

Dossier du Projet (19-20)

Groupe : TORRES Nicolas, MIGNOT Vincent, ZHU Arthur

Documents pour CPOO (sur 27)

1.1 Architecture (9 points)

(MVC) + package

Nous avons adopté l'architecture MVC pour séparer le modèle et la vue. Le modèle et la vue sont reliés grâce au contrôleur qui en fonction des actions utilisateurs modifiera la vue ou/et le modèle.

Nous avons choisi de faire un modèle géré par une partie, contenant un environnement qui contient lui même 3 listes d'entités et le terrain.

Ces entités sont séparés par catégorie : les personnages, les tourelles et les tirs.

Les 3 catégories fonctionnent de la même façon. Elles ont une super-classe qui peuvent être de plusieurs types qui sont des sous-classes ayant des statistiques et des compétences différentes.

Les personnages sont ajoutés automatiquement dans l'environnement par la classe partie. Ils sont plus ou moins nombreux en fonction du niveau de la partie.

Les tourelles sont ajoutés par l'utilisateur via la vue. L'information est récupéré dans le contrôleur qui fait le lien avec le modèle pour ajouter la tourelle.

Les tirs sont créés par les tourelles et se déplacent vers une position cible donnée par la tourelle qui correspond à la position d'un personnage au moment de la création du tir, à l'exception des tirs de la tourelle à fiole qui eux, se créent directement à la position cible.

L'environnement contient aussi une map généré par fichier.

La partie contient une classe score enregistrant les scores et affiche l'historique de ces derniers.

Le package contrôleur écoute les actions de l'utilisateur et le modèle et en fonction va modifier la vue ou/et le modèle. Par exemple, les boutons "Info" présent sur notre vue sont reliés à des méthodes dans le contrôleur qui va modifier la vue. Il y a des Listeners sur chaque liste de l'environnement (tirs, personnages, tourelles) qui écoutent les changements fait sur la liste et va réagir en fonction du changement. Ici on voit en effet l'utilité du contrôleur puisqu'elle relie le modèle et la vue. Par exemple, la tourelle du modèle va créer un tir puis le listener sur la liste de tir va créer la vue du tir pour pouvoir l'afficher à l'utilisateur.

Dans notre package vue se trouve tous les éléments graphiques qui représentent les éléments du jeu comme les personnages, les tourelles, les tirs et la map. Elles sont appelées par les listeners pour pouvoir afficher les éléments du modèle devant être visible par l'utilisateur.

1.2 Détails : diagrammes de classe (9 points)

Diagramme de la classe Tir

Concernant les tirs, la **super classe** est la classe **Tir**.

Cette classe est abstraite car la fonction agit(), va être redéfini dans chacune de ces sous classes. Certains attributs et certaines fonctions sont placés dans cette super classe car ils sont commun à toutes les tourelles.

On distingue à présent les classes **TirDirection** et **TirFiole**, ce sont deux sous classes de Tir, ayant un comportement différent.

En effet, TirFiole se crée directement à la position de l'ennemie visé, tandis que les sous classes de **TirDirection** se créent à la position de la tourelle puis se déplacent avec une direction et une vitesse.

En cas d'impact ou de sortie de portée, ils se suppriment et, le cas échéant, infligent des dégâts.

Tandis que **TirFiole** inflige des dégâts importants pendant plusieurs secondes puis se supprime.

TirDirection a donc besoin de calculer sa direction, tandis que TirFiole n'a besoin que de la position du personnage.

On distingue à présent plusieurs sous classes de TirDirection, cela se justifie car ces différents tirs n'ont pas les mêmes valeurs d'attributs, et TirVaccin a également une fonction collision redéfinie.

