RIBEIRO David, GORLICKI Nicolas, RANJITH Rogith, RAZANAJATOVO Nantenaina

Rapport de Projet

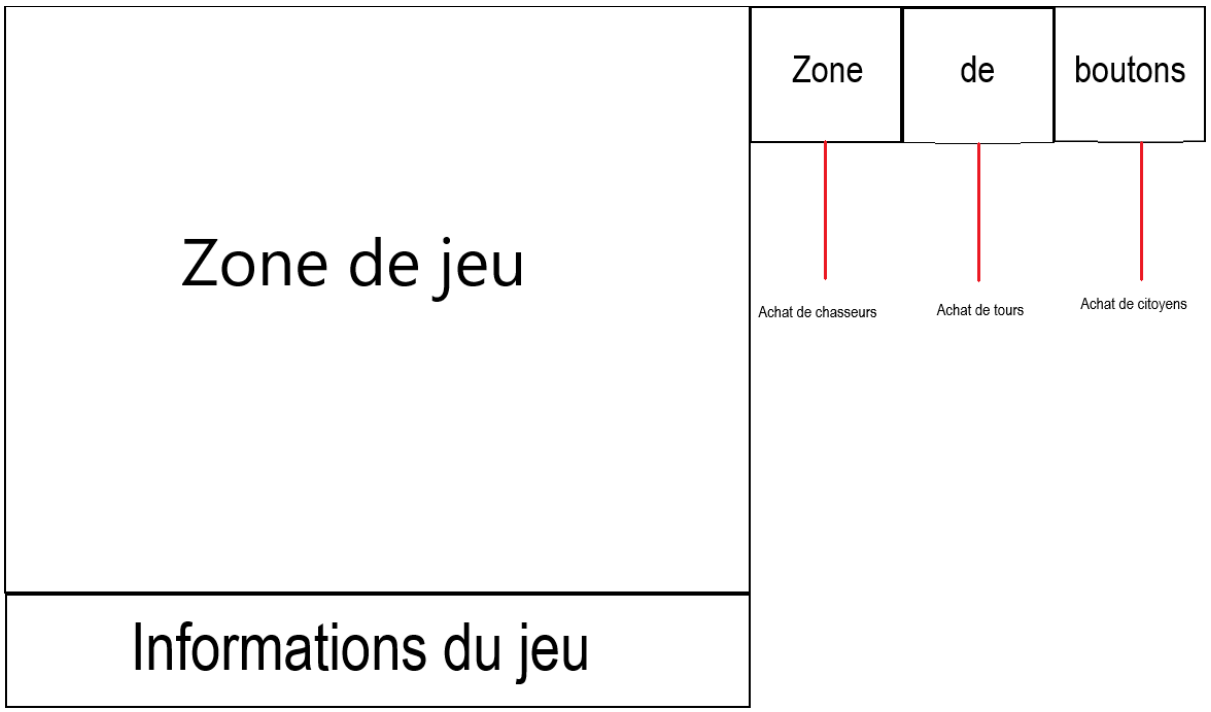
Introduction

L'objectif du projet est de réaliser un jeu de stratégie où l'utilisateur doit tenir le plus longtemps possible. Le but du jeu est de protéger les citoyens et la base des monstres en construisant des tours et en plaçant des chasseurs. Si la base est détruite, la partie est terminée.

scrAvec l'argent, il peut recruter soit des citoyens à partir d'une certaine somme, créer des bâtiments défensifs à partir d'une somme et recruter des héros.
L'utilisateur gère les tours et les chasseurs: il peut déplacer ces derniers, peut placer des tours défensives, peut tuer les monstres lorsqu'ils sont dans les cases adjacentes (ou la même case) des tours ou des chasseurs.
Le joueur obtient un score et pourra rejouer à une nouvelle partie s'il le souhaite.

Analyse Globale

L'interface se décompose en deux parties. L'affichage du jeu et la partie du panneau de contrôle qui se situe à droite de l'affichage.
L'affichage du jeu est une grille de case. Affiche de base une carte avec la base principale, un chasseur et quelques citoyens.
Le panneau de contrôle se situe à droite de l'affichage et est composé de plusieurs boutons de créations (créations des bâtiments défensifs, de citoyens et chasseurs supplémentaires).
L'affichage doit réagir lorsque des unités sont ajoutées, l'affichage est mis à jour et ces unités apparaissent.



Description des fonctionnalitées liées aux unitées:

- Chasseur: unité contrôlable par le joueur qui a un nombre de pv de base et un nombre de points d'attaque fixe. Cette unité a pour but de défendre les citoyens en tuant les monstres. Nous avons besoin de définir plusieurs fonctionnalitées pour définir cette unité:
  Attaquer: Peut réduire les pvs des monstres qui sont dans sa même case.
  Se déplacer: Peut se déplacer de case en case grâce à la sélection qui se fait par le joueur
  Imposition: Récupère l'argent d'un citoyen quand ils se trouvent dans la même case
  Mourir : Disparaît de la grille lorsque les pv sont réduit à zéro

- Citoyen: Le citoyen est une unité non contrôlable, qui a un nombre de pv. Cette unité a pour but de gagner de l'argent. Nous avons besoin de définir plusieurs fonctionnalitées pour définir cette unité:
  Déplacement : Se déplace de case en case automatiquement de manière aléatoire
  Argent : Augmentation linéaire de l'argent qu'il porte sur lui au fil du temps
  Mourir : Disparaît de la grille lorsque les pv sont réduit à zéro

- Monstre: Le monstre est une unité non contrôlable, qui a un nombre de pv et une attaque de base. L'objectif de cette unité est de tuer tous les citoyens. Nous avons besoin de définir plusieurs fonctionnalitées pour définir cette unité:
  Apparition : Se créer et apparaît automatiquement et aléatoirement aux bordures de la carte
  Attaque : Réduit les pv des citoyens qui sont dans sa même case
  Détruit : Réduit les pv du bâtiment qui se trouve dans sa même case
  Déplacement : Se déplace de case en case automatiquement en direction de la base.
  Mourir : Disparaît de la grille lorsque les pv sont réduit à zéro

- Base: le bâtiment principal qui a un nombre de pv. Si ce bâtiment est détruit, la partie se termine.
  Perte de pv: les pv sont réduit lorsqu'un monstre se trouve sur la même case
  Détruit: Lorsque la base n'a plus de pv, elle est alors détruite

