



TIRLEMONT Kierian

Département INFORMATIQUE

1<sup>ère</sup> année – Groupe D

Année 2019-2020

# Rapport final du projet de programmation

Numérisation du jeu « Zen l'Initié »

Destinataires

IUT de Vannes

M. Lefèvre Sébastien



## Table des matières

Table des matières .....	2
I. Mise à jour des diagrammes présentés dans le rapport d'analyse.....	3
a. La partie Match .....	3
b. La partie Player.....	5
c. La partie Game .....	7
d. La partie affichage .....	10
e. La partie Configure .....	12
f. Vue générale du model .....	13
II. Description des choix techniques et algorithmiques .....	15
III. Diagramme de classes obtenu par retro-conception.....	20
a. Vue d'ensemble.....	20
b. Le package gameElement.....	22
c. Le package display .....	23
d. Le package utility .....	24
IV. Description de la campagne de tests effectuée .....	24
a. Fonctionnalité de base .....	24
c. Fonctionnalité évoluée.....	28
V. Etat d'avancement du développement.....	30
a. Fonctionnalité de base .....	30
b. Demande du client .....	33
c. Evolutions .....	34
VI. Synthèse des difficultés rencontrées et des solutions apportées.....	38
VII. Bilan personnel du travail réalisé .....	39

# I. Mise à jour des diagrammes présentés dans le rapport d'analyse

## a. La partie Match

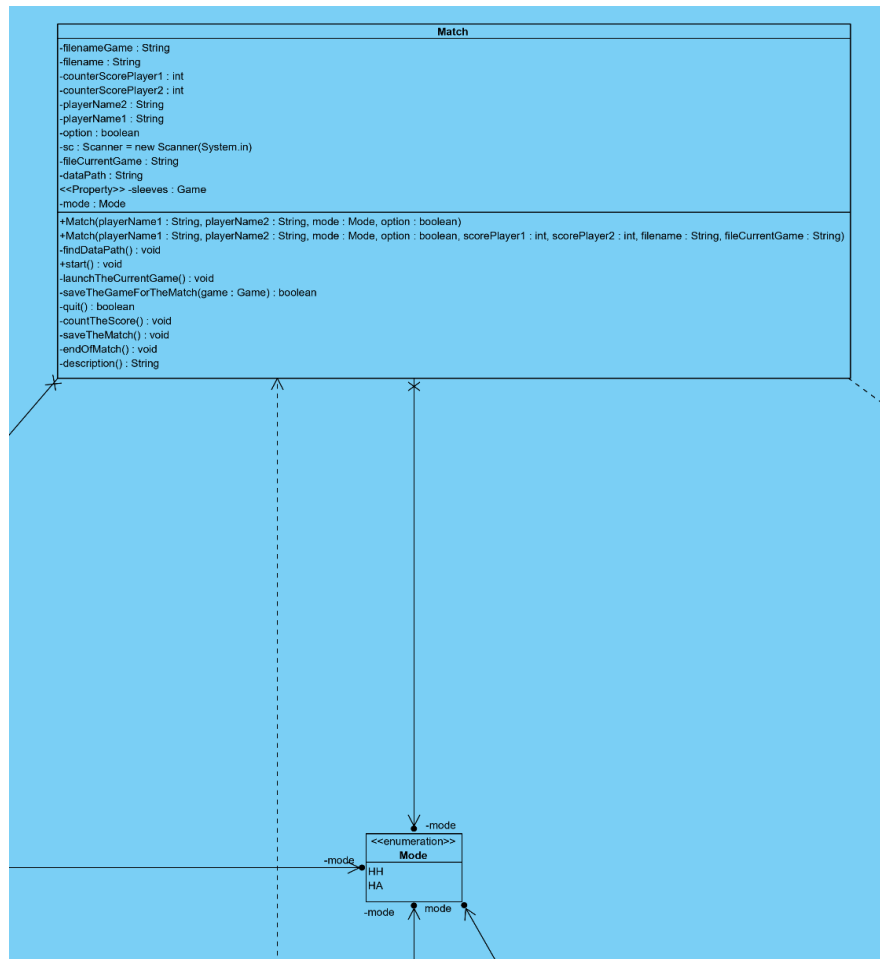


Figure : Mise à jour du diagramme classes de conception.

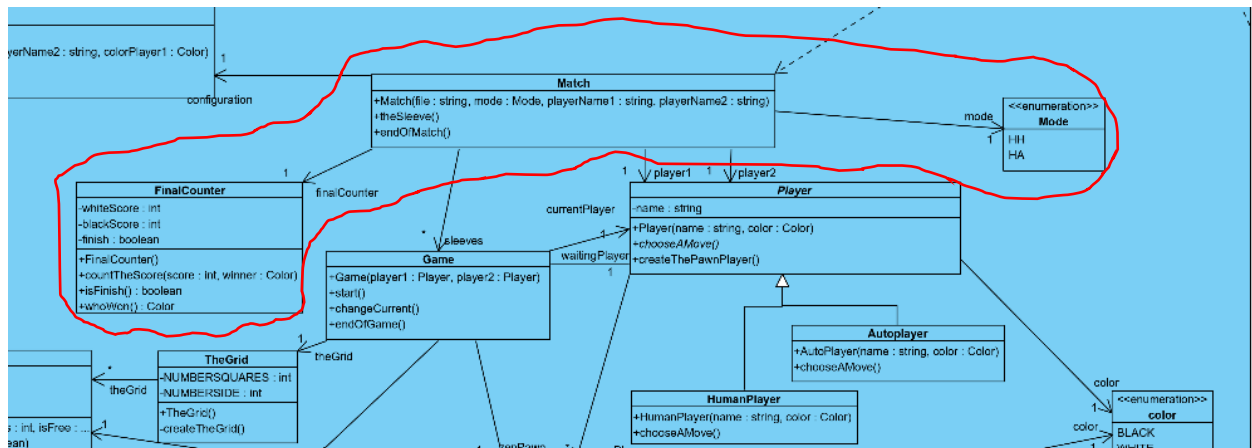


Figure : Diagramme de classes de conception du cahier de d'analyse et de conception.

Nous remarquons que la classe « FinalCounter » a été supprimée. La classe en elle-même n'a pas été écrite, cependant, ces méthodes se retrouvent dans la classe « Match ». Par exemple, le score de chaque joueur est un attribut de la classe « Match », on y retrouve aussi les méthodes « endOfMatch » ou encore « countTheScore ». En effet, nous avons déplacé les attributs et les méthodes de la classe « FinalCounter » dans la classe « Match » puisque c'était bien plus simple de gérer dans la classe « Match », ça a évité de transférer des valeurs comme le score, ou le score de la partie avec son gagnant entre ces deux classes.

On remarque qu'il y a actuellement une méthode « saveTheGame », en effet, je n'ai pas conservé la partie « Configure » mais j'y reviendrai un peu plus tard.

Enfin, j'ai gardé l'utilisation du mode afin de connaître le nombre de joueur humain.