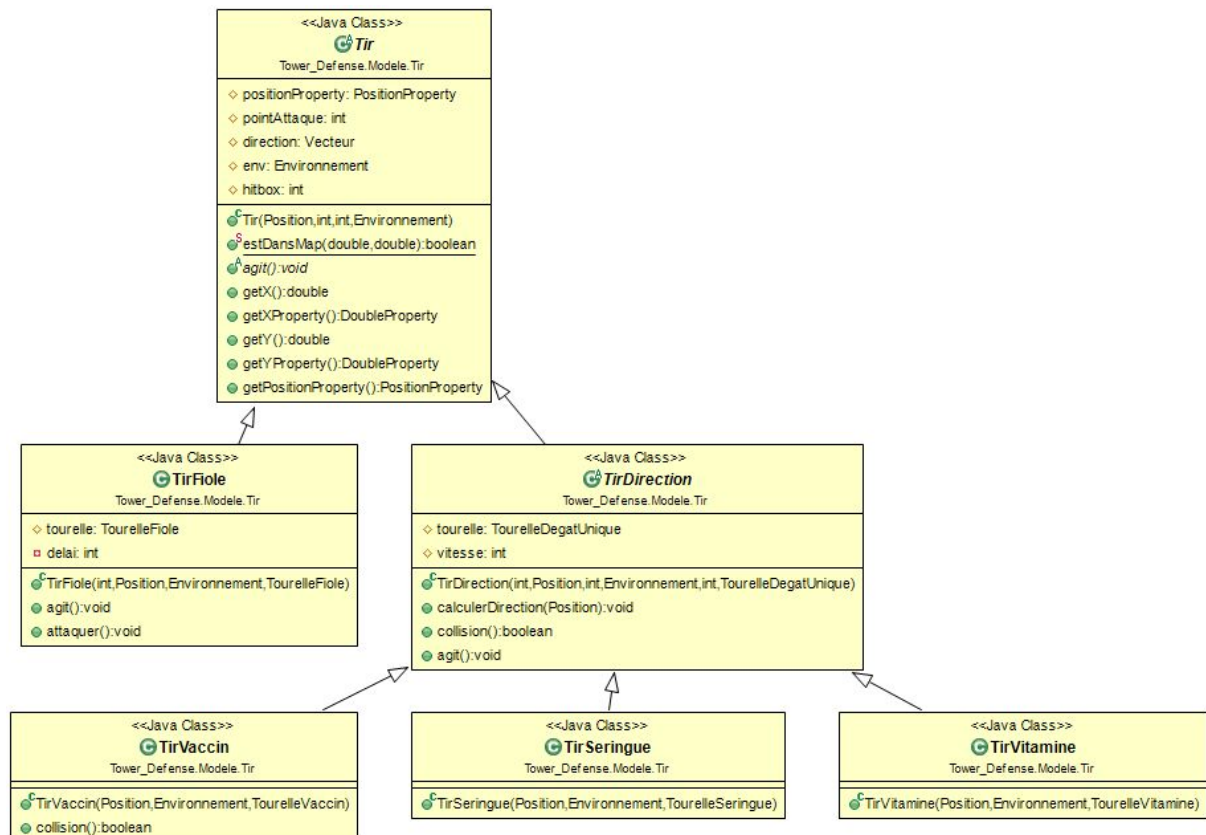


Diagramme de la classe Tourelle

Concernant les tourelle, la super classe est la classe **Tourelle**.

