Rapport PCII

MAHREZ Ali
KUATE FODOUOP Adrien
AGGAB Maryam
AMOSSÉ Hugo

Sommaire :

1 - Introduction

Le projet consiste à développer un jeu vidéo de stratégie en temps réel inspiré de grands jeux tels que Dune 2000, Age of Empire, Warcraft et Starcraft. Le jeu sera en mode mono-joueur, où le joueur contrôle les actions d'unités sur une carte en vue du dessus. Le but du jeu est de maintenir le village en vie pour atteindre un objectif économique en faisant face à un environnement hostile où des monstres attaquent le village. Le jeu inclura les mécanismes de déplacement d'unités, de construction de bâtiments, de collecte de ressources et de "temps réel". Les villageois doivent collecter des ressources pour construire et réparer des bâtiments et protéger le village contre les monstres en attaque. Les villageois peuvent collecter plusieurs ressources différentes dans le jeu tel que :

Des légumes : les villageois peuvent récolter des légumes en cultivant des potagers qui pourront être vendus pour faire des potions servant à redonner des points de vie aux bâtiments.

De l'or : les villageois peuvent trouver de l'or en tuant des monstres. L'or est utilisé pour améliorer les bâtiments ou acheter dans le shop (légumes et bâtiments).

L'interaction avec l'utilisateur se fait à travers une interface graphique constituée de sept boutons qui permettent de manipuler les actions effectuées par les villageois.

Nous aurons une grande liberté dans la conception du jeu, tout en respectant les contraintes fixées pour le projet. Le modèle MVC sera utilisé pour organiser le développement du jeu. Ce rapport de projet décrit les différentes étapes de développement du jeu, de la conception à la mise en œuvre.

2 - Analyse globale:

Nos principales fonctionnalités à développer sont les suivantes :

• Villageois : Les villageois sont des unités contrôlables par le joueur, dont le but est de collecter des ressources et de défendre le village. Pour implémenter cette unité, plusieurs fonctionnalités sont nécessaires :

- • Déplacement : Le joueur doit pouvoir sélectionner le villageois et la destination, et l'unité doit se déplacer automatiquement jusqu'à la destination choisie.
- • Collecte de ressources : Cette action consiste à récupérer les ressources d'un point précis et à les ajouter aux ressources du village. Elle ne peut être effectuée que si le villageois se trouve à proximité de la ressource.
- • Attaque : Cette action consiste à réduire les points de vie d'un monstre jusqu'à sa mort.
- • Inactivité : Le villageois ne peut effectuer aucune action et se déplace aléatoirement sur la carte.

    • Plantation : Cette action permet de planter le légume choisi dans le potager. Le villageois doit se trouver à proximité du potager pour la réaliser.

    • Récolte : Cette action permet de collecter les légumes et de vider le potager. Le villageois doit se trouver à proximité du potager pour la réaliser.

• Défenses : Les défenses sont des bâtiments non-contrôlables gérés par un thread. Ils permettent de défendre le village contre les monstres.

    • Détection de monstres : Le bâtiment est doté d'un rayon qui lui permet de détecter la présence d'une unité de type monstre dans ce rayon.

    • Attaque : Cette action consiste à réduire les points de vie des monstres dans le rayon jusqu'à leur mort.

• Potager : Le potager est un bâtiment non-contrôlable géré par un thread. Il permet de planter des légumes pour obtenir des ressources.

    • Culture : Cette action permet de faire pousser les légumes qui se trouvent dans le potager.

• Monstres : Les monstres sont des unités non-contrôlables par le joueur. Leur objectif est d'attaquer les différents bâtiments pour détruire le village.

    • Recherche de bâtiments : L'unité cherche le bâtiment le plus proche à détruire.

    • Déplacement : L'unité se déplace automatiquement vers le point de destination.

    • Attaque : Cette action consiste à réduire les points de vie des bâtiments jusqu'à leur destruction.

• Cabanes : Les cabanes sont des bâtiments qui permettent de rajouter des villageois dans le village. Plus il y a de cabanes, plus le village aura de travailleurs (croissance démographique).

• Hôtel de ville : Il permet de poser une limite à l'amélioration des bâtiments ou des villageois. Plus le niveau de l'hôtel de ville est élevé, plus il sera possible d'améliorer les défenses. En revanche, plus l'hôtel de ville est élevé, plus les monstres seront forts.

• Panneau de contrôle : Le panneau de contrôle est constitué de différents boutons qui permettent de manipuler les actions effectuées par les villageois. Voici les fonctionnalités qu'il propose :

    • Construire : cette fonctionnalité permet de construire différents bâtiments qui vont servir au développement du village. En cliquant sur ce bouton, le joueur se retrouve dans l'inventaire et sélectionne le bâtiment qu'il souhaite construire (doit l'avoir acheté au préalable dans le shop) et le positionne dans son village.

    • Planter : cette fonctionnalité permet de planter des légumes dans le potager. En cliquant sur ce bouton, le joueur se retrouve dans l'inventaire et sélectionne le légume qu'il souhaite planter (doit l'avoir acheté ou récolté au préalable dans le shop) puis sélectionne un potager libre de son village.

• Récolter : cette fonctionnalité permet aux villageois de récolter les légumes dans le potager. En cliquant sur ce bouton, un villageois va se rendre au potager et récolter les légumes.

• Attaquer : cette fonctionnalité permet aux villageois d'attaquer les monstres sélectionnés. En cliquant sur ce bouton, le joueur doit sélectionner le monstre à attaquer , puis sélectionner le villageois qu'il souhaite utiliser pour l'attaque.

• Ramasser : cette fonctionnalité permet aux villageois d'aller ramasser l'argent qui est laissé par les monstres lorsqu'ils sont tués. En cliquant sur ce bouton, le joueur sélectionne une pièce à ramasser, puis sélectionne le villageois qui va la ramasser.

• Améliorer : cette fonctionnalité permet de sélectionner ce que l'on souhaite améliorer dans le village. Le prix de l'amélioration est affiché et le joueur doit décider s'il souhaite procéder à l'amélioration ou renoncer.

• Utiliser : cette fonctionnalité permet de redonner des points de vie à un bâtiment qui a été détruit par un monstre. En cliquant sur le bouton, le joueur sélectionne une potion puis sélectionne le bâtiment.

• Ressources : Les ressources constituent une monnaie d'échange qui permet le développement du village. Elles peuvent être utilisées pour améliorer ou réparer les bâtiments mais aussi pour acheter de nouveaux bâtiments. Il y a deux types de ressources dans le jeu :

• L'argent : cette ressource permet d'acheter des légumes ou des bâtiments et de les améliorer.

• Les légumes : les légumes récoltés peuvent être vendus sur le marché pour obtenir des potions .

• Interface graphique : Elle est conçue de manière à fournir une expérience utilisateur intuitive pour contrôler et surveiller les fonctionnalités du système. Elle est composée de plusieurs éléments clés, notamment :

- La carte : Elle sera placée au milieu de l'écran. Elle permet d'afficher les différents éléments du jeu (villageois, bâtiments, monstres, ….)

