Présenté par JTRD Studios

THE LAST SURVIVOR

Encadrants:
-M. Rémi Vernay
-M. Philippe Roussille

Membres du groupe:
-Delrieu Jules
-Dunand Tom
-Iammaren Driss
-Nakusi Romain(c)

***Introduction*:**

Commencé en milieu d'année, notre projet arrive enfin à son terme. Suite aux nombreux mois passés à expérimenter, échouer, réessayer, optimiser, corriger ou encore développer nous sommes parvenus à terminer notre jeu et atteindre les objectifs que nous nous étions fixés il y a de cela quelques mois. Nous allons ainsi vous présenter le rapport de notre projet. Pour ce faire, nous allons dans un premier temps rappeler les objectifs fixés en reprenant notre cahier des charges, puis dans un second temps présenter le développement de notre projet de façon chronologique et avec les tâches réalisés par groupe. Ce schéma se découpera en trois parties, représentant les périodes entre chaque soutenance. Enfin nous vous raconterons le récit de la réalisation avec nos joies et nos peines qui nous ont motivés comme ralenti durant ce projet. Cette dernière partie prendra une présentation plus individuelle car plus basé sur des sentiments ressentis pendant le projet.

Plan de notre rapport de soutenance finale:

Partie 1: Reprise du cahier des charges

I) Présentation du projet
II) Répartition et planification du projet

Partie 2: Présentation du projet :

I) Période entre la remise du cahier des charges jusqu'à la première soutenance
II) Période entre la première et la deuxième soutenance
III) Période entre la deuxième soutenance et le rapport final

Partie 3: Récit de la réalisation:

I) Récit de groupe
II) Récits individuels

Conclusion

Partie 1 : Reprise du cahier des charges

I) Presentation du projet

Origine et nature du projet :

L'idée de ce projet nous vient de notre enfance. En effet, Plants vs Zombies fut un jeu très populaire dès la sortie de son premier opus en 2009. Ainsi, il constitue un des symboles de notre enfance et c'est pour cela que nous souhaitons développer un jeu s'en inspirant. Le jeu que nous allons développer est un jeu de Tower defense (qui sera expliqué ultérieurement). Il existe de nombreux jeux de tower defense du fait de la grande diversité des mécaniques de jeu et également du fait que ceux-ci sont particulièrement adapté pour téléphone, qui représente un domaine très attractif du fait de l'utilisation des smartphones dans notre société et donc un marché plus important que les consoles.
Notre jeu sera en 2D et s'inspirera de la plateforme jeu de "plants vs Zombies": un échiquier où chaque ligne est une voie où les ennemis peuvent passer et les tours peuvent être posées . Nous allons partir de cette idée de protéger notre terrain contre des ennemis sur une carte quadrillé, et la développer pour faire de notre projet un jeu agréable à jouer.

Objet de l’étude :

Premièrement, nous réalisons ce projet dans l’optique d’obtenir un jeu vidéo opérationnel, complet, qui tout au long de son élaboration nous permettra d'acquérir les bases de la programmation d’un jeu vidéo, les qualités propres (parmi de nombreuses autres) à un ingénieur telles que l’organisation, l’autonomie (un travail régulier), le sérieux, la gestion et l’entente dans un groupe ainsi que de l'expérience vis-à-vis de ce projet (cela nous aidera beaucoup pour les nombreux projets en groupe à venir tout au long des cycles préparatoires et ingénieurs). Dans un second temps, programmer ce jeu nous permettra de nous familiariser avec le logiciel Godot, utilisé par de nombreuse entreprises professionnelles du jeu vidéo. Enfin, ce projet nous permettra d’augmenter notre niveau de compréhension en algorithmique et donc en programmation (notamment orientée objet), matières principales du cycle ingénieur épitéen. Ce projet constitue un véritable défi tant sur le plan collectif que sur le plan individuel.

État de l’art :

Tout d’abord, Le *Tower Defense*, ou TD, définit un type de *gameplay* dérivé des jeux de stratégie en temps réel. Très basiquement, il s’agira de tenir une position ou une forteresse tandis que des vagues successives d’ennemis tenteront de vous submerger. Le premier tower defense s’appelle Rampart, pour la petite histoire, un beau jour de 1989, John Salwitz (créateur de Rampart), s’occupa de la journée d’anniversaire de son fils aîné, âgé de 6 ans. Un exercice qui tourna vite à la gestion de crise pour le jeune développeur, qui dut alors canaliser 9 enfants animés par une seule et même envie : celle de tout ravager sur leur passage. John s’est employé à défendre chaque élément de mobilier en disposant à des endroits stratégiques de sa maison différents obstacles pour protéger ses meubles. En s’inspirant de cet événement, né l’idée du premier jeu de tower defense qui connaîtra au fur et à mesure un succès mondial. Rampart a été conçu par la célèbre franchise Atari Games en 1990 et remet en scène des batailles médiévales de l'époque. Nous nous inspirons également de kingdom rush et plants vs zombie (piliers du tower defense dont nous parlons juste après) qui à l’aide de défenses empêche les ennemis de passer. Les

graphismes de kingdom rush étant bien faits, nous allons donc nous inspirer de ceux-ci ainsi que du gameplay de plants vs zombies pour obtenir un tower defense complet.

Mais parlons plus en détail de ces jeux qui nous ont poussés à créer un tower defense:

-Plants vs Zombies

C’est un jeu vidéo de tower defense développé et édité par PopCap Games, sorti en 2009 sur PC. Le but du jeu est de défendre sa maison contre une invasion de zombies. Avant de rentrer dans la maison et de manger le cerveau du joueur, les zombies doivent passer par le jardin ou par la cheminée et c’est à l’aide de toute une panoplie de plantes et de champignons que le joueur pourra mettre en place sa défense contre des vagues de zombies.

Nous avons réalisé un tableau avec une échelle des caractéristique de 0 à 10. Cette échelle sert à donner un premier aperçu. 0 correspond à la valeur minimale relative et 10 à la valeur maximale de l'unité.
Les personnages ne sont pas définitifs ils servent uniquement d’exemples pour présenter l’ensemble des caractéristiques des personnages ainsi que les différentes possibilités.

Caractéristiques/Personnages	ennemi corps à corps de faible niveau	distance de base	ennemi à distance enflammé	soigneur
portée	1	5	5	3
vitesse d'attaque	5	3	2	5
vitesse de déplacement	5	2	2	8
puissance de l'attaque	3	2	3/1	0
points de vie	3	3	4	4
type de dégâts	instantané	instantané	instantané/ dégâts sur 3 secondes	instantané
zone de dégâts(en ligne)	1	1	1	3
capacité spéciale	aucune	aucune	insensible au feu, brûle la cible	soins des alliés

Découpage du projet :

Les différentes tâches :

- les attaquants : ce sont des personnages qui devront venir d'un côté. Le joueur ne les contrôle pas. Ils viennent aléatoirement (on ne sait pas quand,ni quels ennemis arrivent). Il faudra donc créer plusieurs types d'ennemis avec différentes caractéristiques. La puissance des attaquants dépendra du niveau et certains ennemis plus forts ne seront pas disponible dans les niveaux plus faciles.

- les défenses :Les défenses seront acquises en échange d'une monnaie gagnée en tuant les ennemies ou bien en temps de survie. Le joueur contrôle la position de ses défenses ainsi que certaines améliorations. Il va devoir poser des tours sur des cases. Elles auront leurs caractéristiques elles aussi avec, différentes portées, des dégâts de zones ou ciblés, un coût plus ou moins élevé. Cependant une fois posées, le joueur ne contrôle pas les défenses celles-ci se défendent automatiquement dès qu'un ennemi entre à distance d'attaque.
- l'ensemble des graphismes et la carte: les images à apporter pour tous les ennemis et tous les défenseurs selon leur état, les terrains des différents niveaux, les effets spéciaux (la mort d'un ennemi ou la destruction de la tour)
- le site web pour présenter le projet.
- le son du jeu : c'est-à-dire tous les bruitages des défenses et des attaquants et la musique de fond.

II/ Répartition et planification du projet

Répartition des tâches par personne:

tâches/responsables	Tom	Driss	Romain	Jules
défense	responsable	suppléant		
attaque			responsable	suppléant
site web	suppléant			responsable
graphismes + carte		responsable	suppléant	
son	responsable	responsable	responsable	responsable

Répartition des tâches pour chaque soutenance :

Le premier objectif est d'avoir au moins la carte quadrillée du premier niveau avec les attaquants qui arrivent aléatoirement

tâche / Soutenance	soutenance 1	soutenance 2	soutenance 3
défense	pouvoir poser les tours dans les différentes cases et commencer à faire les tirs	Fixer les tirs des tours et les bug et en créer de nouvelles avec d'autres capacités	créer des améliorations pour chaque tourelle et éventuellement des tirs spéciaux
attaque	faire les attaquants qui arrivent aléatoirement sur chaque ligne et qui avancent jusqu'à une tourelle	faire différentes troupes qui ont des capacités différentes et des pouvoirs	créer des améliorations pour les tourelles
site web		commencer et avoir toute la structure et les	finir l'ensemble du site

		images	
son	rechercher tous les sons et la musique de fond nécessaire	les synchroniser avec les attaques et défenses	
graphismes + carte	avoir une première carte et les graphiques d'un premier attaquant et défenseur	faire tous les autres graphiques des nouveaux attaquants et défenseurs et commencer d'autres cartes	ajouter plusieurs cartes et potentiellement de nouvelles apparences pour chaque tourelle et chaque attaquant

Partie 2: Présentation du Projet:

I) Période entre la remise du cahier des charges jusqu'à la première soutenance:

a) Scène de la carte:

Nous avons commencé par créer la carte: en effet, nous avons créé une scène avec comme racine un *node2D* pour pouvoir avoir une scène complète.
Cependant, avec l'instanciation à venir, ce noeud va disparaître avec le noeud principal. Nous avons créé un noeud *tilemap* en enfant du *node2D***.** Cette carte est utilisée pour les tests. Nous avons ensuite mis en enfant du *node2D,* un noeud *Path2D* pour tracer les chemins que les ennemis sont censés prendre.

Ensuite, en enfant de celui-ci nous avons ajouté un noeud *Pathfollow2D* pour que les *Sprite* ou *Animated Sprite* (élément graphique respectivement inanimé ou animé) en noeud enfant du *Pathfollow2D* suivent ainsi le chemin indiqué. L'avantage du noeud *Pathfollow2D* est la propriété "Offset" qui permet de se déplacer sur le chemin.
Pour utiliser cette propriété nous avons attaché un script au *Path2D***.** Dans le script nous avons créé un objet de type *Pathfollow2D* que nous avons initialisé dans la méthode "_ready" de Godot comme étant le noeud *Pathfollow2D*. De plus, dans la méthode "Process", nous avons modifié la valeur courante de la propriété "offset" avec un getter et un setter. Cependant, suite aux retours que nous avons reçu lors de la première soutenance, nous avons abandonné le PathFollow2D pour faire directement varier l'abscisse de nos ennemis. En effet ceux-ci se déplaçant en ligne droite, une fonction process permet donc à elle seule de les déplacer correctement.
C'est à ce moment que nous avons compris que nous avions une connaissance du logiciel très approximative notamment dans l'idée de comment créer le jeu.
C'est alors après cette soutenance que nous nous sommes sérieusement mis à travailler et à s'informer tant bien sur la documentation godot que sur les forum et les différent tutoriel.
De plus, nous avons créé un animated sprite avec des actions différentes sur des ennemis différents (par exemple un mort vivant qui attaque ou un zombie qui marche). Enfin, nous avons commencé à essayer d'ajouter un timer pour créer des vagues d'ennemis, qui verrait alors leurs nombres d'ennemis augmenter plus on avance dans le jeu.

b) Scène des ennemis:

Cette scène est la scène qui correspond aux ennemis. Nous l'avons que très peu développé durant cette première période. De plus, nous l'avons fortement modifié par la suite en créant une scène pour chaque type d'ennemi. Dans celui ci il y a différentes classes: une classe où il y a toutes les caractéristiques des ennemis telles que les dégâts, la vie, la vitesse, le type(zombie,archer...), d'autres classes correspondent à chacun des types d'ennemis dans lesquels sont initialisés les attributs.

