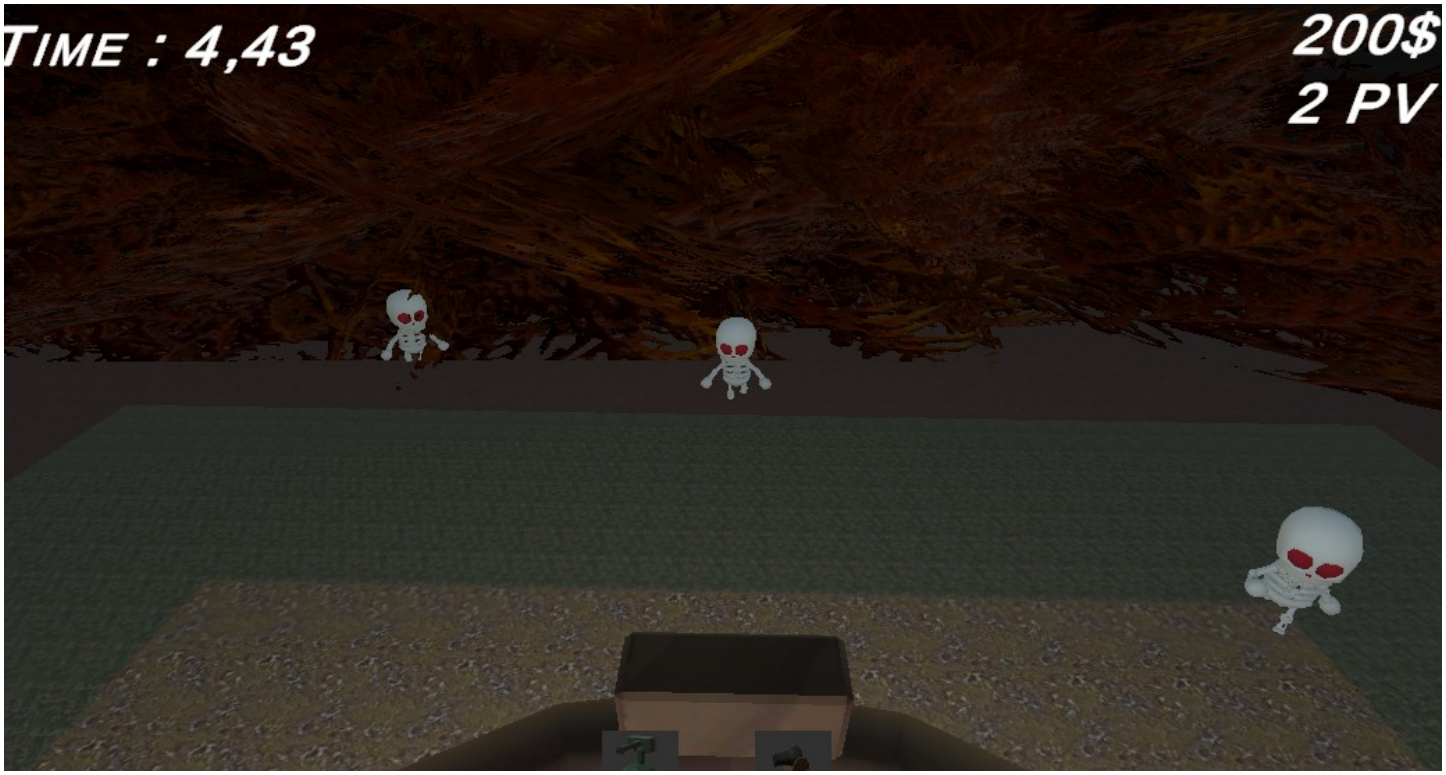
TOTEUF



Dans Toteuf vous êtes au centre de l’attention, alors plus qu’une seule chose à faire vous défendre !
Mais contre qui ? Des armées d’ennemis qui ne veulent qu’une chose vous faire du mal !
Vous devez par conséquent placer des tourelles à des endroits stratégiques et prédéfinis tout en gérant vos ressources pour avoir l’espoir de peut-être survivre...

Table des matières

I. Présentation.

I.1. Présentation du Contexte

Dans le cadre de ma Licence 3 en informatique, il m'a été demandé de réaliser un projet.
Ayant un petit coup de cœur pour le développement de jeu vidéo, je me suis dit pourquoi pas ?
J'ai donc choisi de consacrer mon projet à la création d'un jeu vidéo.
Ce projet me permettra d'apprendre les aspects techniques du développement de jeux, tout en mettant en pratique mes connaissances.
Sachant que je n'ai jamais eu l'opportunité de concevoir un jeu vidéo auparavant.

I.2. Description Générale du Projet

Le projet consiste à développer un jeu vidéo en 3D.
Il s'agit d'un jeu de type TowerDefense dans lequel le joueur devra se protéger des vagues ennemis voulant qu'une seule chose, en découdre.
Le jeu sera développé en utilisant le moteur de jeu Unity et par conséquent programmé en C#.

I.3. Objectifs et Finalités du Projet

- Développer un jeu vidéo fonctionnel et fun.
- Découvrir un domaine qui m'intéresse vraiment, où je n'ai jamais concrètement réalisé de projet.
- Expérimenter l'utilisation du moteur de jeu Unity ainsi que le C#.

I.4. Importance et Justification du Projet

Comme évoqué précédemment j'ai choisi ce projet pour me permettre d'acquérir des compétences techniques et conceptuelles qui me seront utiles pour une potentielle future carrière dans l'industrie du jeu vidéo ou du développement logiciel en général.
Mais aussi par engouement personnel.

I.5. Définition Précise de ce qui est inclus et exclu du Projet

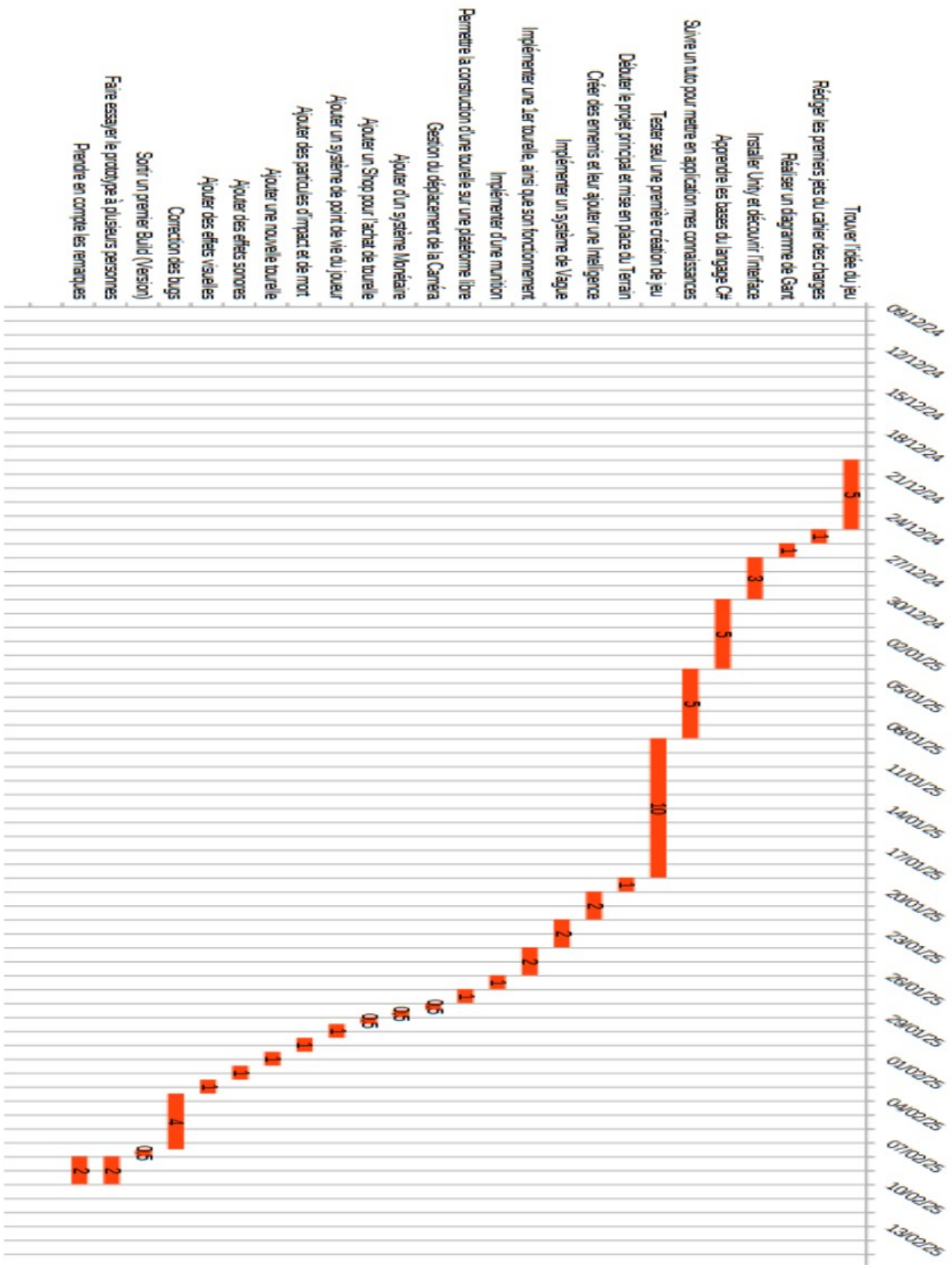
- **Inclus :**
  - La conception et le développement d'un jeu vidéo fonctionnel.
  - L'intégration mécaniques du jeu de base (mouvement, interactions, système de score, etc...).
  - Un univers graphique et sonore minimaliste.
- ***Exclu :***
  - Un jeu expérimental abouti avec un contenu complexe.
  - Un mode multijoueur.
  - Des graphismes poussés.

I.6. Limites et Contraintes du Projet

- ***Contraintes de temps :*** Le projet doit être réalisé dans un délai imparti.
- ***Contraintes de compétences :*** Apprentissages en cours ce qui peut limiter certaines fonctionnalités complexes.

