8INF960 – Principes de conception et développement de jeux vidéo

# Rapport final


| | |
|---|---|
| Titre du jeu | Dodominance ? |
| Promoteurs | (25%) Clément de Gaillande : DEGC30049809<br>(25%) Stéphane Lin : LINS02029809<br>(25%) Basile Nguyen : NGUB28099807<br>(25%) Thibault Sellin : SELT28129806 |
| Session | Automne 2020 |
| Date de Livraison | 15 Décembre 2020 |
| Genre | FPS - RogueLike |
| Contexte | Futuriste |
| Mode | Simple joueur |
| Public cible | Public averti (+12 ans) |
| Plateforme | PC |
| Contrôle | Clavier/Souris |

**I - Bref Rappel du jeu** **3**
a - Histoire 3
b - Concept 3
c - Mécaniques 3
**II - L'intelligence artificielle des ennemis** **3**
a - Mécaniques de combat 3
b - Animations 4
**III - Interface graphique** **4**
**IV - Principales parties du jeu** **4**
**V - Tests manuels** **4**
**VI - Choix de conception** **5**
**VII - Choix de développement** **5**
**VIII - Planification du projet** **5**
**IX - Post-mortem** **5**

# I - Bref Rappel du jeu

## a - Histoire

Le jeu se déroule dans un avenir dystopique. L'humanité est en guerre. Elle doit affronter une coalition d'animaux qu'on pensait auparavant disparus. Composée principalement de dodos, cette coalition rassemble également différents représentants d'espèces d'animaux dangereux, pour lesquels le concept d'extinction n'est pas étranger. Pandas, dinosaures, mammouths et autres ont rejoint la coalition des dodos pour mener une guerre contre l'humanité. A la tête de la coalition se trouve une créature insolite qui semble avoir bravé toutes les étapes de l'évolution. Mi-homme, mi-oiseau, cet être surpuissant est bien déterminé à faire goûter à l'humanité le goût amer de l'extinction. Le protagoniste est un simple soldat sans nom, qui s'est engagé pour protéger son espèce. Des suites de nombreux efforts et opérations de la part de l'humanité, notre protagoniste a réussi à infiltrer le complexe qui sert de base à la coalition. Il faut désormais se frayer un chemin dans la base ennemie pour démanteler les forces adverses. Pour cela, on devra se frayer un chemin dans les nombreux étages regorgeant d'ennemis, et affronter les différents lieutenants de la coalition. Pour notre protagoniste, cette épreuve s'annonce difficile et il faudra surpasser ses limites et défier les rapports de force de la Nature pour y parvenir.

## b - Concept

Dodominance? est un Roguelike de tir à la première personne. Au travers de sa partie, le joueur doit évoluer dans plusieurs environnements labyrinthiques en affrontant les différents ennemis sur son chemin. Entre chaque niveau le joueur devra se battre contre des boss de plus en plus fort jusqu'au dernier étage pour affronter le boss final. Plusieurs personnages pourront aider le joueur pour accompagner son aventure.

## c - Mécaniques

Le jeu possède différentes mécaniques qui en font un Roguelike unique, au-delà du concept peu exploré de Roguelike en jeu de tir à la première personne.
L'une des principales fonctionnalités est le facteur aléatoire de notre jeu. En effet, les niveaux sont générés de façon procédurale. C'est un programme qui décide de l'architecture des niveaux de notre jeu, en assemblant de façon cohérente des salles préfabriquées. Cela permet de rendre chaque nouvelle partie de notre jeu unique.
Le personnage du joueur est également fait pour s'améliorer tout au long de la partie. Un système de récompense est établi pour chaque ennemi vaincu. Après s'être débarrassé d'un ennemi, une récompense monétaire permettant d'acheter de l'équipement est octroyée, au même titre qu'une bonus sur différents stats du joueur, par exemple la vitesse de rechargement ou encore les dégâts de base infligés par une attaque. Cela permet au joueur de sentir une montée en puissance et de donner un sens immédiat à ses actions.

# II - L’intelligence artificielle des ennemis

## a - Mécaniques de combat

Dans le jeu, 6 ennemis différents ont été développés : les poulets, les pandas roux, les pandas, le boss panda, le boss mammouth et le marchand.
Les poulets, les pandas roux et les pandas ont été définis dans le Game Design Document comme étant des ennemis basiques, tourelles et cuirassés. Le système de combat des IA repose sur deux périmètres de distance, un rayon de détection provoquant la poursuite du joueur et un rayon d’attaque qui déclenche une attaque. En dehors de ces deux périmètres, l’AI restera immobile avec une animation d’attente (Idle).
Les IA ayant besoin de connaître continuellement la position du joueur pour se déplacer utilisent un navMeshAgent. Le jeu possédant une génération procédurale, le navMesh doit être généré de manière dynamique après la création du niveau.

Poulet, Panda et Boss Panda:
Ces 3 ennemis possèdent une mécanique de combat très similaire, le joueur rentre dans le périmètre de détection, l’IA poursuit le joueur et si le joueur rentre dans le périmètre d’attaque, l’ennemi effectue une attaque au corps-à-corps. Cependant certaines différences sont notables, le panda qui appartient a le rôle de cuirassé (ennemi lent mais avec une santé importante, cf.GDD) possède une attaque relativement lente, le joueur ne prendra des dommages que s’il reste dans la zone d’attaque pendant plus de la moitié du temps de l’attaque.
Le boss panda ne possède pas d’attaque au corps-à-corps, il utilise deux attaques, la première est une zone autour de lui pour infliger des dommages continus. Les dégâts sont infligés selon un timer tant que le joueur est dans la zone. La seconde attaque est une projection de trois boules de feu explosives, le joueur reçoit des dommages s’il se trouve dans le souffle de l’explosion. Un gameobject vide se trouve devant la mâchoire du boss, il sert de point de départ pour que les boules de feu aient l’air de sortir de sa bouche. Les boules de feu sont des sphères avec un rigidbody auxquelles on applique une force horizontale et une force verticale pour donner un effet de cloche

Panda roux :
Cet IA est un ennemi tourelle, il est immobile mais peut attaquer à distance. Étant immobile, il n’a pas besoin d’un navMeshAgent. Pour attaquer, comme le boss panda, il envoie des balles explosives. Ces projectiles possèdent un rigidbody et explosent à la première collision avec le collider (excepté avec d’autres ennemis)

Boss mammouth :
Le boss mammouth est le second boss du jeu, sa santé est plus légèrement plus importante que le premier boss. La mécanique de ce boss repose sur des dash. Il a 4 phases d’action, dasher, freiner, viser le joueur, charger. Il répète ses actions indéfiniment. Pour dasher, on applique une force horizontale sur le rigidbody pendant un certain laps de temps, lorsqu’on souhaite le freiner, on multiplie sa vélocité par 0.95 à chaque frame. S’il percute le joueur pendant son dash, le joueur prend 35 dommages.

