



UNIVERSITÉ  
DE LORRAINE

IUT nancy Charlemagne  
Informatique

# Étude préalable du projet

**Membres :**

- BORTOLOTTI Florian
- BOURDON Marin
- DUCHÈNE Éloi
- ROTH Tristan

**Titre : Tower defense et Machine Learning****Tuteur : Vincent THOMAS****Année : 2024/2025**

<b>Présentation du projet :</b>	<b>3</b>
<b>Liste des fonctionnalités du système :</b>	<b>3</b>
<b>Acteurs, cas d'utilisation et diagramme :</b>	<b>4</b>
<b>Diagramme de CU de l'utilisateur :</b>	<b>5</b>
<b>Diagramme de CU des ennemis et des défenses :</b>	<b>6</b>
<b>Scénario : Déployer des ennemis...</b>	<b>6</b>
<b>Définition des caractéristiques :</b>	<b>7</b>
Le type :	7
Les comportements :	7
La vie :	7
La vitesse de déplacement :	7
Les dégâts :	8
La vitesse d'attaque :	8
<b>Gestion de l'évolution :</b>	<b>8</b>
<b>Diagramme de classes :</b>	<b>9</b>
<b>Diagramme de séquence système :</b>	<b>13</b>
<b>Diagramme d'activité : déroulement de la partie</b>	<b>14</b>
<b>Diagramme d'état : état des défenses</b>	<b>14</b>
<b>Recensement et évaluation des risques :</b>	<b>15</b>
<b>Planning :</b>	<b>16</b>
Itération 1 :	16
Itération 2 :	17
Itération 3 :	18
Itération 4 :	19
Itération 5 :	19
Itération 6 :	19
Itération 7 :	20

# Présentation du projet :

## **Vision du produit :**

### **1. Qui va acheter/utiliser le produit ? Qui est la cible ?**

- N'importe qui, c'est un jeu que l'utilisateur lance pour regarder ce qu'il se passe, il observe le comportement des IA qui changent et apprennent d'elles-mêmes. Éventuellement des personnes qui sont passionnées par l'algorithmie ou par l'IA, des gens qui sont intéressés par l'apprentissage d'un agent.

### **2. À quels besoins le produit va-t-il répondre ?**

- Notre Tower Defense ne répond pas à des besoins spécifiques, il est avant tout divertissant. Il peut tout de même être un exemple interactif de l'utilisation de l'intelligence artificielle appliquée sur un jeu, c'est un exemple simple d'application d'algorithme d'apprentissage.

### **3. Quelles sont les fonctionnalités critiques pour répondre aux besoins de façon à avoir un produit réussi ?**

- Le plus important dans ce projet est d'obtenir une intelligence artificielle qui fonctionne correctement par apprentissage, ce point est le cœur du projet. On doit faire en sorte qu'à chaque partie lancée, son déroulement soit différent de la partie précédente. Il y a également le fait de posséder une carte fonctionnelle (point de départ, murs, chemins... définis).

### **4. Comment le produit se situe-t-il par rapport aux produits existants sur le marché ?**

- C'est quelque chose de nouveau, les jeux de tower defense traditionnels sont jouables, habituellement le but est de poser des défenses de manière stratégique pour survivre aux vagues d'ennemis. Dans notre cas, le jeu se "joue tout seul", on lance une partie et on observe les ennemis faire du mieux qu'ils peuvent face aux défenses tout en apprenant et en modifiant leur comportement d'une manche à l'autre.

### **5. Quel est le délai pour le développement et la livraison du produit ?**

- Jusqu'au stage

# Liste des fonctionnalités du système :

## ❖ Ennemis :

- Se déplacer (avec Steering Behaviours).
- Choisir un chemin à emprunter lorsque plusieurs sont possibles (avec A\*).
- Attaquer une défense (Lui infliger des dégâts, calculés en fonction du type de l'ennemi et de la défense).
- Prioriser une défense à attaquer (Par rapport aux types notamment, pour que l'ennemi soit le plus efficace possible, voire d'autres caractéristiques que les types ?).

- ❖ Défenses :
  - Attaquer un ennemi lorsqu'il entre dans sa zone de portée.
- ❖ Moteur de jeu :
  - Démarrer une partie
  - Stopper une partie en cours
  - Finir une partie lorsque x ennemis sont arrivés au point d'arrivée.
  - Finir une partie lorsque toutes les vagues sont passées sans que les ennemis aient rempli l'objectif.
  - Faire en sorte de pouvoir accélérer le temps dans une partie.
  - Quitter le jeu.
  - Afficher dans une "logs box" les informations au cours de la partie à titre informatif (et/ou éducatif) pour le joueur. Par exemple, afficher quels types d'ennemis vont s'améliorer à la fin d'une manche, que vont-ils améliorer (leur vitesse, leurs dégâts, leur déplacement...).

## Acteurs, cas d'utilisation et diagramme :

Acteur :