I.7. Phases, Planification et Répartition des Tâches

- **Découpage du projet en différentes phases :**



- **Répartition des tâches :** Étant seul sur le projet, toutes les tâches seront gérées par moi-même.

II. GDD (GDD Game Design Document).

II.1. Concept de Jeu :

- **Nom du jeu :** Toteuf
- **Genre du jeu :** Tower Defense
- **Plateformes :** Dans un 1er temps PC et potentiellement décliner en mobile.
- **Résumé :** Dans Toteuf le joueur doit se défendre contre des vagues d'ennemis en plaçant des tourelles à des endroits stratégiques tout en gérant ses ressources.

II.2. Gameplay et Mécaniques de Jeu :

- **Objectifs du joueur :** Le joueur doit positionner des tours pour se protéger des ennemis de plus en plus coriaces.
- **Contrôles et interface :**
  - Pc : Utilisation principalement de la souris pour sélectionner et placer les tours.
  - « Mobile : Utilisation de l'interface tactile pour sélectionner les tours et les placer. »
- **Mécaniques principales :**
  - Placement stratégique des tours : Le joueur doit placer ses tours sur des emplacements stratégiques pour maximiser son efficacité.
- **Améliorations des tours :** Au début le joueur commence avec une tourelles de niveau simple et ensuite il pourra débloquer des tourelles de niveau plus importantes avec les ressources récoltées en éliminant les ennemis.
- **Gestion des ressources :** Le joueur récolte une monnaie en éliminant des ennemis ou en gagnant des manches.
- **Vagues ennemies :** Les ennemis arrivent par vague successive avec des caractéristiques variées au cours des vagues (vitesse, résistance, capacités spéciales).

II.3. Système de Récompenses et Progression :

- **Récompenses et progression :** Le joueur peut gagner de la monnaie pour débloquer des tourelles plus puissantes, des bonus de vie ou bien de dégât.
- **Progression et niveaux :** Le joueur commence avec des tourelles basique, et débloquer de nouvelles tourelles au fil de sa progression (gain de monnaie).

II.4. Esthétique et Style Visuel :

- **Direction artistique générale :** Le jeu adopte une atmosphère visuelle assez médiévale pour créer une immersion dans le monde de la défense de châteaux comme à l'époque.

- **Design des ennemis et environnements :**
  - Les ennemis sont variés et évoluent au fil des niveaux (squelette, gobelin, etc...).
  - Les tourelles ont des designs distincts pour chaque type (tour d'archet, canon, etc...).

- **Effets visuels et animations :**
  - Des effets spéciaux accompagnent chaque type de tour (explosions, tirs, etc...).
  - Les ennemis vaincus disparaissent dans des éclats.
  - Des transitions de caméra tremblante pour les vagues spéciales.

II.5. Musique et Son :

- **Ambiance sonore :** L'ambiance du jeu comprend des fonds sonores doux pour les moments de préparation et plus dynamique pour les vagues ennemis.

- **Effets sonores :** Utilisation de bruits de tirs et d'ennemis tués, ainsi que des notifications sonores pour les événements importants (début de vague, dégats, etc...).

II.6. Spécifications Techniques :

- **Moteur de jeu :** Le jeu sera développé sur Unity et programmé en C#.

- **Défis techniques :**
  - Création d'une mini IA ennemi : Les ennemis devront suivre des chemins pour atteindre le joueur.
  - Utilisation de Préfab : Utiliser des Préfabs pour permettre une personnalisation flexible au cours du développement.
  - Apprentissage du domaine : N'ayant jamais utilisé Unity ni codé en C# c'est l'occasion d'une nouvelle aventure dans mon apprentissage.

III. Partie personnelle.

III.1. Conception :

III.1.1. Trouver l'idée de jeu (5 jours) :

Tout d'abord, j'avais envisagé de créer un jeu destiné aux enfants, avec des mécaniques qui auraient été à la fois ludiques et éducatives. L'objectif aurait été d'avoir un jeu qui aurait permis d'apprendre des choses tout en s'amusant.
Cependant, j'ai rapidement réalisé que le développement seul, d'un tel jeu m'aurait demandé un temps et des ressources plus importantes que prévu. Je n'aurais pu avoir un projet abouti avec les délais imposés.
Face à cette contrainte, j'ai donc décidé de me recentrer sur un concept de jeu simple, tout autant élaboré mais réalisable dans le temps imparti.
Un jeu de type TowerDefense permettant d'avoir des mécaniques de jeu simples et accessibles pour un premier jeu vidéo.
Malgré ce changement de cap, l'idée initiale du jeu éducatif reste un projet que je garde en tête pour un avenir prochain.

III.1.2. Rédiger les premiers jets du cahier des charges (1 jour) :

Une fois l'idée de jeu trouvée j'ai commencé à rédiger le cahier des charges, à construire les objectifs, les mécaniques, les contraintes du projet.
Pour avoir une vision à court et à moyen terme.

III.1.3. Réaliser le diagramme de Gant (1 jour) :

Le diagramme de Gant m'a permis de planifier les différentes étapes du projet. Même si je suis en autonomie sur ce projet, je trouve primordial de le réaliser, pour avoir une idée visuelle du projet et des contraintes de temps ainsi que les dépendances de certaine tâche.

III.1.4. Installer Unity et découvrir l'interface (3 jours) :

L'installation d'Unity est la première étape technique du projet, j'ai d'abord téléchargé et installé sur ma machine, Unity (qui est un logiciel en open source gratuit).
Une fois l'environnement prêt, j'ai exploré l'interface d'Unity pour me familiariser avec ses composants essentiels comme par exemple :

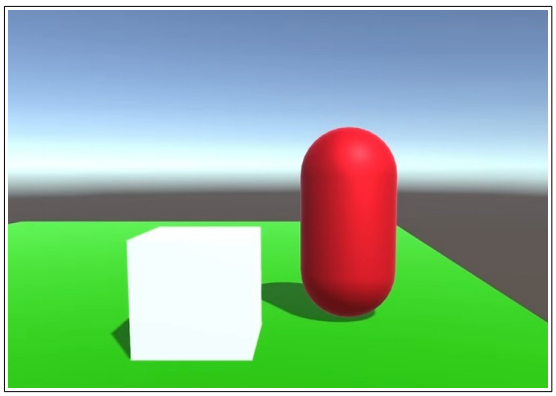
- La scène : Qui est l'espace où les objets du jeu sont placés et manipulés.
- Les composants : Comme le RigidBody qui permet de donner une gravité à un objet.
- L'éditeur de script : Associé l'éditeur de script VisualStudio pour créer et modifier les scripts en C#.
- Et bien d'autres…

III.1.5. Apprendre les bases du langage C# (5 jours) :

N'ayant jamais manipulé ce langage, j'ai alors suivi plusieurs cours et tutoriels qui m'ont permis de comprendre la syntaxe, la structure, les bonnes pratiques de programmation de ce C#.
Cette étape est essentielle pour comprendre non seulement les bons gestes à prendre mais aussi pour éviter les erreurs courantes.
Exemple « GetComponent » qui est une fonction très utile pour récupérer un composant attaché à un objet mais est également une opération très coûteuse en ressources...
J'ai donc compris qu'il est recommandé de l'appeler dans une fonction « Awake() » ou dans les fonctions de « Trigger » pour éviter de l'exécuter avec répétition dans la fonction « Update() » par exemple.

III.1.6. Suivre un tuto pour mettre en application mes connaissances (5 jours) :

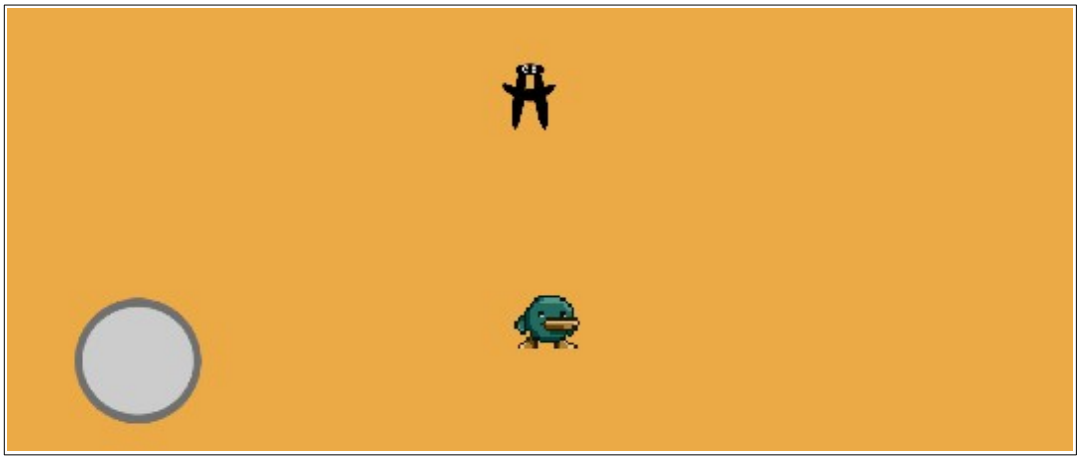
Après avoir compris les bases du langage C#, j'ai décidé de suivre un premier tutoriel sur Unity pour mettre en pratique ce que j'ai appris.
Ce fut un tuto assez simple pour permettre le déplacement d'une capsule avec les touches directionnelles du clavier et gérer la collision :



Capsule qui peut être déplacer avec les touches directionnelles du clavier et pouvoir pousser le cube blanc grâce aux collisions.