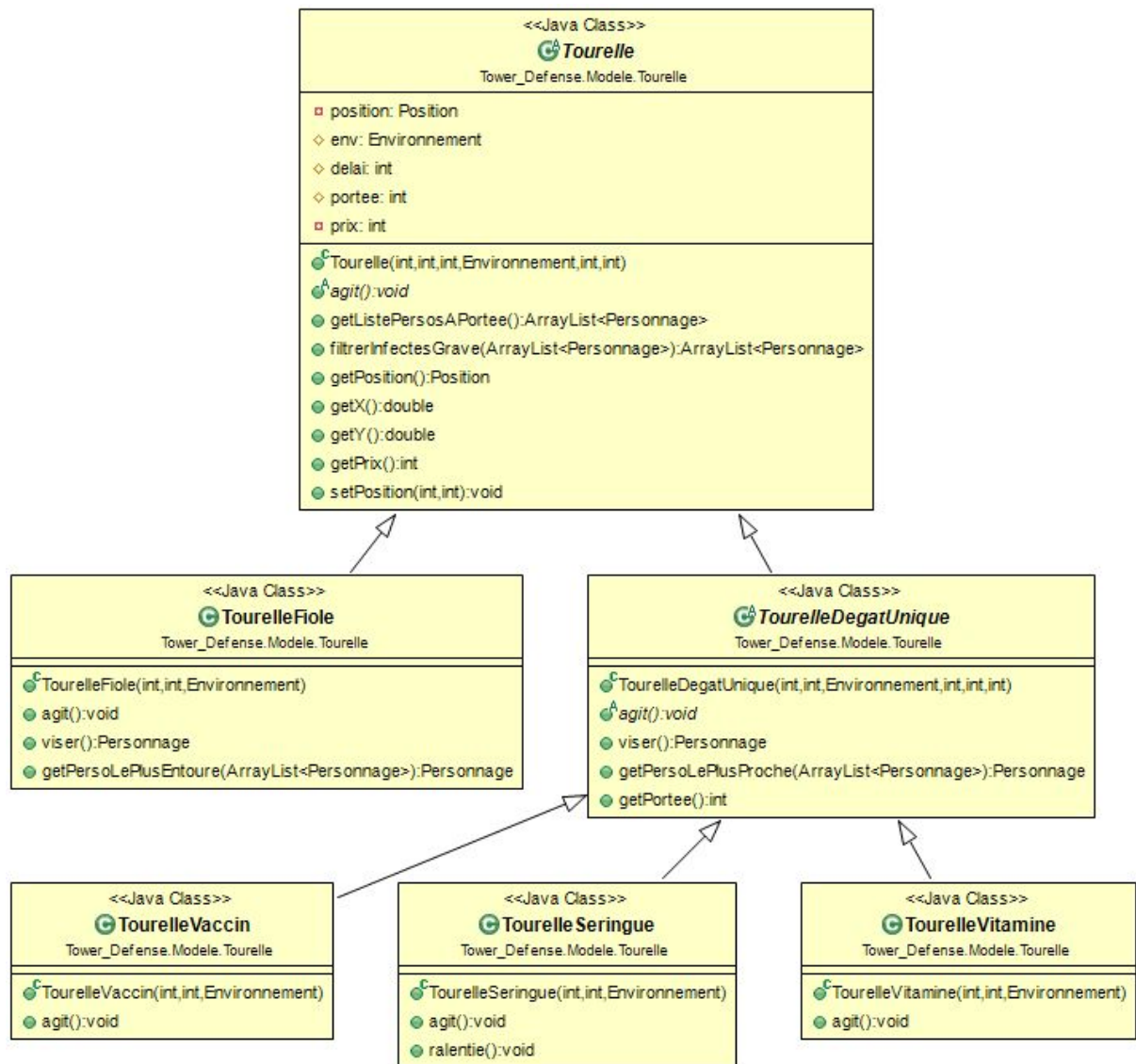
Cette classe est abstraite car la fonction `agit()`, va être redéfini dans chacune de ces sous classes. Certains attributs et certaines fonctions sont placés dans cette super classe car ils sont commun à toutes les tourelles.

On distingue à présent **TourelleFiole** et **TourelleDegatUnique**, ce sont deux sous classes de Tourelle, qui ont un comportement différent.

En effet, **TourelleFiole** ne vise pas en priorité le personnage le plus proche de la tourelle comme le fait **TourelleDegatUnique**, mais le personnage le plus entouré d'autres personnages à moins de 100 pixels.