- Boutons : Placé en dessous de la carte, c'est des éléments interactifs qui vont permettre de contrôler le jeu et donc d'effectuer des actions spécifiques.

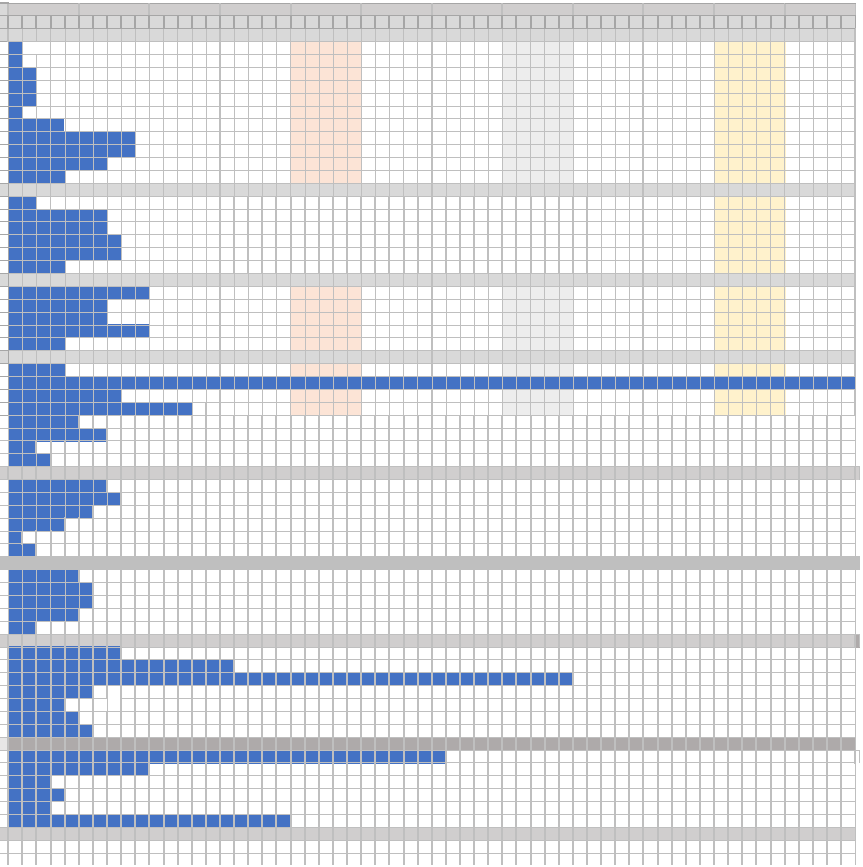
- Panneau d'affichage : Placé à droite de la carte. Il y est affiché l'argent et le shop. Le shop permet au joueur d'acheter des légumes à planter et différents types de bâtiments pour améliorer le village. On peut également vendre des légumes.

-Inventaire : L'inventaire se situe à gauche de la carte. C'est un catalogue de tous les objets achetés ou récoltés par le joueur. Pour effectuer des actions telles que planter des carottes, il faudra naviguer dans cet inventaire afin de pouvoir finaliser l'action.

3 – Plan de développement

Afin de mieux visualiser, l'ensemble des tâches liées au projet (fonctionnalités, …) nous proposons le diagramme de Gantt si dessous :

