# Étude préalable

## Tower defense & machine learning

**Tuteur : THOMAS Vincent**

BOURDON-BORTOLOTTI-DUCHENE-ROTH

# Sommaire

# Présentation du projet

**Vision du produit :**

1. Qui va acheter/utiliser le produit ? Qui est la cible ?
2. À quels besoins le produit va-t-il répondre ?
3. Quelles sont les fonctionnalités critiques pour répondre aux besoins de façon à avoir un produit réussi ?
4. Comment le produit se situe-t-il par rapport aux produits existants sur le marché ?
5. Quel est le délai pour le développement et la livraison du produit ?

# Liste des fonctionnalités

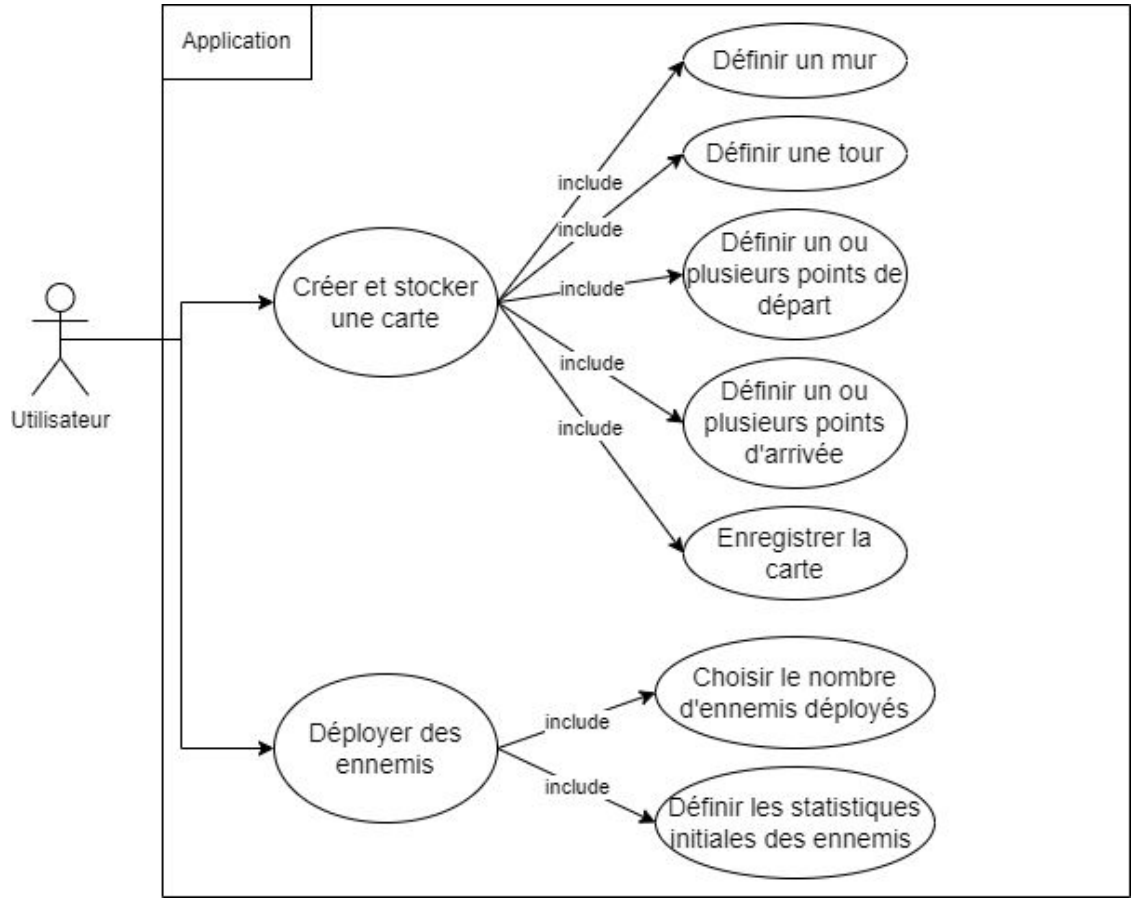
- Ennemis
- Défenses
- Moteur de jeu

# Acteurs et cas d'utilisations

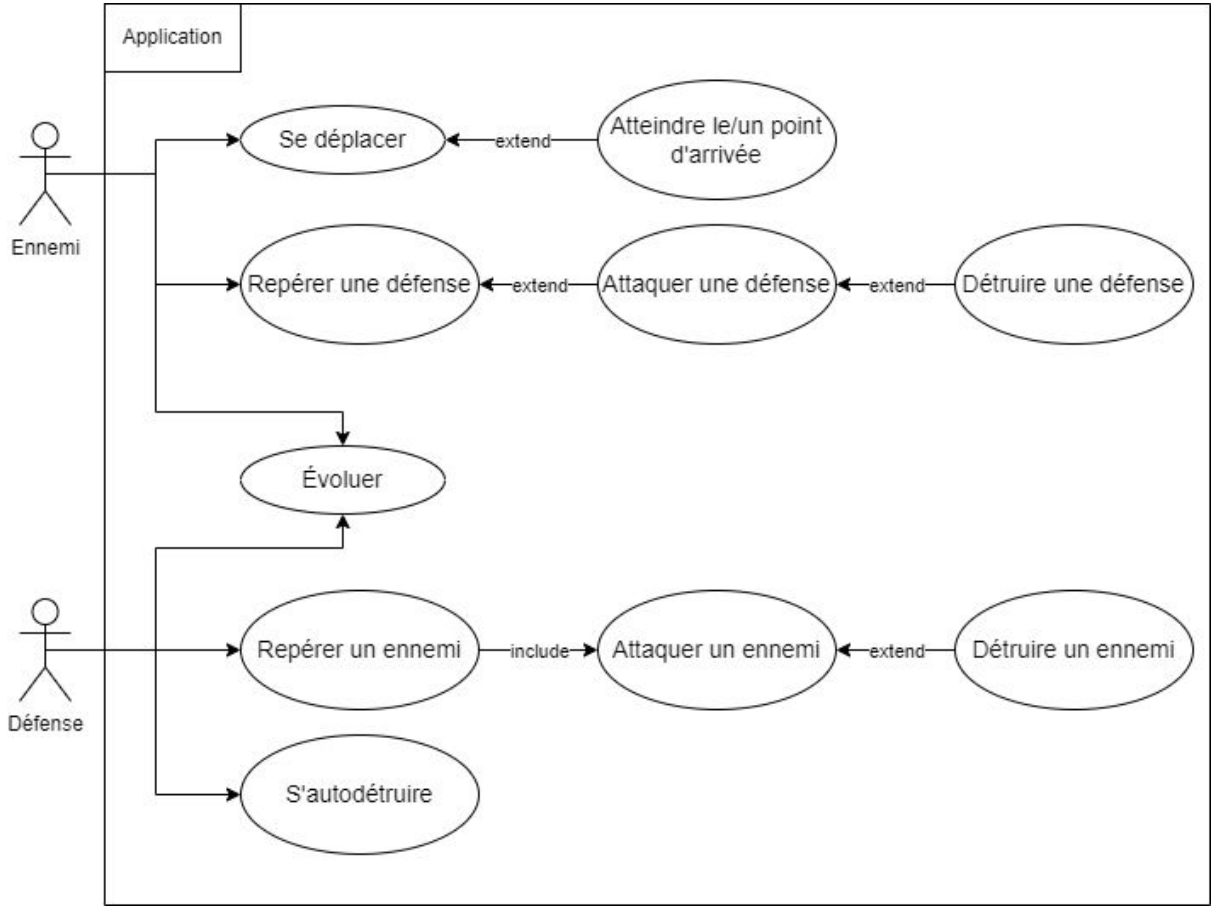
- Utilisateur
- Ennemi
- Défense

