Étudiant : **O. ROGELET, L. ROUILLE, V., J.DEBOUDT, N.MOGIS**
Semestre : **A25**

LO21
PROGRAMMATION ET CONCEPTION
ORIENTÉES OBJET

Rendu suivi 2

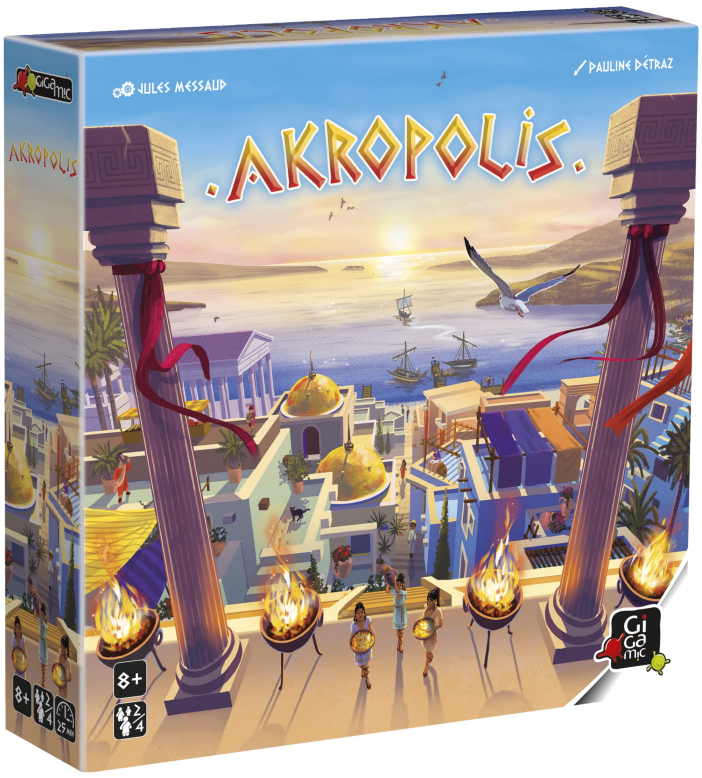

Table des matières

1. Introduction

Cette deuxième phase du projet a marqué le passage de la conception théorique à la mise en oeuvre concrète de notre adaptation du jeu Akropolis. Après avoir établi une analyse conceptuelle solide et défini les principaux éléments du jeu lors du premier compte rendu, nous nous sommes cette fois-ci concentrés sur la construction d'une architecture orientée objet cohérente et sur les premières étapes de l'implémentation.

L'objectif principal de cette phase était de commencer à traduire les concepts identifiés en modules fonctionnels, en posant les bases du code qui constituera le coeur du jeu. Ce travail a nécessité une réflexion approfondie sur les relations entre les classes, la gestion des données et la structure globale du programme.

2. État d'Avancement depuis le livrable 1

2.1 Tâches réalisées

T3 Représentation des tuiles

La tâche de représentation des tuiles a été entièrement réalisée conformément aux objectifs initiaux, en seulement 20 heures, soit 5 heures de moins que la durée estimée.

Un léger décalage dans le planning a toutefois eu lieu : l'implémentation, initialement prévue pour les semaines 3 et 4, a finalement été effectuée durant les semaines 5 et 6. Cette tâche a bien été réalisée par Louane et Valentin comme indiqué dans le premier rendu.

La phase de réflexion a constitué notre premier point d'appui et a duré environ une heure par personne. Nous avons ensuite chacun développé une classe, ce qui nous a pris environ deux heures chacun. Par la suite, une part importante du travail (environ quatre heures par personne) a été consacrée à l'adaptation et à la modification de nos classes initiales afin d'assurer leur compatibilité et leur bon fonctionnement ensemble.

Enfin, la prise en main des outils utilisés pour le projet (notamment Visual Studio et Git) s'est révélée plus complexe que prévu et a entraîné un certain retard dans notre avancement.

- **Responsables :** Louane & Valentin
- **Temps passé :** 20h

2.2 Tâches en cours et prochaines étapes

T4 Gestion des règles de placement

En terme de volume horaire Oscar et Jeanne ont passé 2h chacun à penser à la réalisation du code, ensuite 2h de mise en commun sur les idées de fonctionnement. Puis Oscar a codé placement.h avec les conseils de Jeanne et Jeanne a codé le fichier placement.cpp, ce qui fait un total de 15h de code. Donc finalement un total de 19h.