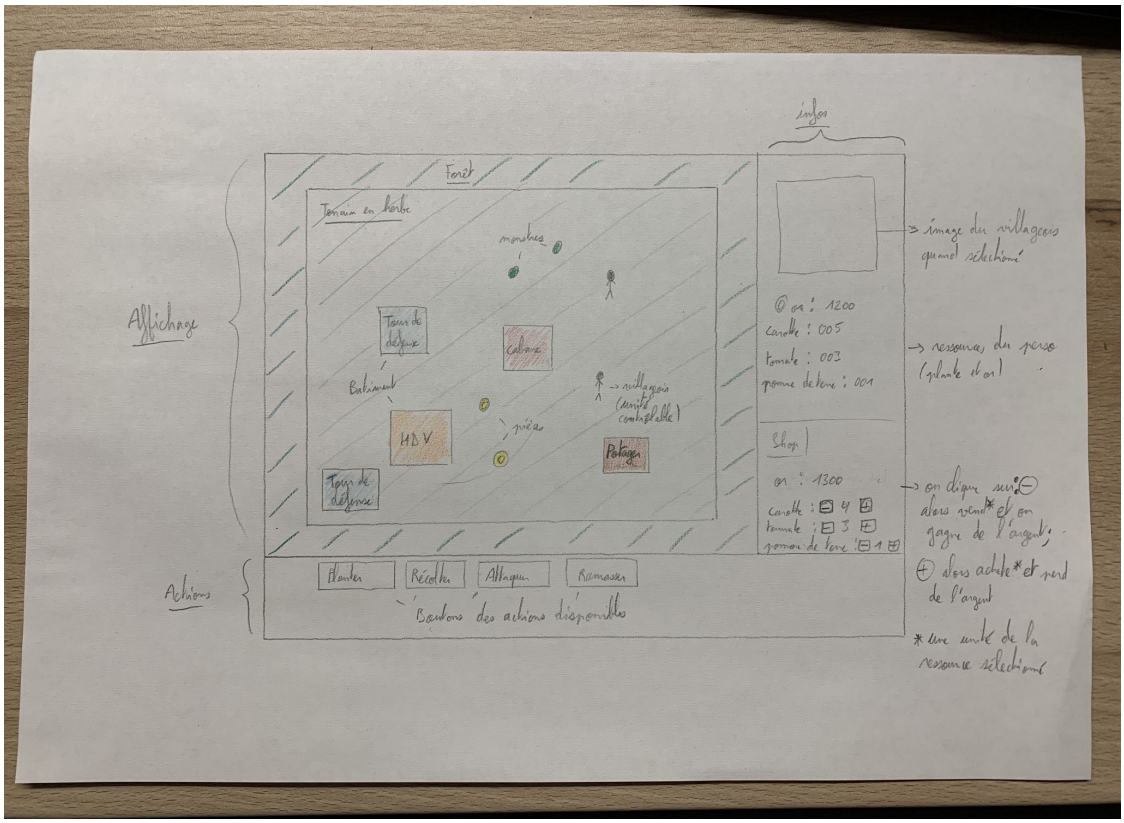
TITRE DE LA TÂCHE	PROPRIÉTAIRE DE LA TÂCHE	QUANTITÉ DE TRAVAIL EN HEURES heures
SEMAINE 1		
Compréhension du projet	Hugo, Ali, Maryam, Adrien	0h30
Définir le concept du jeu	Hugo, Ali, Maryam, Adrien	0h30
Définir les règles du jeu	Hugo, Ali, Maryam, Adrien	1h
Rédaction du cahier des charges et plan de développement	Hugo, Ali, Maryam, Adrien	1h
Répartition des tâches pour le développement	Hugo, Ali, Maryam, Adrien	0h30
Conception de l'architecture séparatant le modèle et la vue	Hugo, Ali, Maryam, Adrien	0h30
Création de la fenêtre graphique et du pannel d'affichage	Maryam	2h
Mise en place du plateau de jeu (grille et cases)	Adrien	3h
Implémentation des villageois	Ali	3h
Ajout des boutons d'interaction pour le joueur	Hugo	2h
Documentation du rapport	Ali, Adrien, Maryam, Hugo	1h
SEMAINE 2		
Debuggage et optimisation	Adrien, Hugo,	1h
Implémentation des cabanes et leur vue	Maryam	3h30
Implémentation des bâtiments et leur vue (hôtel de ville)	Hugo	3h30
Implémentation des vues bâtiments et leur vue (potagers)	Adrien	4h
Implémentation des monstres et de leur déplacement	Ali	4h
Rédaction et documentation du rapport	Hugo, Maryam	2
SEMAINE 3		
Implémentation du déplacement inactif des villageois	Ali	5h
Implémentation des monstres et de leur vues	Hugo	3h30
Implémentation du batiment de défense et sa vue	Maryam	3h30
Ajout des fonctionnalités pour la plantation	Adrien	6h
Rédaction et documentation du rapport	Maryam, Hugo, Ali, Adrien	2h
SEMAINE 4		
Debuggage et optimisation de chaque fonctionnalité	Hugo, Ali, Maryam, Adrien	2h
Implémentation du déplacement Astar	Ali	25h
Implémentation du déplacement des villageois	Ali	4h
Implémentation du déplacement des monstres	Hugo	6h
Fonctionnalité de détection des défenses	Maryam	3h
Fonctionnalité d'attaque des défenses	Adrien	3h
Tester et corriger les problèmes d'interaction entre les différentes fonctionnalités.	Hugo, Ali, Maryam, Adrien	1h
Rédaction et documentation du rapport	Hugo, Adrien, Ali, Maryam	1h30
SEMAINE 5		
Implémentation de l'attaque des monstres	Ali	3h30
Implémentation de l'attaque des villageois	Maryam	4h30
Implémentation de l'argent et de sa vue	Hugo	3h
Fonctionnalité de ramasser des pièces	Adrien	2h00
Debuggage et optimisation de chaque fonctionnalité	Adrien, Ali, Maryam, Hugo	0h45
Rédaction et documentation du rapport	Ali, Maryam	1h00
SEMAINE 6		
Peaufiner l'interface utilisateur.	Maryam	2h30
Implémentation de la console	Hugo	3h00
Fonctionnalité planter du villageois	Ali	3h00
Fonctionnalité récolter du villageois	Adrien	2h30
Rédaction et documentation du rapport	Adrien, Hugo	1h30
SEMAINE 7		
Debuggage et optimisation de chaque fonctionnalité	Ali, Adrien, Maryam, Hugo	4h
Intégration et conception de l'inventaire	Adrien	8h
Intégration et conception du marché	Ali, Hugo	26h
Mise en place du système de gestion des ressources (légumes et argent)	Adrien	3h
Optimisation, tests et ajustements pour le système de gestion des ressources.	Hugo, Adrien	2h
Intégrer le système de choix des légumes	Maryam	2h30
Rédaction et documentation du rapport	Hugo, Ali, Adrien, Maryam	3h
SEMAINE 8		
Fonctionnalité de construction de batiments	Ali, Maryam	14h
Effectuer des tests de jeu complets pour débusquer les bugs et les problèmes de performance.	Ali Maryam, Hugo, Adrien	5h
Dernières optimisations et tests pour les villageois et les monstres.	Hugo	1h45
Dernières optimisations et tests pour les défenses et le potager.	Ali	2h
Dernières optimisations et tests pour l'interface graphique et le panneau de contrôle.	Maryam, Adrien	1h45
Rédaction et documentation du rapport	Ali Maryam, Hugo, Adrien	10h



4 – Conception Générale

Notre jeu est conçu avec un objectif clair : collecter des ressources pour améliorer notre village et atteindre l'hôtel de ville niveau 5 sans qu'il ne soit détruit par les monstres. Pour y parvenir, les joueurs peuvent utiliser plusieurs méthodes pour collecter des ressources, telles que l'envoie de villageois pour en chercher, la destruction des monstres, ou la fin de la construction ou de l'amélioration d'un bâtiment.
Cependant, les monstres constituent une menace sérieuse pour le village, car ils ont la capacité de détruire tous les bâtiments les plus proches d'eux. Par conséquent, il est crucial de construire des bâtiments de défense pour protéger le village. Si les monstres réussissent à détruire l'hôtel de ville, la partie est perdue.
Les villageois sont les unités contrôlables dans le jeu. Ils peuvent être déplacés autour du village pour effectuer des actions telles que la construction, l'amélioration, la plantation, la récolte et l'attaque. Toutes ces actions ne peuvent être réalisées que si le villageois est à proximité de la cible.
En somme, notre conception globale met l'accent sur la collecte de ressources, la protection du village contre les monstres, la construction de bâtiments de défense et l'utilisation stratégique des villageois pour améliorer le village. Les joueurs peuvent interagir avec le jeu à travers un panneau de contrôle, qui leur permet de choisir les actions qu'ils souhaitent effectuer.

Voici un schéma permettant de mieux visualiser le rendu final imaginé au début du projet :



5 – Conception Détaillée

Villageois :

• Déplacement :

Pour implémenter le déplacement des entités dans notre jeu Java, nous avons choisi d'utiliser l'algorithme de recherche de chemin A*. Cet algorithme est particulièrement adapté pour trouver le chemin le plus court entre deux points sur une grille, ce qui correspond parfaitement à notre besoin de déplacer nos entités sur la carte de jeu.
L'algorithme A* fonctionne en utilisant une fonction heuristique pour évaluer le coût de déplacement entre les cases de la grille. Cette fonction estime le coût de déplacement restant entre une case donnée et la case d'arrivée, ce qui permet à l'algorithme de rechercher le chemin le plus court de manière plus efficace que d'autres algorithmes de recherche de chemin.
Pour implémenter cet algorithme dans notre jeu, nous avons créé une classe qui calcule le chemin à partir d'une position de départ et d'une position d'arrivée. Cette classe utilise un Thread pour effectuer les calculs de manière asynchrone et éviter que le jeu ne se bloque pendant la recherche de chemin.
L'algorithme commence par initialiser les cases de départ et d'arrivée, ainsi que deux ensembles de cases : l'ensemble "ouverts" et l'ensemble "fermés". L'ensemble "ouverts" contient toutes les cases qui ont été explorées mais dont les voisins n'ont pas encore été examinés, tandis que l'ensemble "fermés" contient toutes les cases qui ont été explorées et dont les voisins ont été examinés. Au début, seul le point de départ est ajouté à l'ensemble "ouverts".
Ensuite, l'algorithme parcourt les cases en itérant jusqu'à ce que l'ensemble "ouverts" soit vide. À chaque itération, la case ayant le coût minimum (le coût de déplacement de la case de départ à la case courante plus l'estimation heuristique du coût de déplacement de la case courante à la case d'arrivée) est extraite de l'ensemble "ouverts" et ajoutée à l'ensemble "fermés". Si la case extraite est la case d'arrivée, alors le chemin le plus court a été trouvé et l'algorithme reconstruit le chemin en remontant les prédécesseurs de la case courante jusqu'à la case de départ.
Si la case extraite n'est pas la case d'arrivée, alors l'algorithme examine les voisins de cette case. Si un voisin est déjà dans l'ensemble "fermés", cela signifie que le chemin le plus court vers ce voisin a déjà été trouvé, donc il est ignoré. Sinon, le coût pour aller de la case de départ à ce voisin est calculé et comparé au coût actuel pour aller de la case de départ à ce voisin. Si le nouveau coût est inférieur, ou si le voisin n'est pas dans l'ensemble "ouverts", alors le coût et le prédécesseur du voisin sont mis à jour, et le voisin est ajouté à l'ensemble "ouverts".