# Diagrammes

# Diagramme de CU de l'utilisateur :



# Diagramme de CU des ennemis et des défenses :



# Scénario

Déployer des ennemis

⇒ Préconditions

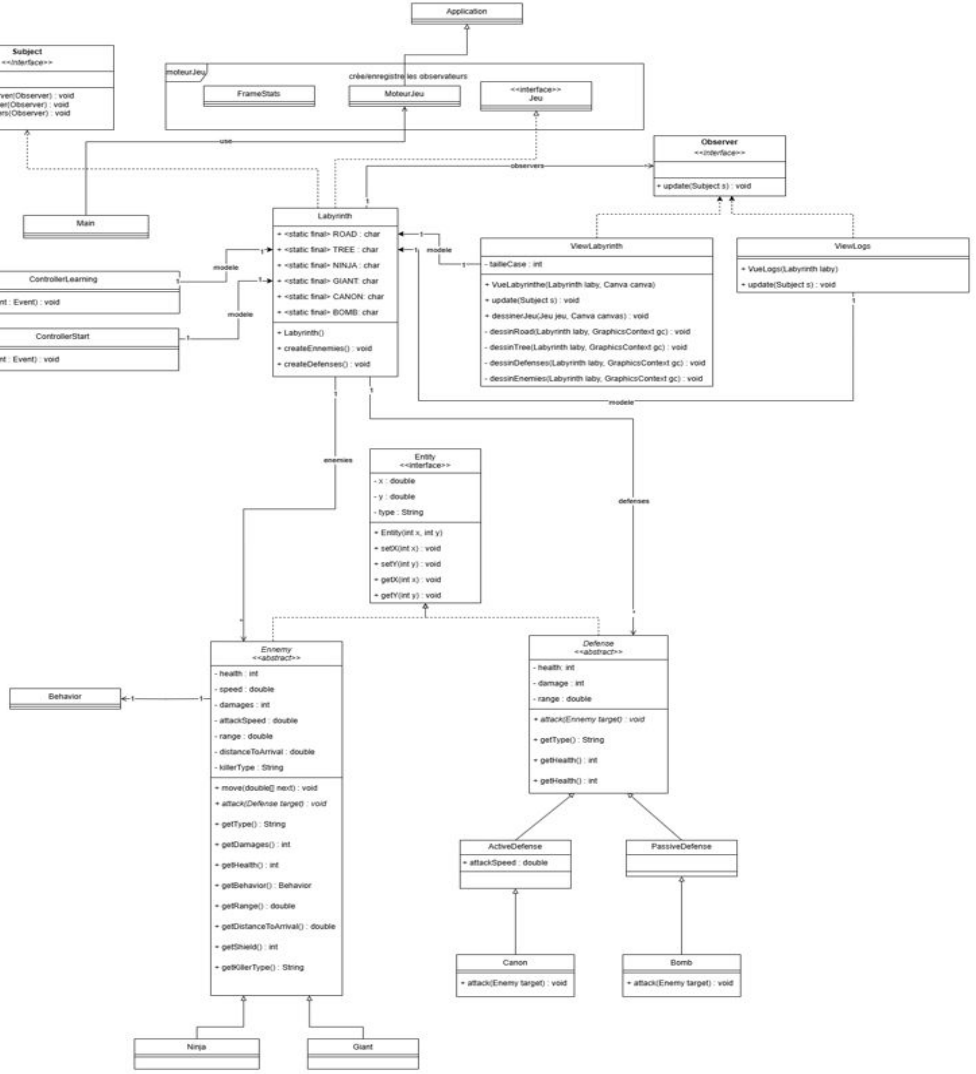
⇐ Postconditions

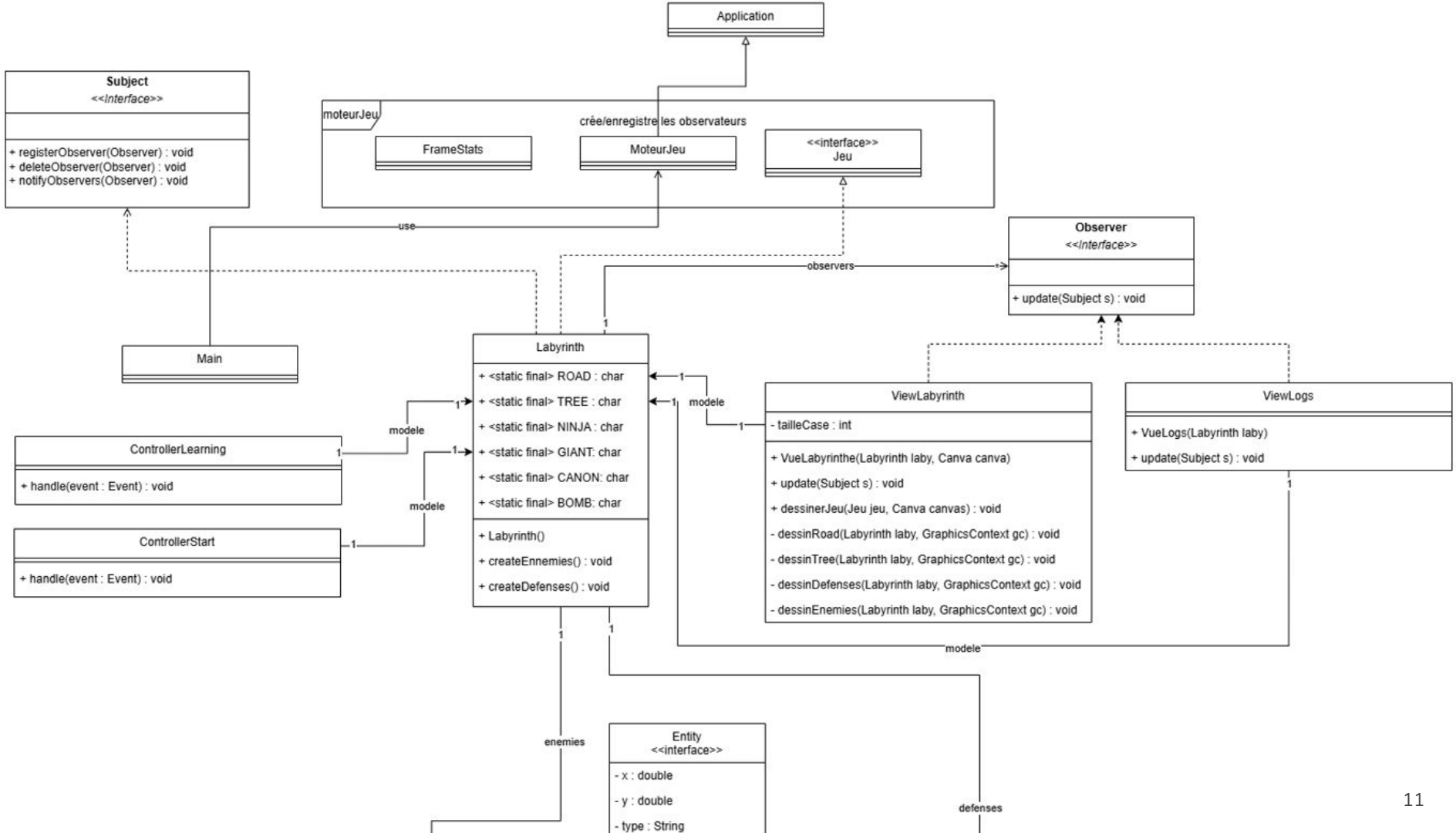
✓ Déroulement normal

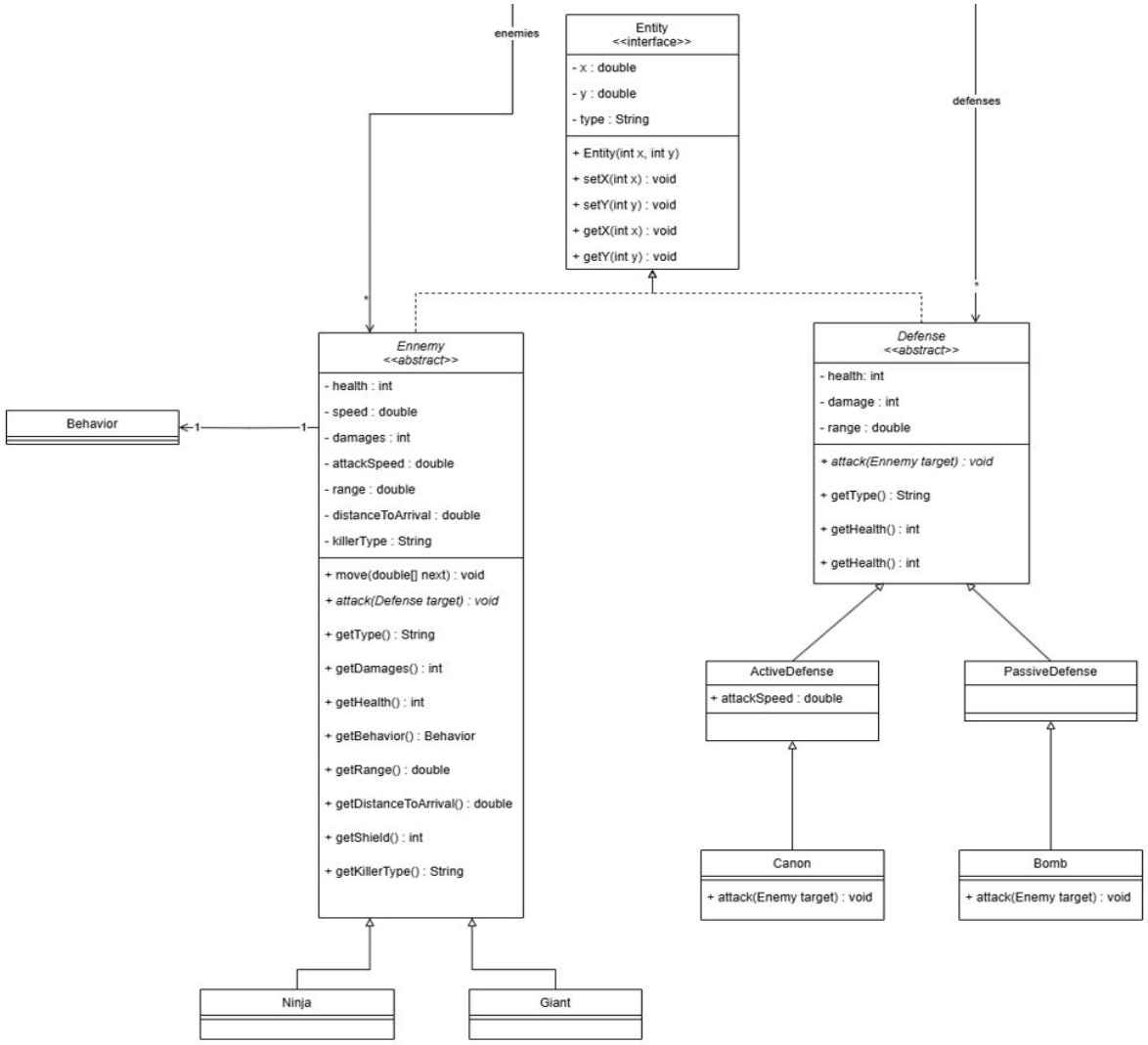
✕ Variantes

⚙ Contrainte non fonctionnelle

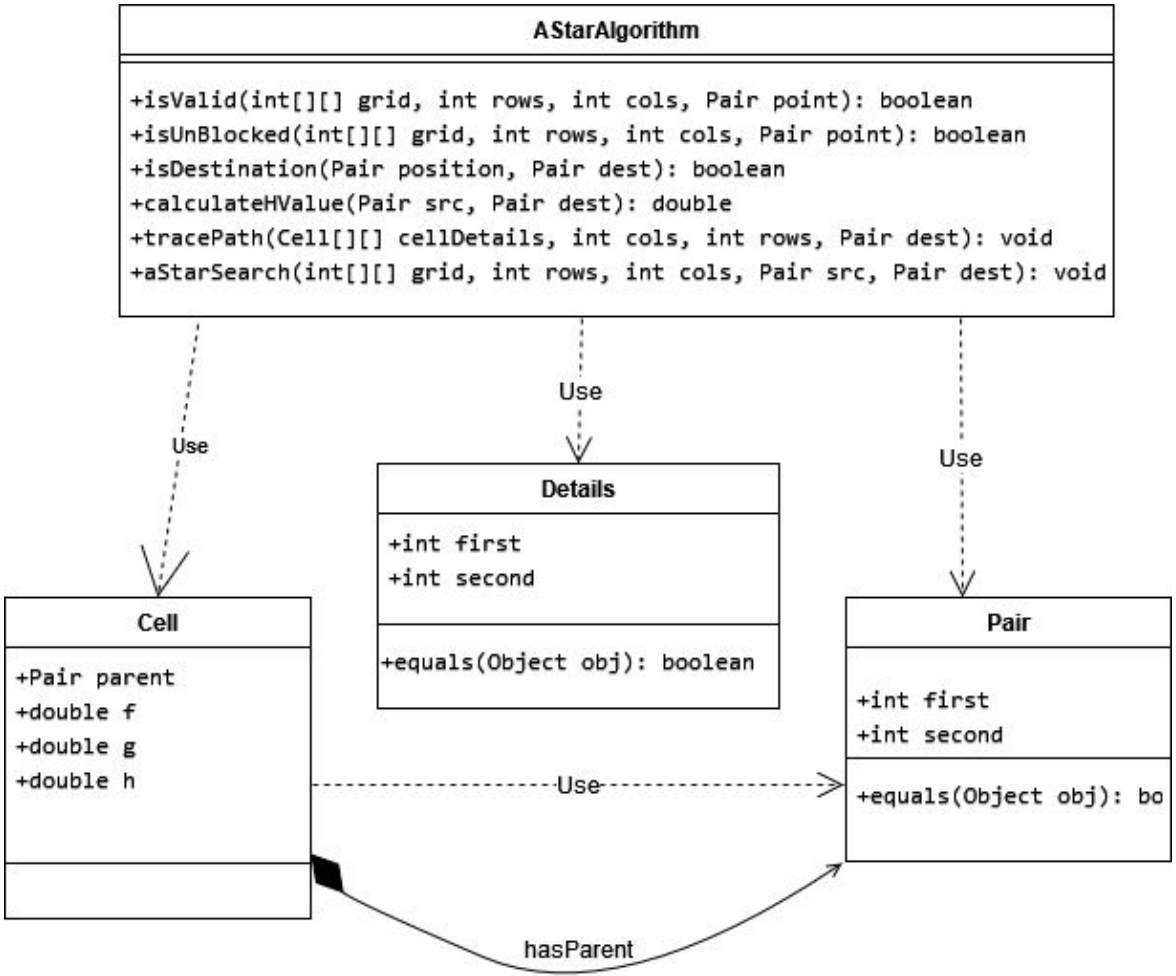
# Diagramme de classes sans les algorithmes



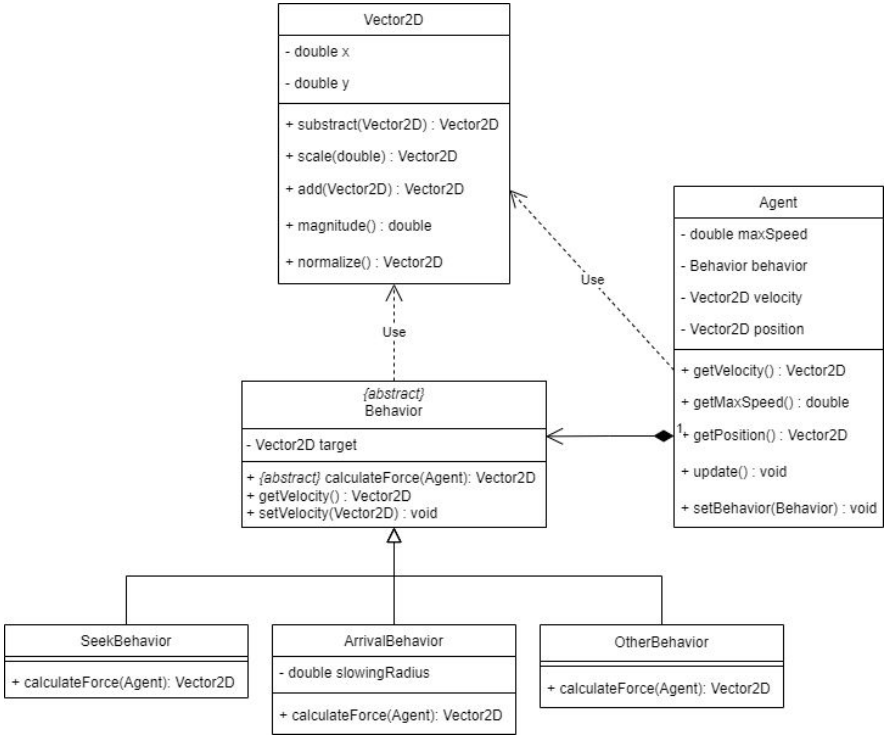

Application
Subject
<<Interface>>
+ registerObserver(Observer) : void
+ deleteObserver(Observer) : void
+ notifyObservers(Observer) : void
moteurJeu
crée/enregistre les observateurs
FrameStats
MoteurJeu
<<interface>>
Jeu
use
Observer
<<Interface>>
+ update(Subject s) : void
observers
Main
Labyrinth
+ <static final> ROAD : char
+ <static final> TREE : char
+ <static final> NINJA : char
+ <static final> GIANT: char
