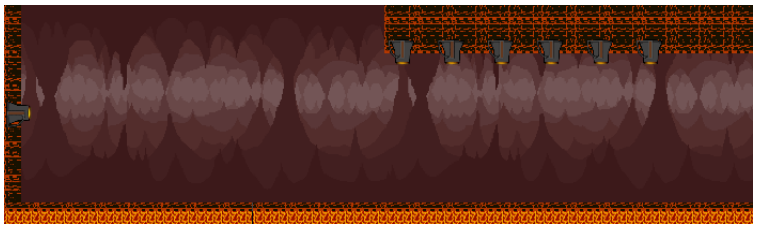


Deuxième partie du niveau de fin avant destruction

Deuxième partie du niveau de fin en cours destruction

- **Partie 3**

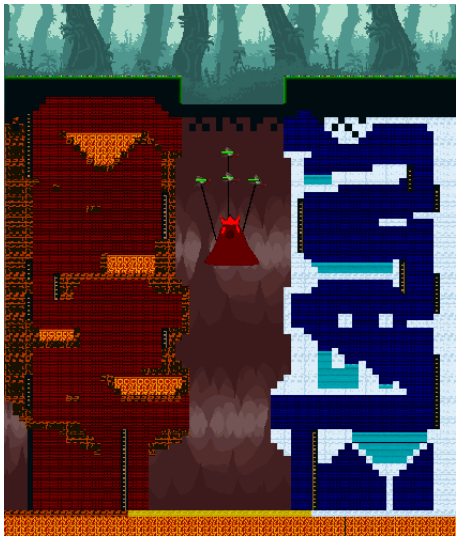
  Débute alors une course-poursuite dans le souterrain qui présente plusieurs siphons à feu grégeois libres. La troisième partie prend fin lorsque le roi Blob arrive à son point d'extraction.



Troisième partie du niveau de fin

- **Partie 4**

  C'est dans la quatrième et dernière partie du niveau que l'on retrouve l'aspect coopératif de *Fracture*. En effet, le boss étant arrivé à son point d'extraction, il est emporté par des harpies transporteuses et ne peut être mis en échec que si les deux joueurs s'entraident. Ils devront grimper de blocs en blocs et échanger de position lorsqu'un obstacle n'est pas franchissable pour leur personnage.

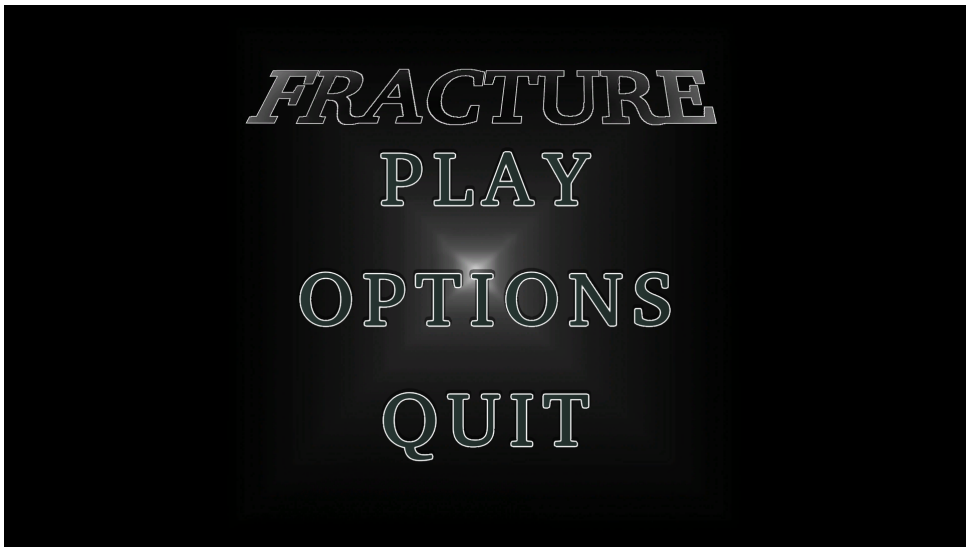


Dernière partie du niveau de fin avec le boss transporté par les harpies

3.2 Raphaël

3.2.1 Interface Graphique

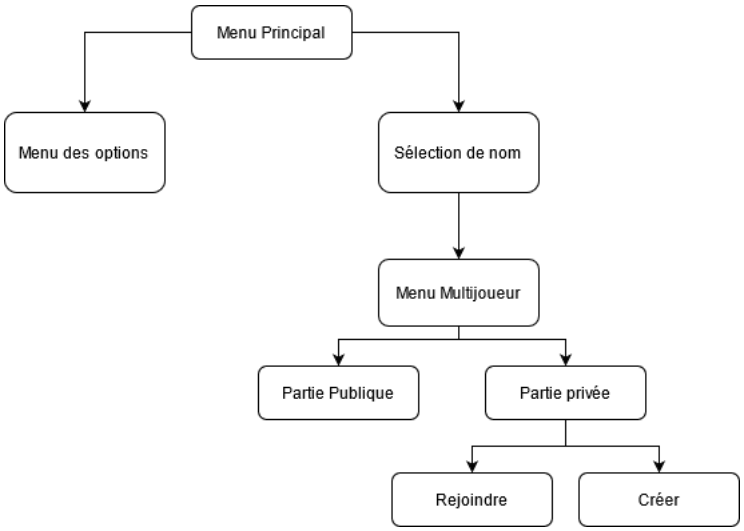
L'interface graphique a au début été faite en parallèle de la partie multijoueur, dans un projet Unity différent. C'était une erreur de début, mais nous avons rapidement réussi à corriger et à travailler sur le même projet. Celle-ci comporte un menu d'options qui à la première soutenance permettait uniquement de changer la résolution du jeu, quitter le jeu ainsi que différents menus qui sont directement liés au multijoueur. Vous pourrez trouver plus de détails sur la liste des menus dans l'organigramme en page suivante.



Menu principal

Menu Principal

Le menu principal, le menu des options et le menu de sélection de nom sont sur la même scène et sont affichés / effacés à l'aide de la méthode d'Unity *gameObject.SetActive(bool)* qui est appelée lorsqu'on clique sur l'un des boutons. Chaque élément d'un menu est en effet "enfant" d'un gameObject vide (sauf dans certains cas où il aura un script CSharp attribué), qui est activé ou désactivé en fonction du bouton cliqué par le joueur (par exemple, cliquer sur le bouton "Play" désactivera le menu principal et activera la sélection de nom, et le bouton "Back" désactivera la sélection et activera le menu principal).



Cheminement des menus

Menu du multijoueur

Ensuite, lorsque le nom a été choisi, on change de scène pour arriver sur celle des menus de jeu en ligne. Ici aussi, les différents sous-menus sont des gameObject parents des éléments de chaque menu.

Nous ne détaillerons pas beaucoup le fonctionnement du menu réseau car il relève plus de la partie multijoueur de Toan, cependant il peut être intéressant de parler de celui des options. Plus d'options s'y sont ajoutées avant la dernière soutenance, lors de la première il était uniquement possible de changer la résolution du jeu ainsi que de choisir s'il s'affichera en plein écran ou en mode fenêtré. Ceci était effectué à l'aide d'une classe Resolutions avec 2 attributs de taille d'écran, ainsi que d'un tableau de résolutions et de la méthode *Screen.SetResolution* de Unity. Le code fut ajusté plus tard pour éviter la séparation entre la liste déroulante et la liste des résolutions, et car il y avait plusieurs problèmes liés à cette classe, qui fut abandonnée plus tard au profit d'un code plus clair et fonctionnel.

Menu de sélection de niveau

Nous avons un écran de sélection de niveaux, qui est généré à partir d'un dictionnaire de niveaux et de leur *buildIndex*, ce qui nous évite de devoir refaire ce menu manuellement si de nouveaux niveaux sont amenés à être ajoutés, ou certains à être supprimés.

Il est possible de quitter ce menu à tout moment, ce qui aura pour conséquence de faire quitter la salle multijoueur aux deux joueurs, pour éviter des conflits si quelqu'un d'autre essayait de rejoindre la salle alors qu'une des personnes aurait quitté.

