Automne 2020

Projet – VI50
TowAR Defense

Rapport

Réalisé par
Alexis GUERET

Table des matières

1 Introduction 3
1.1 Principe du Jeu 3
1.2 Attentes vis-à-vis de l'implémentation 3
2 Idéation du projet 4
2.1 Définition de l'ambiance graphique 4
2.2 Définition des interactions 5
3 Description de l'implémentation 6
3.1 Ennemis 6
3.2 Défenses 10
3.3 Système d'argent 12
4 Déroulement d'une partie 13
4.1 Placement du plateau de jeu 13
4.2 Phase de placement des défenses 13
4.3 Phase de défense 14
5 Bilan et perspectives 15
5.1 Apports de ce projet 15
5.2 Perspectives 15

1 Introduction

Ce rapport traite du travail que j'ai effectué dans le cadre du projet de l'unité de valeur VI50 : Réalité virtuelle et augmentée. J'ai décidé de réaliser « TowAR Defense », un jeu de Tower Defense en réalité augmentée sous Hololens.

1.1 Principe du Jeu

Dans TowAR Defense, le but est de protéger un château de vagues de monstres successives. Pour réaliser cet objectif, le joueur a plusieurs possibilités. La principale arme du joueur est la construction de défenses plaçables sur des endroits prédéfinis. Il reviendra alors au joueur de définir les endroits les plus appropriés pour utiliser les défenses au mieux de leurs capacités. Ces défenses coutent plus ou moins d'argent en fonction de leur effet et de leur utilité. Cet argent est obtenable directement en tuant des monstres.
Le joueur a aussi une compétence qui lui permet de tirer des projectiles directement sur les ennemis. Cette compétence rechargeable mettra à rude épreuve la précision du joueur.

1.2 Attentes vis-à-vis de l'implémentation

Pour implémenter ce jeu, nous pouvons distinguer les principales composantes :

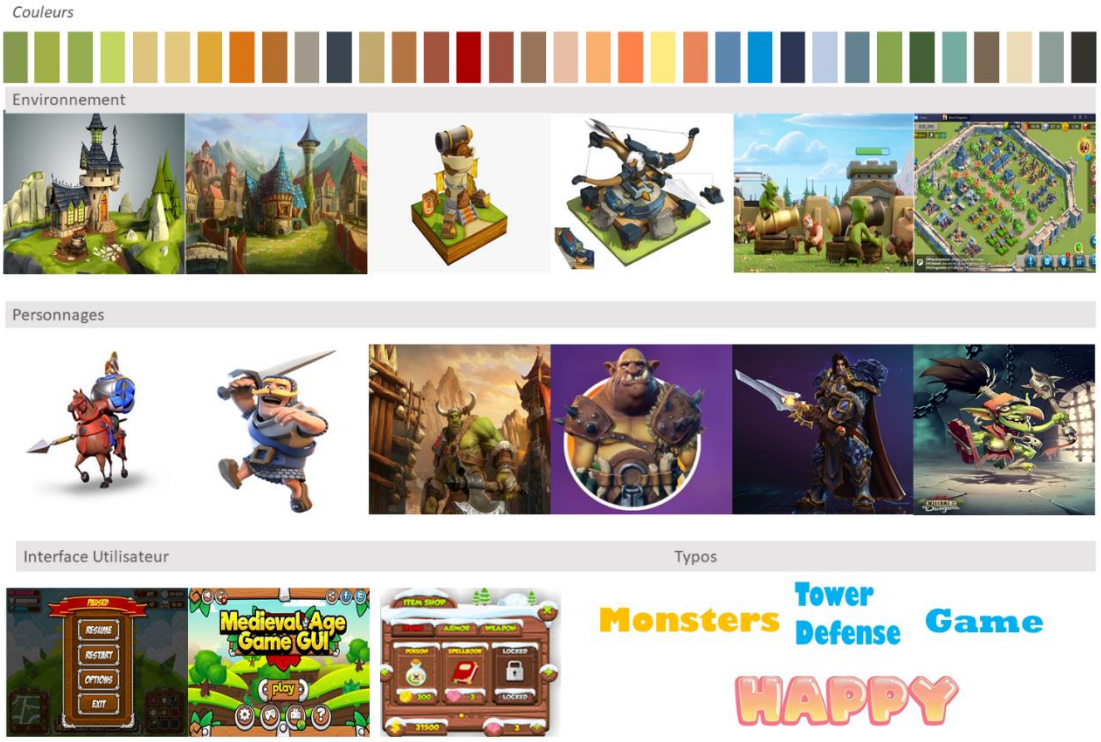
- Fonctionnement des ennemis
- Fonctionnement des tourelles
- Système de boutique.
- Système de compétence
- Game Logic/Loop

