M2107 - Projet de programmation

Projet de programmation

Les Bâtisseurs : Moyen-Âge

Rapport de fin de projet

BOCANDÉ Yvan, 1ère année – Groupe A1, Année 2020-2021

Table des matières

1) Mise à jour des diagrammes du cahier d'analyse

Package Game :

D'un point de vue général, les classes présentes dans le package et la disposition de celles-ci reste plutôt similaire à celles présentes dans le diagramme du cahier d'analyse.
On a juste eu quelques changements.
- Premièrement les classes mode et Game ont disparu car j'ai décidé que l'utilisateur choisirait le - mode de jeu au lancement de la partie et non avec un fichier de configuration.
- De plus, la classe difficultPlayer n'a pas eu le temps d'être implémenté. C'est pourquoi on ne la retrouve pas dans le diagramme final.
- Une nouvelle classe est apparu dans ce nouveau diagramme de classe. C'est la classe chantier qui permet de représenter un bâtiment en construction. Cette classe est nécessaire au bon fonctionnement de mon code car elle permet une grande simplification dans de nombreuses tâches qui permettent aux joueurs d'exercer des actions. On retrouve dans cette classe le bâtiment en construction et les ouvriers qui travaillent en attribut. Ses méthodes permettent d'ajouter des travailleurs ou de consulter l'avancement du « chantier ».
- Enfin de nombreuses dépendances auxquelles je n'avais pas pensé ont été nécessaire au bon fonctionnement du jeu. On peut par exemple penser au fait que j'avais mis qu'un seul deck pour la classe Game alors que deux decks sont indispensables au bon fonctionnement du jeu (un deck d'ouvriers et un deck de bâtiments).

Nous allons maintenant prendre les classes une à une afin d'observer leurs changements.

- Classe Game : On y a rajouté des méthodes permettant à l'interface graphique d'avoir des méthodes spécifiques à ses besoins comme par exemple la méthode classementG() qui returne une ArrayList de String avec le classement tandis que la méthode classement de base affichait juste à l'écran le classement.
  De plus, les quatres attributs Players ont été remplacé par une ArrayList. En effet, ce choix est plus facile à implémenté car nous ne savons pas le nombre de joueurs avant d'avoir lancé une partie.
  Un grand changement est que les méthodes permettant la sauvegarde et le chargement de partie sont maintenant présentes dans la classe Game alors que dans le diagramme initial, elle se trouvait dans un autre package. Ce choix a été fait afin de simplifier le chargement de la partie. De cette manière toutes les opérations de chargement sont dans la même méthode alors qu'elles auraient été divisées si on avait opté pour les mettre dans une autre classe.
  Enfin quelques méthodes comme Lancement() ou fin() ont été ajouté afin de permettre un meilleur découpage du code.

- Classe Deck : On a démultiplié la méthode permettant d'initialiser un deck. En effet, nous avons fait une méthode qui permet d'initialiser chaque type de deck (ouvrier et bâtiments) puisque chaque type de carte possède son fichier d'initialisation différent. On passe donc en paramètre du constructeur un String qui permet de donner le type du deck souhaiter.

- La classe Card a subi le remplacement de l'attribut gold par l'attribut Tuile. Il y avait eu une incompréhension des ressources demandé/produits par ces deux cartes.

- Les classes worker et Bulding n'ont subi qu'un seul changement. On leur a enlevé leur méthode toString car les cartes seront affichées toutes ensembles dans la classes deck sans utiliser de méthode toString.

- La classe Machine a hérité d'un nouvel attribut qui est le nombre de points de victoires. Cet attribut avait été oublié lors de la phase de conception. On y a aussi rajouté un attribut static à 0 qui représente le coût de la machine. Enfin on lui a enlevé sa méthode toString car les cartes seront affichées toutes ensembles dans la classe deck sans utiliser de méthode toString.

- On a rajouté une méthode toString dans la classe Player permettant d'afficher les caractéristique d'un joueur (points de victoire, écus,ects) et ses cartes lors que c'est son tour de jouer. Cela permet donc un affichage plus facile dans la version textuel puisque tous les éléments du joueur seront print ensemble dans la même méthode. De plus, ont aajouté en attribut de cette classe une ArrayList de chantier qui permet de gérer plus facilement les bâtiments en construction que la solution qui devait être implémenté si on avait suivi le cahier d'analyse (une ArrayList de Buldings). De plus on a remplacé l'ArrayList de Workers par une Arraylist de Card parce que les Machines sont aussi des ouvriers lorsqu'elles sont construites.

- Les classes HumanPlayer et auto Player n'ont pas été changé. Elles permettent toujours au joueur de chaque type de jouer.

Le package View :

Ce package a gardé une construction similaire à la construction qui était prévu par le diagramme d'analyse du cahier d'analyse.
On y a rajouté des classes JPanel qui représentent des pages de notre application que nous avons oublié. Il y a par exemple la classe Load qui représente la page permettant le chargement d'une partie ou la classe InitP qui permet de recueillir le nom des joueurs. Ces changements sont mineurs.

Nous allons maintenant voir les changements exercer dans chaque classe :

- La classe menu est très similaire à celle prévu dans le diagramme d'analyse. On y a juste rajouté les boutons quitter et options ainsi que la méthode qui permet de mettre une image en fond. Ces changements résultent d'oublis lors de la création du diagramme d'analyse.

- La classe options a juste eu un petit changement qui est l'ajout d'un JCheckbox en attribut permettant à l'utilisateur d'activer/désactiver la musique facilement.

- La classe Rules a disparu. A la place nous avons choisi d'ouvrir un pdf contenant les règles car nous n'avons pas eu le temps de mettre en forme toutes les règles dans un JFrame.

- Dans la classe Lancement, les 4 JTextField prévu à l'origine ont été remplacé par une ArrayList de JTextField permettant une plus simple initialisation du Panel en fonction du nombre de joueur.