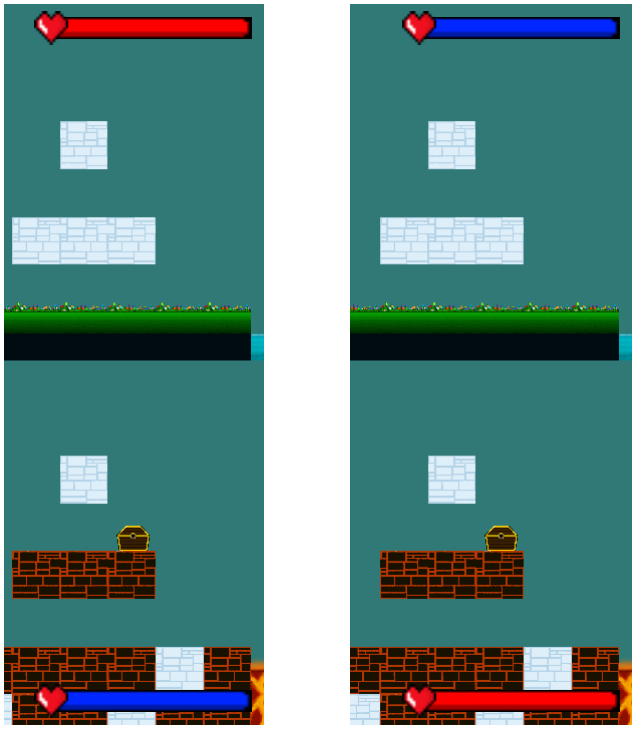
```
public static readonly Dictionary<string,int> scenes = new Dictionary<string,int>()
{
    {"Level 1",3},
    {"Level 2",4},
    {"Level 3",5}
};
```

Dictionnaire de niveaux

Sélection de niveau appelée avec le dictionnaire du dessus

Barre de vie

Un système de vie a également été implémenté avant la deuxième soutenance. Son fonctionnement est relativement simple, avec une fonction *TakeDamage(int i)* qui peut être appelé depuis n'importe quel script. Celle-ci est représentée par une barre de vie comme dans la plupart des jeux, et nous avons fait en sorte que lorsque les deux joueurs échangent de place, celles-ci échangent aussi pour éviter la confusion entre les deux joueurs. Nous avons utilisé le système de RPC afin de synchroniser les barres chez les joueurs, sinon la barre de vie d'un joueur ne se mettait à jour que de son côté, ce qui n'est pas très pratique dans le cas d'un jeu multijoueur.



Barres de vie avant/après échange de position des 2 joueurs

Menu des options

Le menu des options a des refontes importantes depuis la première soutenance : tout d'abord, nous avons abandonné la classe Resolutions précédemment citée, au profit de l'utilistation de fonctions basiques du module UI d'Unity. Cela nous a permis de corriger le problème d'incohérence entre l'affichage des options et la réalité lorsque l'on quittait et rouvrait le jeu (si on avait changé la résolution, le changement persistait entre les lancements mais pas l'affichage de l'option).

Nouveau menu d'options

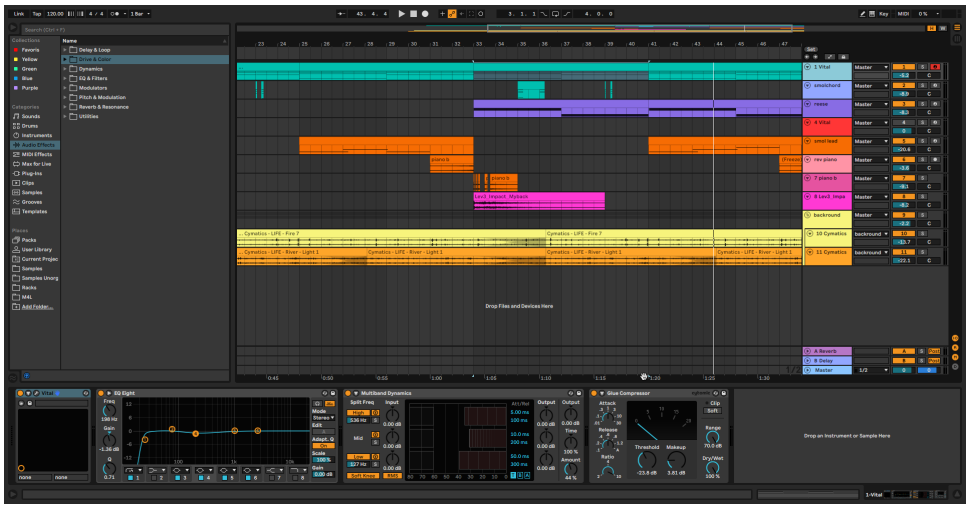
Nous pouvons également remarquer l'ajout le plus important au menu, la configuration de touches. Nous avons dû créer un gestionnaire d'entrée nous-mêmes afin de pouvoir assigner certaines actions à certaines touches, et de pouvoir les réassocier plus tard. Ce gestionnaire garde donc un dictionnaire qui associe a chaque action (représentée par une *string*) un *KeyCode*, qui est donc le paramètre qui se change lorsqu'on change l'assignement d'une touche.

Nous avons également fait en sorte que le joueur ne puisse pas assigner la même touche à deux actions différentes. Il a cependant été impossible de réassigner les touches de déplacement gauche et droit car nous nous sommes servis de l'axe horizontal d'entrée géré par Unity directement, et qui n'est pas surchargeable par des scripts. Nous avons également pu, à l'aide des classes *StreamReader* et *StreamWriter* que nous avons pu découvrir en TP grâce à nos chers ACDC, faire persister les changements de touches. Lorsqu'on cliquera sur le bouton "Back", le jeu enregistrera la configuration sous le format "Action :Touche", et chaque fois que le jeu est lancé, il s'occupera d'aller chercher ce fichier s'il existe et de désérialiser la configuration, sinon d'appliquer celle par défaut.

3.2.2 Audio

Musique de menu

A la première soutenance, nous n'avions qu'une musique de menu, et pas de musique pour les niveaux. C'était la première fois que je devais faire de la musique dans un objectif concret plutôt que d'en faire simplement, et ceci s'est avéré être une tâche plus difficile que je ne le pensais.



Projet du menu principal

Toutes les musiques ont été réalisées sur le logiciel Ableton Live. Pour implémenter la musique (et effets sonores du jeu), nous nous sommes inspirés d'une vidéo de Brackeys afin de créer un petit utilitaire qui sert simplement à jouer / arrêter des sons grâce à une simple commande qui passe par un gameObject présent dans une scène qui ne se décharge jamais.

Nous avons aussi créé une classe Sound qui nous permettra de mieux traiter nos différents fichiers sonores et de leur attribuer une liste définie de paramètres basiques pour les manipuler (volume, ton et capacité à boucler). Ainsi, si on veut jouer un son, on aura simplement à faire

```
FindObjectOfType<AudioManager>().PlaySound("Nom du son");
```

Et à l'inverse pour arrêter la lecture d'un son

```
FindObjectOfType<AudioManager>().StopSound("Nom du son");
```

A noter que par souci d'optimisation on peut également assigner l'AudioManager à une variable afin de ne pas avoir à le chercher à chaque fois.

Effets sonores et première musique de niveau

Nous avons maintenant une nouvelle musique de niveau qui vient s'ajouter à celle du menu, qui était déjà présente lors de la deuxième soutenance. C'est la première fois que nous nous essayons à la composition pour un jeu vidéo et il n'a été possible que d'en ajouter une jusqu'à cette deuxième soutenance ; nous sommes cependant satisfaits du résultat.

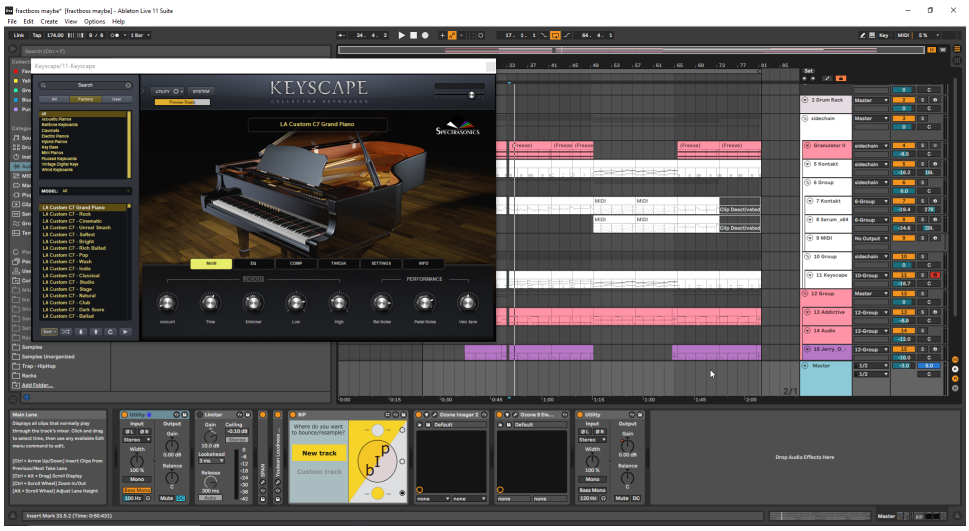
Nous avons modifié notre AudioManager de la première soutenance afin de pouvoir gérer les effets sonores et de pouvoir superposer certains sons et pas d'autres à l'aide d'attributs booléens. Nous avons maintenant des effets sonores pour la marche, le saut et l'accélération. A l'exception de la marche qui a été récupérée sur internet, les effets sonores ont été synthétisés avec Vital, un synthétiseur logiciel gratuit.