À cela il faut ajouter 2h pour le rapport et la mise en commun du code avec les autres.

- **Responsables :** Oscar & Jeanne
- **Progression :** 70%
- **Temps passé :** 21h
- **Temps restant estimé :** 8h

T5 Gestion du décompte des points

Les fonctionnalités principales ont été implémentées avec succès, notamment le calcul des scores selon les types de quartiers, la gestion des pierres et le total des points, incluant les étoiles multiplicatrices et les tuiles superposées.

Certaines fonctionnalités restent à développer, comme la sauvegarde et la reprise de partie, la gestion de l'abandon et le système d'achat de tuiles. Le développement du système de score a pris plus de temps que prévu, ce qui a entraîné un léger retard sur ces fonctionnalités restantes. La gestion de la mémoire de la partie sera la tâche la plus complexe à venir, tandis que l'ajout du système d'achat de tuiles devrait être relativement simple.

- **Responsables :** Noémie
- **Progression :** 75%
- **Temps passé :** 20h
- **Temps restant estimé :** 10h

T6 Gestion du mode multijoueur

La mise en place du mode multijoueur a permis d'assurer la gestion des joueurs, le suivi des points individuels et la détermination du gagnant en fin de partie. L'alternance des tours entre les joueurs a été amorcée mais n'a pas encore été finalisée.

- **Responsables :** Noémie
- **Progression :** 90%
- **Temps passé :** 10h
- **Temps restant estimé :** 5h

T8 Gestion de l'interface console

La gestion de l'interface console représente une étape importante de notre projet puisqu'elle est indispensable pour le test de nombreux ajouts. C'est pourquoi nous nous sommes dans un premier temps concentrés sur l'affichage du plateau et des tuiles, réservant pour la suite les menus et autres fioritures.

Cet affichage a connu de nombreux changements par rapport à ce qui était initialement prévu. En effet, nous désirions dans un premier temps que chaque tuile soit affichée indépendamment dans la console mais cela s'avérait trop compliqué à gérer. C'est pourquoi nous avons finalement opté pour l'affichage d'un quadrillage d'hexagones remplis au fur et à mesure de la partie par le joueur.

Cependant, cette représentation reste pour l'instant un prototype, quelques fonctions de test ont été implémentées et restent à bonifier.

- **Responsables :** Tous
- **Progression :** 10%
- **Temps passé :** 5h
- **Temps restant estimé :** 30h

T11 Création et gestion du repo github

Afin de travailler ensemble efficacement, nous avons mis en place un repository github, ce qui a pris un temps étonnamment long (notamment car nous n'avions presque jamais utilisé cet outil). Ainsi, nous avons créer une nouvelle tâche de gestion du repo.

Ce repo nous aide énormément à versionner notre code et à travailler en équipe mais la découverte du fonctionnement a été très longue (mais enrichissante). Oscar a commencé par créer un repo classique à partir d'un dossier de son PC en y créant ensuite une solution Visual Studio mais le repo contenait donc tous nos fichiers propres à l'utilisateur pour VS, ce qui empêchait le code de s'ouvrir dès qu'une nouvelle personne faisait un commit. Oscar a donc dû presque recommencer pour avoir un .gitignore permettant d'éviter ce problème.

Il a ensuita fallu apprendre à créer et utiliser des branches, même si ce n'est pas toujours clair pour nous. Nous avons souvent dû récupérer des fichiers à la main sur d'autres branches pour faire fonctionner la nôtre. Enfin, le merge des branches a été géré en adaptant les fichiers vcxproj et en corrigeant les conflits entre les branches.

Cette tâche est presque finie car Oscar a compris et expliqué à tout le groupe la majorité des fonctionnalités de base de cet outil.

- **Responsables :** Oscar
- **Progression :** 80%
- **Temps passé :** 8h
- **Temps restant estimé :** 2h

2.3 Affectation des tâches

Au vu de la complexité de la tâche et de sa nécessité pour tester nos codes, nous avons décidé de changer l'affectation de la tâche T8 (avant réservée à Oscar, Noémie et Jeanne) afin que tous les membres puissent participer.

Ainsi, chacun pourra modifier l'affichage afin de le rendre conforme à ce dont il a besoin.

Cette affectation est la seule qui a été changée.

