Rapport Projet Long

27 mai 2024

Table des matières

1 Introduction

Un jeu 4X est un genre de jeu vidéo de stratégie dans lequel le joueur contrôle un empire et dont le gameplay est fondé sur les quatre principes suivants : celui de l'exploration, celui d'expansion, de l'exploitation et de l'extermination (" e**X**plore, e**X**pand, e**X**ploit and e**X**terminate " en anglais). Les jeux les plus connus de ce genre sont Sid Meier's Civilization (en tour par tour) et Age of Empires (en temps réel).

L'objectif de ce projet était donc de développer un jeu du genre 4X en tour par tour en multijoueur local.

Pour cela nous avons utilisé le language de programmation Java, avec libGDX comme framework de développement.

2 Fonctionnalités

Ce projet étant un jeu, il nécessite un grand nombre de fonctionnalité pour avoir un produit fonctionnel, c'est a dire un jeu auquel on peut jouer et gagner, voici une liste des principales fonctionnalités, si elles ont été complétées et l'itération dans lesquelles elles ont été faites.

2.1 Géneration de la grille de jeu

— État : Terminée
— Début et fin d'implantation : Itération 1.

C'est la toute première fonctionnalités qui a été implémentée. En effet tout le jeu se base fondamentalement sur cette grille. Pour compléter cette fonctionnalités il a fallut créer une classe "Tile" qui ont ensuite été composées dans une nouvelle classe "Grid", une Grille étant générée a l'aide de "Perlin Noisemaps" ce qui permet de génerer une grille aléatoirement tout en conservant une certaine "cohérence géographique" (on ne passe pas de la mer a la montagne puis a nouveau la mer en 2 Tiles).

2.2 Graphismes

— État : Terminée
— Début et fin d'implantation : Itération 1.

Les Graphismes sont surement la classe la plus importante pour le projet final, en effet c'est tout ce que l'utilisateur verra fondamentalement. Ils ont été implémentés très tôt dans le processus pour pouvoir facilement tester les implémentations des fonctionnalités suivantes, a l'aide d'un support graphique. cette fonctionnalité est implémentée dans la classe IsometricRenderer.

FIGURE 1 – Image d'une ville et du menu s'affichant a la droite de l'écran

2.3 IHM + gestion des inputs

— État : Terminée
— Début d'implantation : Itération 1.
— Fin d'implantation : Itération 3.

la gestion des inputs est une fonctionnalité qui bien qu'elle été implémentée dès le début de la première Itération avec des inputs basique (information sur une tuile, zoom, mouvement de la caméra, bouton quitter), elle a été constamment mis a jour avec l'ajout de nouvelle fonctionnalités. En effet a chaque fois qu'une nouvelle fonctionnalité été implémentée il fallait ajouter un bouton on un input qui permettait d'interagir avec, ce qui a causé a la classe GameUI de grandir beaucoup plus que prévu (Tout en restant a peu près compréhensible). la classe GameInputProcessor est la deuxième classe qui implémente cette fonctionnalité

2.4 Gestion du tour par tour

— État : Terminée
— Début d'implantation : Itération 1.
— Fin d'implantation : Itération 2

La gestion du tour par tour est une fonctionnalité qui englobe a la fois :
— la gestion de ce qu'un joueur peut faire pendant un tour (Implémenté a l'aide de variable boolénne notamment dans les classes Troop et City qui indique si une unité a déja bougée a ce tour et ne peut donc plus bouger)
— la mise en place du début du tour d'un joueur (Implémentée avec la méthode setUpTurn dans la classe Player)
— le fait d'empecher un joueur de jouer hors de son tour (implémenté avec les mêmes variables que le premier point)

2.5 Mise en place d'une partie

— État : Terminé
— Début et fin d'implantation : Itération 1.

La mise en place d'une partie, c'est tout les menus d'options avant le jeu, c'est a dire la classe MenuScreen (qui permet de lancer une partie et de quitter le jeu) ainsi que la classe OptionScreen (qui permet de choisir la taille de la carte). C'est aussi le lancement de la partie c'est a dire la création de la grille et l'instanciation de toute les classes importantes dans la classe GameScreen. Enfin la mise en place de la partie consiste en l'appel de la méthode setUpGame de la classe PlayerManager qui donne a chaque joueur son unité de départ et la place sur la carte.