+ <static final> CANON: char
+ <static final> BOMB: char
+ Labyrinth()
+ createEnnemies() : void
+ createDefenses() : void
modele
ViewLabyrinth
- tailleCase : int
+ VueLabyrinthe(Labyrinth laby, Canva canva)
+ update(Subject s) : void
+ dessinerJeu(Jeu jeu, Canva canvas) : void
- dessinRoad(Labyrinth laby, GraphicsContext gc) : void
- dessinTree(Labyrinth laby, GraphicsContext gc) : void
- dessinDefenses(Labyrinth laby, GraphicsContext gc) : void
- dessinEnemies(Labyrinth laby, GraphicsContext gc) : void
ViewLogs
+ VueLogs(Labyrinth laby)
+ update(Subject s) : void
ControllerLearning
+ handle(event : Event) : void
ControllerStart
+ handle(event : Event) : void
enemies
defenses
Entity
<<interface>>
- x : double
- y : double
- type : String


enemies
Entity
<<interface>>
- x : double
- y : double
- type : String
+ Entity(int x, int y)
+ setX(int x) : void
+ setY(int y) : void
+ getX(int x) : void
+ getY(int y) : void
defenses
*
*
Ennemy
<<abstract>>
- health : int
- speed : double
- damages : int
- attackSpeed : double
- range : double
- distanceToArrival : double
- killerType : String
+ move(double[] next) : void
+ attack(Defense target) : void
+ getType() : String
+ getDamages() : int
+ getHealth() : int
+ getBehavior() : Behavior
+ getRange() : double
+ getDistanceToArrival() : double
+ getShield() : int
+ getKillerType() : String
Behavior
1
1
Defense
<<abstract>>
- health: int
- damage : int
- range : double
+ attack(Ennemy target) : void
+ getType() : String
+ getHealth() : int
+ getHealth() : int
ActiveDefense
+ attackSpeed : double
PassiveDefense
Canon
+ attack(Enemy target) : void
Bomb
+ attack(Enemy target) : void
Ninja
Giant

