**Léo DESMONTS**

**1B1**

**IUT Vannes – INFORMATIQUE - 2020**

Une image contenant ordinateur, jouant

Description générée automatiquementUne image contenant dessin

Description générée automatiquementProjet de programmation : Zen l’Initié

Rapport final

Table des matières

[1. Diagramme modifié (version console) 3](#_Toc43711525)

[a. Analyse 3](#_Toc43711526)

[b. Conception 4](#_Toc43711527)

[2. Détails des choix techniques 6](#_Toc43711528)

[3. Diagramme en retro-conception (version GUI) 7](#_Toc43711529)

[4. Tests : 10](#_Toc43711530)

[5. Avancement du projet : 11](#_Toc43711531)

[6. Difficultés rencontrées : 11](#_Toc43711532)

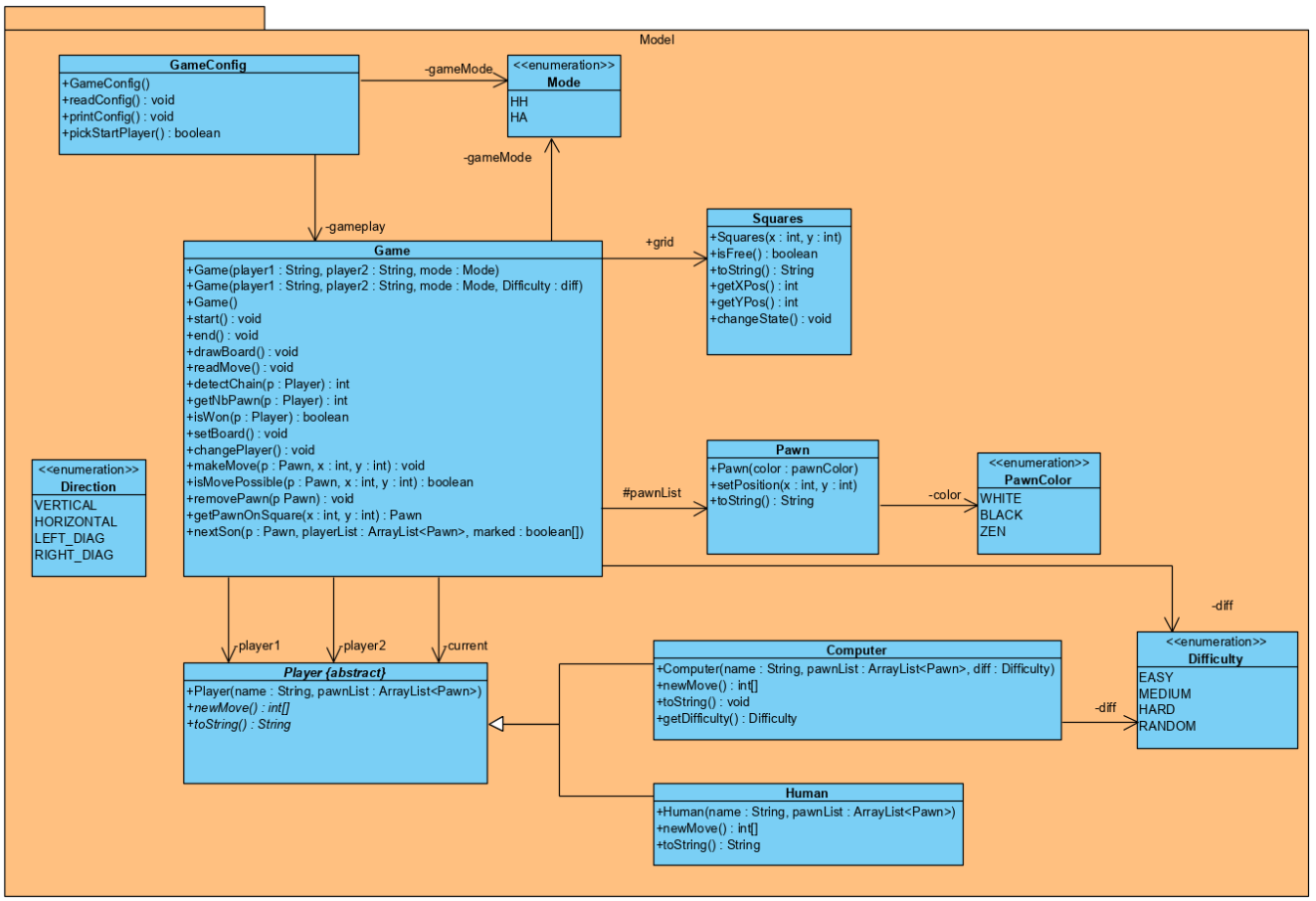
[7. Bilan personnel : 11](#_Toc43711533)