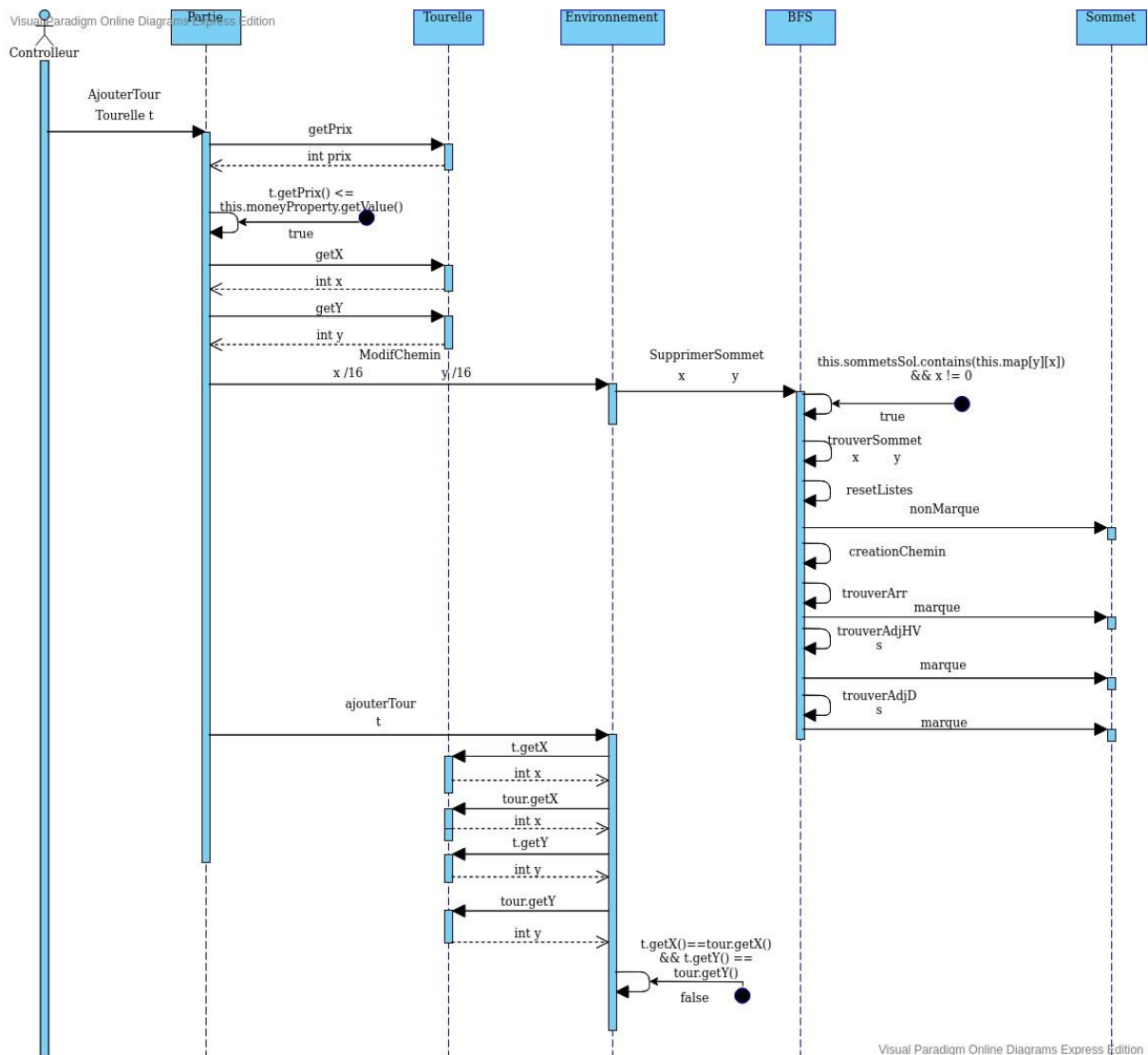
On distingue à présent plusieurs sous classes de **TourelleDegatUnique**, cela se justifie car ces différentes tourelles n'ont pas les mêmes valeurs d'attributs et agissent différemment.

À titre d'exemple la **TourelleSeringue** va ralentir les ennemis et tirer moins souvent que **TourelleVitamine**, tandis que **TourelleVitamine** tir plus souvent que **TourelleVaccin** et **TirSeringue**. La **TourelleVaccin** inflige en plus un soin sur la durée.