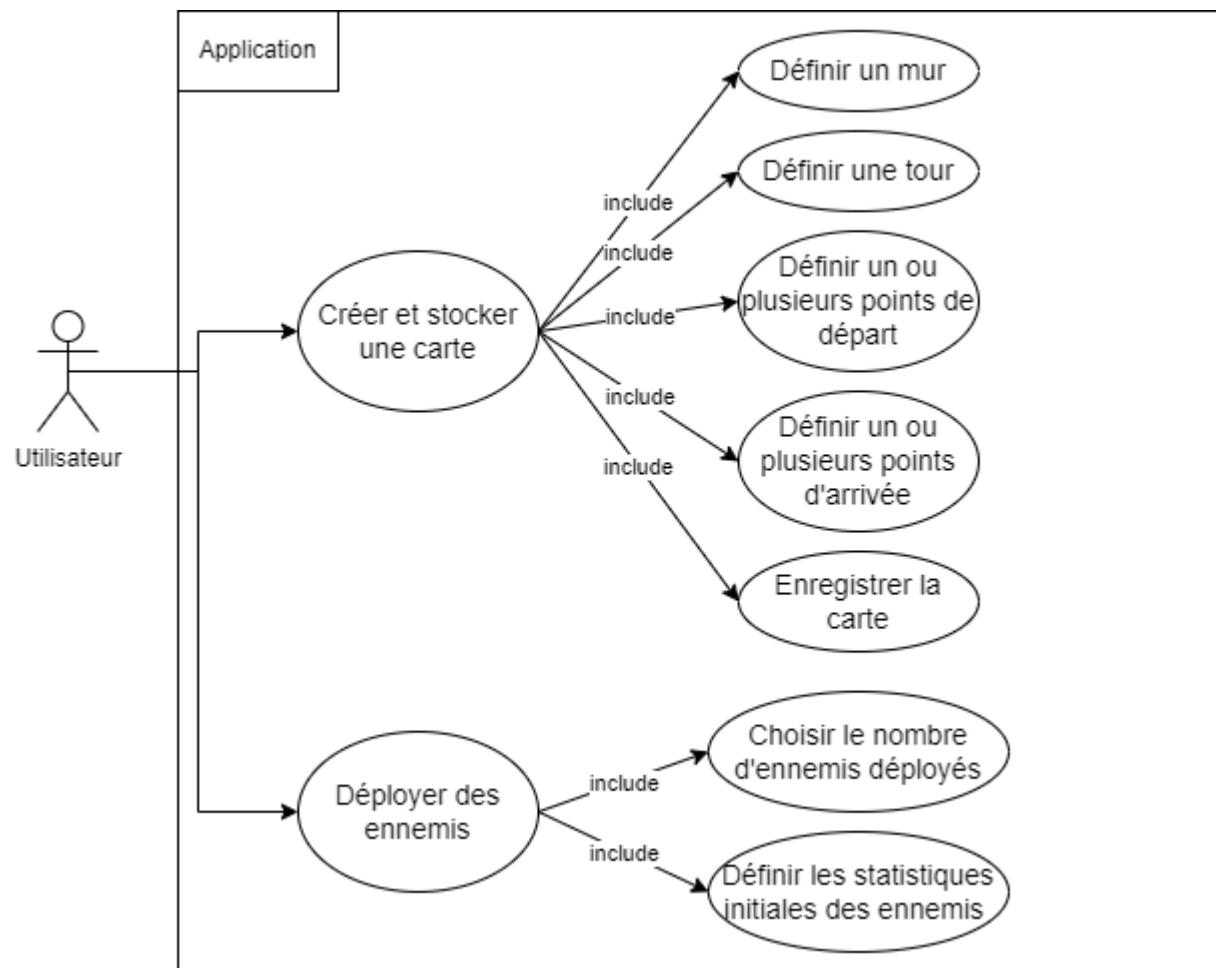
- ❖ Utilisateur
- ❖ Ennemi
- ❖ Défense

Cas d'utilisation :

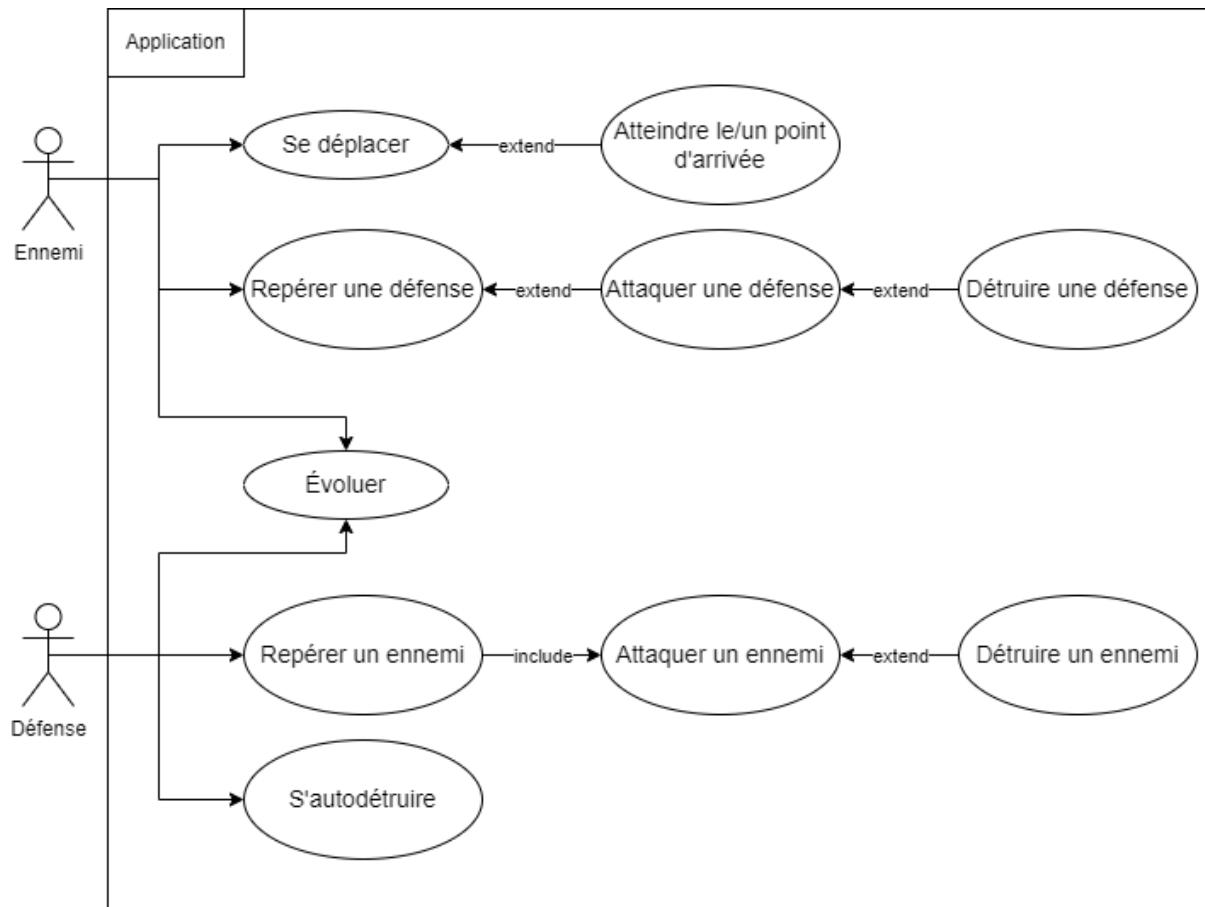
- ❖ Créer et stocker une carte
  - Définir un mur sur la carte
  - Définir une tour sur la carte
  - Définir un ou plusieurs points de départ
  - Définir un ou plusieurs points d'arrivée
  - Enregistrer la carte (txt)
- ❖ Déployer des ennemis
  - Choisir le nombre d'ennemis déployés
  - Définir les statistiques initiales des ennemis
- ❖ Se déplacer
  - Atteindre le /un point d'arrivée
- ❖ Repérer une défense
  - Attaquer une défense
    - Détruire la défense
- ❖ Évoluer
- ❖ Repérer un ennemi
  - Attaquer un ennemi
    - Détruire un ennemi

- ❖ S'autodétruire

## Diagramme de CU de l'utilisateur :



# Diagramme de CU des ennemis et des défenses :



## Scénario : Déployer des ennemis

### Préconditions :

- Avoir une carte valide

### Postconditions :

- Le nombre d'ennemis donné est déployé
- Chaque ennemi a ses statistiques définies

### Déroulement normal :

1. Donner un nombre d'ennemis à déployer
2. Donner les statistiques initiales des ennemis par rapport à un total défini

### Variantes :

- Donner un nombre d'ennemis négatif
- Indiquer un total de statistiques supérieur au total défini

### Contrainte non fonctionnelle :

- temps d'exécution raisonnable

# Définition des caractéristiques :

## Le type :

Les types sont les suivants : feu, eau, feuille.