- La classe interfce a été remplacé par la vueTotale. Le rôle de la classe est le même mais le nom a été changé dans un souci de clarté du nom des classes. On a rajouté en attribut tous les panels qu'on a jouté dans le package view. On a donc ajouté tous les getters correspondants à ces attributs.
  De plus on y a ajouté deux méthodes. :
  - La méthode createInitP permet à la vue totale de crée cette page une fois que le nombre de joueurs a été choisi sur une autre page. En effet, la création de cette page ne peut être créer dans le constructeur puisque nous ne savons pas lors de la création de la vueTotale le nombre de joueurs dont il faudra demander le nom.
  - De plus, on a ajouté une méthode initListenerBoard dans un souci de meilleur découpage de code. En effet, il était plus lisible d'initialiser le listener de tous les attributs de Board dans une méthode à part du constructeur.

- La classe Board a subi beaucoup d'ajouts.
  On a ajouté en attributs deux JToolbar permettant une meilleur lisibilté du Board lors du jeu.
  De plus, lors du cahier d'analyse j'avais décidé de représenter toutes les bâtiments et ouvriers du joueur par seulement deux JLabel. Ces deux JLabels ont été remplacés par deux ArrayList de JButton. On a choisi de représenté chaque carte par un JButton car cela les rends cliquables et c'est donc plus facile pour l'implémentation des interactions de l'utilisateur.
  De plus, j'ai choisi de représenter chaque carte de la pioche par un JButton en attribut. Cela facilite l'implémentation et l'affiche de la pioche.
  On a aussi rajouté un JLabel qui où il est écrit tour où est le jeu.
  Enfin, nous avons rajouter des boutons en attribut permettant de prendre des écus (ils avaient été oublier lors de la phase d'analyse.
  Quant aux méthodes, on a ajouté des méthodes permettant de mettre à jour les pioches et les cartes des joueurs. Ces méthodes sont indispensables au bon fonctionnement du jeu et avaient été oublié lors de la phase d'analyse.
  De Plus, on a ajouté des méthodes pour mettre à jour les différents Label d'information (écus, nombre d'actions restantes, etc…).
  Une méthode a aussi été rajouté afin de formater le texte des JLabels afin de ne pas répéter plusieurs actions successives dans le code de manière répété. Cette méthode a donc été implémenté dans un souci de lisibilité du code.

Package Controler

Le package n'a pas changé et contient toujours une seule classe qui permet le contrôle de toutes les pages de l'interface graphique.

La classe Listener a subi des ajouts de méthodes. Ces méthodes sont utilisées pour un meilleur découpage du code afin de rendre le code plus lisible. En effet, par exemple la méthode ListenerBoard permet de gérer exclusivement les interactions avec le board. Cela permet de rendre plus lisible les réactions aux interactions utilisateurs.

On a rajouté une méthode load qui permet de charger une partie lors du clic sur le bouton prévu à cet effet. On aurait pu se passer de cette méthode et le mettre dans la méthode principale mais ici encore, on aurais perdu en lisibilité du code.

Pour résumer, toutes les méthodes rajouter dans le listener ont été rajouté dans un objectif de meilleure lisibilité du code.

Package Utilitaire :

Deux nouvelles classes font leur apparition dans ce package. Elles n'étaient pas présentes dans le diagramme de conception du cahier d'analyse parce que je n'avais pas pensé qu'elles pourraient être utiles.

- La classe Scan permet l'entrée d'un entier dans la console. De plus sa méthode vérifie que l'entrée est bonne. Cette classe permet de factoriser énormément de code dans la package Game puisque il n'y a plus besoin de vérifier la validité de l'entrée de l'utilisateur à chaque entrée, la méthode scanNum() s'en occupe.
- La classe Musique permet de lancer une musique et n'avais pas été mis dans le diagramme de base puisque ce n'était pas une fonctionnalité de base.

2) Description des choix techniques et algorithmique

a) Version terminale

Le lanceur créé une instance de Game, qui s'occupe de gérer la boucle de jeu, la création de partie, la demande de nom des joueurs.

Premièrement Game lance le différent menu permettant l'initialisation des joueurs et des paramètres de la partie (nouvelle partie, partie chargée etc). En suite la classe la boucle du jeu.

La **boucle du jeu** ne s'arrête pas tant que la méthode Agagner ne retourne pas de joueur gagnant. Chaque joueur utilise sa méthode play tour à tour tant qu'il reste des actions restantes à chaque joueur. Ces actions sont réinitialisées avant le début du tour de chaque joueur. La méthode play demande au joueur l'action qu'il souhaite effectuer et, si cela est possible, l'action est effectuée.

La **fin du jeu** se déclenche lorsqu'un joueur possède plus de 17 points. A ce moment, le joueur gagnant est affiché dans la console et le classement des points est aussi affiché. Enfin, le joueur peut retourner au menu principal ou recommencer sa partie.

b) Version graphique

Le lanceur crée une instance de vueTotale qui va permettre au joueur d'initialisé les joueurs ainsi que tous les paramètres du jeu. Le joueur va sélectionner tous les paramètres en naviguant dans le menu proposé par la vueTotale. En fonction de tous les paramètres, un Board qui va permettre le jeu va être créé.

La **boucle du jeu** est gérée par le listener. En effet chaque joueur va exercer ses actions tour à tour en appuyant sur les cartes qui correspondent aux actions qu'il souhaite faire. Lorsqu'il n'a plus d'action, il appui sur « finir mon tour » ce qui passe au joueur suivant. On peut donc dire que ce sont les utilisateurs qui font avancer la boucle du jeu à la vitesse qu'ils le souhaitent.
La « boucle » se termine donc une fois qu'un joueur à 17 points et que tous les joueurs ont joué le même nombre de tour. Une interface de fin de jeu est affichée à ce moment-là. Une autre manière de terminer la « boucle « est d'appuyer sur le bouton « save » qui va permettre à la partie d'être sauvegarder et ce qui va fermer le jeu. Toute la boucle se passe sur le JPanel Board.
On remarque donc que dans le cas d'un joueur humain, la méthode play n'est pas utilisé. En effet,

Une fois que le jeu est fini, on peut retourner sur le menu principal si la partie à été terminé ou relancer le jeu si la partie à été sauvegarder

c) Fonctionnement du bot

Le mode fonctionne de manière aléatoire avec des probabilités et des restrictions.
En effet celui-ci a plus de probabilité de d'effectuer certaines actions (envoyer un ouvrier travailler ou finir un chantier). Cela permet de diminuer le nombre de tours dont va avoir besoin le bot afin de gagner ses 17 points.