Capture d'écran de Vital (Saut)

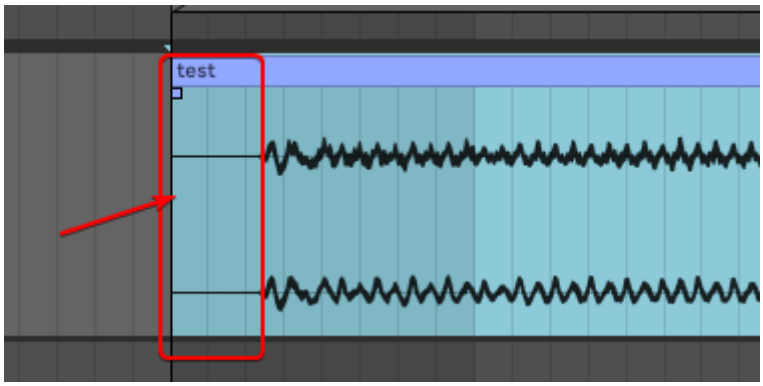
Dernières musiques

Pour la dernière soutenance, nous avons pu réaliser deux musiques de plus afin de satisfaire au mieux les objectifs que nous avions fixés dans le cahier des charges. Nous avons maintenant une musique pour le boss (qui a été détaillé dans la partie de Toan), et une musique de niveau supplémentaire afin de varier. La musique du boss était pour moi l'occasion d'expérimenter avec une signature rythmique moins commune (9/8), et d'essayer de composer sans me fixer des règles strictes (comme une sélection de notes qui vont bien ensemble) pour changer. Je suis satisfait du résultat final, bien que je pense pouvoir faire mieux si je réessaye.

La musique de boss est en réalité constituée de deux fichiers audio séparés, car il ne pouvait pas boucler dans le cas contraire. Ce problème m'a permis de me rendre compte d'un problème avec le format audio mp3 : il induit un micro-décalage au début de chaque son. Ceci n'est pas un problème dans la plupart des cas, cependant dans l'objectif de faire un son qui boucle parfaitement, cela induit une transition moins fluide avec un "click" très audible lorsque l'audio finit une boucle. Pour remédier à ce problème, j'ai dû réexporter toutes les musiques en WAV, puis les réencoder en OGG (Vorbis) pour garder une taille raisonnable.



Projet de la musique du boss



Microdécalage induit par le format mp3

3.3 Adam

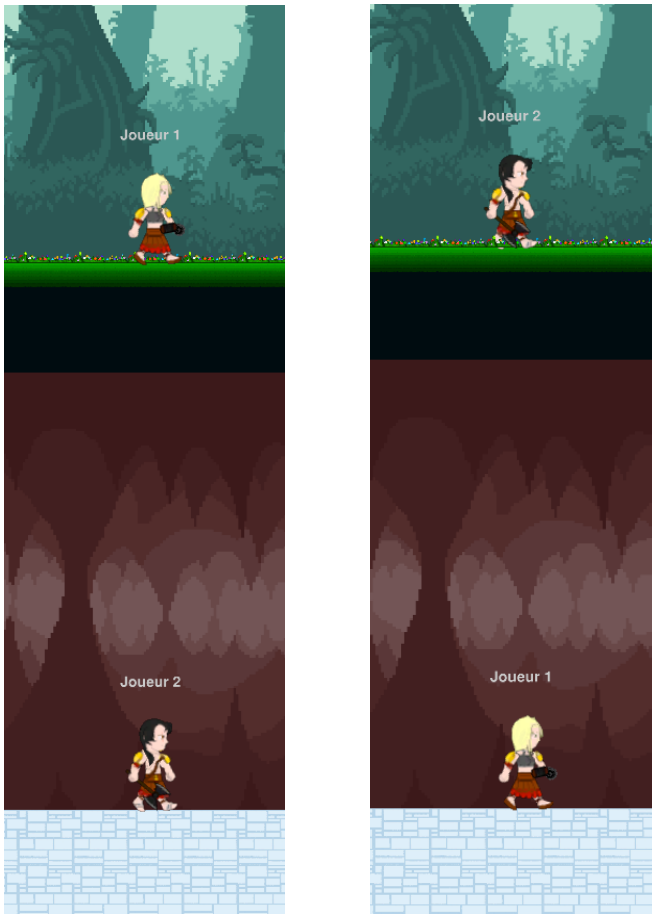
3.3.1 Construction des mécaniques

Marche La première mécanique implémentée du jeu a été la marche. Elle permet simplement de se déplacer sur l'axe horizontal. Le joueur s'arrête de bouger dès que la touche est relâchée et se déplace dès qu'elle est enfoncée, il n'y a donc pas de délai.

Saut Vient ensuite la mécanique de saut qui permet au joueur de se déplacer sur l'axe vertical. Une fois en l'air, il est tout de même possible de se déplacer sur l'axe horizontal mais avec une vitesse réduite. Cette mécanique est assignable dans le menu des contrôles.

Attaque L'attaque permet au joueur d'infliger des dégâts aux ennemis. Elle a une portée limitée. Cette mécanique est assignable dans le menu des contrôles et possède un temps de rechargement de 1 seconde.

Echange de position Il s'agit de la mécanique principale du jeu. Elle permet aux deux joueurs d'échanger leur position. Elle est également assignable dans le menu des contrôles et possède un temps de rechargement partagé entre les deux joueurs.

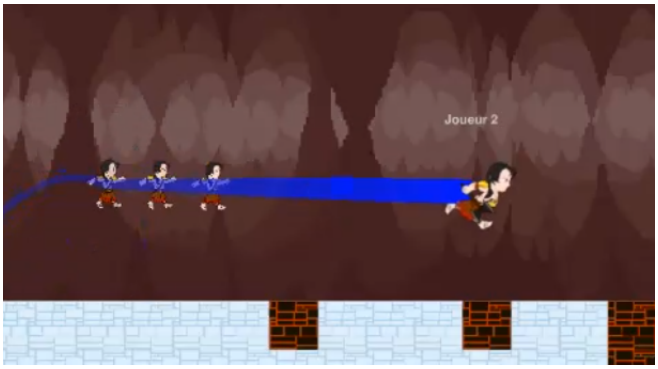


Position des joueurs avant/après activation de la mécanique d'échange

Double saut Comme son nom l'indique, il s'agit de la mécanique de saut actionnée à deux reprises. Cette mécanique n'est utilisable que par le joueur ayant le personnage féminin. Elle est elle aussi assignable dans le menu des contrôles.

Mécanique du double saut

Propulsion en avant Cette mécanique permet au joueur contrôlant le personnage masculin de se propulser vers l'avant à toute vitesse. Elle possède un temps de rechargement de 1 seconde et laisse des particules du joueur derrière elle.



Mécanique de propulsion en avant

Escalade Il est possible pour chacun de joueurs d'escalader les parois qui comportent une échelle sur un fond noir. Afin de grimper, il faut s'accrocher au mur et sauter en même temps. Il est également possible de glisser le long de l'échelle. Pour cela il suffit de rester accroché à la paroi.

Mécanique d'escalade

Saut depuis l'échelle Tout en escaladant ou en se laissant tomber le long d'une échelle, il est possible de réaliser un long saut en se retournant tout en gardant enfoncé la touche assignée à l'escalade. Le joueur réalisant cette action verra son personnage de retourner, comme s'il était prêt à sauter.

Mécanique de saut depuis l'échelle

3.3.2 Modification de mécaniques

Pendant cette dernière période de travail nous avons modifié des mécaniques, par exemple la propulsion a vu sa distance modifiée et le saut mural n'est possible que sur certaines parois définies par des échelles sur des blocs noirs.