Dans la classe "ennemis" il y a une méthode qui retourne la vitesse de l'ennemi en fonction de son type.

c) Graphisme:

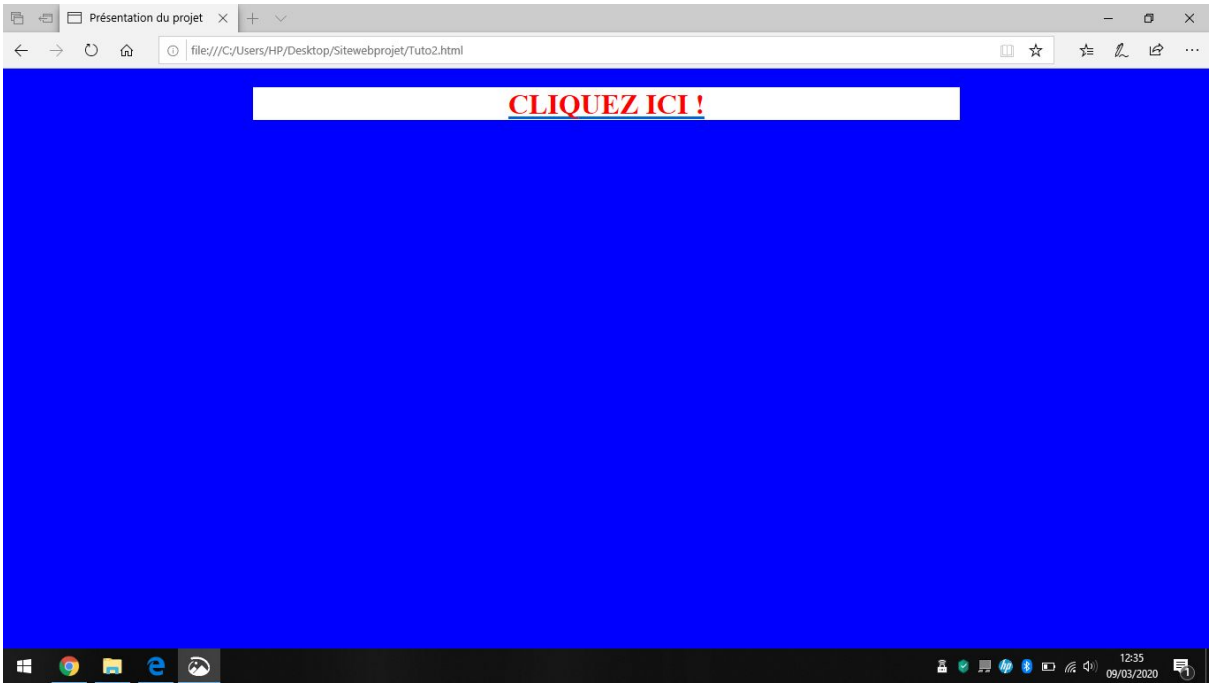
Pour la tilemap:

- A l'aide d'une texture sur le site Kenney, nous avons créé différentes tuiles. Elles ont permit de créer la map en les plaçant sur les cases.
- A l'aide du logiciel "Piskel" nous avons créé des tuiles de différentes couleurs.
- De plus, nous avons pu trouver de nombreux graphismes pré-existants avec différentes animations différentes, sur le site opengameart.org, conseillé par la documentation de godot. Ceci nous a donc fait gagner beaucoup de temps au niveau de la recherche des graphismes. Les graphismes disponibles ont donc inspirés l'univers dans lequel se déroulera notre jeu vidéo, qui était encore indéterminé au moment de la validation du cahier des charges. Il s'agira donc d'un jeu qui se deroulera dans un univers fantastique avec différentes sortes de créatures surnaturelles comme des zombies ou des elfes qui attaqueront et seront alors repoussé par des tourelles comme des tanks.

d) Site

Tout d'abord, nous avons commencé à créer notre site web, nous nous sommes donc inspiré d'un tutoriel pour apprendre à coder en html et en css: en effet, dans un premier temps le code en html va nous permettre de créer des titres, des zones de

textes, et des images que l'on peut centrer, aligner. Tout ceci est permis grâce à des balises qui vont servir à soit mettre du texte, soit faire des listes, soit mettre des images et pleins de choses encore... Ainsi, la création de notre site web va être très intéressante car nous allons devoir constamment améliorer celui-ci tant sur la structure du site que sur son illustration(ce qui est propre au travail d'un ingénieur: innover constamment et toujours optimiser au mieux son travail). De plus, grâce à la fonction pour les textes<a href="le chemin d'arrivée" notre texte et pour les images <a href="le chemin d'arrivée" <img src="notre image", nous avons pu créer des liens qui amènent vers d'autres pages html que nous avons nous même confectionné. Nous avons donc mis des boutons sur un bandeau en haut de page qui amèneront aux différentes pages de notre site. Pour l'instant, les liens mènent vers des pages vides puisque ce n'est que le début de notre site web.



Voici des photos de la première version de notre site web.

```html
<!DOCTYPE html><!--On dit quel language on utilise-->

<html>
    <head><!--Les trucs qu'on écrit et qu'on veut pas forcément voir-->
        <meta charset="utf-8"><!--codage de caractères informatiques-->
        <title>Site web de notre jeu</title><!--titre de la page-->
        <link rel="stylesheet" href="style.css"><!--Permet de changer la couleur du texte-->
    </head>

    <body>
        <!--Nom de notre balise pour la modifier dans le css-->
        <div id="Nomdutruc"><!--Créer des blocs de contenus, genre des blocs de couleur.-->

        <!--Le texte qui sera écrit sur le site-->
        <h1>Notre site web</h1><!--C'est le titre important, h2 le titre moins important...-->

        <a href="Tuto2.html"><!--On indique vers quel fichier va le lien-->
            <img src="C:\Users\HP\Desktop\Sitewebprojet\Terreordi.gif"></a><!--On insère une image et on fait de celle-ci un lien à suivre
            grâce à la balise a.-->

        <p><!--Zone de texte mamen--> Plants versus Zombies</p>

        <h2><!--Titre un peu moins important--> Notre plan:</h2>

        <ul><!--On fait une liste de choses-->

            <a href="Tuto3.html"><li><!--Premier point de la liste-->Présentation du projet<br><br></a>
                <img src="C:\Users\HP\Desktop\Sitewebprojet\Hacker.jpg"></li>

            <a href="Tuto3.html"><li><!--Deuxième point de la liste-->Archive du rapport et du projet<br><br></a>
                <img src="C:\Users\HP\Desktop\Sitewebprojet\Iphone.gif"></li>
                <br>
```

Voici un bout de notre code html, servant notamment à afficher le texte de notre site web.

```css
h1
{
        color: red;/*Changez la couleur du texte compris dans la balise h1*/
}

body
{
    background-color: blue;
}

#Nomdutruc
{
    width: 800px;/*largeur du bloc qui est de 800 pixels*/
    margin:0 auto;/*place la zone de 800 pixels au milieu de la page*/
    background-color: white;/*couleur du fond du bloc soit blanche*/
    text-align: center;
    align-content: center;/*centrer le texte et les images*/
}

ul li/*ça sert à ne plus afficher de petits points quand on fait une liste*/
{
    list-style-type: none;
}
```

Voici notre code css servant à colorer le texte, ainsi qu'à aligner le texte.

Avec le temps, les différents liens (présentation du projet etc) mèneront vers 3 parties principales concernant notre projet telles que :

La partie 1:

- Elle comprend en grand titre "Présentation du projet" où on y retrouve l'historique, des détails sur les membres du groupe, la chronologie de réalisation, les problèmes rencontrés ainsi que les solutions envisagées.

La partie 2:

- Cette partie s'intitulera les "Les liens sur les sites" et elle comprendra des informations sur les membres, les logiciels utilisés et pourquoi, les images, les différents sons utilisés pour le jeu, la bibliographie avec certaines de nos sources, les applets (petites applications ou logiciels qui nous aura été utile) et d'autres éléments qu'on pourrait utiliser.

La partie 3:
- "Archives du rapport et du projet" comprenant une version allégée de celui-ci (sans les introductions, séquences AVI et toutes choses inutiles à l'exécution du projet).

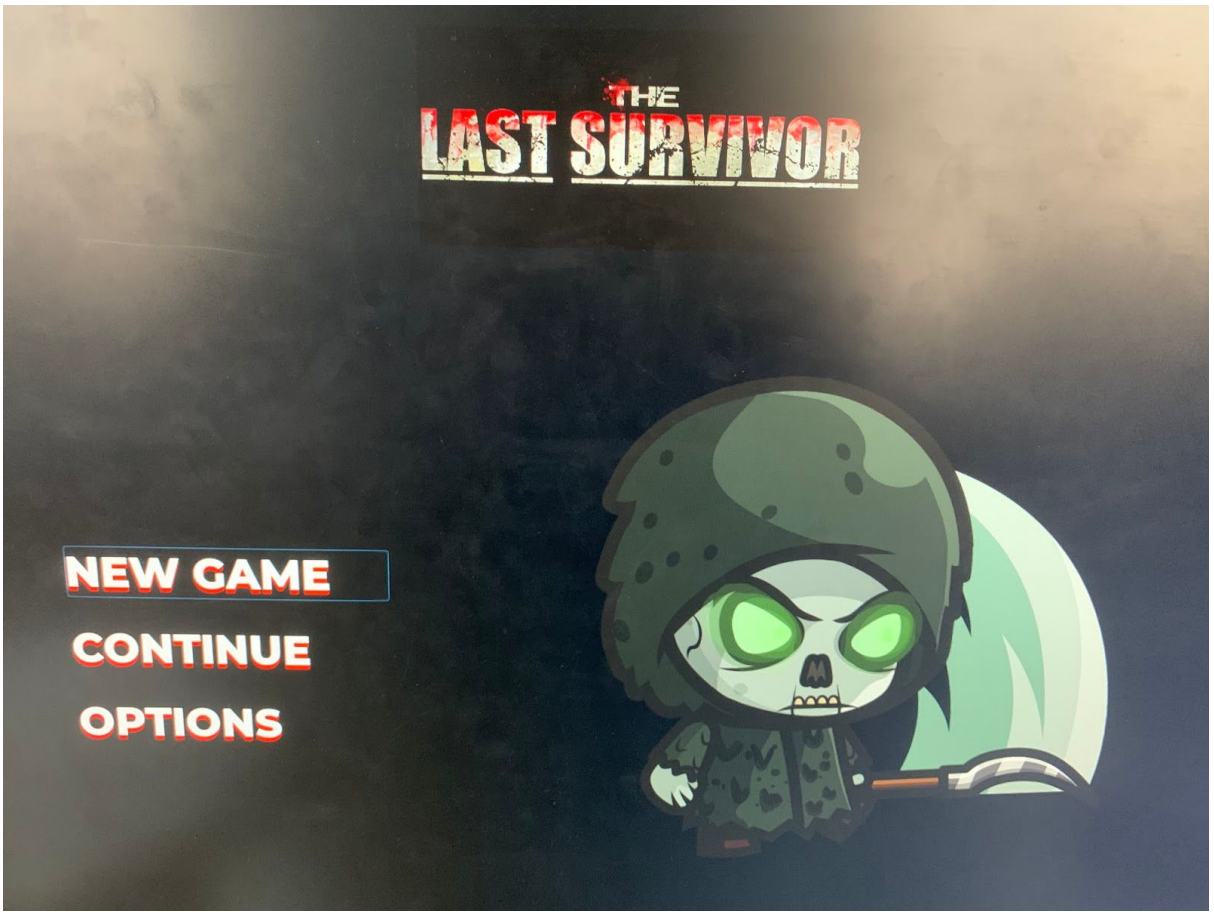
Ensuite, l'idéal serait que le site web soit publié sur internet: en effet cela permettrait de pouvoir y accéder sans avoir à télécharger tous les fichiers le

composant tel que les images, les gif, le code html et css. Cela serait donc un gain de temps énorme et nous avons donc prévu de mener cette idée à bien(cela ferait aussi beaucoup plus professionnel).

e) Menu

Nous avons aussi créé un menu servant d'écran d'accueil, cependant suite à quelques problèmes d'utilisation de git nous avons perdu celui-ci. Nous disposons (pour l'instant) seulement d'une photo de ce menu avec le téléphone de Driss Iammaren et nous comptons donc refaire ce même menu bien-sûr. Il est composé d'une image avec le nom de notre jeu en haut au centre. En bas à gauche nous avons disposé trois boutons: le premier est le New Game qui amènera vers une nouvelle partie quand nous aurons relié notre menu avec le jeu. On retrouve aussi le bouton Continue permettant de continuer une partie en cours ainsi que le bouton Options où le joueur pourra par exemple choisir la difficulté qu'il souhaite et d'autres actions que nous implémenterons plus tard.Selon les difficultés rencontrées pour la sauvegarde d'une partie en cours et donc le bouton continue, celui ne figurera peut-être pas sur la version finale bien que nous essayerons d'y parvenir. Il y a aussi une plus petite zone de texte où est écrit : "Version 1.0 JTRD Studios" permettant de connaître l'avancement et les mises à jours de notre jeu. Nous tenons à vous faire savoir que JTRD Studios vient tout simplement de la première lettre de nos prénoms respectifs à savoir Jules, Tom, Romain et Driss. Une image est aussi présente à droite (environ la même que sur la première page de ce rapport de soutenance) pour montrer à quoi s'attendre en termes d'ennemis dans le jeu: il y en aura de toutes sortes, des ninjas, des zombies… et la bataille qui va faire rage dans ce jeu de Tower Defense ne pourra donner lieu qu'à un seul survivant d'où le nom de notre jeu "The last Survivor", qui a pour but avec la couleur sombre du menu, la couleur des boutons en rouge et blanc d'instaurer une atmosphère d'angoisse mais aussi et surtout d'envie de jouer(l'atmosphère du jeu et la musique que nous associerons plus tard avec le menu intéressera le joueur et le poussera donc à jouer).
D'un point de vue plus technique, pour créer ce menu, nous nous sommes de nouveau aidé d'un tuto: en effet, on a premièrement changé la couleur du fond en noir et on a inséré le noeud principal nommé TitleScreen qui aura la dimension de la fenêtre qui s'affiche lorsqu'on compile(F5). Il a un noeud enfant Menu qui va donc être un rectangle dans lequel on y mettra les différents boutons. Ce noeud Menu a lui même plusieurs noeuds enfants: Logo(où on met le logo de notre jeu) et CenterRow(ligne centrale). Dans CenterRow, on retrouvera alors les boutons NewGame, Continue et Options ainsi que l'image montrant un des personnages du

jeu. Enfin, toujours en noeud enfant de Menu, on rajoute VersionLabel où figure Version 1.0 JTRD Studios.