De plus, j'ai restreint le nombre de chantier qu'il peut ouvrir à 4 et le nombre d'ouvriers qui peut avoir en même temps à 6 car sinon il ouvre un nombre de chantiers beaucoup trop grand et il ne finit jamais de chantier. On peut donc dire que ce principe améliore le bot.

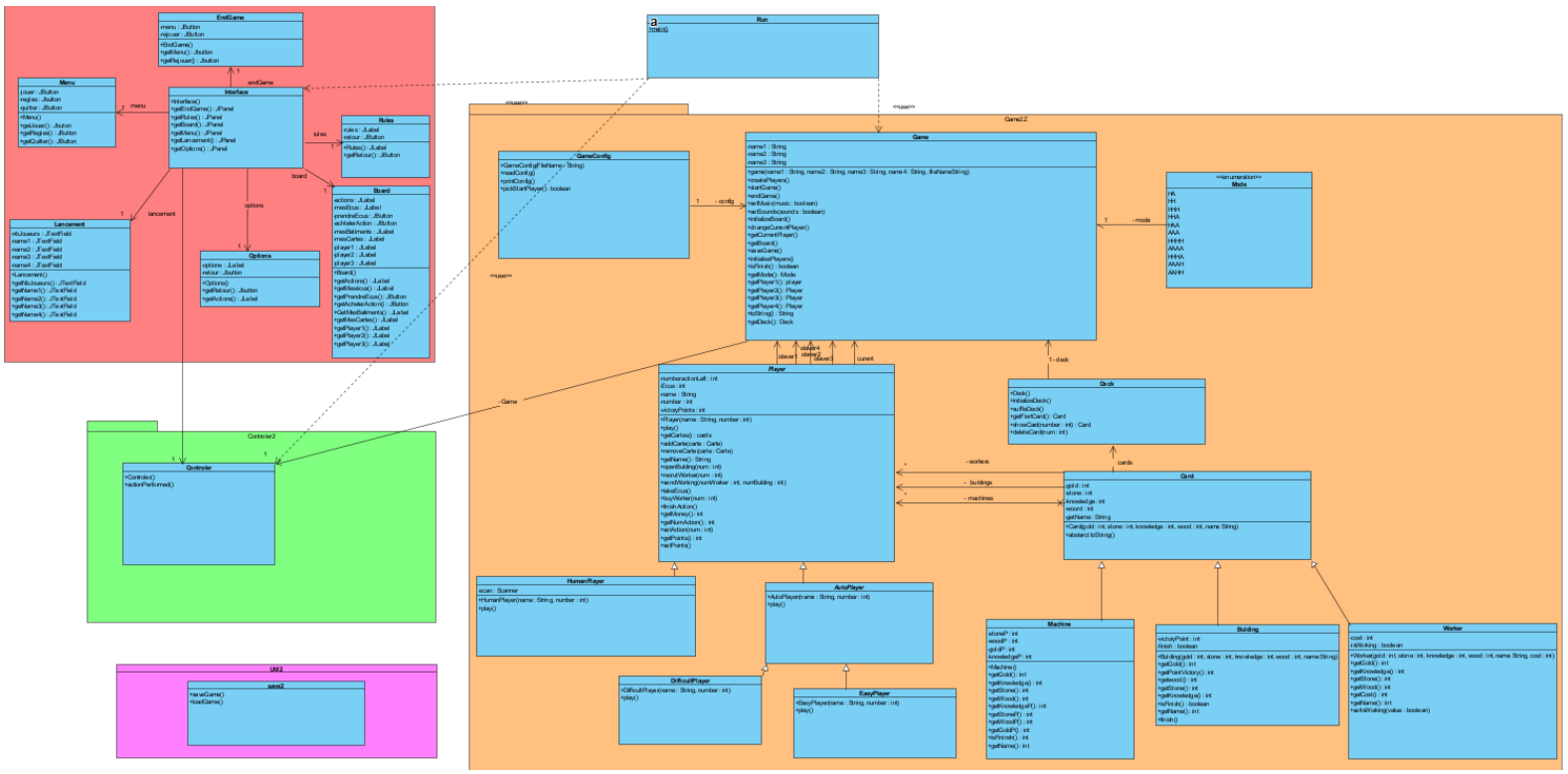
Je pense que les petites améliorations que j'ai mis en place sur le bot permettent une réelle utilité des bots qui vont être un minimum compétitif plutôt que d'avoir des bots qui ne gagneront jamais et qui n'ont donc pas de réelle utilité.