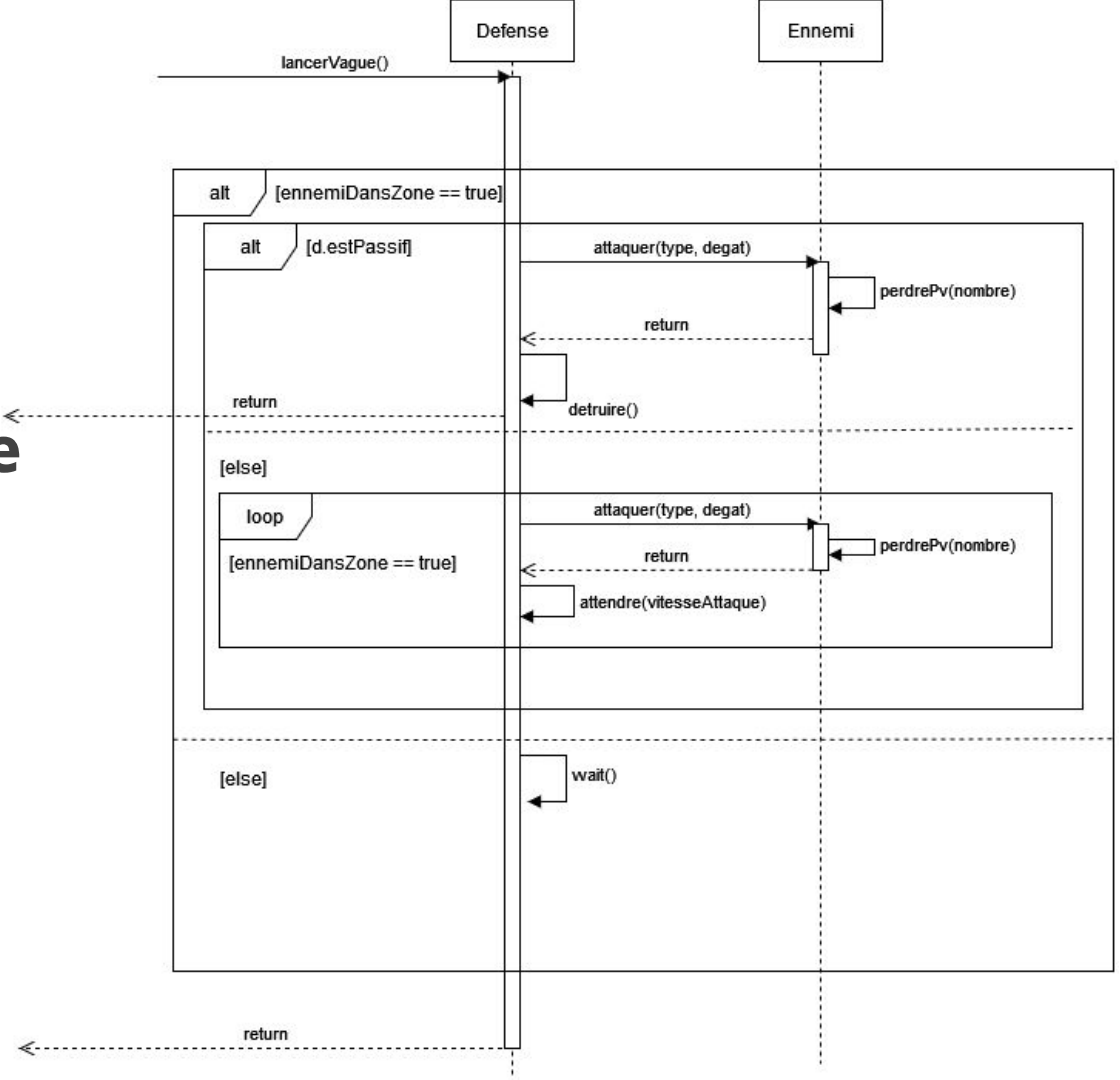
# Diagramme de classe du prototype de A* :