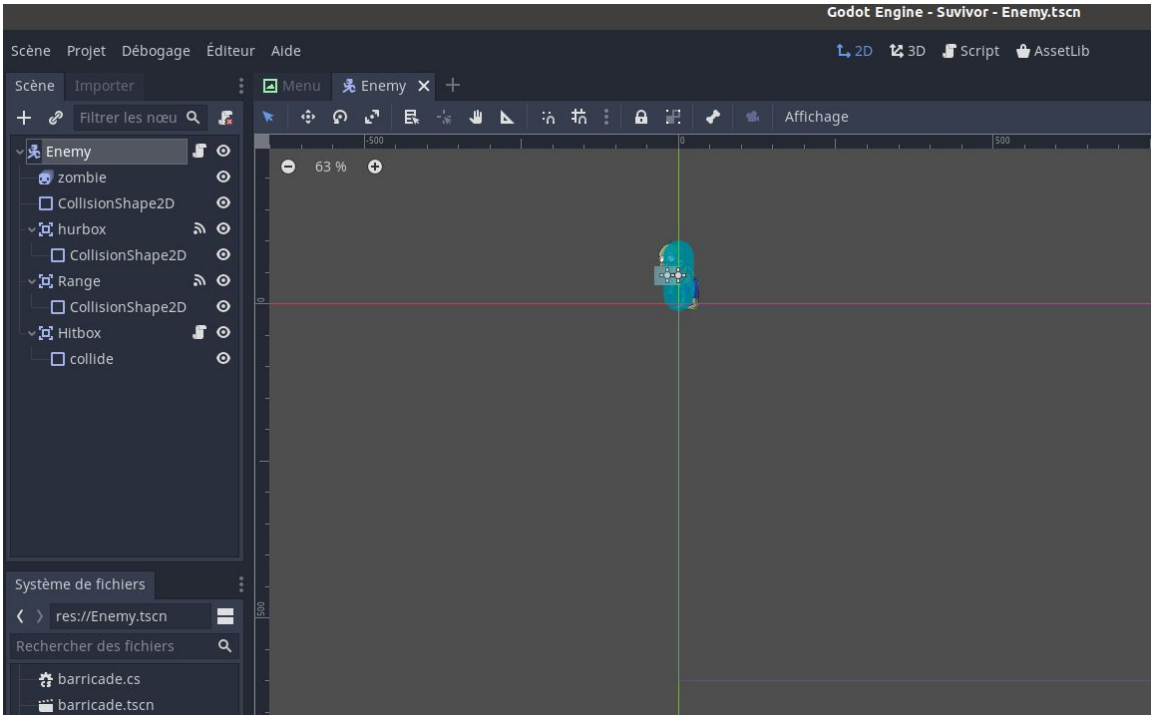
Voici la seule photo de notre menu

II) Période entre la première et la deuxième soutenance:

a) Attaque:

Nous avons construit une scène attaque qui servira de modèle définitif pour chaque personnages.
Elle est constituée d'un noeud Node2D qui sera la base de notre attaquant. C'est ensuite un noeud KinematicBody2D qui forme l'objet personnage, auquel nous avons attribué une forme et une apparence grâce aux noeuds CollisonShape2D et AnimatedSprite2D en noeuds enfants. Ainsi le personnage pourra interagir comme un corps physique et pourra être détecté par un noeud Area2D. En enfant du noeud

Node2D.



Il y a trois Area2D avec chacun une forme et une fonction différente. En effet, nous avons besoin de ces trois aires de détection, connectées au script, pour implémenter les actions d'un attaquant.
La première est une "hurtbox" c'est à dire une aire permettant de détecter les noeud KinematicBody2D si le noeud entrant à la valeur d'un projectile une méthode "gethurt" est lancée pour prendre des dégâts, et selon le nombre de points de vie restants, lancer l'animation "Death" du noeud AnimatedSprite2D puis disparaître si l'objet représentant sa vie est de valeur nulle.
La seconde est une "hitbox" c'est à dire une aire qui permet de se faire détecter par les défenses. Ce noeud à peu de code attribué dans cette scène car il servira à ce que les tours puissent prendre des dégâts. C'est pour cela que cette aire n'est active qu'à une image précise de l'animation "Attack" pour que les tours prennent des dégâts au bon moment.
La dernière est une aire ayant pour but de détecter les noeuds StaticBody2D ayant la valeur d'une défense. En effet, lorsqu'une tour est détectée l'attaquant s'arrête et l'animation "Attack" est lancée et lorsque la tour n'est plus détectée par exemple si elle disparaît le personnage reprend son chemin et l'animation "walk" est lancée. C'est le seul que nous avons trouvé pour faire s'arrêter le personnage devant une défense. Cependant, nous cherchons un autre moyen qui ne nécessiterait pas un noeud Area2D en plus.
Les mouvements du personnage sont implémentée avec une fonction "physique process"dans laquelle la propriété position de la classe attaquant héritant du noeud Node2D donc de l'ensemble de l'objet attaquant est modifié.

La scène disparaîtra une fois le terrain traversé ou le personnage “mort”.

b) Invocateur (nommé spawner)

Cette scène a pour utilité de faire apparaître des attaquants elle est constituée de deux noeud un Node2D où est attaché le script et un noeud Timer avec un signal connecté au script. Pour que les personnages apparaissent la scène attaquant est chargé et chaque seconde une variable stock la scène instancié qu’on ajoute ensuite en enfant du noeud contenant le script.

c) Projectile

Cette scène va de paire avec la scène Defense, elle est constituée d’un noeud Node2D auquel est attaché le script avec en enfant un noeud KinematicBody2D ayant la forme et l’apparence du projectile désiré en l'occurrence pour l’instant un missile.
De même qu’un noeud Area2D qui envoie un signal au script quand un noeud KinematicBody2D est détecté. Si celui-ci a la valeur d’un attaquant la scène projectile est supprimé.
Ce noeud sert uniquement à pouvoir supprimer la balle quand elle rentre en contact avec un attaquant.
Le projectile avance avec une fonction “physique process” dans lequel on modifie la position de la classe du projectile cette classe hérite des propriété d’un noeud Node2D. Si le projectile dépasse sa portée, il disparaît. Nous avons par la suite revu notre façon d’infliger des dégats aux ennemis pour simplifier l’amélioration des tours défensives.

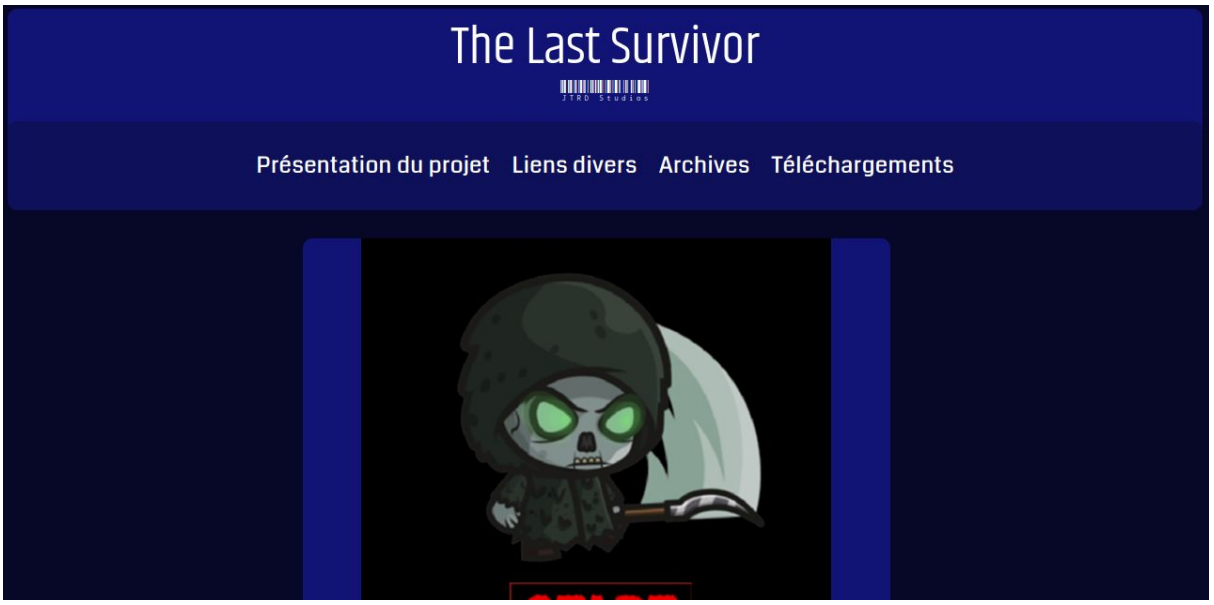
d) Défense

La scène défense actuelle servira tout comme la scène attaque actuelle de modèle pour tous les types défenses . Elle est constitué d’un noeud Node2D où est attaché le script avec en enfant de celui-ci un noeud StaticBody2D qui formera l’objet défense avec une forme et une apparence tout comme l’attaque grâce aux noeuds CollisionShape2D et Sprite.
Il y’a pour l’instant en enfant du noeud de base en plus du noeud formant le personnage, un noeud Area2D représentant la portée de la défense et un noeud Timer représentant la cadence de tir.
En effet tant qu’un attaquant est dans l’aire de tir, la scène projectile chargé initialement est instancié dans une variable qui est ensuite ajouté au noeud Node2D.
Si il n’y a plus d’attaquant la scène projectile n’est plus instancié.

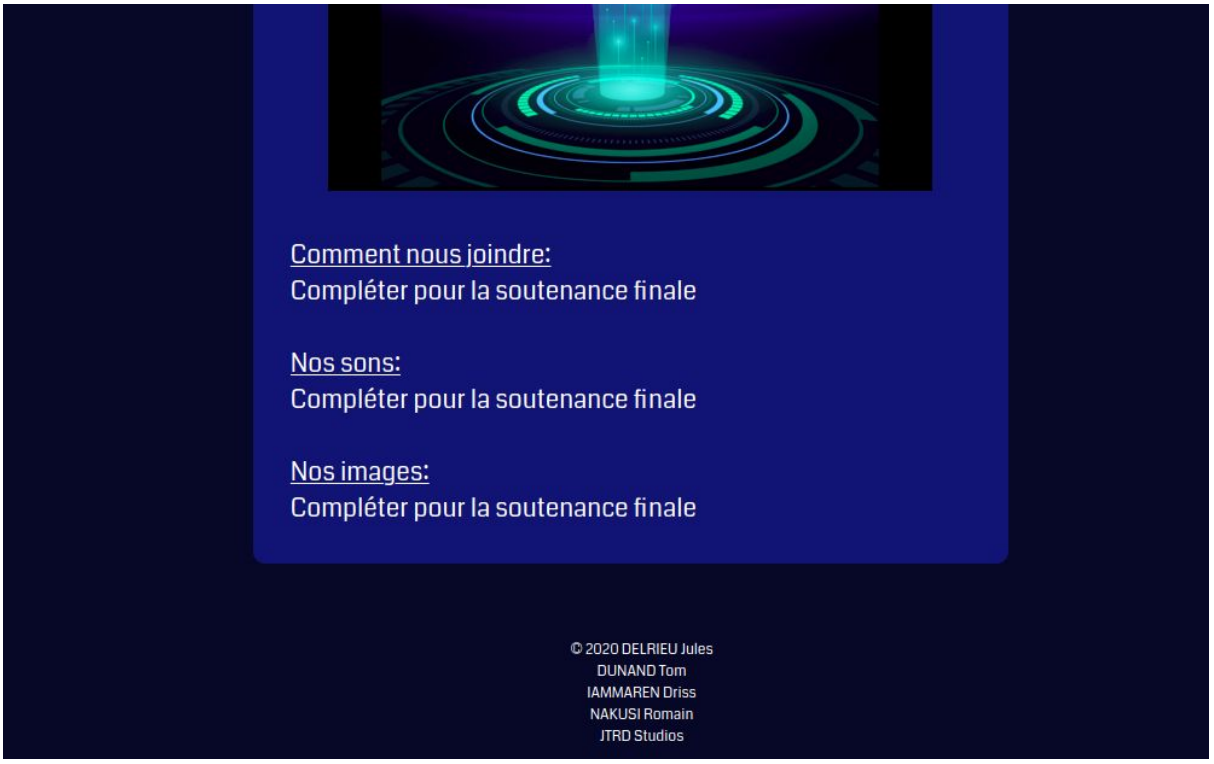
e) Site:

Concernant le site web, sa conception a grandement avancé: en effet, nous sommes passés d'un site "bâteau" à un site bien développé, et ceci notamment grâce au fait que nous sommes devenus beaucoup plus à l'aise en html et en css.
Notre site pour la première soutenance:

Notre site pour la deuxième soutenance:



On peut voir ici un fragment de notre site et de ses options. Nous l'avons organisé de façon à ce qu'il soit complet en effet, on remarque 4 différentes parties: Présentation du projet, archives, liens divers et une nouvelle partie nommé "téléchargements" sera présente où l'on pourra télécharger notre jeu. Pour l'instant, à par l'accueil de notre site web, aucunes autres pages n'est définitive (en terme de texte et d'images). De plus, les couleurs et la forme du site pourrait changer selon nos goûts du moment. Pour rendre le site sérieux, nous avons attribué la même forme à chaque page. Pour la page d'accueil, on retrouve un slider avec différentes photos de notre projet ainsi qu'une courte description de celui-ci et de notre groupe. La conception de ce site nous a pris énormément de temps car il compte pour l'instant environ 500 lignes de code mais c'est un choix de notre part pour qu'il soit vraiment complet. Nous utilisons une seule page Css pour la forme de notre site ce qui permet notamment de mieux s'y retrouver lorsque nous codons.