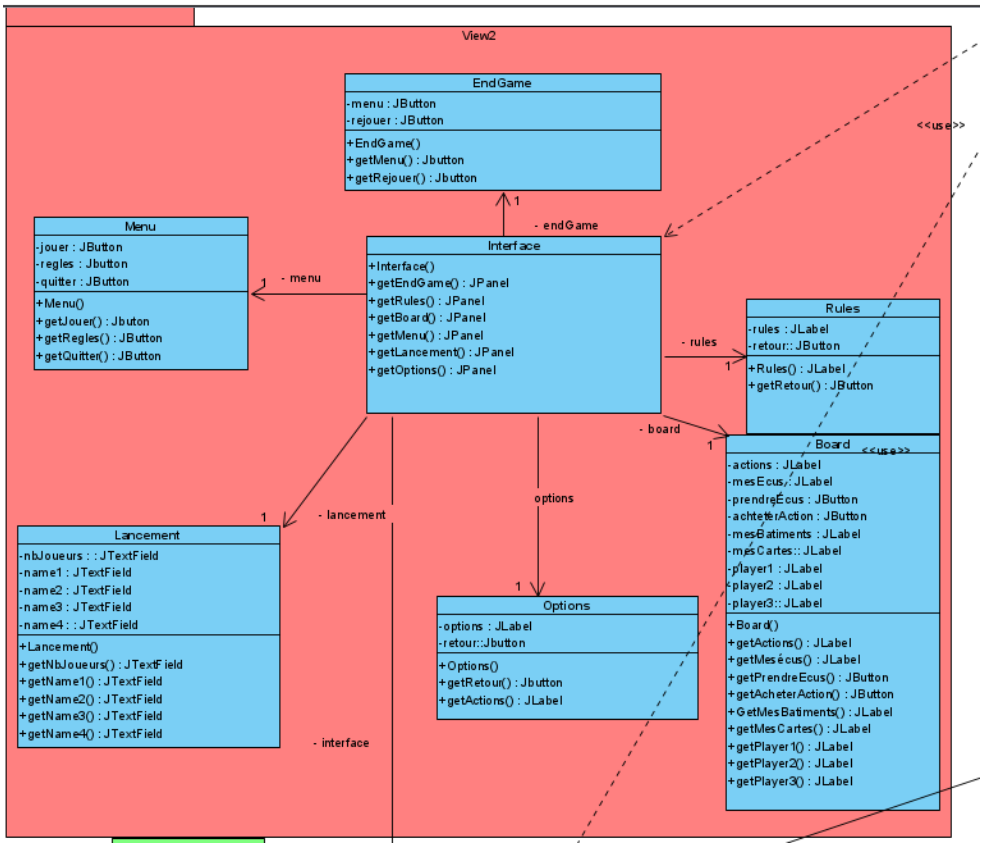
d) Gestion des sauvegardes

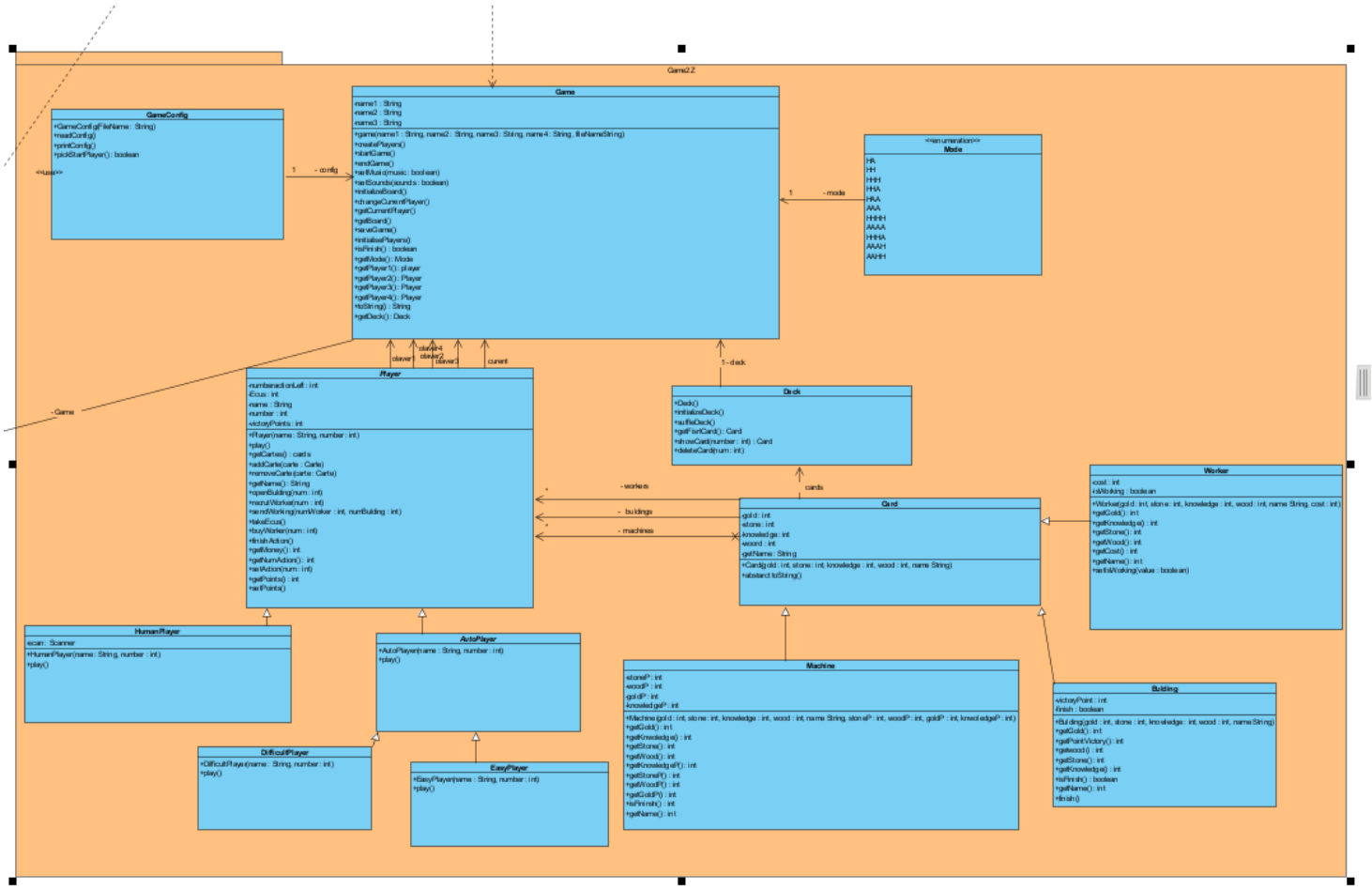
Les parties sont sauvegardé en utilisant la méthode de la sérialisation de la classe Game dans un fichier.txt. Pour charger une partie, on prend tous les caractéristiques de la classe sérialiser et on les attributs à la classe Game en cours. Enfin on passe directement à la boucle du jeu puisqu'on n'a pas besoin d'initialiser tous les paramètres de Game.

