

Documents Pour les SAEs S2

2022-2023

Chabane Juba

Shinwari Saidkamal

Ngatchou NganKam Antoine

Diagrammes de classe :

Diagramme complet du modèle :

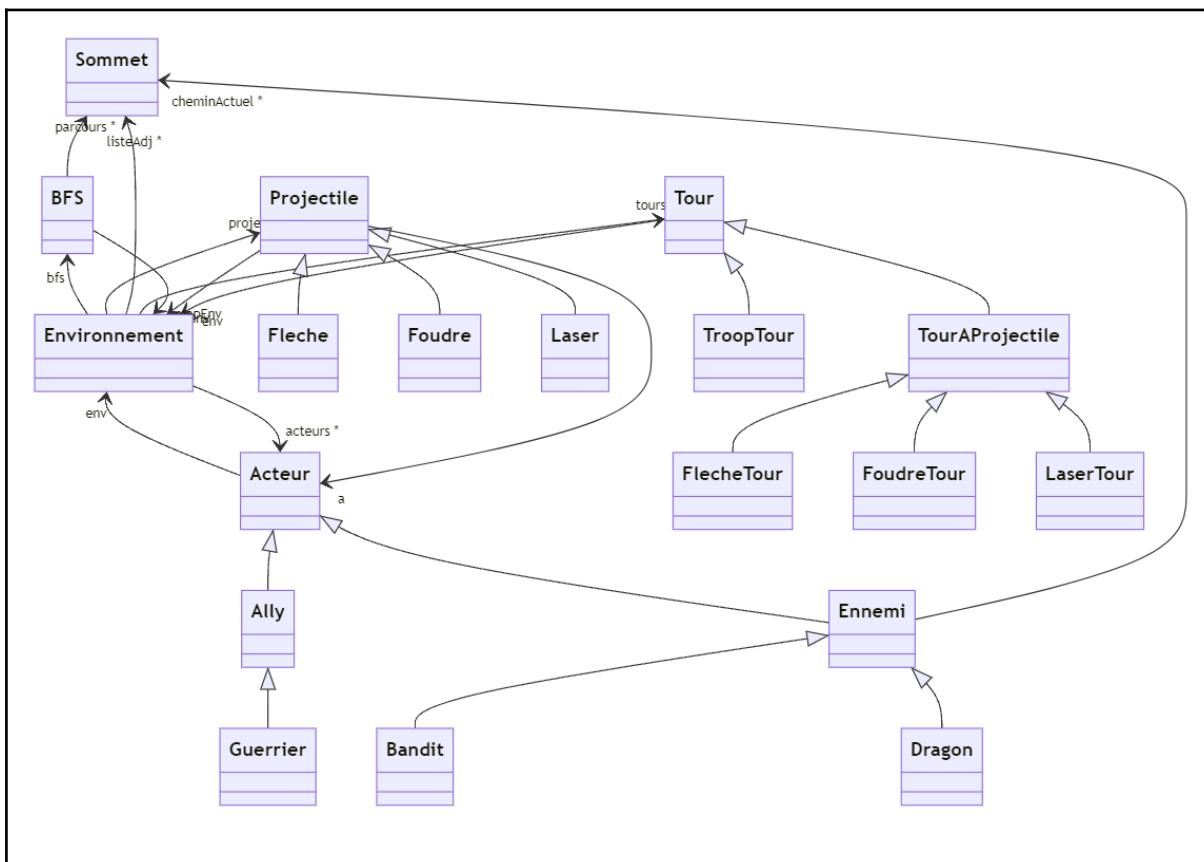


Diagramme des Acteurs :

Nous avons décidé de vous présenter la partie du modèle qui concerne les acteurs, car c'est là où l'on a eu le plus recours au polymorphisme.

Pour commencer, on a choisi de mettre la super classe Acteur en abstraite, car tous nos Acteurs possèdent une méthode `agir()` et une méthode `attaquer()`, mais ils n'ont pas tous la même manière d'attaquer, donc nous les avons déclaré abstraite dans la super-classe Acteur et nous les avez codé dans les sous-classes.

Les fonctions tels que `estVivant()`, `meurt()`, `decrementationPv(int pv)`, etc. dont les mêmes pour tous les acteurs, un Ennemi et un Ally perdent leurs PV de la même manière donc on l'a codé dans Acteur et nous en héritons dans toutes les sous-classes.