Voici à quoi ressemble la page “Liens divers” pour le moment

Nous avons eu quelques difficultés concernant les images : en effet, pour le slider, les images devaient être de même tailles, ainsi il fut au début très dur de les intégrer à notre site. Nous avons donc eu l’idée de redimensionner et retoucher nos photos à l’aide de Photoshop afin que les images soient parfaites. Nous avons donc pu facilement insérer n’importe quelle photo de n’importe quelle taille à notre site.

f) Menu:

Au niveau du menu, nous en avons recréé un avec une animation qui va de haut en bas avec le personnage. Pour ce faire, Nous avons créé une scène composée de 5 noeuds: un noeud principal, TextureRect que nous avons renommé Background car c’est en fait le noeud qui correspond à notre fond de menu (en noir sur les photos). Le noeud Background est ensuite composé de 4 noeuds enfants; en premier lieu un noeud TextureRect à nouveau, renommé Logo et qui correspond tout simplement à notre personnage, ensuite nous avons un deuxième noeud enfant, un Texturebutton qui correspond à notre bouton start, pour l’instant nous n’avons créée que l’image de celui-ci à l’état normal mais il se peut que nous créons aussi à l’aide de photoshop son image à l’état pressé. Nous avons aussi un troisième noeud enfant qui est un Label et qui sert tout simplement à afficher le nom de notre groupe JTRD Studios en bas à gauche du menu. Enfin, nous avons notre quatrième et dernier

noeud enfant qui est un AnimationPlayer que nous avons renommé Animation. Il sert à créer un mouvement du personnage qui lorsque l'on lance le jeu apparaît en descendant pour s'arrêter au milieu de la page. Pour créer cette animation, nous avons utiliser l'éditeur d'animation puis attribué au noeud Logo 2 positions différentes : la première position (118, -400) pour que le personnage soit en dehors de la fenêtre puis la seconde position (118, 0) pour qu'il s'affiche au milieu de notre menu.

Voici une explication plus compréhensible en image:

Personnage à la position (118, -400), il n'est donc pas visible sur l'écran

Voici maintenant le personnage à la position (118, 0), il effectue donc un glissement du haut vers le bas:

III) Période entre la deuxième soutenance et le rapport final:

a)Attaque

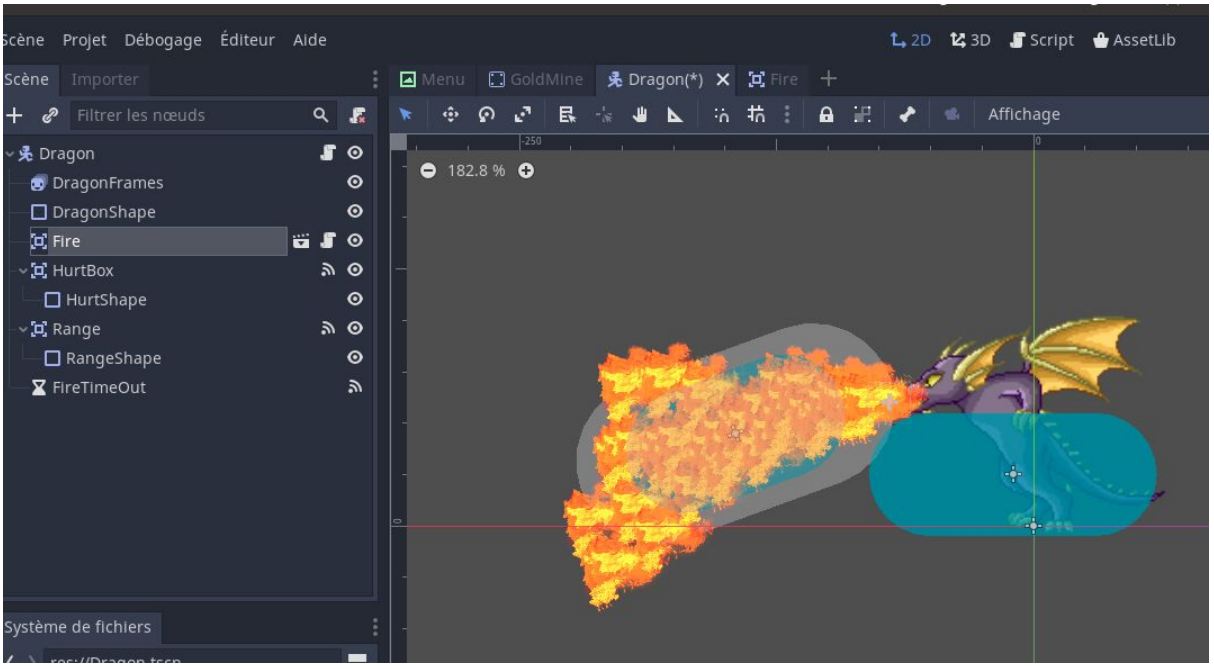
Au niveau de l'attaque nous nous sommes servis de la base créée afin de faire différentes sortes d'ennemis possédant tous des caractéristiques différentes afin de faire varier nos vagues d'ennemis. Chaque ennemi possède malgré tout une base similaire. Au niveau des noeuds, les ennemis possèdent tous une racine commune, un noeud KinematicBody2D. En effet nous cherchons à avoir des ennemis mobiles pouvant être détectés par les aires de détection. Ils ont tous également un Animated Sprite du fait qu'ils soient mobiles, l'utilisation d'un sprite inanimé aurait donné un effet de glissement et non de déplacement des ennemis, contrairement aux tours défensives, qui elles sont immobiles. De plus on retrouve trois Area2D avec un noeud enfant CollisionShape, une qui sert de Hurtbox, c'est à dire la zone qui doit être touchée pour qu'ils reçoivent des dégâts, une Hitbox qui au contraire sert à infliger des dégats et enfin une range, qui représente la portée des attaques.

Cependant certains ennemis possèdent des scènes différentes du fait de leurs caractéristiques que nous verrons plus tard. Au niveau du script, la base du script possède de nombreuses méthodes. Tout d'abord tous les ennemis possèdent une variable hp qui représente leurs points de vie. Lorsque cette variable descend à 0 ou

en dessous, les ennemis sont éliminés et ne deviennent ainsi qu'une animation jouant leur mort après la désactivation de la "Hurtbox"puis disparaissent. De plus, une variable speed qui, elle est la vitesse à laquelle ils se déplacent. tout comme les méthodes, les variables varient grandement en fonction de chaque ennemi. Voyons maintenant les différentes méthodes des ennemis. Tout d'abord ils possèdent tous une méthode Process, qui s'exécute très rapidement et automatiquement à chaque images (frames). Cette méthode leur permet d'avancer de manière fluide notamment et de disparaître lorsqu'ils arrivent de l'autre côté de la carte, et ainsi d'enlever une des vies du joueurs, qui perdra le niveau si trop d'ennemis réussissent à traverser.
Les ennemis ont des méthodes faisant quatre choses distinct prendre des dégâts, attaquer, avancer-s'arrêter et jouer les animations correspondantes.

En effet l'ennemi prend des dégât lorsqu'il détecte un projectile d'une défense à l'aide d'une méthode qui prend en paramètre les dégâts du projectiles détecté et les soustrait à sa vie. Alors si sa vie est inférieur ou égale à zéro un booléen présent dans la fonction Process devient vrai ce qui qu'un autre partie du code est lancée. Ainsi l'ennemie n'avance plus, n'attaque plus...
Pour l'attaque il y'a une méthode qui active une aire de détection la "HitBox" et la désactive synchronisé avec l'animation adaptée pour que les défenses détectent l'attaque de l'ennemi au bon moment.
Cette méthode est alors utilisé dans Process.
Ainsi beaucoup de méthodes ne sont que des conditions vérifiées à chaque image (frame) grâce à la fonction Process.
Les déplacements sont influencés d'une part avec la fonction Process qui est fait en sorte que les ennemies avance à vitesse constante de manière fluide et d'autre part par les détections des défenses grâce à l'aire de détection représentant sa portée. En effet quand une tour est à porté l'ennemi s'arrête et quand elle est détruite l'ennemis reprend sa course.
Enfin il y a les méthodes pour l'animation qui servent uniquement à rendre le jeu plaisant visuellement et servir de base pour l'attaque. En effet l'attaque est lancée uniquement à une certaine image de l'animation prévu à cet effet elle même lancée lorsqu'une défense est à porté. De même l'animation "Walking" fait pour quand le personnage avance est lancée quand sa vitesse n'est pas nulle.
Enfin lorsque l'ennemi n'a plus de vie le booléen "isko" devient vrai et une méthode désactivant sa "Hurtbox" et lançant l'animation "Dying" est lancé dans la méthode Process ainsi quand l'animation est finie le personnage meurt. Cela représente donc la base des ennemis c'est à dire les ennemis au corps-à-corps.
Cependant il existe aussi la création des personnages avec des caractéristiques avancées. L'implémentation et la création de ces personnages est différente sur certains points.

Notamment “ l'archer” un personnage à distance qui tire des flèches en courbe selon des paramètres physiques en partie à l’aide d’un noeud “Area2D” situé en tête de flèche qui permet de simuler la courbe et la rotation d’une flèche affecté par la gravité. La courbe est affecté de même par la distance entre celui-ci et la défense détecter la plus proche qui définit le vecteur initial de lancement des flèche. Ainsi la cible peut être touché peut importe la distance tant que cela reste dans la portée de l’archer. L’objet distance est un entier non statique pour que chaque archer ait sa propre distance, elle est transmise aux flèches par un objet statique dans le script des flèches qui sert juste d’intermédiaire entre les deux scripts pour donner le vecteur initiale au jeu. Les flèches sont alors une scène à part instancier de façon synchronisé avec l’animation adapté.
L’archer a été conçu pour changer de cible si une défense plus proche à été construite et de garder son ancienne cible de côté.
Cela a été rendu possible uniquement grâce à une liste de couple corps physique correspondant à la défense et décimal correspondant à la distance les séparants. Chaque nouvelle défense détecter est associé à son couple respectif et est inséré de façon croissant pour que le dernier couple soit celui de la défense la plus proche qui donne alors sa valeur distance à la flèches. En effet la distance est négative donc le dernier élément de la liste est la plus petites distance.
Ainsi nous pouvons savoir si une défense à été détruite et laquelle en utilisant cet liste dans une fonction qui se lance lorsqu’une défense détecté est détruite.
On peut alors supprimer le couple contenant cet défense grâce à une méthode prenant en paramètre le corps détecté et ainsi avoir toujours la bonne cible et trajectoire.
De même, il y a le “dragon” qui crache du feu créé à l’aide de particules paramétrées correctement pour reproduire l’idée que l’on a des flammes d’un dragon. Celles-ci sont émises pendant trois secondes puis infligent des dégâts à la fin de ces trois secondes. L’animation de l’attaque est alors arrêtée le temps de la flamme.
Si la tour disparaît avant la fin des trois secondes les flammes s’arrêtent et le dragon recommence à marcher.
De plus la forme des animations du dragon change donc il fallait réajuster à chaque fois l’aire de collision.
Cela a été rendu possible grâce à une méthode lancée dans Process qui réajuste la position et la rotation des différents noeuds “CollisionShape2D” qui épousent la forme du dragon.

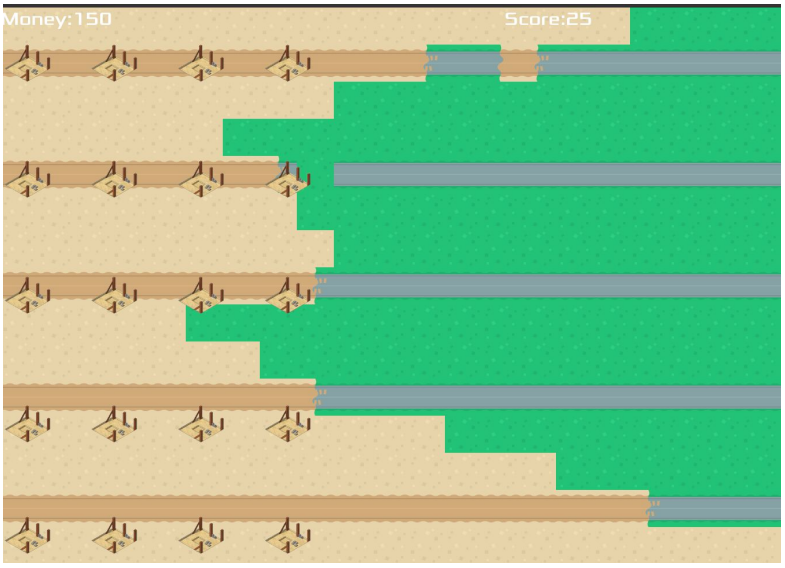


En plus de cela le dragon joue une animation et s'arrête quand il se prend des dégâts pour rajouter un côté visuelle et "réaliste" au personnage.
C'est le personnages le plus abouti du jeu avec l'archer.

b) Différence entre les niveaux

Afin d'apporter une bonne expérience de jeu au joueur il est nécessaire de permettre au joueur de progresser au fur et à mesure du jeu. C'est grâce à cela que beaucoup de jeu nous stimulent et nous donnent envie de continuer. Un jeu trop difficile d'entrée n'est pas amusant, et un jeu trop facile est très vite lassant.
Pour reproduire cela nous avons donc créé plusieurs niveaux avec une difficulté croissante. Réussir à rendre la difficulté des niveaux variables sans les rendre impossible a été difficile. La principale difficulté qui change dans les niveaux c'est la difficulté des vagues. Les attaquant arriveront en plus grand nombre et avec une plus grande fréquence. Ainsi le joueur devra s'occuper de plus d'ennemis à la fois.
De plus nous avons fait varier la quantité d'argent au début du jeu. En effet, suite aux nombreux essais du jeu que nous avons fait il apparaît que le début de partie est le moment le plus compliqué et en même temps le plus intéressant. Bien commencer est donc primordial c'est pour cette raison que nous avons décidé de diminuer l'argent de départ ainsi que de ne pas faire des vagues trop longues. C'est également le principal but du dragon, l'ennemi dévastateur qui à lui seul peut raser

quasiment une ligne entière et ainsi rééquilibrer le rapport de force dans les moments où le défenseur possède le plus souvent l'avantage.
La deuxième difficulté qu'il va rencontrer c'est le nombre de voies qui passe de 3 puis 4 et 5 pour le troisième niveau. Avec cela le joueur aura non seulement plus d'attaquants à défendre mais aussi plus de voies. Il aura à poser plus de tourelles donc à avoir une meilleur économie. Bien utiliser l'argent de son jeu est la difficulté principale de ce jeu.

