# Introduction

Java est un language de programmation orienté objet qui permet de produire des applications sécurisées et qui offre une portabilité avancée du code grâce à une machine virtuelle.

Lors de ce projet de programmation, je développerais le jeu "MoonQuest" grâce à cet outil. Cela me permettra de découvrir de nombreuses facettes du language et me forcera a manipuler un projet complexe avec rigueur.

# Analyse du Sujet

En lisant le sujet, nous allons essayer de déterminer quels seront les principaux challenges à décortiquer lors de l’implémentation de notre jeu.

### Les Pieces

Le jeu MoonQuest est présenté sous la forme d'un échiquier sur lequel des pièces sont placées. Ces dernières sont des candidates parfaites pour être codées par des classes en Java. Ainsi, nous essaierons de réduire au maximum les répétitions de code en créant un maximum de sous classes pour généraliser les objets manipulés.

### Le Plateau et les Cases

Le plateau de jeu sera formé de cases qui contiendrons des pièces ou non. Nous pourrons aussi créer un clase Case qui nous permettra de généraliser la production de ces dernières. Un des enjeux est de faire apparaître ces cases sous forme de plateau à l'écran. Ce plateau, devra s’actualiser à chaque tour pour permettre au joueur de consulter l’avancement de la partie.

### Les Règles de Déplacement

De nombreuses règles sont présentes dans le jeu, il faudra les implémenter sous forme de méthodes() qui seront contenues dans les sous classes. L’utilisation de super() nous permettra de moduler la réponse des méthodes en fonction de la sous classe utilisée.

### La Création des Modes de jeu et des Joueurs

Il sera aussi nécessaire de créer des joueurs IA pour effectuer des déplacements sur le plateau automatiquement. Leur nombre devra être modulable pour permettre à l’utilisateur de choisir entre un 1 contre 1, un jeu contre l’IA ou simplement regarder des IA s’affronter.

D'autre part, nous aurons besoins des menus pour permettre au joueur d’interagie avec le programme, donner la case à jouer, et paramétrer la partie.

### L’initialisation de la Partie

Grace à l’interaction du joueur avec les menus, il faudra créer une partie correspondant à ses choix. Il sera aussi nécessaire de placer toutes les pièces sur le plateau, aléatoirement pour les nuages et de façon déterminé pour les autres.

### Le Tour par Tour

Enfin, nous implémenterons dans une classe Main, toutes les boucles qui appellent les différentes méthodes pour le bon déroulement de la partie. Elles devront solliciter les joueurs a leur tour un par un.

### La Fin de Partie

En fin de partie, il faudra arrêter le jeu et déclarer un vainqueur. De plus, comme stipulé dans la consigne, tous les coups des joueurs devront êtres affichés pour résumer la partie. Cela implique de consigner chaque tour dans un cahier de log.

### Les Extensions

Notre jeu devra implémenter deux des trois extensions obligatoires parmi :

* La surcharge de ressource,
* La grille infinie,
* Et la Sauvegarde.

# Solutions Envisagé

Avant de développer le jeu, j’ai passé un long moment pour déterminer la façon la plus efficace et simple de le coder. Comme nous allons le voir par la suite, cela fut un échec sur certains points.

## Pour les Extensions

Le choix des extensions est crucial car les trois propositions peuvent changer drastiquement la façon de coder le jeu. J’ai donc décidé de choisir celle qui a mon sens étaient les plus simple a implémenter soit la grille infinie et la surcharge de ressource.

J’ai réalisé plus tard que la surcharge de ressources était en réalité un cauchemar à implémenter car elle implique énormément de vérifications et de contraintes auquel je n’avais pas pensé (pour en citer : La superposition de deux pièces avec des couleurs différentes, la suppression des pièces au sein du vecteur empêche la lecture, le changement dynamique de la taille de la police pour accommoder plusieurs pieces…)

## Pour l’Affichage

Afin de pousser le projet au bout, j’ai décidé de faire un affichage graphique dans une fenêtre a part du terminal. Pour ce faire, je voulais utiliser la bibliothèque Swing qui géré la création de fenêtres. En actualisant le contenu de cette dernière a chaque tour, je pourrais avoir un affichage dynamique du jeu.

Après avoir déjà réalisé des affichages de plateaux de jeu dans un terminal au semestre dernier. J’ai voulu m’éloigner de ce mode de fonctionnement archaïque qui est très rébarbatif dans la conception. D’autre part, Java est un language originellement développé pour les applets qui semble très adapté à ce genre d’affichage graphique simple.

Ainsi, je comptais créer une classe Plateau qui entendrais la classe Jframe pour stocker les cases.

## Pour le Plateau

Le plateau sera donc une Jframe contenant un vecteur. Ce dernier contiendra des objets de la classe Case qui étendra la classe Jlabel. Les Jlabel sont des classes qui peuvent être placés dans les Jframes. Elles peuvent prendre des couleurs différentes, contenir des chaines de caractères et, en étendant la classe, nous pourrons aussi ajouter d’autres variables intéressantes comme un vecteur contenant les pièces sur la case.

Fain de faciliter la lecture du jeu, les cases qui sont situées sur la première ligne et la première colonne, ne contiendront qu’un caractère indiquant le nom de la ligne/colonne. Cela permettra au joueur de savoir les coordonnées de la pièce qu’il souhaite déplacer.

Les cases jouables seront créées à l’initialisation du plateau dans le constructeur en alternant leurs couleurs. Toutes les cases contiendront une chaine de caractère contenant leurs noms, par exemple, la case sur la deuxième ligne, deuxième colonne sera ‘B2’.