L'ordre de victoire est eau → feu → feuille → eau. Il se peut qu'il n'y ait pas de type auquel cas l'élément concerné n'a ni faiblesse, ni force (toutes ces attaques sont x1). Si le type est un type plus fort, l'attaque voit ses dégâts augmentés 1.5 fois. Si le type est un type plus faible, l'attaque voit ses dégâts diminués 0.5 fois. Si les deux types sont les mêmes, l'attaque ne voit pas ses dégâts modifiés.

## Les comportements :

Les comportements sont les suivants : Normal, Fuyard, Healer, Kamikaze

Ils définissent pour un ennemi la manière de se déplacer :

- Normal : passe au chemin le plus court, sans considérer les tours.
- Fuyard : évite les tours, sauf obligation.
- Kamikaze : passe au chemin le plus court, en attaquant la première tour rencontrée.
- Soigneur : prend le chemin pris par le plus d'ennemis et fait un soin de zone.

Au départ, chaque comportement est associé à un personnage :

- Normal : Géant
- Fuyard : Ninja
- Kamikaze : Berserker
- Soigneur : Druide

## La vie :

- Géant : 200
- Ninja : 80
- Berserker : 100
- Druide : 100
- Tour : 500

## La vitesse de déplacement :

- Géant : Déplacement lent
- Ninja : Déplacement rapide
- Berserker : Déplacement normal
- Druide : Déplacement normal

## Les dégâts :

- Géant : Beaucoup de dégâts
- Ninja : Peu de dégâts
- Berserker : Dégâts normaux
- Druide : Soigne 3 PV par zone
- Tour : Dégâts normaux

## La vitesse d'attaque :

- Géant : Attaque lente
- Ninja : Attaque rapide
- Berserker : Attaque normale
- Druide : Attaque normale
- Tour : Attaque normale

## Gestion de l'évolution :

### - Évolution choisie pour les **ennemis** :

On récupère les 2 meilleurs ennemis pour chaque catégorie d'ennemis (Fuyard, Normal, Healer, Kamikaze).

Les meilleurs ennemis sont calculés par une formule de score :

Score = 1/distancé à l'arrivée + 0.2 \* temps de survie en secondes + (10000 si l'ennemi atteint la ligne d'arrivée)

- Si un seul ennemi dans une catégorie :

- Si il passe la ligne d'arrivée, on le garde

- Sinon, on change son type par un autre au hasard (mais pas le sien), et il sera considéré comme un enfant.

Pour chaque couple parent défini, on attribue les nouvelles stats aux ennemis de même catégorie qui sont morts en faisant des moyennes:

- vitesseEnnemiN+1 = moy(parent1, parent2)
- vieEnnemiN+1 = moy(parent1, parent2)
- typeEnnemiN+1 = typeCounter(killerType)

Enfin, on fait de la mutation pour tous les nouveaux ennemis, on va ajouter ou retirer quelques points à chaque statistique de manière aléatoire:

- Par exemple: Si on a une base de 100 de vie, on va faire vie = vie + [randomEntreXEtY(-5,5)], et on fait de même pour la vitesse.

Si le couple des 2 meilleurs ennemis d'un certain type sont seuls (c'est à dire il n'y a aucun enfant de leur catégorie à faire évoluer), alors ils

mutent eux même de la même manière, sinon les ennemis parents n'évoluent pas).

On a donc une population nouvelle contenant pour chaque catégorie d'ennemi présent : soit un parent qui a passé la ligne d'arrivée et qui sera donc certainement très efficace, soit les deux parents les plus efficaces avec des enfants mutés par rapport à leurs statistiques (qui ont donc des statistiques similaires), soit deux parents seuls qui sont mutés de la même manière.

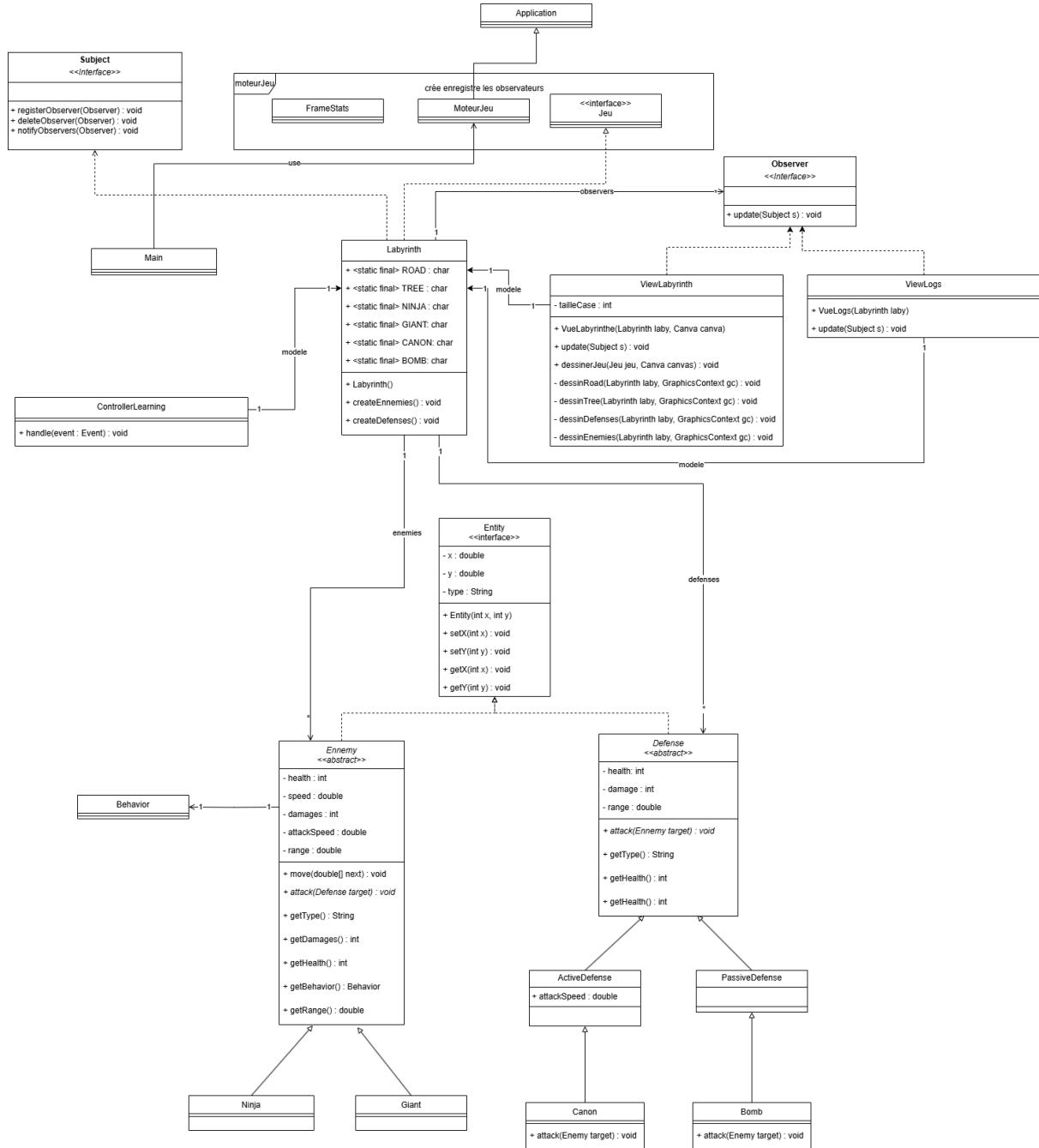
- Évolution choisie pour les **défenses** :