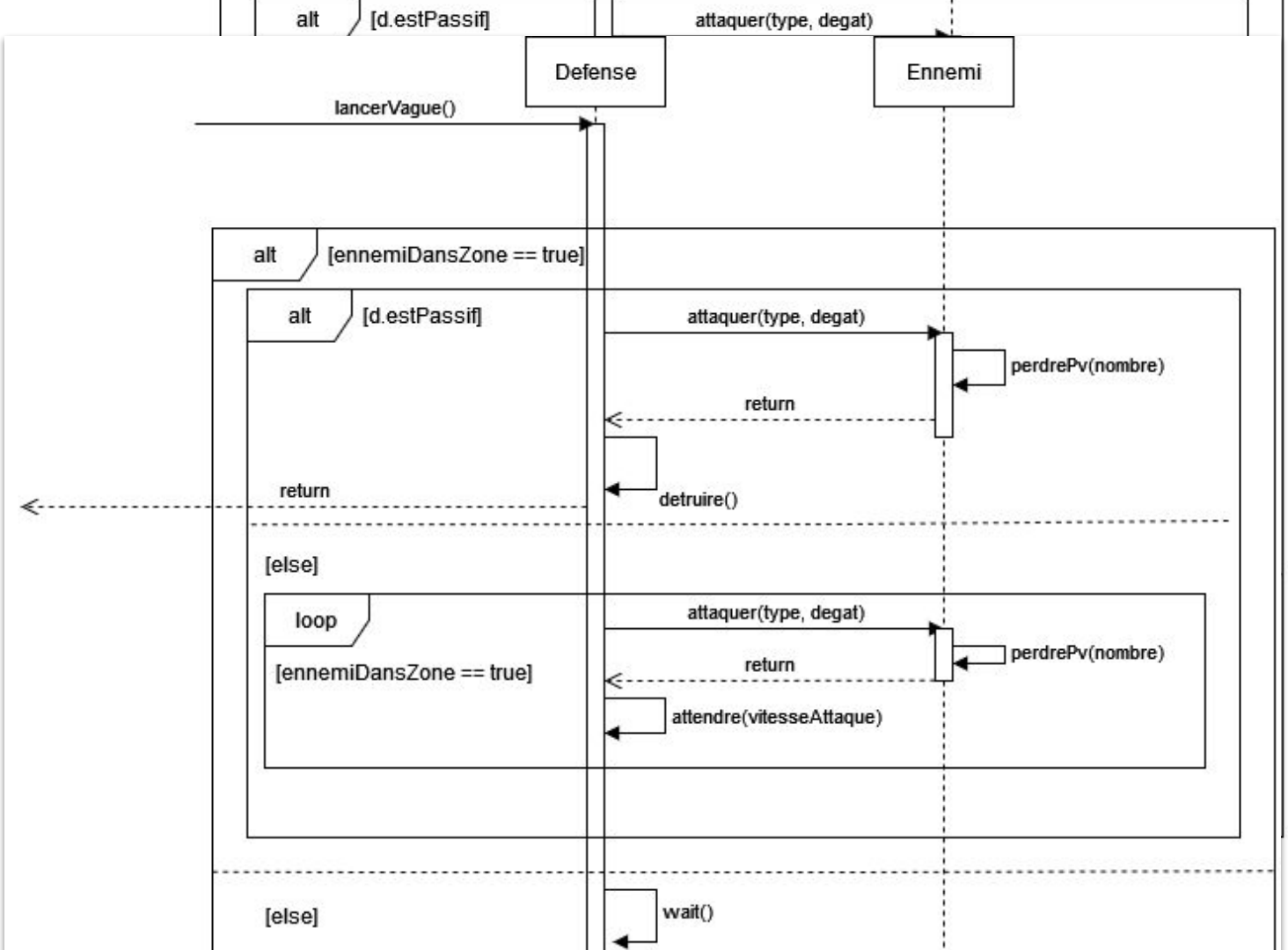


# Diagramme de classe du prototype de steering behaviour

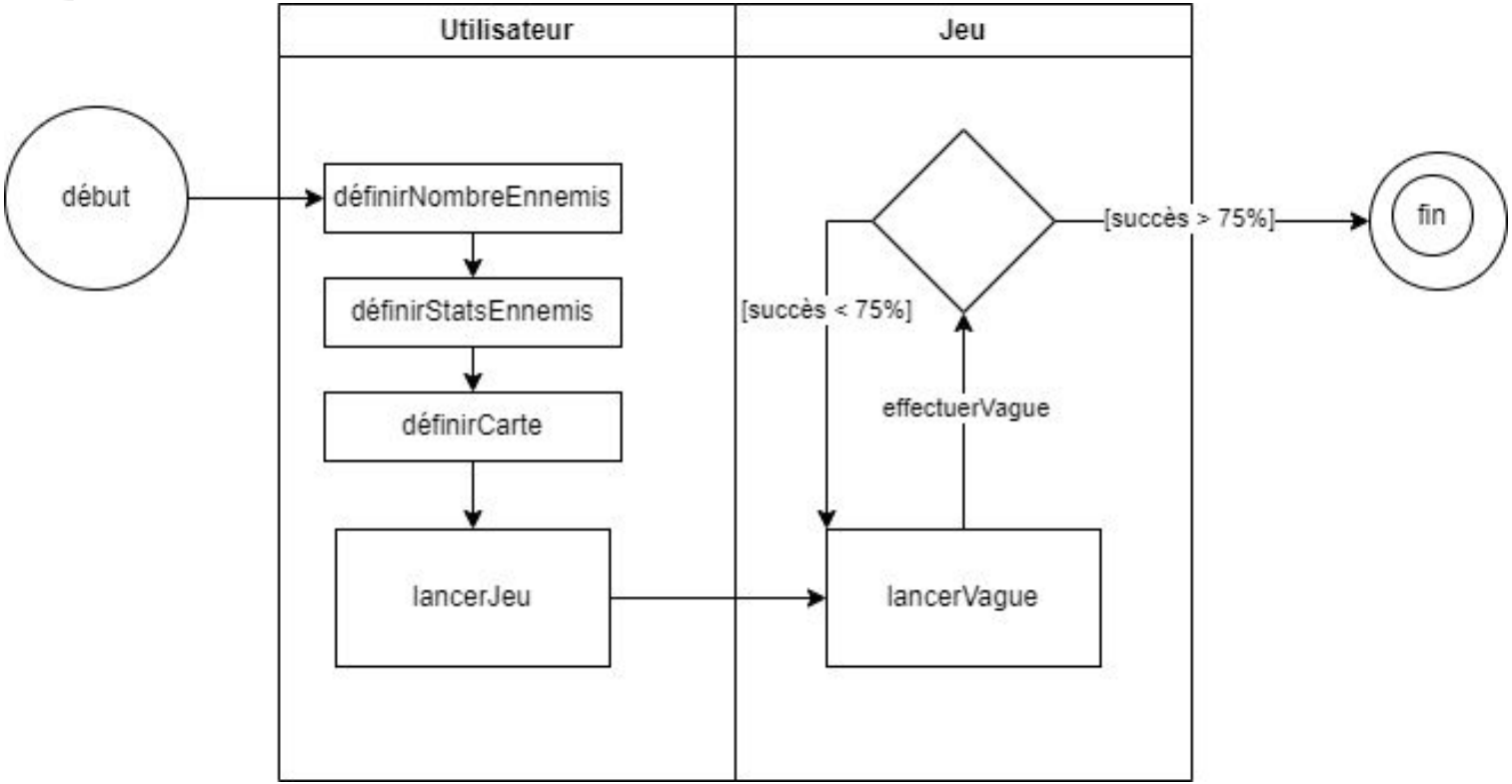


# Diagramme de séquence

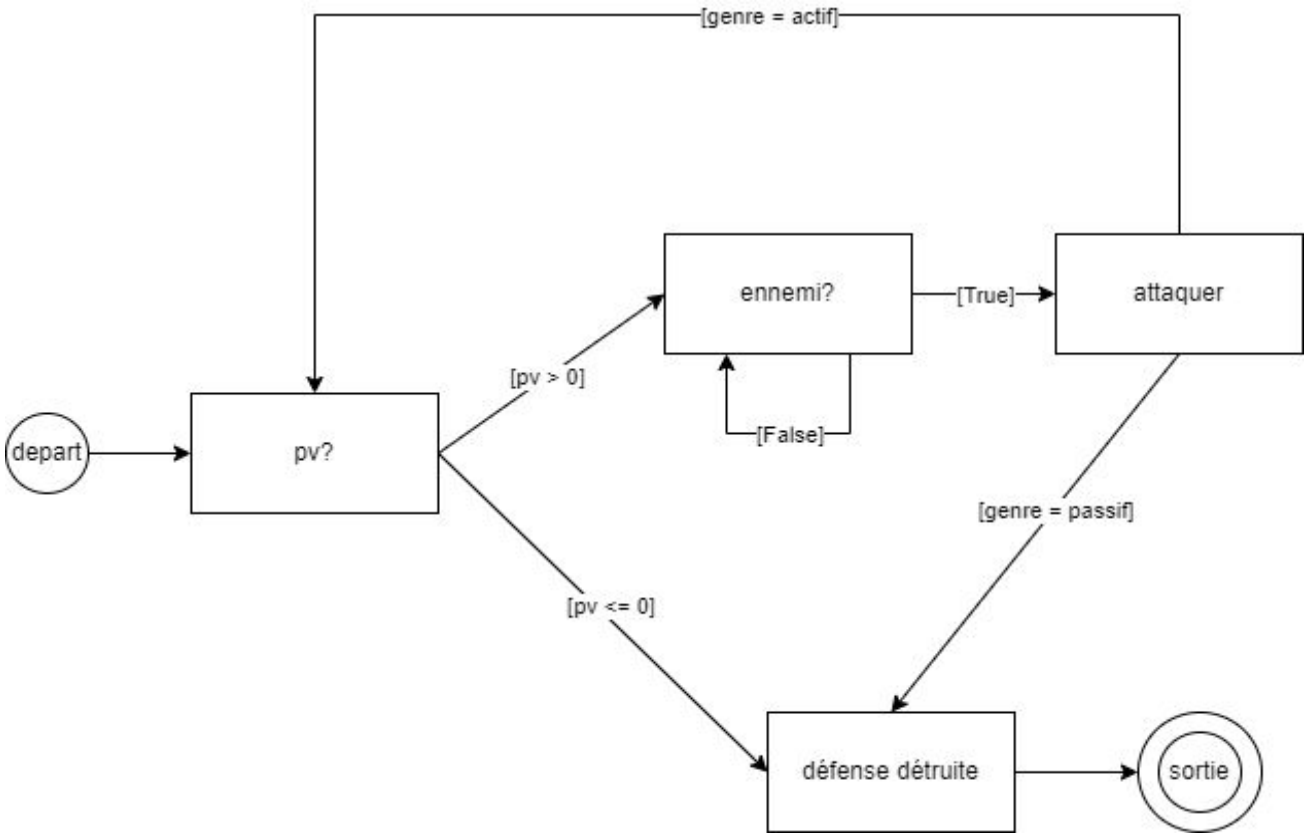



alt
[d.estPassif]
attaquer(type, degat)
Defense
Ennemi
lancerVague()
alt
[ennemiDansZone == true]
alt
[d.estPassif]
attaquer(type, degat)
perdrePv(nombre)
return
return
detruire()
[else]
loop
[ennemiDansZone == true]
attaquer(type, degat)
perdrePv(nombre)
return
attendre(vitesseAttaque)
[else]
wait()

# Diagramme d'activité : déroulement d'une partie



# Diagramme d'état des défenses



# Gestion de l’évolution des ennemis

- On récupère le **meilleur couple** par catégorie (Normal, Kamikaze, Healer, Fuyard).

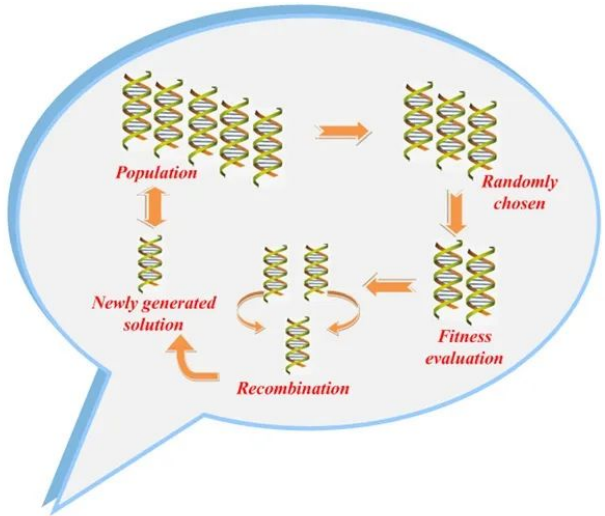
  On les calcule à l’aide d’une formule de score :

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

↳ Cas un seul ennemi