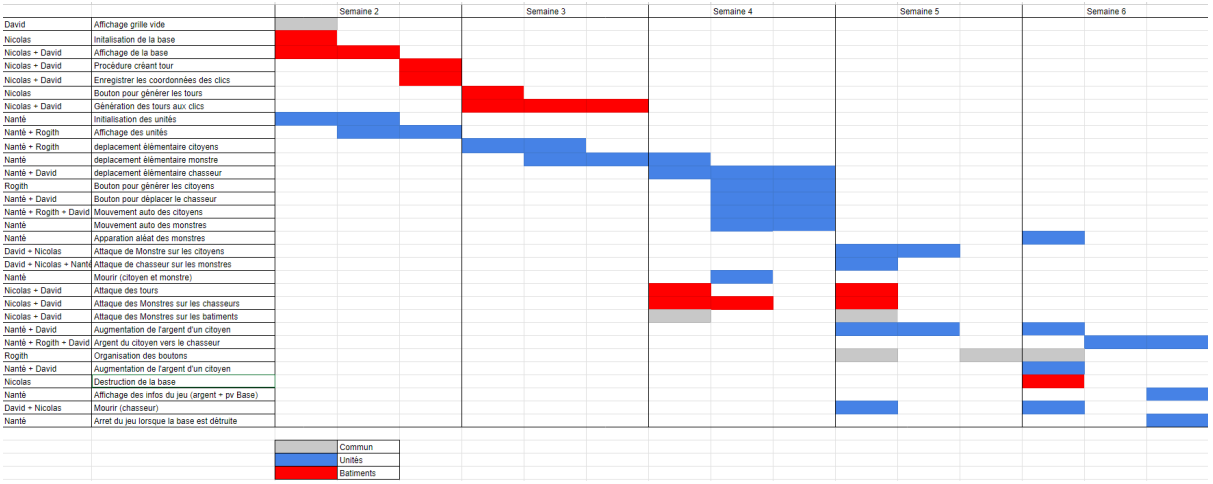
- Tour défensive: bâtiment qui a un nombre de pv et une attaque de base. Cette unité a pour but d'aider les chasseurs dans la protection des citoyens. Nous avons besoin de définir plusieurs fonctionnalitées pour définir cette unité:
  Attaquer: Réduit les pv des monstres qui se trouvent dans son rayon d'action

- Contrôleur est un panneau de contrôle qui permet d'ajouter:
  Création de tour : Créer une tour grâce à un clic de souris qui va récupérer les coordonnées souhaitées suivi de l'appuie du bouton correspondant si l'économie du jeu le permet.
  Création de citoyen: Créer un citoyen grâce à un clic de souris qui va récupérer les coordonnées souhaitées suivi de l'appuie du bouton correspondant si l'économie du jeu le permet.

Création de chasseur: Créer un chasseur grâce à un clic de souris qui va récupérer les coordonnées souhaitées suivi de l'appuie du bouton correspondant si l'économie du jeu le permet.

La mise en mouvement des unités et l'interaction entre les unités et les bâtiments sont les fonctionnalités qui nous ont paru les plus compliquées. Du fait que l'on souhaitait à ce que tous les mouvements soient automatiques, l'interaction entre les unités devaient également se faire automatiquement

Plan de développement



2ème semaine :

- Création et affichage de la grille (15 min)
- Initialisation de la base (20 min)
- Affichage de la base (15 min)
- Procédure créant tour (45 min)
- Enregistrer les coordonnées des clics (15 min)
- Initialisation des unités (30 min)
- Affichage des unités (25 min)
- Documentation (30 min)
- Rédaction du rapport (45 min)

3ème semaine :

- Bouton pour générer les tours (20 min)
- Génération des tours au clics (1h)
- Déplacement élémentaire des citoyens (1h)
- Déplacement élémentaire des monstres (45 min)

4ème semaine :

- Déplacement élémentaire des chasseurs (1h)
- Bouton pour générer les citoyens (30 min)

- Bouton pour déplacer le chasseur (30 min)
- Mouvement auto des citoyens (30 min)
- Mouvement auto des montres (25 min)
- Mourir (Citoyen et Monstres) (15 min)
- Attaque des tours (25 min)
- Attaque des tours sur les monstres (2h)
- Attaque des monstres sur les bâtiments (1h30)

5ème semaine :

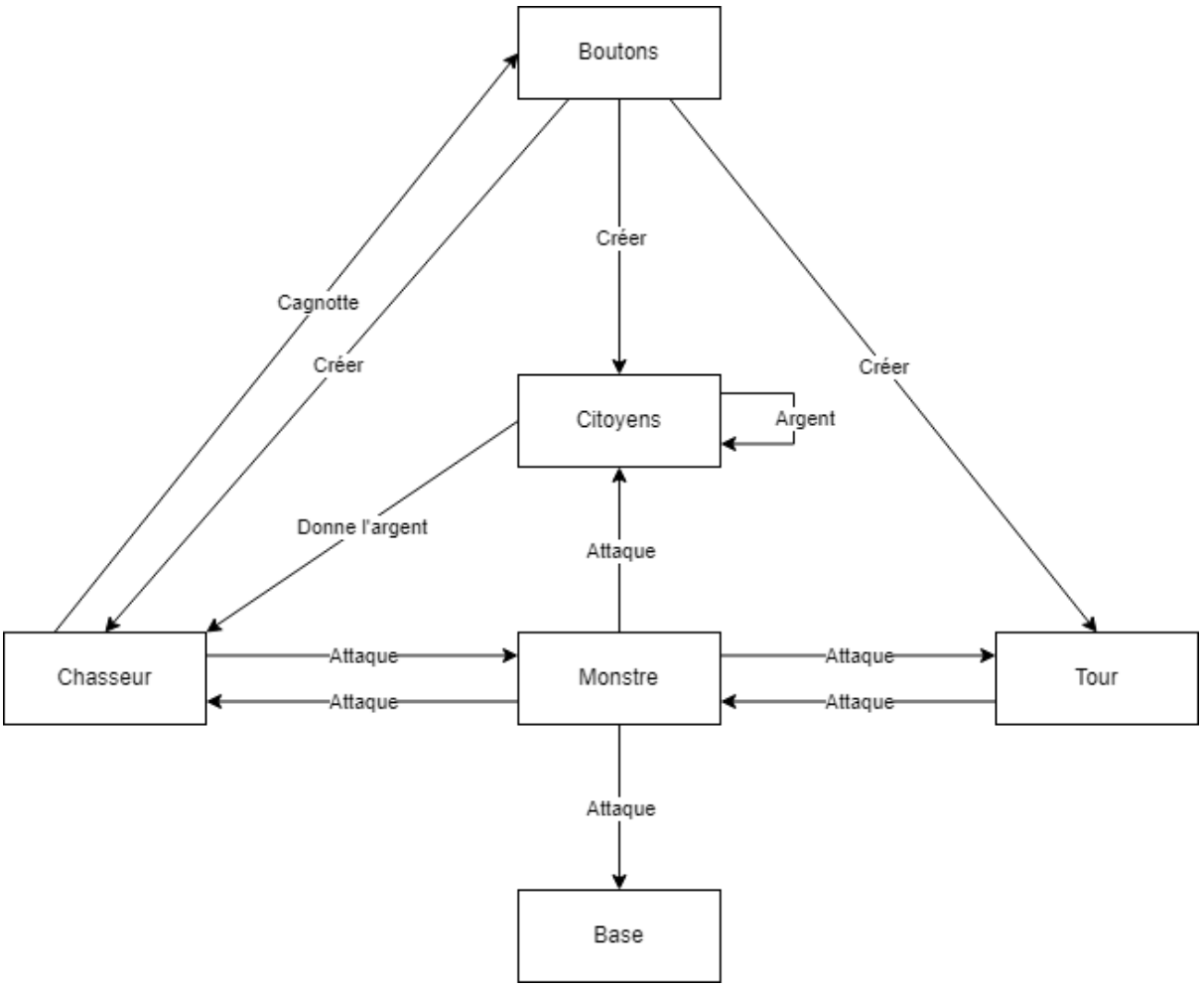
- Attaque des monstres sur les citoyens (1h)
- Attaque des chasseurs sur les monstres (1h)
- Augmentation de l'argent d'un citoyen (30 min)
- Organisation des boutons (2h)
- Mourir (chasseur) (30 min)