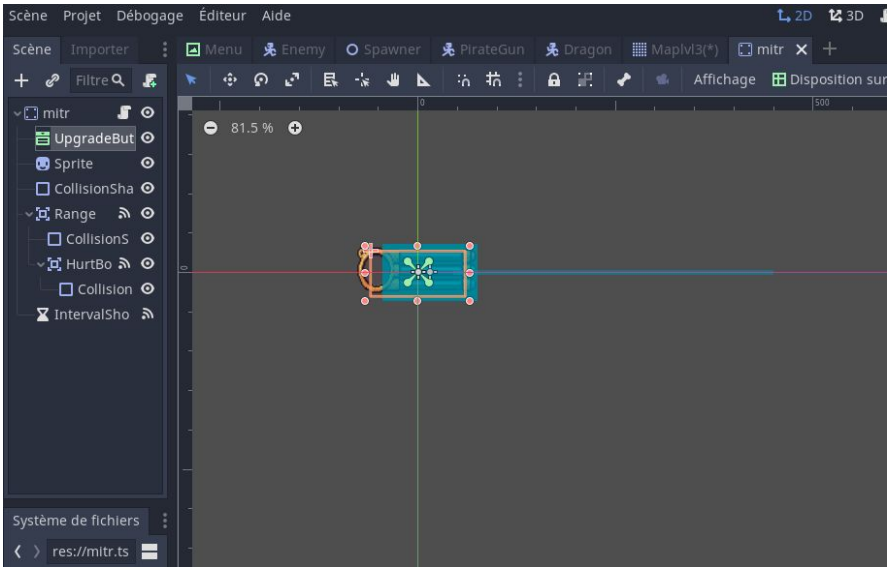


L'objectif des trois niveaux reste le même : survivre.

c) Création des défenses :

La création des nouvelles tourelles se fait sur la base première défense créé c'est à dire le tank. La base pour les tours défensives se compose, du point de vue des noeuds d'un StaticBody2D. Son premier fils est le MenuButton "UpgradeButton" qui permet de réaliser deux actions: améliorer une tour contre de l'argent afin d'augmenter sa puissance de feu ainsi que ses points de vie. Il y a trois niveau possibles, niveau 0,1 et 2. Nous avons fait ce choix par rapport à 1,2 et 3 pour faciliter la recherche en fonction de l'index, qui représente le niveau dans les listes pour les dégâts et les coûts d'amélioration. La seconde action est la destruction de la tour afin que celle-ci puisse être remplacée par une tour différente. La base est ensuite composée d'un Sprite. En effet les tours étant immobiles, ainsi que le choix des assets rend l'animation inutile pour le tank, car le missile donne réellement l'impression de partir du canon. Nous avons songé à ne pas mettre de Sprite et directement ajouter l'image en icône du MenuButton mais nous avons rencontrés certains problèmes de synchronisation entre les animations et les actions pour les défenses qui utilisent un Animated Sprite et pour garder une base commune ainsi que la possibilité de changer au dernier moment par un Animated Sprite qui offre plus d'options.

Ensuite, nous avons créé deux Area2D qui possèdent un enfant CollisionShape2D. Ces deux aires, à l'instar de celles des ennemis permettent de de détecter la portée de la défense, c'est l'aire Range. La deuxième est la Hurtbox, qui sert à subir les dégâts. En revanche pour les défenses nous n'avons pas fait de Hitbox, l'aire qui permet d'infliger les dégâts. En effet les défenses infligent leurs dégâts via les missiles qui eux transmettront les dégâts aux ennemis, nous verrons de quelle manière ultérieurement. Enfin, nous avons ajouté un Timer nommé IntervalShoot, qui comme son nom l'indique, déclenche les tirs à intervalle 0régulier.



Au niveau du script et des méthodes des défenses, celui-ci est plus riche que les attaquants.Initialement, nous avions stocké les dégâts des attaquants directement dans le script des défenses. En effet ceux-ci sont constants du fait que les ennemis sont pré-définis. Nous avons donc créé une liste d'entiers qui stocke les dégâts des ennemis et en fonction de l'ennemi qui attaque, nous déclenchons la méthode GetDamage, qui comme pour les ennemis et la méthode GetHurt, est celle qui inflige les dégâts à la défense, avec l'index associé pour infliger les bons dégats. Cependant afin de simplifier le code nous l'avons déplacé dans le script de la scène Tower Contain, que nous verrons plus tard, du fait que cette scène sert de parents aux défenses dans le jeu. Ensuite, nous avons créé une liste damage qui stocke les dégâts des tours en fonction du niveau de celle-ci ainsi qu'une liste des prix de ameliorations des tours.
Dans la fonction Ready de la tour, nous avons tout d'abord légèrement modifié sa position pour que visuellement celle-ci apparaisse au milieu du bouton d'invocation des tours en non pas en dessous ou au dessus. Nous avons ensuite initialisé le ToolTipe, qui permet d'afficher un message lorsque l'on maintient la souris dessus afin d'afficher le niveau actuel de la tour (donc à sa création). Enfin nous avons initialisé la plupart des variables et connecté un signal au parent de son parent, qui

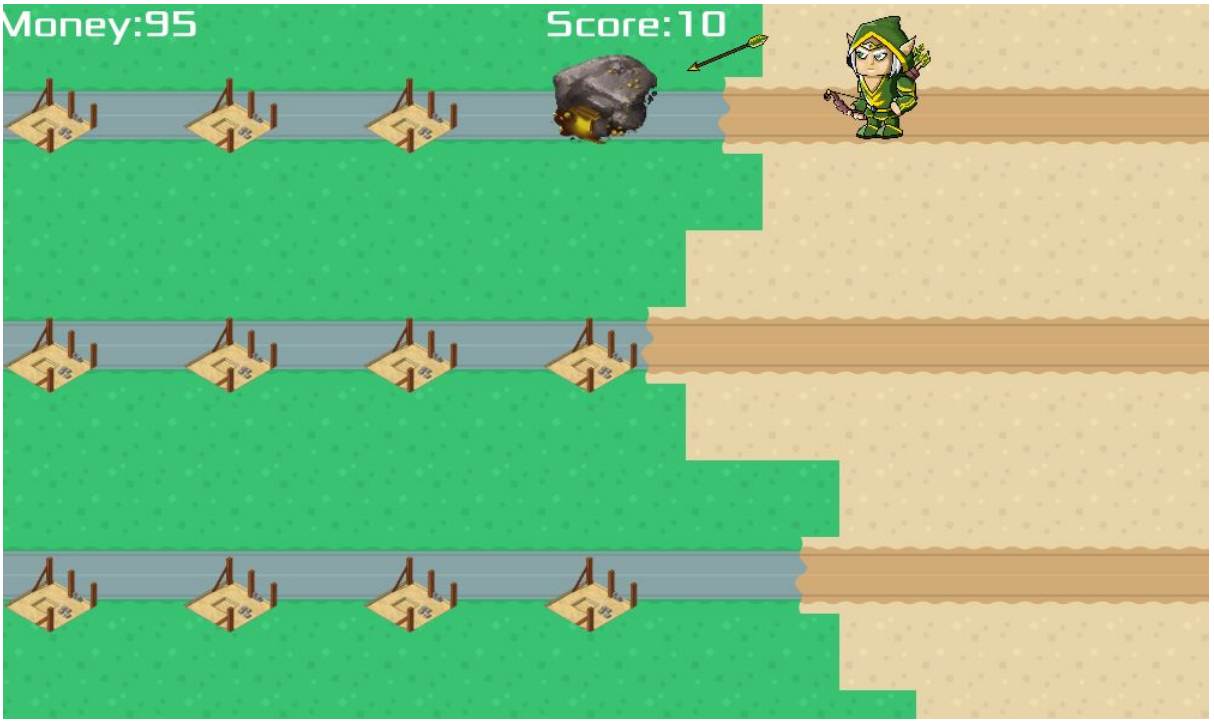
représente la scène map, afin de pouvoir enlever l'argent lors de l'amélioration des tours.
Ensuite, nous avons la méthode Getdamage qui enlève les points de vie à la tour et déclenche la méthodes TowerDeath en cas de mort de la tour afin de la faire disparaître et de faire réapparaitre le bouton où la tour était et donc de pouvoir reconstruire une nouvelle tour. Nous avons rencontré des problèmes pour réaliser cette tâche car nous ne savions dans un premier temps pas comment savoir de quel bouton provenait la tour venant d'être détruite. Nous avons donc ajouté une variable string qui retient le nom du bouton duquel provient la tour. Ensuite nous avons récupéré le bouton en utilisant les méthodes GetParent et GetNode de Godot. Nous pouvons donc réactiver le bouton et le rendre visible de nouveau à la destruction d'une tour.
Pour la détection d'ennemis et le déclenchement du tir, nous avons tout d'abord une variable qui compte le nombre d'ennemis présents dans l'aire de portée. Cette variable est donc initialisé à 0 et avec l'aire Range, dès qu'un ennemi entre dans cette zone, nous l'augmentons de 1, et lorsqu'un ennemi, normalement suite à sa mort, nous réduisons cette variable de 1. Pour la détection nous utilisons donc les méthodes de Godot. Lorsque le nombre d'ennemis est de 0 et qu'un ennemi entre à un, nous déclenchons le Timer IntervallShoot, puis on incrémente le nombre d'ennemis. Si le nombre d'ennemis était déjà supérieur à 0, nous incrémentons seulement car cela signifie que le timer était déjà activé. La fonction lorsqu'un corps sort fait exactement l'inverse, elle arrête donc le Timer seulement si le nombre d'ennemi tombe à 0.
C'est donc le Timer ainsi que sa méthode Timeout, qui se déclenche chaque fois que celui-ci arrive à 0. A ce moment, la tour va instancier un missile, l'ajouter en tant qu'enfant et l'envoyer vers les ennemis. Nous verrons plus tard comment fonctionnent les missiles.

Pour la méthode lorsqu'un corps entre dans l'aire Hurtbox, celle-ci est simple, elle regarde quel corps vient d'entrer et s'inflige les dégâts appropriés. Enfin il reste les méthodes permettant l'amélioration d'une tour. La première est déclenchée par un signal lorsqu'on clique sur l'un des deux bouton du menu. Si l'index de l'item pressé est 0, c'est que c'est une amélioration alors la méthode Upgrade est déclenchée. Celle-ci ajoute alors des points de vie à la tour, augmente son niveau de 1, envoie un signal à la map, avec le coût de l'amélioration, met à jour le ToolTipe pour qu'il affiche le bon niveau lorsqu'on maintient la souris sur une tour, puis, si le niveau est égal à 2 désactive l'utilisation de l'item amélioration. Si l'item pressé est destruction, la méthode TowerDeath est lancée. Enfin une fonction process est utilisé et permet de désactiver le bouton d'amélioration lorsque l'argent n'est pas suffisants pour réaliser une amélioration.

Après avoir recréé la scène pour la tour et celle de son missile avec les assets correspondant, il faut adapter les caractéristiques à ce qu'on souhaite mettre. Par exemple pour la mitraillette : il faut modifier ses dégâts et sa cadence de tir.
Pour la cadence de tir il faut modifier le timer IntervalShoot.

Pour créer de nouvelles unités, il suffit d'adapter les Sprite en trouvant des assets adaptés à nos différents type de défenses pour obtenir des graphismes différents, puis de faire varier leurs statistiques. Par exemple pour la mitraillette, nous avons réduit les dégâts des balles mais augmenté sa cadence de tir. Cependant sa portée est plus courte que celle du Tank. C'est donc une arme qui peut être dévastatrice sur les ennemis proches mais leur laisse la possibilité d'approcher. Elle possède également moins de vie. Il est simple de faire varier ces statistiques, il suffit seulement de changer les variables stockées dans le script ou de faire varier la durée du Timer IntervalShoot pour la fréquence de tir.
Pour les dégâts il faut rajouter le nombre de dégât que fait chaque missiles dans la liste des différent dégâts que font les tourelles. Pour la mitraillette ils sont plus faible car on considère logiquement qu'une balle de mitraillette fait moins mal qu'un tir d'un tank.
Pour le sniper, nous avons allongé la distance à partir de laquel il peut tirer car on considère que l'avantage de cette unité est sa portée. En contrepartie sa cadence de tir est réduite pour ainsi simuler le rechargement entre chaque balles.
Afin d'allonger la portée de cette troupe il suffit d'agrandire horizontalement le CollisionShape 2D.
Il reste donc deux défenses, qui ont été réalisées différemment du fait que contrairement aux autres, elles n'attaquent pas les ennemis. Il y a tout d'abord la mine d'or, qui est l'une des défenses les plus importante étant donné qu'elle permet de rapporter de l'argent, qui est la ressource la plus importante du jeu. Pour la réaliser, nous avons juste repris la scène des Tanks, et retiré l'Area2D "Range" qui sert de portée. En effet celle-ci n'attaquant pas, elle n'a aucun intérêt à repérer les ennemis. Nous avons également renommé le Timer en IntervalGain car la mine ne tir pas mais rapporte à une certaine cadence de l'or. Au niveau de son script il change également très peu. Nous avons juste ajouté un signal relié à la scène Map qui ajoute de l'argent. Nous avons cependant rencontré des difficultés pour synchroniser l'animation de la mine d'or avec la gain d'argent. En effet, la mine d'or possède deux images différentes. L'une où la mine est éteinte et l'autre éclairée. Nous voulions que le gain d'argent se fasse au moment où la mine s'éteint pour que cela représente l'or rapporté pendant le forage. Nous avons donc désactivé son animation par défaut, et au moment de sa construction, nous la démarrons pour qu'elle soit bien synchronisé.