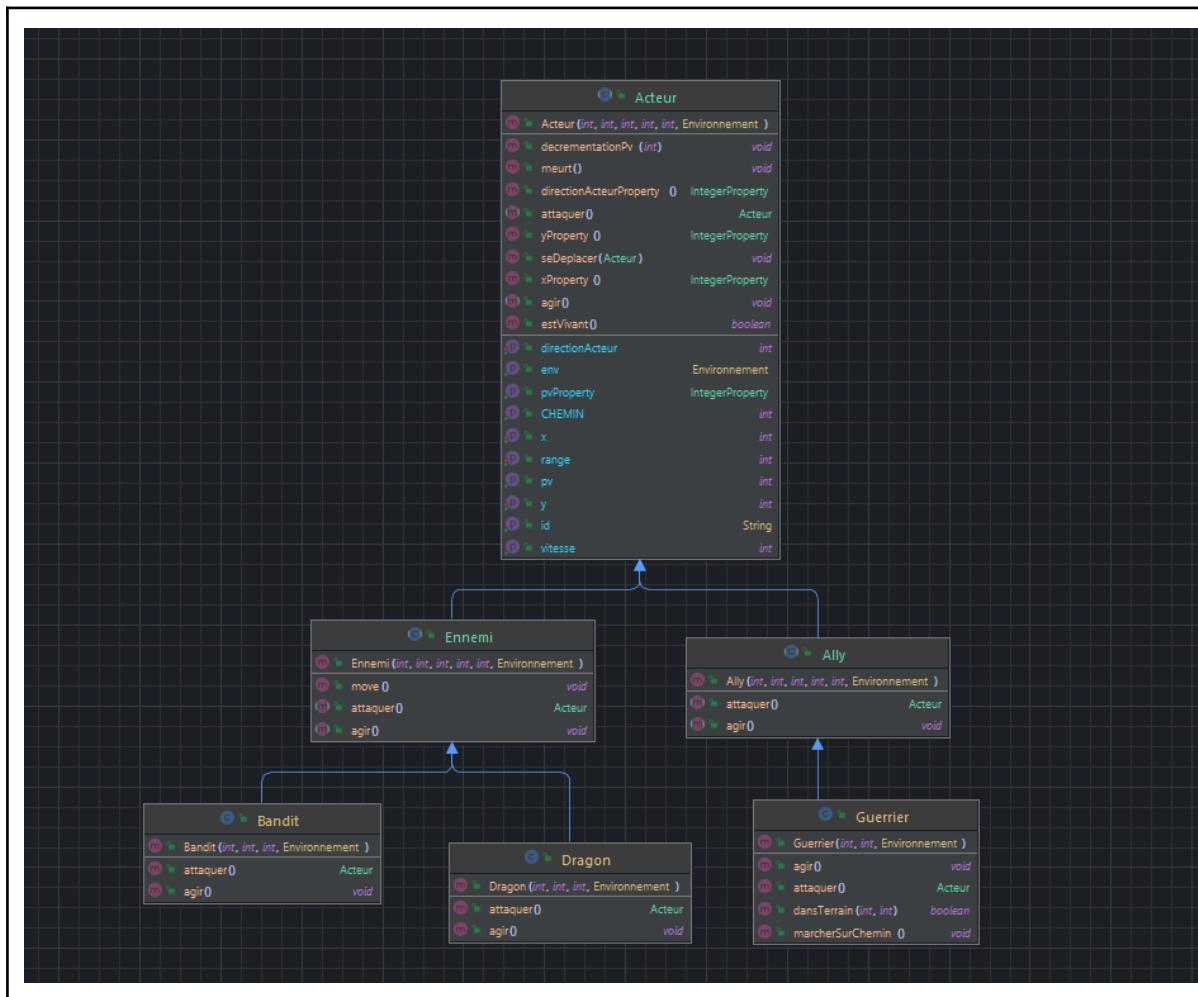
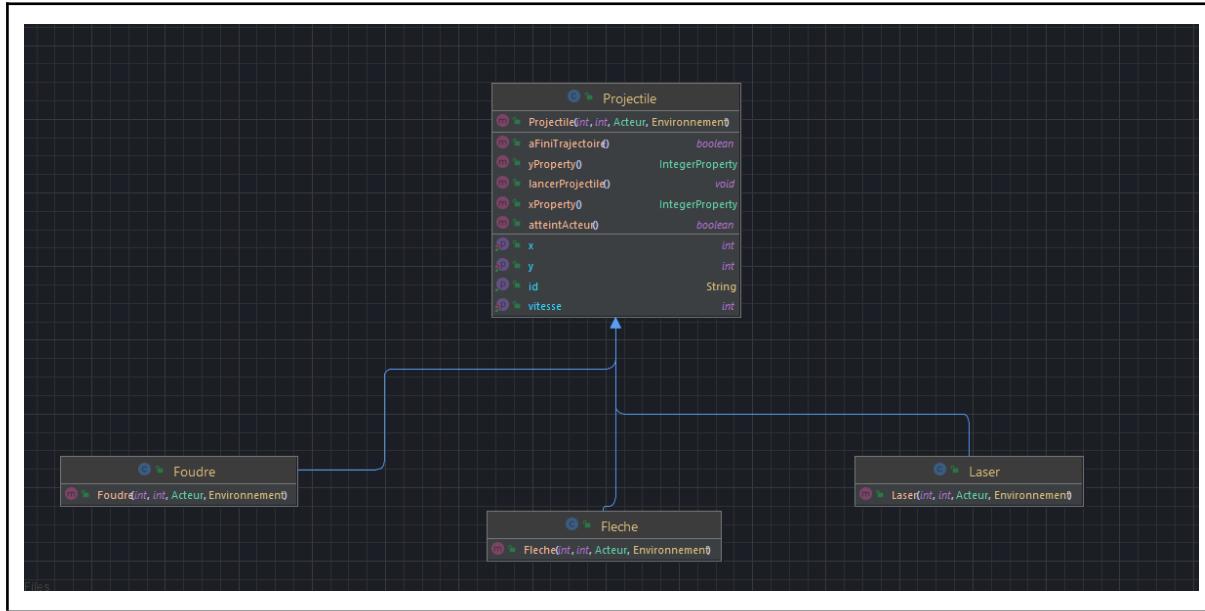