Pour représenter le chemin obtenu par l'algorithme A*, nous avons utilisé une ArrayList de cases qui représente les cases à parcourir pour aller de sa position de départ à la position d'arrivée. Ensuite nous avons une autre classe (DéplacementVillageois) qui s’occupe du déplacement du villageois en utilisant un thread et le chemin à parcourir. Étant donné que le calcul de chemin donné par A* vérifie que la case est libre pour l’ensemble des cases occupées par le villageois, ce système facilite le déplacement et permet au villageois de se déplacer directement

vers sa position cible. Une fois arrivé à sa position, le villageois effectue l'action qui lui a été demandée par l'utilisateur.

• Attaque :

Une fois qu'un villageois et un monstre ont été sélectionnés dans le jeu, le villageois peut attaquer le monstre en utilisant la méthode d'attaque qui déclenche le thread AttaqueVillageois. Cette classe prend en paramètre le villageois et le monstre et commence l'attaque sur le monstre. L'objectif est que le villageois attaque le monstre jusqu'à ce qu'il soit mort.
Pour cela, chaque villageois a un attribut de dégât qui permet de connaître le nombre de dégâts qu'il peut infliger aux monstres. Les monstres ont également une méthode subitDegat qui prend le nombre de dégâts en paramètre et réduit leur nombre de points de vie jusqu'à atteindre 0, qui est l'état auquel ils sont considérés comme morts.
Dans le jeu, des sprites ont été ajoutés pour rendre l'attaque plus visuelle. Pendant l'attaque, le villageois change d'état à chaque intervalle de temps pour donner l'effet qu'il effectue une véritable attaque.
La classe AttaqueVillageois utilise un thread pour que l'attaque se déroule de manière asynchrone par rapport au reste du jeu. La boucle while et les deux compteurs gèrent la vitesse à laquelle l'attaque se déroule et déterminent quand le monstre doit subir des dégâts.
Une fois que le monstre est mort, la classe remet le villageois à son état normal qui est le mode inactif et il reprend son déplacement dans la grille. Le monstre, quant à lui, est enlevé de la grille.
En résumé, la classe AttaqueVillageois permet de gérer l'attaque d'un villageois sur un monstre. Elle utilise un thread pour permettre une attaque asynchrone, gère la vitesse et les dégâts infligés pendant l'attaque, et remet le villageois à son état normal une fois que le monstre est mort.

• Inactivité :

Pour simuler les mouvements aléatoires du villageois sur la grille lorsqu'il n'a pas d'action à effectuer, nous utilisons la classe DeplacementHasard. Cette classe est un thread qui comprend les éléments suivants : l'objet Villageois qui va se déplacer, l'objet Grille sur laquelle les déplacements se font, ainsi que deux verrous, lockDeplacement et lockEtatSelectionnee.
Le thread est initialisé au début du jeu et continue à s'exécuter en boucle sans s'arrêter. La boucle while vérifie si le villageois a une action à effectuer. Si ce n'est pas le cas, le villageois se déplace de manière aléatoire. Pour cela, il choisit une direction au hasard en utilisant la méthode changeDirection(). Cette méthode utilise le verrou lockDeplacement pour changer la direction après un certain nombre de répétitions, ce qui rend le jeu plus réaliste.
Une fois la direction choisie, le villageois commence à se déplacer. Selon la direction choisie, il faut vérifier si la case cible est libre. Pour cela, nous examinons toutes les cases occupées par le villageois plus une case supplémentaire dans la direction choisie. Si la case cible est libre, le villageois s'y déplace et la grille est notifiée de ce changement. La vue est alors mise à jour grâce à la méthode notifieObservers(), ce qui permet une animation fluide du déplacement.

Nous utilisons plusieurs sprites pour donner un effet réaliste au déplacement du villageois. À chaque déplacement, nous voulons changer l'état de marche du villageois. Pour garantir une animation fluide, nous avons ajouté un deuxième verrou, lockEtatSelectionnee, qui permet de changer l'état de marche uniquement après un certain temps.

• Plantation :

Une fois que l'utilisateur a sélectionné le villageois, le potager et le légume à planter, et que le villageois se trouve à proximité du potager, la classe Planter est appelée pour effectuer l'action de plantation. Pour permettre une exécution asynchrone par rapport au jeu, un thread est utilisé.
La classe Planter a pour rôle de modifier l'état du potager pour donner l'impression que le villageois plante des graines dans le potager. Pour cela, le thread met à jour l'état du potager à intervalle de temps régulier, passant de l'état initial à l'état où toutes les cases du potager ne contiennent rien à l'état final de plantation dans lequel elles contiennent toutes des graines. Cet effet visuel donne l'impression que le villageois est vraiment en train de planter les graines. Une fois la plantation terminée, le villageois redevient inactif et se déplace de manière aléatoire dans la grille.
La classe Planter permet donc au joueur de planter un légume dans le potager sélectionné en utilisant un thread pour mettre à jour l'état du potager de manière asynchrone. Une fois la plantation terminée, une autre classe prend le relais pour permettre au potager de se développer jusqu'à la récolte.