6ème semaine :

- Apparition aléatoire des monstres (45 min)
- Argent du citoyen vers le chasseur (1h)
- Utilisation des boutons en fonction de l'économie (15 min)
- Destruction de la base (10 min)
- Affichage des infos du jeu (argent, PV Base, légende) (15 min)
- Arrêt du jeu lorsque la base est détruite (30 min)

Conception générale

Blocs fonctionnels :



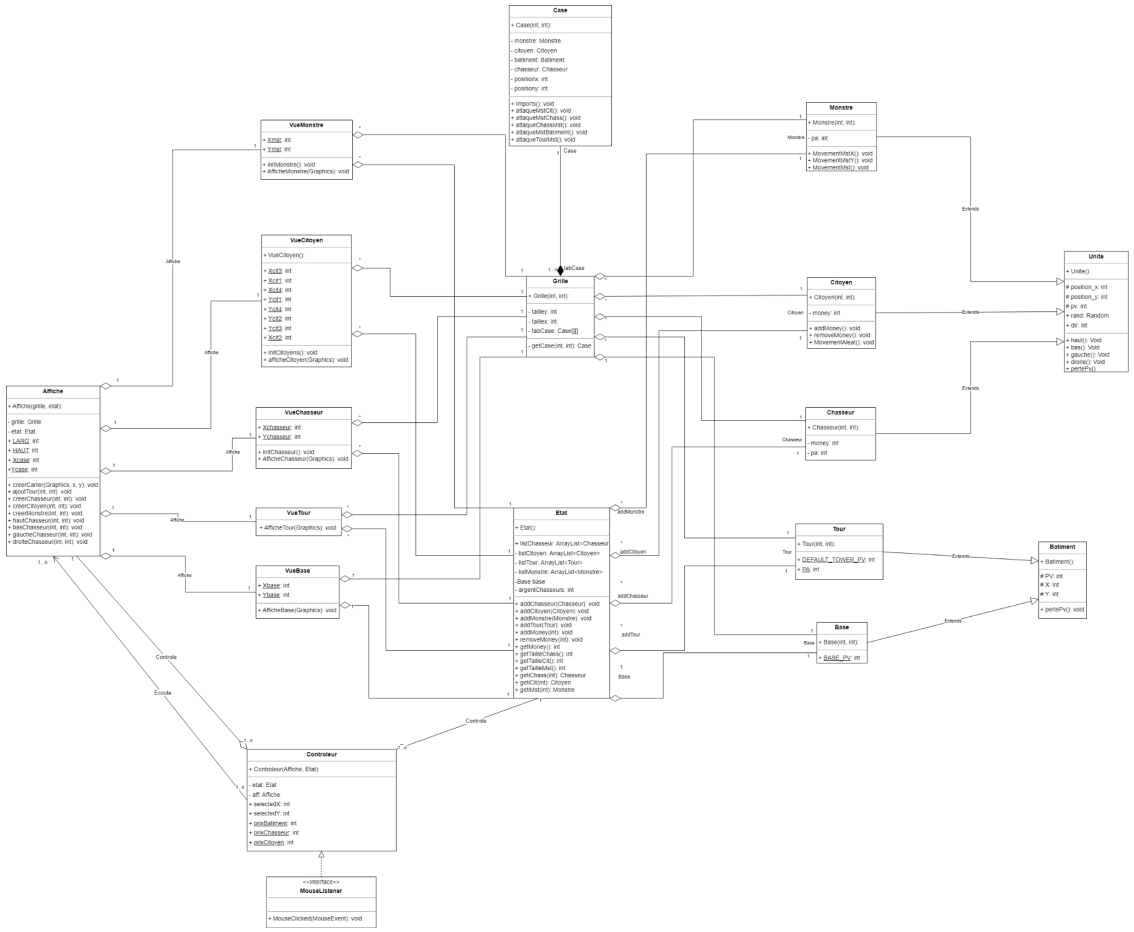
Nous avons adopté le motif MVC pour le développement du jeu.
Le motif MVC (Modèle, Vue, Contrôleur) est un motif de conception proposé par le génie logiciel permettant de structurer les classes en séparant chaque fonctionnalité :