# Diagramme modifié (version console)

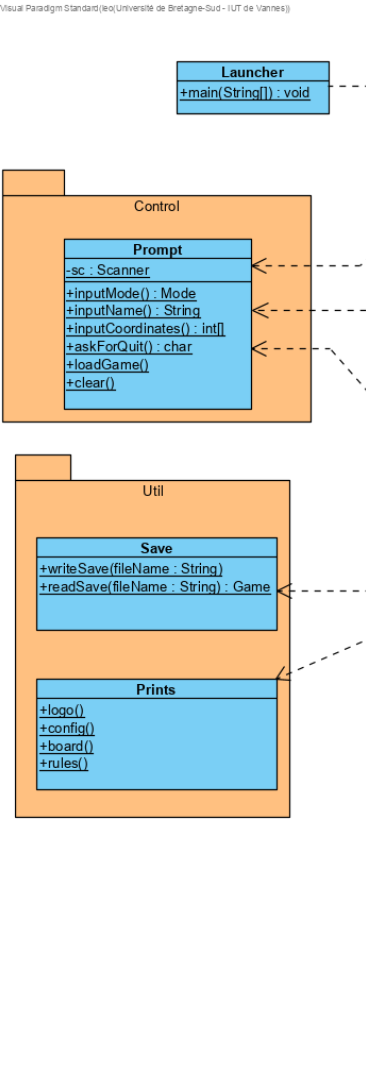
Autres packages et launcher

## Analyse

Package model



## Conception



Les diagrammes ci-dessus sont une version modifiée des diagrammes émis dans la phase de conception. Ils concernent uniquement la version console. Pour voir le diagramme de classe de la version GUI, il a été généré par retro-conception dans la partie 3 de ce dossier.

Les changements majeurs proviennent de l’ajout de la classe Prints dans le package util et de quelques ajouts de méthodes. Le fonctionnement est comme suit :

* La classe Launcher créer une instance de GameConfig.
* GameConfig Lis la configuration désirée par le joueur, en faisant un ensemble successif de Prints (clear, logo, config, etc) et de Prompts (inputMode, inputNames, etc). Une fois tous les paramètres déterminés, la classe créer une instance de Game.
* La classe Game, selon les paramètres qu’elle a reçus, créer les objets dont elle aura besoin (deux Player, un tableau de Square, un tableau de Pawn…). Elle lance ensuite sa boucle de jeu principale, qui n’est d’’autre qu’une boucle qui demande au joueur courant de saisir des coordonnées, qui bouge un pion, et change de joueur. La condition de sortie de la boucle est la victoire d’un des joueurs.
* On utilise des Enumeration (Mode, PawnColor) pour ne pas avoir des valeurs numériques qui ne serait pas très évocatrices.

# Détails des choix techniques

Parmi les choix que j’ai dû faire, je vais commencer avec le stockage des pions. Ces derniers sont tous, indifféremment de leur couleur stocké dans une seul ArrayList. Je sais que certains de mes camarades ont choisi une solution différente, c’est pour cela que je l’explique ici. Cette manière de faire me permet de toujours avoir le pion ZEN lié aux autres, étant donné que ce dernier peut changer de camps à chaque tour. Le point négatif est qu’il faut régulièrement boucler dans la liste de pions pour récupérer dans des variables locales les pions d’un joueur en particulier (dans la méthode detectChain par exemple).

D’un point de vu algorithmique il convient d’expliquer la manière dont on vérifie si la victoire est remportée et si un mouvement est possible.

Pour vérifier la victoire, on utilise la méthode isWon, qui compare le résultat de detectChain, avec le nombre de pions encore en jeu du joueur testé. Si les deux nombres sont égaux, c’est une victoire. La méthode detectChain récupère tous les pions du joueur dans une arrayList, et effectue un parcours de graph en longueur, en partant du premier pion de la liste comme premier sommet. Lorsque tous les sommets accessibles sont marqués, il renvoie le nombre de ces sommets. S’il ne correspond pas au nombre de pions encore en jeu, c’est que certains pions ne sont pas accessibles et sont donc dans une autre composante connexe ; ils ne sont donc pas tous liés ensemble.