- On attribue les stats des couples gagnants aux ennemis morts de meme catégorie en faisant des moyennes

  **vitesseEnnemiN+1** = moy(parent1, parent2) de même catégorie
  **vieEnnemiN+1** = moy(parent1, parent2) de même catégorie
  **typeEnnemiN+1** = typeCounter(killerType)




# Gestion de l’évolution des défenses

**Mutation :** pour faire évoluer les ennemis **morts** à partir des couples gagnants (parents).

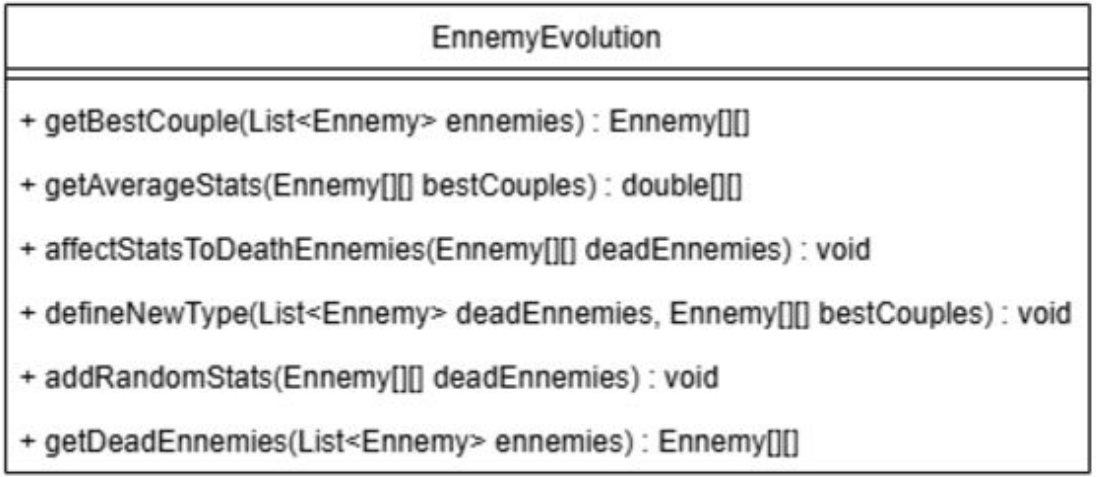
Après avoir affecté les stats moyennes des couples gagnants :
On va ajouter/retirer quelques unités aléatoirement pour faire muter les ennemis à évoluer:

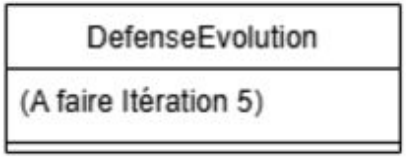
- Exemple : avec une base de **100pv** pour la vie :

  **vie** += **randomEntre**[-10, 10]
  (même chose pour la vitesse)

  Un couple seul (sans enfant à évoluer) -> Ils mutent **eux même**, sinon les ennemis parents n’évoluent pas.