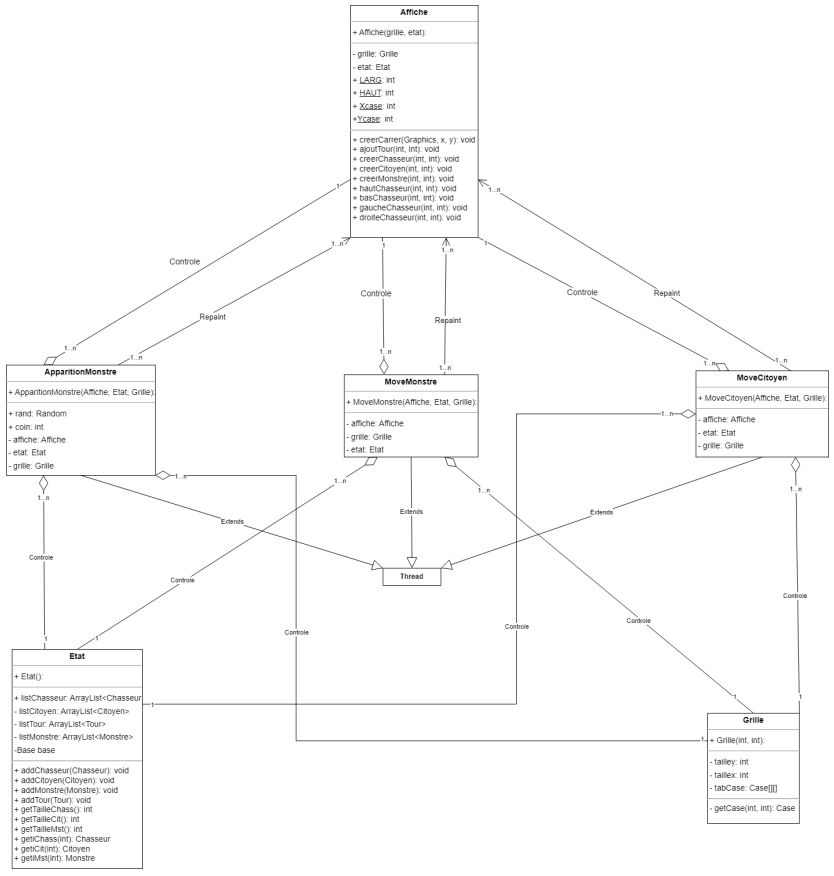
- Le Modèle est l'ensemble des données qui caractérisent l'état du jeu.
  La modification de ces données correspond à un changement de l'affichage dans l'interface graphique.
- La Vue définit l'affichage des éléments et de l'interface du jeu c'est-à-dire comment l'état du Modèle est rendu visible à l'utilisateur.
- Le Contrôleur est l'ensemble des supports d'intéractions de l'utilisateur. C'est ce qui fait le lien entre le Modèle et la Vue. Il effectue les changements dans le modèle et informe la vue d'un changement. Lorsque l'interface est interactive, le Contrôleur gère aussi les évènements.

Les avantages de l'utilisation de ce motif sont :

- Avoir des classes bien structurées pour faciliter la compréhension du développement du jeu.
- Lorsque un problème ou bug apparait, facilite l'identification de la source du problème et le débogage.

Conception détaillée





Création de la grille :
La fonctionnalité est implémentée dans la classe Grille. Elle utilise deux attributs taillex pour le nombre de case en longueur et tailley pour le nombre de case en largeur et un tableau à deux dimensions. Ce tableau est initialisé et rempli avec des objets de type Case dans le constructeur de la classe Grille.

Affichage de la grille :
La fonctionnalité est implémentée dans la classe Affiche. On créer dans la méthode paint deux boucles for imbriquées qui parcours le tableau à deux dimensions créer précédemment et pour chaque case du tableau, on appelle la méthode creerCarre qui créer un carré blanc de taille 30x30.

Création et initialisation de la base :
La fonctionnalité est implémentée dans la classe Base qui hérite de la classe Bâtiment. Elle possède un attribut Base_PV qui correspond au PV de la base. Elle a des coordonnées x et y. On initialise la base en l'ajoutant dans la grille.

Affichage de la base :
La fonctionnalité est implémentée dans la classe VueBase. On utilise une méthode AfficheBase qui créer un carré de couleur gris de taille 10x10 et le positionne au milieu de la grille. On appelle ensuite la méthode AfficheBase dans la méthode Paint de la classe Affiche.

Création et initialisation des tours :
La fonctionnalité est implémentée dans la classe Tour qui hérite de la classe Bâtiment.
Elle possède un attribut pour les points de vie et un autre pour les points d'attaque. Elle a des coordonnées x et y. On ajoute la tour dans la liste listTour et dans la case cliquée lors de l'activation du bouton Tour.

Affichage des tours :
La fonctionnalité est implémentée dans la classe VueTour. On utilise une méthode AfficheTour qui crée un carré vert, la positionne sur une case ne contenant pas de bâtiment et dans les limites de la grille. La méthode AfficheTour est appelée dans la méthode Paint de la classe Affiche.

Enregistrer les coordonnées des clics :
La fonctionnalité est implémentée dans la classe Controleur. On utilise deux attributs selectedX et selectedY pour stocker l'abscisse et l'ordonnée du clic. On utilise ces attributs en paramètres de méthode de création d'unités ou bâtiments lors d'un clic sur un bouton pour positionner une unité, bâtiment ou pour les déplacements du chasseur.

Initialisation d'un chasseur:
La fonctionnalité initChasseur() est implémentée dans la classe VueChasseur qui va utiliser les variables static Xchasseur et Ychasseur et ainsi créer un chasseur en l'ajoutant à la liste de chasseurs dans l'état et l'ajouter dans la grille grâce à ces coordonnées. L'objet chasseur a pour attribut les pa (points d'attaques) et hérite de la classe abstraite Unite. Cette classe a pour attribut les pv (points de vie) et les coordonnées de l'unité.

Initialisation des citoyens:
La fonctionnalité initCitoyens() est implémentée dans la classe VueCitoyen qui va utiliser les variables static Xcit, Ycit1 jusqu'à Xcit4, Xcit4 et ainsi créer des citoyens en les ajoutant à la liste de citoyens dans l'état via addCitoyen(X, Y) et les ajouter dans la grille grâce à leurs coordonnées via setCitoyen(X, Y). L'objet citoyen a pour attribut money et hérite de la classe abstraite Unite. Cette classe a pour attribut les pv (points de vie) et les coordonnées de l'unité.

Initialisation du monstre:
La fonctionnalité initMonstre() est implémentée dans la classe VueMonstre qui va utiliser les variables static XMst et YMst et ainsi créer un monstre en l'ajoutant à la liste de monstres dans l'état et l'ajouter dans la grille grâce à ces coordonnées. L'objet monstre a pour attribut les pa (points d'attaques) et hérite de la classe abstraite Unite. Cette classe a pour attribut les pv (points de vie) et les coordonnées de l'unité.

Initialisation des unités:
La fonctionnalité initUnite() est implémentée dans la classe Affiche et est utilisée dans son constructeur. Elle permet d'initialiser toutes les unités de base grâce aux classes d'initialisation de VueChasseur, VueCitoyen et VueMonstre.

Affichage des chasseurs:
L'affichage des chasseurs, afficheChasseur(Graphics), est implémentée dans la classe VueChasseur et permet de parcourir la liste de chasseurs de l'état, parcourue grâce à un for entre 0 et getTailleChass(), et ainsi dessiner les chasseurs en bleu avec leur coordonnées correspondantes, obtenues avec les getPosition_x() et getPosition_y() ajustées pour le dessin. La méthode afficheChasseur est appelée dans le paint de la classe Affiche.