## b. La partie Player

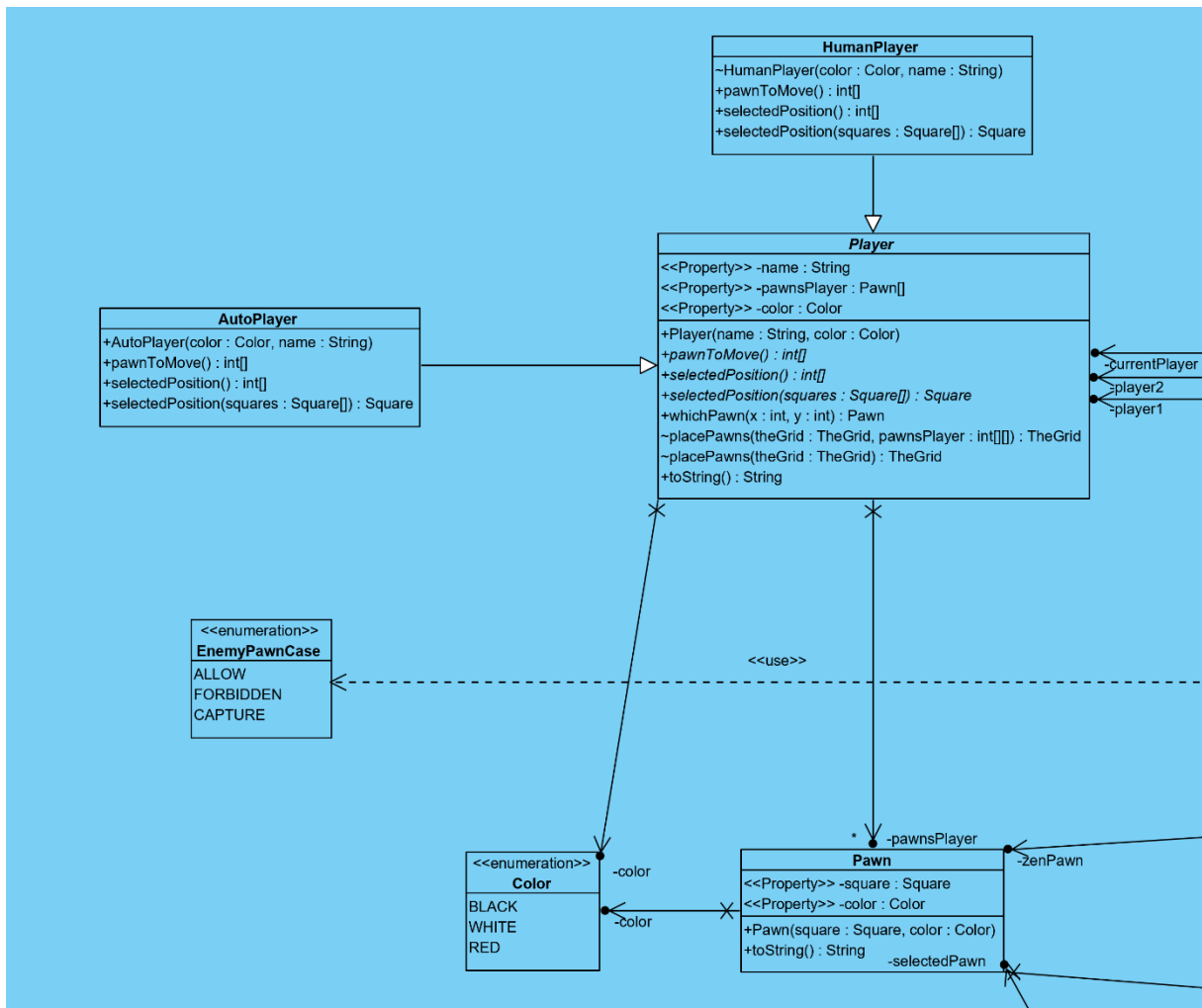


Figure : Mise à jour du diagramme classes de conception.

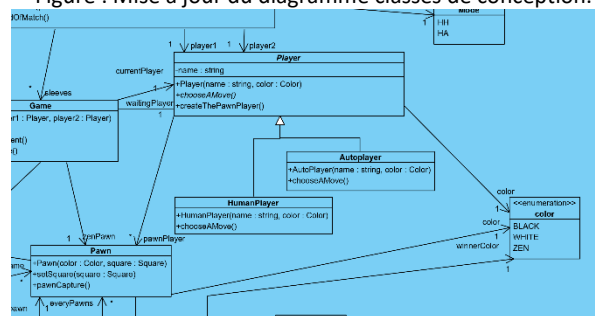


Figure : Diagramme de classes de conception du cahier de d'analyse et de conception.

Cette partie a assez peu changé, nous avons toujours une classe abstraite « Player » qui est héritée par deux classes : « Humanplayer » et « AutoPlayer ». Dans la classe « Player », nous avons juste rajouté deux méthodes qui permettent de placer les pions en début de partie et au lieu d'utiliser une méthode

pour le déplacement d'un pion, il y en a deux. Ces trois méthodes permettent dans un premier temps de choisir le pion puis de choisir sa position finale au lieu de faire ça dans une seule méthode. Les attributs restent les mêmes avec un tableau de pions en attribut de classe « Player », la couleur des pions et le nom du joueur.

### c. La partie Game

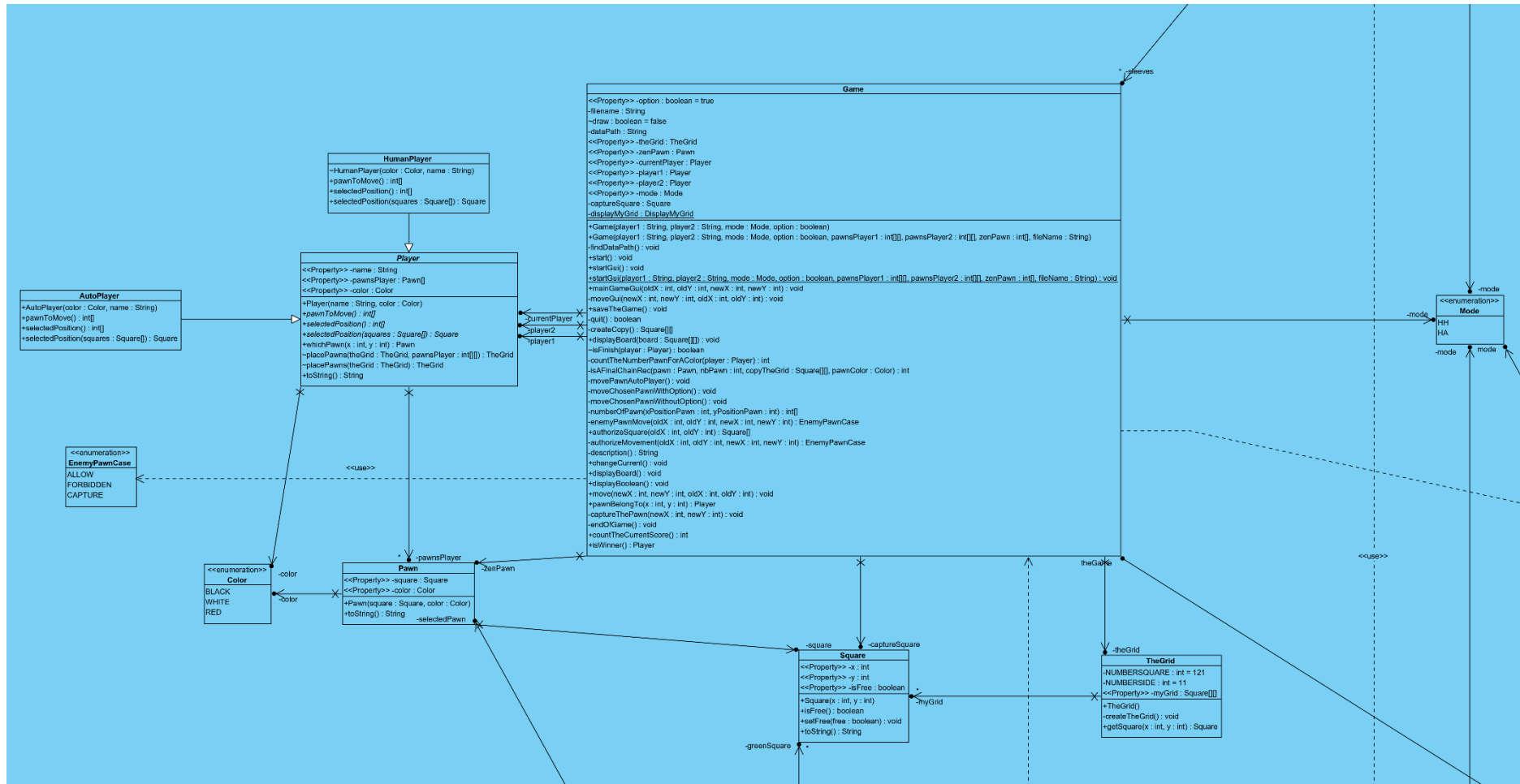


Figure : Mise à jour du diagramme classes de conception.

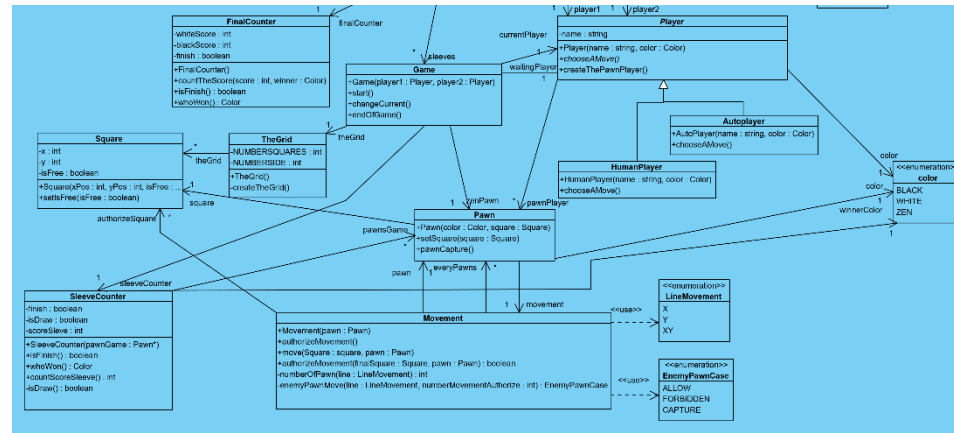


Figure : Diagramme de classes de conception du cahier de d'analyse et de conception.