III.1.7. Tester seul une première création de jeu (10 jours) :

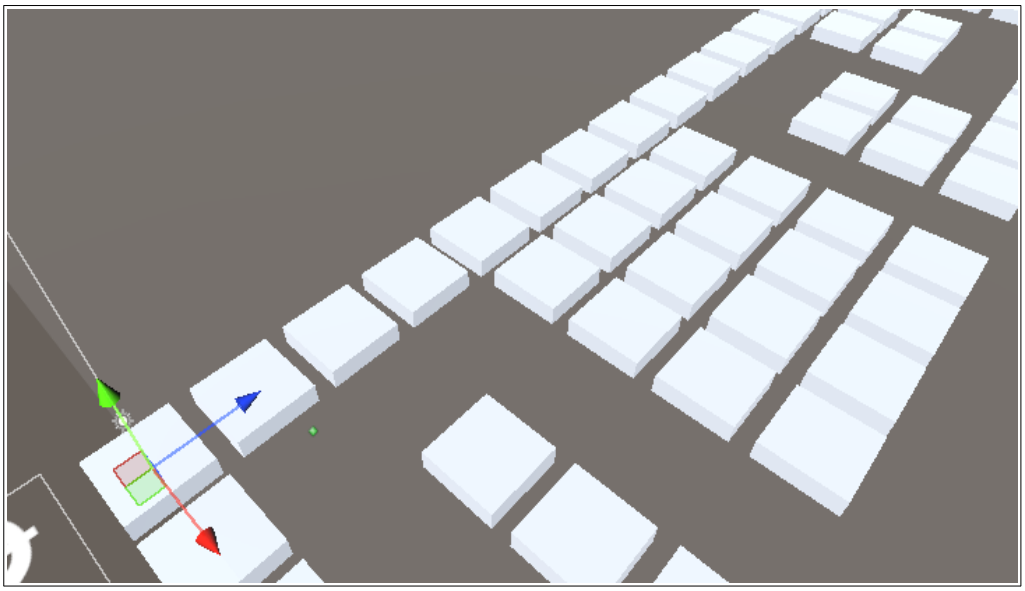
Ensuite, j'ai voulu me lancer en autonomie dans la découverte d'Unity alors j'ai essayé de coder un début de mini-jeu.
Mon objectif était de créer un début de jeu fonctionnel, j'ai donc lancé un projet en 2D…
Tout d'abord j'ai importé un premier personnage dans mon jeu « un Sprite ».
Un Sprite est une image 2D composée de plusieurs images qui permettent lorsqu'on les assemblent de bout en bout de créer une animation de déplacement ou autres (le même principe qu'un dessin animé).
J'ai ensuite décidé de le déplacer avec l'implémentation d'un joystick virtuel sur l'écran.
Enfin, j'ai inclus un deuxième personnage. Celui-ci était programmé pour fuir lorsque je m'approchais de lui avec le premier personnage (Une sorte d'intelligence artificielle fuyeuse).
Cette première tentative m'a permis de bien comprendre le fonctionnement de base d'Unity.



III.1.8. Débuter le projet principal et mise en place du terrain (1 jours) :

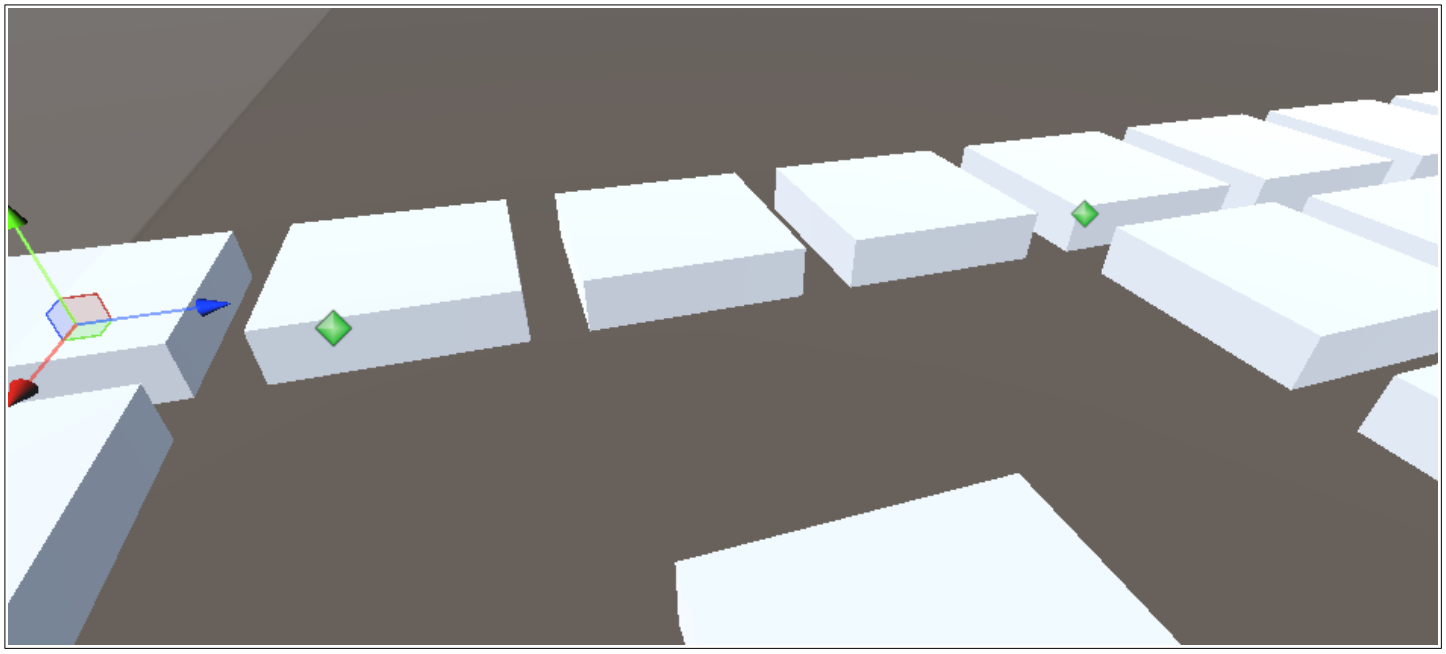
Pour débuter, j'ai crée un nouveau projet Unity en 3D, une fois dans ma scène vide j'ai commencé par concevoir la base qui est le terrain.
Pour gagner du temps et optimiser mon travail, j'ai choisi de me baser sur un ensemble de « Préfab ».
Qu'est-ce qu'un « Préfab » ? Le « Préfab » est une sorte de moule à gâteau qui permet de définir un objet de référence qui pourra être dupliqué X fois tout en restant lié à son modèle d'origine.
Imaginons que je souhaite ajouter 10 objets dans ma scène, par exemple un vase, basé sur le même Préfab et qu'une fois placés je veuille changer la taille de tous les vases car cela ne me convient pas.
Je ne vais pas modifier manuellement la taille de chacun, il me suffit plutôt de modifier le Préfab, grâce à ça, tous les objets qui en sont issus seront automatiquement mis à jour.
Cette utilisation permet d'avoir une meilleure organisation ainsi qu'une efficacité dans le développement et également une forte maniabilité.
J'ai donc commencé la création de mon terrain prototype avec des rectangles comme base de plateforme.
Pour plus tard, leurs assigner une texture ou de les transformer en un autre élément au besoin, sans devoir changer manuellement toutes les plateformes dans la scène.

Le Préfab de la plateforme.


La création du terrain test avec les préfabs de plateformes.

III.1.9. Créer des ennemis et leur ajouter une intelligence (2 jours) :

Après avoir mis en place le terrain, je me suis attaqué aux ennemis, pour ce faire j'ai une nouvelle fois utilisé la méthode des « Préfabs », en effet j'ai dans un premier temps temporairement défini que mon ennemi sera une simple sphère, ce choix permet ainsi de me focaliser sur la mécanique de jeu et non sur le graphisme qui pourra être amélioré ultérieurement.
Pour gérer les déplacements des ennemis j'utilise un système de « Waypoints ».
Un « Waypoints » est un point dans le monde qui sert de repère de coordonnées, j'ai ainsi placé mes Waypoints à chaque tournant du chemin.
Pourquoi à chaque tournant ? Cela me permet de créer des trajectoires que l'ennemi devra suivre, une fois qu'il arrive à un Waypoint je calcule la direction à laquelle est le prochain et déplace l'ennemi dans cette direction jusqu'à ce que ses coordonnées coïncident avec celles du prochain Waypoint et ainsi de suite jusqu'à la fin du parcours.
Les Waypoints sont tous référencés/stockés dans un tableau.



Les Waypoints sont les petits points verts que vous voyez sur l'image ci-dessus. (C'est juste une icône dans le mode éditeur il peut être représenté par n'importe quelle autre icône.)

Le préfab de l'ennemi.

III.1.10. Implémenter un système de vague (2 jours) :

Pour gérer l'apparition progressive des ennemis, j'ai une nouvelle fois fait appel à un Waypoint qui me permet de référencer le point de Spawn (de début).
Ensuite suivant le temps qui passe je fais apparaître un ennemi de plus à chaque nouvelle vague à intervalles réguliers.
J'ai rencontré ici mon premier blocage, en effet comment faire apparaître les ennemis à intervalles réguliers ? D'abord j'ai envisagé d'utiliser une simple boucle « for » ou « while », mais ceci aurait bloqué l'exécution du programme en attendant que chaque ennemi Spawn, et c'est la que j'ai découvert les « coroutines » !
Les coroutines sont des fonctions en C# qui permettent d'exécuter des actions de manière asynchrone sans interrompre le reste du programme, exactement ce qu'il me fallait !
Voici un exemple simple pour faire apparaître et disparaître un objet avec intervalle :