Affichage des citoyens:
L'affichage des monstres, afficheCitoyen(Graphics), est implémentée dans la classe VueCitoyen et permet de parcourir la liste de citoyens de l'état, parcourue grâce à un for entre 0 et getTailleCit(), et ainsi dessiner les citoyens en orange avec leur coordonnées correspondantes, obtenues avec les getPosition_x() et getPosition_y() ajustées pour le dessin. La méthode afficheCitoyen est appelée dans le paint de la classe Affiche.

Affichage des monstres:
L'affichage des monstres, afficheMonstre(Graphics), est implémentée dans la classe VueMonstre et permet de parcourir la liste de monstres de l'état, parcourue grâce à un for entre 0 et getTailleMst(), et ainsi dessiner les monstres en orange avec leur coordonnées correspondantes, obtenues avec les getPosition_x() et getPosition_y() ajustées pour le dessin. La méthode afficheMonstre est appelée dans le paint de la classe Affiche.

Déplacement d'un chasseur:
La fonctionnalité est implémentée dans la classe Controleur et permet de déplacer le chasseur dans la direction souhaitée (bouton directionnel). Chaque bouton est composé de la même base: si les pv de la base sont supérieurs à zéro, la méthode de déplacement correspondant à la direction du bouton sera alors appelée (hautChasseur, basChasseur, gaucheChasseur ou droiteChasseur). Ces méthodes sont elles-mêmes implémentées dans

la classe Affiche et permettent d'utiliser les coordonnées du clic où se trouve le chasseur. Ainsi, cela permet de modifier la grille pour effacer le chasseur de ces coordonnées et le dessiner aux nouvelles. De même pour l'Etat, où le chasseur correspondant aux coordonnées du clics sont modifiés grâce aux méthodes directionnelles (haut(), bas(), gauche(), droite()). Le repaint permet ainsi de mettre à jour l'affichage.

Récupération de l'argent:
La fonctionnalité est implémentée dans la classe MoveCitoyen et permet de transférer l'argent d'un citoyen à la cagnotte totale lorsqu'un chasseur passe sur sa case. Pour se faire, avant de mettre en mouvement le citoyen, on parcourt la liste des citoyens avec un for de 0 à getTailleCit() et récuperer ses coordonnées (x,y). Il y a un parcours de la liste des chasseurs avec un foreach, pour vérifier s'il y a un chasseur dans la case grâce à la récupération des coordonnées du chasseur avec getPosition_x() (et y). Si c'est le cas, alors l'argent du citoyen est obtenu grâce au getMoney() et ajouter à la cagnotte grâce au addMoney(). L'argent du citoyen est ensuite mis à 0 grâce au removeMoney(). L'argent est affiché dans le paint de la classe Affichage grâce à un drawString.

Mort d'un chasseur:
La fonctionnalité, remove(i) permet de supprimer le chasseur à l'indice i de la liste de chasseur de l'état. Cette fonctionnalité est implémentée dans la classe MoveMonstre et permet de vérifier si les monstres ont leur pv au-dessus de zéro. Cette vérification s'effectue grâce à une boucle for qui va parcourir chaque monstre dans la liste des monstres de l'État et va stocker les coordonnées du monstre dans des variables x et y. Ainsi grâce à ces coordonnées et un foreach qui parcourt la liste des chasseurs, on vérifie si les pvs du chasseur, obtenu grâce à la fonction getPv(), sont inférieurs ou égaux à zéro, alors le chasseur est supprimés de l'état grâce à la méthode remove(i) (i l'indice du monstre), et est aussi supprimé de la case, de coordonnées (x,y), de la grille. Le repaint permet ainsi de mettre à jour l'affichage.

Création d'un citoyen:
La fonctionnalité creerCitoyen(x, y) s'implémente dans la classe Affiche et permet d'ajouter un nouvel objet Citoyen aux coordonnées (x,y), via addCitoyen(new Citoyen(x, y)) ainsi que l'ajout dans la grille à la case de ces coordonnées, via setCitoyen(new Citoyen(x, y)). L'ajout ne peut se faire que si la case correspondante est vide.

Création d'un chasseur:
La fonctionnalité creerChasseur(x, y) s'implémente dans la classe Affiche et permet d'ajouter un nouvel objet Chasseur aux coordonnées (x,y), via addChasseur(new Citoyen(x, y)) ainsi que l'ajout dans la grille à la case de ces coordonnées, via setChasseur(new Chasseur(x, y)). L'ajout ne peut se faire que si la case correspondante est vide.

Mouvement automatique d'un citoyen:
La fonctionnalité est implémentée dans la classe MoveCitoyen, et permet de mettre en mouvement les citoyens toutes les 3 secondes. On effectue le mouvement sur chaque citoyen de la liste de citoyens d'Etat, liste qui sera parcourue. Leur coordonnées actuelles sont sotckées dans des variables x et y. Le mouvement du citoyen s'effectue via l'État grâce à la méthode MovementAleat() dans laquelle la variable dir stock un random aléatoire entre 1 et 4 qui permet de déterminer la direction de déplacement du citoyen (et effectue la

modification dans l'État grâce aux méthodes haut(), bas(), gauche(), droite()). Le repaint permet ainsi de mettre à jour l'affichage.

Gain d'argent linéaire du citoyen:
La fonctionnalité est implémentée dans la classe MoveCitoyen et permet d'augmenter linéairement l'argent de chaque citoyen à chaque déplacement. Pour cela, on parcourt la liste de citoyens afin de faire l'opération sur chacun d'entre eux grâce à une boucle for entre 0 et getTailleCit(). Ainsi, pour chaque citoyen (getiCit(i), i l'indice de parcours) est utilisée la méthode add.Money() qui permet d'augmenter l'argent du citoyen de 5 à chaque déplacement.

Mort d'un citoyen:
La fonctionnalité est implémentée dans la classe MoveCitoyen et permet de vérifier si les citoyens ont leur pv au-dessus de zéro. Cette vérification s'effectue grâce à une boucle for, entre 0 et getTailleCit(), qui va parcourir chaque monstre dans la liste des monstres de l'État et va stocker les coordonnées actuelles du monstre dans des variables x et y. Ainsi, si les pvs du monstres, obtenu grâce à la fonction getPv(), sont inférieurs ou égaux à zéro, alors le citoyen est supprimés de l'état grâce à la méthode remove(i) (i l'indice du citoyen), et est aussi supprimé de la case, de coordonnées (x,y), de la grille. Le repaint permet ainsi de mettre à jour l'affichage.