Diagramme des Projectiles:



Voici le comment sont gérés les projectiles, on a une superclasse abstraite **Projectile** qui gère tout, nous avons décidé de procéder ainsi, car nos méthodes peuvent s'appliquer à tous les projectiles, la seule chose qui change d'un projectile à un autre, c'est la vitesse, et cela est automatiquement géré à la création du projectile dans le constructeur.

4 SAE S2.02

Structures de données :

Dans la classe BFS, les listes de structures de données sont :

1. **Map<Sommet, Set<Sommet>>** : Il s'agit d'une Map où chaque clé est un sommet (Sommet) et chaque valeur est un Set de sommets. Cette structure de données est utilisée pour représenter la liste d'adjacence
2. **ObservableList<Sommet>** : Il s'agit d'une liste observable de sommets. Cette structure de données est utilisée pour stocker les obstacles
3. **Map<Sommet, Sommet>** : Il s'agit d'une Map où chaque clé et chaque valeur est un sommet. Cette structure de données est utilisée pour représenter les prédecesseurs de chaque sommet.
4. **LinkedList<Sommet>** : Il s'agit d'une LinkedList de sommets. Cette structure de données est utilisée dans la méthode **algoBFS()** pour maintenir une file d'attente des sommets à visiter.
5. **HashSet<Sommet>** : Il s'agit d'un HashSet de sommets (Sommet). Cette structure de données est utilisée dans la méthode **adjacents()** pour stocker et manipuler les sommets adjacents.

Dans la classe Environnement, les listes de structures de données sont :

1. **ObservableList<Acteur>** : Il s'agit d'une liste observable d'objets de type Acteur. Cette structure de données est utilisée pour stocker les acteurs.
2. **ObservableList<Tour>** : Il s'agit d'une liste observable d'objets de type Tour. Cette structure de données est utilisée pour stocker les tours.
3. **ObservableList<Projectile>** : Il s'agit d'une liste observable d'objets de type Projectile. Cette structure de données est utilisée pour stocker les projectiles.

Algorithmique (70 points)

Dans la classe BFS, des algorithmes intéressants sont :

1. **construit()** : Initialise le graphe de recherche en parcourant toute la grille de jeu et en créant, pour chaque case, un sommet associé à un ensemble de sommets adjacents.
2. **algoBFS()** : Implémente l'algorithme de recherche en largeur (BFS) pour explorer le graphe depuis le sommet source vers tous ses voisins, enregistrant le chemin parcouru dans la variable prédecesseur.
3. **adjacents(Sommet s)** : Renvoie un ensemble de sommets adjacents accessibles depuis le sommet S, en tenant compte des obstacles et des restrictions de type de terrain.