• Récolte :

La récolte des légumes est une étape cruciale dans notre jeu. Tout d'abord, le joueur doit sélectionner le villageois pour effectuer la récolte, puis le potager correspondant doit être à l'état de récolte, c'est-à-dire que les légumes qui y poussent doivent avoir atteint leur maturité.
Une fois que le villageois est à proximité du potager, l'action de récolte est déclenchée en appelant la classe "Récolter". Afin de permettre une exécution asynchrone par rapport au jeu, nous avons utilisé un thread dédié pour cette action.
La classe "Récolter" a pour rôle de modifier l'état du potager jusqu'à ce qu'il retrouve son état initial, c'est-à-dire vide. Pour cela, le thread met à jour les cases du potager à intervalles réguliers en passant par tous les états intermédiaires, depuis celui où toutes les plantes ont fini leur développement jusqu'à celui où le villageois retire chaque plante individuellement. Cette animation visuelle donne l'impression que le villageois récolte les légumes un par un, ce qui ajoute un aspect de réalisme au jeu.
En plus de cela, chaque légume récolté est ajouté à l'inventaire du joueur, augmentant ainsi sa récolte. Une fois la récolte terminée, le potager retrouve son état initial et le villageois redevient inactif, prêt à se déplacer sur la grille.
En somme, la classe "Récolter" permet aux joueurs de récolter des légumes dans les potagers sélectionnés en utilisant un thread pour mettre à jour l'état du potager de manière asynchrone et rendre l'expérience de jeu plus immersive. La récolte de légumes est une fonctionnalité importante dans notre jeu, et grâce à la classe "Récolter", elle est exécutée de manière efficace et réaliste.

• Collecte de ressources :

Dans notre jeu, l'argent est la principale ressource utilisée pour les achats et les ventes. Pour récupérer une pièce d'argent, il faut sélectionner à la fois la pièce et le villageois. Lorsque le villageois est à proximité de la pièce, il la ramasse automatiquement et la pièce est ajoutée à la trésorerie du village. La pièce disparaît ensuite de la grille de jeu. Une fois la pièce ramassée, le villageois redevient inactif et se déplace de manière aléatoire sur la grille.
Cette mécanique de jeu est importante car elle permet aux joueurs de gagner de l'argent et de l'utiliser pour acheter des objets ou des améliorations qui peuvent aider à progresser dans le jeu. En utilisant une approche de sélection et de récupération de la pièce par le villageois, nous avons créé une expérience de jeu immersive qui ajoute une couche de réalisme au jeu.

Défenses :

• Détection de monstres :

En ce qui concerne la détection des monstres dans notre jeu, nous avons implémenté une méthode dans le thread principal du jeu, appelé "ThreadJeu". Cette méthode s'appelle "verifieMonstreDansRayon()" et elle est chargée de détecter la présence d'un monstre dans le rayon de chaque bâtiment du jeu.
Lorsque cette méthode est appelée, elle parcourt la liste des monstres présents sur la grille et calcule la distance entre chaque monstre et le bâtiment en question. Si la distance est inférieure ou égale au rayon du bâtiment, alors le monstre est considéré comme étant dans le rayon du bâtiment et la méthode retourne cette information.
Plus précisément, pour chaque monstre, la méthode calcule la distance entre le centre du monstre et le centre du bâtiment en utilisant la formule de distance euclidienne. Si cette distance est inférieure ou égale au rayon du bâtiment, alors le monstre est considéré comme étant dans le rayon et sa position est enregistrée.
Il est important de noter que cette méthode est appelée à chaque itération de la boucle principale du jeu et qu'elle est utilisée pour déterminer si un bâtiment peut attaquer un monstre dans son rayon d'action. En utilisant cette méthode dans le thread principal, nous évitons d'avoir un thread dédié à la recherche de monstres, ce qui simplifie l'architecture de notre jeu.

• Attaque:

Une fois que le monstre est détecté par la tour de défense, le bâtiment entre automatiquement en mode d'attaque. Cela déclenche le thread de défense qui prend en paramètre l'objet "monstre" détecté et lui inflige des dégâts à intervalles réguliers, tant que le monstre est en vie ou que le bâtiment n'a pas été détruit par les monstres.
L'objet "bâtiment de défense" dispose d'un attribut spécifique nommé "dégât", qui permet de déterminer le nombre de dégâts que le bâtiment est capable d'infliger aux monstres. Ce nombre est défini en amont lors de la création du bâtiment de défense et est ajustable selon le niveau du bâtiment.
Une fois que le bâtiment de défense commence à attaquer le monstre, ce dernier fait appel à sa méthode "subitDegat" qui prend en paramètre l'attribut "dégât" de la tour de défense. Cette méthode permet de diminuer les points de vie du monstre à chaque intervalle de temps.
Pour rendre l'attaque du bâtiment de défense plus réaliste, nous avons ajouté plusieurs sprites qui s'affichent à l'écran. Ces sprites permettent de visualiser les différentes phases de l'attaque, de l'impact des projectiles jusqu'à la destruction du

monstre. Et pour déterminer ces différents états, nous avons ajouté un nouvel attribut qui correspond à l'état de l'attaque. Cet attribut change à chaque intervalle de temps ce qui permet de changer l'état de l'attaque en fonction de sa progression. Cela permet de rendre l'attaque plus réaliste.
Enfin, une fois que le monstre est mort, il est mis à null et sera enlevé de la grille de jeu dans le thread principal du jeu JeuThread. Cette étape permet de libérer de l'espace.

Potager :

• Culture :

Dans notre projet, la croissance des plantes est gérée par une classe appelée Potager. Une fois que le villageois a planté les graines, la classe Potager est lancée et est gérée par un thread qui surveille la croissance des plantes tout au long de leur développement.
Pour garantir une croissance optimale des plantes, nous avons dû reprendre l'état du potager à partir du moment où les graines ont été plantées. À chaque intervalle de temps, le potager change d'état en fonction de l'état actuel des plantes, jusqu'à ce qu'il atteigne son état final de développement, qui est l'état 12. À ce stade, le potager est prêt pour la récolte.
Grâce à cette méthode, nous pouvons surveiller la croissance des plantes et nous assurer qu'elles se développent correctement.
En somme, la classe Potager est un élément crucial de notre projet et nous permet de gérer efficacement le développement des plantes.

Monstres :

• Recherche de bâtiments :

Lorsque les monstres apparaissent, leur emplacement est déterminé aléatoirement dans une case de la carte. Une fois cette case trouvée, il est nécessaire de trouver le bâtiment le plus proche pour le monstre, afin qu'il puisse s'y diriger et causer des dégâts.

Pour ce faire, nous avons une méthode de recherche des bâtiments disponibles batimentCible() (à l'exception des potagers) et qui n’ont toujours pas été détruits, en calculant la distance euclidienne qui les sépare de la case d'apparition du monstre. La méthode choisit ensuite le bâtiment le plus proche en fonction de la distance la plus courte.
Pour identifier l'emplacement idéal du monstre, la première étape consiste à localiser le bâtiment le plus proche. Ensuite, nous explorons toutes les cases autour du bâtiment pour déterminer celles qui sont disponibles en utilisant la méthode EmplacementCible(). Une fois que la liste de points est établie, nous examinons toutes les positions possibles autour du bâtiment, à savoir en haut, en bas, à gauche et à droite, afin de trouver l'emplacement le plus proche du monstre.

• Déplacement :

Après avoir déterminé la position idéale du monstre, la prochaine étape consiste à le déplacer vers le bâtiment cible. Pour ce faire, nous avons utilisé la Classe A* afin de déterminer le chemin le plus court et optimisé, en prenant en compte les dimensions du monstre pour éviter les collisions avec les bâtiments environnants.