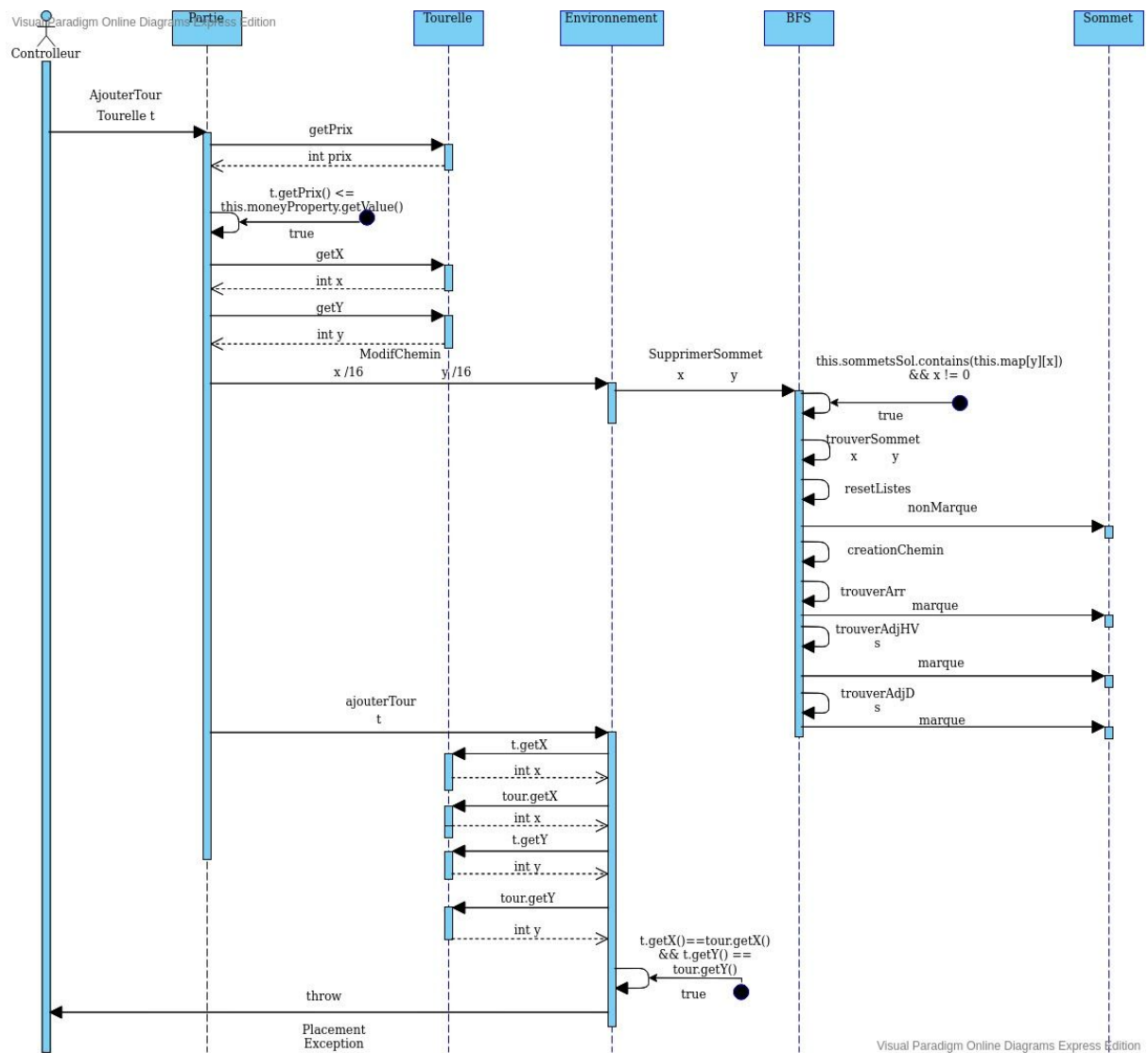
1.3 Diagrammes de séquence (9 points)

Ce diagramme de séquence présente la méthode **ajouterTour** de la partie quand il n'y a aucune exception. La fonction **ajouterTour** de la partie vérifie que le prix de la tourelle est inférieur à l'argent disponible, si c'est le cas elle appelle la fonction **ModifChemin** de l'environnement qui appelle **SupprimerSommet** du BFS. Cette méthode va se charger de supprimer le sommet correspondant à la position demandé pour la tourelle en vérifiant que le sommet rentre dans les sommets autorisés. Après la suppression du sommet, le chemin est actualisé en refaisant l'algorithme du BFS. Une fois le chemin actualisé, la tourelle est ajouté à l'environnement par l'appel de **ajouterTour**, qui vérifie qu'un tourelle n'existe pas déjà à cette position et si ce n'est pas le cas l'ajoute.



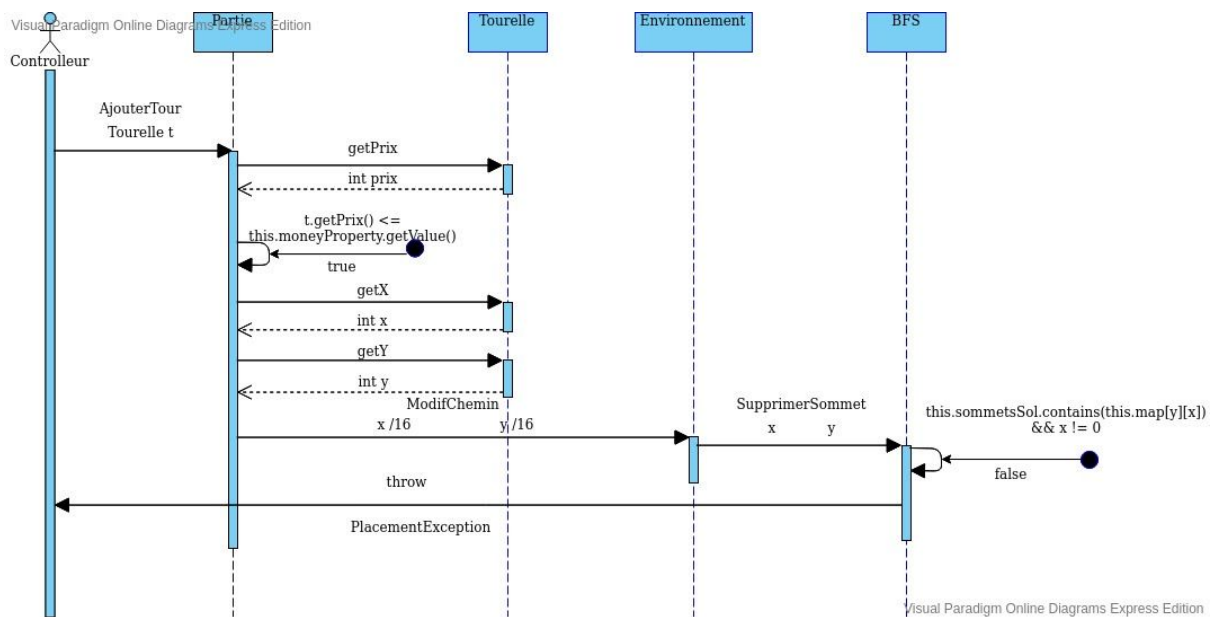
Ce diagramme de séquence reprend la méthode **ajouterTour** de la partie quand on essaie de placer la tourelle sur une autre.