3.3.3 Rappels sur les premiers niveaux et leur conception

Fracture repose sur un écran scindé en permanence. De ce fait, nous ne pouvons pas utiliser pleinement les axes X et Y. De plus, nous sommes obligés de placer des éléments bloquants pour les joueurs, les obligeant à échanger leurs positions. Cette restriction se fera sous la forme de deux niveaux asymétriques ne proposant pas les mêmes obstacles à passer. C'est ce qui permet d'avoir une rejouabilité mais surtout, c'est ce qui rend notre jeu unique. Notre jeu se base donc sur l'asymétrie des parcours des deux joueurs qui doivent donc échanger leurs positions pour progresser. Cette mécanique peut sembler restrictive autant pour les joueurs que les concepteurs, cependant cela nous offre une possibilité de créations originales et assez exceptionnelles, et pour le joueur cela le satisfait sur tous les points. Pour les joueurs qui cherchent une expérience amusante, ils pourront entre amis user des mécaniques et des coups fourbes, comme par exemple échanger pile au-dessus de la lave. Mais pour les joueurs qui voudront jouer au jeu plus sérieusement en cherchant à le faire le plus vite possible seront également servis car il y a beaucoup de possibilités pour optimiser leur parcours. Notre mécanique est donc de qualité pour tous types de publics.

Niveau 1

Il s'agit du niveau le plus simple du jeu. Les joueurs sont censés échanger de position au moins 2 fois pour venir à bout de ce niveau. Les ennemis présents sont

des blobs et des harpies. Dans ce niveau, la caméra suit les deux joueurs en même temps pour les contraindre à rester ensemble et ainsi renforcer leur coopération.

Début du niveau 1

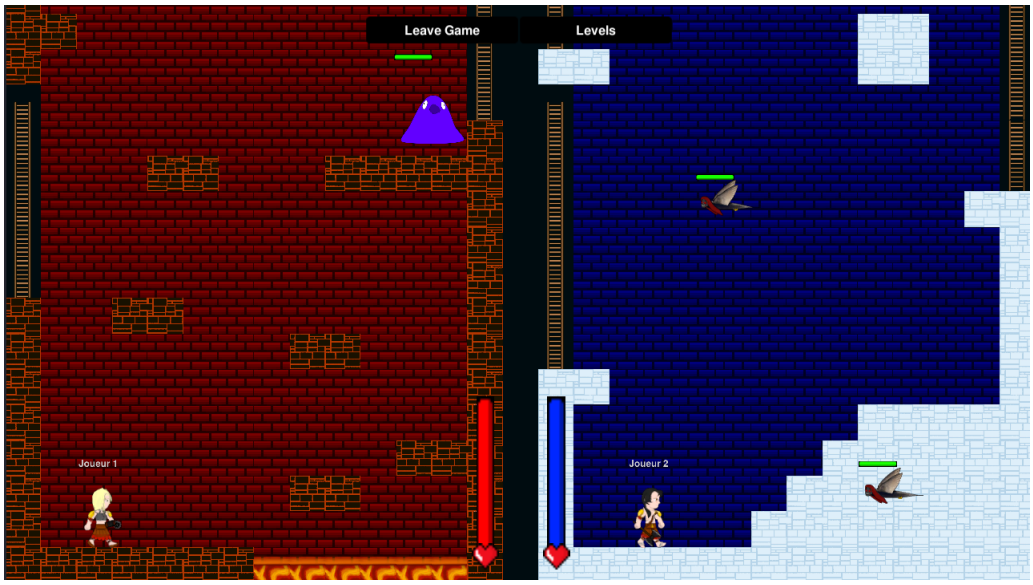
Niveau 2

La difficulté augmente d'un cran au deuxième niveau du jeu. Les joueurs doivent faire usage de la mécanique d'échange plus souvent. On remarquera aussi la présence de canons pour la première fois. Tout comme dans le niveau 1, la caméra contraint les joueurs à rester ensemble.

Début du niveau 2

Niveau 3

Le niveau 3 est le premier niveau vertical. Ici, la caméra suit chaque joueur individuellement. Il est donc normal de ne plus voir l'un des deux joueurs au cours du niveau.

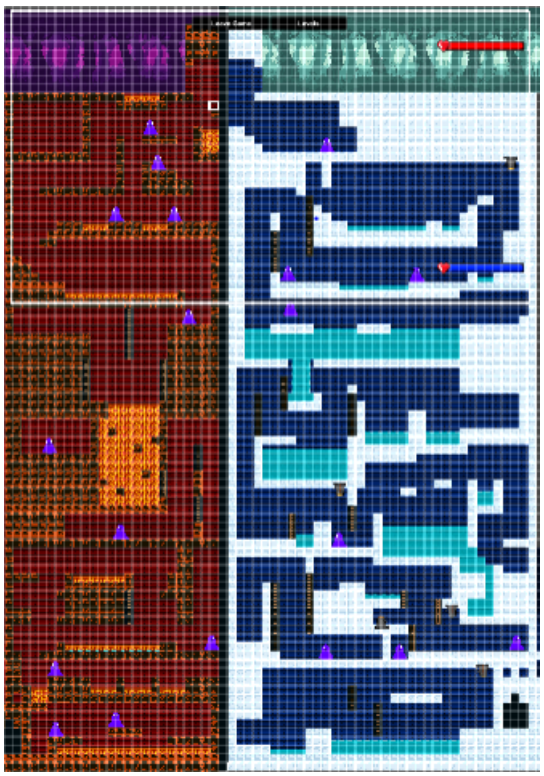


Début du niveau 3

3.3.4 Nouveaux niveaux

Lors de cette dernière période de travail nous avons implémenté 2 niveaux, ces deux niveaux apportent leur lot de nouveautés et se démarquent par rapport aux autres niveaux de par leur difficulté et leurs concepts.

Niveau 4



Ce niveau est le plus atypique de tous, tout d'abord vous pouvez voir sur l'image ci-dessus que le niveau commence en haut et finis en bas. De plus, vous pouvez voir qu'il exploite pleinement les deux axes X et Y . Ensuite, l'asymétrie du niveau est encore plus visible que sur n'importe lequel des autres. De plus le niveau est truffé d'ennemis, nous avons face à nous le niveau avec le plus d'ennemis ce qui le rend d'autant plus difficile. Mais il y a 2 spécificités de plus a ce niveau que vous pouvez observer ci-dessous.

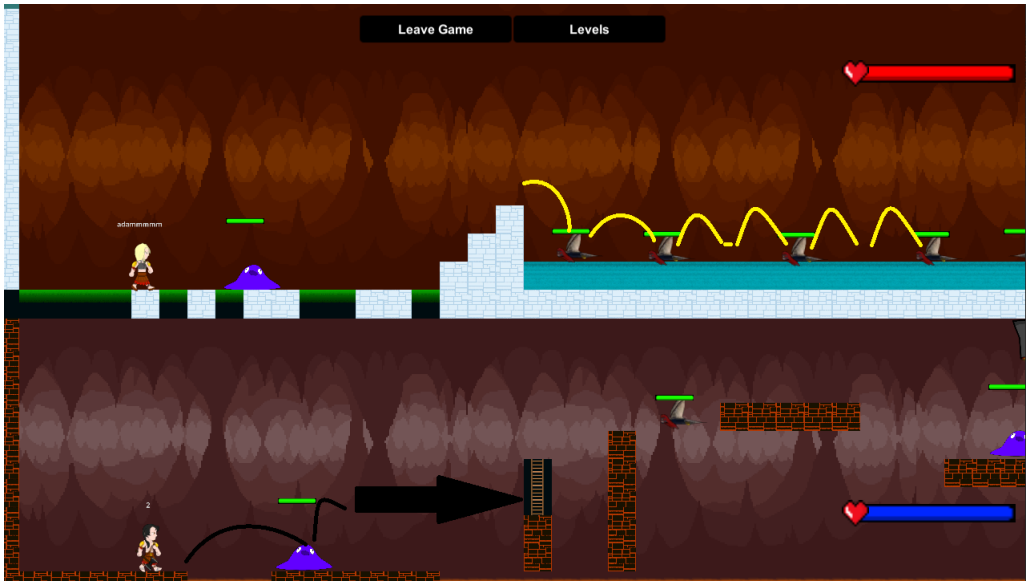
La première d'entre elles est l'apparition de chemins alternatifs que vous pouvez observer avec les chiffres de couleur bleue et rouge. Le joueur peut emprunter les chemins 1 ou 2 puis 1.1 ou 1.2 ou 3, cela offre de la liberté au joueur et donc plusieurs possibilités pour finir ce niveau, mais chaque choix apporte son lot de complication. Par exemple au premier coup d'oeil suivre le chemin 1 puis 1.1 amène à la route avec le moins d'ennemis, cependant le chemin 1 est difficile à suivre au début car le blob ne peut pas être tué en lui sautant dessus, il faut donc lui mettre plusieurs coups. De plus, on rencontre plus d'harpies à la fin. Le 3 quant à lui est plus simple au début, mais le second blob est assez difficile à tuer, cependant il y a moins d'harpies à tuer sur la fin. Le 2 est un entre-deux. Ensuite sur la partie bleue, vous pouvez observer quelque chose de spécial. Certes, il y a deux chemins, mais à première vue, ils ont l'air d'être impossibles à emprunter car il y a un mur. C'est ici la seconde nouveauté de ce niveau : les faux sols. Cela pourra surprendre le joueur tout en lui donnant une sensation de vitesse. Par ailleurs, cela donne de possibles passages secrets qui pourraient permettre de passer de grandes parties du niveau. Néanmoins, ces derniers sont durs à trouver, il faudra donc un investissement du joueur pour les découvrir.