Cette partie a beaucoup changé dans son aspect même si le fonctionnement prévu et le fonctionnement réel sont globalement la même chose. Premièrement, la classe « Game » est bien plus fournie en méthode à cause de deux aspects : les méthodes à laquelle je n'avais pas pensé comme la méthode « pawnBelongTo » qui permet de trouver à quel objet « Player » il est lié. Et il y a les méthodes qui devaient être dans d'autres classes, mais ces classes étant inutiles, leurs méthodes se sont retrouvées dans la classe « Game ». On compte toutes les méthodes de la classe « Movement » et de la classe « SleeveCounter ». En effet, il était bien plus compliqué de transférer des valeurs comme la position des pions en temps réel avec deux autres classes que d'écrire les méthodes dans la classe qui centralise déjà toutes les informations. Pour conclure, le fonctionnement est le même puisque j'utilise les mêmes méthodes mais leur emplacement final est la classe « Game » au lieu de « Movement » ou « SleeveCounter » pour éviter de devoir transférer des valeurs entre ces classes.



Pour la partie création de la grille, on remarque qu'elle est identique, en effet, c'était une bonne idée que j'ai laissée et qui n'a pas évolué. Il y a juste le positionnement des pions sur la grille qui sont passés dans la classe « Player ».

Au niveau des objets « Player » dans la classe « Game », nous avons changé le fonctionnement. Avant nous devions à chaque tour intervertir le nom des variables avec leur adresse. Tout simplement, « currentPlayer » devenait « waitingPlayer » et inversement. Ce qui n'était pas très pratique. C'est pourquoi, maintenant nous avons deux objets joueurs liés pour toujours à leur adresse. Cependant, il y a un troisième objet « Player » qui va pointer un tour sur le joueur un puis au tour suivant vers le joueur deux et ainsi de suite.

Enfin, nous n'avons pas gardé l'énumération « LineMovement » puisqu'à la fin, nous nous sommes rendu compte que nous ne l'utilisons pas.



Figure : Mise à jour du diagramme classes de conception.

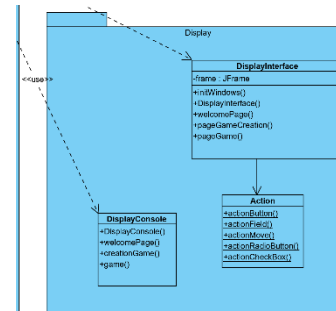


Figure : Diagramme de classes de conception du cahier de d'analyse et de conception.

Au moment de la réalisation des diagrammes de classes, je n'avais aucune idée de comment gérer l'affichage avec un menu et une interface. Donc je suis reparti de zéro et je suis arrivé à ce diagramme : une partie affichage en mode console et une partie affichage en mode graphique, toutes deux liées à des actions, parfois communes, parfois non communes. Dans la partie graphique une classe est liée à une page tandis que dans la partie console, il y a une classe principale avec toutes les pages. Seule la page de tutoriel est en dehors.



## f. Vue générale du model

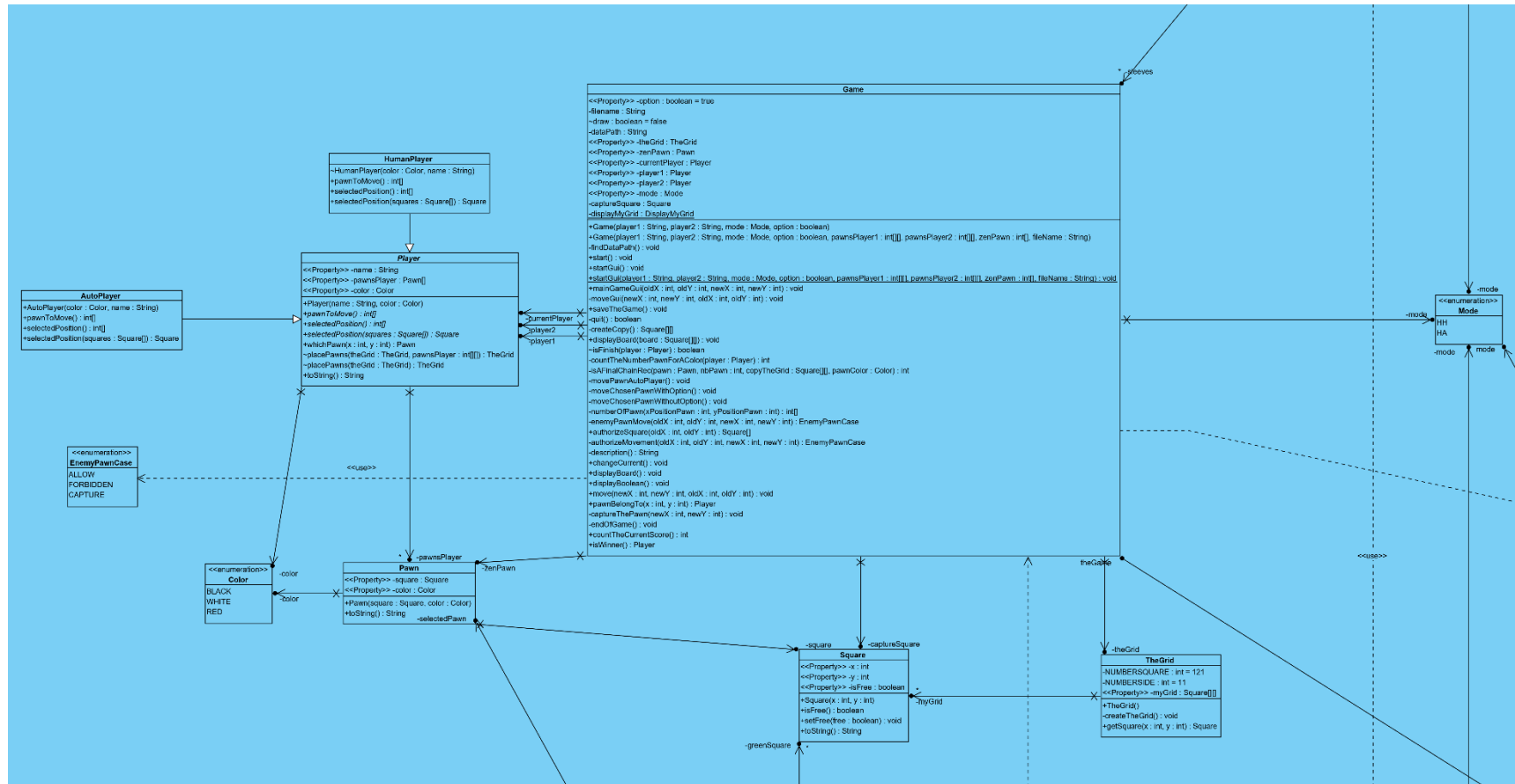


Figure : Mise à jour vue générale du model.