Mouvement automatique d'un monstre:
La fonctionnalité est implémentée dans la classe MoveMonstre, et permet de mettre en mouvement les monstres toutes les 6 secondes. On parcourt chaque monstre de la liste de monstres d'Etat afin de tous les déplacer. Leur précédente coordonnées sont sotckées dans des variables x et y. Le mouvement du monstre s'effectue d'abord dans l'état grâce à la méthode MovementMst(). Cette méthode utilise elle-même deux méthodes MovementMstX() et MovementMstY() qui gèrent chacune le mouvement du monstre sur leur axe. Pour que le monstre continue d'aller vers la base, il n'a pas besoin d'aller au-delà de la valeur des coordonnées de la base. Grâce à ces deux fonctions, et au random aléatoire entre 1 et 2 stocké dans la variable dir (soit la direction sur X soit sur Y), le mouvement du monstre s'effectue de manière aléatoires sur une des deux directions mais reste concentrée vers la base. Le repaint permet ainsi de mettre à jour l'affichage.

Apparition aléatoire des monstres:
Cette fonctionnalité est implémentée dans la classe ApparitionMonstre et permet de faire apparaître aléatoirement les monstres sur les extrémités de la carte. Elle appelle dans un thread la fonction Apparition toutes 15 secondes puis effectue un repaint pour mettre à jour à l'affichage. Apparition utilise la variable spawn qui est affecté entre 1 et 4 aléatoirement et permet ainsi d'une part de choisir la ligne à faire apparaître. Lorsque cette ligne est choisie, creerMonstre est appelé et prend en paramètre une coordonnée correspondant à la ligne et au random de la localisation sur cette ligne.

Mort d'un monstre:
La fonctionnalité est implémentée dans la classe MoveMonstre et permet de vérifier si les monstres ont leur pv au-dessus de zéro. Cette vérification s'effectue grâce à une boucle for qui va parcourir chaque monstre dans la liste des monstres de l'État et va stocker les coordonnées actuelles du monstre dans des variables x et y. Ainsi, si les pvs du monstres,

obtenu grâce à la fonction getPv(), sont inférieurs ou égaux à zéro, alors le monstre est supprimés de l'état grâce à la méthode remove(i) (i l'indice du monstre), et est aussi supprimé de la case, de coordonnées (x,y), de la grille. Le repaint permet ainsi de mettre à jour l'affichage.

Attaque d'une tour sur un monstre:
La fonctionnalité est implémentée dans la classe MoveMonstre. Dans la méthode Run de la classe (étant donné que MoveMonstre hérite de Thread), toutes les 6 secondes chaque monstre se déplace. Lorsque cela se passe, pour chacun de ces monstres, le code analyse si les nouvelles coordonnées du monstre coïncident avec les coordonnées de chacune des tours présentes dans la grille, ainsi que des cases situés juste au dessus, en dessous, à gauche et à droite des tours. La quantité de vie perdue par le monstre correspond alors à la base de points d'attaque des tours donnée par la constante PA de la classe Tour.

Attaque d'un chasseur sur un monstre:
La fonctionnalité est implémentée dans la classe MoveMonstre. Dans la méthode Run de la classe, toutes les 7 secondes chaque monstre se déplace. Lorsque cela se passe, pour chacun de ces monstres, le code analyse si les nouvelles coordonnées du monstre coïncident avec les coordonnées de chacun des chasseurs présents dans la grille, ainsi que des cases situés juste au dessus, en dessous, à gauche, à droite, et en diagonale des chasseurs (ce qui ressemble à la zone d'action du roi aux échecs). La quantité de vie perdue par le monstre correspond alors à la base de points d'attaque des chasseurs donnée par la méthode getpa(), renvoyant l'attribut pa de la classe Chasseur.

Attaque d'un monstre sur un citoyen:
La fonctionnalité est implémentée dans la classe MoveCitoyen. Dans la méthode Run de la classe, toutes les 3 secondes, les citoyens se déplacent. Lorsque cela se passe, pour chacun des citoyens, si ses coordonnées sont identiques à celles d'un des monstres, alors le citoyen perd une quantité de vie égale aux points d'attaque du monstre obtenus via la méthode getpa() de la classe Monstre. Les coordonnées des monstres et des citoyens sont obtenues via les méthodes get_positionx() et get_positiony() de leurs classes respectives.

Attaque d'un monstre sur un bâtiment:
La fonctionnalité est implémentée dans la classe MoveMonstre. Dans la méthode Run de la classe, toutes les 6 secondes chaque monstre se déplace. Lorsque cela se passe, pour chacun de ces monstres, le code analyse si les nouvelles coordonnées du monstre coïncident avec les coordonnées de chacune des tours présentes dans la grille, ainsi que celles de la base. La quantité de vie perdue par le bâtiment correspond alors à la base de points d'attaque des monstres donnée par la méthode getpa() de la classe Monstre.

Attaque d'un monstre sur un chasseur:
La fonctionnalité est implémentée dans la classe MoveMonstre. Dans la méthode Run de la classe, toutes les 6 secondes chaque monstre se déplace. Lorsque cela se passe, pour chacun de ces monstres, le code analyse si les nouvelles coordonnées du monstre coïncident avec les coordonnées de chacune des tours présentes dans la grille, ainsi que celles de la base. La quantité de vie perdue par le chasseur correspond alors à la base de points d'attaque des monstres donnée par la méthode getpa() de la classe Monstre.

Destruction d’un bâtiment:
La fonctionnalité se trouve dans MoveMonstre, non loin de l’attaque d’un bâtiment par un monstre. Pour chacune des tours, si ses points de vie tombent à zéro, alors alors est supprimée de la grille et de la liste de tours d’Etat. Pour la base, si ses points de vie tombent à zéro, alors elle est supprimée de la grille et de la classe Etat, et étant donné que le dessin de la base dans la classe paint() de Affiche n’est accessible que si les points de vie de la base sont supérieurs à zéro, alors elle n’est plus affichée.

Achat d’une tour:
La fonctionnalité de l’achat de la tour est implémenté dans la classe Controleur, via un JButton et permet au joueur d’ajouter une tour sur la map s’il a assez d’argent et que le jeu ne s’est pas terminée. La vérification de la suffisance de l’argent se fait grâce à la récupération de l’économie via getMoney() moins la variable prixTour. Le tout doit être supérieur ou égal à zéro et dans ce cas alors on appelle ajoutTour(X, Y) aux coordonnées du clic, puis on retire à l’économie prixTour, le prix d’une tour. Un repaint permet ainsi de mettre à jour l’affichage.