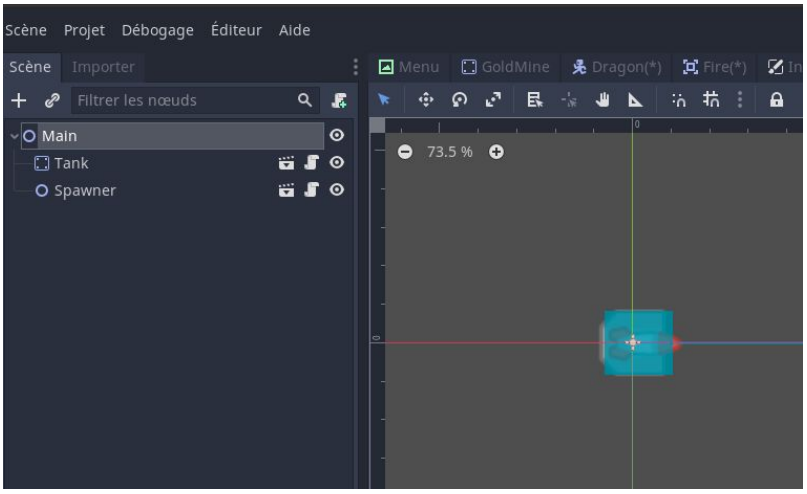
Enfin la barricade est une défense servant à ralentir les ennemis. Nous avons donc seulement repris la base. La barricade n'attaque pas et n'inflige pas de dégâts. Elle subit donc seulement pour laisser les tours éliminer les ennemis.



exemple d'un archer tirant sur un mine d'or.

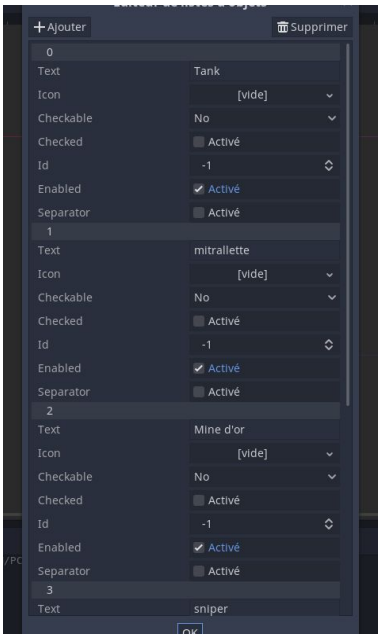
d) Projectiles:

La base des projectiles avait déjà été conçu pour la soutenance précédente cependant nous y avons ajouté quelques améliorations.
En fait nous avons changé la façon dont les ennemi prennent les dégâts. En effet avant les dégât des projectiles était stocké dans une listes placé dans le script de chaque attaquant et ainsi lorsqu'un projectiles était détecté l'ennemi prenait des dégât en fonction de l'index de la liste correspondant au projectile. Cependant nous avons voulu pouvoir améliorer les défense et du coup les dégâts des projectiles respectif.
L'implémentation précédente était alors devenu trop compliqué car il fallait connaître le niveau de chaque projectile et créer des listes en plus en fonction du niveau de celle-ci. Nous avons alors préféré un autre système que celui-ci précédemment expliqué.
En effet maintenant les dégât sont stockés dans le script défense dans un objet non statique. Le projectile récupère alors la valeur dans un objet identique et lorsque le projectiles est détecté par un attaquant une méthode prenant en paramètre les dégât du projectile se lance et soustrait alors de la vie à l'ennemi.
Ainsi nous pouvons améliorer les défense sans problème.



e) Bouton d’invocation des tours:

Le bouton d’invocation des tours est une scène importante de notre jeu. En effet, c’est celle-ci qui permet de faire apparaître les tours là où on le désire ce qui représente la base de notre jeu. Pour créer ce bouton d’invocation, nous avons tout d’abord créé un MenuButton que nous avons intitulé “BuildTowerButton”. Nous y avons alors ajouté une image, pour qu’il soit visible sur la carte. Dans un premier temps, nous y avions seulement ajouté un cercle blanc puis pour des soucis d'esthétisque nous l’avons remplacé par une image de chantier pour garder la cohérence avec les constructions. De plus, nous avons donc ajouté chaque tour possible dans la liste des items. Puis via un signal, nous avons relié l’index de l’item pressé avec une méthode qui déclenchera donc la construction de la bonne tour. Cette méthode est principalement dans le TowerContain. La partie dans le script de BuildTower sert à connecter le signal, le transmettre, désactiver le bouton et transmettre son nom au TowerContain, qui le transmettra alors à la tour construite pour que le bon bouton soit réactivé à la destruction.



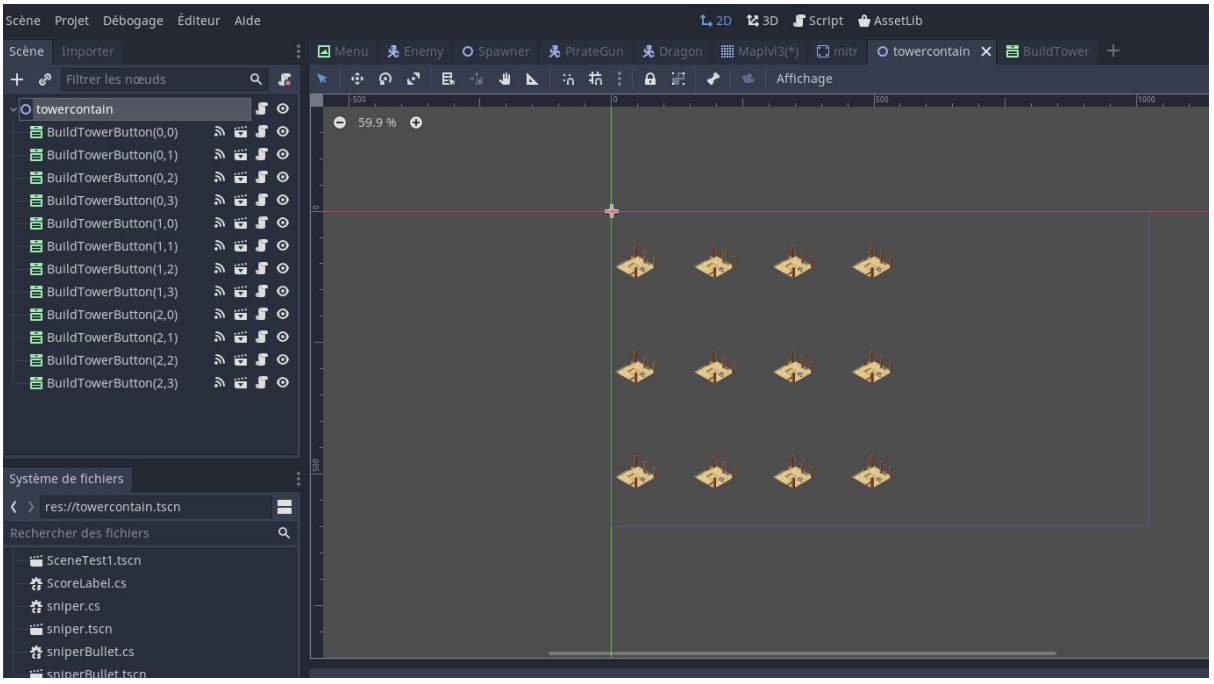
f) Spawner:

Cette classe sert à invoquer des ennemis, le but est d'instancier la scène de l'ennemi désiré.
Pour ce faire nous avons chargé les différentes scènes dans le constructeur Ready.
Nous avons de même créé une énumération de tous les ennemis et une file.
Le procédé est le suivant : une méthode sert à associer chaque type de l'énumération à la scène correspondante. A l'instar d''une méthode qui prend en paramètre le nombre d'ennemi et la scène et qui enfile un nombre de fois la scène désiré.
Tout cela sert dans une tiers méthode pour enfiler les différents nombres et types d'ennemis selon le niveau choisi.
Enfin, il y une méthode reliée à un timer qui instancie à chaque intervalle de temps donné le dernier élément de la file.
De plus, c'est ici qu'est décidé le chemin sur lesquelles les ennemis apparaissent à l'aide d'une fonction qui choisit aléatoirement une ordonné qui est ensuite la valeur avec une constante a l'abscisse de la position de l'objet instancié.

g) TowerContain:

Cet classe à la même utilité que la classe "Spawner" pour les défenses mais pas exactement.
Elle sert à faire apparaître des défenses au niveau des boutons(buttons). Pour ce faire il y a une méthode principale qui est en fait reliée à un signal provenant des boutons(buttons).

En fait, cette méthode prend en compte l'identité de la tour qui est en fait un entier et selon cet entier la scène contenant la défense voulu est instanciée. On donne à la scène instanciée la position du bouton grâce une variable statique qui sert d'intermédiaire et le nom du bouton que l'on transmet à la défense instanciée par le même procédé. Ainsi, la défense pourra retrouver le bouton d'origine.
Cela nous servira pour réactiver le bouton, désactivé préalablement à l'aide d'une méthode, lorsque que la défense se situant sur le bouton est détruite.
Enfin, il y a une méthode qui permet d'ajouter un nombre de bouton donné en paramètre et les lignes choisies aussi.
Cette méthode est lancé à la création du plateau de jeu.



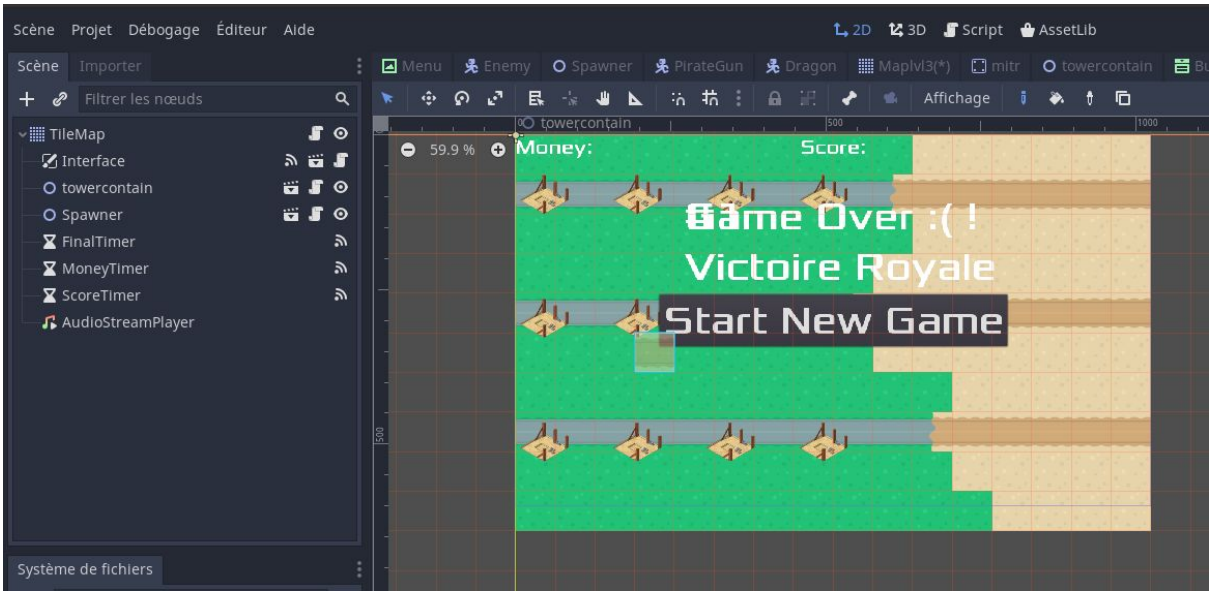
h) Interface:

L'interface du jeu est également un élément important. En effet, c'est ici que l'on va afficher les labels, qui représente le score, l'argent et tous les messages comme ceux de fin de partie. C'est de là qu'on va démarrer une partie. Cette scène est constituée d'un noeud CanvasLayer, qui possède 5 noeuds enfants, quatres Labels qui affichent respectivement l'argent du joueur, le score du joueur, le message de victoire et le message de défaite. Son dernier fils est un Bouton pour lancer la partie.
Son script est relativement court. En effet, on y retrouve seulement des méthodes qui permettent d'actualiser le score et l'argent durant la partie, méthodes qui seront appelées dans le script de Map. Ces méthodes transforment l'argent et le score en chaîne de caractère puis l'affichent. La dernière fonction émet le signal vers la scène

Map pour lancer une partie lorsque nous cliquons sur le bouton de lancement de partie.

i) Map:

La scène “Map” sert de scène principale du jeu malgré le fait que ce ne soit pas la première scène lancée. En effet, nous lançons d’abord le menu du jeu mais c’est bien la scène “Map” la principale. Il y a en réalité trois scène “Map”, une pour chaque niveau mais le script est identique, et seule le noeud racine de la scène change: le TileMap qui sert à dessiner la carte du jeu.
Le scène comporte donc comme dit ci-dessus un noeud TileMap.
Nous avons ensuite instancié la scène Interface, la scène Spawner et la scène TowerContain. De plus, nous avons ajouté une musique de fond pour rendre le jeu plus complet et agréable à jouer. Enfin, la scène Map possède trois Timer, le “MoneyTimer”, le “ScoreTimer” et le “FinalTimer”. Les deux premiers tournent en permanence et permettent de respectivement gagner de l’argent et augmenter son score. Le dernier quant à lui ne se lance qu’à la fin de la partie pour laisser le message de fin affiché suffisamment longtemps avant de retourner au menu principal.