3) Diagramme de classes d'implémentation obtenu par rétro-conception

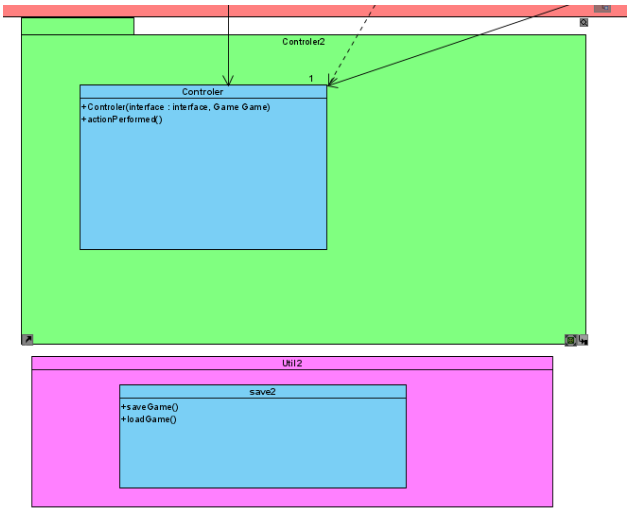
a) Diagramme du cahier d'analyse




View2
EndGame
-menu : JButton
-rejouer : JButton
+EndGame()
+getMenu() : Jbutton
+getRejouer() : Jbutton
<<use>>
1
- endGame
Menu
-jouer : JButton
-regles : Jbutton
-quitter : JButton
+Menu()
+getJouer() : Jbuton
+getRegles() : JButton
+getQuitter() : JButton
1
- menu
Interface
+Interface()
+getEndGame() : JPanel
+getRules() : JPanel
+getBoard() : JPanel
+getMenu() : JPanel
+getLancement() : JPanel
+getOptions() : JPanel
- rules
1
Rules
-rules : JLabel
-retour:: JButton
+Rules() : JLabel
+getRetour() : JButton
- board
1
Board
<<use>>
-actions : JLabel
-mesEcus : JLabel
-prendreÉcus : JButton
-acheterAction : JButton
-mesBatiments : JLabel
-mesCartes:: JLabel
-player1 : JLabel
-player2 : JLabel
-player3:: JLabel
+Board()
+getActions() : JLabel
+getMesécus() : JLabel
+getPrendreEcus() : JButton
+getAcheterAction() : JButton
+GetMesBatiments() : JLabel
+getMesCartes() : JLabel
+getPlayer1() : JLabel
+getPlayer2() : JLabel
+getPlayer3() : JLabel
1
- lancement
Lancement
-nbJoueurs : : JTextField
-name1 : JTextField
-name2 : JTextField
-name3 : JTextField
-name4 : : JTextField
+Lancement()
+getNbJoueurs() : JTextField
+getName1() : JTextField
+getName2() : JTextField
+getName3() : JTextField
+getName4() : JTextField
- interface
options
1
Options
-options : JLabel
-retour::Jbutton
+Options()
+getRetour() : Jbutton
+getActions() : JLabel