La seule différence est la dernière vérification qui va provoquer la création d'une exception sur le placement depuis l'environnement car la tourelle ne peut pas être posé sur une autre.



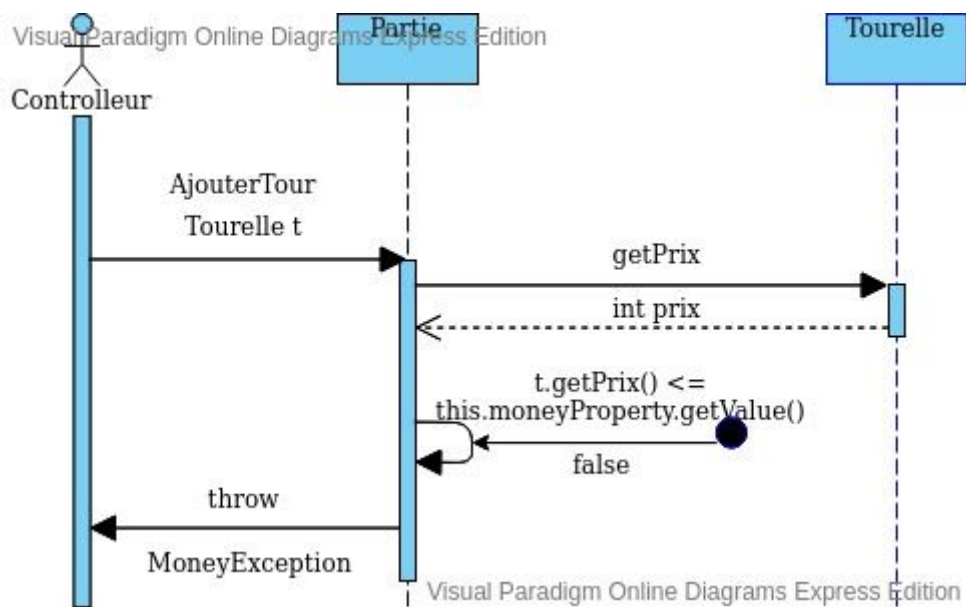
Ce diagramme de séquence reprend la méthode **ajouterTour** de la partie quand la tourelle n'est pas positionnée sur le sol ou si elle est positionné sur la première colonne.

Au moment du test dans le BFS pour vérifier que la tourelle est positionnée sur un emplacement autorisé, le résultat est faux et la méthode envoie une exception sur le placement.



Ce diagramme de séquence reprend la méthode **ajouterTour** de la partie quand l'argent du joueur est insuffisant pour acheter la tourelle.

Suite au test dans la méthode **ajouterTour** de la partie, le résultat est faux et une exception sur l'argent est envoyée.



1.4 Structures de données :

Concernant les personnages visés par les tourelles dans la classe **Tourelle**, nous utilisons une ArrayList de **Personnage** afin de trier les personnages. L'insertion, la suppression, et le parcours dans une ArrayList étant simple.

Dans l'**Environnement**, les tourelles, personnages et tirs sont stockés dans des ObservableList, afin d'avoir les avantages des ArrayList, mais de pouvoir les observer pour le Listener, car les éléments de ces listes doivent s'ajouter à la vue ainsi que se supprimer.

La map est stocké dans un tableau d'entier à 2 dimensions, de 30 par 50 et stocker dans un fichier csv.