Au niveau du script de cette scène, tout d'abord dans la fonction ready, nous récupérons la valeur du niveau, qui est définie dans le Menu, celui où nous choisissons le niveau ainsi que le nombre de vies en fonction du niveau. Ensuite, la fonction _newgame, qui comme son nom l'indique permet de lancer la partie. Celle-ci est activée lorsque nous cliquons sur le bouton de l'interface, via un signal. La méthode va alors exécuter de nombreuses actions pour lancer le jeu. Elle va tout d'abord définir le score et l'argent à leurs valeurs de départ, pour le Score et pour l'argent.

Ensuite elle va déclencher le démarrage des Timer "ScoreTimer" et "MoneyTimer", qui feront respectivement augmenter le score d'une unité chaque seconde et l'argent de 10 unités toutes les 5 secondes. La méthode va également, en fonction du niveau de jeu choisi ajouter des lignes de boutons d'invocation de tour afin de pouvoir jouer sur le bon nombre de lignes.

Elle va ensuite appeler la méthode de Spawner, Level qui va alors en fonction du niveau préparer les vagues d'ennemis à venir, puis lancer le Timer d'invocation des ennemis qui se trouve dans la scène spawner. Enfin, le score et l'argent vont être actualisés via les méthodes UptadeScore et UpdateMoney présentes dans la scène Interface et ainsi bien afficher les valeurs courante à l'écran. Il reste donc les méthodes permettant de mettre fin au jeu Il y a tout d'abord la méthode _on_ennemy_passed qui chaque fois qu'un ennemi traverse entièrement la carte, celui-ci envoie un signal à la scène Map pour la déclencher. Elle va alors enlever une vie au compteur, et si le nombre de vies tombe à 0, alors va être déclenché la fin du jeu. Le Timer de spawner s'arrête pour que les ennemis ne viennent plus, le score tombe à 0 puis les timers ScoreTimer et MoneyTimer s'arrêtent. Enfin le message GameOver de l'interface apparait et le FinalTimer est lancé. Le paramètre OneShot du Timer est activé, il va donc être activé une seule fois. Lorsque celui-ci arrive à 0, la scène Map se ferme et le menu principal revient. En réalité, ce timer sert uniquement à laisser le message affiché suffisamment longtemps et non pas disparaître immédiatement. En revanche, pour détecter une victoire du joueur, ceci est plus compliqué, pour ce faire, nous avons utilisé une fonction Process. Celle-ci déclenche la méthode Win si trois conditions sont remplies. Tout d'abord il faut que les vagues d'ennemis aient bien été ajouté à la file qui fait ensuite apparaître les ennemis. Nous avons donc ajouté un premier booléen dans Spawner pour s'en assurer. Lorsque ce booléen est vrai et que la file est vide alors le booléen finish, également dans Spawner prend la valeur true. La deuxième condition est que le nombre d'enfants de Spawner doit être inferieur à 2, ce qui signifie que tous les ennemis sur la carte sont morts. Enfin le dernier, flag sert seulement à lancer une unique fois la méthode Win. Celle-ci est similaire à celle de la défaite. Elle se contente seulement d'arrêter les timer de score et d'argent, d'afficher le message de victoire et de déclencher le Timer qui retournera alors au menu principal.

Une partie en cours.

j) Menus principaux:

Lorsqu'on lance un jeu, il est rare de rentrer directement dans une partie sans passer par un menu principal. Nous avons donc créé également un menu pricipal, qui nous redirige ensuite vers le menu des niveaux qui enfin nous renvoie en partie. Comme vous pouvez le remarquer, le menu n'a fait que changer à chaque soutenance, il ne nous plaisait pas assez. Cependant, celui-ci est bien notre menu définitif et rassemble toutes les qualités que nous recherchons pour notre menu : il est en rapport avec le thème de notre jeu (on aperçoit deux unités qui sont des sorciers). De plus, les boutons start sont en fait les projectiles d'une défense bien spécifique : le Tank, défense la plus équilibrée du jeu.
Nous allons donc vous expliquer plus en détail la conception de notre menu:
Dans un premier temps, nous avons un noeud TextureRect qui constitue le fond de notre menu donc ici le paysage que l'on voit. En noeud enfant, nous avons tout d'abord créé deux autres noeud TextureRect qui vont chacun représenter un des deux sorciers du menu. Ensuite, nous avons créée un noeud label qui va nous permettre d'afficher le nom de notre groupe en bas à gauche : JTRD Studios. De plus, nous avons créé un noeud textureButton représenté ici par la fusée qui nous permettra dans un second temps d'arriver au menu du choix des niveaux. Enfin, l'ultime et dernier noeud qui arbore fièrement le nom de notre jeu The Last Survivor est aussi un noeud label. Pour rendre le menu plus vivant comme nous le voulions lors des dernières soutenances, nous avons ajouté des animations qui font descendre le titre de notre jeu et monter la fusée afin de les apercevoir sur le menu.

Et la touche finale, un noeud AudioStreamPlayer qui nous permet d'insérer de la musique dans nos menus.

Passons maintenant au second menu de notre jeu, on y aperçoit trois fusées avec chacunes un niveau. Pour rendre ce menu un peu plus sympathique, nous avons décidé de modéliser grâce à Photoshop ces fusées et ainsi de les personnaliser: en effet, si on fait attention aux détails, la fusée du niveau 1 ne compte qu'une flamme tandis que la fusée du niveau deux en a deux et celle du niveau trois en présente trois. Grâce au code qui nous permet de passer d'une scène à l'autre, ces fusées vont donc diriger le joueur vers les niveaux respectifs. Pour ce menu, nous avons utilisés un noeud TextureRect pour le fond ainsi que trois noeud TextureButton pour les trois niveaux.

Vous le remarquerez certainement pendant le déroulement du jeu, chaque niveau a sa propre musique. Nous avons fait exprès d'utiliser des musiques motivantes, qui captent l'attention et plongent entièrement le joueur dans l'univers de The Last Survivor.

k) Passage d'une scène à l'autre:

Pour passer d'une scène à l'autre, nous avons créé un script à part, qui nous permet de le réaliser. C'est le script global, que nous avons ensuite Autoload pour pouvoir le charger même sans qu'il soit rattaché à une scène. Celui-ci comporte deux méthodes. Une première, qui prend en paramètre le chemin de la scène que l'on souhaite ouvrir. Cette méthode va appeler la suivante avec le même paramètre mais en faisant un appel retardé, pour que le script de la scène en cours ait le temps de s'exécuter intégralement. La deuxième méthode elle va réellement changer la scène active en fermant celle actuelle, puis en ouvrant celle attendue.

l) Débogage :

Une des tâches les plus laborieuses et que l'on a sous-estimé fut le débogage.
En effet nous ne nous attendions pas à découvrir de nombreux bogue au fur et à mesure des tests.
Il fut ensuite compliqué de revenir sur le code pour trouver l'origine de l'erreur.
Nous avons par la suite testé individuellement le comportement de chaque objet et observé si leur comportement fut bien celui attendu. Notamment le dragon qui avant débogage crachait des flammes même si une tour était détruite ou quand il jouait l'animation mort.
Mais aussi les défenses dont la cadence de tir était en "autostart" c'est à dire qu'elle tirait après un nombre de secondes aléatoire dans l'intervalle de temps donné initialement.
Ce qui posait problème pour les défenses ayant une cadence de tir lente car avant que le premier projectile soit tiré l'ennemi était déjà sur la tour ce qui implique que la portée de la défense n'était pas du tout au rendez-vous.
Maintenant, après débogage les défenses tirent un projectile dès qu'elle détecte le premier ennemi puis lancent le timer pour simuler la cadence.
De même, certains objets ne faisaient pas du tout ce que nous pensions qu'il allait faire.
Par exemple l'archer ne tirait pas sur la bonne cible et parfois ignorait certaines cible,s il a donc fallu revoir intégralement son système de tir.

m) Site:

C'est certainement l'une des parties de notre projet qui a le plus évolué au cours du temps. En effet, nous l'avons constamment amélioré afin qu'il soit le plus complet possible et qu'il informe au mieux le joueur sur notre jeu. On peut retrouver sur le site les notes des membres de JTRD Studios, la chronologie de réalisation de notre projet, les problèmes rencontrées lors de la réalisation de celui-ci (notamment lié au confinement), l'essentiel pour bien connaître le jeu, vous pouvez d'ailleurs retrouver sur cette page toutes les descriptions des défenses et bâtiments du jeu… En voici par exemple une, elle concerne l'archer : Ces tireurs d'élite préfèrent garder leurs distances, aussi bien sur le champ de bataille que dans la vie. Ils n'aiment rien tant que de voir tomber la cible sur laquelle ils ont jeté leur dévolu. On retrouve aussi sur le site une partie liens divers où sont notamment disponible nos sons. Ensuite, on retrouve une partie archive avec les archives de nos trois rapports de soutenance et enfin une section téléchargement où l'on peut télécharger The Last Survivor. Selon nous, le point fort de notre site est qu'il présente tous les personnages et bâtiments présents dans l'univers de The Last Survivor et le joueur prend alors part à ce monde

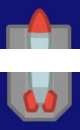
fantastique. Il faut savoir que les photos utilisées sur ce site proviennent uniquement de notre jeu. La création du site étant totalement achevé, il est maintenant disponible sur internet grâce à l'hébergeur gratuit github.

Voici quelques photos de la page d'accueil de notre site.

The Last Survivor

JTRD Studios

Sur cette page, vous trouverez toutes les informations nécessaires concernant le jeu, que ce soit des personnages ou encore des défenses, en passant évidemment par quelques petites astuces si vous ne parvenez pas à passer nos niveaux.
Tout d'abord, voyons quel type de défenses et de batiments nous allons pouvoir utiliser pour venir à boût de nos ennemis:

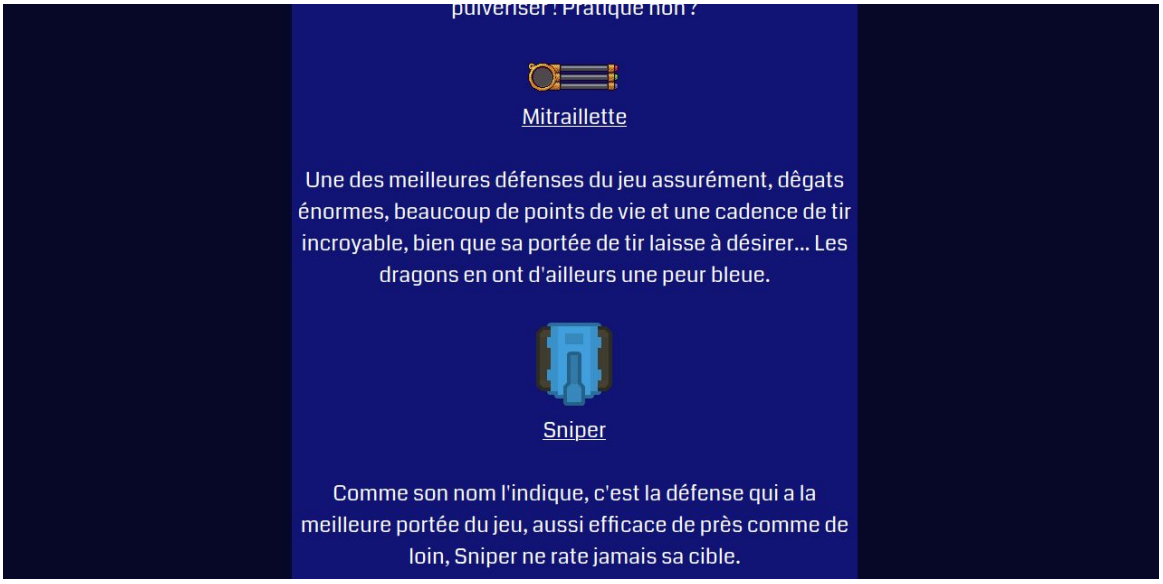


Tank

Voici notre cher Tank, la défense de base du jeu, elle dispose d'une bonne portée, d'une bonne santé(points de vie) et fait pas mal de dêgats. Ce sont ces qualités principales qui font d'elle l'arme la plus équilibrée du jeu.

Barricade

Bien qu'elle ne puisse pas faire de dêgats aux ennemis, c'est une défense redoutable notamment grâce à ses points de vie, pendant qu'elle encaisse les dêgats ennemis, les autres défenses en profitent alors pour les pulvériser ! Pratique non ?