Ces éléments font de ce niveau le plus atypique du jeu.

Niveau 5

Ce niveau revient sur des bases plus simples, on repart sur un niveau horizontal qui va de gauche à droite, cependant il apporte quelques subtilités.

Vous pouvez voir sur cette image que le niveau reprend la base du jeu : les échanges pour passer les obstacles. Cependant, il ajoute sa spécialité : donner la possibilité aux meilleurs joueurs et aux connaisseurs du niveau de passer certains obstacles en faisant preuve de beaucoup d'adresse. Sur cette image, vous pouvez voir les trajets les plus optimaux.



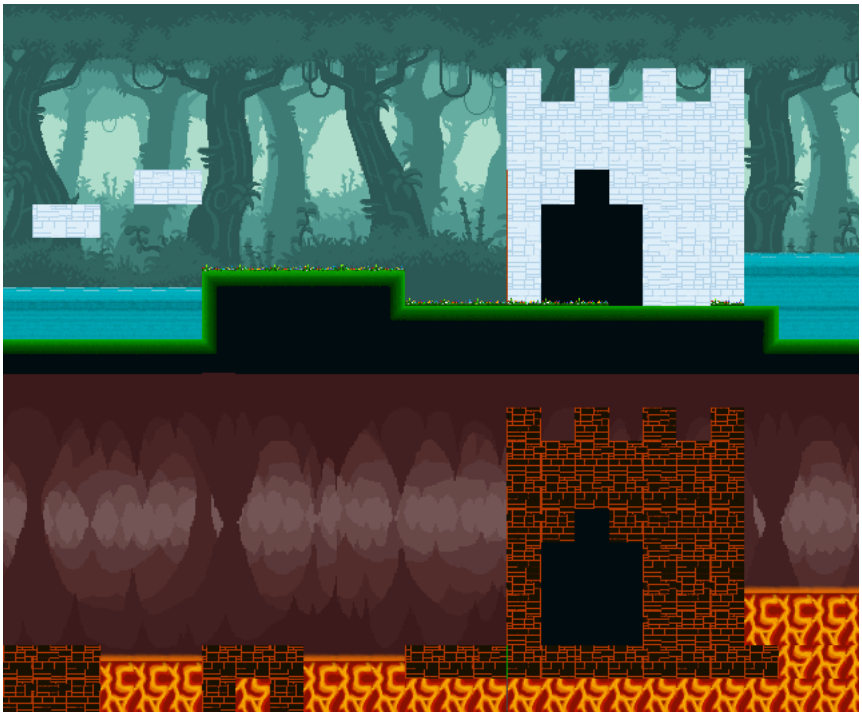
Il est possible de passer cet obstacle, cependant il est très dur de le faire. Cela va dans la continuité de la philosophie dictée plus haut. Tous les types de joueurs peuvent s'y retrouver. Enfin, ce niveau est plus dur que tous les autres, ce qui est assez logique étant donné que c'est le dernier avant celui du boss.

Enfin les niveaux suivent une certaine cohérence au niveau de la manière dont ils sont parcourus, on passe d'une plaine, en passant par une grotte, à une tour à escalader, puis à une autre tour à descendre et enfin à une dernière plaine finale avant la fin de ce jeu .

3.3.5 Fin de niveaux

A chaque fin de niveau se trouve un petit château. Ce dernier est composé de plusieurs couches de tuiles. La couche se trouvant sur la droite du château recouvre le joueur pour lui donner l'impression qu'il est à l'intérieur.

Lorsque les deux joueurs sont à l'intérieur du château, le jeu le détecte et affiche l'écran de victoire. Un système de particules est également activé dans chaque château.



Fin du niveau 1

3.4 Maëlys

3.4.1 Graphismes

Les graphismes du jeu actuel sont tous originaux. Ils ont été réalisés par Maëlys, avec le logiciel gratuit Gimp.

Personnages et armes

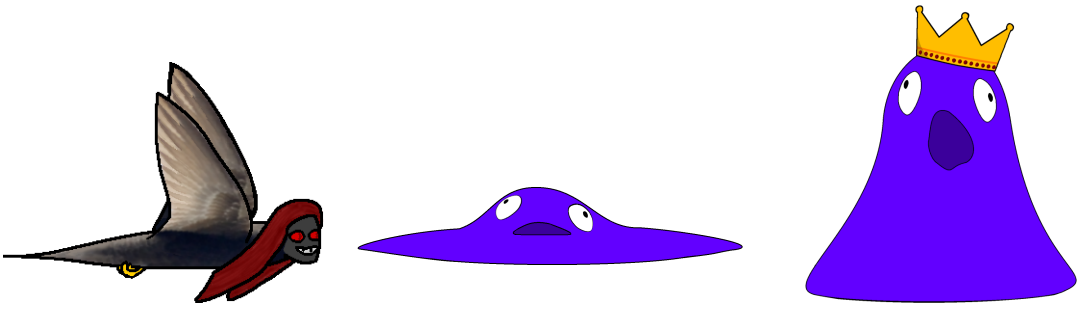
Nous avons à présent 2 personnages fonctionnels et possédant chacun une arme. L'implémentation du second personnage a été plus rapide que celle du premier, grâce à l'expérience accumulée au cours du projet. Cependant, l'arme du second joueur, un fouet, a causé de nombreux problèmes d'animation lors de son implémentation. Le lutin de chaque joueur est "découpé" en plusieurs calques correspondant aux membres du personnage, ce qui permet de les animer plus facilement. Il en est de même pour le fouet, qui est constitué d'un grand nombre de calques correspondant à toutes ses articulations.

Premier personnage et sa hache

Second personnage et son fouet

Ennemis

Depuis la dernière soutenance, la harpie a été implémentée, ainsi que le blob king. Au total, nous avons donc 2 ennemis et un boss. Nous avons peu à peu abandonné l'idée de créer 2 versions de chaque ennemi (un pour le "monde du haut" et un pour le "monde du bas") car cela aurait été très chronophage. Enfin, nous avons ajouté le feu grégeois, une arme de l'Antiquité, qui s'apparente à un canon tirant des boules de feu.

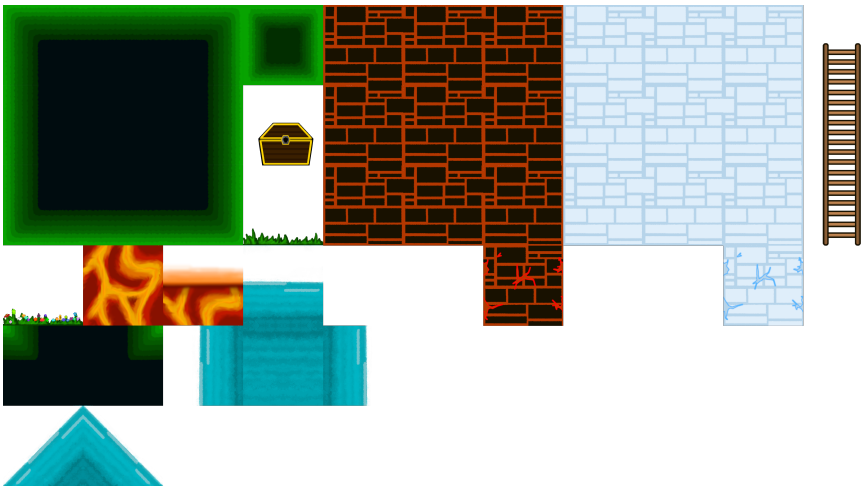


La harpie, le blob et le blob king

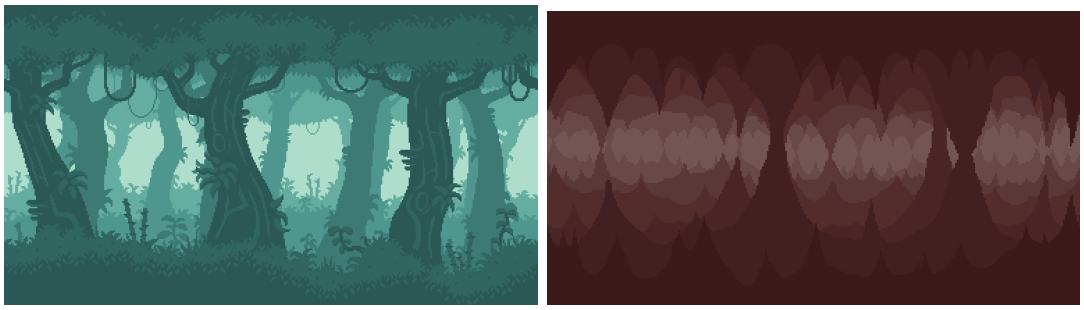
Le feu grégeois

Niveaux

L'ensemble de tuiles a évolué au fil des soutenances et des besoins. Nous avons maintenant à disposition des blocs agrippables (qui n'ont pas ou peu été utilisés), ainsi que des variations de blocs d'eau. Les fonds des niveaux sont la seule partie graphique qui n'est pas originale : ils viennent d'Internet. Nous les avons sélectionnés afin qu'ils correspondent à l'ambiance voulue pour les niveaux.