## Pour les Pieces

La classe abstraite Piece sera la base pour la construction de tout le système, elle contiendra la plupart des méthodes pour nécessaires au mouvement de toutes les pièces du plateau.

On créera ainsi des sous classes pour les glaces, les nuages et les véhicules. Comme les différentes pièces ne sont pas soumises à des règles identiques, les méthodes quelles contiennent ne renvoient pas exactement les mêmes résultats.

## Pour les Joueurs et les Logs

Initialement, je comptais créer une classe Tour qui appelle les différents intervenants dans la partie et qui gère les Pieces inertes. Comme nous le verrons, cette idée fut rapidement abandonnée.

De plus, il sera aussi nécessaire de construire un log qui sera affiché à la fin de la partie. Ainsi, je comptais profiter de la classe tour pour faire une pierre de coups, et, tout en gérant le bon déroulement de la partie inscrire les opérations dans ce dernier.

# Algorithmes Choisis

Après une longue réflexion, j’ai commencé à me pencher sur le code en lui-même. Après avoir codé pendant plusieurs heures, j’ai fait face a un mur. En effet, placer les instances de Pièces a l’intérieure même des cases du plateau rend très compliqué l’interaction entre ces dernières.

## Pour l’Affichage du Plateau et des Cases

### Les Cases Jouables

Ainsi, j’ai donc complétement modifié mon code pour séparer le plateau logique qui contient les pièces et le plateau graphique qui contient les cases.

Après chaque coup, le plateau graphique accède au contenu du plateau logique qui stocke un vecteur de pièces. Après avoir nettoyé toutes les valeurs des cases il commence à les remplir.

Chaque case stocke les informations qui permettes de déterminer à quoi elle ressemble. Outre la couleur du fond, on lui attribue une chaine de caractère correspondant à la concaténation du nom de toutes les pièces qui lui sont liés. Afin de gérer la surcharge de ressources, on divise la taille de la police en fonction du nombre de caractères. Finalement, on détermine la couleur de la police. Si, toutes les pièces contenues appartiennent au même joueur, ou qu’une pièce est seule, la case hérite de leurs couleurs. Sinon, afin d’éviter les incompréhensions, en cas de discorde sur la couleur des pièces présentes, le noir est choisi.

Une fois paramétrées, toutes les cases qui sont en réalité des Jlabel sont placés dans l’ordre sur le plateau qui est en réalité une Jframe.

### Les Cases Non Jouables

Il existe quelques cases non jouables sur le plateau, elles servent uniquement de repère pour le joueur, et affichent leurs noms seulement. Pour ne pas les mètre a jour et éviter qu’elles se fassent nettoyer avant chaque coups, on leur attribue un booléen isSide. Si isSide est « true », alors la case esquivera toutes les méthodes de mise à jour.

## Pour les Règles de la Partie.

## Pour les Pieces

## Pour les Joueurs

## Pour le Log de la partie

## Pour les Extensions

# Au-delà du Code

En plus du code lui-même, de nombreux outils ont été découverts lors de ce projet. Ces derniers ont largement participé à bon déroulement du travail et ont leur place dans ce rapport.

## La GUI

Visual Studio Code a permis d’éviter de nombreuses prises de têtes grâce à la coloration de la syntaxe, le formatage automatique, et l’autocomplétions des méthodes. Sans lui, ce projet aurait vite viré au cauchemar.

https://code.visualstudio.com/

## Le Versioning

Grace à GitHub, le suivi du travail et la création du code fut relativement simple. Il a notamment permis de travailler sur plusieurs machines lors des nombreux déplacements ce semestre.

https://github.com/

## Le Framework

Le Framework Maven a permis de faciliter la gestion de la forme du programme grandement. L’outil fut découvert par hasard lors de recherches bibliographiques sur Java.

<https://maven.apache.org/>

## La Documentation

Grace au livre « Programmer en Java » de Claude Delannoy, disponible à la bibliothèque universitaire ce projet a pu être développé avec les bonnes pratiques de java en tête. La bibliothèque Swing été facilement apprise et de nombreuses bases ont étés approfondies.

https://babordplus.hosted.exlibrisgroup.com/primo-explore/fulldisplay?vid=33PUDB\_UB\_VU1&id=991003988839704672&inst=33PUDB\_UB&context=L

# Conclusion

Lors de ce projet, de nombreux problèmes se sont présentés auquel il a fallu trouver des solutions. J’ai découvert un bon nombre d’outil qui seront utiles pendant toute ma vie de bio-informaticien.

Afin de finaliser ce travail. J’ai généré une Java-doc complète, un fichier .jar pour tester le programme facilement, ainsi qu’une page gitHub pour le partage du projet.

Bien que très satisfait du travail que j’ai produit, si le temps ne manquait pas, voici les éventuelles améliorations que j’aurais apporté au programme :

* La sauvegarde de la partie en cours. Cela aurait pu être facilement fait en exportant le plateau logique.
* Des logos pour les pièces auraient été bien plus lisibles que les lettres et couleurs actuelles.
* Une vraie IA qui ne fait pas des coups au hasard rendrait le jeu plus intéressant.
* Finalement le Drag and Drop est probablement ce qui manque le plus à ce jeu. En effet, pouvoir directement prendre les pièces et les déplacer sur la case souhaitée aurait été plus facile pour le jouer. Malheureusement, la documentation que j’ai utilisé pour Swing n’aborde pas ce sujet.

Finalement, en regard du nombre d’heure que j’ai passé à coder un simple plateau d’échec, il m’est désormais plus facile d’apprécier le temps de développement qui a de l’être injecté sur d’autres jeu codé en Java comme Minecraft. La charge gargantuesque de travail que représente un système aussi complexe m’impressionne.