Dans la classe Projectile, des algorithmes intéressants sont :

1. **lancerProjectile()**: Calcule la direction du projectile en direction de l'acteur cible, déplace le projectile selon cette direction à une vitesse définie, vérifie si le projectile a atteint sa cible et, le cas échéant, inflige des dégâts à l'acteur.
2. **atteintActeur()**: Vérifie si le projectile a atteint l'acteur cible (hitbox) en comparant les coordonnées des quatre coins du projectile avec celles de l'acteur, et renvoie un booléen indiquant si une collision s'est produite.

Dans la classe Tour, des algorithmes intéressants sont :

1. **attaqueEnnemi()** : Il s'agit d'une méthode abstraite qui sera implémentée dans les sous-classes de Tour. Elle est responsable de l'attaque des ennemis par la tour.
2. **ennemiPlusProche()** : Trouve l'ennemi le plus proche dans le rayon d'attaque de la tour en parcourant tous les ennemis dans l'environnement et en comparant leur distance à la tour.

Dans la classe TroopTour, des algorithmes intéressants sont :

1. **attaqueEnnemi()**: Cette méthode est l'implémentation spécifique de la méthode attaqueEnnemi() de la classe Tour pour TroopTour. Plutôt que d'attaquer directement les ennemis, cette tour génère une nouvelle instance de Guerrier sur le champ de bataille à des intervalles de temps spécifiés (toutes les 15 unités de temps). Les Guerrier sont placés le long d'un chemin spécifié dans la carte de l'environnement.

Dans la classe Acteur, des algorithmes intéressants sont :

1. **seDeplacer(Acteur a)** : Cette méthode déplace l'acteur dans l'environnement en fonction de la position d'un autre acteur passé en argument. L'acteur se déplace vers la position de l'autre acteur à une vitesse définie, et si l'acteur atteint une certaine distance de l'autre acteur, il change sa direction à une valeur spécifiée pour représenter un combat.

Dans la classe Bandit, des algorithmes intéressants sont :

1. **attaquer()** : Cette méthode est conçue pour que l'acteur "Bandit" identifie un autre acteur du type "Ally" dans son environnement immédiat. Si un allié se trouve à portée de l'acteur "Bandit", c'est-à-dire dans une plage spécifique autour de sa position actuelle, alors cet allié est renvoyé comme l'acteur à attaquer. Autrement, si aucun allié n'est à portée, la méthode renvoie nul.

Dans la classe Ennemi, des algorithmes intéressants sont :

1. **move()** : cette méthode est responsable du déplacement de l'ennemi le long d'un chemin prédéfini. L'ennemi se déplace d'un point à l'autre sur le chemin et sa direction est mise à jour en fonction du mouvement. Si l'ennemi atteint la fin du chemin, il est retiré de l'environnement et une certaine quantité de points de vie est retirée de l'environnement.

Dans la classe Guerrier, des algorithmes intéressants sont :

1. **marcherSurChemin()** : cette méthode permet au guerrier de se déplacer sur le chemin défini dans l'environnement. Le guerrier choisit aléatoirement une direction (haut, droite, bas, gauche) et vérifie si la case suivante sur le chemin est accessible. S'il peut se déplacer, il met à jour sa position et sa direction en conséquence.
2. **attaquer()** : cette méthode vérifie si un ennemi se trouve à portée d'attaque du guerrier. Si c'est le cas, il renvoie l'ennemi ciblé. Sinon, il renvoie nul.

Dans la classe PoserTour, des algorithmes intéressants sont :

1. **MettreEnPlaceTourDeplacable()** : Cette méthode configure un bouton de tour pour être déplaçable et lie sa désactivation à la disponibilité de suffisamment de pièces pour acheter la tour.
2. **MettreEnPlaceZoneDepot()** : Cette méthode définit la zone de dépôt des tours dans le panneau de jeu, permettant d'accepter les objets glissés-déposés. Elle extrait les informations de l'objet et vérifie si l'emplacement de la tour est valide dans la carte avant de créer une nouvelle instance de la classe de tour correspondante et de l'ajouter à l'environnement du jeu.