2.6 Gestion des Joueurs

— État : Terminé
— Début d'implantation : Itération 1.
— Fin d'implantation : Itération 3.

Chaque instance de la classe Player représente un joueur qui est réprésenté par 3 "choses"
— des variables entières "score", "science" et "gold" (ses ressources globales) et "number" (qui est un id pour le joueur)
— un tableau de villes "cities" (représentant les villes possédées par le joueur)
— un tableau de troupes "troops" (représentant les unités que le joueur contrôle)

Tout ces joueurs sont alors rassemblés dans une instance de la classe PlayerManager ce qui facilite la gestion de tout les joueurs. La classe playermanager permettant un accès facile a chaque joueur car l'id du joueur correspond a sa position dans le tableau.

2.7 Gestion des Villes

— État : Terminé
— Début et fin d'implantation : Itération 1.

La ville est un élement très important puisque c'est avec ces villes que le joueur va produire toutes ses ressources et troupes. Les villes peuvent être fondées par un Settler (Colon) qui disparait au moment de fonder la ville. Une ville est composée d'un tableau "tiles" qui représente les cases possédées par la ville, le premier élement de ce tableau étant le centre ville qui est la ou un joueur peut cliquer pour lancer la formation de troupes ainsi que la construction de batiments, ces batiments augmentant la quantité de ressources locales. la méthode updateParameters permet de calculer toute les ressources locales a la ville :

— La nourriture : influe sur la population de la ville, plus il y a de nourriture plus il y a de populations

— La production : qui influe sur la vitesse de production des batiments et la formation d'unités.

les tuiles non centre villes servent aussi puisqu'elles génerent toutes de la production et peuvent être aménagés pour augmenter leur rendement.

2.8 Formation des unités

— État : En cours, fonctionnel
— Début d'implantation : Itération 1.
— Derniers ajouts : Itération 3.

Différentes unités ont été créées avec différents rôles et spécialités propre à chacune de ces unités :

— Warrior : Combattant de base
— Archer : Combattant a distance
— Settler : Unité permettant de former des villes
— Builder : Batisseur permettant d'aménager des cases

Il est également possible de déplacer ces unités et (pour le Warrior et l'archer) de se battre avec les unités ennemies. Ces unités peuvent être achetés ou "formées" ce qui aura un coût en production plutôt qu'en or.

2.9 Construction des batiments

— État : Terminée
— Début d'implémentation : Itération 1.
— Fin d'implémentation : Itération 3.

La construction des batiments peut soit se payer avec de l'or soit avec de la production sur plusieurs tours. Ces batiments permettent d'augmenter la production des ressources de la ville.

2.10 Aménagement des ressources

— État : Terminée
— Début d'implantation : Itération 1.
— Fin d'implantation : Itération 2.

Les aménagements sont fait par les batisseurs. Les tuiles d'une ville peuvent etre aménagés pour augmenter leur production passive de ressources de la ville. certaines tuiles possède déja des ressources a exploiter comme de la pierre ou du bétail ce qui fait qu'un aménagement dessus augmentera encore plus les ressources de la ville.

2.11 Victoire d'un joueur

— État : Terminée
— Début d'implantation : Itération 2.
— Fin d'implantation : Itération 3

Il existe 2 types de victoires :

1. Au score. S'il reste plus d'un joueur lors du dernier tour c'est celui qui possède le score le plus élevé qui gagne la partie. Le score est une valeur propre a un joueur qui augmente au cours de la partie en fonction de ses actions. Le score augmente pour chaque ville fondée, chaque troupe produite, chaque ville capturée...
2. Domination. Lorsqu'un joueur a conquit toutes les villes des joueurs ennemis il active la victoire par domination.

2.12 Mecanique de science

— État : Terminée
— Début et fin d'implantation : Itération 3

La science est une ressource globale de chaque joueur, chaque ville en produit une certaine quantité, et permet d'améliorer son empire. chaque technologie a un "seuil", quand la quantité de science dépasse ce seuil il débloque une nouvelle technologie.

2.13 Mécanique de prise de ville

— État : Terminée
— Début et fin d'implantation : Itération 3

Quand une unité de combat (Warrior ou Archer), pénètre dans une ville, elle "capture" cette ville qui change alors de controlleur. Cela permet de progresser dans la victoire par domination.