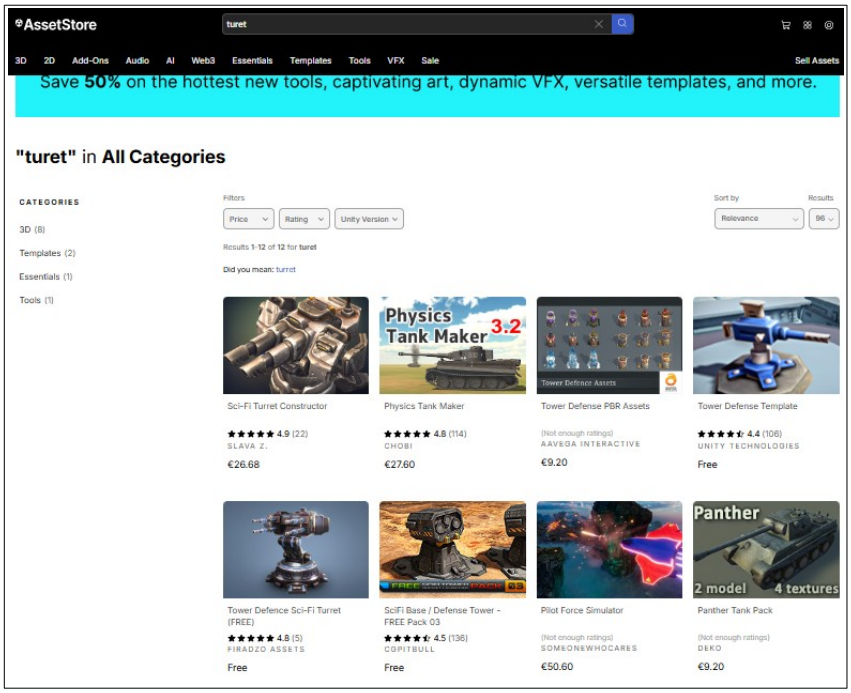
Je définis la fonction en tant que «IEnumerator » et grâce au yield return new je retourne le temps que je souhaite avant de ré-exécuter une nouvelle fois (dans cet exemple 2f soit 2sec).

```
IEnumerator AppaDispa() {
        while (true) {
                objectRenderer.enabled = !objectRenderer.enabled;
                yield return new WaitForSeconds(2f);
        }
}
```

Petite subtilité, la fonction ne peut être appelée comme une fonction normale, je dois utiliser un StartCoroutine(**AppaDispa()**); pour que Unity sache qu'il s'agit d'une coroutine.
Et pour le cas ici d'une boucle infinie dans ma fonction, la condition d'arrêt est StopCoroutine(myCoroutine);.

III.1.11. Implémenter une 1er tourelle, ainsi que son fonctionnement (2 jours) :

Pour cette première tourelle, je suis allé visiter le magasin d'assets d'Unity, c'est un site où l'on peut y trouver énormément de modèles 3D, animations, textures et bien d'autres.



Une fois ma tourelle trouvée je dois l'ajouter à mes assets :

Et pour l'importer rien de plus simple j'ai juste à me rendre dans unity → windows → packageManager puis sélectionner le package de ma tourelle finalement cliquer sur importer.
Nous voici maintenant avec notre tourelle dans notre scène.

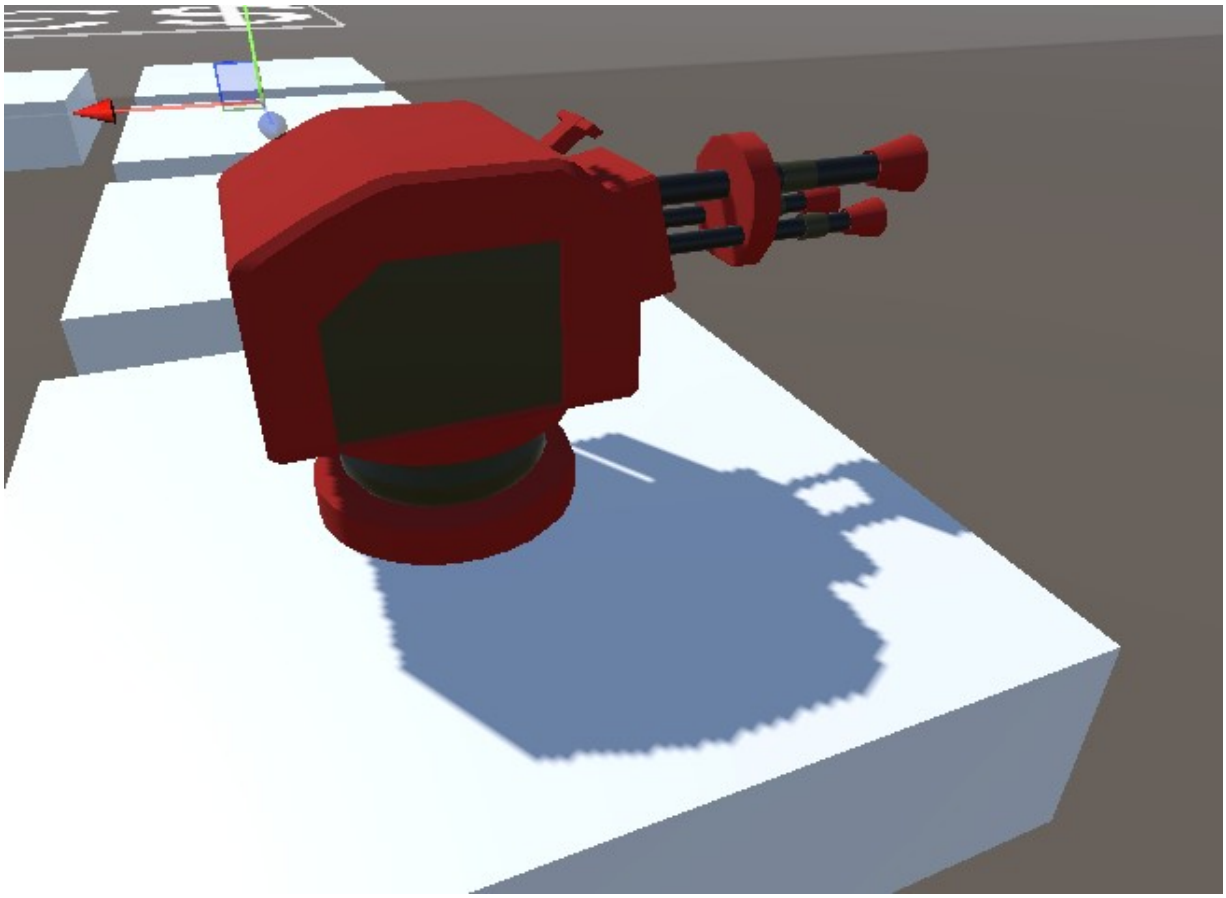
Maintenant vient le moment de coder son comportement.
Mon script pour cette tourelle commence tout d'abord par, définir les variables, par exemple la variable « target » pour pouvoir garder la référence de l'ennemi à cibler, ou bien « cadenceTire » qui comme son nom l'indique définit la cadence à laquelle notre tourelle tire, etc...
Ensuite dans la fonction « Start() » qui est la fonction qu'Unity appelle pour chacun des scripts présent au lancement du jeu, j'utilise une fonction assez similaire à la coroutine de tout à l'heure mais tout aussi géniale : La « InvokeRepeating() » elle permet d'appeler une fonction passée en paramètre tous les X temps.
Je m'en sers ici pour actualiser la cible toutes les 0,5 secondes.
Viens ensuite la fonction « Update() » qui elle est appelée à chaque frame par Unity, à l'intérieur on retrouve une condition qui nous permet de vérifier si la tourelle a une cible en vue, si oui on invoque une munition et on lui assimile la cible, sinon on stop la fonction pour pas aller plus loin avec un « return ».

III.1.12. Implémenter une munition (1 jour) :

Notre tourelle fait spawner une munition, il faut alors créer le paterne de cette munition.
J'utilise une nouvelle fois les Préfabs mais cette fois pour la munition qui sera dans un premier temps une sphère simple.
Au niveau du code, lorsqu'une munition est appelée, elle récupère en paramètre la cible à atteindre (ici un ennemi) et dans la méthode « Update() » on déplace constamment la direction de notre munition vers sa target.
Si elle parvient à la toucher cela appelle une fonction TargetToucher() qui applique des dégâts à l'ennemi, sinon si elle perd son ennemi de vue, elle se détruit.

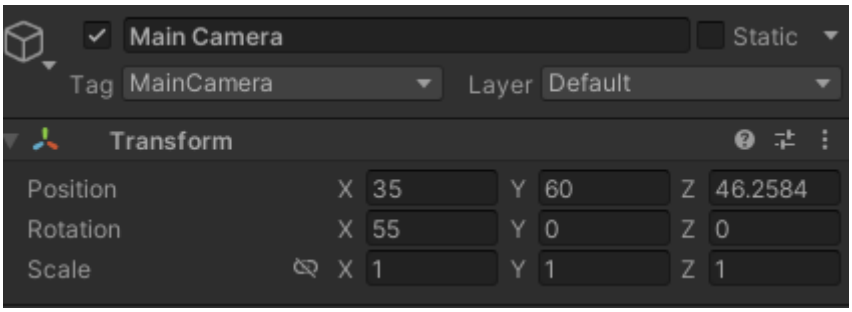
III.1.13. Permettre la construction d'une tourelle sur 1 plateforme libre (1 jour) :

Nous avons à présent des tourelles qui tirent des munitions, mais faut-il pouvoir les placer sur le terrain de jeu.
Pour cela, dans le script de la plateforme, je viens rajouter une variable de type public qui stockera si une tourelle est déjà présente ou non.
Ensuite dans un nouveau script dédié à la construction des tourelles, je vérifie l'emplacement de la souris du joueur à chaque instant du jeu, si elle se situe sur une plateforme elle changera de couleur pour indiquer si oui ou non le joueur peut poser une tourelle.
Si il peut poser sa tourelle et qu'il clique, j'invoque une tourelle aux coordonnées de la plateforme et change l'état de la variable pour signaler qu'une tourelle y est installée.



III.1.14. Gestion du déplacement de la caméra (0,5 jour) :

Plaçons maintenant la caméra, je trouvais qu'avoir une caméra fixe n'était pas la meilleure des solutions, car si on voulait avoir le terrain en entier, la caméra devait être assez éloignée. Sauf que à cette distance on ne voyait pas vraiment bien les éléments du jeu…
Donc création d'un nouveau script pour la caméra qui permettra au joueur de tourner la caméra en dirigeant sa souris sur les bords de son écran et aussi de pouvoir zoomer avec la molette de la souris.
Pour ça c'est assez simple, on vérifie si la souris est sur un bord de l'écran, si oui on inflige une rotation à la caméra dans la direction associée.