Dans la classe VagueEnnemie, des algorithmes intéressants sont :

1. **creeVague()** : Cette méthode génère une vague d'ennemis dans l'environnement à intervalles réguliers. Elle utilise un minuteur pour exécuter le code à intervalles prédéfinis. La méthode choisit aléatoirement entre deux types d'ennemis (Bandit ou Dragon) et crée une instance correspondante. L'ennemi est ensuite ajouté à l'environnement à l'aide de Platform.runLater() pour s'assurer que l'opération est exécutée sur le thread d'application JavaFX. Le nombre d'ennemis créés dépend du nombre de vagues en cours.

Dans la classe Environnement, des algorithmes intéressants sont :

1. **readMap()** : Cette méthode permet de charger la configuration de la carte à partir d'un fichier externe. Elle lit les données du fichier ligne par ligne et les stocke dans la structure de données map, qui est un tableau à deux dimensions représentant la carte du jeu. Cette méthode utilise des concepts de lecture de fichiers, de manipulation de chaînes et de conversion de types de données.
2. **unTour()** : Cette méthode représente une itération d'un tour de jeu. Elle effectue les actions nécessaires pour faire avancer le jeu d'un tour, telles que le déplacement des acteurs, les attaques des tours et le lancement des projectiles. Elle vérifie également s'il y a des ennemis sur le terrain et crée une nouvelle vague d'ennemis si nécessaire. Cette méthode utilise des boucles et des conditions pour effectuer les actions sur les différents acteurs, tours et projectiles du jeu.

5 Document utilisateur (40 points pour SAE S2.05) :

- description du jeu (son objectif, son univers...)
- Présentation du jeu avec captures d'écran.
- Mécanique des actions utilisateurs
- Caractéristiques des entités du jeu (les tours et les ennemis) et leur dépendances éventuelles.
- Toute fonctionnalité définie et utilisable doit être documentée. A l'inverse, ne pas documenter les fonctionnalités non encore utilisables.

Univers :

On est en 1770, suite à la mort du roi Maverick Greedlock par les soldats du roi Lesion Castle, une tension est présente entre les deux familles, quelques année plus tard , la famille Greedlock décide de se venger et de lancer un assaut sur le château des Castle.

Description & objectif :

Tower Defense Team est un jeu 2D où le but est de survivre aux vagues d'ennemis qui apparaissent en bas de la carte et qui essaient de traverser le chemin. Pour les empêcher, vous devez acheter des tours, toutes ayant des attaques différentes. Pour les acheter, vous devez posséder l'argent nécessaire et une fois la somme cumulée, il suffit de cliquer et de glisser sur les emplacements désignés sur la carte, car oui, on ne peut pas poser ces tours n'importe où. Une fois la partie commencée, vous débutez avec une certaine somme qui vous permet de bien démarrer. Ensuite, pour gagner l'argent nécessaire, il faut tuer des ennemis, chaque ennemi tué donne 10 pièces. Une fois que vous avez abattu toutes les vagues de sbires, vous avez gagné, mais faites attention, si un certain nombre de sbires arrivent au bout du chemin, vous aurez perdu tous vos PV et aurez perdu la partie. Mais pas d'inquiétude, vous pouvez recommencer autant de fois que vous voulez.

Lancement du jeu :