2.14 Sauvegarde

— Etat : en cours
— Début d'implantation : Itération 3

la sauvegarde permettrait au joueur d'interrompre sa partie, de fermer le jeu et de reprendre plus tard, l'implantation est toujours en cours car l'architecture de notre Jeu rend l'enregistrement de données assez complexe.

3 Architecture et réalisation du projet

3.1 Architecture globale

Pour réaliser le projet différentes classes ont été conçues et celles-ci peuvent être séparés dans les grandes catégories suivantes :
— Les écrans
— Les troupes
— Grille du jeu
— Les joueurs

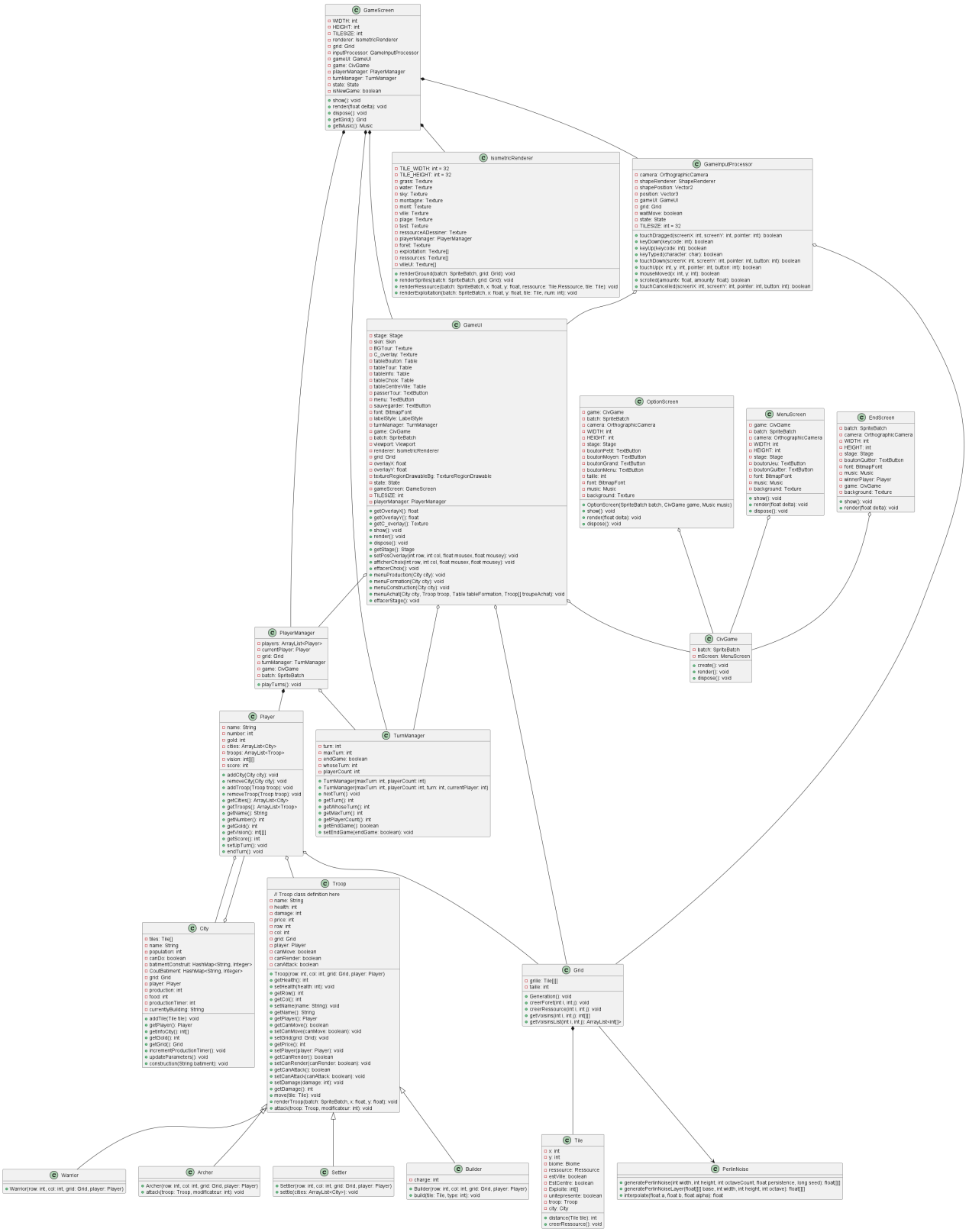

FIGURE 2 – Diagramme UML complet

voici une courte explication du diagramme UML :

en bas a droite on a Grid, Tile et PerlinNoise qui sont les 3 classes liées a la grille (une grille est composée de tuile et est générée a l'aide d'une noisemap de perlin). Chaque tuile possède toute les informations nécessaire sur ce qu'il se passe dessus (ville, troupe, ressources, exploité ou non ...)