III.1.15. Ajouter un système monétaire (0,5 jour) :

Pour l'ajout du système monétaire, je crée un script PlayerStats où je vais venir y stocker sa monnaie dans une variable Static.
Une variable Static est une variable qui est « unique », qu'elle est partagée entre tous et surtout peut être accessible directement sans devoir créer un objet.
Ce qui permet d'avoir la monnaie du joueur de manière globale et d'y accéder facilement depuis d'autres scripts.
Comme par exemple le script de gestion de la construction, vérifie d'abord si le joueur possède assez d'argent pour pouvoir oui ou non, construire une tourelle.
Et le script ennemis, qui lors de la mort de ce dernier, incrémente la monnaie du joueur directement.

```
public static int money;
```

```
public void BuildTurret(Node node){
    if(PlayerStats.money < turretToBuild.cost){
        Debug.Log("Pas assez d'argent pour ceci...");
        return;
    }
    PlayerStats.money -= turretToBuild.cost;
```

III.1.16. Ajouter un Shop pour l'achat de tourelle (1 jour) :

Il faut bien avoir un endroit où choisir la tourelle que l'on souhaite construire. Pour cela, vient la création d'un « UI » qui permet d'afficher l'image et le prix de la tourelle.
Chaque tourelle est associée à un bouton permettant de la sélectionner.

III.1.17. Ajouter un système de point de vie pour le joueur (0,5 jour) :

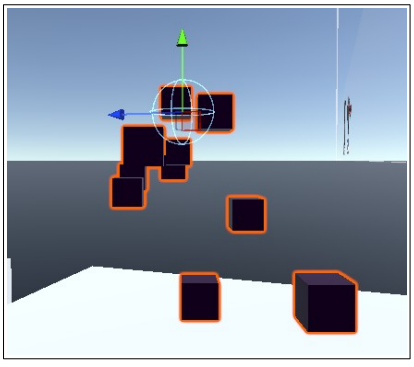
Pour rendre le jeu plus interactif et ajouter du suspense/défi, il faut que le joueur ait quelque chose à perdre. Ici ses points de vie, l'idée est que lorsque l'ennemi atteint la fin du parcours, on supprime l'ennemi et on décrémente la vie du joueur.
Pour stocker la vie du joueur j'utilise le même principe que pour la monnaie.

```
public static int lives;
```

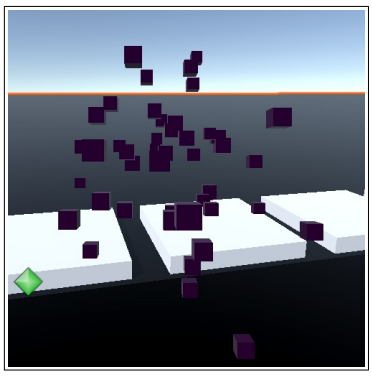
```
private void EndParcours(){
    PlayerStats.lives--;
    Destroy(gameObject);
}
```

III.1.18. Ajouter des particules d'impact et de mort (1 jour) :

Maintenant que le jeu dispose d'un terrain, d'ennemis, de tourelles qui tirent mais aussi un système de vagues et de points de vie, il est fonctionnel. Mais il faut améliorer l'esthétique pour rendre ça un peu plus fun. En effet, actuellement, quand la tourelle élimine un ennemi, il disparaît instantanément (ce qui n'est pas très joli, et peu réaliste), je vais donc améliorer ce dynamisme en ajoutant des particules.
Unity propose un composant permettant l'implémentation de particules, elles se gèrent comme un objet dans la scène et je vais une nouvelle fois ici créer un Préfab pour chacun des effets que je souhaite.
Donc par conséquent, un pour l'effet d'impact et un pour l'effet de mort.



Effet d'impact



Effet de mort

Ils ont tous les deux le même matériel (couleur) que l'ennemi. Ce qui est je le rappelle est un prototype pour ensuite importer de vrai modèle 3D, je travaille sur l'aspect fonctionnel dans un premier temps.
Ensuite pour les implémenter je rajoute dans le script munition que lors de l'impact je fasse apparaître l'effet d'impact et le détruit X secondes après.

```
private void TargetToucher(){
    GameObject effet = Instantiate(impactEffet, transform.position, transform.rotation);
    Destroy(effet, 5f);
```

De même dans le script ennemi pour la mort de ce dernier.

```
GameObject EffectDie = (GameObject)Instantiate(dieEffectPrefab, transform.position,
Destroy(EffectDie, 2f);
```

III.1.19. Ajouter une nouvelle tourelle (1 jour) :

Grâce à l'utilisation des préfabs, l'ajout d'une nouvelle tourelle est une tâche relativement simple, car au lieu de récréer tout depuis le début pour en ajouter une nouvelle, il me suffit de dupliquer le préfab de la tourelle déjà existante et d'apporter quelques modifications générales.
La nouvelle tourelle sera une sorte de lance-missile, donc il faut modifier les variables de cadence de tir, de portée, de dégâts, d'impact, etc.
Je me suis basé sur le rayon de l'impact, c'est à dire que dans le script principal, je vérifie si le rayon est supérieur à mon seuil, si oui alors c'est un lance-missile qui s'apprête à tirer.
Donc lorsque le projectile touche l'ennemi il fait apparaître une sphère (invisible en mode jeu) qui répertorie tous les ennemis à l'intérieur de celle-ci et leur inflige à tous des dégâts.
Nous avons maintenant une tourelle qui fait des dégâts de zone !
Je l'ajoute évidemment dans le Shop.


```
void Explode(Transform target){
    Collider[] colliders = Physics.OverlapSphere(transform.position, explosionRadius);
    foreach(Collider collider in colliders){
        if(collider.CompareTag("Ennemi")){
            Damage(collider.transform);
        }
    }

}
```

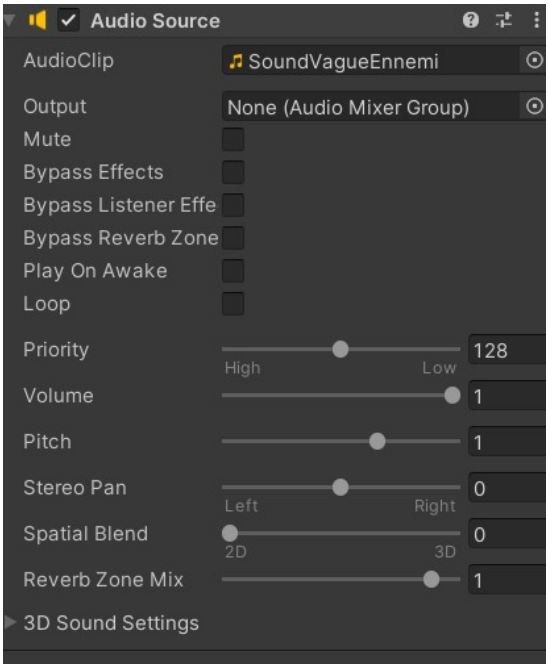
III.1.20. Ajouter des effets sonores (1 jour) :

Pour ce qui est des effets sonores, l'ajout se fait principalement à l'aide d'un composant présent dans Unity appelé « AudioSource ».
Il permet de jouer des fichiers audio définis, comme des bruitages ou bien des musiques d'ambiance.
Et il permet aussi de régler le volume, la spatialisation ou l'activation de la lecture en boucle du son.
La démarche est assez simple :

-Premièrement on importe le fichier audio dans le projet.

-Ensuite on l'ajoute au composant « AudioSource »

Maintenant plus qu'à déclencher la bande son au moment voulu, en l'appelant dans une fonction (par exemple celle de l'ennemi qui se prend un dégât).
Nous voilà avec des effets sonores qui viennent enrichir l'expérience de jeu.

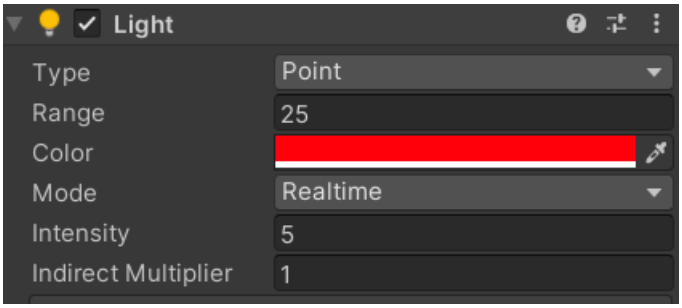


III.1.21. Ajouter des effets visuelles (1 jour) :

Bien que nous ayons déjà des effets de particule pour la mort d'un ennemi ou bien la construction d'une tourelle, il est envisageable d'ajouter plus d'effets visuels pour améliorer une fois de plus l'expérience du joueur.
Dans cette optique, j'ai tout de suite pensé à la prise de dégâts du joueur.
Pour cela lorsqu'un ennemi inflige un dégât au joueur qui perd alors une vie, l'animation que j'ai mise en place se déclenche.
Elle consiste en :

- Une brève apparition d'un flash de lumière rouge.

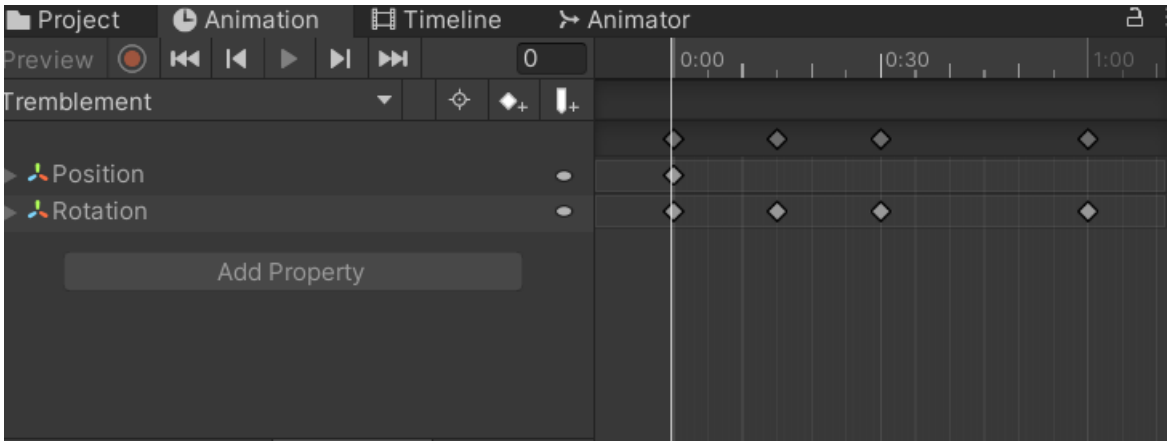
Ici, je crée d'abord un préfab d'une lumière rouge forte, que je viens par la suite invoquer et détruire au moment voulu pour donner l'illusion d'un flash.