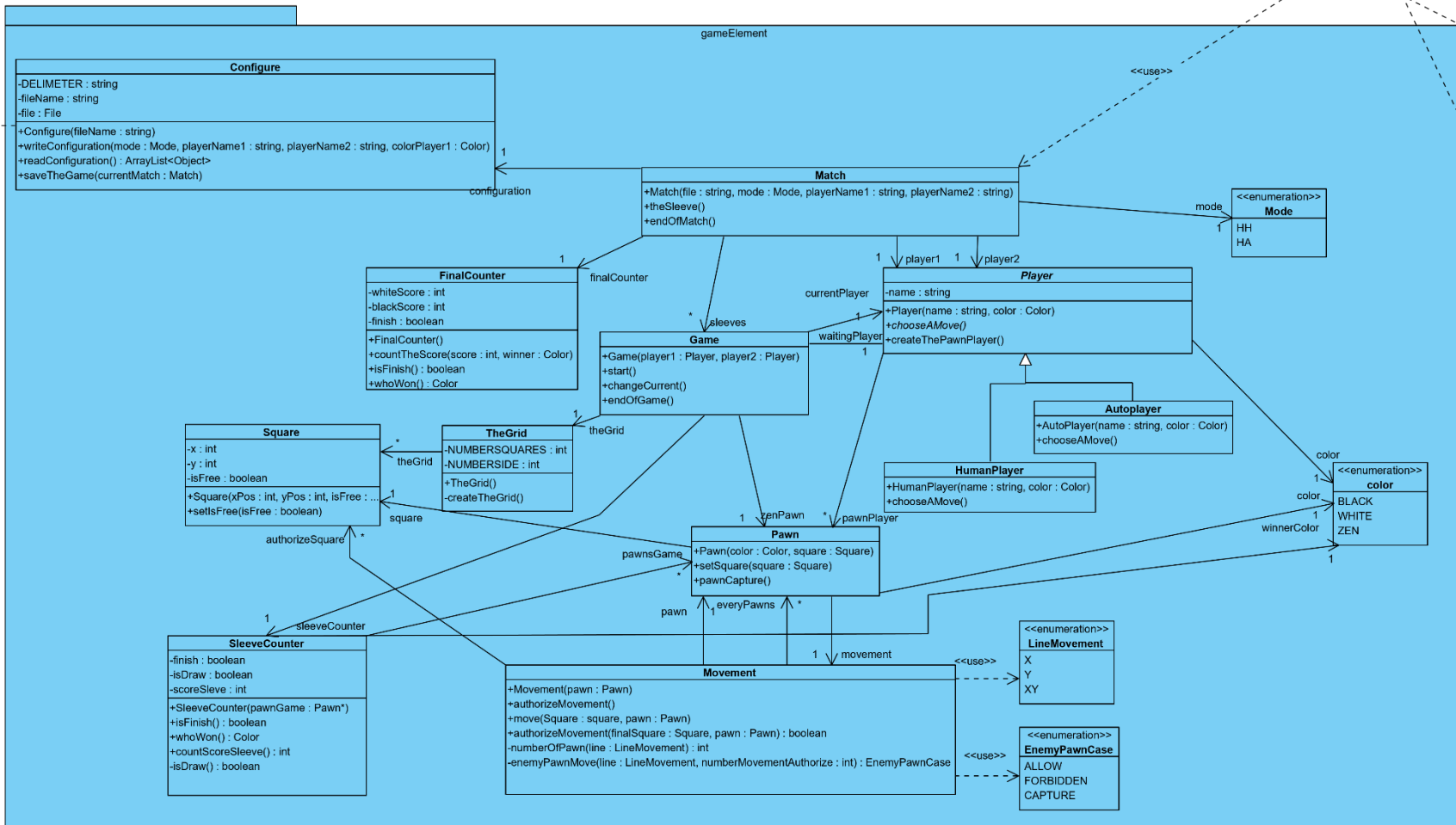


Figure : Diagramme de classes de conception du cahier de d'analyse et de conception.

Pour conclure, malgré ces changements, le fonctionnement général reste le même, nous avons juste déplacé des méthodes dans d'autres classes afin d'éviter le transit de beaucoup de variables entre classes.

## II. Description des choix techniques et algorithmiques

Nous disions lors du cahier des charges ce que devait faire le programme :

*Si on quitte la présentation quelques secondes, et que l'on regarde cela d'un point de vue de la programmation, il faudra laisser la possibilité aux joueurs de jouer le zen en tant qu'ami ou ennemi. De plus, il faudra générer un tableau de 11 cases par 11 cases avec 25 pions avec 12 pions amis et 12 pions ennemis.*

Afin de créer la grille, nous avons écrit une classe « Square », cette classe est définie par un entier x, un entier y et un booléen qui indique si l'objet est occupé. Ainsi, nous avons généré un tableau de deux dimensions de Square. Il s'agira donc de la grille de jeu, elle sera de type TheGrid, une classe contenant le tableau de Square et la taille du tableau.

Afin de créer les pions, nous avons écrit une classe « Pawn » qui représentera un pion. Un pion est défini par sa couleur et par une case sur la grille c'est à dire un objet Square. Les douze pions d'un joueur sont générés dans la classe « Player », ainsi que leur positionnement sur la grille. Je vais appeler la méthode « placeThePawns(theGrid : TheGrid) » et nous allons venir modifier la grille en changeant l'état des booléens à faux quand un pion est sur la case. Le « zen » est un objet de la classe « Pawn » et est un attribut de la « Game » et je viens le positionner sur la grille dans le constructeur de la classe « Game ».

J'affiche la grille dans la classe « Game », cette méthode qui permet l'affichage va regarder case par case l'objet de TheGrid, si la case est occupée alors le code va demander aux deux objets « Player » à qui appartient ce pion, et le programme vérifie évidemment si c'est le zen. Si le pion est introuvable c'est alors une erreur et là, le programme imprime le mot « Error ».

*Le programme devra ainsi pouvoir faire se déplacer un pion dans toutes ces directions, il devra compter le nombre de pion sur la ligne de déplacement pour savoir le nombre de mouvement possible. Il devra aussi repérer les pions ennemis et refuser le déplacement s'il y a un pion ennemi sur le chemin mais s'il y a un pion ennemi sur la case d'arrivée, il doit comprendre que le déplacement est possible et que le pion ennemi est mis hors-jeu.*

Pour déplacer les pions, j'avais prévu lors du cahier d'analyse et de conception, de créer une classe « Mouvement » mais ça complexifié la programmation, alors le déplacement d'un pion est possible grâce aux méthodes de la classe « Game ». Dans un premier temps, le programme va demander au joueur quel pion veut-il déplacer. A partir de ce moment, le programme va venir compter le nombre de pion sur les lignes. Nous avons compris qu'il y avait quatre lignes de déplacement à gérer.

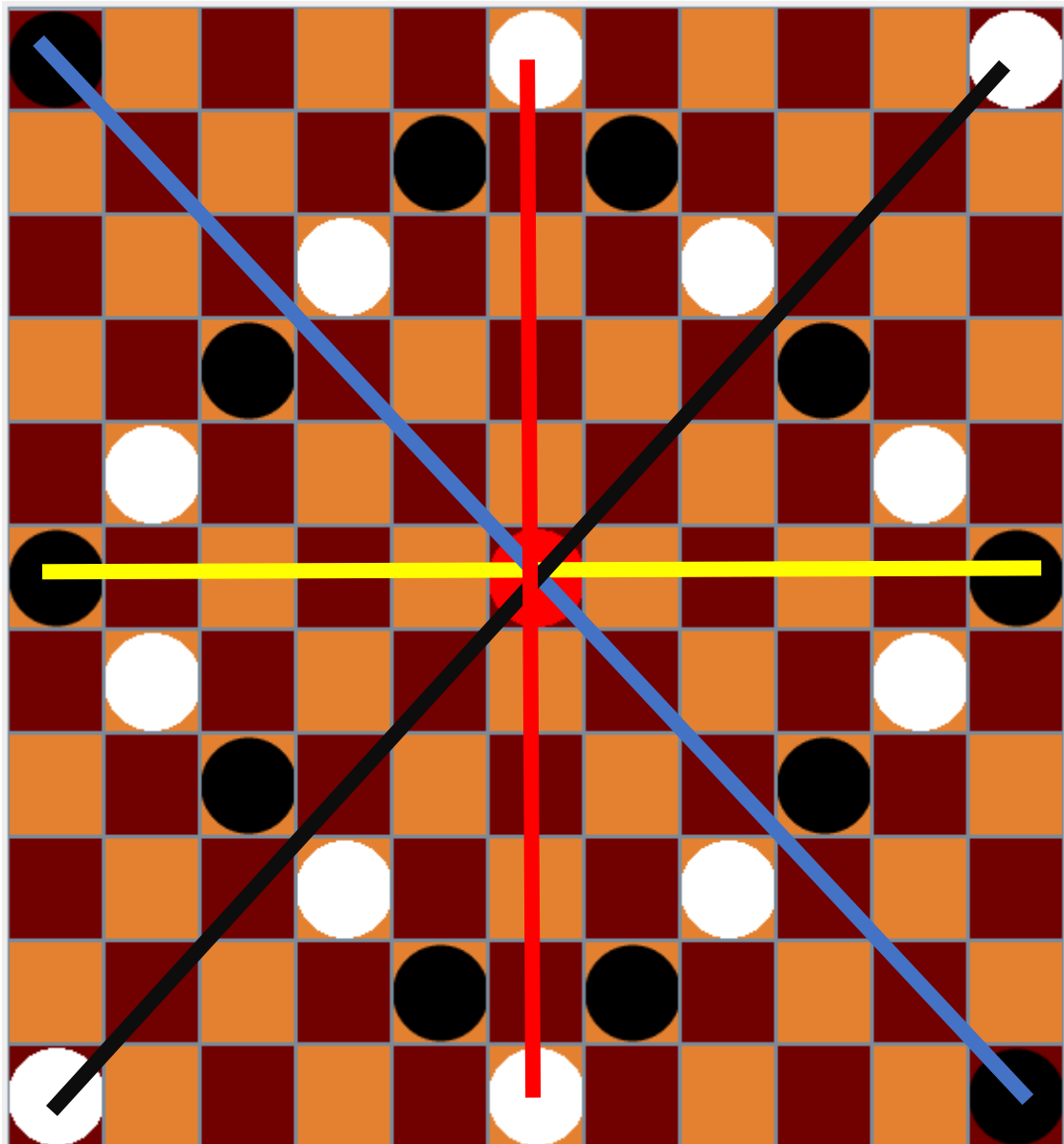


Figure : Visualisation des quatre déplacements possibles.

Dans notre programme, nous avons appelé cela : X, Y, XYBottom (flèche bleu), XYTop (flèche noir). Pour compter le nombre de pions sur une ligne de déplacement, le programme va chercher dans les cas à aller le plus possible vers la gauche. Une fois arrivée sur la case la plus à gauche ( $x = 0$ ), le



programme va aller vers la droite et dès qu'il va rencontrer un pion, il va incrémenter la variable. Une fois arrivée dans la case la plus proche de  $x = 10$ , on sait combien il y a de pions sur la ligne de déplacement.