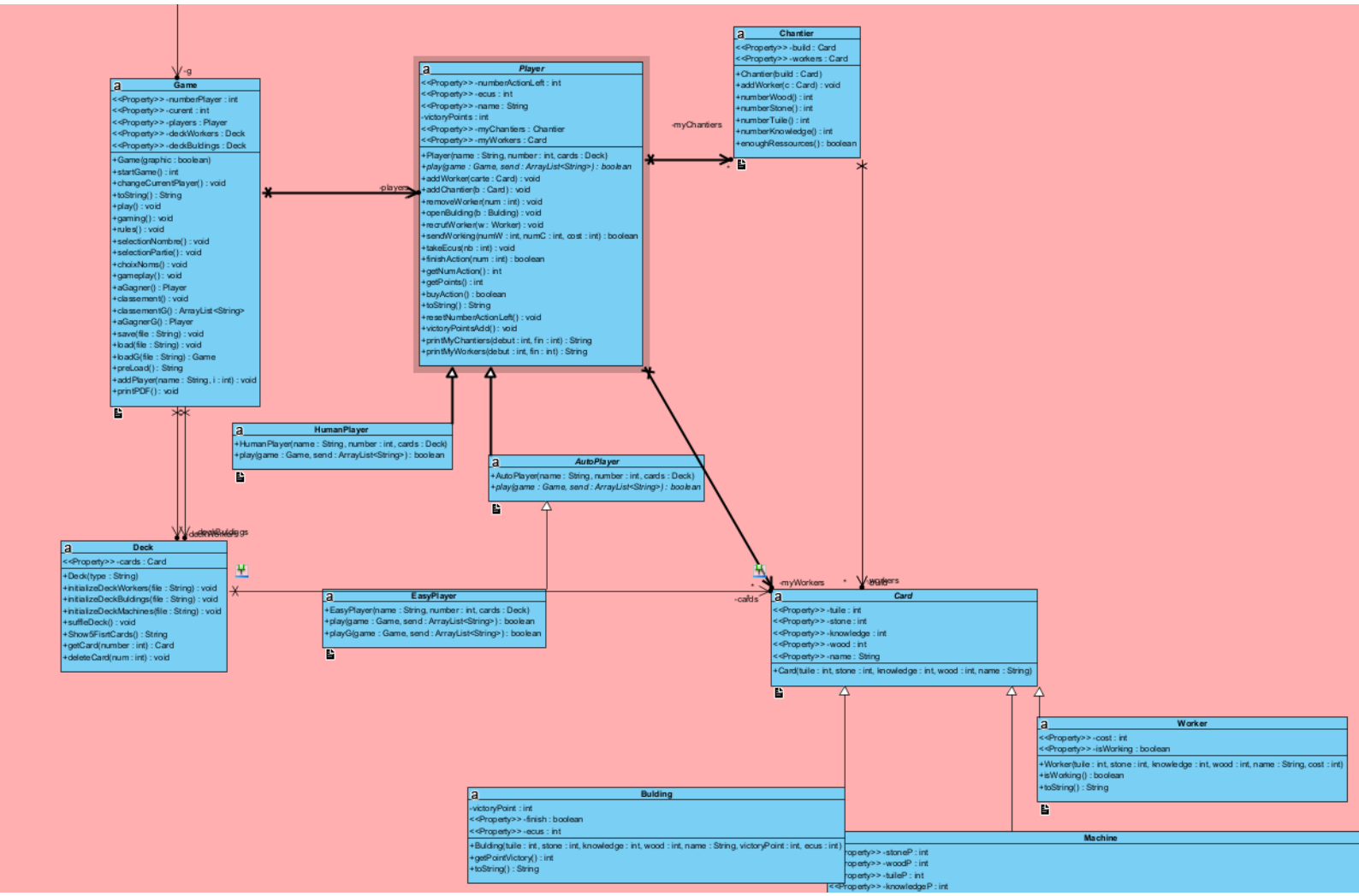


A retrouver en annexe



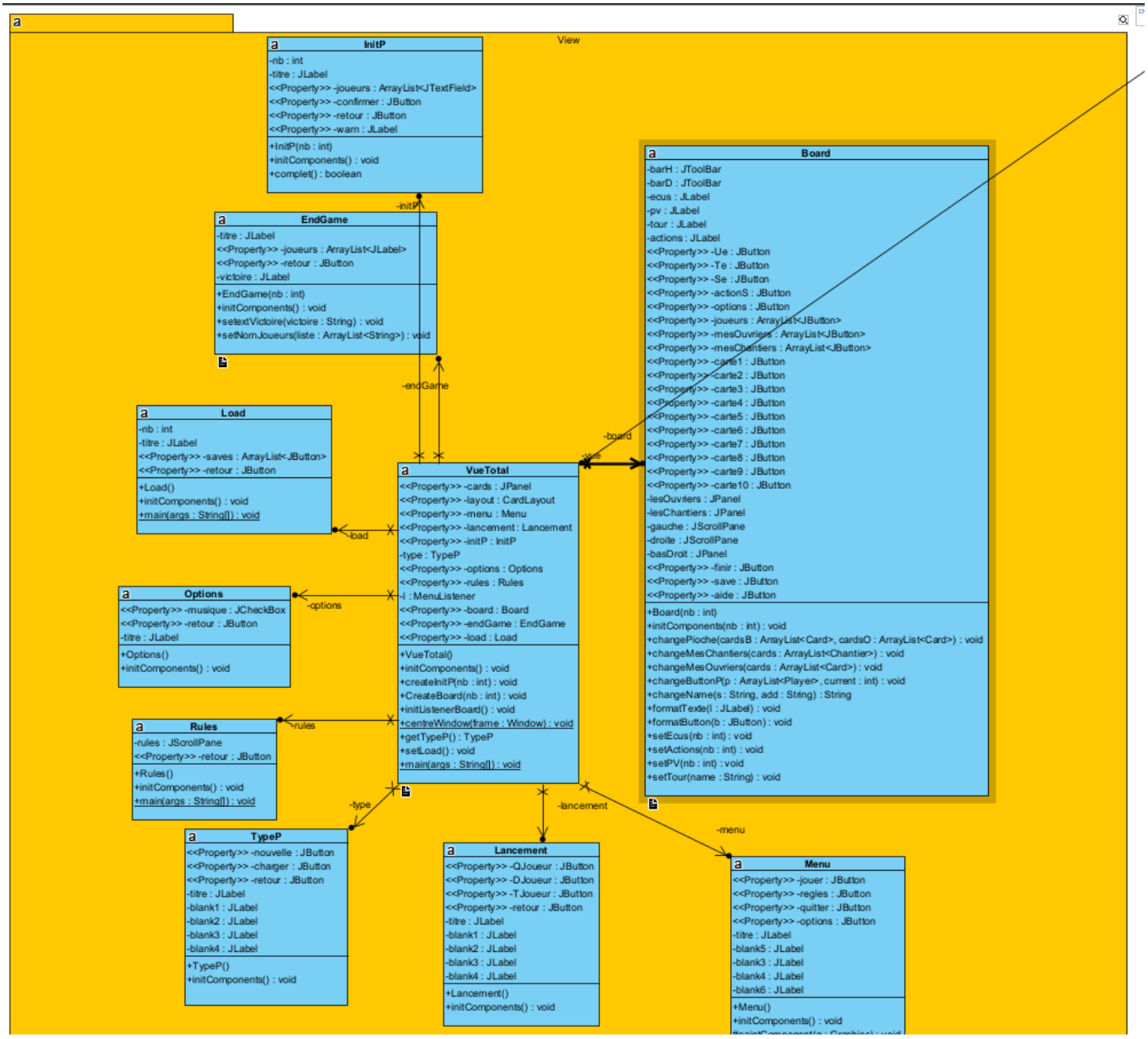
b) Diagramme de conception à la fin de mon projet