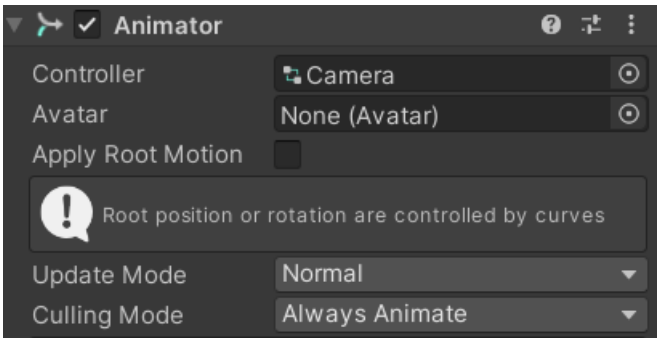


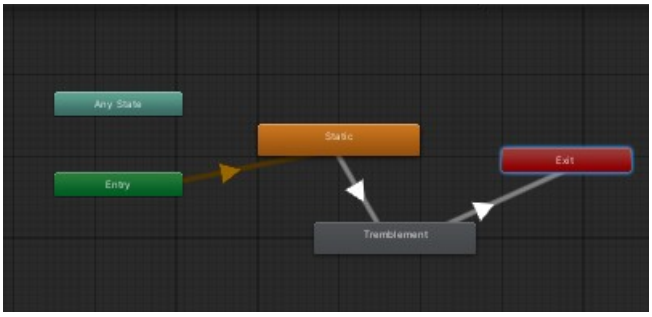
- Un tremblement de la caméra du joueur.

Ici, je donne à la caméra au moment voulu, une rotation dans plusieurs directions de manière contrôlée grâce à un « Animator » ce qui nous donne une illusion d'un tremblement.

« L'Animator » sur Unity est un système qui permet de gérer des animations pour les objets de la scène. On définit les positions, rotations et échelles d'un objet à différents moments clés, et Unity génère une transition entre ces états, comme dans un dessin animé.







- L'ajout d'un bruit de dégât (mais cette partie est plutôt de type sonore et non visuelle)

III.1.22. Correction des bugs (4 jour) :

J'ai consacré plusieurs jours à identifier et corriger les bugs qui survenaient. Cela inclut des problèmes mineurs, comme par exemple des erreurs d'affichage ou de collisions, ainsi que des bugs plus complexes.
Pour chaque bug, j'ai suivi cette approche :

- Identification : Reproduction du bug et analyse du code.
  Utilisation de Debug.log() et printf() pour savoir comment se comporte le code, et trouver d'où vient le problème.
- Résolution : Correction du code, modification, suppression ou ajustement des scripts.

- Tests : Vérification des changements effectués pour s'assurer que le bug était bien corrigé et qu'aucun nouveau bug était apparu. (Problèmes de programmeur ;)).

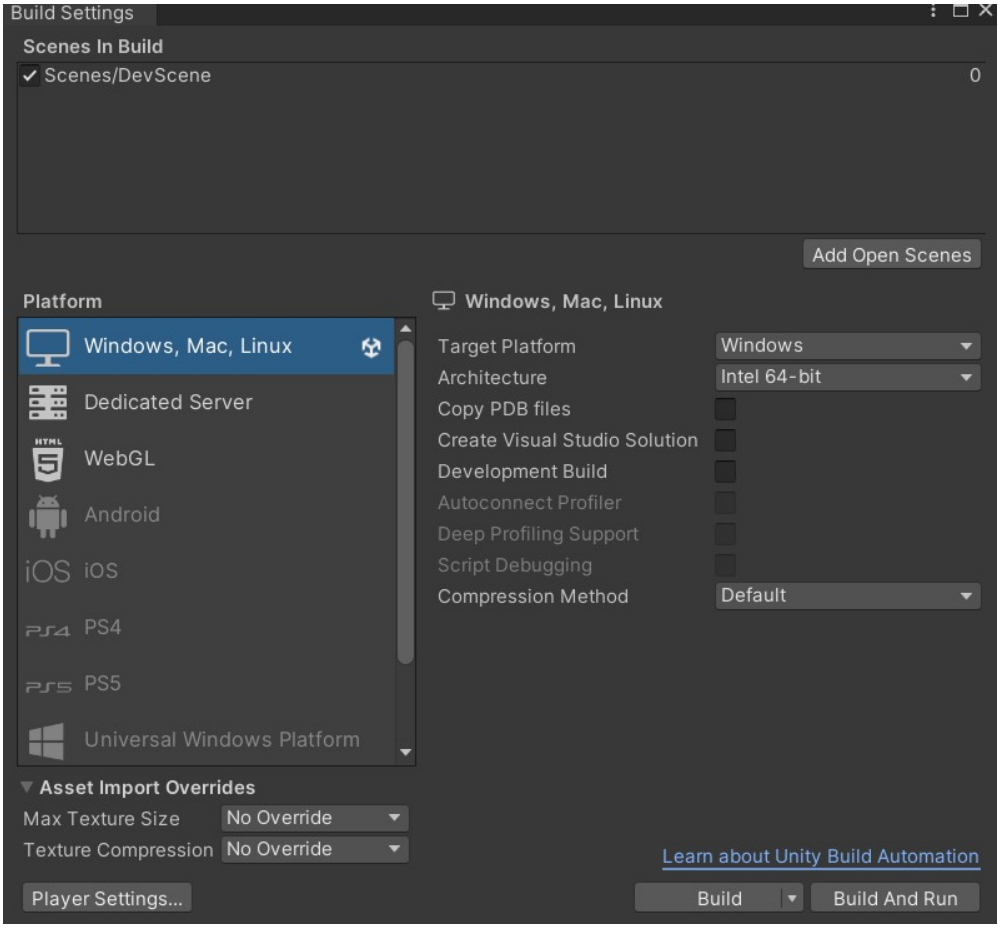
III.1.23. Sortir un premier Build (Version) (0,5 jour) :

Après avoir terminé les fonctionnalités principales ainsi que corriger les bugs majeurs, j'ai généré un build du jeu.
Le build est une version compilée et exécutable du jeu.
Pour ça, il m'a fallu :

- Configurer les paramètres du build : Choix de la plateforme, configuration des résolutions et des paramètres tiers.

- Générer le build : Utilisation de la fonction « Build and Run » dans Unity pour compiler et créer le fichier exécutable.

- Tests : Vérifier le bon fonctionnement du jeu sur la plateforme cible pour être sûr que le build s'est bien créé.



III.1.24. Faire tester le prototype à plusieurs personnes (2 jour) :

J’ai fait tester le prototype du jeu à plusieurs personnes, en ciblant différents profils (Joueur de jeu-vidéo ou non) pour avoir un retour diversifié sur l’expérience de mon jeu.

III.1.25. Prendre en compte les remarques (2 jour) :

Chaque testeur a été observé par mes soins pendant sa session pour repérer les éventuels problèmes ou améliorations possibles. J’ai pris des notes sur leurs impressions, et les points qui ont généré des frustrations. Ces retours sont essentiels pour affiner le jeu dans le futur.

IV. Suite et fin de partie générale.

IV.1. Conclusion et perspectives :

- **Bilan du projet :** Ce projet a été une très belle expérience pour moi.
  En partant d'une feuille vierge, j'ai pu apprendre et appliquer les concepts clés de Unity, tout en me débrouillant pour faire en sorte d'avoir un début de jeu fonctionnel.
  Il n'a pas été de tout repos de mettre en corrélation toutes les informations acquises au cours du projet mais en surmontant ces défis, je ressors plus grandi de cette expérience !
  Cela m'a également permis de découvrir le haut potentiel de Unity en tant que moteur de jeu gratuit.
  Je suis réellement satisfait de mes résultats obtenus, bien que je reconnaisse qu'il existe encore des axes d'amélioration, que je continue d'ailleurs d'y travailler.

- **Axes d'amélioration et fonctionnalités futures :** Bien que ce début de jeu soit fonctionnel, plusieurs axes d'amélioration restent à venir.
  Voici quelques pistes :

- **Système de progression et de récompenses :** Ajouter un système d'améliorations pour les tourelles et le joueur. Par exemple, des upgrades de dégâts, de portée ou de vitesse de tir, ou encore des compétences spéciales pour le joueur, afin de maintenir l'engagement et la motivation du joueur.

- **Optimisation des performances :** Bien que le jeu fonctionne bien sur les configurations actuelles, une optimisation plus poussée serait à envisager pour garantir un bon fonctionnement sur des machines moins puissantes comme sur mobile par exemple.