Pour vérifier qu’un mouvement est possible, une batterie de test est effectuée. On commence par vérifier que les coordonnées du pion et de la case d’arrivée sont bien dans le tableau, puis que le pion que l’on essaye de déplacer appartient bien au joueur dont c’est le tour. Puis on détermine la direction du mouvement en utilisant les positions relatives des pions. Enfin on trouve la distance entre les deux cases et le nombre de pions présents sur la ligne de déplacement (ces deux nombres doivent correspondent), et qu’aucun pion adverse ne se trouve sur la trajectoire. Si la case d’arrivé est occupée, et que le pion est un pion advers, on retire ce pion du jeu et on place le pion déplacé sur cette case. Si c’est un pion allié on annule le mouvement. Si le pion est le ZEN, des test supplémentaires seront effectués. Il faut vérifier que parmi les 8 cases qui l’entourent, au moins une est utilisé par un pion.

Une fonctionnalité que je n’ai pas peu implémentée mais qui aurait été intéressante d’un point de vu algorithmique est un niveau de difficulté supplémentaire pour le bot. Je vais donc expliquer son fonctionnement ci-dessous.

Le principe aurait été de parcourir la liste des pions du bot, et de déterminé celui dont la distance avec une case prédéfinis (de préférence avec la case du milieu 5,5) est la plus élevé, en utilisant les positions relatives comme pour detectChain. Ensuite, pour ce pion, il faut déterminer les mouvements possibles en faisant une méthode dont le fonctionnement serait l’inverse de isMovePossible (au lieu de vérifier qu’un mouvement est possible, on renvoi les mouvements possibles). On obtient alors une liste de case, et on choisira cette fois la case la plus proche de la case prédéfinis, suivant la même technique que précédemment.

De cette manière, les pions vont se rapprocher de plus en plus du centre, et avoir beaucoup plus de chance de former une chaine.

# Diagramme en retro-conception (version GUI)Une image contenant capture d’écran, carte Description générée automatiquement

Une image contenant capture d’écran, carte

Description générée automatiquement

Diagramme version GUI de retro-conception

Le launcher crée une instance de MainWindow. Ce JFrame est le conteneur de tous les JPanel. Il est basé sur un CardLayout. Chaque JPanel est considéré comme une carte, qui est ramener sur le dessus en fonction des besoins. Le premier JPanel est le MainMenu.

Tous les boutons dans le menu sont gérés par le ListenerButtonGame. En fonction du bouton, le listener se réfère à des méthodes différentes présente dans MainWindow, qui permettent d’aller un JPanel spécifique. Dans certain cas la méthode aura en plus du changement un autre rôle (créer une instance de game, sauvegarder une partie, etc).

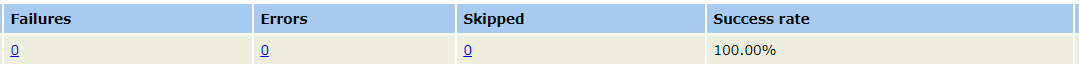
Les classes GUISquare et GUIPawn modelise les pions et carrés qui seront affiché sur le plateau de jeu, et avec lesquels on pourra interagir avec les ListenerCase et ListenerPawn. Ces deux listener ont quasiment les mêmes méthodes, mais les deux sont indispensable. En effet, lorsqu’un joueur clique sur une case qui est occupé par un pion, la case en dessous de reçoit l’ActionEvent, il faut donc que le pion récupère les informations et les relais aux cases. La classe board est un ensemble de pions et de cases.

Les autres classes sont toutes des JPanel avec un certain nombre d’attributs statiques auxquels les autres classes peuvent accéder facilement, principalement des JButton.

On retrouve :

* MainMenu, qui offre le choix de quitter le jeu, d’afficher la page de règles, ou de jouer.
* Rules, qui affiche les règles, et qui offre un bouton « back » pour revenir au menu principal.
* Saves, qui permet de choisir entre une nouvelle partie ou d’en charger une. Elle offre également un troisième bouton, « back », pour revenir au menu principal.
* SavesList, qui list toutes les sauvegardes disponibles dans une JList, et un bouton pour charger la sauvegarde choisit. Elle offre également un bouton « back » pour revenir à la page Saves.
* GameMode, qui permet de choisir entre le mode 1 joueur ou 2 joueurs. Elle offre également un bouton « back » pour revenir à la page Save.
* PlayerName qui permet de choisir les nom du/des joueurs en fonction du mode choisis. Elle offre un bouton pour lancer la partie, et d’un bouton « back » pour revenir à la page GameMode.
* PlayPanel, qui est l’interface de jeu. Ce Panel permet d’afficher le Board, ainsi que le joueur dont c’est le tour. Elle offre un bouton pause, et un bouton quitter. Le bouton pour quitter propose ou joueur de sauvegarder ou non. Dans les deux cas, le joueur est ramené au menu prinscipal MainMenu.
* Pause, qui est une image fixe en pause, qui une fois cliqué renvoie vers PlayPanel.