A partir du moment où le programme connaît le nombre de pion par ligne de déplacement, on va chercher les cases finales possibles, et on va retirer dans un premier temps celles qui sortent du tableau. Et dans un second temps, on va regarder pour chaque possible restant si durant le déplacement, le pion va être gêné par des pions ennemis, entraînant l'impossibilité du mouvement ou la capture d'un pion ennemi. Ainsi le programme en déduit les cases possibles et compare ces cases possibles avec la case que l'utilisateur a rentré.

Le programme devra ainsi repérer les chaînes de pions. Et aussi, incorporer la possibilité de capturer un pion ennemi.

*Le programme devra ainsi repérer les chaînes de pions. Et aussi, incorporer la possibilité de capturer un pion ennemi.*

Afin de repérer les chaînes de pions, il nous faut trois méthodes. La principale méthode est la méthode qui permet le comptage de la chaîne principale. Pour écrire cette méthode, nous avons utilisé la récursivité. Le programme est assez simple malgré ses cinq cents lignes de vérification. Pour un pion donné, le programme rentre dans une boucle, la condition de sortie est de ne plus trouver de pion autour. Donc le programme va regarder chaque position autour s'il trouve un pion alors il refait un tour de boucle et il va se demander s'il y a d'autres pion autour autres que ceux déjà trouvés. Une fois qu'il n'y plus de pion, il sort de la boucle et pour chaque pion trouvé dans la boucle, le programme va rappeler cette méthode en incrémentant le compteur en paramètre. Finalement le compteur nous donne le nombre de pion sur cette chaîne. L'avantage d'une méthode récursive est que si le programme prend un mauvais chemin, il est capable de revenir en arrière et prendre une autre route afin de compter tous les pions de la chaîne. Nous avons une autre méthode qui compte le nombre de pion pour un joueur donné. Et une dernière méthode qui compare les résultats retournés par les deux méthodes, si la valeur retournée par ces deux méthodes sont égales alors la partie est terminée sinon la partie continue.

Pour permettre la capture, nous avons créé un attribut de classe de « Game ». Cet attribut a pour type « Square » et ses attributs sont égaux à  $x = -1$  ;  $y = -1$ . Vous vous dites sûrement que cette case n'existe pas dans la grille, et vous avez raison. Cette case sert de prison pour les pions capturés. Donc quand

un pion est capturé, son attribut Square change pour prendre comme attribut cette « case prison ». Evidemment sur la grille le booléen qui dit que la case est occupée change d'état.

*Le programme devra être capable de percevoir ses subtilités entre un match nul et un match avec un gagnant.*

Afin de vérifier que le match est nul, le programme compare le gagnant au joueur ayant effectué le dernier coup, si le joueur ayant fait le dernier coup est le gagnant alors la partie a bien un gagnant, cependant si le gagnant n'est pas le joueur ayant fait le dernier coup, alors la partie est nul. En effet, nous avons remarqué que dans tous les cas où le joueur ayant fait le dernier coup est différent du joueur gagnant, alors la partie était nul car il a entraîné la victoire de son adversaire.

*Le programme devra être capable de lancer un match en sept points. On comprend alors que le programme devra être capable de compter et stocker les points. Pour cela, il devra distinguer la chaîne principale du perdant de ses pions isolés. Il devra ainsi continuer la partie tant qu'un des joueurs n'a pas atteint 7 points.*

(Nous pensons au début que cette règle était une règle pour tous les joueurs.)

Afin de lancer un match en sept points, j'ai tout simplement créé une classe « Match » ayant pour attribut une collection d'objet « Game ». En effet, la méthode « start » de Match va permettre de créer autant d'objet Game nécessaire pour que l'un des joueurs atteignent les sept points. Pour compter les points, le score des deux joueurs est un attribut de la classe « Match ».

Pour compter le score, nous avons écrit une méthode dans la classe « Game », qui permet de compter le score de la partie. Tout simplement la méthode, va tout d'abord trouver qu'elle est la plus grande chaîne de pion. Pour cela elle va appeler la méthode récursive dont j'ai parlé un peu plus tôt. Une fois que la taille de la plus grande chaîne est trouvée. Le programme demande pour chaque pion s'il fait partie de cette chaîne principale, en comparant le nombre de pion de la chaîne à partir de ce même pion et la taille de la plus grande chaîne. Si la taille est égale, alors ce pion fait bien parti de cette chaîne maximale sinon ce pion ne fait pas partie de la chaîne principale alors j'incrémente le score car le pion ne fait pas partie de la chaîne principale.

*Il sera aussi possible de jouer à deux ou seul contre l'ordinateur.*

Pour arriver à ce but, ma méthode « start » inclut une condition pour différencier ces deux modes et faire des actions différentes en fonction du mode. De plus, nous avons une classe « HumanPlayer » qui hérite de la classe « Player », qui permet de choisir le mouvement en fonction de ce que l'utilisateur

veut. Et nous avons une classe « AutoPlayer » qui hérite aussi de la classe « Player », qui permet de choisir le mouvement automatique et au hasard. En effet, je n'ai pas eu le temps de gérer plusieurs niveaux de difficultés d'intelligence artificiel.

*Il y aura la possibilité d'enregistrer une partie en cours pour pouvoir la reprendre plus tard.*

Nous allons ici différencier, l'enregistrement d'une simple partie et d'un match en sept points.

Tout d'abord, nous allons nous intéresser à une simple partie. Pour enregistrer une partie, nous avons écrit une méthode dans la classe « Game » qui permet d'écrire dans un fichier texte les éléments importants de la partie telle que la position des pions, le nom des joueurs, le mode, et la sélection de l'option. Pour cela j'utilise une classe écrite il y a quelques semaines « RWFile ». Cette classe comprend uniquement des méthodes statiques qui permettent de faire des actions sur des fichiers comme créer ou supprimer un fichier, le renommer, etc. Le nom du fichier est normalisé : « saved\_game0.txt », il faut imaginer que le chiffre est incrémenté à chaque fichier de sauvegarde. Ensuite, pour afficher à l'utilisateur les fichiers de sauvegarde qu'il peut lancer, le programme est capable de juste afficher les fichiers ayant pour nom « saved\_gameN.txt » (N étant un chiffre). Par la suite, l'utilisateur peut soit continuer de cette partie ou la supprimer. Pour la continuer, mon actionneur va lire les configurations, et créer un objet « Game » avec ces informations. Une fois l'objet « game » créé, on appelle la méthode « start » qui continuera la partie.

Ensuite, nous avons l'enregistrement d'un match en sept qui est un peu plus complexe. Tout d'abord, il est possible d'enregistrer un match entre deux parties, c'est assez simple, on écrit dans un fichier texte le score de chaque joueur, le nom des joueurs, le mode, et la sélection de l'option. De la même manière, on va afficher uniquement les matchs avec ce nom : « saved\_match0.txt ». Et de la même manière, le match va être chargé dans les actionneurs. Mais il y a une subtilité, en effet, un joueur pourrait décider d'arrêter le match en pleine partie. Pour résoudre ce problème, lorsque l'utilisateur quitte la partie en cours, un fichier de sauvegarde est créé, mais le programme va venir le renommer en saved\_game\_for\_match0.txt comme ceci l'utilisateur ne pourra pas supprimer ce fichier depuis le logiciel et faire planter l'application. Le nom de la partie va être écrit dans le fichier texte de sauvegarde du match. Ainsi, s'il y a le nom d'un fichier d'une partie alors la classe « Match » va appeler le constructeur de « game » qui permet de reprendre une partie en cours puis les autres objet « game » normalement.

*C'est pourquoi, j'ai eu l'idée de créer un didacticiel où le débutant pourrait être guidé dans une première partie afin de découvrir les règles de ce jeu de stratégie.*

Pour créer ce didacticiel, nous avons ajouté deux classes. Une classe pour créer le tutoriel en ligne de commande et un tutoriel pour créer le tutoriel en mode graphique. Je vais tout simplement afficher du texte permettant à l'utilisateur de comprendre les règles et en mode console le programme fait pratiquer l'utilisateur.

*C'est pourquoi, pour les joueurs débutants, quand on clique sur une pièce, il est possible de distinguer les cases sur lesquelles il peut se déplacer.*

Le programme connaît déjà les cases possibles grâce à la méthode « authorizeSquare » de Game. Donc, il suffit de demander à l'utilisateur s'il veut activer cette option. En fonction de sa réponse, un booléen va illustrer son choix. A l'aide d'une condition, le programme affiche ou non les cases possibles.