En plus des points susmentionnés, nous aborderons les étapes d'optimisation, les difficultés rencontrées et les résultats obtenus.

2 Idéation du projet

Avant d'entamer un projet de cette envergure, il est bien plus raisonnable de commencer par une étape d'idéation afin de bien définir quels sont les objectifs du projet, l'ambiance graphique du projet ainsi que les interactions du joueur avec le produit final.

2.1 Définition de l'ambiance graphique



Pour implémenter ce jeu, nous pouvons distinguer les principales composantes :

Pour ce jeu, j'ai choisi une apparence relativement cartoon et colorée pour donner un aspect fun et exagéré au jeu qui je trouve, se prête plutôt bien aux Tower Defense. Le 2e point qui m'a poussé à choisir ce type d'ambiance graphique est sa popularité, impliquant une grande quantité d'assets disponibles gratuitement sur le Unity Asset Store. Enfin, cette ambiance graphique pouvant être assez sommaire, il est aisé d'incorporer les modèles 3D que j'aurais fais moi même sans faire tâche avec les assets importés.

2.2 Définition des interactions

Pour définir les interactions de mon projet, j'ai appliqué la méthodologie des $3I^2$ vue en cours :

Les I^2 fonctionnelles sont les suivantes :

- Utilisation d'un pouvoir permettant de lancer des boules de feu pour tuer les ennemis. 1)
- Placer des défenses sur le plateau de jeu. 2)
- Le joueur peut se déplacer autour du plateau de jeu pour interagir plus facilement avec celui-ci. 3)
- Le joueur peut placer le terrain où bon lui semble. 4)
- Le joueur doit pouvoir discerner les emplacements où il peut placer des défenses. 5)

Les PCV sont :

- Observer le monde virtuel. : 3/5
- Se déplacer dans le monde virtuel : 3
- Agir sur le monde virtuel : 1/2/4

Schèmes et métaphores :

1 : Utilisation d'une métaphore de « tapping » pour créer des boules de feu
2 : Utilisation d'une métaphore de « tapping » sur les endroits attribués aux défenses pour créer des défenses
3 : Exploitation du schème de déplacement dans 3 dimensions.
4 : Exploitation du schème de préhension et de manipulation spatiale du plateau de jeu

Interfaces comportementales :

- Observation :
    - Ecran de l'HoloLens.

- Manipulation du plateau et tapping :
    - Reconnaissance des mouvements des mains de l'HoloLens