Achat d’un chasseur:
La fonctionnalité de l’achat de la tour est implémenté dans la classe Controleur, via un JButton et permet au joueur d’ajouter un chasseur sur la carte s’il a assez d’argent et que le jeu ne s’est pas terminé. La vérification de la suffisance de l’argent se fait grâce à la récupération de l’économie via getMoney() moins la variable prixChasseur. Le tout doit être supérieur ou égal à zéro et dans ce cas, alors on appelle creerChasseur(X, Y) aux coordonnées du clic, puis on retire à l’économie prixChasseur, le prix d’un chasseur. Un repaint permet ainsi de mettre à jour l’affichage.

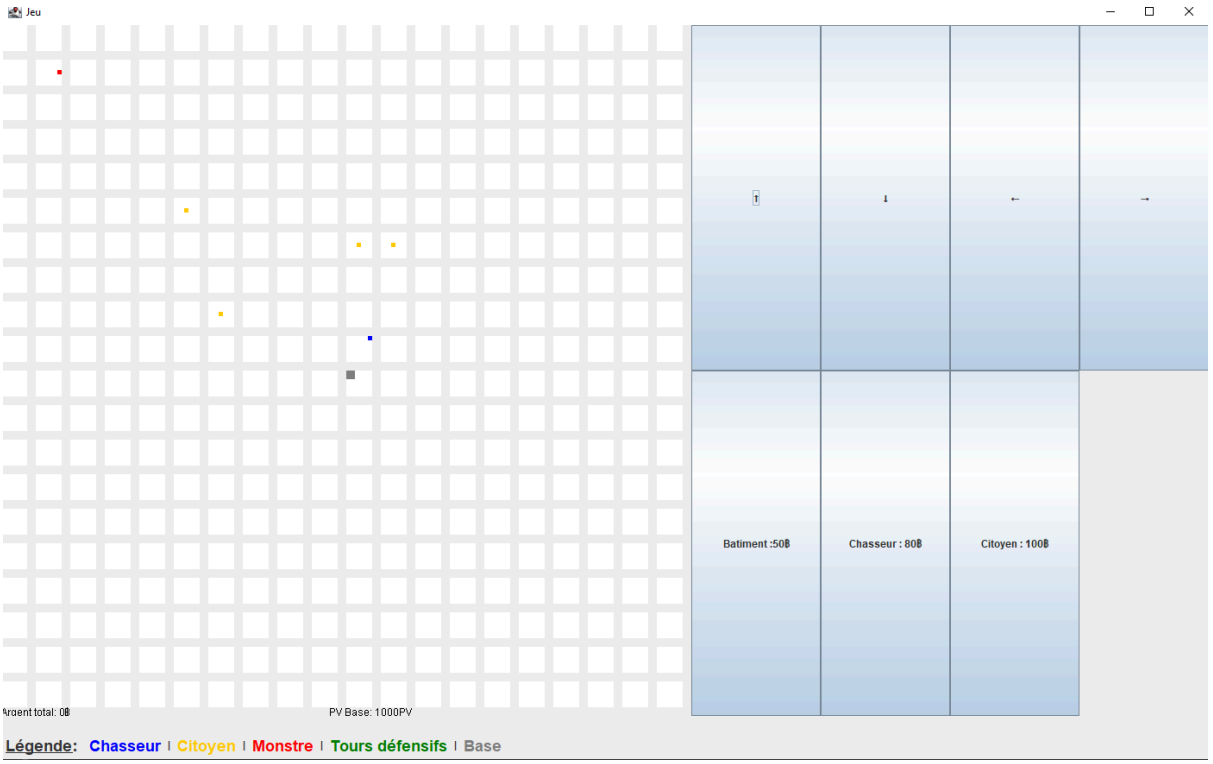
Achat d’un citoyen:
La fonctionnalité de l’achat de la tour est implémenté dans la classe Controleur, via un JButton et permet au joueur d’ajouter un citoyen sur la carte s’il a assez d’argent et que le jeu ne s’est pas terminé. La vérification de la suffisance de l’argent se fait grâce à la récupération de l’économie via getMoney() moins la variable prixCitoyen. Le tout doit être supérieur ou égal à zéro et dans ce cas, alors on appelle creerCitoyen(X, Y) aux coordonnées du clic, puis on retire à l’économie prixCitoyen, le prix d’un citoyen. Un repaint permet ainsi de mettre à jour l’affichage.

Affichage des infos du jeu (argent, PV de la base, légende) :
Les fonctionnalités de l’affichage de l’argent et des PV de la base sont implémentées dans la classe Affiche. On utilise la méthode drawString pour afficher l’argent et les PV de la base. La fonctionnalité de la légende est implémentée dans la classe Main. On utilise un JLabel contenant des balises HTML.

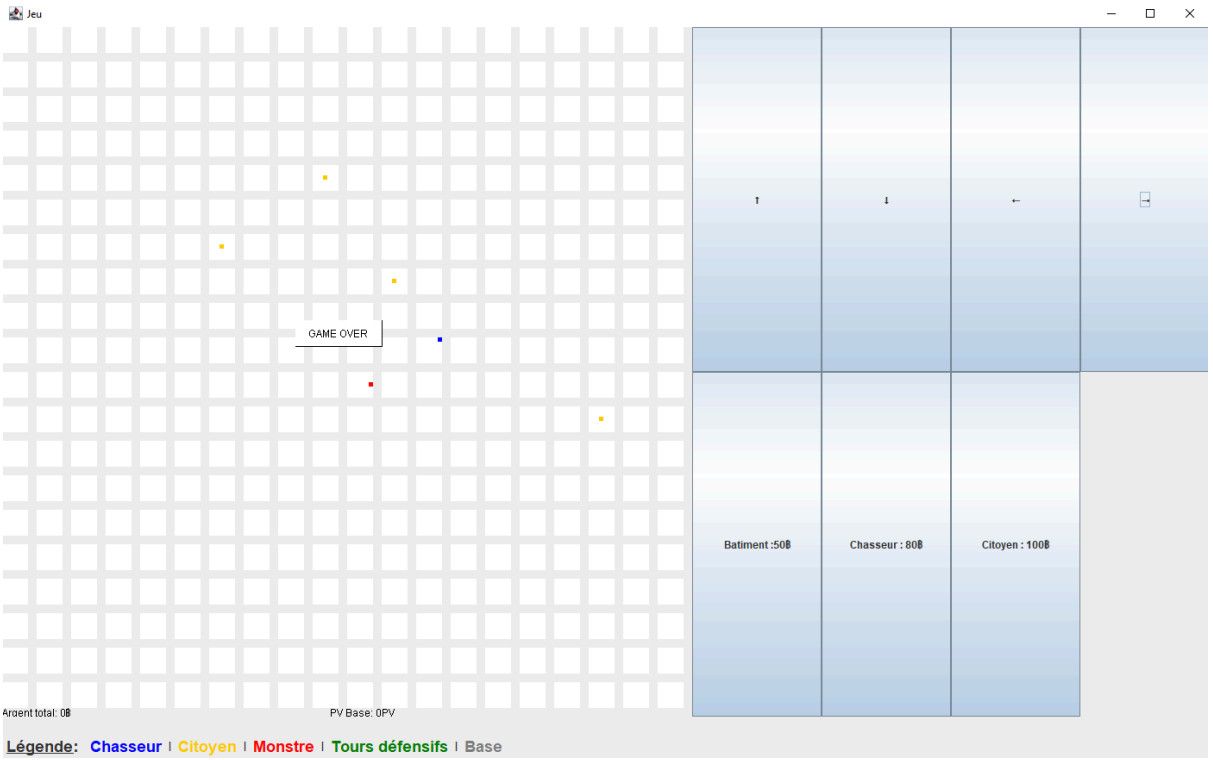
Arrêt du jeu lorsque la base est détruite :
Le principe de la fonctionnalité est de vérifier si les PV de la base sont toujours supérieurs à zéro. Cette vérification se fait via la méthode getPv utilisée sur l’objet base dans la classe Etat. La condition est vérifiée au niveau des threads (MoveMonstre, MoveCitoyen, ApparitionMonstre), au niveau des boutons et dans la méthode paint de la classe Affiche. Le tout permet d'arrêter le jeu, de rendre les boutons utilisables et afficher le Game Over.