Package Game

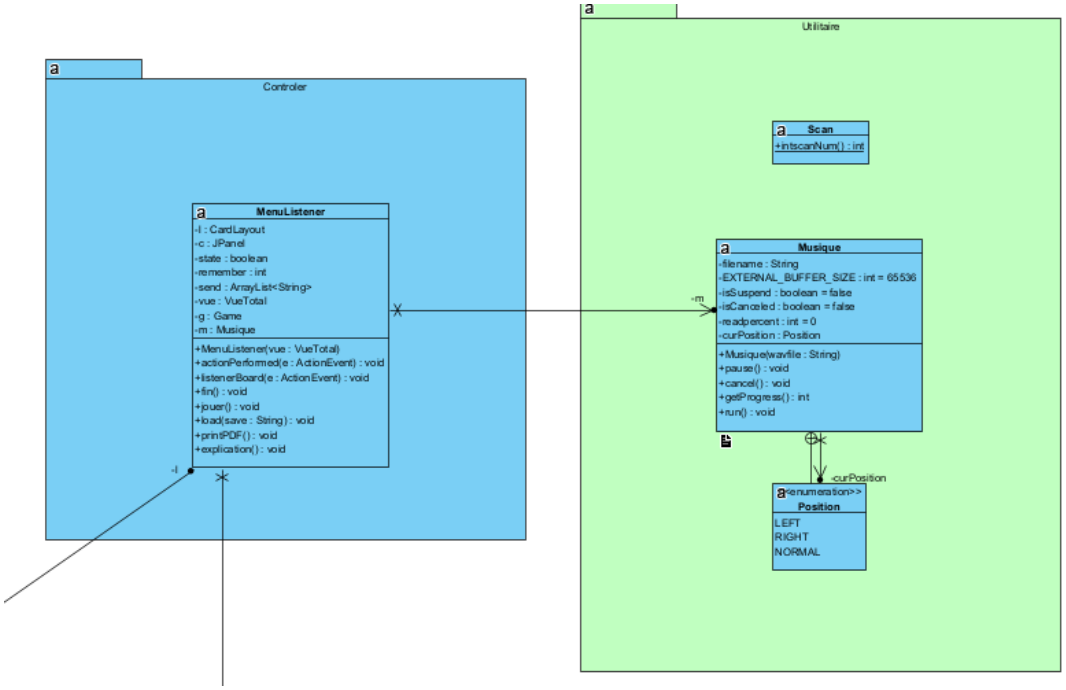


A retrouver en annexe

Package View



Package Control et Utilitaire :


Controler
MenuListener
-l : CardLayout
-c : JPanel
-state : boolean
-remember : int
-send : ArrayList<String>
-vue : VueTotal
-g : Game
-m : Musique
+MenuListener(vue : VueTotal)
+actionPerformed(e : ActionEvent) : void
+listenerBoard(e : ActionEvent) : void
+fin() : void
+jouer() : void
+load(save : String) : void
+printPDF() : void
+explication() : void
-l
Utilitaire
Scan
+intscanNum() : int
Musique
-filename : String
-EXTERNAL_BUFFER_SIZE : int = 65536
-isSuspend : boolean = false
-isCanceled : boolean = false
-readpercent : int = 0
-curPosition : Position
+Musique(wavfile : String)
+pause() : void
+cancel() : void
+getProgress() : int
+run() : void
-m
-curPosition
<<enumeration>>
Position
LEFT
RIGHT
NORMAL

4) Description de la campagne de tests

a) Tests des fonctionnalités obligatoire

Fonctionnalité	Test	Résultat attendu	résultats
Jouer en mode 2, 3 ou 4 joueurs	Sélectionnez « Nouvelle partie » puis essayez « 2 joueurs », « 3 joueurs » puis « 4 joueurs » un à un. Suivre les règles du jeu puis faire tous les coups possibles en entrant des bonnes et mauvaises valeurs.	Une partie se lance ainsi quel'interface de jeu (texte ou graphique). S'il y a une mauvaise saisie. Un message d'erreur apparait et je joueur doit recommencer son action	Ok, parfaitement déroulé
Jouer contre un ordinateur	Sélectionnez de « Nouvelle partie » puis essayez « 2 joueurs », « 3 joueurs » puis « 4 joueurs » un à un. Puis sélectionnez l'option « ordinateur » au moment d'entrer le nom du joueur. Suivre les règles du jeu puis faire tous les coups possibles en entrant des bonnes et mauvaises valeurs.	Une partie se lance ainsi quel'interface de jeu (texte ou graphique). S'il y a une mauvaise saisie. Un message d'erreur apparait et je joueur doit recommencer son action L'ordinateur joue tous les coups qu'il doit jouer.	Ok, parfaitement déroulé

Les différentes actions proposées aujoueur sont toutes implémentées	Lancez une partie avec n'importe quel nombre de joueurs et avec ou sans ordinateur. Vérifier que toutes les actions sont bien disponibles en jeu.	Toutes les actions peuvent êtreutilisé pendant le jeu.	Ok, parfaitement déroulé
Quitter le jeu	Depuis le menu principal, sélectionner l'option « quitter le jeu ». Aussi disponible dans les paramètres en jeu.	Un message d'avertissements'ouvre avec « voulez-vous vraiment quitter le jeu ? » Si « oui » est sélectionner alors le jeu se ferme. Sinon on retourne à l'endroit précédent.	Ok, Par contre le message d'erreur n'a pas été implémenté.
Possibilité d'enregistrer une partie	Commencer une nouvelle partie. Cliquer sur le bouton « enregistrer la partie » en haut de l'écran ou	Un fichier contenant les informations de la partie doit être créé.	Ok, parfaitement déroulé

b) Tests des fonctionnalités optionnels

Fonctionnalité	Test	Résultat attendu	Résultats réels
Implémenté plusieurs niveaux de difficulté pour l'ordinateur.	Sélectionnez de « Nouvelle partie » puis essayez « 2 joueurs », « 3 joueurs » ou « 4 joueurs ». Puis sélectionnez le niveau de l'ordinateur. Joueur une partie dans chaque niveau de difficulté.	Le joueur doit ressentir une variation significative du niveau desordinateurs au cours des différentes parties.	Non implémenté
Menu options fonctionnel	Testez le bon fonctionnement des boutons « son » et « musique ». Pour cela veuillez les activer et désactiver. Pour la version textuelle changez le caractère de remplissage.	Les options doivent fonctionner demanière immédiate.	Ok, parfaitement déroulé Seulement le bouton musique
Menu expliquant les règles	Dans le menu principal sélectionnez le bouton « règles du jeu » ou faites l'équivalent pour la version textuel.	Une page expliquant les règles dujeu doit s'afficher de manière claire.	Ok, parfaitement déroulé Le bouton ouvre un pdf clair

c) Résultats de la campagne JUnit

A la fin de l'implémentation d'une classe, une classe de test utilisant JUnit a été implémentée afin de tester les différentes méthodes de la classe (lorsque celle-ci le permet) de manière automatisée.