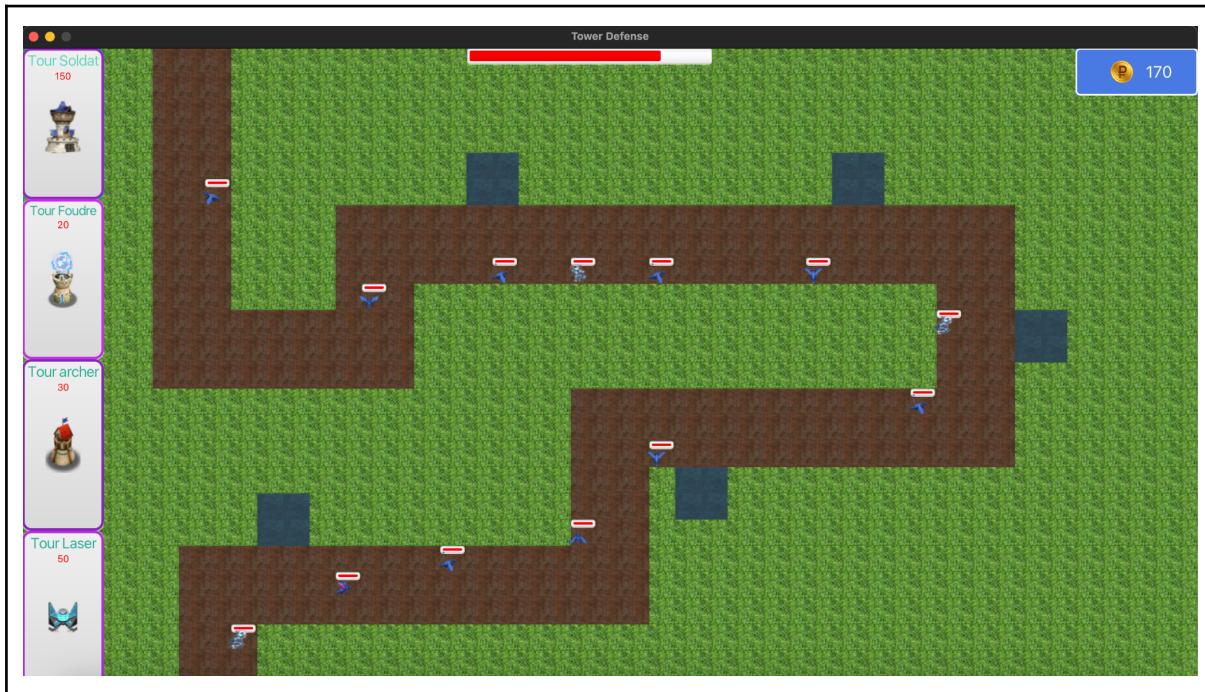
Nous avons un menu qui nous permet de lancer et de quitter si on souhaite :



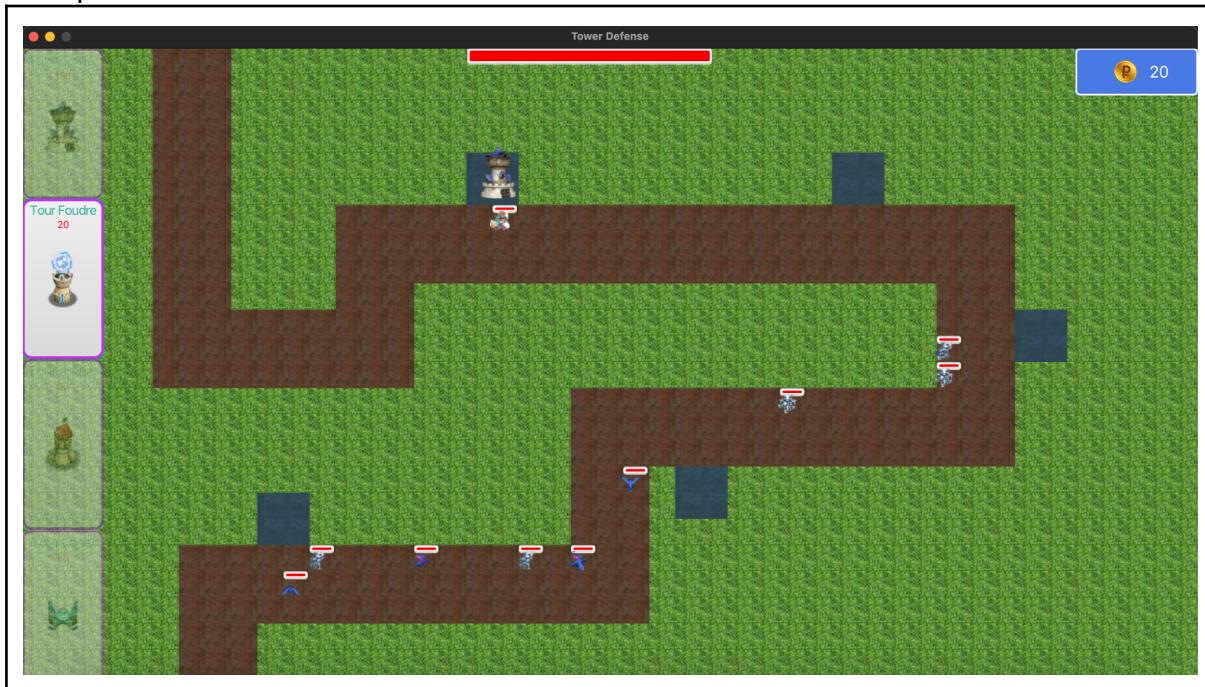
Une fois le bouton Lancer appuyé, le jeu de lance et nous pouvons jouer :