# Tests :



J’ai dû me résoudre à supprimer certains de mes tests concernant des classes qui demandais dans leur constructeur des inputs utilisateur (configuration de partie etc), ce que JUnit ne gère pas du tout. J’ai cependant effectué des tests unitaires comme vu en M1103 en début d’année. Ce genre de test est très long et devient rapidement illisible sur de grandes batteries de test, c’est pour cela que j’ai décidé de ne pas en inclure ici. Ces derniers aurait uniquement pris de la place et auraient de toute manières été très compliqué à lire.

Concernant les tests JUnit établis durant la phase d’analyse, la plupart ont correctement fonctionné, et m’ont permis de relever des erreurs. J’ai rencontrée quelque erreurs provenant des tests eux même, mais ces derniers venaient uniquement de classe que j’avais renommé (Color => PawnColor, pour ne pas confondre avec la classe java.awt.Color, utilisé dans la partie graphique de l’application).

La compilation du projet avec Ant (là où les tests étaient lancés) prenait un peu de temps (a cause de la compilation, des tests, de la javadoc, etc), ce qui est handicapant quand on a besoin de compiler très souvent, par exemple durant le développement de l’interface graphique. J’ai donc la plupart du temps utilisé un petit script bash qui ne faisait que compiler. Je m’assurais donc de penser à lancer les tests avec Ant à chaque grande étape : classe terminé, fonctionnalité majeure, etc.

Les test JUnit ne sont pas applicable aux classes graphiques, c’est pour cela qu’il n’y avait pas de nouveaux tests dans la deuxième version du projet. J’ai donc totalement retiré les dossiers test, testReport et le fichier build.xml qui étaient devenu inutile dans la version GUI.

Pour la partie graphique, j’ai donc effectué (entre autre) les test défnit dans le cahier des charges. Bien d’autres test on été effectué sur cette partie graphique, mais tous les afficher dans une tableau encombrerait une vingtaines de pages, ce qui n’est pas le but ici. Quelques exemples :

|  |  |  |  |
| --- | --- | --- | --- |
| **Fonctionnalité** | **Scenario de test** | **Résultat attendu** | **Résultat réel** |
| Couper les sons et musique  (On pourra faire les deux indépendamment pour un meilleur test) | * Se déplacer dans les menus de la manière suivante   + Paramètre * Décocher la case « son » * Décocher la case « musique » | Les musiques et sons du jeu ne doivent plus être audible en se déplaçant dans les menus ainsi qu’en jeu | NON IMPLEMENTE |
| Afficher les règles du jeu | * Se déplacer dans les menus de la manière suivante   + Règle du jeu | Une fenêtre affiche les règles du jeu | Une fenêtre affiche les règles du jeu |
| Quitter le jeu | * Se déplacer dans les menus de la manière suivante   + Quitter   + Cliquer oui sur le pop-up | Le jeu doit se fermer totalement | Le jeu doit se fermer totalement |
| Lancer une nouvelle partie contre l’ordinateur | * Se déplacer dans les menus de la manière suivante   + Jouer   + Nouvelle partie   + Solo   + Démarrer | Une partie contre l’ordinateur se lance | Une partie contre l’ordinateur se lance |
| Sauvegarder une partie | * Se déplacer dans les menus de la manière suivante   + Jouer   + Nouvelle partie   + Solo   + Démarrer   + Pause   + Sauvegarder * Quitter la partie puis se déplacer dans les menus de la manière suivante   + Jouer   + Reprendre une partie * Choisissez la sauvegarde précédente | La partie reprise doit être au même état d’avancement que le la partie précédemment quittée. | La partie reprise est au même état d’avancement que le la partie précédemment quittée. |

# Avancement du projet :

Toutes les fonctionnalités « de base » ont été implémenté, mise à part de différents thèmes graphiques. La seule fonctionnalité avancée implémenté est la gestion de plusieurs sauvegardes, qui me paraissait finalement plus importante que plusieurs thèmes graphiques. La principale raison que j’évoquerais ici est le manque de temps. En effet je trouve personnellement qu’une semaine de codage intensif c’est plutôt court. Je trouve ça dommage car j’ai beaucoup aimé travailler sur ce projet, et j’aurais aimé implémenter plus de fonctionnalités, tel que l’aide à la décision (les mouvements possibles pour le pion sélectionné son coloriés), une partie scénarisée pour apprendre les subtilités du jeu qui est néanmoins plutôt complexe, ou encore au moins un niveau « d’IA » plus avancé que les mouvements aléatoires actuel, qui ramènerait les pions du bot le plus au milieu possible.