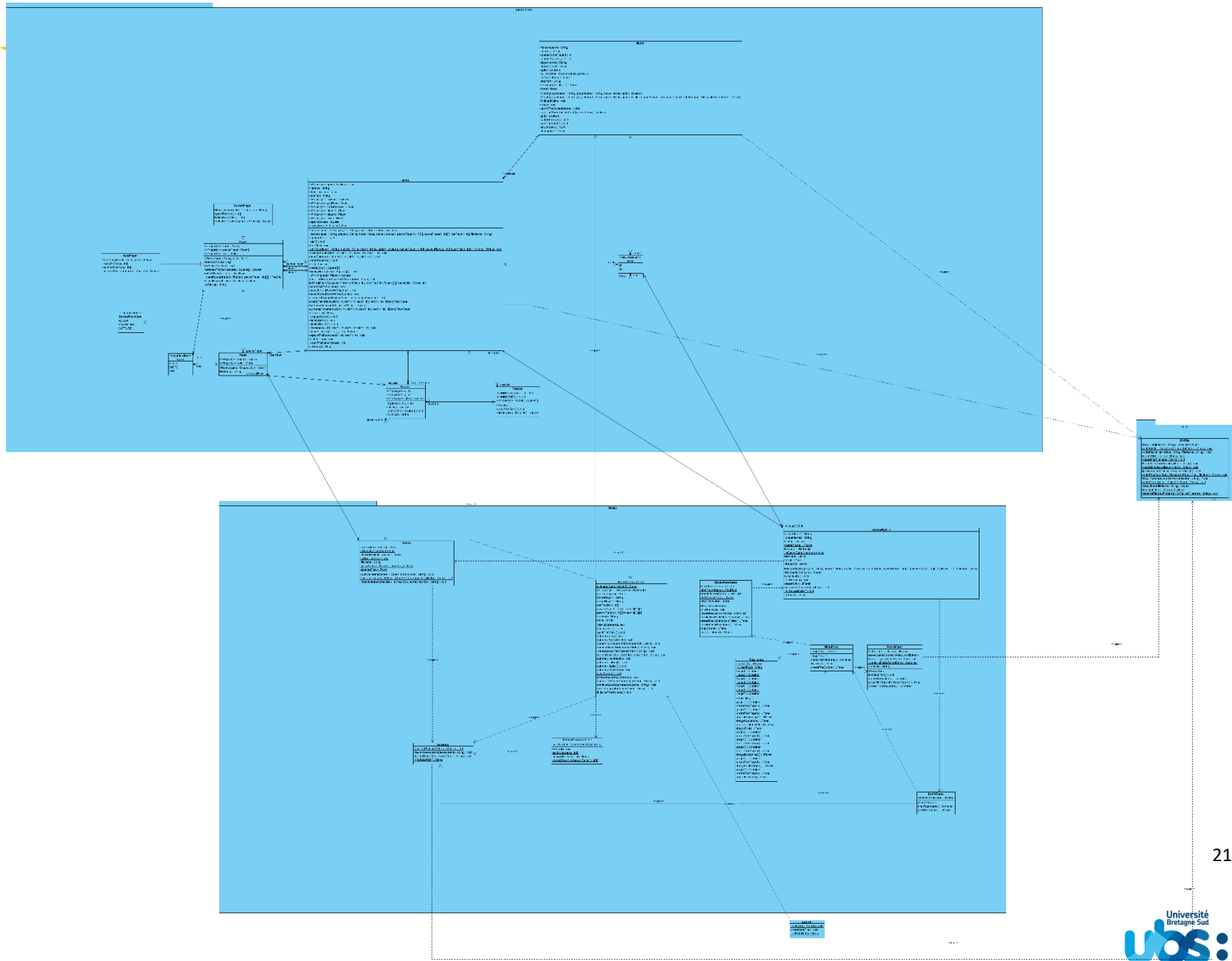
*Le programme devra avoir une version graphique.*

Pour afficher la grille en version graphique, j'ai créé un tableau de JButton, dont je change la couleur de fond en fonction de sa position et dont je change l'image à l'aide de l'image d'un pion en png.

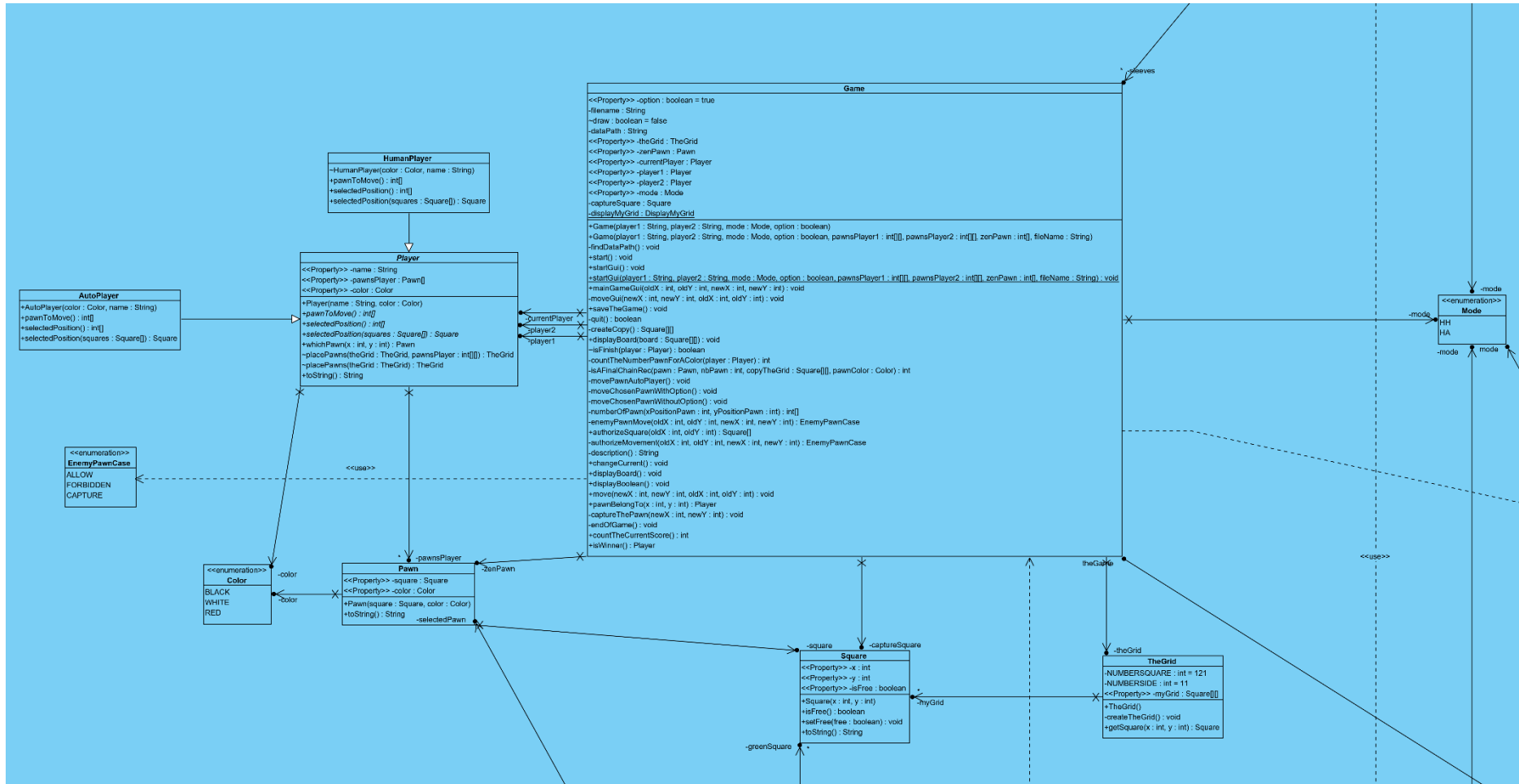
Pour conclure, le programme est bien avancé et les fonctionnalités de base ont été réalisées, et une partie des fonctionnalités évolutives ont été réalisés. Le seul problème est que l'application n'a pas été testé par un beta testeur qui permettrait d'avoir un avis extérieur et pourrait faire un rapport sur les bugs rencontrés. En effet, j'ai testé l'application de façon théorique avec JUnit, j'ai joué un petit peu au jeu mais un beta testeur aura un avis impartial qui me permettrait d'améliorer l'application et corriger certains bugs que je n'aurais pas vu.