Ces tests ont été réalisés via le logiciel ANT, ce qui donne ce résultat :

```
test-compile:
    [javac]  Compiling 6 sources to /users/Yvan/Documents/Semestre2/M2107/code/src/test

test:
    [junit] Running test.BuldingTest
    [junit] Tests run: 8,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.014sec
    [junit] Running test.GameTest
    [junit] Tests run: 12,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.031sec
    [junit] Running test.HumanPlayerTest
    [junit] Tests run: 1,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.005sec
    [junit] Running test.MachineTest
    [junit] Tests run: 10,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.025sec
    [junit] Running test.PlayerTest
    [junit] Tests run: 3,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.008sec
    [junit] Running test.WorkerTest
    [junit] Tests run: 8,Failures: 0, Errors: 0,Skipped 0, Time elapsed 0.002sec
```

5) Etat d'avancement réel du développement comparé avec les objectifs annoncés dans le cahier des charges

Nous allons comparer les objectifs annoncés dans le cahier des charges et les comparer avec le résultat final du projet :

a) Fonctionnalités de base

- **Version du jeu en version textuelle** → cette version est bien disponible et est fonctionnelle
- **Version du jeu en version graphique** → cette version est bien disponible et est fonctionnelle
- **Tous les types de cartes sont implémenté** → oui
- **Mise en place de la situation de départ** → oui
- **Les différentes actions proposées à un joueur doivent toutes être implémentées. 3 actions possibles** →oui toutes les actions sont bien implémentées.
- **Possibilité de jouer à 2, 3 ou 4 joueurs humain ou ordinateur** →oui c'est fonctionnel en version graphique mais aussi textuelle.
- **Possibilité d'enregistrer une partie en cours pour pouvoir la reprendre plus tard.** → oui c'est fonctionnel en version graphique mais aussi textuelle.
- **Implémentation en Java, avec un code source documenté et testé**. → oui la javadoc a été réaliser
- **Gérer la victoire d'un joueur et la fin de la partie** → oui en version textuelle et en version graphique.

b) Fonctionnalités optionnelles (en fonction du temps restant)

- **Possibilité d'avoir un ordinateur avec plusieurs niveaux de difficultés.** → Pas implémenté
- **Disposer d'un volet d'options (thème de couleur, son, …)** → en petite partie. Seul la musique a été implémenté.
- **Menus expliquant les règles**. → oui, ouverture d'un pdf avec les règles
- **Choix du nom des joueurs en début de partie.** → oui fonctionne parfaitement
- **Jouer en ligne sur un réseau local** → pas implémenté

On peut donc en conclure que peu des fonctionnalités optionnelles ont été implémentés mais les versions de base sont toutes implémentés et le jeu est jouable en version graphique et textuel.

c) Avenir du projet

Le projet dans sa version actuelle sera disponible en open source sur GitHub sur mon profil (https://github.com/YvanBocande) et pourra être mis à jour par toute personne souhaitant participer au projet ou par moi-même.

6) Synthèse des difficultés rencontrés et des solutions apportées

La plus grosse difficulté à laquelle j'ai été confronté est la mise en place d'une interface swing qui soit moderne et fonctionnel. En effet, il m'a fallu me former sur le sujet afin de comprendre comment mettre en place de Boutons et des textes moderne qui soient fonctionnels. De plus, la mise en place des layout a été très difficile pour moi car je ne trouvais pas cela intuitif.
J'ai contré ce manque de connaissances sur swing par ma formation sur des sites qui expliquaient comment mettre en place tels ou tels élément. J'ai joint cette formation supplémentaire à la formation initiale que nous avons reçu à l'iut.

Une difficulté que j'ai eu est qu'au début du projet j'avais inversé les valeurs de bois et de Tuile lors de l'initialisation. Donc au moment de la mise en place de l'interface graphique il y avait en problème entre les graphismes de la carte et ce qu'elle produisait vraiment. J'ai eu beaucoup de mal à trouver la source du bug et cela m'a pris beaucoup de temps.
Pour faire face à ce problème, j'ai remonté toutes les méthodes et fichier qui permettait l'initialisation et l'affichage de mes cartes. J'ai fini par remonter à la source de la création de mes cartes où se situait le souci.

De plus j'ai eu beaucoup de mal à implémenté la sauvegarde en partie graphique. En effet, lorsque que je chargeais une partie graphique l'interface se bloquait sur le menu et ne voulait plus bouger. Je ne comprenais pas ce bug.
J'ai résolu ce bug en testant beaucoup d'autres manières différentes de charger le jeu et (par chance) une de ces manières à fonctionné.

Aussi, la mise en place des tests JUnit a été long et fastidieux car je ne connaissais absolument pas JUnit.
J'ai résolu cette difficulté en me formant avec des vidéos youtube et en demandant l'aide de mes camarades.

Pour finir, la plus grande difficulté que j'ai rencontrée est le manque de temps. En effet, le temps alloué à chaque alloué à chaque tâche était très court et ne me permettait pas de perdre du temps ou de prendre du retard. En effet, chaque bug qui me faisais perdre du temps remettait en cause le fait que je puisse rendre le projet en temps en temps et en heure.
Pour faire face à cela, j'ai essayé d'être le pieux organisé possible et de réduire le nombre de tests quand j'avais confiance en ma partie de code.

7) Bilan personnel du travail réalisé

Premièrement je suis globalement content du rendu de projet que j'ai réalisé car j'ai réussi à fournir à mon client toutes les fonctionnalités qu'il attendait.
Ce projet m'a appris à gérer un planning. Il m'a aussi permis d'avoir une expérience dans un projet de programmation d'un logiciel informatique qui se rapproche fortement de la réalité. Ce projet a aussi amélioré ma capacité d'analyse dans la réalisation d'une solution informatique.

J'ai cependant deux regrets sur ce projet. Le premier regret est de ne pas avoir eu plus de temps pour rendre un produit qui sois plus finalisé. En effet, j'aurais aimé avoir plus de temps à consacré pour avoir une version graphique qui sois plus esthétique et non pas seulement fonctionnelle. De plus ce temps supplémentaire m'aurait permis de réaliser plus de tests et donc de m'assurer du fait qu'il soit infaillible.
Mon second regret t se porte sur le fait que le projet n'ait pas été réalisé en groupe ce qui aurait permis de réellement mettre à l'épreuve la cohésion de groupe et le travail d'équipe avec lequel nous devrons être à l'aise dans l'avenir.