En haut on a le "GameScreen" qui est la classe qui instancie toute les autres classes et qui est le "centre" (les autres écrans sont anecdotiques et permettent simplement de choisir quelques options ou d'afficher le gagnant)

tout ce qui est affiché a l'écran est géré par l'IsometricRenderer qui gère tout l'affichage (sauf les boutons et menus)

La classe GameUI gère toute l'IHM elle est donc plutôt tentaculaire comme elle gère tout les menus, tout les boutons qu'on a vu (qui nécessite l'accès au informations de tout les joueurs, de la grille...)

les autres input sont géré par le GameInputProcessor qui fait la gestion de tout les clics hors des menus qui permettent de sélectionner une tuile pour informations ou attaquer ou bouger, plus c'est lui qui gère le zoom et le mouvement de la caméra.

Le PlayerManager est la liste de tout les joueurs ce qui facilite la gestion du tour par tour et le lancement de la partie (elle garde aussi en mémoire le joueur courant)

Chaque "Player" du PlayerManager possède un attribut or, score, science et enfin 2 listes "cities" et "troops" qui sont la liste de toutes les villes et toutes les troupes possédée par le joueur.

la classe troupe étant une classe dont hérite les 4 vraies troupes, guerriers, bâtisseurs, colons et archer.

la classe City gère tout ce qui a un rapport avec la ville que ce soit le calcul des ressources locales (nourritures, production) ou bien la formation d'unités, construction de bâtiments.

3.2 Choix de conception

Nous avons fait le choix d'utiliser le framework libGDX qui nous a facilité la création d'une interface graphique. En effet la perspective isométrique a fait que les coordonées de l'écran ne correspondait pas au coordonnées de la grille de jeu. libGDX a des outils qui nous on permis de facilement gérer tout cela. Le framework libGDX est en plus assez simple a utiliser et a prendre en main.

3.3 Difficultés rencontrées et solutions

Une des difficultés rencontrées se trouve dans le déplacement et la méthode d'attaque des unités. En effet pour initier un déplacement il fallait d'abord cliquer sur le bouton "Move" qui apparait lorsqu'on clique sur une tuile sur laquelle se trouve une unité. ce bouton est instancié dans la classe GameUI. Or pour cliquer sur la case ou on veut déplacer notre unité il faut cliquer sur la grille. Mais cela est géré par la classe GameInputProcessor. Le problème est alors qu'au moment du clic, le GameInputProcessor doit décider si le déplacement est valide (que l'unité se déplace sur une case valide dans sa portée de déplacement), il a alors besoin de connaitre la position initiale de l'unité. Pour résoudre ce problème nous avons créés une classe State, qui au moment de cliquer sur le bouton "Move" Enregistre la position de l'unité, cette instance de State est aussi accessible par l'inputProcessor qui peut alors vérifier que le déplacement est valide. l'instance de State possède aussi une variable booléene qui indique si la position stocké a l'intérieur correspond a une unité qui essaye de se déplacer actuellement.

4 Organisation de l'équipe

Séparations de tâches :

Louis-Clément	IHM + gestion des inputs (XXL), Graphismes (XXL), Implémentation et géneration aléatoire de la grille (XL),Gestion du tour par tour (XL), Mise en place d'une partie (MenuScreen et OptionScreen) (XL), Gestion des villes (L), Gestions des joueurs (L), aménagements des ressources (XL).
Raid	Mécaniques de construction de batiments et formations des troupes (XL)
Thomas	Gestion du score et Victoire au score (XL)
Ossama-Moussa	Mécanique de science (L), évolution de la civilisation en fonction de la science (M)
David	Mécanique de combat (L), Mécanisme de prise de villes (S)
Antoine	

5 Annexe

Librairies utilisées :
Framework libGDX
Géneration de perlin noisemaps