L'ensemble de tuiles final



Les fonds des niveaux

Menus

Le fond noir et blanc des écrans de menu était au départ temporaire. Finalement, nous avons décidé de l'adopter définitivement. Les images du menu se trouvent dans la partie de Raphaël.

Jaquette

La jaquette est l'un des derniers éléments à avoir été réalisés. Elle utilise des graphismes du jeu et a été approuvée par le reste du groupe.

La dernière version de la jaquette

3.4.2 Animations

Maëlys a réalisé certaines des animations, comme le saut des personnages et l'animation du fouet. Elle a créé la structure initiale des os et des poids qui ont servi comme base aux animations (voir illustration en annexe).

Comme dit dans la partie "Graphismes", les animations ont nécessité un découpage particulier du personnage. En effet, chaque membre doit être séparé du reste du corps, afin qu'il puisse être indépendant. Ce découpage est fait automatiquement par Unity grâce au paquet "PSD Importer". Gimp ne supportant pas le format ".psb", il a fallu renommer les fichiers graphiques avec cette extension. Par la suite, en les exportant dans Unity, chaque calque est automatiquement transformé en une "partie" du personnage.

Par la suite, il a fallu créer les "os" du personnage. Cela se fait grâce au "Skinning Editor" de Unity. Sur l'image en annexe, les traits de toutes les couleurs représentent les os. Chaque membre du personnage est pourvu d'un os qui permettra de le faire bouger dans le cadre des animations. On peut également voir que le personnage est multicolore : il s'agit des poids, qui déterminent la dépendance de chaque membre par rapport aux os. Par exemple, la tête est uniquement dépendante de l'os vert, ce qui signifie que modifier la rotation de l'os bleu n'aura pas d'influence sur la tête. La modification de certains os peut entraîner des déformations ; par exemple, si on bouge un avant-bras du personnage, le reste du bras se "plie" pour le suivre. Les

déformations ainsi créées dépendent de la géométrie du lutin, et donc des vecteurs, qui sont les traits blancs que l'on peut voir partout dans le personnage.

Une fois les os et les dépendances gérés, il est possible de commencer à créer des animations grâce à l'éditeur d'animation d'Unity. Celui-ci nécessite le paquet "2D Animation". Les animations sont très intuitives à créer, car il suffit de faire bouger les os du personnage en fonction de ce que l'on souhaite lui faire faire : si on veut qu'il lève le bras, alors on lui fait lever le bras. Il faut donc créer différentes poses que l'on agence comme on le souhaite dans une ligne du temps. Après avoir créé l'animation étape par étape, celle-ci se fluidifie automatiquement entre les différentes poses que l'on a pu donner au personnage. Les os et animations du second personnage ont pu être "copiés" aisément du premier personnage, car ils ont la même morphologie.

L'animation du fouet a nécessité le paquet "2D IK". Celui-ci, attaché au fouet, calcule automatiquement les déplacements des différents os du fouet afin d'arriver à la position souhaitée. Cela évite de devoir déplacer soi-même chaque os du fouet (il y en a 54) pour l'animer.

Enfin, l'animation du blob est particulière, puisqu'elle est la seule à être faite image par image. Il s'agit de la première animation du jeu, et il aurait été difficile d'animer le blob avec un système d'os, car il n'a pas de membres. Les images successives sont ensuite agencées dans l'éditeur d'animations afin d'aboutir au résultat souhaité.

Au départ, nous avions commencé à faire des animations image par image pour tous les personnages et toutes les animations. Cependant, cette idée a été abandonnée après quelque temps au profit des animations par système d'os. En effet, la première méthode prenait énormément de temps et n'aboutissait pas à des animations fluides. Le blob est donc le seul à avoir gardé son animation originale. Une animation de marche et une animation d'attaque ont quant à elles été abandonnées (voir annexe).

3.4.3 Intelligence Artificielle

Maëlys a réalisé le script d'intelligence artificielle du blob. Certains ennemis utilisent le même principe pour se déplacer. De plus, les ennemis ont un "point faible" au-dessus de leur tête ; si un joueur saute dessus, il tue l'ennemi en un coup et rebondit.

3.4.4 Autres

Dans ce projet, nous avons tenté de faire des échanges de tâches afin de participer à différentes parties du projet. Par exemple, Maëlys a échangé quelques animations avec Toan, contre la gestion des dégâts infligés par les ennemis. Lorsqu'un ennemi touche un joueur, le joueur perd de la vie. Il devient également invincible pendant quelques secondes, le temps de s'éloigner de l'ennemi ; cet état est symbolisé par un clignotement du lutin du joueur. De plus, des animations ont aussi été échangées avec Adam, contre la création d'un niveau : le niveau 2.

4 Déroulement du projet

4.1 Travail d'équipe

Ce projet était pour nous le premier avec une très grande importance des méthodes de travail de groupe, contrairement à ceux que nous avons pu avoir au lycée où les outils et méthodes que nous utilisions passaient au "second plan". Nous nous sommes rapidement rendus compte que sans organisation et bons outils pour mettre en commun notre travail, nous allions avoir des difficultés pour réaliser le projet.

Nos méthodes de collaboration au début du projet étaient relativement peu efficaces : en effet, nous avions commencé à travailler chacun sur des projets Unity différents, sans trop penser à comment nous allions fusionner nos résultats avant la première soutenance. Cela nous a coûté un temps non négligeable passé à mettre en commun notre travail à la main, scène par scène tout en adaptant les scripts. Nous avons cependant réussi à prendre ce moment relativement en avance, afin de pouvoir repartir sur une base viable pour la collaboration avec git et GitHub.

Une fois le dépôt git créé il fut plutôt facile de travailler chacun de notre côté, nous n'avions pour la plupart du temps pas de conflits, sauf lors de mises en commun majeures (avant les soutenances par exemple). Nous avons pu réaliser différentes fonctionnalités chacun de notre côté comme par exemple l'interface utilisateur, les niveaux, le réseau et les graphismes puis les réunir dans notre branche principale sans soucis.

Pour nous organiser, nous avons utilisé Trello pour se définir des objectifs à atteindre par périodes de 2 semaines. On nous a conseillé pour travailler la méthode "agile", avec un système où chaque mise à jour / fonctionnalité doit être validée par au moins un autre membre du groupe pour la confirmer.

Après la première soutenance, nous avons mis en avant le partage de tâches. En effet, jusqu'à la première soutenance, nous nous occupions chacun des tâches qui nous avaient été attribuées dans le cahier des charges. Cette approche avait pour avantage de fluidifier le travail, car chacun faisait son travail et n'avait pas besoin d'attendre les autres, mais elle avait pour inconvénient de ne pas permettre une bonne compréhension de l'ensemble du projet. Dans ce but, Toan a par exemple créé des animations pour le joueur ; des tâches de "mécaniques de jeu" ont aussi été assignées à Maëlys (dégâts des ennemis) et à Raphaël (barres de vie, réassignation des touches).

Avant la dernière soutenance, nous avons remarqué une nette amélioration de l'organisation, notamment concernant git : nous n'avons eu que très peu de conflits, et ceux-ci ont été gérés sans trop de problèmes grâce à l'outil de merge intelligent proposé par Unity, YAMLMerge. Celui-ci s'occupe de comprendre si un conflit en est un faux (si quelque chose dans un fichier a changé lorsque factuellement il reste pareil), et de le résoudre.