Le marchand :

Cette IA est particulière car elle est pacifique tant qu'on ne l'attaque pas. Une fois qu'on l'a attaqué, le script qui permet de lui acheter des items est désactivé et un script semblable à celui du poulet est activé. Étant donné qu'il a été réalisé en dernier, le marchand possède un une mécanique technique légèrement plus poussée que le reste des ennemis. Le marchand attaque avec le poing droit, un collider est présent sur le poing et le joueur prend des dommages quand il rentre en collision avec ce collider.

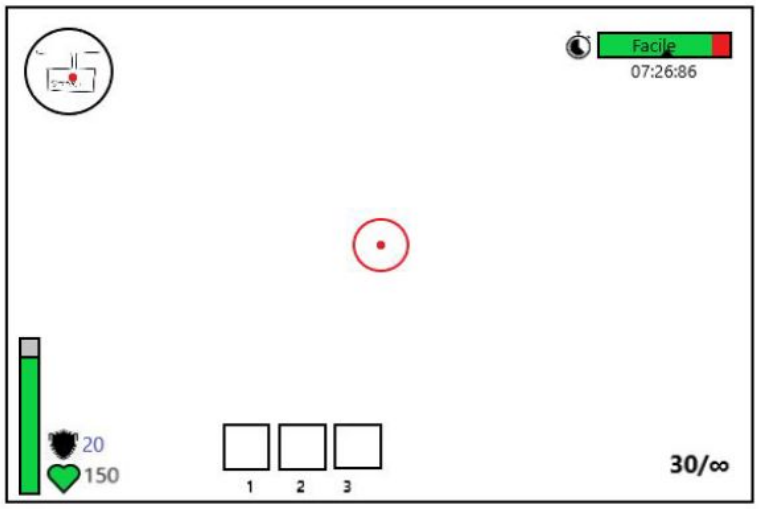
## b - Animations

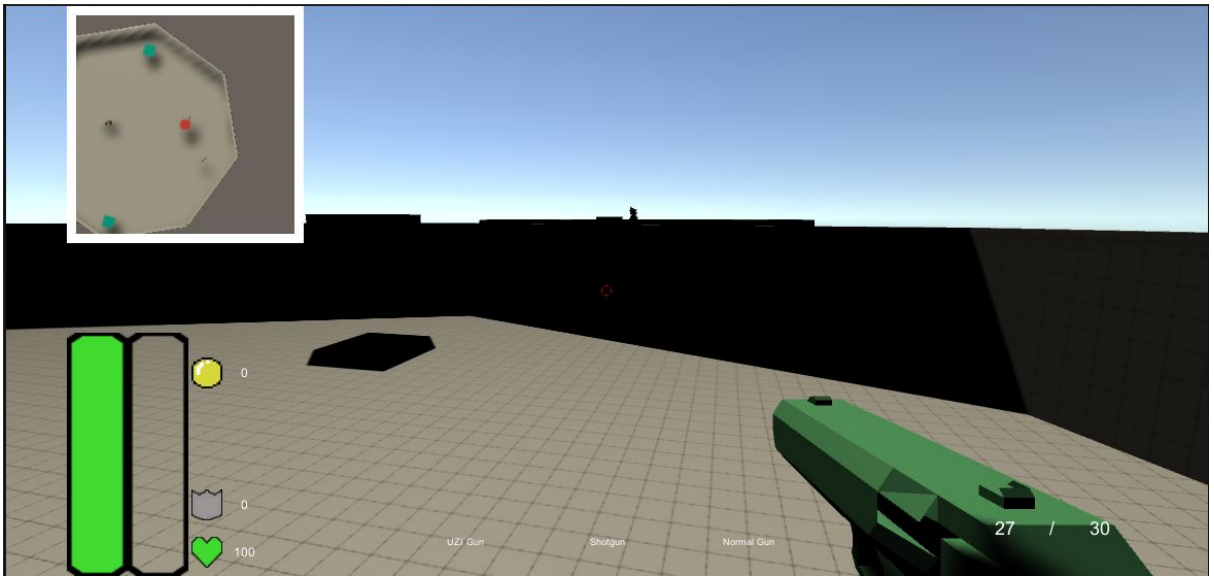
Pour gérer les animations, on utilise un AnimatorController, les transitions sont déclenchées par des booléens gérés dans le script. Pour tous les animaux, nous avons utilisé des animations et des modèles qui étaient déjà contenus dans des asset, cependant pour le marchand et clara, nous avons utilisé des asset d'animations indépendantes des asset du modèles.

Le vortex de la zone du boss panda, la fumée au pied du boss mammouth et l'explosion des projectiles proviennent tous du même asset qui ont été modifiés en augmentant le nombre de particules, changeant la couleur ou augmentant la taille. Pour ajouter un effet visuel aux objets avec des mouvement rapides, nous avons ajouté un composant de trail renderer pour laisser une traînée derrière l'objet en mouvement.

# III - Interface graphique

L'interface graphique du jeu a été réalisée pour correspondre au maximum au design d'origine du Game Design Document.




Le HUD ainsi que les différents menus du jeu ont été réalisés avec des canvas. Presque toutes les fonctionnalités ont été implémentées. Tous les éléments de l'interface sont initialisés par le script central du jeu : GameManager. Par la suite, c'est le script du joueur qui gère les éléments au travers de la partie. Les barres de vie et d'armures sont des sliders. En bas, il s'agit de la liste des armes ou items possédés par le joueur, chaque item possède un nom unique qui est affiché dans cette zone.  A droite, les munitions sont mises à jour avec le script du slotsController / du script attaché à l'équipement utilisé qui affiche le nombre de munitions en fonction de l'arme, le nombre d'exemplaires d'un item (ex : nombre de potions). La minicarte est le rendu d'une caméra qui se situe au-dessus du joueur.