- On prend les meilleures défenses, celles qui ont administré le plus de dégâts, et on fait évoluer toutes les défenses sur cette base.
- On effectue une mutation qui ajoute ou retire certains degrés de caractérisation. Les variables des tours qui seront modifiées par l'évolution sont :
  - La vie
  - La vitesse de tir
  - Les dégâts
  - La zone d'attaque
  - Le type (il est choisi en prenant le type fort du plus grand nombre des ennemis de la vague précédente, par exemple s'il y a plus d'ennemis de type eau qui passent dans la zone, la tour passera type feuille.) Si la tour n'a pas de type, l'ennemi ne prend pas de type ou aura un type aléatoire.

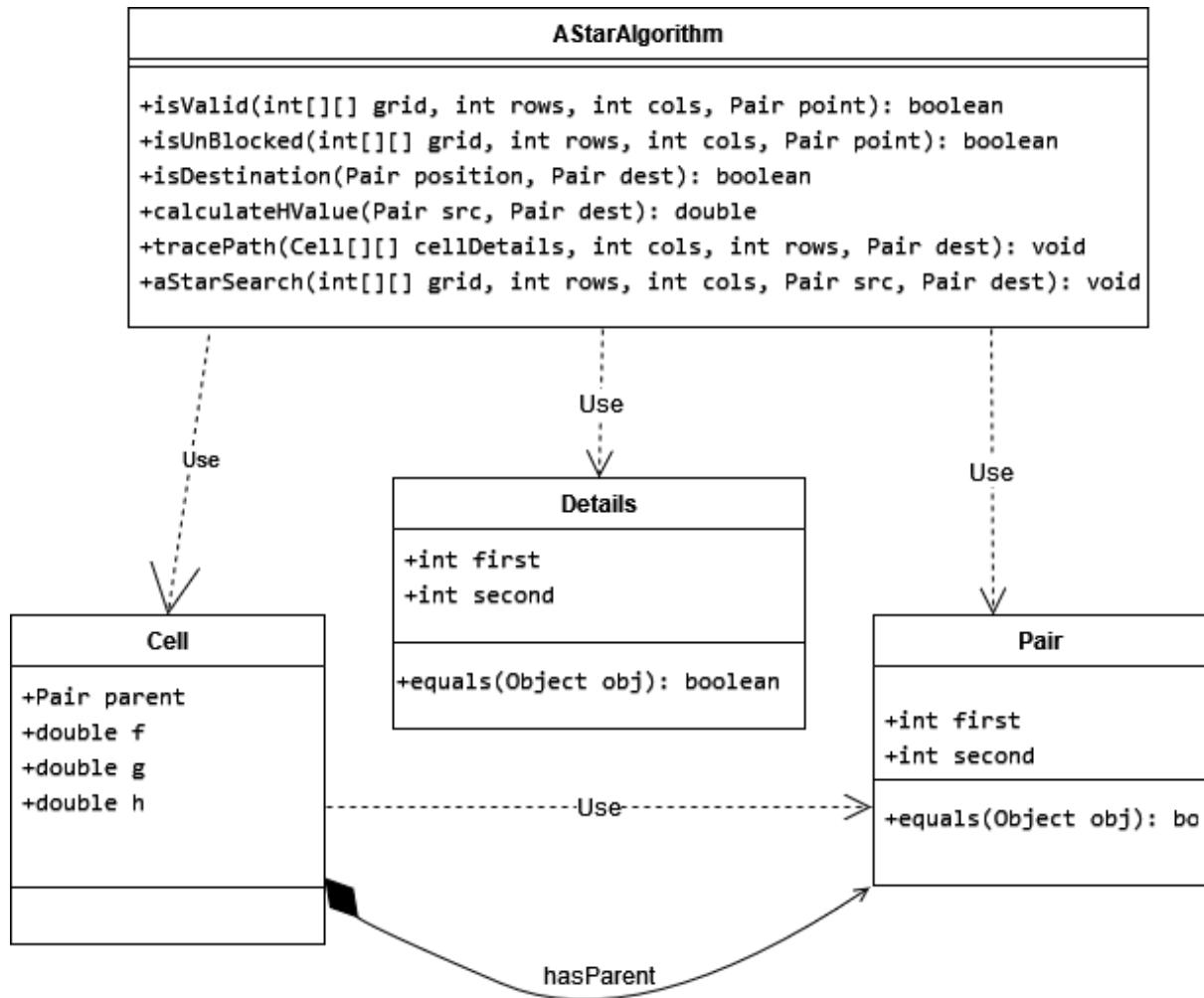
## Gestion du déplacement :

Le déplacement n'est pas évolutif, l'algorithme de déplacement (**Steering behaviors**) combiné à celui du choix de chemin (**A\***) va permettre aux ennemis de se déplacer selon le **chemin le plus optimal** et de manière réaliste en fonction d'une **courbe**. En fonction du comportement d'un ennemi, l'algorithme ne calcule pas le même chemin optimal, il prend en compte les contraintes imposées par celui-ci.

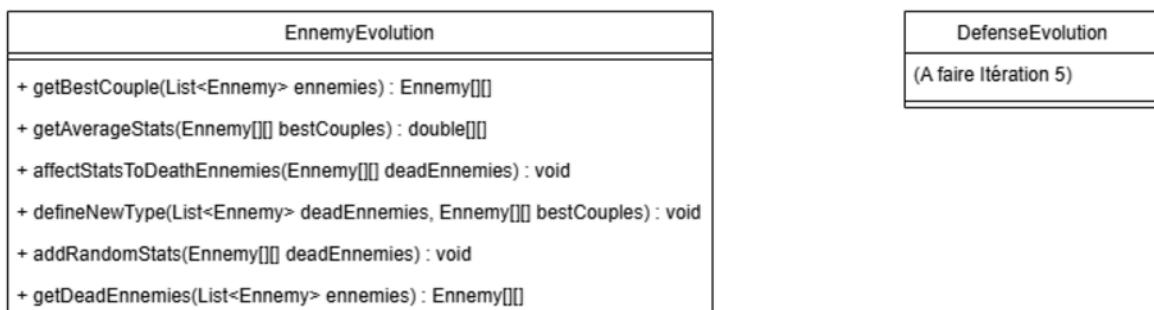
# Diagramme de classes :