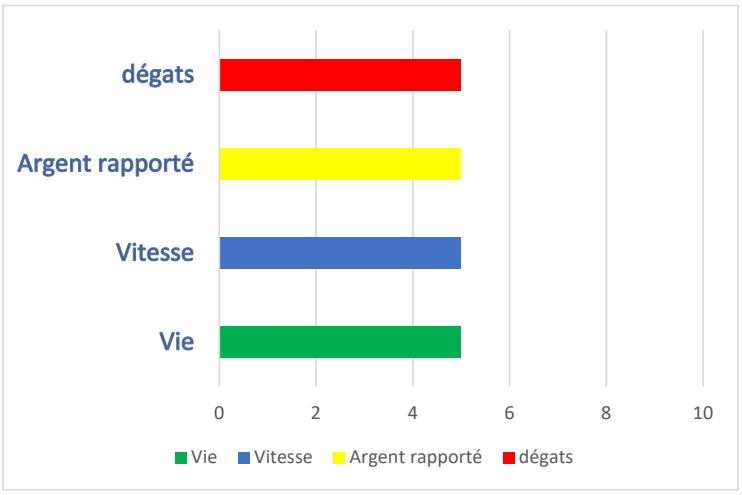
3 Description de l'implémentation

Cette partie décrit la partie implémentation du projet ainsi que les différentes décisions prises pour passer de l'idéation à la réalisation.

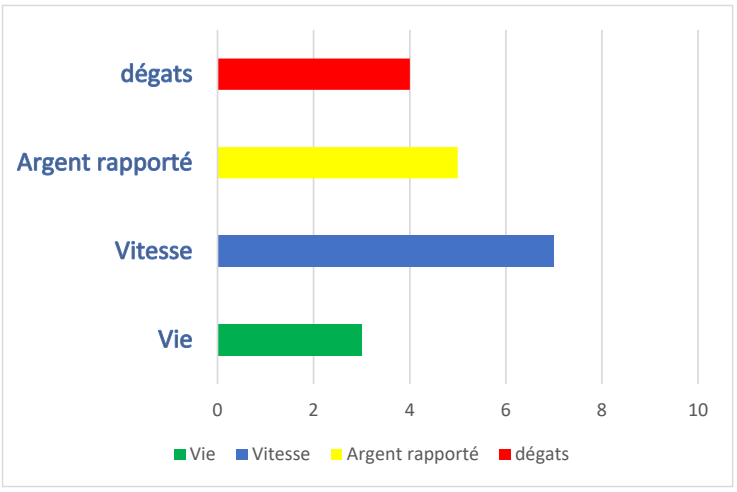
3.1 Ennemis

Le comportement des ennemis/monstres est assez basique. En effet, il se résume à avancer de leur point d'apparition jusqu'au château qu'ils doivent détruire. Une fois à portée de ce château, ils commencent à attaquer.
Ils sont caractérisés par 4 caractéristiques variables : Les dégâts qu'ils infligent à chaque attaque, leurs points de vie, leur vitesse de déplacement et l'argent qu'ils rapportent lorsqu'ils sont tués.
De ces distinctions, j'ai créé 3 types d'ennemis :

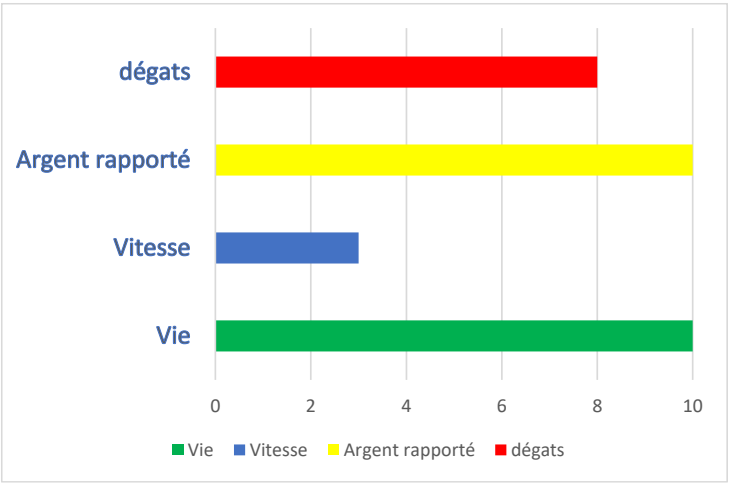
- Le squelette :



- Le diable : vie faible, vitesse élevée, argent rapporté et dégâts moyens.



- Le Boss : Vie très élevée, vitesse faible, argent rapporté et dégâts élevés.