Nous avons également développé une classe appelée "DeplacementMonstre" qui implémente un thread et qui est responsable de gérer le déplacement du monstre sur la grille en suivant le chemin prédéfini par A*. Le monstre se déplace vers le bâtiment cible tant qu'il est en vie ou que le bâtiment n'a pas été détruit.
Le déplacement du monstre est effectué en deux étapes. Tout d'abord, la grille est mise à jour pour retirer le monstre de sa position actuelle et le placer sur la case suivante dans la direction déterminée. Ensuite, la position du monstre est mise à jour pour refléter sa nouvelle position sur la grille.

• Attaque :

Une fois que le monstre est à proximité du bâtiment, il doit commencer à l'attaquer pour causer des dégâts. Pour ce faire, nous avons mis en place une classe appelée "AttaqueMonstre" qui est responsable de gérer l'attaque du monstre sur le bâtiment cible. Cette classe prend le bâtiment cible en paramètre et permet au monstre de faire baisser ses points de vie en infligeant des dégâts.
Pour déterminer la puissance d'attaque du monstre, nous avons ajouté un attribut "dégâts" à la classe Monstre. Cet attribut représente le nombre de dégâts que le monstre peut infliger au bâtiment à chaque attaque. Ainsi, lorsque le thread de l'AttaqueMonstre est lancé, le monstre commence à attaquer le bâtiment cible en infligeant des dégâts proportionnels à la valeur de cet attribut. La valeur des attributs dégâts et hp augmentent selon le niveau de l'hôtel de ville qui est récupéré à l'aide d'un getter "getNiveau" dans le constructeur de Monstre.
Le processus d'attaque se poursuit tant que le monstre est en vie et que le bâtiment cible n'est pas détruit. Pendant ce temps, le bâtiment cible perd des points de vie à chaque intervalle de temps à l'aide de sa méthode "subitDegat", qui prend en paramètre le nombre de dégâts infligés par le monstre. Cette méthode calcule la quantité de dégâts infligés en fonction de la valeur de l'attribut "dégâts" du monstre, et met à jour les points de vie du bâtiment en conséquence.
Une fois que le bâtiment est détruit, le monstre commence à chercher un autre bâtiment à attaquer. Pour cela, nous utilisons la même méthode de recherche pour déterminer la position idéale du monstre.
Ce processus d'attaque et de déplacement se répète jusqu'à ce que tous les bâtiments soient détruits ou que le monstre soit tué par les défenses ou les villageois.

Inventaire :

L'inventaire est une classe qui représente les ressources disponibles pour le joueur dans le jeu. Les ressources sont stockées dans des listes qui contiennent des objets de différents types : les Potagers, les Batiments et les Defenses.

Les attributs carottes, baies et betteraves représentent la quantité de chaque type de légume disponible dans l'inventaire. Les attributs potionsCarottes, potionsBaies et potionsBetteraves représentent la quantité de potions de soin disponibles pour chaque type de légume.

La méthode "add" de chaque liste permet d'ajouter des objets correspondants à l'inventaire. La méthode "remove" permet de retirer des objets de l'inventaire après que le joueur les ai utilisé ou vendus.

Cabanes :

Les cabanes peuvent être construites à différents emplacements sur la grille.

Pour construire une cabane, le joueur doit sélectionner l'emplacement de la cabane sur la grille et cliquer sur le bouton de construction de cabane. La méthode addCabanes(Point p) est appelée avec la position sélectionnée en tant que paramètre.

La méthode addCabanes(Point p) de la classe Grille permet d'ajouter une cabane à la grille. Elle prend en paramètre la position de la cabane sur la grille (de type Point). La méthode commence par vérifier s'il y a déjà cinq cabanes sur la grille. Si ce n'est pas le cas, elle crée une nouvelle cabane à la position p. Elle ajoute la nouvelle cabane à la liste des cabanes et à la liste des bâtiments sélectionnés. Ensuite, elle parcourt les cases correspondant à la taille de la cabane et les marque comme occupées. Elle crée également deux nouveaux villageois à proximité de la cabane, un à gauche et un à droite. Enfin, elle met à jour la position de l'emplacement de la prochaine cabane.

Hôtel de ville :

Pour ajouter l'objet HotelVille dans le jeu, nous avons ajouté un nouvel attribut de type Batiment appelé "hotelVille" à la classe Grille. Cet attribut représente le bâtiment principal du jeu autour duquel les autres éléments sont placés.

Dans le constructeur de la classe Grille, nous avons initialisé l'objet HotelVille en spécifiant sa position initiale, son nom, sa largeur et sa hauteur. Pour cela, nous avons créé un nouvel objet Point qui représente la position initiale de l'hôtel de ville dans la grille de jeu. Nous avons ensuite créé l'objet Batiment en passant ces valeurs en paramètres. Nous avons ensuite défini la position de l'hôtel de ville en utilisant la méthode "setPosition" de l'objet Batiment.

Enfin, nous avons parcouru les cases associées à l'hôtel de ville à l'aide de deux boucles imbriquées pour définir chaque case comme occupée par l'hôtel de ville en utilisant la méthode "setBatiment" de l'objet Case.

Panneau de contrôle :

Pour toutes les actions, on utilise un actionEvent pour savoir sur quel boutons nous avons cliquer et qui affichera un message en conséquence sur le panneau d'affichage.

On utilise des KeyListener qui "mettent sur écoute" les flèches du clavier pour nous permettre de naviguer dans la carte et alors de sélectionner un villageois, un potager ou autre .

Ensuite pour valider les actions on appuie sur la touche entrée et l'action sera validé grâce à un getter "getKeyCode" qui vérifiera bien que l'on a appuyer sur entrée.

Nos différents messages visibles sur le panneau de contrôle sont affichés à l'aide de JLabel et sont mis à jour en fonction des différents évènements qui se produisent (appuis sur une touche du clavier par exemple).

Ressources :

L'argent : On affiche l'argent disponible à l'aide de Label et qui s'actualise grâce à notre méthode "update" dans notre classe "VueTresorerie" .

Shop :

Le shop est un élément à part entière du jeu qui nous permet notamment de nous fournir en graine afin de planter quelque chose dans le potager, d'acheter des bâtiments, des potions, des outils, ou encore de vendre nos plantes récoltées pour gagner de l'argent.

Le shop permet donc l'ajout de ressources à notre inventaire ou au contraire de vendre nos légumes afin d'obtenir des potions.

Le shop est visible à droite de l'écran par un carré noir, ressemblant à une console. Cette vue est faite à l'aide de JPanel qui permettent un affichage et de JLabel qui permettent d'envoyer et d'afficher des messages sur les JPanel.