### Diagramme de classe du prototype de A\* :



### Diagramme de classe du prototype de l'algorithme d'évolution des ennemis :

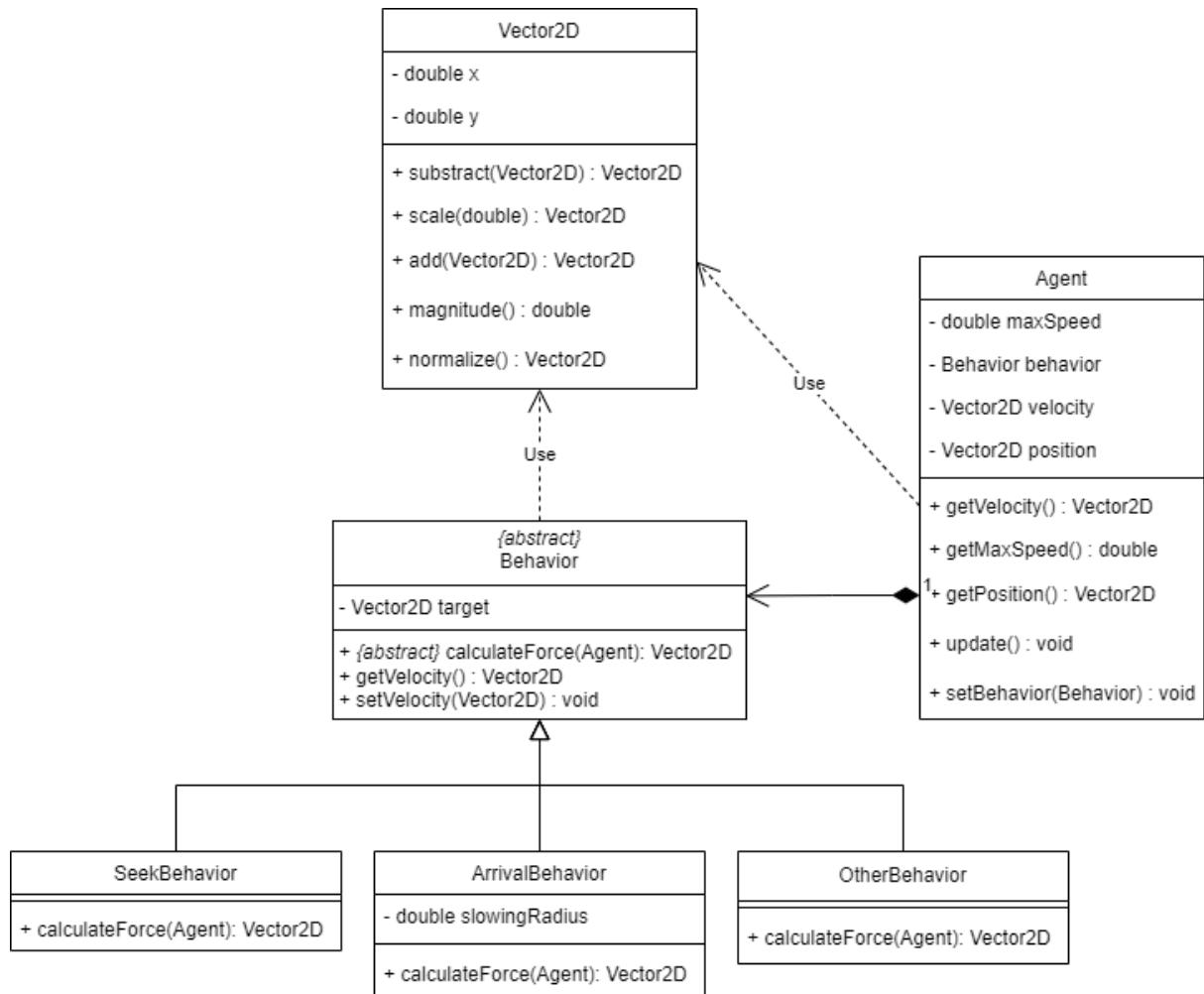


### Diagramme de classe du prototype de Steering Behaviors :

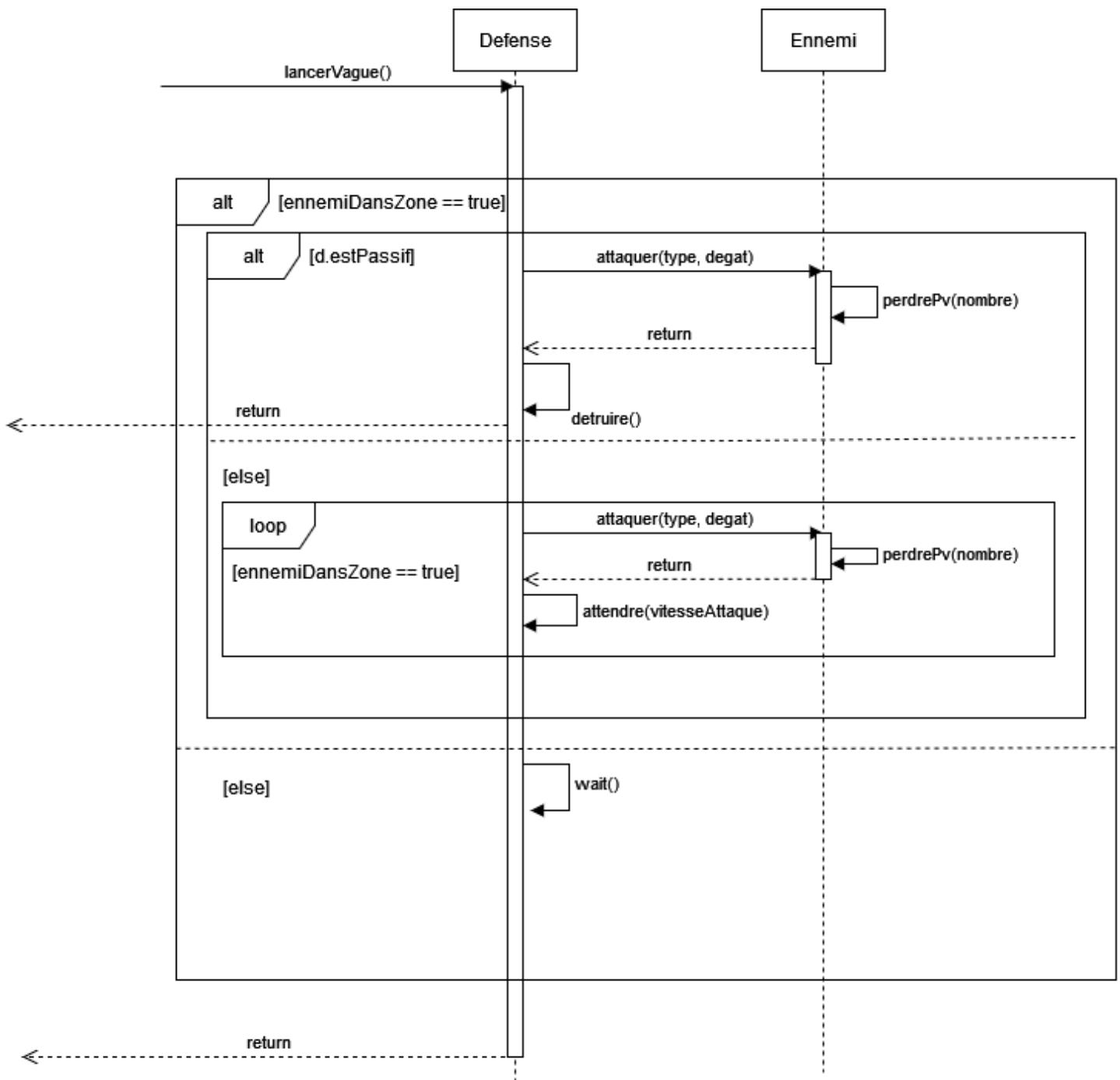
Implémentation java de steering behaviors :