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



# Gestion du déplacement

- **Steering behaviors** combiné à **A*** permet aux ennemies de se déplacer selon le **chemin le plus optimal** et de manière réaliste en fonction de **courbes**.

- 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 :

  **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.

# Recensement et évaluation des risques

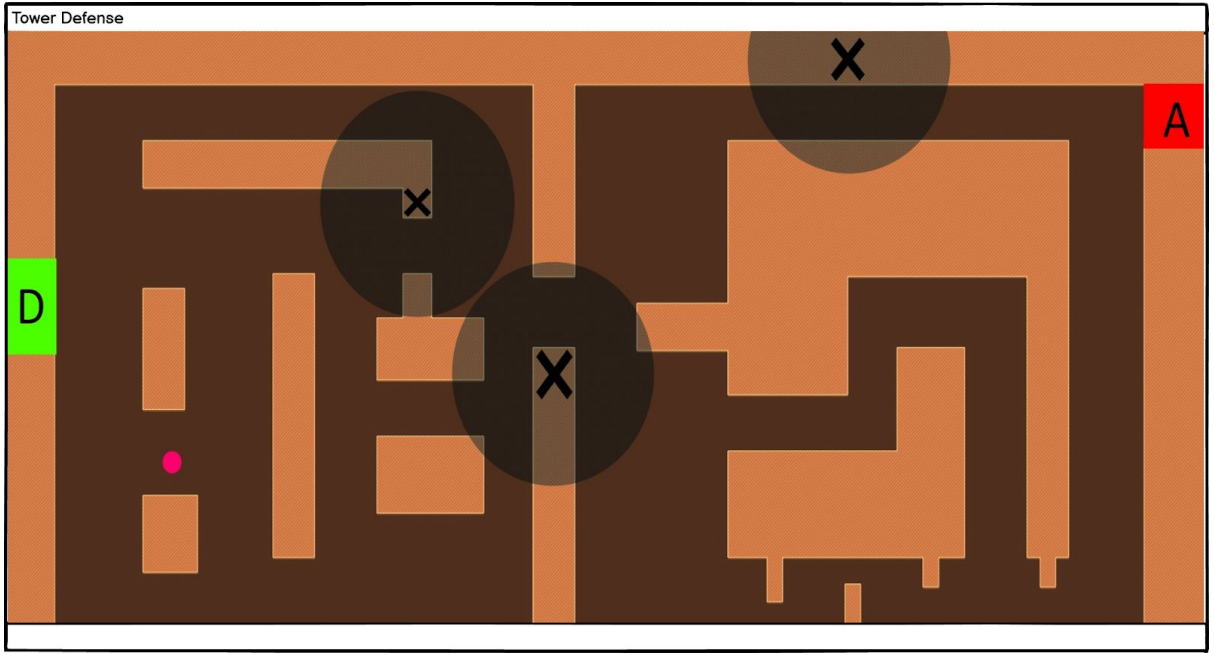
1. Compréhension universelle du sujet
2. Surcharge de ressources
3. Équilibrage du jeu
4. Collaboration des algorithmes entre eux
5. Unifications des outils
6. Gérer le temps

# Planning

# Itération 1

- Prototypes
- Créations des éléments

**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.

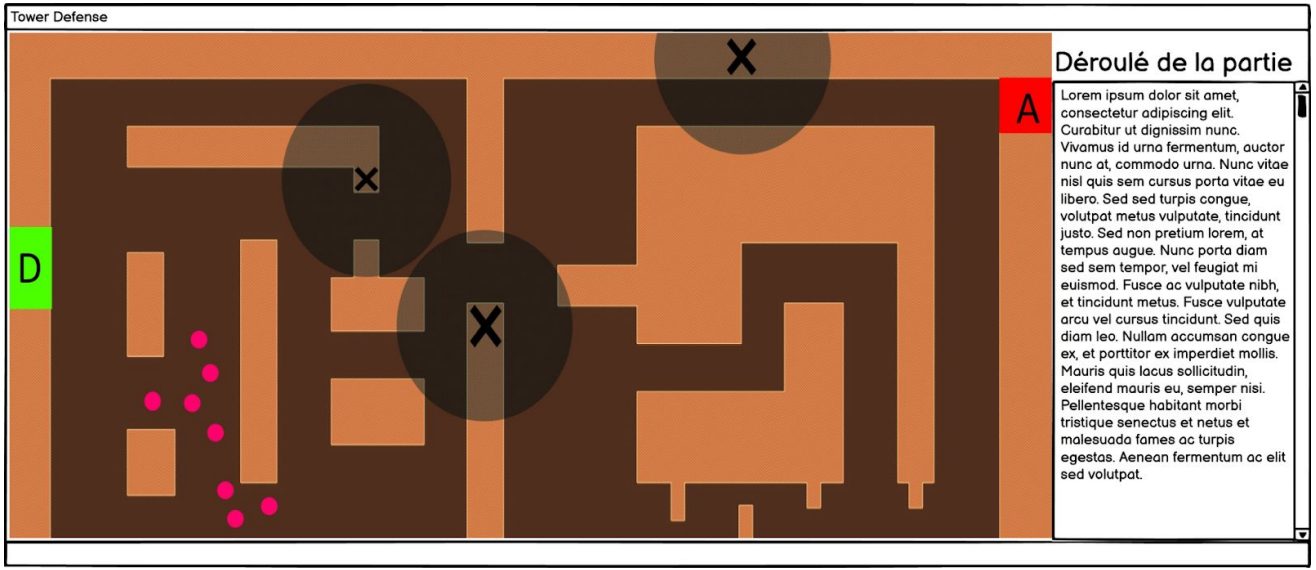


# Itération 2

- Fusion des prototypes
- Ajout panneau de “logs”

**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.



# Itération 3

- Créations de types d'ennemis et de défenses
- Création des comportements
- Ajout des sprites
- Attaque 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.

# Itération 4,5,6,7

- En fonction des itérations précédentes.
- Course à l’armement
  - Règles de victoire et de défaite

# Merci de votre attention