3.1.1 Path

Le Path est un Prefab Unity utilisé pour définir le chemin que les ennemis doivent suivre. Il est caractérisé par un ensemble de Transform indiquant la position de tous les points du chemin par lesquels les ennemis vont passer.

```
private void Start()
{
    Transform[] pathTransforms = GetComponentsInChildren<Transform>();
    pathNodes = new List<Transform>();
    for (int i = 0; i < pathTransforms.Length; i++)
    {
        if (pathTransforms[i] != this.transform)
        {
            pathNodes.Add(pathTransforms[i]);
        }
    }
}
```

3.1.2 EnemyHandler

A chaque ennemi est attribué le script « EnemyHandler ». Il définit l'ensemble du comportement des ennemis : leurs déplacements, leurs attaques, les dégâts qu'ils prennent et leur mort.
Chaque ennemi est construit de la même façon, encore une fois via les préfabs Unity : Un modèle 3D auquel est attaché le script « EnemyHandler » et une barre de vie liée au script précédemment cité. Etant donné que tous les ennemis possèdent le même script de gestion, il faut pouvoir lui faire comprendre à quel type d'ennemi il est attaché. Ceci est fait par l'utilisation de tags Unity correspondant à chacun des 3 types d'ennemis.

```
switch(this.gameObject.tag)
{
    case "Skeleton":
        this.hp = 50;
        this.dmg = 5;
        this.attackSpeed = 1;
        this.movementSpeed = 5*scale;
        this.goldReward = 50;
        break;

    case "Devil":
        this.hp = 30;
        this.dmg = 4;
        this.attackSpeed = 1;
        this.movementSpeed = 7*scale;
        this.goldReward = 50;
        break;

    case "Bull":
        this.hp = 100;
        this.dmg = 8;
        this.attackSpeed = 1;
        this.movementSpeed = 3*scale;
        this.goldReward = 100;
        break;
}
```

Figure 1 Initialisation des ennemis dans la fonction Start

Le comportement des ennemis est dicté par une boucle très simple :
Tant que la porte du château n'est pas atteinte, il se déplace vers celle-ci. Une fois à portée de la porte, il l'attaque jusqu'à ce que la porte soit détruite.

```
private void Update()
{
    // goToGate is true while the gate has not been reached
    if (goToGate)
    {
        //function that handles the movement
        this.MoveToGate();
    }
    else
    {
        //goes through this only once when the attack is possible (ie the enemy is close enough to the gate and is not attacking)
        if (!isattacking)
        {
            this.canAttack = true;
            this.gateHandler = GameObject.FindGameObjectWithTag("CastleGate").GetComponent<GateHandler>();
            isattacking = true;
        }

    }
    //if the coroutine to attack has not already been started, start the fighting corountine
    if (canAttack && !isAttacking)
    {
        StartCoroutine(EnemyDamage());
    }
}

IEnumerator EnemyDamage()
```

Fonction de déplacement :

```
private void MoveToGate()
{
    //first call of the function
    if(!this.isWalking)
    {
        pathNodes = Object.FindObjectOfType<Path>().getPathNodes();
        this.isWalking = true;
        this.anim.SetBool("Walking", true);
        this.currentDestination = pathNodes[currentNode];
    }
    //Agent reaches step
    if ((transform.position - currentDestination.position).magnitude < 1*scale)
    {
        //Agent reaches last step
        if(currentNode==pathNodes.Count-1)
        {
            goToGate = false;
            this.anim.SetBool("Walking", false);
        }
        else
        {
            currentNode++;
            this.currentDestination = pathNodes[currentNode];
            this.transform.LookAt(this.currentDestination);
        }

    }
    else
    {
        transform.position = Vector3.MoveTowards(transform.position, currentDestination.position, this.movementSpeed * Time.deltaTime);
    }

}
```

Coroutine d'attaque :

```
IEnumerator EnemyAttack()
{
    //bool so that the corountine is not called multiple times
    isAttacking = true;
    // repeat while the gate is not destroyed
    while (gateHandler!=null)
    {
        gateHandler.takeDamage(this.dmg);
        this.anim.SetTrigger("Attack");
        yield return new WaitForSeconds(1/attackSpeed);
    }
}
```

3.2 Défenses

Il existe 4 types de défense ayant chacune des caractéristiques et effets différents.