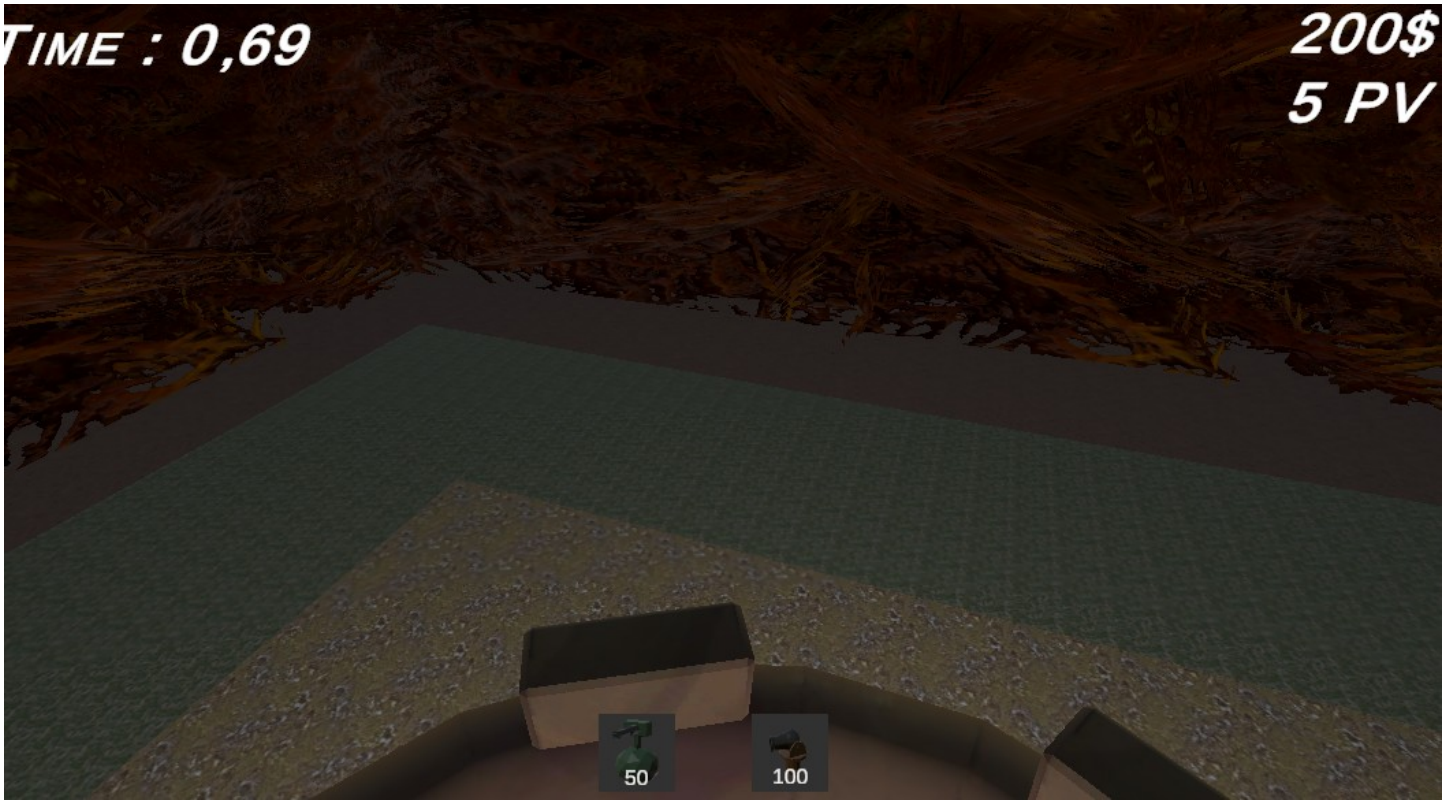
- **Ajout d'un menu principal :** Un des ajouts majeurs serait l'intégration d'un menu principal pour le jeu. Ce menu permettrait de pouvoir démarrer une nouvelle partie, d'en continuer une anciennement sauvegardée, d'accéder aux options de réglage, ou tout simplement de quitter.

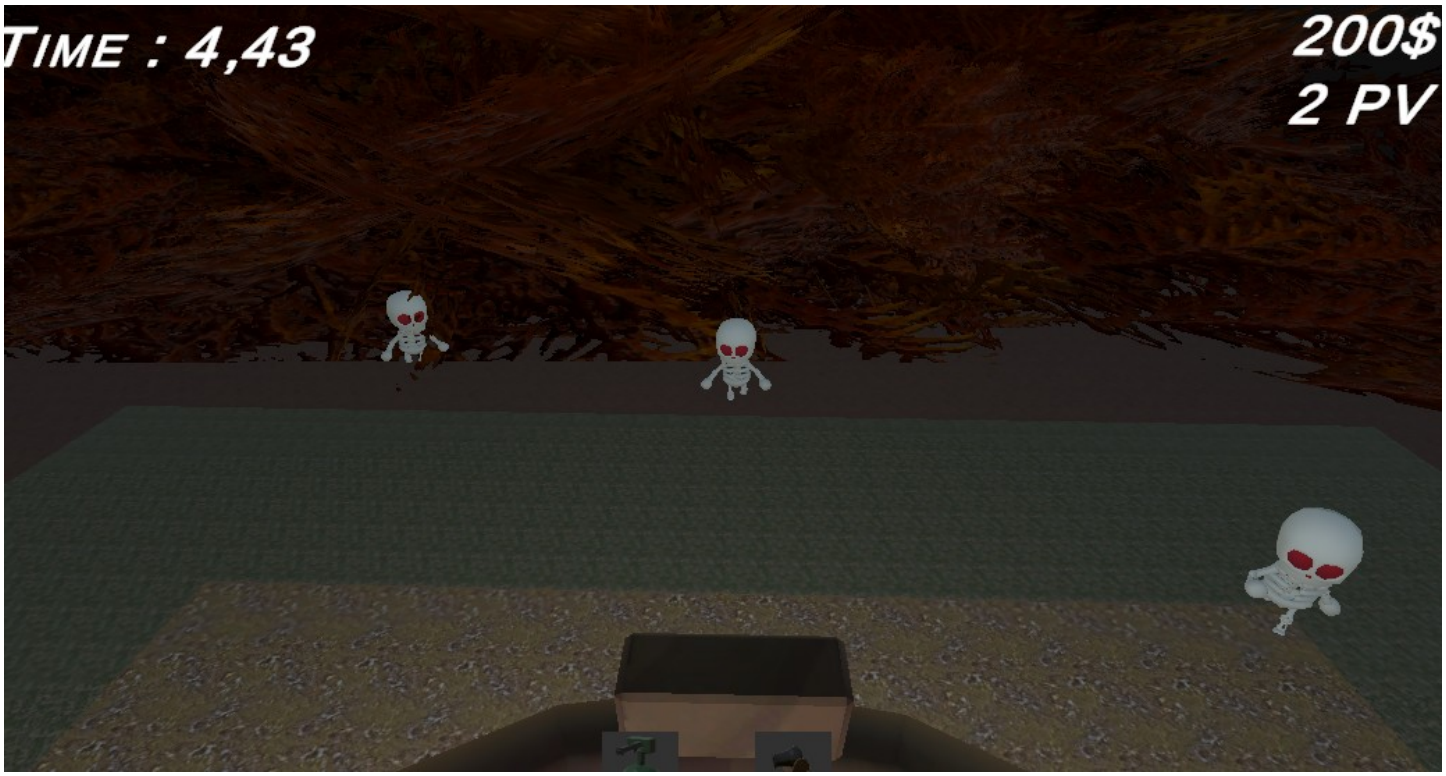
- **Implémentation d'un système de sauvegarde :** Ajouter un système de sauvegarde de l'avancée du joueur. Malheureusement Actuellement, le jeu ne permet pas de conserver ses progrès une fois la session éteinte.

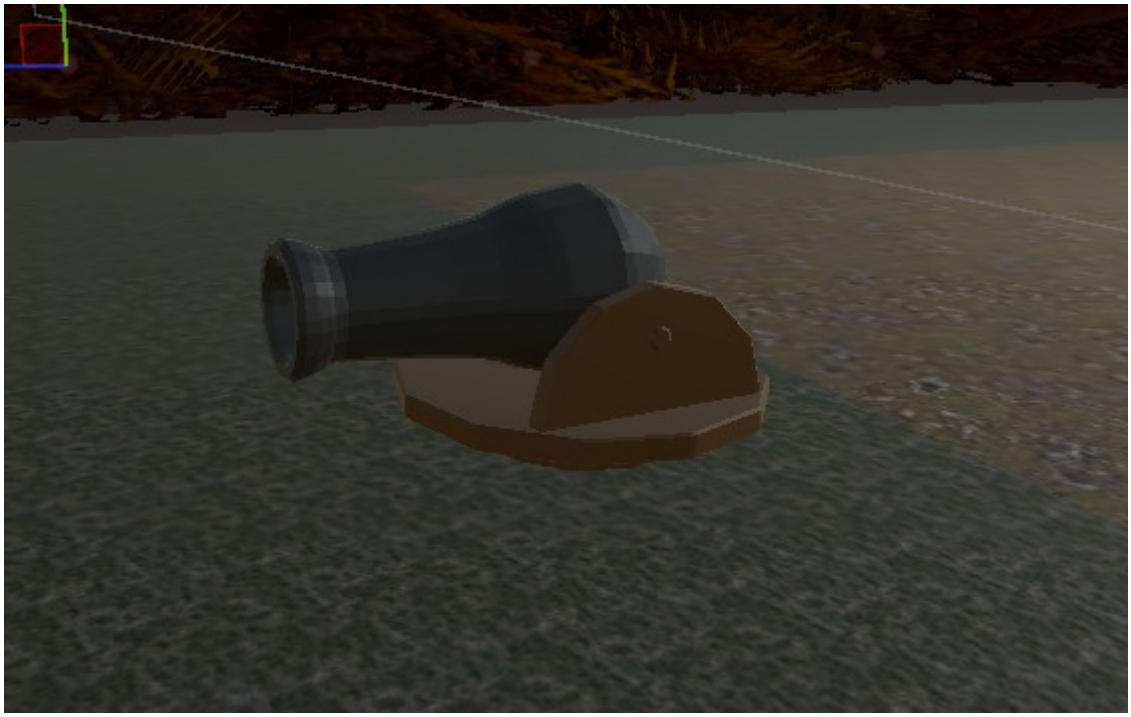
- **Un mode multijoueur :** Pourquoi ne pas envisager un mode multijoueur avec comme principe l'entraide des deux joueurs au sein de la même tour, ou une sorte d'1v1 où le Joueur-1 doit se défendre de l'autre joueur, qui lui, ferait apparaître les ennemis.