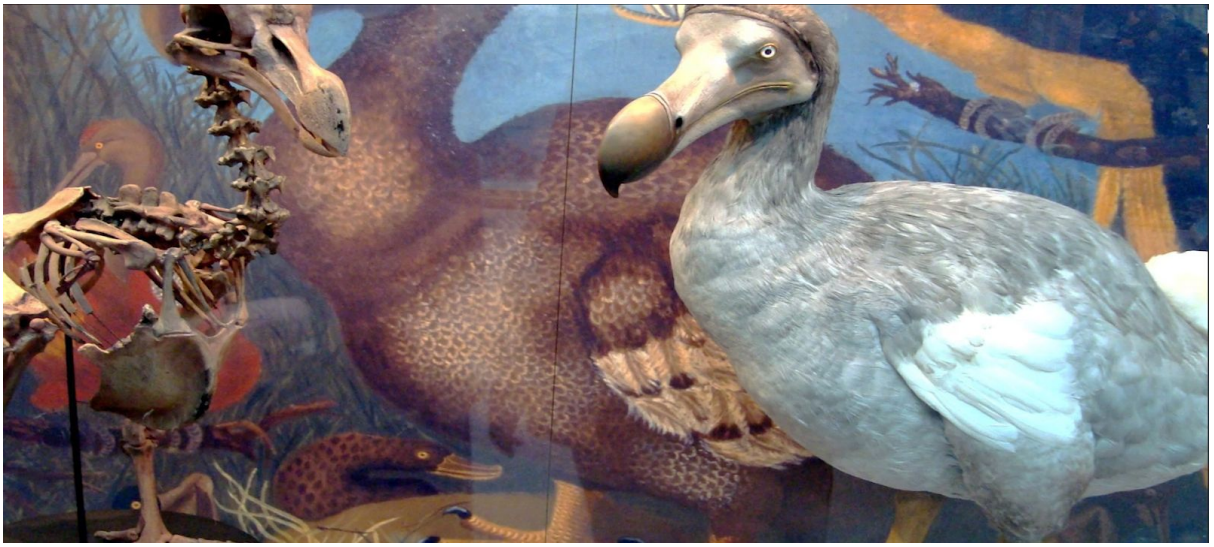
En plus de l'interface disponible en plein jeu, différents menus ont également été créés afin de naviguer dans la structure du jeu. Un menu principal, un menu de pause, des crédits, un menu de game over, un menu de chargement du monde ont été implémentés. Ces menus sont créés à partir de l'outil canvas et utilisent des boutons / des touches pour les activer et désactiver. Le script MenuInteract.cs, présent sur le GeneralCanvas, permet d'associer des appels de fonction à l'appui des boutons depuis l'inspecteur. D'autres, comme le bouton pour quitter le shop, ont été alloués dynamiquement.

Voici des captures d'écran du menu principal, du menu de pause et du menu de game over.



FIN DU JEU.
Réessayer.
Quitter.
242
0
0
24 / 30

# IV - Principales parties du jeu



Le jeu démarre sur la cinématique d'introduction, puis le menu principal. Il peut quitter à tout moment en choisissant l'option adéquate, visionner les crédits, ou démarrer sa partie en cliquant sur "jouer".

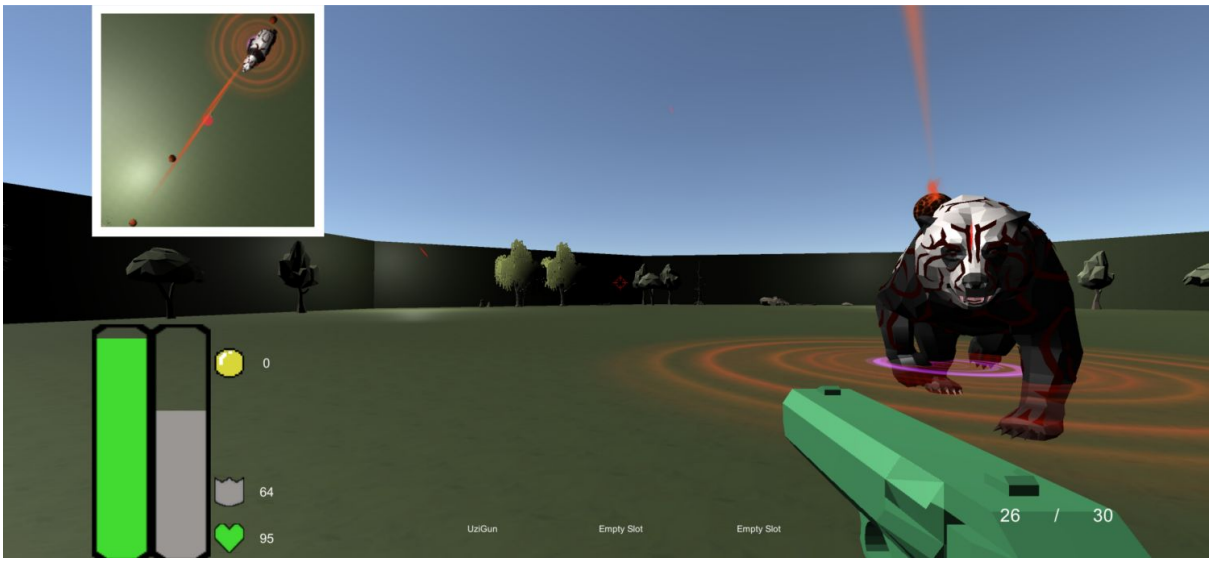

Le joueur débute dans une petite pièce, où se trouvent les outils dont il a besoin pour démarrer sa partie. Différents équipements lui sont présentés : pistolet, Uzi et fusil à pompe. Il peut récupérer l'équipement dont il a besoin, en sachant qu'il possède seulement 3 slots d'équipement.
Un PNJ du nom de Clara se trouve dans cette zone. Il s'agit du personnage chargé du tutoriel. Le joueur peut lui parler en appuyant sur E, ce qui ouvrira un dialogue avec elle dans lequel elle expliquera brièvement l'objectif et les contrôles du jeu à qui veut l'entendre.

Le joueur peut ensuite interagir avec le cube de couleur cyan qui se trouve dans le coin de la pièce pour se téléporter dans le premier étage.