Résultats

Capture d’écran au lancement du jeu :



Capture d’écran lors du Game Over:



Documentation utilisateur

Prérequis : Installer un IDE avec Java.

Mode d'emploi pour le lancement du jeu : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis "Run as Java Application". Le jeu se lance.

Règles du jeu :
La base est située au centre de la carte. Les monstres apparaissent à chaque intervalle de temps. Les tours attaquent les monstres situés dans les cases adjacentes de la tour. Les chasseurs attaquent les monstres sur les cases adjacentes et diagonales. Les citoyens se déplacent librement sur la carte et produisent de l'argent. Cet argent doit être récupéré par les chasseurs lorsqu'ils passent sur leur position. Les monstres attaquent tout ce qui est sur sa propre case (Base, Tour, Chasseurs et Citoyens).
Le jeu se termine si la base n'a plus de point de vie et un message *Game Over* apparaît.

Mode d'emploi du jeu :
Avant d'utiliser n'importe quel boutons situés sur la droite de la fenêtre, il faut d'abord cliquer sur une case du jeu.
Ensuite, vous pouvez effectuer différentes actions :
Les quatres boutons représentés par des flèches directionnelles permettent de déplacer un chasseur. Pour cela, cliquez sur une case ou se situe un chasseur (une case avec un carré bleu) puis choisissez et cliquez sur un des boutons de déplacement pour déplacer le chasseur dans la direction souhaitée.
Vous pouvez placer des unités ou bâtiments. Il y a donc trois boutons : Tour, Chasseur et Citoyen.
Cliquez sur une case libre, choisissez et cliquez sur un des trois boutons. Notez qu'il faut avoir assez d'argent pour pouvoir placer une tour ou unité.
Pour quitter le jeu, cliquez sur la croix située en haut à droite de la fenêtre du jeu.
Si vous souhaitez rejouer, quittez le jeu et effectuez cette étape : *sélectionnez la classe Main à la racine du projet puis "Run as Java Application".*

Documentation développeur

Consulter les classes Grille, Case et Etat où sont contenues les données principales du jeu.
Consulter les classes contenues dans le package Affiche pour ce qui concerne l'affichage des différents éléments du jeu.
Consulter la classe Controleur pour voir les différents boutons.
Consulter les classes MoveMonstre et MoveCitoyen pour comprendre le fonctionnement des mouvements automatique des monstres et citoyens.
Consulter les classes Base, Batiment, Unité, Chasseur, Citoyen, Monstre pour voir les caractéristiques des différentes unités et bâtiments.
Consulter la classe Apparition Monstre pour comprendre la génération aléatoire des monstres.

Pour augmenter ou diminuer la difficulté du jeu, il est possible de modifier les variables des points de vie et d'attaque des tours, des monstres et des chasseurs, ainsi que les points de vie des citoyens.
On peut aussi modifier les variables de position de la base ou du chasseur qui apparaît au lancement du jeu.

Au départ il était prévu d'ajouter, parmi les bâtiments disponibles pour le joueur, des mines. Les mines auraient fonctionné comme les tours, mais avec un rayon d'action bien plus large, des dégâts très élevés et une utilisation unique. C'est-à-dire que la mine aurait été supprimée aussitôt activée.
Ensuite a été tenté de modifier le mécanisme de déplacement des chasseurs: au lieu de cliquer sur la case puis de cliquer sur un bouton de déplacement, il aurait dû être possible de déplacer un chasseur en cliquant sur la case, puis en appuyant sur une des flèches de direction du clavier (avec l'utilisation de KeyListener avec MouseListener.

Conclusion et perspectives

Nous avons pu réaliser la plupart des fonctionnalités que nous avions prévu de faire.
Les seules fonctionnalités qui manquent sont le déplacement automatique des chasseurs (grâce à un algo a*search par exemple) ainsi que l'affichage d'une fenêtre lorsque la partie se termine avec le Game Over sur la nouvelle fenêtre avec la possibilité de rejouer grâce à cette fenêtre.

Les principales difficultés rencontrées ont été le strict respect du modèle MVC qui créait de temps en temps des problèmes de communication variables et de leurs valeurs entre les classes. La synchronisation entre les versions a également créé des problèmes étant donné que chacun devrait posséder une version à jour du code. La solution à ce problème a été d'utiliser Git, néanmoins il n'était pas toujours facile de faire communiquer du code Java produit sous Eclipse avec Git.
Ce projet nous a permis d'apprendre à utiliser les threads, à structurer un projet qui comportent énormément de fonctionnalités et de classes grâce au modèle MVC.

On peut imaginer bien des façons de faire évoluer ce code. Tout d'abord il serait possible d'implémenter les fonctionnalités prévues ou envisagées mais non intégrées au produit final. On pourrait aussi songer à créer un menu propre et développé permettant entre autre de changer le niveau de difficulté en modifiant les constantes (ce qui permettrait alors d'avoir un jeu plus dur avec des monstres plus solides, infligeant plus de dégâts). Ainsi que l'esthétisme de l'affichage, c'est-à-dire, remplacer les carrés par des images, utiliser un système de fenêtre dynamique (même s'il l'on change la taille de la fenêtre, les dimensions de l'affichage s'adaptent).