Voici maintenant quelques photos de la page provenant de la partie Présentation du projet: “L’essentiel pour profiter pleinement du jeu”. (page au dessus)
Pour accéder au site, nous mettons son lien à disposition :
https://iammaren.github.io/thelastsurvivor/index.html

Partie 3: Récit de la réalisation:

Nous allons maintenant développer le récit de la réalisation du projet au niveau du ressenti de chacun. Cette partie présentant nos joies et nos peines, il est évident que nous n’avons pas tous eu les mêmes sentiments, et nous avons donc divisé le récit en deux, une partie qui donnera les péripéties du groupe puis une partie individuelle pour pouvoir ainsi partager quelque chose de personnel.

I) Récit de groupe:

1)Joies:

Commençons par la fin: notre plus grosse joie est d’avoir rendu un jeu fonctionnel et plutôt plaisant à jouer en tout cas pour notre part.
Le jeu est visuellement correcte et jouable.
C’est encore plus satisfaisant après avoir travaillé dessus durant des mois et avoir eu beaucoup de mal à le faire.

2)Peines:

Nous avons découvert trop tard que les méthodes et les objets communs au attaquant et au défense pouvait être implémenté dans des classes abstraites dont tous les attaquants et les défenses hériteront . Cela aurait été moins éprouvant et plus propre au niveau du code.
Notre plus gros regret est de ne pas avoir pensé avant à la manière d'implémenter nos objets.
Cela nous servira de leçon pour le prochain jeu.
De plus, les conditions de réalisation de ce projet ont été totalement chamboulées par la pandémie. En effet, le développement du projet a eu une tournure totalement inattendue suite à l'instauration du confinement. Nous avons donc dû totalement revoir notre organisation pour pouvoir continuer à avancer le mieux possible.Nous avons alors rencontré de nombreux problèmes liés à cela. Cependant, la recherche de solutions pour y remédier constitue une grande partie de l'apprentissage de ce projet tant pour développer nos compétences en télétravail et à mis à l'épreuve notre capacité d'adaptation. Premièrement nous avons rencontrés des difficultés au niveau matériel. En effet, nous n'avons plus à disposition les ordinateurs de l'école qui était déjà bien configurés pour faire marcher Godot. Nous avons rencontrés certaines difficultés à le faire marcher sous Windows et nous avons donc eu recours à des machines virtuelles ou des multiboot pour pouvoir continuer de développer sous Linux. De plus, pour la mise en place du travail de groupe nous avons eu recours à l'application Discord qui nous a offert de nombreuses fonctionnalités pour nous permettre d'avancer en contournant les problèmes de présence, notamment les appels de groupes ainsi que le partage d'écran en simultané. Nous pouvons donc conclure que vis à vis du confinement, la difficulté du travail a été légèrement augmenté mais a permis à la richesse de l'apprentissage d'être beaucoup plus vaste.
Enfin, la partie des graphismes, a été plus difficile que prévu. Nous voulions rester dans un thème précis, mais la difficulté d'obtention de graphismes de bonnes qualités, dans un seul et même thème et gratuit s'est révélé impossible. Nous avons donc dû mélanger les thèmes en essayant de garder de la cohérence, le fantastique pour les ennemis et des armes à feux ou véhicule blindé pour les défenses.

II) Récits individuel:

a)DELRIEU Jules:

1)Joies:

Pour ma part ce projet m'a beaucoup apporté notamment pour comprendre les aspects techniques lors de la conception d'un jeu mais aussi l'apprentissage du travail en groupe. En effet, je joue depuis petit aux jeux vidéos sans ne m'être jamais réellement penché sur le code en lui même. Maintenant bien que je n'ai vu qu'une infime partie de ce que représentait la conception d'un gros jeu tel que Grand Theft Auto ou d'autres, je peux comprendre certaines logiques. De plus, afin d'aboutir à ce jeu nous avons dû travailler en groupe et apprendre à échanger nos idées, les informations qu'on recueillait, et donc s'entraider en prenant en compte les points forts et les connaissances déjà acquises de chacun. Même si nous avons déjà eu ce genre d'expérience comme avec le TPE en 1ere, elle reste totalement différente et bien plus compliqué que les autres. La complexité de ce projet est dans l'autonomie qui est requise. Pour un des groupes n'ayant jamais fait d'informatique dans le passé et devoir coder un jeu nous même, cela paraissait très compliqué mais c'est qui la rendu encore plus satisfaisant. Nous étions très enthousiaste au début et à chaque fois que le jeu avançait nous l'étions encore plus. Ce travail de groupe nous donne un aperçu de notre futur travail car le métier d'ingénieur consiste à travailler en groupe afin de résoudre un problème donné(ou plusieurs).

2) Peines :

Personnellement, le plus contraignant dans ce projet ce sont les textures des défenses qui ne correspondent pas vraiment à ce que nous attendions contrairement à l'attaque.
D'autre part j'avais au tout début des problèmes de compilation dû à Godot et de nombreux bogues lors de son utilisation. C'est pourquoi j'ai installé un double boot.
Un des problèmes qui est apparu est un problème de duplication de scène. Pour construire une des tours similaire à une déjà existante nous avons essayé de dupliquer celle déjà existante, or lors de la modification du script les deux scènes étaient modifié. Ce qui a créé de nombreux bogue et nous a contraint à refaire une scène à part entière.
Une des complications rencontré est le manque de documentation. Malgré de nombreuses documentations explicatives et une communauté très collaborative, le nombre de vidéos expliquant certaines fonctionnalités ne sont pas nombreuses. Ce qui est normale car la communauté de Godot reste bien plus petite que par celle de Unity par exemple.

b)DUNAND Tom:

1)Joies:

Ma première joie est tout d'abord d'avoir découvert Godot. En effet, bien que ce moteur de jeu semble difficile au début du fait de la vastité des possibilités, les

nombreux forums ou serveurs Discord rassemblant des utilisateurs plus expérimentés qui permettent de franchir le premier obstacle, la compréhension du logiciel. Il devient ensuite plus simple d'utiliser la documentation du site lorsqu'on a acquis des bases et ainsi le travail de développement de jeu sur Godot devient très agréable et beaucoup plus productif. De plus, il permet de mieux comprendre la difficulté de réaliser un jeu parfait, d'une part sur l'équilibrage du jeu, ou sur la détection de la moindre erreur dans le jeu. En effet, toute personne ayant déjà passé du temps sur un jeu vidéo a pu connaître la frustration de perdre une partie à cause d'un bogue ou d'un item du jeu ou un ennemi trop puissant. Réaliser ce jeu m'a donc fait relativiser sur ce domaine en comprenant mieux le développement d'un jeu. De plus, l'une de mes joies est d'avoir réussi à réaliser la mine d'or du jeu exactement avec l'image que je me faisais mentalement de celle-ci avant le début de sa réalisation. La recherche de l'image adapté ayant été laborieuse, j'ai eu la satisfaction de réussir à en trouver une qui correspondait à mes attentes ainsi que de réussir à la faire fonctionner comme prévu. De plus je suis satisfait de la progression que j'ai fait sur Godot durant ce projet. En effet, au début du projet, la réalisation d'une défense par exemple me paraissait irréalisable et bien que celles-ci peuvent encore être améliorées au niveau du code, il n'en reste pas moins que nous en avons réalisés de nombreuses. Enfin, je suis satisfait que nous ayons pu accomplir tout nos objectifs notamment les améliorations des tours, à l'exception du son des personnages, dont nous avons fait le choix de retirer pour deux raisons, la difficulté d'en obtenir ou en réaliser de bonnes qualités, et la présence d'une musique de fond qui aurait rendu l'audio du jeu désagréable en raison du trop grand nombre de sons.

2)Peines:

Bien que dans l'ensemble, il y ait eu beaucoup plus de joies que de peines, il n'en reste pas moins de nombreux éléments qui nous ont retardé. Le premier est le matériel. N'ayant pas accès aux ordinateurs de l'école j'ai donc dû utiliser mon ordinateur personnel qui malheureusement manquait de puissance à certains moments et rendait l'avancé du projet difficile. De plus, l'apprentissage des bases de Godot a été éprouvant, bien qu'à long terme cela rentre dans les joies du projet. Le travail de groupe a perdu une part de son efficacité avec la distance bien que cela fasse partie de l'apprentissage, des conditions plus classiques auraient été préférables. Enfin, certaines fausses manipulations avec git nous ont fait recommencer certaines parties du projet, principalement dans les premières semaines, mais ce sont ces erreurs qui m'ont le plus appris.

c)IAMMAREN Driss:

1)Peines:

Nous avons évidemment rencontré de nombreuses difficultés, qui ont parfois freinées l'avancée du jeu. Je pense par exemple à mes premières semaines sur Godot ainsi que la première version de notre menu que j'ai perdu juste avant la première soutenance en faisant de fausses manipulations de sauvegarde sur Godot et sur Git. Ou encore le confinement qui a totalement bouleversé notre manière de s'organiser, de travailler et de penser. J'ai aussi parfois douté quant à la version finale de notre jeu, je me posais des questions : aller-t-il être jouable ? Complet ? Réussi ? Au début de l'année, je ne réalisais pas l'importance de ce projet, ses difficultés ainsi que l'énorme temps de travail nécessaire pour des résultats parfois décevants. Un simple jeu en 2D, et plus particulièrement un petit Tower Defense de quelques niveaux prend beaucoup de temps à développer et ce même avec un groupe de quatre personnes. Je comprends maintenant pourquoi les grandes entreprises de jeux vidéos mettent autant de temps à sortir des jeux vidéos aux graphismes quasiment parfait. Cependant, c'est en affrontant les échecs que j'ai finalement appris le plus.

2)Joies:

C'est alors là que le travail de groupe a prit tout son sens et a permis de surmonter n'importe quelles difficultés, et rien n'est plus satisfaisant au monde que de créer quelque chose qui nous appartient, qui nous est propre. J'ai alors pris énormément de plaisir avec mes camarades. L'entraide a certainement été un de nos points forts, et refaire ensemble une version d'un des jeux les plus populaires sur mobile et qui a bercé notre enfance m'emplit de joie. Cette expérience a été plus que pertinente, et je rêverai de continuer à participer à l'élaboration de ce jeu pour le rendre toujours plus complet et passionnant à jouer : l'élaboration de nouveaux niveaux, de nouvelles cartes, de nouveaux ennemis, de nouveaux modes de jeu, de nouvelles interfaces ainsi que de mises à jours régulières m'excite particulièrement et rend notre projet unique, car la seule limite à celui-ci est notre imagination. Cette expérience m'a vraiment donné envie de coder à nouveau, et plus généralement de créer, de comprendre, de rechercher et d'innover.

d)NAKUSI Romain:

1)Joies:

Nous avons pu trouver différents assets de textures adapté au thème. De plus, certaines troupes possèdent des caractéristiques spéciales qui me semble plutôt réussie notamment l'archer qui tire des flèches en courbe en fonction de la distance avec la défense et tire toujours la bonne cible et la cible la plus proche. Je suis plutôt

satisfait de la manière selon laquelle la trajectoire de la flèche et le choix de la cible ont été implémentés. Après de nombreux tests, de nombreux échecs et de nombreux débogage je ne perçois plus de bogue concernant cette troupe.
De même, il y a le dragon qui crache une flamme qui me semble réussi pendant un intervalle de temps donné.
Cette troupe présente des méthodes en plus par rapport à une troupe basique notamment une animation supplémentaire quand il se prend des dégâts ou encore une méthode qui arrange la forme du dragon en fonction de l'image et la position de celui-ci par quand le dragon se relève.
De plus, lorsque celui-ci attaque, il crache des flammes ,construites avec un noeud "Particles2D" longuement paramétré, pendant une période de temps donné.
Il peut même toucher deux défenses en même temps si celle-ci sont rapprochées et que le dragon est assez avancé.
Le dragon est la troupe la plus aboutie et l'archer la plus complexe à réaliser.
Cependant même si cela ne paraît pas grand chose, c'est un plaisir de réaliser ce genre d'objet.
Ma joie la plus récente fut la réalisation de la tour de sorcier notamment de la boule de feu qui fut particulièrement compliqué même si elle ne présente que très peu de code.
Nous avons ajouté quelques fonctionnalités supplémentaires à celle-ci.
En effet, si une boule de feu est lancée et n'atteint pas sa cible elle s'éteint et dans le cas contraire explose.
Enfin, les ennemis reculent à son contact sauf le dragon qui lui ne recule devant rien.

2)Peines:

Même si cela à été dit au préalable ma plus grande peine est dans la manière d'organiser les script des différents objets. Nous nous sommes rendu compte que trop tard que chaque type d'objet pouvait être regroupés dans une classe abstraite représentant un type d'objet plus large. Cela aurait été plus simple et plus propre au niveau du code.
De plus, pour aimer faire un jeu, prendre du plaisir à le réaliser, il faut au début commencer par apprendre avec la documentation et les tutoriels et y passer du temps. Et malheureusement, je ne m'en suis rendu compte qu'après la première soutenance.
Pour moi le confinement m'a permis de prendre du temps pour ce projet et c'est donc là que j'ai pris énormément de plaisir.

CONCLUSION:

Finalement, nous avons réussi à faire un jeu abouti en atteignant tous nos objectifs malgré de nombreuses difficultés et imprévus tel que ce confinement. Ce jeu nous aura permis de comprendre un grand nombre de mécanisme utilisé dont nous n'avions jamais entendu parler et la complexité de créer ne serait-ce qu'un petit jeu. D'autre part nous avons constaté l'infinité de possibilités que propose un éditeur de jeu tel que Godot.