### III. Diagramme de classes obtenu par retro-conception

#### a. Vue d'ensemble

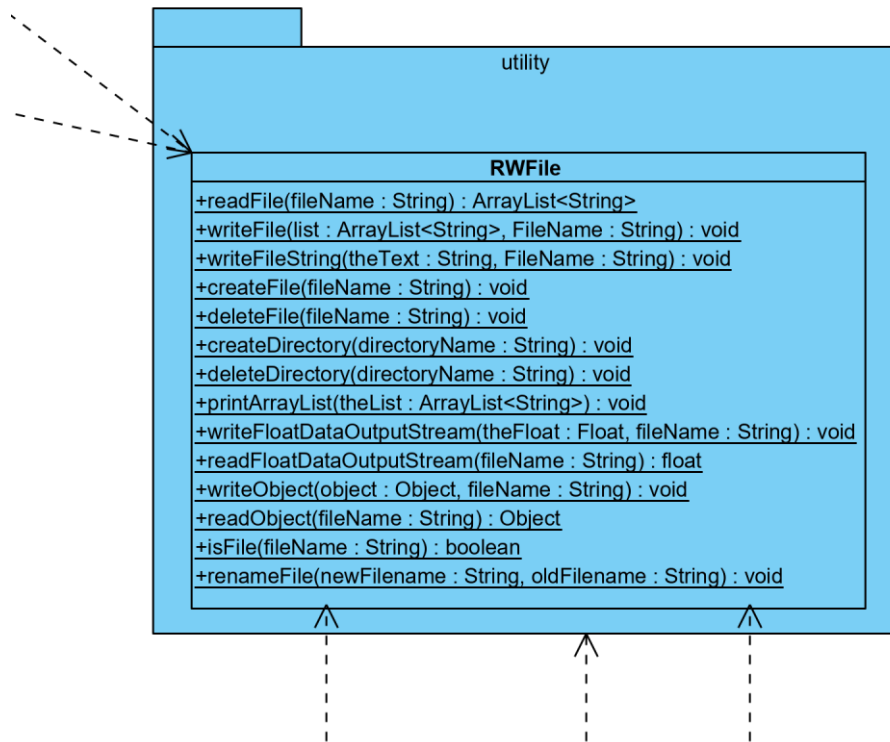


## b. Le package gameElement





#### d. Le package utility



#### IV. Description de la campagne de tests effectués

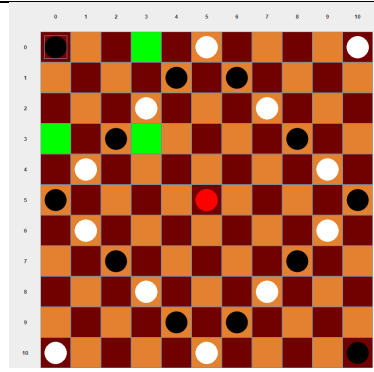
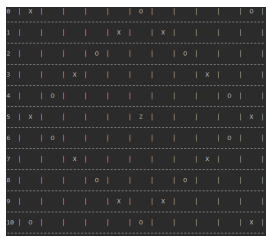
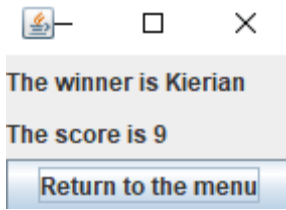
Ces tests ont été réalisés par moi-même, ils ne sont donc pas forcément complets comme l'aurait fait un beta testeur neutre. De plus, les tests JUnit n'ont relevé aucune erreur.

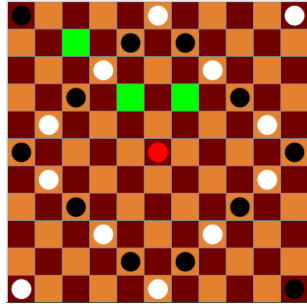
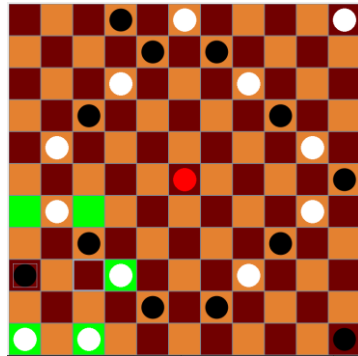
##### a. Fonctionnalité de base

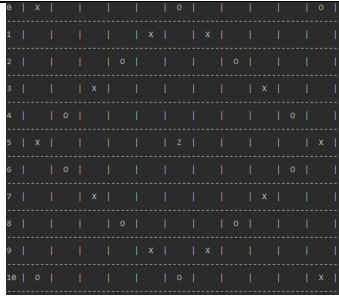

##### b.

Test	Résultat
Tout d'abord, il faut vérifier que l'interface choisie soit correcte à celle définie, qu'il n'y ait pas d'aberration comme une case au mauvais endroit. Et il faudra vérifier que les interfaces soient bien fonctionnelles, c'est-à-dire que si on	Ce test est un succès, tous les boutons et autres composants font les actions requises, tout fonctionne sans erreur.



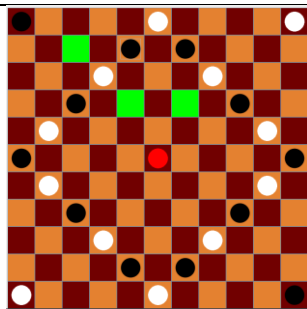
<p>clique sur un bouton, il fait bien ce qui est demandé.</p>	
<p>De plus, il faudra vérifier que la dimension du tablier est bien de 11 cases par 11 cases avec 25 pions avec 12 pions amis et 12 pions ennemis. Il faudra tout d'abord vérifier que le zen est bien présent puis il sera nécessaire de tester la possibilité aux joueurs de jouer le zen en tant qu'ami ou ennemi.</p>	<div data-bbox="909 380 1284 750" data-label="Image">  </div> <p>Figure : Affichage de la grille en mode graphique.</p> <div data-bbox="965 801 1232 1037" data-label="Image">  </div> <p>Figure : Affichage de la grille en mode console.</p> <p>On observe que tout est affiché correctement. Le zen peut-être jouer en tant qu'ami ou ennemi. Ce test est réussi.</p>
<p>Pour repérer qui a gagné, le programme doit compter les chaînes de pions. Pour cela, il faudra vérifier qu'il ne se trompe pas dans le comptage. Et aussi vérifier que le logiciel repère bien la victoire d'un joueur en repérant la chaîne principale sans pions qui ne sont pas rattachés à la chaîne principale. Enfin, un bon comptage des chaînes de pions des joueurs permettra de compter les points durant la partie. Il faudra ainsi vérifier que le logiciel compte et stocke bien les points sans erreur. Il faut donc vérifier que le programme distingue la chaîne principale du perdant de ses pions isolés. Il faudra vérifier que</p>	<p>Tout ceci fonctionne selon mon expérience du jeu que ce soit en mode humain contre humain ou humain contre ordinateur.</p> <div data-bbox="954 1415 1241 1624" data-label="Image">  </div> <p>Figure : Exemple de fin de match.</p>

<p>le jeu continue la partie tant qu'un des joueurs n'a pas atteint 7 points.</p>	
<p>Ensuite, le déplacement se faisant dans toutes les directions, il faudra vérifier que les déplacements effectués par l'ordinateur et les déplacements autorisés pour le joueur soient corrects, c'est-à-dire vérifier que ces déplacements ne sont pas contre les règles du jeu. De plus, il faut vérifier que le programme repère bien les pions ennemis, par exemple, s'il y a un pion ennemi sur le chemin alors il faut vérifier que le jeu refuse le déplacement (en effet, il est impossible de survoler un pion ennemi). En revanche, il faut vérifier que si le pion ennemi se trouve sur la case d'arrivée alors on doit voir que le programme permet la capture du pion ennemi.</p>	<div data-bbox="943 383 1251 685" data-label="Image">  </div> <p>Figure : Exemple de déplacement possible pour un pion.</p> <div data-bbox="916 745 1275 1099" data-label="Image">  </div> <p>Figure : Exemple de capture possible.</p> <p>Le programme est opérationnel dans plus de 95% des cas mais il me semble avoir vu une ou deux fois des choses étranges lors de certains déplacements ou propositions de déplacements en mode graphique, mais étant des bugs isolés je ne m'en suis pas occupé par manque de temps. Je me permets de me corriger, en disant, que j'ai modifié le programme au dernier moment (cf. partie sur les problèmes rencontrés), et je pense que les choses étranges que j'ai vues ont été résolues.</p> <p>En mode console, je n'ai jamais eu de problème.</p>
<p>Enfin, il faut vérifier que le programme comprend et repère la différence entre un match</p>	<p>Dans les quelques tests que j'ai faits, je n'ai vu aucun problème.</p>