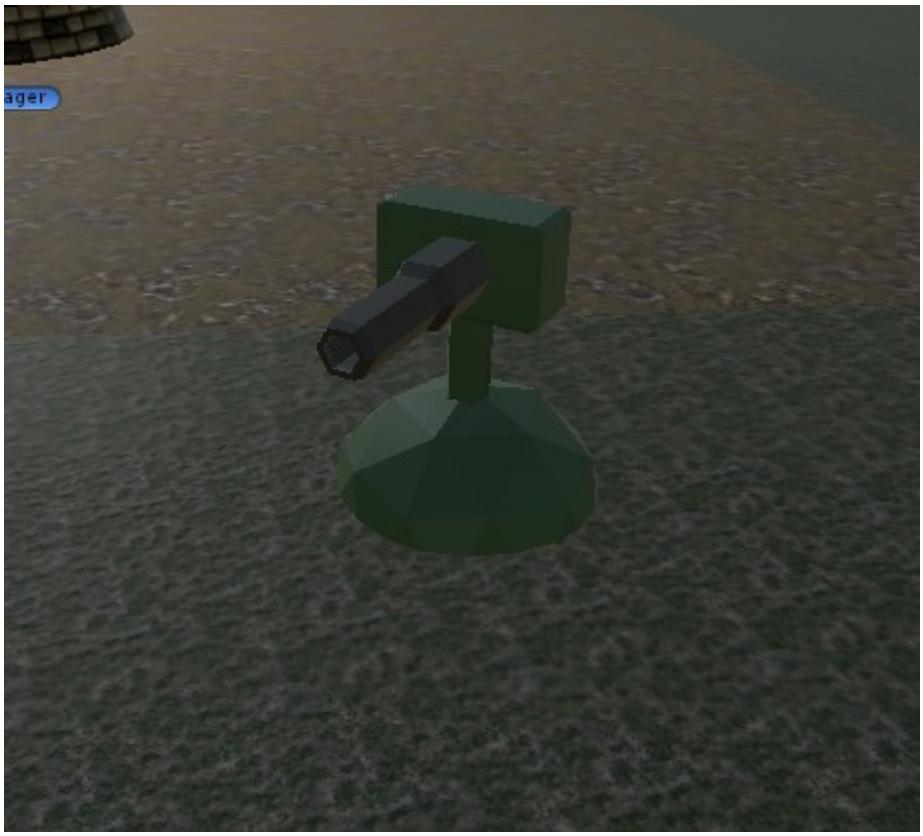
IV.2. Annexes:

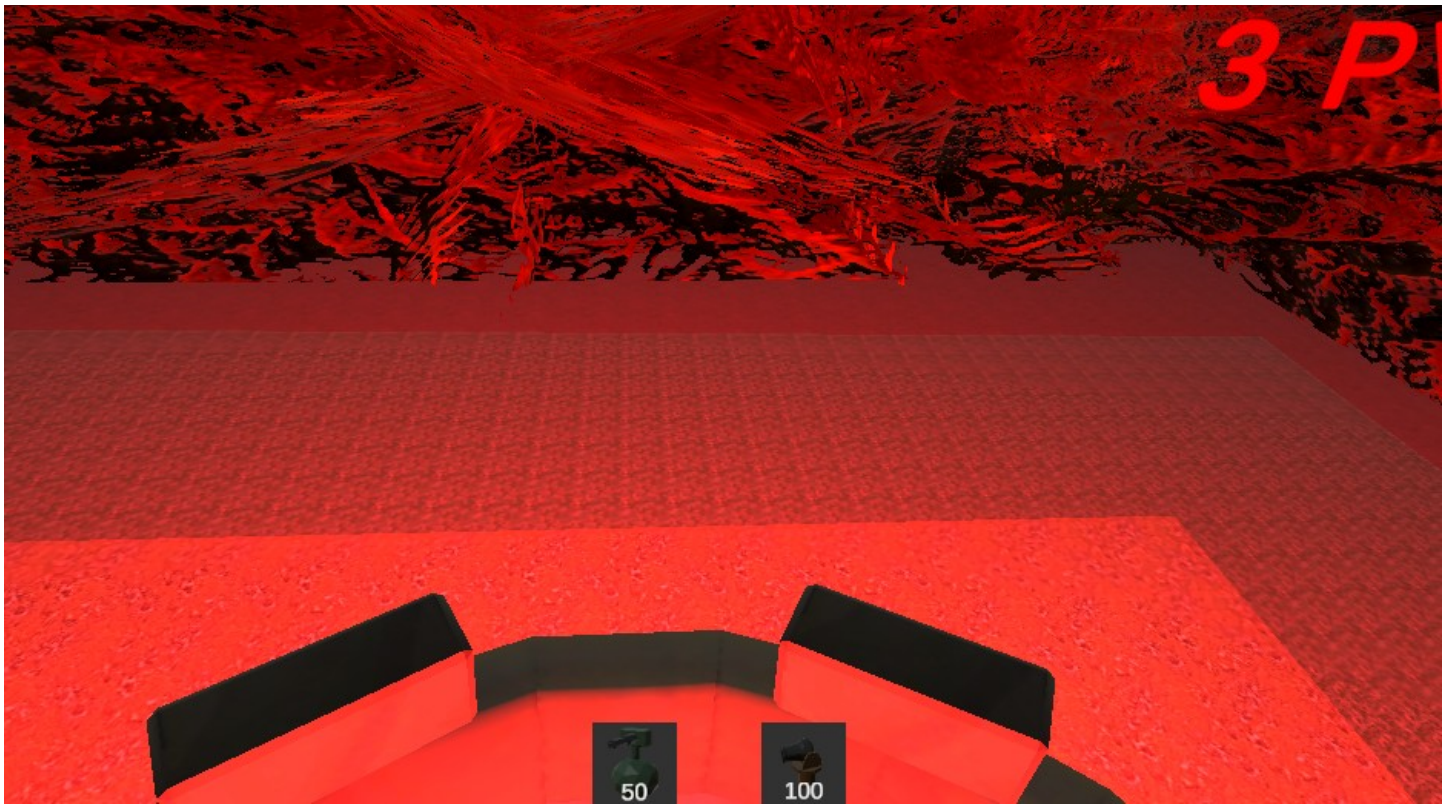
- **Lien dépôt Git :** https://github.com/Jomare59/Tower_Defense.git
- **Capture d'écrans** :

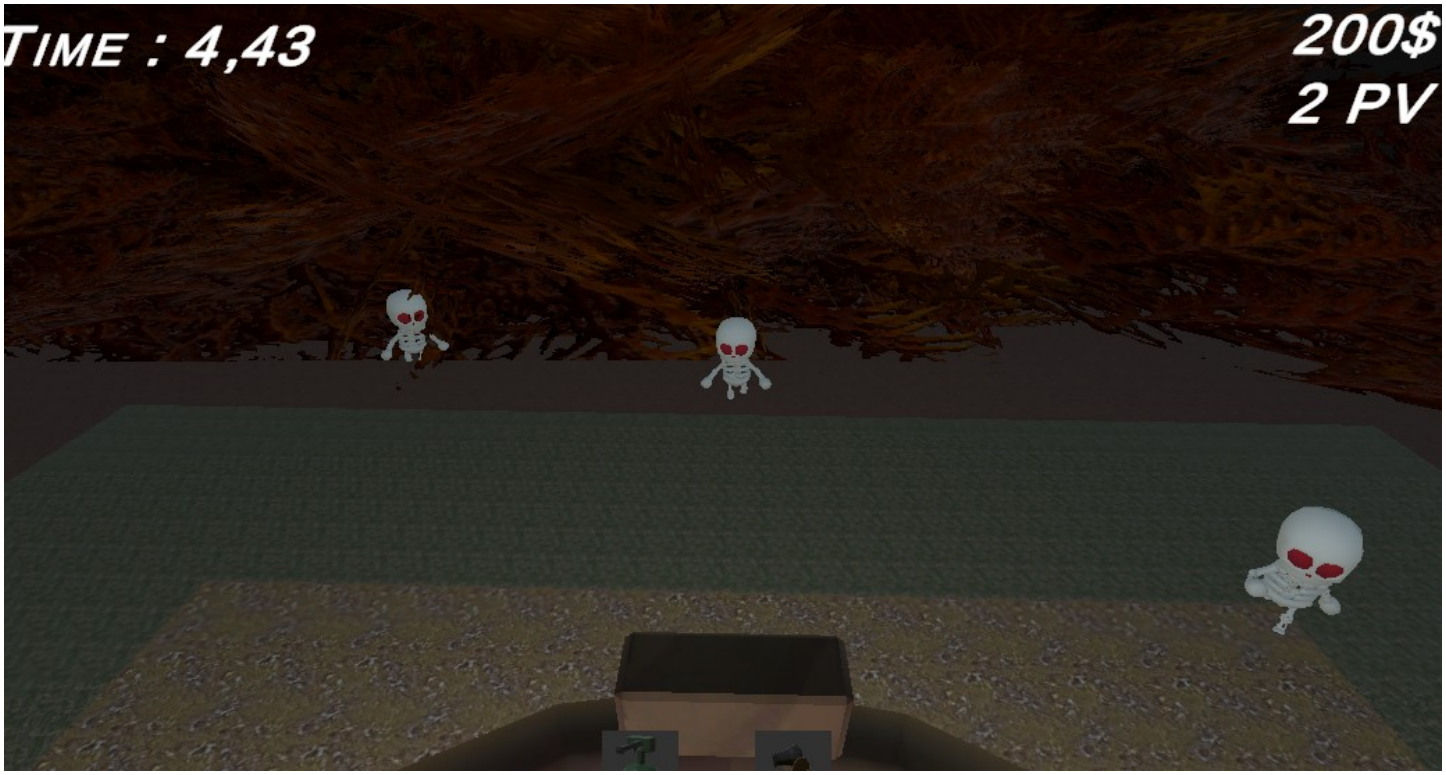









50
100



IV.3. Bibliographie :

Unity Documentation

APPRENDRE LE C# [TUTO PROGRAMMATION COMPLET DÉBUTANT]

Apprendre le C# - Épisode 1 : Les variables (Unity 3D)

[TUTO] Unity débutant : apprendre à créer des Jeux-Video avec Unity : les bases - YouTube

Livre BU Mont houy : 24H pour créer un jeu vidéo.