3. Conception orientée objet préliminaire

3.1 Description des classes principales

T3 Représentation des tuiles

La représentation des tuiles respecte ce qui était convenu lors du premier rapport : la création d'une classe Tuile qui compose une classe Hexagone. Cette dernière est donc friend class de Tuile.

La classe Hexagone est représentée par un niveau, un nombre d'étoiles, un type (habitation, marché, caserne, temple, ...) et un booléen (recouvert ou non).

Chaque tuile est représentée par un ID, une orientation, un prix ainsi que trois hexagones. Ces deux classes comportent évidemment les méthodes classiques telles que les constructeurs et destructeurs privés car gérés par une classe Partie ainsi que des getteurs.

En ce qui concerne les méthodes supplémentaires : hexagone contient une méthode recouvrir qui permet d'indiquer que l'hexagone a été recouvert et affiche qui permet d'afficher le type de l'hexagone sous forme de lettre (H pour Habitation, M pour Marché, ...). Cette dernière sera utilisée dans les fonctions d'affichage console du plateau.

T4 Gestion des règles de placement

Le placement des tuiles est géré par la classe Plateau qui utilise une structure std ::unordered_map<Coord, Hexagone&, CoordHash> pour représenter l'espace 3D du jeu. Après hésitation nous avons décidé d'opter pour cette représentation car contrairement à un tableau dynamique 3D qui nécessite d'allouer de la mémoire pour toutes les positions (même vides), la map ne stocke que les hexagones placés par l'utilisateur sur le plateau. L'accès en temps constant via la fonction de hachage CoordHash permet des vérifications rapides de disponibilité avec la méthode estLibre().

La fonction placer() permet de poser une tuile sur le plateau avec la distinction de niveau. Pour le niveau de base (z == 0), la tuile doit obligatoirement toucher au moins un bord d'une tuile déjà présente sur le plateau. Cette vérification est effectuée par la méthode privée voisinsTuile() qui retourne un std ::vector<Coord> contenant les coordonnées des hexagones adjacents

à la dite tuile. Le choix du std ::vector pour stocker temporairement ces voisins est justifié par sa gestion automatique de la mémoire.

Pour les niveaux supérieurs (z>0), la fonction vérifie deux contraintes : une tuile posée sur un niveau supérieur doit être construite sur 3 hexagones (elle ne peut pas se trouver au-dessus d'un espace vide) et être à cheval sur au moins 2 tuiles différentes. Pour cela nous avons utilisé les fonctions getTuile() et estLibre().

Par ailleurs, pour le choix des coordonnées nous avons opté pour le système suivant :

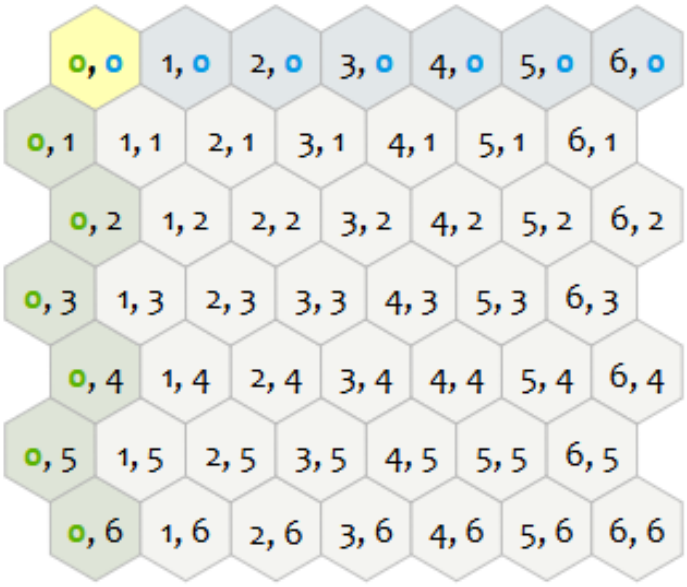

FIGURE 3.1 – Système de coordonnées

La disposition horizontale « even-r » décale les lignes paires vers la droite.

T5 Gestion du décompte des points

Cette classe a pour rôle de gérer le calcul et le suivi des points associés à chaque joueur au cours de la partie. Elle est directement liée à un objet Joueur et à une Cité, ce qui lui permet d'évaluer le score en fonction de l'état actuel du plateau.

La classe contient une structure interne de type std ::map<TypeQuartier, int> permettant d'associer à chaque type de quartier le nombre de points correspondants. Un attribut total regroupe la somme de ces points pour obtenir le score global du joueur.