- **Canon Tower**

Tour basique tirant des boulets de canon infligeant des dégâts à intervalles réguliers.
Cout : 500 d'or
Amélioration : augmentation des dégâts

- **Death Tower**

Dérivé de la canon Canon Tower. Cette tour inflige d'énormes dégâts mais dispose d'un temps de recharge très long.
Cout : 1000 d'or
Amélioration : augmentation des dégats

- **Ice Tower**

Son tir n'inflige pas de dégâts aux ennemis mais les ralentis de moitié sur une certaine durée. Ne cible que les ennemis qui ne sont pas déjà sous ses effets.
Cout : 750 d'or
Amélioration : augmentation de la durée du ralentissement

- **Fire Tower**

Son tir brule les ennemis infligeant des dommages aux ennemis sur la durée. Comme la Ice Tower, elle ne cible que les ennemis qui ne sont pas affectés par une brulure.
Cout : 750 d'or
Amélioration : augmentation des dégâts

La boucle des défenses est encore une fois relativement simple : chercher une cible, tourner le canon pour faire face à l'ennemi et tirer.

```
// Update is called once per frame
void Update()
{
    //Get a target to shoot
    if(id==ICE_ID)
    {
        GetIceTarget();
    }
    if(id == FIRE_ID)
    {
        GetFireTarget();
    }
    if(id==CANON_ID || id == DEATH_ID)
    {
        GetTarget();
    }

    //rotates the tower so that it aims at the target
    if(this.target!=null && this.id!=FIRE_ID && this.id!=ICE_ID)
        LookTarget();
    //Starts shooting coroutine if not already started
    if (!isShooting)
    {
        if(id==CANON_ID)
            StartCoroutine("Shoot");
        if (id == DEATH_ID)
            StartCoroutine("DeathShoot");
        if (id == FIRE_ID)
            StartCoroutine("FireShoot");
        if (id == ICE_ID)
            StartCoroutine("IceShoot");
    }
}
```

Les tours de glace et feu ont un système de recherche de cible différent car elles ne prendront pas pour cible les cibles respectivement gelées et en feu.

```
private void GetTarget()
{

    // Get all objects that are at shooting distance
    col = Physics.OverlapSphere(this.transform.position, shootingDistance);

    // Get the ennemies among those objects
    foreach (var hit in col)
    {
        if ((hit.tag == "Skeleton" || hit.tag=="Devil" || hit.tag=="Bull") && this.target == null)
            this.target = hit.gameObject;
    }

    // Delete the current target if it gets out of range
    if (this.target!=null)
    {
        if ((this.target.transform.position - this.transform.position).magnitude > this.shootingDistance)
        {
            this.target = null;
        }
    }
}
```

L’utilisation d’un IEnumerator dans la mécanique de tir permet de simuler le temps de recharge des canons.

```
private IEnumerator Shoot()
{
    this.isShooting = true;
     while(true)
      {
        if(target!=null)
        {
            ShootTarget(CANON_ID);
            yield return new WaitForSeconds(RELOAD_TIME_CANON);
        }
        else
        {
            yield return new WaitForSeconds(0.2f);
        }
    }
}
```