# Difficultés rencontrées :

Je n’ai quasiment pas rencontré de problèmes non ponctuels (les « googlesolvables »). La seule chose qui m’a posé problème était la manière de lier la model de jeu utilisé dans la version console au package view, chargé de de l’affichage graphique.

J’ai finalement décidé de faire une classe principale qui sera un JFrame, qui sera la fenêtre affichée, et qui viendra accueillir les différents JPanel (les onglet) qui seront affiché en fonction des besoins (basé sur un CardLayout, qui permet ce genre de choses très facilement). La classe principale contient également la totalité des méthodes (static) qui permettent de naviguer d’un onglet à l’autre.

La boucle de jeu dans la classe model.Game n’est plus nécessaire. Il y a deux listeners concernant le jeu directement, qui sont lié aux cliques sur des cases ou des pions, et qui font appel aux méthodes de model.Game. Le dernier listener permet de gérer tous les boutons dans les menus.

Un autre petit problème que j’ai rencontré était lié à ma classe Prompt. En effet, à chaque fois que j’appelais une méthode de cette classe je créais un Scanner qui écoutait sur l’entrée standard « System.in » A chaque fin de méthode, je fermais le scanner (scan.close()). Ce que je ne savais pas était que cette instruction ne ferme pas uniquement le scanner, mais également le flux d’entrée (ici l’entrée standard), ce qui m’empêchait de la réutiliser par la suite. Appeler deux méthodes de la classe Prompt se soldait donc systématiquement par le levé d’une exception. Pour pallier ce problème, j’ai créé un attribut statique (car toute mes méthodes sont statiques) Scanner sur System.in, qui n’est jamais fermé, et auquel toutes les méthodes font appelées.

Le seul vrai problème rencontré est lors de la sauvegarde d’une partie. Je n’avais pas précisé de valeur pour le serialVersionUID car j’ignorais son existence. Il s’agit d’un hash SHA, qui agit comme une signature lors de la serialization, notamment pour que java reconnaissent la version avec laquelle il travail. Si cette valeur n’est pas explicitement exprimée, une valeur par défaut est calculée. Le calcul de cette est extrêmement sensible au moindre changement dans la classe, et si ce dernier ne correspond pas à celui d’une partie chargée, une exception est lancée, et la partie ne se charge pas. Le problème n’est pas compliqué à résoudre, mais il faut en comprendre son origine.

# Bilan personnel :

J’ai personnellement beaucoup aimé travailler sur ce projet. Il m’a permis de voir un « vrai » projet dans sa totalité. Il est vrai que je mène beaucoup de projet informatique sur mon temps libre, mais les phases d’analyses sont **très** brèves, alors qu’elles sont très importantes, et facilite grandement la suite, notamment l’implémentation. Le seul aspect qui manquait ici était la communication avec le client, mais nous avons pu nous familiariser quelques peu avec ce côté-là durant le projet web, ainsi que le projet citoyen.

Si je devais reprocher quelque chose à l’organisation de ce projet, ce serait sa réalisation seul (la plupart des projets dans des entreprises se font en équipe), ainsi que sa répartition : une semaine de codage, par rapport au temps disponible pour faire le cahier des charges par exemple, c’est assez peu. Un peu plus de temps m’aurait permis de faire un travaille dont j’aurais été encore plus fier. Je suis cependant très fier du travail que j’ai produit, et je n’ai aucun regret. Mais il faut garder à l’esprit que les conditions de travail cette année étaient très exceptionnelles, et que ces reproches ne sont donc que théoriques et dirigés vers personnes.

Enfin si je devais conseiller les étudiants de l’année prochaine, je dirais qu’il faut ne surtout pas négliger la partie conception, et commencer à sérieusement à réfléchir au codage avant la semaine de codage, surtout si elle ne dure qu’une semaine, car même avec peu de semaine, le temps passe très vite. L’organisation est la clef. Il faut se fixer des objectifs, et s’y tenir le plus possible. Dans son planning il peut être judicieux de réserver une journée « libre » avant le rendu. En effet, malgré un planning strict et beaucoup de volonté, on n’est jamais à l’abris d’un problème qu’on peine à résoudre et qui nous met en retard. Cette journée peut permettre de contrer un tel retard. Dans le cas contraire elle peut permettre de peaufiner les éléments de rendu, ce n’est donc pas du temps de perdu. Dernier conseil, pour chaque problème, Google est notre meilleur ami, mais il ne faut pas oublier qu’on a avec soit une promotion de 100 étudiants, pour la plupart près à venir en aide, il ne faut donc pas hésiter à demander, et aussi d’en offrir lorsque c’est possible.