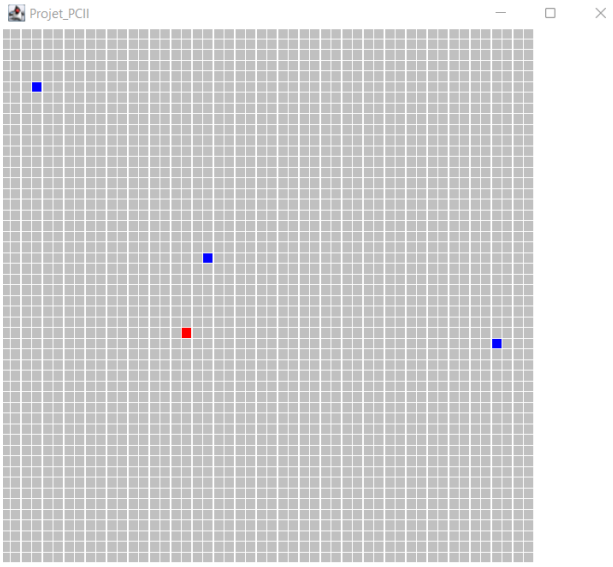
On se déplace dans le shop à l'aide des flèches du clavier et on valide chacune de nos actions à l'aide de la touche entrée, pour cela on a mis en place des KeyListener et des ActionEvent qui font que le shop soit intéractif et mis à jour.

Afin de visualiser tout cela, vous trouverez en pièces jointes le diagramme de classe fait à la main (il y a encore tout les attributs sur le diagramme car en voulant supprimer les attributs liées par composition ou aggrégation à d'autres classes, cela provoquait des bugs).

Vous trouverez le diagramme dans le rapport page 23, mais si vous voulez plus de lisibilité, le fichier .drawio est disponible en fichier joint.

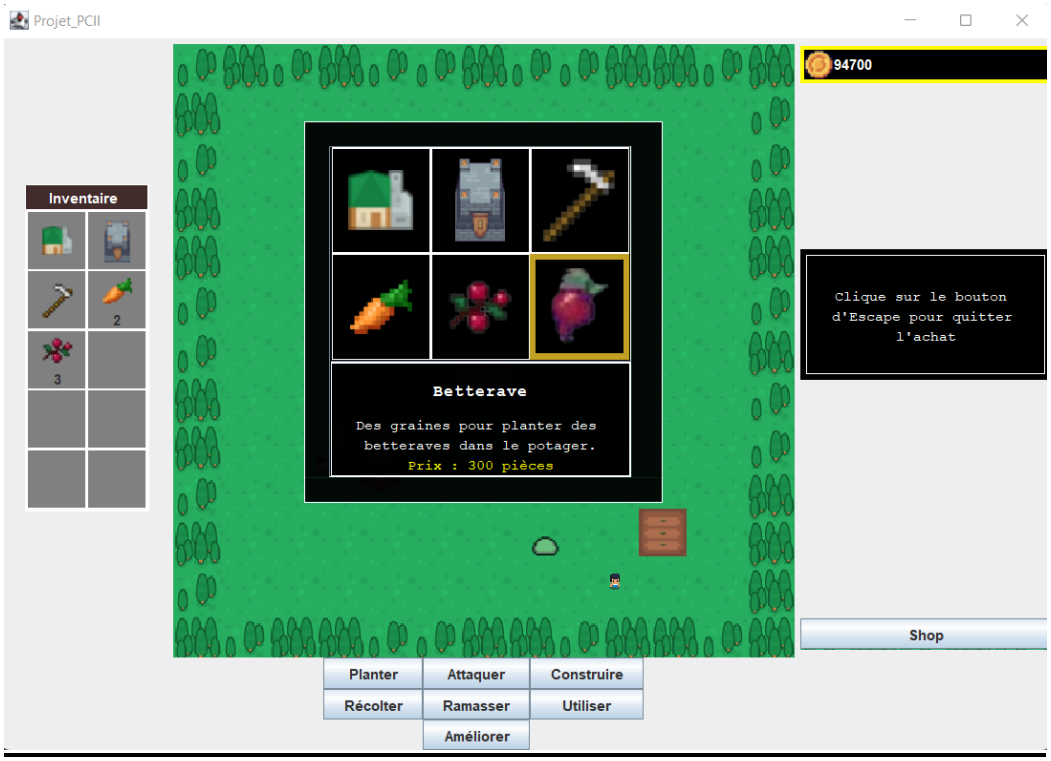
6 - Résultat

A la fin de la Semaine 1, on avait un résultat semblable à celui-ci :



A la fin de la Semaine 3 notre projet ressemblait déjà plus au rendu final :

Enfin voici le Résultat final du projet :



7 - Documentation utilisateur

- Prérequis : Java avec un IDE (ou Java tout seul si vous avez fait un export en .jar exécutable)
- Mode d'emploi (cas IDE) : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis « Run as Java Application ».
- Gameplay :
    - Achat et vente dans le shop :
        - Quitter le shop:
            - appuyez sur echap
        - Vente de légumes pour potion:
            - cliquez sur le bouton shop
            - descendre avec les flèche sur vente puis faire entrée
            - sélectionnez dans le shop la potion que vous souhaitez synthétiser avec les flèches
            - appuyez sur entrée pour valider la sélection
            - appuyez sur entrée pour valider l'achat/ pour l'annuler descendre avec la flèche sur 'non' et appuyez sur entrée
        - Achat d'un légume/bâtiment :
            - cliquez sur le bouton shop
            - sélectionnez dans le shop le bâtiment/légume que vous souhaitez construire avec les flèches
            - appuyez sur entrée pour valider la sélection
            - appuyez sur entrée pour valider l'achat/ pour l'annuler descendre avec la flèche sur 'non' et appuyez sur entrée
    - Mise en place et modification du village:
        - Construire un bâtiment :
            - cliquez sur le bouton construire
            - sélectionnez dans l'inventaire le bâtiment que vous souhaitez construire avec les flèches
            - appuyez sur entrée
            - positionnez le bâtiment à l'endroit souhaité à l'aide des flèches
            - appuyez sur entrée pour valider la construction/ pour l'annuler descendre avec la flèche sur 'non' et appuyez sur entrée
        - Améliorer :
            - cliquer sur le bouton améliorer
            - sélectionner le bâtiment à améliorer avec les flèches
            - appuyez sur entrée pour valider la sélection

        - appuyez sur entrée pour valider l'amélioration/ pour l'annuler descendre avec la flèche sur 'non' et appuyez sur entrée
- Utilisation des villageois :
    - Planter des légumes :
        - cliquez sur bouton planter
        - sélectionnez un légume dans l'inventaire avec les flèches et faire entrée
        - se positionner sur un potager avec les flèches
        - lorsque potager souhaité est entouré en jaune appuyez sur entrée
        - se positionner sur un villageois que l'on veut utiliser avec les flèches et appuyez sur entrée
    - Récolter des légumes :
        - cliquez sur bouton récolter
        - sélectionnez un potager avec les flèches et faire entrée
        - se positionner sur un villageois que l'on veut utiliser avec les flèches et appuyez sur entrée
    - Attaque :
        - cliquez sur le bouton attaquer
        - sélectionnez le monstre que vous souhaitez attaquer et faire entrée
        - sélectionnez avec les flèches le villageois que vous souhaitez utiliser pour l'attaque puis appuyez sur entrée
    - Ramasser :
        - cliquer sur le bouton ramasser
        - sélectionner la pièce avec les flèches puis appuyez sur entrée
        - sélectionner un villageois avec les flèches puis appuyez sur entrée
- Utilisation des potions :
        - cliquez sur utiliser
        - sélectionnez la potion dans l'inventaire
        - sélectionnez le bâtiment que vous voulez soigner et appuyez sur entrée