Les étages, qui sont les principales zones que le joueur traverse durant son périple, sont des niveaux générés aléatoirement. Il s'agit d'un agencement logique de salles et de couloirs les reliant. Les différents prefabs utilisées pour la création de l'étage sont :

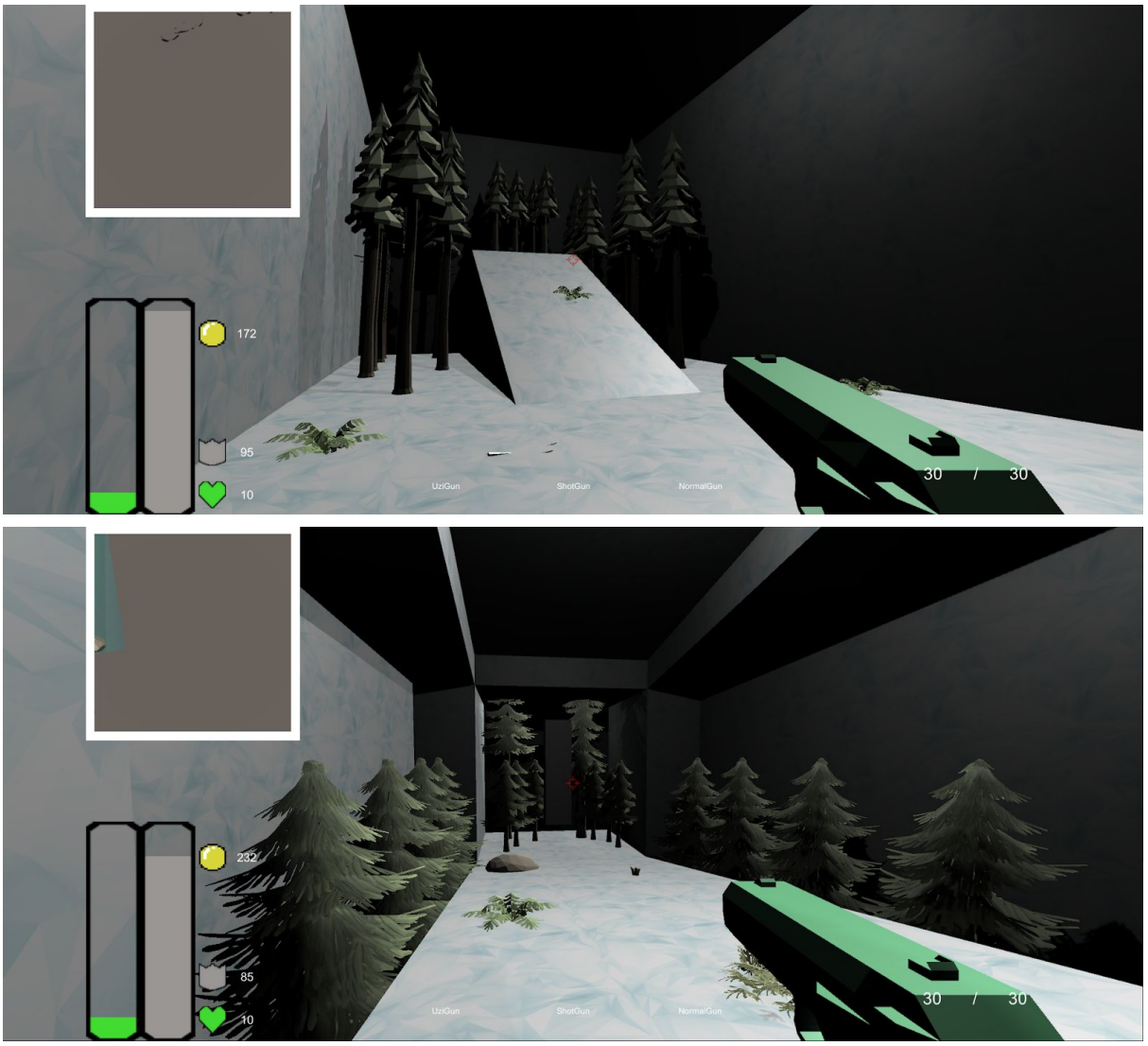
- la salle de départ, qui est la salle dans laquelle est transporté le joueur après s'être téléporté.
- la salle finale, où se trouve le téléporteur que le joueur peut emprunter pour quitter l'étage.
- des salles de monstres
- des couloirs en L
- des couloirs en T.


Les agencements de salles, reliés par les couloirs, sont générés de façon procédurale, rendant chaque étage unique mais cohérent.

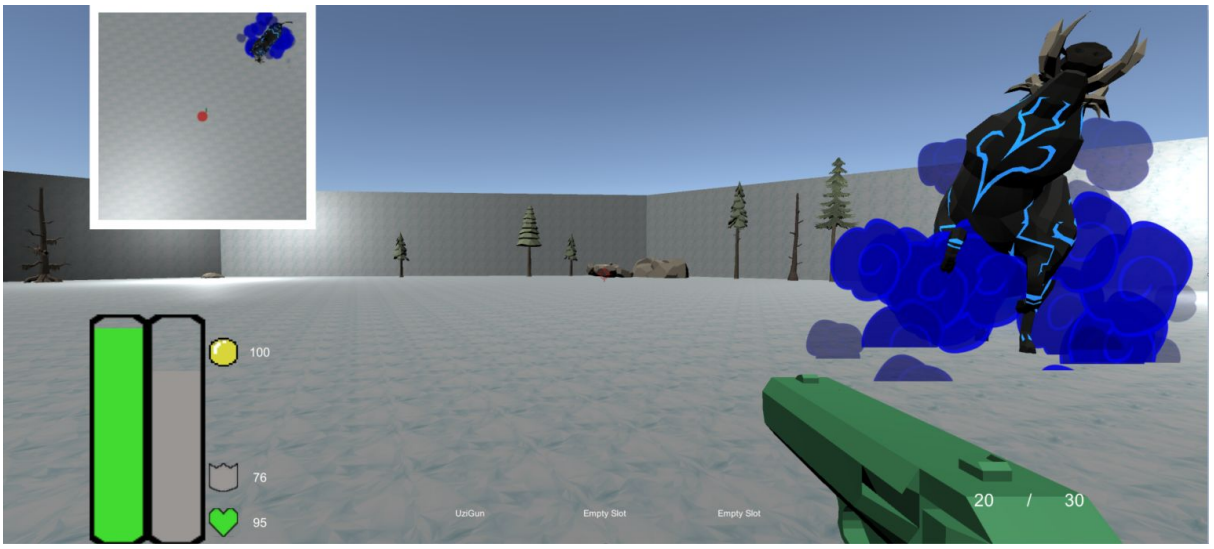



Après avoir utilisé le téléporteur de la salle finale de l'étage pour le quitter, le joueur est transporté dans une arène où il devra affronter un boss. Ce n'est qu'après avoir remporté ce combat ardu qu'un téléporteur lui permettra de quitter l'arène et de continuer le jeu.


Le téléporteur de la première arène de boss envoie le joueur dans la salle du marchand. Très similaire à la salle de départ, il s’agit d’une salle où l’on peut converser avec le PNJ du marchand. Le joueur peut alors choisir de commercer avec lui pour obtenir des équipements, l’ignorer ou bien l’attaquer.