3.3 Système d’argent

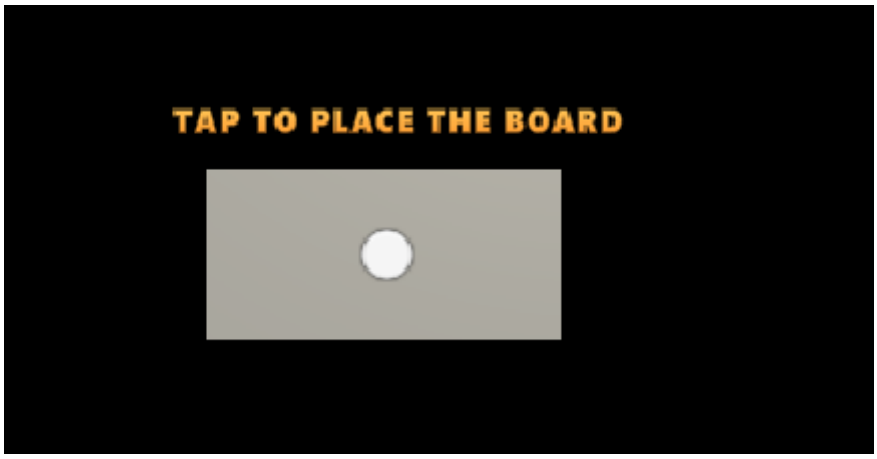
L’argent est la seule récompense obtenue au cours du jeu. On peut en récupérer en tuant des ennemis. Il permet d’acheter des tourelles ainsi que de les améliorer.

Cette partie est régie dans le script « goldhandler » en majeure partie et « GameController » dans une moindre mesure.

4 Déroulement d'une partie

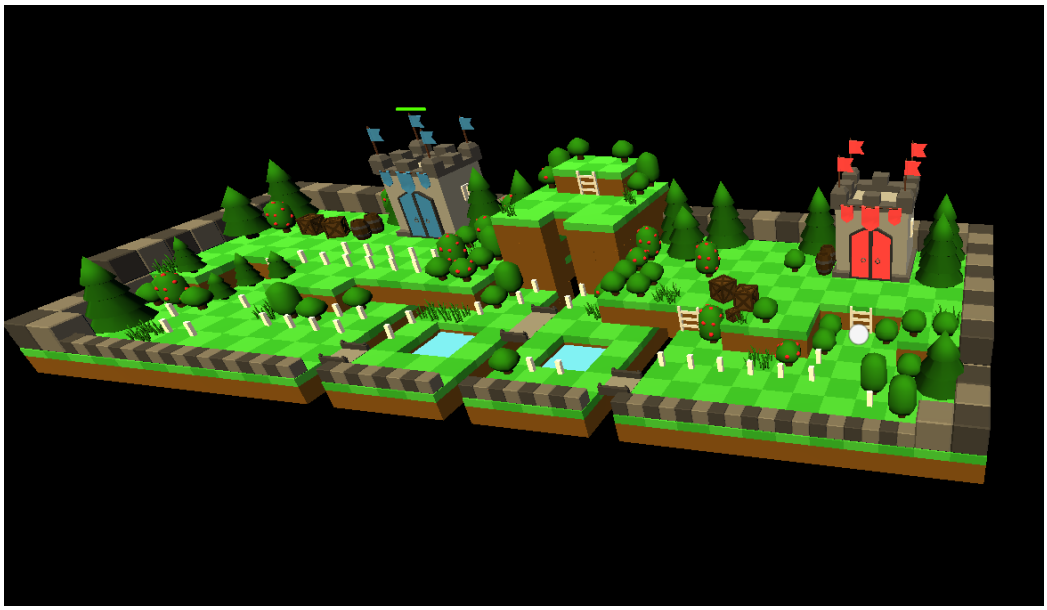
4.1 Placement du plateau de jeu

Au lancement du jeu, il vous est demandé de placer le terrain de jeu. Pour se faire, un cube symbolisant le terrain est affiché à l'écran, on peut le manipuler grâce aux fonctionnalités de l'HoloLens. Une fois en place, il suffit d'utiliser le bouton indiquant « tap to place the board » pour placer le terrain.



4.2 Phase de placement des défenses

Une fois le plateau de jeu en position, il est possible de placer des tourelles de défense aux emplacements indiqués par un rectangle blanc. Pour se faire, il faut effectuer un « air tap » sur le sol où l'on veut placer la défense, une interface apparait alors proposant les 4 différentes tours à construire sous forme de boutons. L'appuie sur chacun de ces boutons entraine la construction de la tour associée. Les ennemis viennent de la tour rouge et vont vers la tour bleue. Il faut donc placer les défenses en fonction de ce principe.