Les méthodes principales comprennent notamment calculerScore() et calculerScoreType(), qui déterminent respectivement le score total et celui d'un quartier donné. Les différents accesseurs ont aussi été définis pour accéder aux attributs de la classe.

T6 Gestion du mode multijoueur

La gestion des joueurs sera faite avec la classe Joueur, cette classe a pour attribut le nom, le nombre de points et le nombre de pierres des joueurs. On met à jour les points et les pierres des joueurs et pour chaque recouvrement effectué on ajoute le nombre de pierres correspondant au nombre de tuiles recouvertes.

On aura une classe Partie permettant de gérer le nombre de joueurs et chaque joueur aura également sa propre pile. Concernant l'alternance des tours, la fonction jouerTours servira à donner la main à chacun des joueurs. Cette fonction n'est pour l'instant qu'une ébauche, mais elle attribuera les points et donnera les scores au fur et à mesure de la partie et mettra fin à la partie quand les piles des joueurs seront vides. Enfin, il y aura une comparaison finale des scores pour déterminer le gagnant. Cette méthode à été commencée mais pas finalisée car le débogage entre les fichiers a pris beaucoup de temps et l'alternance des affichages de plateau n'a pas encore été possible.

La classe Pile permettra d'acheter des tuiles avec les pierres, elle sera composée de tuiles générées aléatoirement et sa taille diminuera au fur et à mesure de la partie. Une fois qu'une pile est vide, cela met fin à la partie. La pile n'est pas finalisée et ne contient pour l'instant que la méthode qui permet de retirer une tuile. Il manque l'ajout de tuiles aléatoire dans celle-ci.

Cette tache à pris moins de temps que prévu, ce qui laisse du temps pour finaliser la méthode jouerTours et la pile et finir les objectifs de la tache 5 également.