Enfin, nous avons tenté de travailler "en groupe" autant que possible malgré le distanciel. Le principe était de travailler plus souvent à plusieurs en étant en appel sur Discord afin de créer une meilleure ambiance de travail et d'être plus efficace, à base de réunions tous les Lundis (voire plus en période de pré soutenance). Bien que

nous ayons appliqué cela quelques fois, cela n'a pas été un franc succès, car nous ne travaillons pas toujours aux mêmes moments au sein du groupe. Cependant, le peu de travail de groupe que nous avons fait a permis de renforcer les liens du groupe.

4.2 Ambiance du groupe

L'ambiance dans notre groupe est bonne sur tous les points, tous les éléments du groupe s'entendent bien avec les autres, de plus chaque membre du groupe aide les autres quand l'un d'eux a besoin d'aide. De plus le groupe a une grande complémentarité sur les diverses tâches.

4.3 Difficultés rencontrées et leur résolution

4.3.1 Toan

Tout au long du développement du projet, j'ai rencontré plusieurs difficultés. C'est tout d'abord Photon qui m'a posé problème. En effet, le fait que PUN2 soit récent (sorti en 2018) explique qu'il y n'y ait pas autant de discussions à son sujet qu'on le désirerait. J'ai aussi trouvé qu'il avait quelques fois tendances à rester dans l'ombre de son prédécesseur, PUN1, ce qui peut prêter à confusion si on commence à mélanger les deux versions. A part ça, une fois avoir lu la documentation et compris le système de callbacks et le fonctionnement des composants Photon, j'ai rapidement pu développer le multijoueur de notre jeu.

Un peu plus tard dans le développement du jeu, c'est la mécanique d'échange de position qui m'a posé problème. En effet, lorsque j'ai commencé à la développer, je ne connaissais pas les *appels de procédures à distance* ou RPC et ai d'abord tenté plusieurs approches plus compliquées les unes que les autres et peu efficaces. J'ai finalement découvert cette fonctionnalité et l'ai utilisé à de nombreuses reprises pour les mécaniques, pour les variables relatives aux objets telles que le nombre de points de vie ou l'invincibilité et l'apparition de particules chez les deux joueurs.

De plus, j'ai également eu des difficultés à réaliser le régulateur d'animations. De fait, il y a de nombreux paramètres à prendre en compte pour chaque transition et le timing entre les animations est souvent très serré.

Concernant le site web, j'avais tout d'abord l'intention de le réaliser entièrement sans l'aide d'un générateur de site statique. J'ai finalement changé d'avis puisque c'était justement pour moi l'occasion d'apprendre à en utiliser un et j'ai choisi HUGO car j'en avais déjà entendu parler. La difficulté principale rencontrée en construisant le site est le fait que chaque thème de HUGO est différent au niveau structurel. Il n'y a donc pas de tutoriels universels et seuls les thèmes les plus populaires sont couverts. Ayant pris un thème peu connu, j'ai dû comprendre son fonctionnement pas à pas, fichier après fichier. Finalement, les pièces du puzzle se sont assemblées et j'ai réalisé le site en moins de 24h ce qui n'aurait pas été le cas si j'avais gardé mon idée initiale.

4.3.2 Raphaël

Je n'avais jamais utilisé Unity avant et il a été difficile pour moi de comprendre comment il fonctionnait, et je n'ai pas encore tout compris même si j'ai pu assimiler quelques concepts basiques tels que les objets, et des interactions basiques possibles avec les scripts.

Le premier problème que j'ai rencontré en travaillant sur l'interface utilisateur était les méthodes de travail et de fonctionnement d'Unity (par exemple ce qu'était un gameObject, comment les scripts s'y appliquaient, etc...). J'ai également eu du mal à décider entre faire plusieurs scènes ou avoir tous les menus dans une seule scène, ce qui a généré d'autres problèmes quand nous avions dû mélanger ma partie avec celle de Toan qui était dans un projet Unity différent. J'ai également eu des problèmes pour changer la résolution, cependant comme expliqué dans ma partie ce fut corrigé avant la dernière soutenance.

J'ai également eu du mal avec git, principalement lorsque nous avons dû régler des conflits. Au fur et à mesure du projet, nous avons pu apprendre à l'utiliser plus efficacement, et avons eu très peu de problèmes liés aux conflits de mise en commun avant la dernière soutenance. Néanmoins git est resté pour moi une grande source de maux de tête notamment avant la deuxième soutenance, car nous avons eu un nombre considérable de conflits et avons du repartir depuis la branche de Toan, qui avait les modifications les plus importantes, pour ensuite refaire notre travail à la main par dessus le sien.

Il a aussi été difficile pour moi de trouver certaines ressources sur internet pour des problèmes plus spécifiques. La plupart des idées basiques ont pu être réalisées à base de tutoriels vidéos, ou sur les forums en ligne Unity. Cependant, par exemple lorsque nous devions proposer au joueur de choisir un niveau lorsqu'il meurt, nous avons eu un problème qui rendait impossible de recharger le même. Même si nous avons par la suite reussi à régler ce problème, très peu de ressources pouvaient y répondre car aucun jeu n'est codé de la même manière

Comme expliqué dans les difficultés communes, j'ai eu du mal à gérer mon temps entre les examens, TPs et le projet. J'ai aussi eu quelques soucis pour la gestion de la barre de vie afin de la synchroniser entre 2 clients même si j'ai au final réussi à les régler avec l'aide de Toan et des forums Unity.

4.3.3 Adam

Je n'avais jamais utilisé tous les logiciels nécessaires à la réalisation de ce projet, et j'ai donc du apprendre leurs fonctionnement, leurs subtilités. Cela fut extrêmement difficile surtout pour l'utilisation de git. La manière dont git marche et toutes ses subtilités furent compliquées a appréhender comme par exemple le fonctionnement des branches. Certes j'ai perdu plus une fois de plus mon travail, cependant maintenant leur utilisation m'est complètement familière.

Les niveaux ont été un casse-tête à faire , le niveau 4 et le niveau 5 ont été difficiles car étant des niveaux de fin de jeu il a fallu les faire plus difficiles que

les niveaux précédents. Cependant, il a fallu ne pas les faire trop durs, et trouver un juste milieu qui fut assez compliqué à repérer. Les retouches des niveaux furent nombreuses et complexes. De plus Unity avait beaucoup de soucis à lancer le jeu sur mon ordinateur, il a donc été nécessaire de régler ces problèmes.

Enfin une difficulté réellement personnelle a été de respecter les délais de rendus et le fait d'avoir dû limiter le nombre de niveaux pour le bien du projet.

4.3.4 Maëlys

J'ai d'abord eu des difficultés d'adaptation aux outils comme Gimp ou Unity, ainsi que Git. Ce problème était facile à résoudre, car il existe de nombreux tutoriels et aides sur Internet. Ensuite, le temps que prennent certaines tâches, notamment d'animation, est énorme ; c'est pourquoi j'ai décidé de changer d'approche à la première soutenance, et d'utiliser davantage les outils d'animation d'Unity. Enfin, il a fallu adapter l'implémentation de l'intelligence artificielle et des animations avec le multijoueur ; avec l'aide de Toan et de quelques scripts Photon, nous avons finalement réussi à rendre les ennemis visibles et synchronisés pour les deux joueurs.

De plus, j'ai eu des problèmes techniques au niveau des logiciels ; j'ai dû télécharger et apprendre à utiliser Visual Studio car Rider ne fonctionnait pas, de mon côté, avec Unity. Finalement, je me suis rendue compte qu'il me manquait la librairie supportant Rider. Rider me pose toujours des problèmes (pas d'autocomplétion) mais il fonctionne. Au fil des soutenances, la compréhension des logiciels est devenue un problème moindre. En effet, j'ai dû apprendre à créer et utiliser les systèmes d'os et de dépendances pour les animations sur Unity, mais après quelques heures de pratique, cela était bien plus simple. Ensuite, il s'agit surtout de répéter les mêmes procédures pour créer le second joueur, le second ennemi, etc. Le plus compliqué a été de m'adapter aux outils de Unity : j'ai dû modifier les graphismes du joueur de nombreuses fois, car je ne comprenais pas bien comment implémenter les animations. Grâce à des vidéos YouTube, j'ai finalement réussi à comprendre le fonctionnement du système d'animations. Un autre problème était la gestion de mon espace de travail : ayant un ordinateur portable et un fixe, je dois m'organiser afin de toujours travailler sur un seul des deux à la fois. Pour résoudre ce problème, j'ai simplement décidé de travailler exclusivement sur mon ordinateur portable. Ensuite, j'ai rencontré des problèmes d'ordre matériel, avec un disque dur dysfonctionnel à cause duquel je n'ai pas eu accès à mon travail pendant plusieurs jours. Récupérer mes données a été long, mais heureusement, rien n'a été perdu.