<https://github.com/juniorbl/java-steering-behaviors/tree/master>

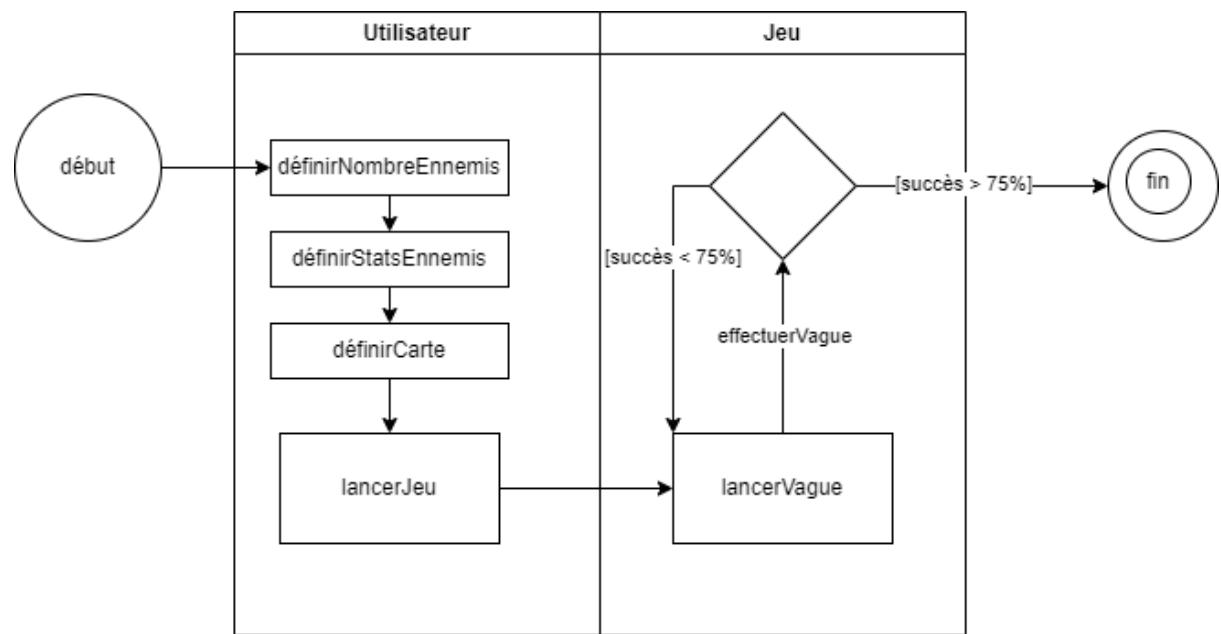
Diagramme de classe :



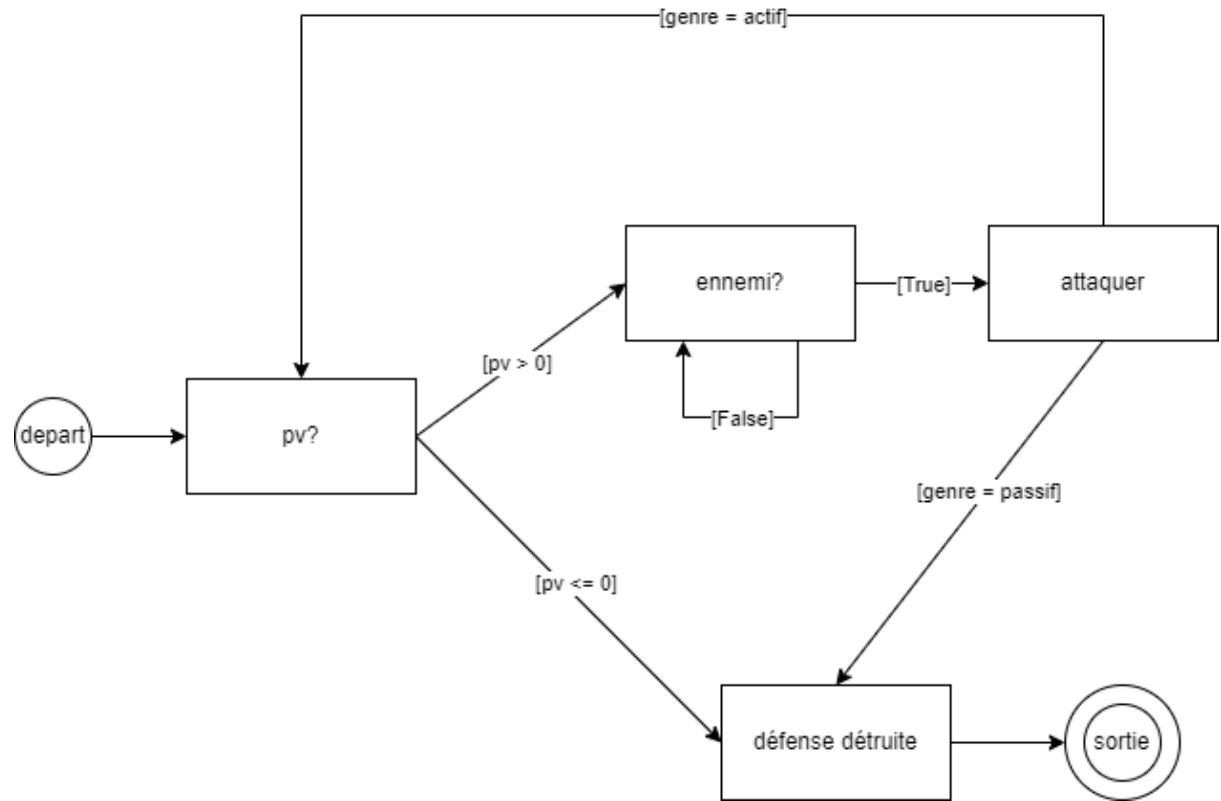
## Diagramme de séquence système :



## Diagramme d'activité : déroulement de la partie



## Diagramme d'état : état des défenses



# Recensement et évaluation des risques :

## 1. Compréhension universelle du sujet :

La mise en place d'un jeu comportant de l'IA peut se révéler très complexe, notamment à cause de la compréhension globale de tous les membres du groupe. Nous devons nous assurer que tout le monde dans le groupe a bien compris le fonctionnement de l'IA dans le jeu. C'est en effet le cœur du projet et tout le monde doit se mettre d'accord sur comment les agents vont fonctionner.