8 - Documentation développeur

Si un développeur souhaite améliorer le code, il devra tout d'abord se familiariser avec la structure du projet. Pour cela, la classe à regarder en premier est la classe FenetrePrincipal, qui contient les différents éléments de l'interface utilisateur tels que

l'affichage du jeu, le marché, la console, etc. Cette classe lui donnera une bonne idée de la répartition des éléments dans la fenêtre.
Pour modifier la taille de l'affichage du jeu, le développeur devra aller dans la classe Affichage et modifier les constantes FENETRE_HAUTEUR et FENETRE_LARGEUR. Cependant, il est important de noter que la taille choisie doit être un multiple de 20 pour que le jeu fonctionne correctement.
Pour manipuler les ressources du jeu, le développeur doit se rendre dans la classe Game pour accéder à l'argent et dans la classe Inventaire pour accéder aux autres ressources telles que les potions, les graines et les bâtiments.
Dans la classe "Grille", le développeur peut trouver les objets principaux du jeu tels que les villageois, les monstres, les batiments, etc. Cette classe est essentielle pour le bon fonctionnement du jeu car elle gère l'initialisation et la mise à jour de tous les éléments. En explorant cette classe, le développeur peut trouver des méthodes pour ajouter de nouveaux types d'entités, améliorer les mécanismes de collision, ou encore modifier la manière dont les entités interagissent entre elles. Pour améliorer la qualité du code, le développeur peut également réfléchir à la création de classes abstraites pour faciliter l'ajout de nouvelles fonctionnalités.

Dans le package Controler, deux classes jouent un rôle primordial dans l'expérience utilisateur.
Dans la classe "Control", le développeur peut trouver des méthodes pour gérer les interactions utilisateur avec le jeu. Cette classe joue un rôle essentiel dans l'expérience utilisateur car elle permet aux joueurs d'interagir avec les boutons et les commandes du jeu. En explorant cette classe, le développeur peut trouver des méthodes pour ajouter de nouvelles actions à chaque bouton, ou encore modifier la structure de l'interaction utilisateur avec le jeu. Pour faciliter la compréhension du code, le développeur peut également ajouter des commentaires pour expliquer le rôle de chaque méthode.

La deuxième classe importante est la classe JeuThread, qui maintient le jeu à jour. En effet, cette classe agit comme une classe update qui met à jour l'état du jeu à chaque intervalle de temps en fonction de l'état de chaque objet.
Enfin, pour accélérer le déplacement des entités, le développeur peut modifier les valeurs de Thread.sleep qui se trouvent dans les classes liées aux déplacements de ces dernières. Cela peut améliorer la fluidité du jeu.
En ce qui concerne la classe "Main", elle est mise à l'écart pour permettre à une personne externe de la trouver facilement. Ce choix de conception a été fait pour faciliter la maintenance du code, car cette classe est souvent considérée comme la porte d'entrée du projet. En effet, c'est dans cette classe que la méthode "main" est définie, et c'est donc elle qui est exécutée en premier lors du lancement du jeu. En modifiant cette classe, le développeur peut ajouter de nouvelles fonctionnalités telles que des options de configuration ou des écrans de démarrage personnalisés.

Enfin, pour améliorer les performances du jeu, le développeur peut modifier les valeurs de "Thread.sleep" qui se trouvent au niveau des classes liées aux déplacements des entités. En réduisant le temps de sommeil de ces threads, le déplacement des entités sera plus rapide, ce qui améliorera l'expérience utilisateur. Cependant, il est important de noter que cette modification peut également augmenter la charge de traitement du CPU, ce qui peut entraîner des ralentissements ou des plantages du jeu. Il est donc conseillé de tester soigneusement ces modifications avant de les mettre en production.

9 - Conclusion et perspectives

- Qu'avez-vous réalisé ?

Nous avons réalisé un jeu en 2D, dont la mission principale est d'atteindre le niveau 3 de l'hôtel de ville. Pour cela il nous faut de l'argent que l'on peut obtenir en tuant les monstres qui attaquent notre village et en vendant les fruits et légumes que nous récoltons dans les potagers. Une fois l'argent obtenu, on peut améliorer nos bâtiments. Le but est donc de développer notre village.

- Quelles étaient les difficultés et, surtout, comment les avez-vous résolues ?

La mise en place de l'algorithme A* a nécessité un investissement important en termes de temps et de ressources. Nous avons dû comprendre le fonctionnement de l'algorithme en profondeur pour pouvoir l'adapter à notre environnement de jeu. Cette tâche s'est avérée plus difficile que prévue en raison de la complexité du processus de déplacement des entités sur plusieurs cases. En effet, chaque objet est placé sur plusieurs cases, ce qui implique que pour chaque déplacement, il faut vérifier l'état de toutes les cases suivantes afin de déterminer si elles sont libres ou occupées. Pour intégrer la recherche de chemin A*, nous avons ajouté des attributs spécifiques à l'algorithme, tels que "parent" qui indique la case parent … Une fois que nous avons intégré l'algorithme, nous avons constaté que des bugs récurrents affectaient le jeu sans que nous sachions vraiment d'où ils venaient. Nous avons dû consacrer plusieurs jours à la recherche de l'origine du problème. Nous avons finalement réalisé que l'erreur venait du fait que nous n'avions pas rafraîchi les valeurs des attributs cases à la fin de l'appel de l'algorithme. Ce manque de rafraîchissement a eu pour conséquence de bloquer le jeu car le chemin n'était plus trouvé à partir d'un certain point et la boucle tournait indéfiniment.

- Qu'avez-vous appris ?

A s'organiser en équipe et bien se répartir les tâches.

Se tenir informés régulièrement les uns les autres des avancements ou problèmes rencontrés. Le faire régulièrement est très important car cela permet de rester sur la même longueur d'onde et de ne pas se disperser sur des idées différentes qui à la fin ne concorde plus ensemble.

- Que voyez-vous comme évolution future de ce travail ?

La suite logique de ce projet est de continuer de l'améliorer en proposant plus d'actions, plus de contenus, un jeu plus complet. Par exemple on aurait pu rajouter différents types d'armes pour se défendre contre les monstres, d'autres types de bâtiments permettant de nouvelles fonctions comme une mine d'or qui permet de récolter des pièces en fonction du temps. Avoir la possibilité d'agrandir sont villages en coupant des parcelles de forêts…