Le **BFS** utilise des structures de type sommet qui permet d'avoir les coordonnées et l'état de marquage des sommets dans l'algorithme et de comparer les sommets entre eux.

Dans les Listeners et dans le **BFS**; nous utilisons les HashMap pour faciliter la correspondance entre les éléments par exemple un personnage et sa vue ou un sommet avec le sommet auquel il est rattaché. Cela facilite la suppression des éléments dans la vue depuis la modification des listes de la vue par les listeners. La HashMap utilisé dans le **BFS** permet au personnage de trouver très rapidement le chemin qu'il doit emprunter sommet par sommet.

Concernant les **positions**, la structure de donnée est la classe **Position**, contenant un x et un y de type nombre décimale. Elle est utilisé afin d'indiquer la position des tourelles, et de pouvoir calculer des distances.

La variante de cette classe, **PositionProperty**, a la même utilité, à l'exception que les x et y, sont des **SimpleDoubleProperty**, afin de pouvoir lier la vue au modèle, pour permettre le déplacement des vues des tirs.

Concernant les directions, la structure de donnée est la classe **Vecteur**, cette classe permet de stocker via deux nombres décimaux, un x et un y, et d'y appliquer des formules dans le but de calculer des directions.

1.5 Exception :

Il y a la gestion de 2 exceptions dans la classe Controleur sur la saisie du nom par l'utilisateur dans le textfield présent au début de chaque nouvelle partie qui sont NomTropLongException (envoyé quand le nom saisi est trop long) et PasDeNomException (envoyé quand il n'y a aucun nom saisi).

On gère 2 autres exceptions concernant les tourelles dans le contrôleur, une exception sur le placement qui peut être envoyé dans le bfs ou dans l'environnement et une exception sur l'argent envoyé dans la partie. Ces 2 exceptions mènent à un affichage pour prévenir l'utilisateur qu'il n'a pas le droit de placer les tours n'importe où et si les tours sont plus chères que son argent actuel.

1.6 Utilisation maîtrisée d'algorithmes intéressants :

Tourelle

La fonction viser() de **TourelleDegatUnique**, va premièrement vérifier qu'il y a bien des personnages à portée en appelant la fonction getListePersoAPortee(), si cette liste est vide elle ne va rien retourner, sinon elle va ensuite filtrer les infectés grave dans cette liste de perso à portée : filtrerInfectesGrave(getListePersoAPortee()).

Si cet appel de fonction ne retourne pas une liste vide elle va retourner le personnage le plus entouré d'autres personnages de cette liste, sinon de tous les personnages à portée car les Tourelles visent en priorité les infectés graves.

Pour cela, on appelle la fonction : getPersoLePlusEntouré().

Elle compare la distance entre tous les Personnages passés dans l'ArrayList en paramètre, et retourne le **Personnage** ayant le plus de Personnage autour de lui à une distance inférieure à 100 pixels.

Les tourelles qui extends la classe **TourelleDegatUnique** ont également une fonction `viser()` similaire, la distinction se fait par le remplacement de la méthode `getPersoLePlusEntouré()` par `getPersoLePlusProche()`, ces tourelles tirent sur le personnage infecté grave le plus proche ou non-infecté grave le plus proche.

Tir

À la création d'un **TirDirection**, la fonction `calculerDirection()` est appelé, afin d'attribuer des valeurs exactes à son attribut de classe **Vecteur**, via des formules mathématiques.

Par la suite, sa fonction `agit()`, vérifie si après déplacement, le **TirDirection** sera dans la carte ou à une portée définie, le supprime si non.

La fonction vérifie ensuite s'il y'a eu collision avec un **Personnage**, si oui les dégâts sont infligés et le tir supprimé.

Sinon le tir poursuit son mouvement.

BFS

Pour le **BFS**, l'algorithme permet de parcourir les chemins possibles et de les garder en mémoire pour pouvoir se déplacer de manière optimale sur le map en direction de l'arrivée.

Lors de la création du BFS, les sommets correspondants à du sol sont créés puis le parcours est effectué en partant de la tuile qui fait office d'arrivée.

Lors de l'ajout d'une tourelle, le sommet correspondant à son emplacement est supprimé et le parcours est effectué de nouveau.

1.7 JUnits :

Les classes couvertes par nos JUnits sont BFS, Personnage, Tourelle et Tir car l'on vérifie lorsque la tourelle agit que le tir est créé puis se supprime à la collision.