## 2. Surcharge de ressources :

Nous avons vu que chaque algorithme a une complexité, certains plus élevés que d'autres et certains avec une complexité variable. Il est donc important de prendre ces paramètres en compte et faire attention à ne pas créer des programmes trop voraces qui affectent les performances du jeu en créant des ralentissements.

## 3. Équilibrage du jeu :

Il est important de bien gérer l'équilibrage du jeu et des algorithmes, par exemple une partie ne doit pas se terminer en deux manches, car l'IA s'est beaucoup trop améliorée après la première vague.

## 4. Collaboration des algorithmes entre eux :

Notre projet comportera plusieurs algorithmes différents qui devront fonctionner ensemble, en effet, il sera important de veiller à bien pouvoir tout regrouper à la fin. On imagine que certains membres du groupe travaillent sur un algorithme et d'autres sur un autre, il faudra donc réussir à les faire fonctionner ensemble par la suite.

## 5. Unifications des outils :

Il sera important de veiller à ce que tous les membres du groupe soit à la même échelle, il serait problématique qu'un membre utilise tel ou tels outils que les autres n'utilisent pas et qu'il en résulte un conflit dans le projet. De même pour l'utilisation des librairies, si quelqu'un importe une librairie, tout le monde doit le faire aussi.

## 6. Gérer le temps :

Sans surprise, il faudra veiller à réaliser les fonctionnalités prévues dans le temps imparti, faire attention aux parties qui pourraient prendre plus de temps que prévu et ne pas se faire rattraper par le temps. C'est en ce sens qu'il est important de bien organiser le travail à faire et de s'adapter en fonction du

temps utilisé. Certaines fonctionnalités sont secondaires par rapport à d'autres et il est nécessaire de prioriser les plus importantes.

## Planning :

### Itération 1 :

- Implémentation du moteur de jeu. On peut lancer une partie avec différents ennemis et différentes défenses, toutes ces entités sont clairement représentées sur le labyrinthe.
- Prototypage pour les steering behaviors
- Prototypage de l'algorithme évolutif.
- Prototypage de l'algorithme A\*.

#### **Critères de validation :**

- Les prototypes sont fonctionnels
- Création des éléments que nous retrouverons dans le jeu (défenses, ennemis, labyrinthe).

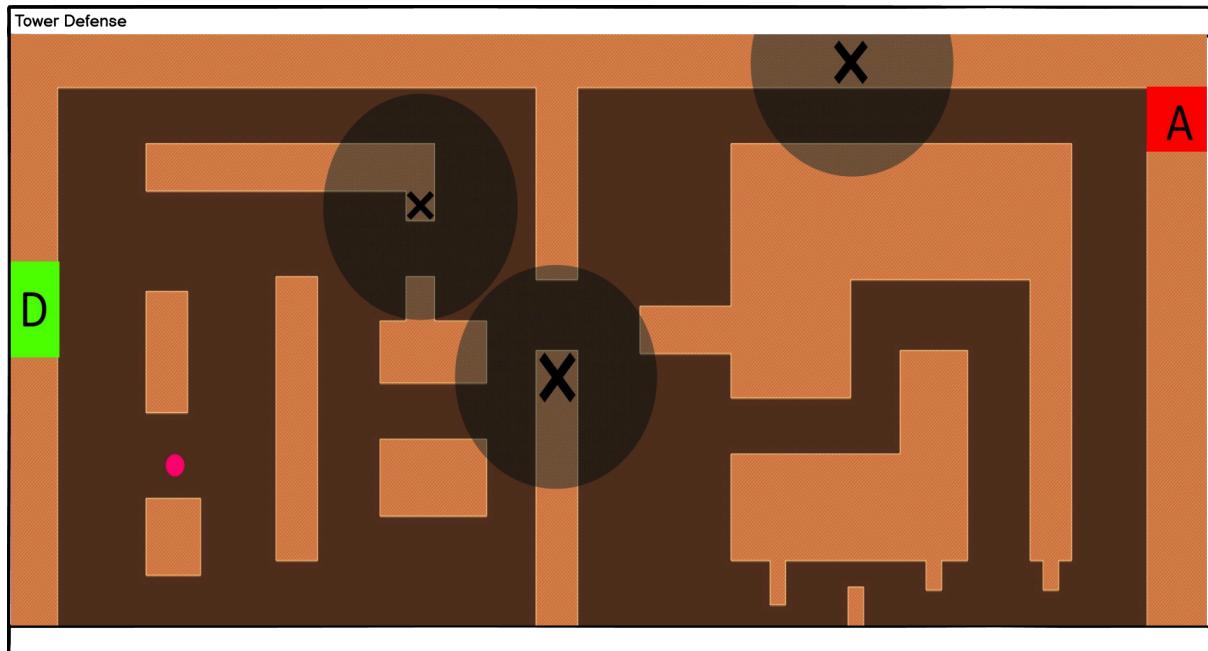
#### **Critères de validation :**

- Les ennemis peuvent se déplacer, les tours sont présentes avec leur zone, le labyrinthe se génère.

### Répartition des tâches :

- Éloi → A\*
  - Florian → Algorithme Steering Behaviors
  - Marin → Algorithme évolutif
  - Tristan → Moteur de jeu
  - Ceux qui termineront leur prototype en 1<sup>er</sup> pourront se mettre sur la création des éléments (défenses, ennemis, labyrinthe).
- 
- **Produit Minimum Viable :** Les quatre prototypes sont fonctionnels et nous pouvons générer une carte à partir d'un fichier .txt avec un ennemi qui peut se déplacer de manière autonome, mais limité.

Maquette :



## Itération 2 :

- Fusion des prototypes de l'itération 1.

### Critère de validation :

- Les prototypes fonctionnent les uns avec les autres.
- Ajout d'un panneau latéral contenant les informations du déroulement de la partie.