Une fois le téléporteur de la salle du marchand emprunté, le joueur doit traverser un deuxième étage, lui aussi généré de façon procédurale. Tout comme le premier étage, il se termine sur une salle contenant un téléporteur, qui transporte le joueur dans une nouvelle arène, où se trouve le 2ème boss du jeu. Une fois celui-ci éliminé, la partie prend fin.


100
76
95
UziGun
Empty Slot
Empty Slot
20 / 30

# V - Les classes

Notre jeu possède un total de 39 classes. La plupart héritent de MonoBehaviour, mais quelques sont des classes classiques dans lesquelles on a ajouté le champ [System.Serializable] qui permet de les utiliser et de les définir dans l'inspecteur.

La classe principale du jeu est le GameManager. Elle est rattachée à un empty game object dans la scène possédant le même nom. Cette classe est chargée de garder une référence à tous les objets de la scène afin de remplir les dépendances des classes qui sont initialisées dynamiquement. Elle regroupe également les méthodes qui permettent d'organiser le jeu, comme l'initialisation de la création des niveaux, l'instantiation des nouveaux ennemis / gear / npc … présents dans la scène lors du lancement d'une partie, la logique permettant de recommencer une partie (supprimer toutes les entités qui peuvent être ajoutées dynamiquement puis peupler à nouveau le jeu ...). C'est la classe centrale du projet. Elle permet également de gérer certaines input, comme les interactions avec l'environnement du joueur avec E (qui comprend le loot de Gear, l'activation des téléporteurs, l'activation des dialogues avec un NPC …) ou encore le jet du gear actif avec G.

La classe LevelBuilder nous permet de créer un niveau de manière procédurale. On l'attache également à un empty gameObject par niveau (dans notre scène Level1 et Level2). Cette classe utilise la classe Room et ses dérivées, associées aux préfabs des différentes salles, pour créer un niveau à partir d'une certaine position. Elle vérifie qu'il n'y ait pas de collisions entre les salles, puis instancie les téléporteurs permettant de sortir du niveau ainsi que les ennemis à des spawn point définis dans les prefabs des salles. Une fois la génération d'un niveau terminé, elle appelle une méthode du GameManager qui va pouvoir lancer la génération du niveau suivant, puis l'initialisation des différentes entités présentes.

La classe AudioManager est attachée au GameObject vide AudioManager. Elle comporte une liste de Sound (une classe possédant le champ [System.Serializable] qui nous permet d'ajouter des AudioClip à notre jeu). L'audio Manager est utilisé pour jouer tous les sons du jeu (lors du tir d'une arme, de la mort d'un ennemi, du rechargement d'une arme etc…).

La classe CanvasManager, qui est attachée au GameObject vide GeneralCanvas, permet de gérer l'affichage des différents menus ainsi que la variable GameIsPaused bloquant les interactions avec le jeu lorsqu'un menu est ouvert. Cette classe permet de gérer les différents CanvasController. La classe CanvasController possède un champ canvasType, qui peut prendre les différentes valeurs listées dans l'enum définie dans le CanvasManager, ainsi qu'un booléen isShown. Cette classe est attachée à chaque menu présent dans l'objet GeneralCanvas, et permet au CanvasManager de savoir à quel menu il a à faire et dans quel état il se trouve (caché/ visible). Le seul menu présent dans GeneralCanvas n'ayant pas de script CanvasController est l'objet Dialogue box, mais nous y reviendront plus tard. Lorsque le jeu est lancé, on peut accéder au menu de pause en appuyant sur P.

La classe MenuInteract est attachée aux différents CanvasManager. Il permet de définir des fonctions qui seront utilisées pour associer des comportements aux boutons des menus depuis l'inspecteur.

Le Canvas du shop est ouvert après un dialogue avec le NPC (dont la classe sera décrite plus tard) ShopKeeper. Le ShopKeeper possède un script ShopController qui permet de gérer le son inventaire. Chaque ShopKeeper instancié possède un inventaire différent. Lorsqu'un ShopKeeper est instancié, il lui est associé 4 ShopItem aléatoires. Les ShopItem sont choisis aléatoirement parmi les différents prefabs de Gear, ils possèdent tous un prix, un nom et un bouton d'achat. Lorsqu'un ShopItem est acheté, il est détruit dans le Shop, et lorsque le joueur quitte le Shop les différents ShopItem achetés sont instanciés autour du marchand.

Le HUDCanvas est un peu particulier. Il va permettre d'afficher tous les éléments permettant de donner des informations au joueur pendant qu'il joue (barre de vie / d'armure, munitions, slots etc…). Ces différents éléments sont gérés par plusieurs classes : les HUDText (un HUDText par texte dans le HUD), les HUDBar (une HUDBar par barre de vie / d'armure) et le MiniMapController (permettant à la caméra gérant la minimap de suivre le joueur).

Enfin, le menu DialogueBox est géré par le DialogManager, qui est associé au gameObject vide DialogManager dans la scène. Le DialogManager va gérer l'affichage et l'interaction avec différentes DialogData. On peut passer à la DialogData suivante en appuyant sur entrée, et on peut naviguer dans les choix en utilisant les flèches directionnelles Haut et Bas. Les DialogData sont des listes de string associées aux différents NPC avec lesquels le joueur peut interagir.

La classe NPC est associée au différents PNJ du jeu (Clara, qui est disposée directement dans la scène, et les différents ShopKeeper pouvant être instanciés depuis un prefab). Cette classe utilise le DialogManager pour lancer un dialogue lorsque le joueur s'approche du PNJ associé et utilise la touche d'interaction.

Contrairement à beaucoup de nos camarades, notre joueur est géré par un CharacterController plutôt qu'un RigidBody. Pour ça, nous avons une classe PlayerController attachée à l'objet FirstPersonPlayer dans la scène. Cette classe nous permet d'utiliser le composant CharacterController avec ZQSD et de le faire sauter avec la barre espace. Elle nous permet également de stocker les stats du joueur, comme sa vie max, sa vie actuelle, sa monnaie actuelle, son armure etc. Elle nous permet également de vérifier si le joueur est au sol. La vision du joueur, elle, est gérée par la classe PlayerView attachée à la caméra PlayerCamera (qui est un fils de FirstPersonPlayer) dans la scène, qui est la caméra principale. Cette classe nous permet de faire pivoter le joueur lorsque la souris se déplace de gauche à droite et de faire pivoter la caméra verticalement lorsque la souris se déplace de haut en bas.