3.2 Nouvelle analyse conceptuelle

UML

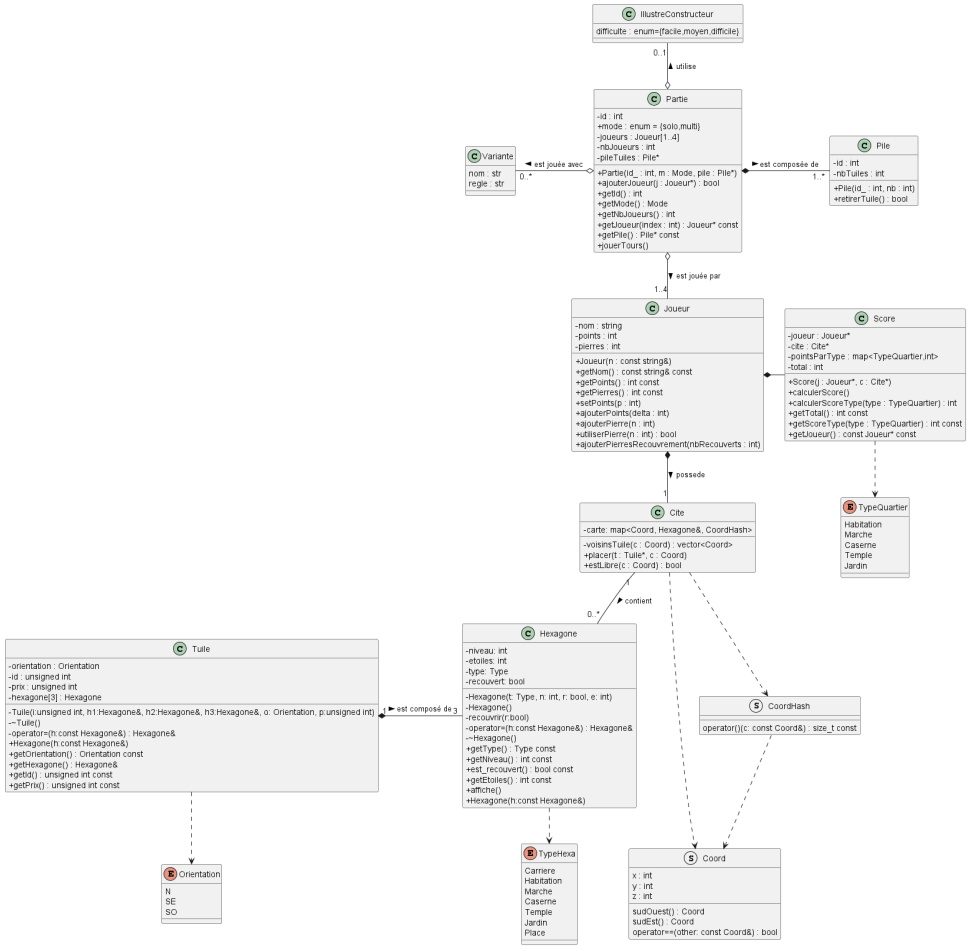

FIGURE 3.2 – Modèle conceptuel du jeu

Justifications UML

Depuis le dernier livrable, nous avons mis à jour l'uml et notamment les attributs et méthodes de certaines classes sur lesquelles nous avons travaillé.
Pour commencer, nous avons transformé l'héritage entre Hexagone, Quartier, Place et Carrière en choisissant un héritage par référence, c'est-à-dire que la classe mère Hexagone garde les attributs des classes filles. De plus, nous avons rajouté la composition entre Tuile et Hexagone et ainsi rajouté l'attribut hexagone[3] qui est un tableau de 3 hexagones composants la tuile.

Pour la classe Cite, la carte contient de multiples hexagones, ce qui constitue une composition : la cité possède les hexagones sur la carte, mais les hexagones peuvent théoriquement être manipulés par d'autres structures si nécessaire. La cité dépend également de Coord et CoordHash : ces structures sont utilisées pour localiser et indexer les hexagones dans la map. Il s'agit de dépendances techniques plutôt que de relations de contenance.

La relation entre Tuile et Orientation est une dépendance : chaque tuile possède un attribut de type Orientation qui définit son orientation sur la carte. De même, Hexagone dépend de l'énumération TypeHexa qui permet de connaître le type de terrain, et Score dépend de TypeQuartier pour calculer les points par type. Ces relations sont des dépendances simples car il n'existe pas de contenance réelle.

Concernant la classe Partie, la pile de tuiles est une composition forte : la vie de la pile est liée à celle de la partie. Les joueurs participant à la partie sont représentés par une agrégation faible, car les joueurs existent pour l'instant indépendamment de la partie. Les variantes de règles utilisées par la partie sont également représentées par une agrégation faible : elles modifient le comportement du jeu mais existent en dehors de la partie. Enfin, la relation avec un éventuel IllustreConstructeur est aussi une agrégation faible, car ce joueur particulier peut exister indépendamment de la partie et n'est utilisé que lorsqu'il est intégré à celle-ci.

4. Bilan de cohésion et d'implication

La cohésion d'équipe s'est révélée être un véritable atout au cours de cette phase. Chacun a su faire preuve d'écoute et de disponibilité envers les autres, favorisant une entraide constante, notamment lors de la mise en oeuvre des concepts les plus complexes. Les échanges réguliers ont permis d'assurer une progression harmonieuse, malgré les difficultés techniques rencontrées.

Cependant, la période des examens médians a considérablement réduit notre disponibilité commune, limitant le temps consacré aux réunions de groupe. Ce contretemps a légèrement ralenti l'intégration de nos travaux respectifs. À ce jour, les différents modules fonctionnent encore de manière relativement indépendante, et la prochaine étape consistera à fusionner nos branches sur Git afin d'assurer la compatibilité de l'ensemble des fonctionnalités développées.

Malgré ces contraintes, la motivation de l'équipe demeure intacte. Chacun reste pleinement investi dans le projet, avec la volonté commune de voir le jeu prendre forme et de parvenir à une version jouable complète. La communication au sein du groupe reste fluide, efficace et bienveillante, ce qui constitue un facteur essentiel de réussite pour la suite du développement.

5. Conclusion

Cette deuxième phase du projet nous a permis de franchir une étape essentielle dans la concrétisation de notre adaptation numérique du jeu Akropolis. Après la modélisation conceptuelle initiale, nous disposons désormais d'une base technique solide sur laquelle nous pourrons poursuivre le développement.

Les premières implémentations ont mis en lumière les points d'articulation entre nos différents modules, ouvrant la voie à un travail d'intégration plus poussé dans les prochaines semaines.

Nous abordons la suite du projet avec confiance, forts de la cohésion et de la rigueur acquises jusqu'ici. Les objectifs techniques et organisationnels sont désormais clairement identifiés, et tout porte à croire que la phase suivante permettra de consolider notre travail et de rapprocher encore un peu plus le jeu de sa version finale.