### Critère de validation :

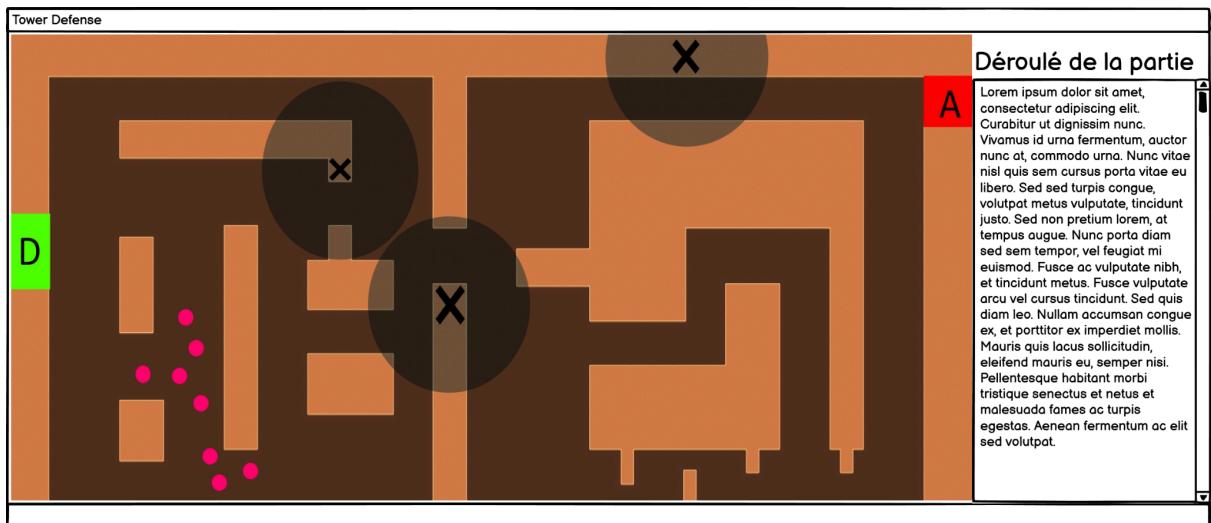
- Les informations affichées sont concordantes avec les évolutions choisies

### Répartition des tâches :

- Éloi → Fusion
- Florian → Fusion
- Marin → Fusion
- Tristan → Fusion
- Ceux qui auront terminé la fusion en premier feront le panneau latéral.

- **Produit Minimum Viable :** Les ennemis peuvent se déplacer de manière autonome en suivant un chemin défini par l'algorithme A\*. Ils évoluent entre les différentes vagues via l'algorithme évolutif. Ils évoluent seulement sur leurs caractéristiques intrinsèques (vitesse, vie).

Maquette :



## Itération 3 :

- Créations de types d'ennemis et de défenses
- **Critères de validation :**
  - Les dégâts subis sont bien modifiés selon la relation des types (x1.5 si bon type attaquant, x0.5 si mauvais type attaquant, x1 sinon)
- Création des comportements, qui vont complexifier les mouvements des ennemis.

### Critères de validation :

- Un comportement modifie bien le choix du chemin adopté par l'ennemi.
- Ajout des sprites.
- Les tours peuvent tirer lorsque les ennemis sont dans leur zone de tir.

### Critères de validation :

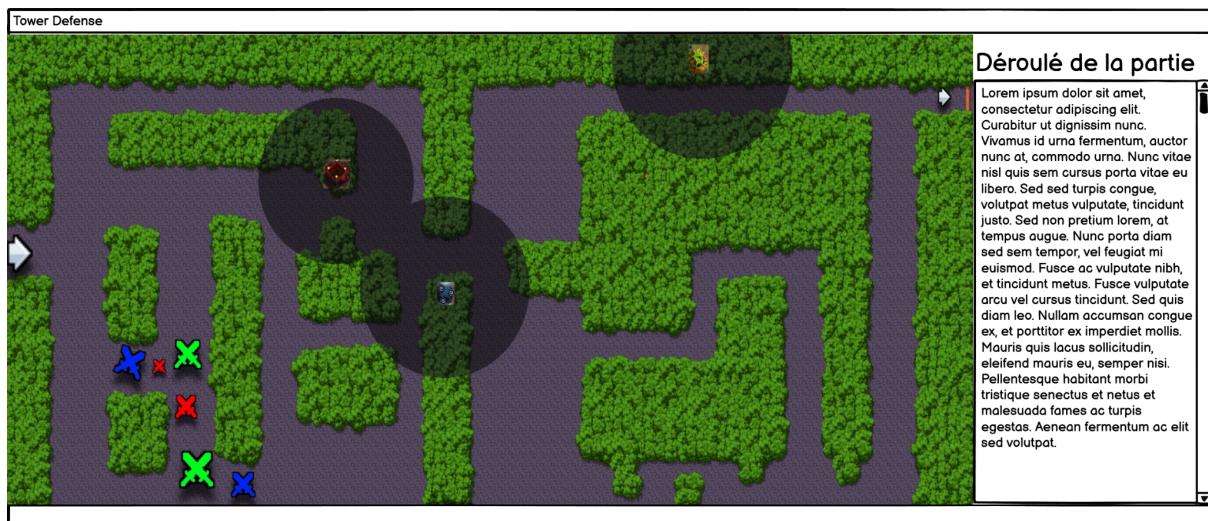
- Les dégâts des tirs sont pris en compte avec le type de la tour et de l'ennemi.
- Une tour ne peut attaquer qu'un seul ennemi à la fois
- Une tour attaque le même ennemi tant que celui-ci n'est pas mort, ou est sorti de sa portée d'attaque.
- Lors de son changement de cible, elle prend pour cible en priorité l'ennemi le plus proche de l'arrivée.

### Répartition des tâches :

- Éloi → Sprites / Types
- Florian → Comportement
- Marin → Tir des tours
- Tristan → Tir des tours

- **Produit Minimum Viable :** Les ennemis ainsi que les tours peuvent avoir des types. Les ennemis évoluent également sur les types et leur comportement.

## Maquette:



## Itération 4 :

- Nous gardons l'itération 4 pour résoudre les éventuels problèmes des itérations précédentes (exemple : hit box des ennemis, passage par un chemin qu'une défense ne couvre pas totalement). Si plus aucun problème, améliorer l'évolution des ennemis en changeant la diversité des vagues d'ennemis de manche en manche (exemple : on retire des géants pour les remplacer par des ninjas).
- Préparation de la soutenance
- **Produit Minimum Viable** : Déplacement et évolution fonctionnels

À partir de l'itération 5, nous allons commencer la “course à l'armement”, qui consiste à faire évoluer les défenses en même temps que les ennemis.

## Itération 5 :

- Adaptation de l'algorithme d'évolution pour les tours.
- Ajouts des critères de victoires et de défaites.
  - Victoire des ennemis :
    - Un certain nombre d'ennemis passent l'arrivée dans un certain nombre de vagues.
- **Produit Minimum Viable** : Défenses adaptatives

## Itération 6 :

- Résolution des erreurs et ajout des fonctionnalités pas encore pensées.
- **Produit Minimum Viable** : Produit final

## Itération 7 :

- Préparation de la soutenance finale et du document final
- **Produit Minimum Viable :** Version finale avec toutes les fonctionnalités