L'objet PlayerCamera possède également un fils GearSlots, qui lui même possède trois fils GearSlot1/2/3. GearSlots a le script SlotsController attaché, qui permet de gérer les trois GearSlot qui ont un script Slot attaché. Slot est une classe avec le champ [System.Serializable]. Cela nous permet d'associer un Gear à ce slot, et de changer dans le HUD le nom du slot pour qu'il corresponde au Gear associé. Il possède également un booléen slotEmpty permettant au SlotsController de déterminer si un Gear est associé à ce Slot. Le SlotsController possède donc une liste de Slot faisant référence aux différents GearSlot, ainsi qu'un Slot activeSlot déterminant quel Slot est actif pour le joueur. Le slot actif peut être changé en faisant glisser la molette. La classe SlotsController définit les

méthodes permettant de looter une arme ou de jeter l'arme associée au slot actif (s'il y en a une). Enfin elle possède une liste d'Arsenal, qui représentent les boost des Weapon. La classe Arsenal est également une classe avec le champ [System.Serializable]. Elle permet d'influer sur le fireRate, le temps de recharge et les dégâts d'une Weapon. Les boost sont appliqués lorsqu'on loot un nouveau boost et lorsqu'on change de slot actif, ce qui signifie qu'ils boostent les Weapons dans la globalité et non juste la Weapon active au moment où on loot un Arsenal. Lorsque le joueur meurt / relance une partie, la liste des Arsenal est réinitialisée.

Nous allons maintenant parler de GearController, Weapon et Consumable. Tous les objets utilisables par le joueur sont regroupés sous la classe générique GearController. Elle définit une énumération GearType qui permet de distinguer si un Gear est une Weapon ou un Consumable, ainsi qu'une méthode permettant de le reset. La classe Consumable sera associée aux prefabs des consommables du jeu. Elle définit l'effet du consommable, qui sera appliqué lors du clic gauche, ainsi que le nombre de consommable restants. Lorsque ce nombre tombe à zéro, le consommable se détruit et disparaît des slots du joueur. La classe Weapon permet de définir les armes du jeu. Elle possède des champs (qui seront set dans la prefab) permettant de déterminer les dégâts de l'arme, sa portée, son fireRate, ses munitions restantes dans le chargeur, la capacité du chargeur, son temps de recharge, si l'arme est automatique ou non (est-ce qu'en restant appuyé sur le tir l'arme tir en rafale ou non). La classe possède une méthode de tir qui utilise un raycast. Le comportement va changer en fonction de la cible atteinte par le raycast, ce qui va permettre notamment d'instancier un TextAnimator affichant les dégâts subis si la cible est un ennemi. Le prefab TextAnimator possède un script FloatingText attaché qui permet de jouer l'animation de dégâts reçus.

La classe SceneSwitch n'est pas utilisée, elle devait nous permettre de changer de scène, mais tout notre jeu se passe sur la même scène.

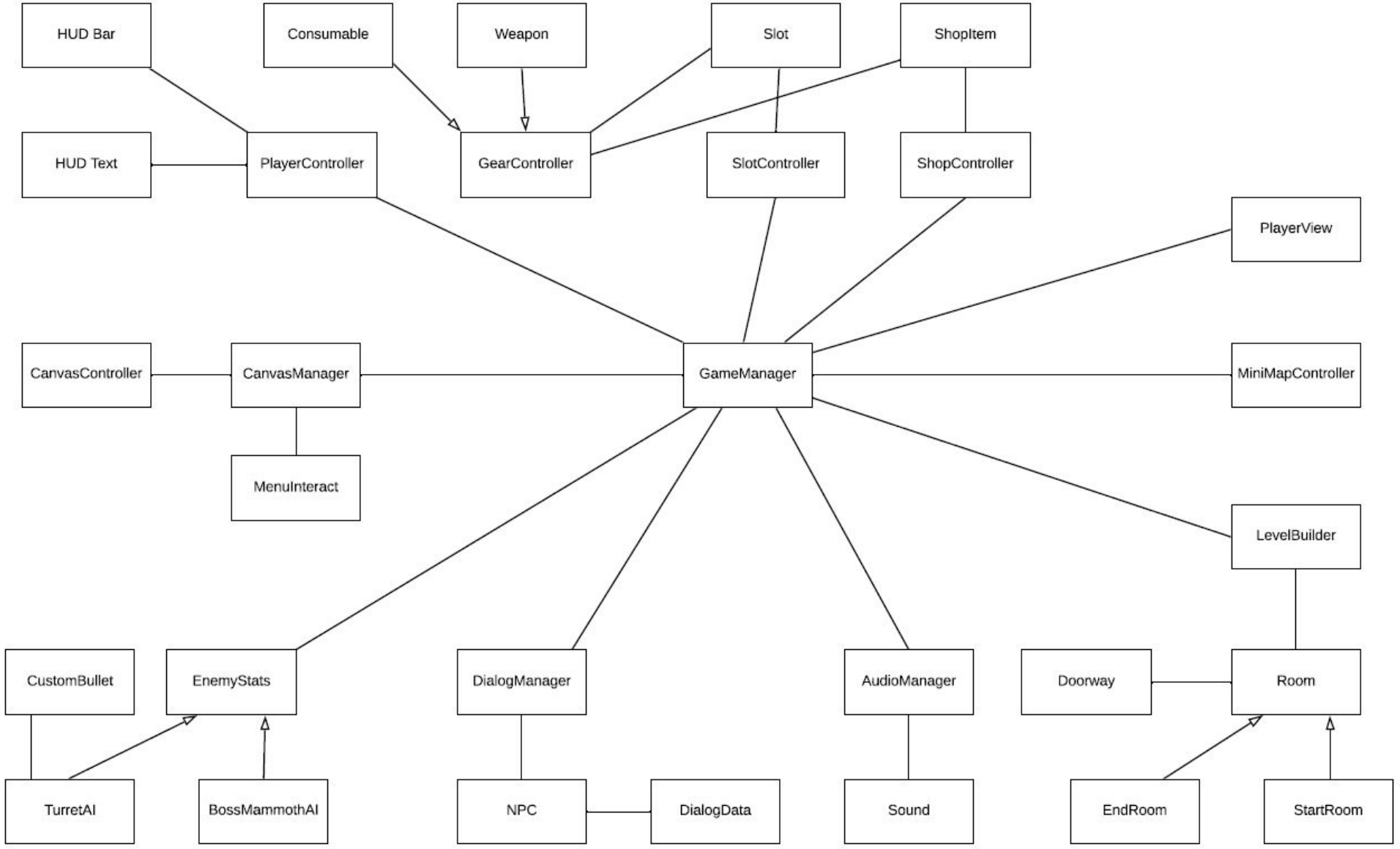
La classe RotateLoadingLogo sert à donner une rotation à l'image de cible dans l'écran de chargement.