Les ennemis apparaissent et essayent d'atteindre le bout que l'on a entouré en rouge, quand un ennemi atteint ce sommet là, nous perdons 10 PV, sachant qu'on en a 100 au lancement :

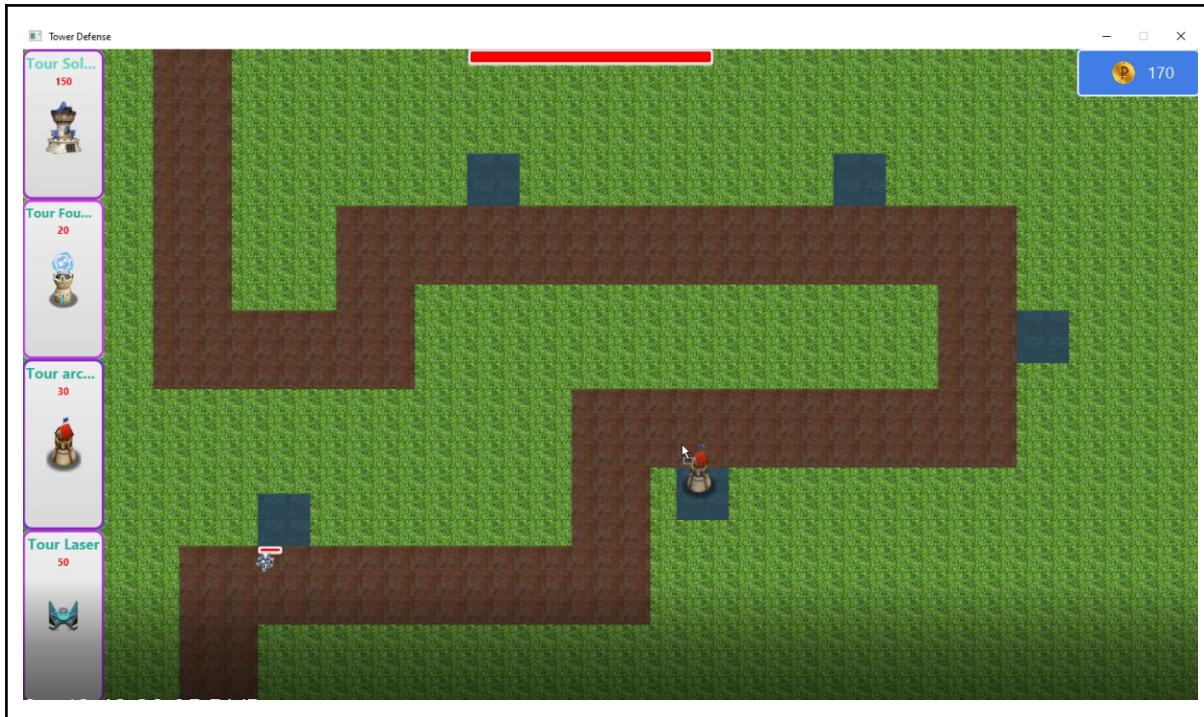


Pour conter cela, le joueur doit poser des tours qui coutent un certain prix, une fois la tourelle posée, le joueur se voit soustraire le cout de cette dernière de son argent affiché en haut à droite. Si on a pas assez d'argent, les tourelles qu'on peut pas acheter son 'grisée' et donc pas achetables.



Ici, nous avons posé une tour à soldats qui fait sortir des soldats, cette dernière coûte 150 pièces donc nous nous retrouvons à 20 pièces étant donné que l'on commence à 170 ($170 - 150 = 20$).

Les tourelles sont posée avec un système de drag and drop qui vérifie que l'emplacement est bon :



Les soldats qui sortent de cette tour se dirigent vers les ennemis qui s'approchent de leur range et les attaquent.



Si l'ennemi meurt durant ce combat, on gagne un certain nombre de pièces, on en gagne aussi en tuant les ennemis avec toutes nos tourelles à projectiles.



Quand on a tué tous les ennemis des 10 vagues, nous gagnons le jeu :



Si on perd tous nos PV, un menu de défaite s'affiche et nous proposent de relancer une partie si on le souhaite.



Pourcentage de participation de chaque étudiant :