2 Documents pour Gestion de projet

2.1 Document utilisateur (8 points) :

- Notre Tower defense se déroule dans un univers apocalyptique moderne, les personnages sont tous infectés avec des taux de contamination plus ou moins élevé.

Votre objectif sera de guérir ces infectés en plaçant différentes tourelles, à des points stratégiques. Vous devrez guérir le maximum de personnages avant qu'ils n'atteignent le bunker de personnages sains.

À chaque vague, les infectés sont plus nombreux et vous devrez donc atteindre le plus de vague possible.

Liste des tourelles

Tourelle vitamine



Type de tir : Mitraillette

Soin : faible

Portée : faible

Cadence : rapide

Cible : l'infecté le plus proche (s'il n'y a pas d'infectés graves)

Coût : 1000

C'est la tourelle avec la cadence de tir la plus élevée mais soigne peu les infectés.

Tourelle seringue



Type de tir : Coup par coup

Soin : moyenne

Portée : légère

Cadence : moyenne

Cible : l'infecté le plus proche (s'il n'y a pas d'infectés graves)

Coût : 2500

Cette tourelle coûte un peu plus cher que la tourelle vitamine mais elle a l'effet de ralentir les infectés proches de la tourelle. Elle a une cadence de tir moyenne mais soigne plus.

Tourelle seringue



Type de tir : Sniper

Soin : important

Portée : grande

Cadence : faible

Cible : l'infecté le plus proche (s'il n'y a pas d'infectés graves)

Coût : 3000

Cette tourelle applique un effet de soin sur la durée sur les infectés touchés. Elle soigne de beaucoup à l'impact du tir sur les infectés mais à une cadence de tir faible.

Tourelle fiole



Type de tir : Dégât de zone

Soin : très important

Portée : très grande

Cadence : faible

Cible : l'infecté le plus entouré d'autres d'infectés à sa portée (s'il n'y a pas d'infectés graves)

Coût : 8000

Cette tourelle tire sur l'infecté le plus entouré d'autres d'infectés à sa portée. C'est la tourelle la plus chère du jeu car elle possède un soin très important.

Liste des personnages

Infecté sans symptôme



Contamination : 100

Vitesse de déplacement : 4

Infecté jogger



Contamination : 150

Vitesse de déplacement : 4

Il accélère quand on lui tire dessus, sa vitesse est doublée (4 -> 8).

Plus contaminé que l'infecté sans symptôme. Même vitesse de déplacement.

Infecté grave



Contamination : 300

Vitesse de déplacement : 2

Il est beaucoup plus contaminé que les autres et il possède la vitesse de déplacement le plus faible parmi les ennemis. Il force l'attaque des tourelles sur lui.

Infecté qui tousse



Contamination : 150

Vitesse de déplacement : 4

Il empêche les tirs des tourelles sur les infectés à proximité de lui quand on lui tire dessus.

Personnage sain



Contamination : 0

Vitesse de déplacement : Dépend de l'infecté original

C'est un personnage sain.

Effets possibles

Soin sur la durée





Personnage qui subit un soin sur la durée causé par la tourelle vaccin.

Protection



Personnage protégé des tirs par l'effet actif de l'infecté qui tousse.

Tableau résumant le scoring selon les infectés

	10
	25

	50
	75

Bouton “Règles”

Le bouton est utilisable pour afficher les règles du tower defense. Après avoir été cliqué, un bouton apparaît et peut être utilisé pour masquer les règles.

Bouton “Scores”

Le bouton est utilisable pour afficher les scores du tower defense. Après avoir été cliqué, un bouton apparaît et peut être utilisé pour masquer les scores.

Liste des personnage et des statuts cliquable

Ces listes d’images représentant les différents personnages et statuts sont cliquables pour afficher une description dans un bloc d’information placé juste en dessous.

Boutons “Info” sous les tourelles

Ces boutons servent à afficher la description de la tourelle quand on clique dessus.

Drag and Drop des tourelles sur le plateau de jeu

On peut cliquer puis glisser une tourelle sur le plateau de jeu pour la placer sur le plateau de jeu.

Sauvegarde des scores

Le joueur doit saisir son nom au début de chaque nouvelle partie qu'on utilise pour stocker le score du joueur a la fin de la partie dans un fichier. Ce fichier est lu quand on clique sur le bouton "Scores" pour afficher les scores.

Son a la fin d'une vague et quand la partie est perdu

Un son est joué quand le joueur finit une vague et quand il perd la partie.

Recommencer une partie

À la fin d'une partie, le joueur peut recommencer une partie.