Enfin, la classe générique EnemyStats permet de définir le comportement des ennemis. Elle définit une enum EnemyType permettant au différents scripts d'identifier quel type d'ennemi est instancié. La classe EnemyStats définit un ensemble de variables qui seront changées par les différentes classes AI héritant d'EnemyStats. Elle contient également les méthodes permettant aux ennemis d'infliger des dégâts au joueur et de provoquer la mort de l'ennemi quand celui-ci est à court de points de vie. Il existe 6 classes d'AI, une pour chaque prefab d'ennemi (Chicken, Red Panda, Panda, Boss Panda, Boss Mammoth, ShopKeeper). Toutes les AI sont mobiles, sauf l'AI du Red Panda qui est statique. Les AI mobiles vont toujours chercher à se rapprocher du joueur. Les AI Red Panda et Boss Panda possèdent des attaques à distance et lancent donc des projectiles définis par la classe CustomBullet. Enfin, la classe HitboxShopKeeper permet de détecter si le coup de poing du marchand, qui ne peut être lancé que si le joueur a tiré au préalable sur le marchand, entre en contact avec les joueurs, auquel cas les dommages du marchand sont appliqués au joueur.

# VI - Choix de conception

Lors de la conception du jeu, nous avons fait beaucoup de choix pour créer le meilleur environnement pour le joueur et le maintenir dans le flow. Pour ce faire, il fallait réussir à créer une atmosphère équilibrée, assez stressante avec du challenge sans être trop oppressante et trop frustrante. Pour immerger le joueur dans notre univers, nous avons fait le choix du tir à la première personne, ce choix se justifie par la vitesse à laquelle le joueur parvient à s'identifier au personnage qu'il incarne. De plus, une vue fps dans un labyrinthe ajoute une dimension pesante à la partie, les dimensions exiguës obligent le joueur à faire plus attention et à se préparer à chaque virage.
Cependant nous sommes conscients qu'un jeu entier dans cette ambiance est trop éprouvant. Dans cette optique, nous avons choisi que les arènes de boss doivent être larges et vastes pour permettre au joueur de souffler et varier son gameplay. Pour éviter de surcharger mentalement le joueur, nous avons également fait le choix de faciliter certains aspects du rogueLike/Fps comme la gestion des munitions qui doit être infinie.

Un des problèmes du genre Roguelike est la répétition des niveaux, des monstres, des armes… Pour palier à ce problème, nous avions imaginé un nombre important de mécaniques supplémentaires pour surprendre le joueur et se renouveler. Tout d'abord, une génération du labyrinthe de manière procédurale. De nombreux ennemis différents étaient prévus, des armes avec des efficacités et des pouvoirs variés. Pour créer une tension continue dans la partie, le jeu devait comporter un minuteur qui renforce les monstres au bout d'un certain laps de temps. De plus, des salles spéciales avaient été imaginées comme des salles de soins, des salles avec des mini boss, des salles d'épreuves proposant un challenge au joueur avec beaucoup de butin à la clé. L'objectif était de proposer beaucoup de choix au joueur pour avoir de nombreuses façons différentes de terminer le jeu.


HUD Bar
Consumable
Weapon
Slot
ShopItem
HUD Text
PlayerController
GearController
SlotController
ShopController
PlayerView
CanvasController
CanvasManager
GameManager
MiniMapController
MenuInteract
LevelBuilder
CustomBullet
EnemyStats
DialogManager
AudioManager
Doorway
Room
TurretAI
BossMammothAI
NPC
DialogData
Sound
EndRoom
StartRoom

# VII - Choix de développement

En premier lieu, pour travailler ensemble, nous avons choisi d'utiliser la fonctionnalité Collaborative de Unity qui permet une mise en commun des travaux du groupe. Le premier souci que nous avons rencontré venait du nombre limité de participants possible sur le projet. Seul trois personnes pouvaient travailler sur le projet avec la fonctionnalité Collaborative, Stéphane Lin a dû travailler de manière indépendante sur sa partie avant de nous envoyer son travail.

Lors du développement du projet, au vu des limitations de temps et de notre absence d'expérience sur Unity, nous avons dû faire certains choix, laisser certaines fonctionnalités de côté pour se concentrer sur l'essentiel. Nous avons partiellement sacrifier la diversité au profit d'un gameplay solide et fonctionnel. Nous nous sommes limités à 4 équipements différents, 3 ennemis différents, 2 boss et 2 niveaux.

Nous avons notamment décidé de récupérer plusieurs banques d'assets d'Unity pour réaliser l'aspect visuel de notre jeu. Les prefab des salles et l'arme du joueur ont été faits à la main sur le logiciel Blender. Les modèles de PNJ, d'ennemis, les effets de particules et les différents éléments du décor proviennent, eux, de banques d'assets gratuits fournis au Département d'Informatique et de Mathématiques. Ils permettent de donner au jeu son identité sans que leur conception ne prenne trop de temps sur la partie qui nous intéresse : les mécaniques de jeu. Nous avons réalisé tous les effets sonores du jeu nous-même, afin d'y ajouter une touche humoristique et personnelle, et pour permettre au joueur de bien discerner une action qui a lieu.

Lors du développement, nous avons essayé de généraliser nos classes au maximum afin de faciliter leur réutilisation. Dans cette optique, nous avons tenté de centraliser toutes les références à des éléments de la scène pour pouvoir instancier correctement nos objets de façon dynamique. Avec une telle implémentation il sera aisé d'augmenter le nombre d'étages, le nombre de boss, les possibilités du labyrinthe, le nombre d'ennemis et d'autres caractéristiques intéressantes permettant de donner plus de profondeur à notre gameplay.