Le nombre d'ennemis que vous allez affronter ainsi que leur type est indiqué avant chaque manche. Ceci permet de s'adapter au mieux à la manche qui suit.
Des tours de glace peuvent être par exemple très intéressantes contre les ennemis rapides.
Vous pouvez également changer le type des tourelles déjà présentes sur le terrain ainsi que les améliorer pour accroitre leurs capacités.

Une fois que vous êtes prêt, Appuyez sur « START » pour lancer une manche.

4.3 Phase de défense

Dès le lancement de la partie, des ennemis apparaissent. La fréquence d'apparition des ennemis ainsi que leur nombre augmentent au fur et à mesure de l'avancement. Une manche peut se terminer de 2 façons :

- Défaite : Les ennemis parviennent à détruire le château à défendre.
- Victoire : tous les ennemis sont vaincus.

Deux moyens vont vous permettre de vaincre les ennemis : les défenses que vous avez installées avant le début de la manche et des boules de feu que vous pouvez lancer en effectuant le geste « air tap » de Hololens.

5 Bilan et perspectives

5.1 Apports de ce projet

Travailler sur ce projet pendant presque tout un semestre m'a énormément apporté.
Bien sûr, cela m'a permis d'améliorer mes compétences en Unity et en développement de jeux en général. J'ai beaucoup appris sur l'approche à adopter vis-à-vis de la conception du code pour les jeux vidéo. Le fait d'avoir été capable de réaliser un tout cohérent à partir de nombreux scripts et ainsi délivrer l'expérience que j'avais imaginé est vraiment gratifiant.

Ensuite, cette expérience m'a permis d'approfondir mes compétences en réalité augmentée ainsi que de découvrir l'HoloLens afin de réaliser un projet entier sur cette plateforme. C'est une technologie que je trouve particulièrement intéressante et permet des projets réellement innovants.

Enfin, il s'agit sans doute du projet le plus important et ambitieux sur lequel j'ai eu l'occasion de travailler durant ma scolarité. Cela autant en termes de difficulté théorique que simplement en termes de charge de travail. Je pense que les compétences d'organisation, d'adaptation et d'idéations que j'ai dû développer durant ce projet me seront très utiles durant ma future carrière.

5.2 Perspectives

Je vois beaucoup d'évolutions possibles à ce projet.

1) Création du terrain par le joueur
   Lorsque j'ai commencé ce projet, j'avais dans l'idée de laisser au joueur le soin de réaliser lui-même son environnement en plaçant directement chaque « case » par laquelle les ennemis doivent passer. Ceci ajouterait de l'immersion. Quand j'ai essayé de l'implémenter, j'ai trouvé que cela alourdissait quelques peu le début de la partie et n'était pas très « user-friendly ». C'est pourquoi j'ai décidé de le retirer. Je suis cependant convaincu que bien implémenté, cela pourrait beaucoup apporter.

2) Prise en compte du monde réel
   On peut imaginer une reconnaissance du monde réel permettant de s'affranchir totalement du plateau de jeu. On définirait simplement un point de départ et un point d'arrivée et les ennemis prendraient en compte la topologie du terrain pour se frayer un chemin. Ceci créerait par exemple des endroits où l'ennemi est obligé d'escalader pour poursuivre son chemin. Je trouverai ça vraiment exceptionnel d'avoir une armée de squelettes chez moi qui essaye d'escalader mes meubles !
   Ceci pourrait aussi pousser les joueurs à créer des barrières naturelles dans le monde réel, entrainant, encore une fois, une immersion accrue.

3) Développements Logiques

   La suite la plus logique de ce projet serait d'ajouter des types d'ennemis et de tourelles. J'ai pensé à des tourelles qui ne tirent pas mais rapportent de l'argent périodiquement, des ennemis qui attaquent à distance, des ennemis nécessitant des tourelles spécifiques pour être attaqués.