Jusqu'à cette dernière soutenance, mon principal problème a été la motivation et la régularité. En effet, le semestre se termine, nous avons passé nos partiels et la dernière ligne droite est difficile. C'est grâce à l'ambiance de mon groupe, plutôt bonne, que j'ai continué d'avancer jusqu'au bout.

4.3.5 Difficultés communes

Mise en commun du travail et Git

Une des premières erreurs que nous avons faites était de commencer avec plusieurs projets Unity différents. Il a été fastidieux de "mélanger" nos différents travaux à la main, comme par exemple pour appliquer l'interface utilisateur de Raphaël à la partie jeu en réseau de Toan. Une fois ce travail effectué, nous sommes passé à git, qui a bien fonctionné hormis quelques conflits lorsque nous avons tenté de fusionner 2 branches : certains fichiers "prefabs" d'Unity avaient des changements de référence alors qu'une des branches ne leur avait pas apporté de modification, ce qui a créé de nombreux conflits de fusion. Hormis ce problème la majorité de la gestion des branches s'est plutôt bien passée, cependant nous n'avons pas trouvé de vraie solution pour les prefabs qui se changeaient "tout seuls" et nous devons donc être attentifs aux fichiers qui changent lors des futurs *commits*.

Jusqu'à la seconde soutenance, notre organisation se portait mieux qu'avant la première, et nous avions plutôt bien réussi à travailler sur des branches différentes et à mettre en commun nos résultats toutes les 2 semaines. Cependant, lors du merge final avant la soutenance, nous avions souffert de nombreux "merge conflicts", que nous n'avions pas réussi à résoudre, principalement sur des fichiers scènes d'Unity. Nous avons essayé le YAML Merge tool proposé par Unity, qui avait marché avant la première soutenance mais qui n'avait pas fonctionné ici, ainsi que de les résoudre à la main sur l'interface web de Github. Nous avons fini par opter pour refaire les travaux les moins longs sur la branche qui contenait les modifications les plus importantes, ce qui a plutôt bien marché, même si beaucoup plus fastidieux que si le merge s'était bien passé. Nous avons décidé d'être plus vigilants quant aux modifications de scènes dans le futur : les scènes sont les principales sources de conflits lors des merges.

Enfin, jusqu'à cette soutenance, nous avons appris de nos erreurs et les conflits ont été bien moins nombreux. Bien sûr, il est arrivé que des incompréhensions ou des malentendus mènent à des conflits mineurs, mais il ne s'agissait plus de centaines de conflits pour un seul merge. De ce fait, moins de tensions liées à Git ont précédé cette dernière soutenance.

Compréhension des parties des autres membres du groupe

Avant même le début du projet, nous avons séparé les tâches pour chaque membre du groupe. Si cela nous a permis d'avancer plus vite sur nos parties respectives, nous avons également eu du mal à comprendre les parties de nos camarades, et pour certains nous en avons encore (notamment concernant la partie réseau, qui est sans doute la plus complexe de ce que nous avons produit actuellement). C'est en partie pour cela que nous avons mis en place des échanges de tâches.

Distanciel

Nous nous en doutions, mais le distanciel n'a pas facilité le travail en groupe. Même si nous avons réussi à rester relativement organisés avec des réunions régulières, il manquait cette atmosphère de travail en équipe en face à face. De plus, même si nous avions pu nous réunir à l'école nous n'aurions pas pu tous travailler étant donné que nous ne sommes pas tous équipés d'un PC portable assez puissant pour Unity, et que Unity n'est à priori pas installé sur les machines de l'école.

Travail du projet en plus du travail pour les examens

Les dernières soutenances faisant suite à des périodes d'examens, il a été difficile pour nous de consacrer du temps au projet car nous devions réviser et faire nos TP, qui augmentent en difficulté. Nous avions tout de même réussi à travailler sur le projet avant ces périodes, mais nous n'avons pas pu bénéficier autant des semaines précédant la seconde soutenance que lors de la première. La semaine consacrée au projet, avant la dernière soutenance, a été bénéfique. Il a également été compliqué de commencer le rapport de soutenance en avance, la semaine de "vacances" ayant été dédiée principalement aux derniers ajouts sur le projet et à la mise en commun du travail. La création des pièces à rendre (Jaquette pour la clé USB, manuel d'utilisation) a également été tardive.

Relâchement avant la dernière soutenance

La fin des partiels étant une période marquante, nous avons pour certains sans forcément le vouloir fait preuve d'un relâchement au début de la semaine allouée à la finalisation du projet, ce qui a été une source de stress. Cependant ce stress nous a aussi permit de nous remettre à travailler et de compléter nos objectifs fixés.

5 Conclusion

Ainsi, nous arrivons au terme de ce projet, qui aura duré 6 mois. Après maintes péripéties, nous sommes enfin capables de présenter un jeu terminé, avec un début et une fin, en passant par différents niveaux. Nous avons beaucoup appris lors de ce projet, notamment l'utilisation des outils comme Unity, mais aussi le travail de groupe et quand il le fallait, l'autonomie.

Cette dernière version comporte une grande majorité des éléments prévus au départ, certains ayant été modifiés pour être plus réalistes. En effet, elle est composée d'une interface graphique sobre mais élégante représentant différents menus. Elle est également munie d'une musique principale et de musiques de niveaux et permet aux joueurs d'effectuer toutes les actions nécessaires au passage des niveaux. Le joueur a le choix entre plusieurs systèmes de jumelage pour rejoindre un coéquipier dans une partie et plusieurs niveaux. Une fois entré dans le niveau, le joueur peut constater le travail fourni au niveau des graphismes 2D faits maison mais aussi de la réflexion nécessaire à la conception du niveau et des mécaniques de jeu. Le jeu possède deux personnages et assez d'ennemis pour les mettre à l'épreuve. Il nous semble important de préciser que la partie créative de notre jeu est entièrement conçue par nous-même, à l'exception des fonds des niveaux.

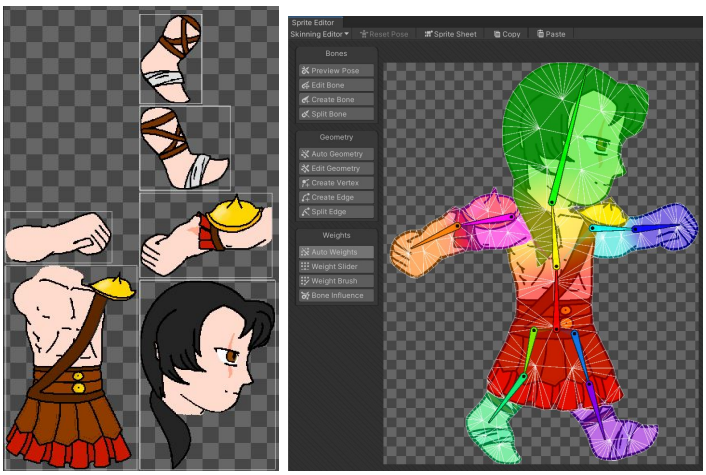
Certaines des difficultés que nous avons rencontrées tout au long du projet on persisté jusqu'à la fin : le problème du distanciel, le problème des délais ainsi que les périodes d'examens. Cependant, en conservant le rythme des réunions de groupe, nous avons réussi à forcer l'implication de tous les membres, et donc l'avancement du projet.

Globalement, le jeu auquel nous sommes arrivés correspond à nos attentes du cahier des charges. Nous espérons que les prochains à tester le jeu s'amuseront autant que nous lors de sa conception !

6 Annexes

6.1 Captures d'écran

6.2 Interfaces

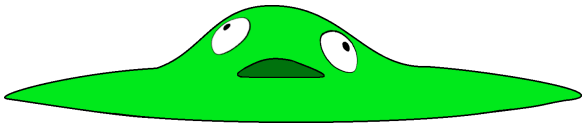


Découpage du personnage dans l'éditeur de lutin d'Unity / Os et poids d'un personnage

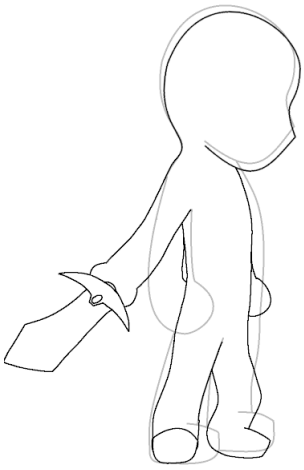
6.3 Art conceptuel du jeu

Art conceptuel du premier personnage

Art conceptuel du second personnage



Le blob vert, non utilisé



Une tentative d'animation d'attaque, non utilisée