# VIII - Planification du projet

Le premier sprint consistait à :

- Prendre en main Unity (Tout le monde)
- Créer un Player avec ses déplacements (Thibault Sellin)
- Créer un monstre qui se déplace et attaque (Basile Nguyen)
- Avoir une interface utilisateur (Clément de Gaillande)
- Explorer un premier étage (Stéphane Lin)
- Interagir avec différents items. (Clément de Gaillande)

Le second sprint a été consacré à :

- La notion de dégâts sur les monstre (Basile Nguyen)
- L'application des effets des items (Clément de Gaillande)
- La notion de drops des monstres tués (Clément de Gaillande)
- L'apparition de boss dans les arènes (Thibault Sellin et Stéphane Lin)
- La génération procédurale du premier niveau (Stephane Lin)
- La notion de dégâts infligés au joueur (Thibault Sellin)
- La création de différents monstres (Basile Nguyen)

Le troisième sprint a permis de :

- interagir avec un marchand (Clément de Gaillande)
- avoir un tutoriel (Basile Nguyen)
- générer procéduralement chaque niveau avec des monstres (Stephane Lin)
- avoir des menus (Thibault Sellin)
- avoir une ambiance sonore (Thibault Sellin)

Suite à la présentation, les éléments suivants ont été ajoutés :

- Les textures et le décor des salles (Stephane Lin)
- Des navMesh dynamiques (Basile Nguyen)
- La création dynamique des entités étant recréées à chaque nouvelle partie (Clément de Gaillande)
- IA ajustée sur le marchand (Basile Nguyen)
- Canvas Loading (Basile Nguyen)
- Cinématique (Thibault Sellin)
- Fonctionnement des consommables (stackable)(Clément de Gaillande)
- Correction des bugs de tirs (Cléments de Gaillande)
- Correction des bugs liés au téléporteur (Clément de Gaillande)
- Correction des bugs liés à la minimap
- Implémentation de la mécanique permettant de rejouer le jeu en boucle (Clément de Gaillande)
- Correction du fonctionnement des ennemis morts (Clément de Gaillande)
- Implémentation de la pause (Clément de Gaillande)
- Revue du projet (suppression de scripts obsolètes, fusion de scripts redondants, corrections d'erreurs d'implémentation dans la scène) (Clément de Gaillande)
- Ajout de la cinématique (Clément de Gaillande)
- possibilité de skip la cinématique (Clément de Gaillande)
- Création d'un exécutable (Clément de Gaillande)

# IX - Post-mortem

Le projet nous a permis de nous lancer dans le développement de jeux vidéo de façon pratique. Débutant sans connaissance préalable des techniques de conception et de développement de jeux vidéo, nous avons su en acquérir par la pratique lors de ce projet et tout au long de la théorie présentée en cours. Nous avons également pu illustrer différents acquis des autres modules orientés jeux vidéo de l'UQAC, comme le concept de RigidBody introduit en Mathématiques et Physique pour le Jeu Vidéo.

Le développement du jeu a connu des moments difficiles. Nous avons eu beaucoup de difficultés à coordonner l'équipe. L'outil Unity Collaborate, qui est le meilleur outil de gestion de versions pour Unity, avait une limite d'utilisateurs simultanés sur un projet. Cela fait que Stéphane Lin n'a pas pu avoir accès au projet que le reste du groupe développait. Cela l'a forcé à s'occuper de sa partie majoritairement seul, avec l'aide occasionnelle de différents membres du groupe pour intégrer son progrès dans le jeu. Nous devrons donc prendre cela en compte lors des prochains projets de développement sur Unity : la mise en place d'un gestionnaire de versions doit se faire dès le départ et de façon minutieuse, afin de permettre un bon workflow.

Nous avions une bonne cohésion d'équipe. Il n'y a pas eu de désaccord majeur entre les membres. Les outils comme Trello et Discord ont été efficaces et y ont grandement contribué.

Post-mortems personnels :

Clément de Gaillande :

A la fin de cette semaine, je suis vraiment satisfait des avancées faites sur le jeu, même si à chaque bug fix ou à chaque nouvelle fonctionnalité implémentée, une nouvelle venait à l'esprit. Ce serait ma seule frustration à l'issue de ce projet : ne pas pouvoir implémenter toutes ces fonctionnalités qui nous ont enthousiasmé (comme le fait que le marchand drop les items qu'il vendait si on le tuait, le système de badges, le système de rôles sur les ennemis …). Si je devais recommencer le projet, je pense également que j'éclaterais le gameManager en différentes classes, il est responsable de trop de fonctionnalités actuellement et je pense que cela impacte nos performances. Au-delà de ça, ça rend la lecture et la compréhension de cette classe plus difficile que nécessaire, tout comme le débuggage ou la refactorisation. Mais dans l'ensemble je suis très satisfait de notre jeu et des progrès et des connaissances acquises lors de la session grâce à ce projet.

Basile Nguyen :

Après la présentation devant le groupe, il nous restait encore du travail, le jeu n'était pas complet, il manquait les textures et certains bugs étaient encore présents. Durant le temps qu'il nous restait, nous avons réussi à beaucoup avancer et arriver à un résultat plus satisfaisant. Pour cela nous avons beaucoup échangé tous les jours avec une répartition efficace des tâches. Pour conclure le projet, le plus gros problème aura été la liaison avec le

travail de Stéphane. Notre communication a été efficace malgré quelques légers soucis au début. Notre GDD était ambitieux et intéressant mais la planification du projet aurait pu être améliorée en ajustant les tâches de chacun et en se concentrant sur le gameplay. Malgré cela, au terme du projet, le jeu est jouable, avec beaucoup de fonctionnalités comme le générique, les menus, les effets sonores… J'ai énormément appris et découvert au cours de ce travail, autant sur les IA que sur les animations, les décors, les assets et les UI.

Thibault Sellin :

Ce projet a été une bonne initiation au développement de jeux sur Unity et a donné un aperçu global des différentes problématiques auxquelles l'on fait face en débutant la conception d'un jeu vidéo à partir d'une simple idée. Le fait de devoir rester synchronisé avec l'équipe a parfois posé problème : certains assets peuvent avoir une taille très importante, or le débit de ma connexion Internet était parfois très faible, ce qui faisait que j'avais parfois des difficultés lorsqu'il fallait importer de nouveaux assets ou re-télécharger le projet, voire Unity en cas de crash très important. Établir du meilleur matériel sera important pour les prochains projets. Malgré cela, j'ai apprécié le fait de pouvoir découvrir une approche plus artistique à la programmation et d'avoir pu implémenter certains éléments qui me plaisent dans le jeu vidéo, comme la cinématique d'introduction, produite après la présentation.

Stephane Lin :

Pour un premier projet de développement d'un jeu sur Unity, je trouve le résultat global très satisfaisant. On a pu réunir les idées de chacun et faire un jeu qui représente bien notre groupe. Ma partie pour ce projet était un gros morceau qui a toujours quelques bugs. La génération procédurale des niveaux m'a permis de voir différents aspects du développement. Par exemple, créer ses propres modèles 3D avec Blender et ensuite les importer dans Unity et les manipuler à ma guise. Le problème avec Unity Collaborate m'a bloqué et je n'ai pu intervenir que pour rectifier les bugs sur la carte ou proposer des idées sur d'autres fonctionnalités. Enfin ajouter les différents assets pour le décor a donné une vraie immersion dans notre monde. Ce projet a été une grande et bonne introduction au monde du jeu vidéo.