<p>nul, à cause par exemple de la capture d'un pion ennemi isolé qui conclut à la victoire de l'ennemi alors le logiciel doit comprendre que le match est nul ou en cas de déplacement du zen qui conclut à la connexion de la chaîne de l'adversaire, d'un match gagnant (c'est-à-dire tous les autres cas).</p>	
<p>Tout d'abord, une des fonctionnalités majeures est le fait de jouer dans une console en mode texte et aussi de jouer en mode interface graphique. Il faut ainsi maintenant tester le mode texte. Pour cela, il faut vérifier que l'interface soit correcte à ce qui était convenu. De plus, il faudra vérifier que ce mode texte est jouable, c'est-à-dire que l'affichage du tablier se fait bien, les pions sont bien distinguables, on peut facilement donner l'instruction de mouvement à nos pions.</p>	 <p>Figure : Grille en mode texte.</p> <p>Le jeu est bien jouable dans les deux modes, il n'y a pas de soucis de ce côté-là.</p>
<p>Ensuite, il faudra tester la possibilité de jouer contre l'ordinateur. Il faut aussi voir si l'ordinateur n'enfreint pas les règles du jeu. Et, il faut aussi vérifier la possibilité de jouer à deux.</p>	<p>Aucun souci du côté du joueur automatique. De plus, le jeu est jouable à deux joueurs humains.</p>
<p>Enfin, il faudra tester la possibilité de sauvegarder une partie en cours, il faudra vérifier qu'il n'y a pas eu de changement entre le moment de la mise en sauvegarde et le moment de récupération de la partie, par exemple au niveau de la disposition des pions et aussi au niveau du comptage des pions.</p>	 <p>Figure : Affichage des parties enregistrées.</p> <p>Il est possible de sauvegarder une partie et de la reprendre plus tard. Il est même possible en</p>

	mode console d'enregistrer un match. Tout ceci est fonctionnel.
--	---

### c. Fonctionnalité évoluée

Pour tester le didacticiel, il faudrait qu'une personne qui ne connaît pas le jeu fasse le didacticiel. Ensuite, il faut lui demander si elle a bien compris les règles et si elle est capable de faire une partie. Si cette personne n'a pas bien compris alors il faudra repenser le fils conducteur du didacticiel.	Je n'ai pas pu faire tester le tutoriel à des personnes ne connaissant pas ce jeu. Donc, ce test n'est pas validé.
Pour tester l'affichage des cases possibles, il faut tout simplement regarder si les déplacements proposés sont possibles. On devrait voir qu'en comptant manuellement les déplacements possibles, on arrive à trouver les cases que propose le logiciel.	 <p>Figure : Exemple d'affichage des cases possibles pour un pion.</p> <p>Je n'ai jamais remarqué de problèmes et comme vous pouvez le voir, ça fonctionne correctement.</p>
Pour tester les différents niveaux de difficulté contre l'ordinateur, il faut tout d'abord regarder que le déplacement de l'ordinateur est bon. C'est-à-dire, vérifier que l'ordinateur respecte bien les règles. Ensuite, pour vérifier les niveaux de difficultés, le joueur doit, tout d'abord, voir une différence dans le jeu de l'ordinateur. Mais aussi, s'il fait plusieurs parties pour chaque niveau, il doit perdre plus souvent avec le niveau	Cette fonctionnalité n'a pas été écrite donc c'est un échec.

le plus dur, à l'inverse, il doit gagner plus dans un niveau plus facile.	
---	--

## V. Etat d'avancement du développement

### a. Fonctionnalité de base

Tout d'abord, nous avons affirmé dans le cahier des charges du six avril que le jeu devrait respecter les règles du jeu Zen l'Initié. Comparons alors les règles mentionnées dans le cahier des charges et la réalité.

*« Tout d'abord, comme tout logiciel, il y a des contraintes à respecter, c'est pourquoi le logiciel sera implémenté en java, le code source sera bien évidemment documenté et testé unitairement. Et, le logiciel sera fourni sous forme d'un JAR signé, accompagné d'un manuel utilisateur. »*

Le programme est bien en java et utilise les librairies java.awt et java.swing. Le code source est bien évidemment documenté grâce à la javadoc et est testé unitairement grâce au Framework JUnit. Le logiciel est fourni avec un jar signé accompagné d'un petit (par manque de temps) manuel utilisateur.

*Le jeu que nous allons programmer pour notre client M. Lefèvre est Zen l'Initié, un jeu de stratégie qui se joue sur un plateau de 11x11 cases (121 cases). Il y a en tout 25 pions. Il se trouve qu'il y a deux équipes, une avec 12 pions noirs et une avec 12 pions blancs. Les plus perspicaces d'entre vous auront remarqué que la somme des pions de ces deux équipes donne 24 pions. En effet, il y a un pion supplémentaire qui n'appartient à aucune équipe, et qui se nomme le « zen ». Ce pion particulier peut-être joué en tant qu'ami ou ennemi de notre équipe.*

En effet, cette partie est entièrement réalisée que ce soit en mode console ou en mode graphique, nous jouons bien sur un plateau de onze par onze soit cent vingt et une cases. Il y a bien douze pions par équipe et un zen. Le zen pouvant être joué en tant qu'ami ou ennemi : cette règle a été respecté. Voici une image d'illustration du plateau de jeu en mode graphique, il est globalement identique en mode console.

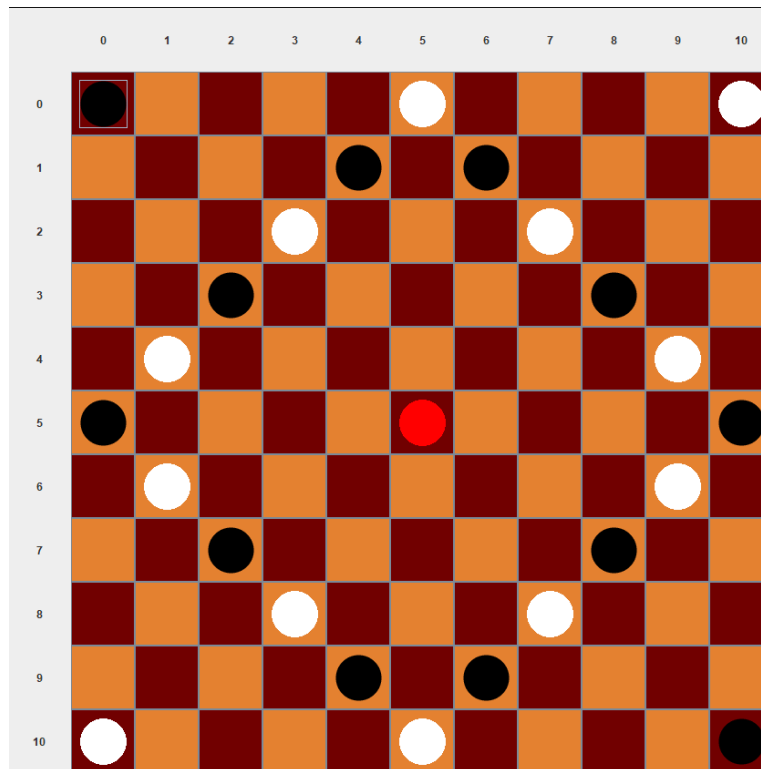


Figure : Présentation du plateau de jeu en mode graphique

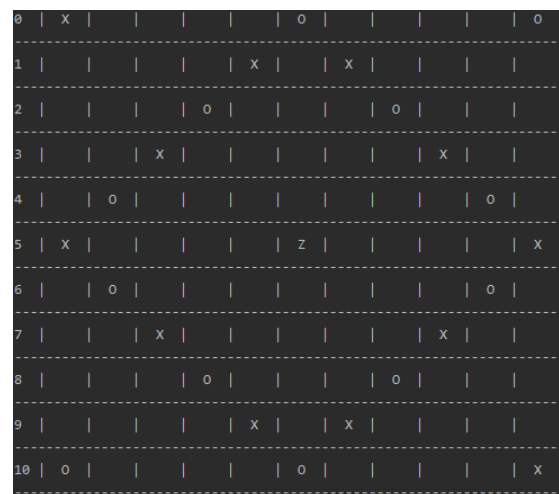


Figure : Présentation du plateau en mode console

*Dans un second temps, le but du jeu est de former une chaîne continue avec la totalité de ses pions se trouvant encore sur le plateau, y compris le « zen » si celui-ci est encore en jeu. De plus, il est possible de capturer les pions ennemis en se plaçant sur la case occupée par le pion ennemi.*

Aujourd'hui, il est possible avec notre logiciel de repérer une chaîne de pions grâce à une méthode récursive qui compte la taille d'une chaîne au hasard et si celle-ci est de même taille que le nombre de

pions du joueur avec le zen s'il est encore en jeu alors le jeu se termine. Et notre jeu inclus la possibilité de capturer un pion ennemi.

*Les pions peuvent se déplacer en ligne droite dans n'importe quelle direction, c'est-à-dire sur la ligne horizontale, verticale et diagonale. Ensuite, tout pion doit toujours se déplacer d'autant de cases qu'il y a de pions sur la ligne de déplacement choisie. A noté, qu'un pion peut passer par-dessus ses pions alliés mais ne peut pas passer par-dessus les pions ennemis.*

Notre logiciel incorpore cette fonctionnalité de déplacement, en effet, lors de la sélection d'un pion, ces cases d'arrivées sont cherchées avant même que l'utilisateur ne tape la case finale qu'il veut pour ce pion. Ensuite, notre code va comparer la case rentrée par l'utilisateur et va comparer avec les cases possibles. A noté que ci-dessous, vous trouverez une capture d'écran de la sélection d'un pion et de l'impression des cases possibles (cette option est désactivable mais nous en reparlerons, ici c'est à titre d'exemple).

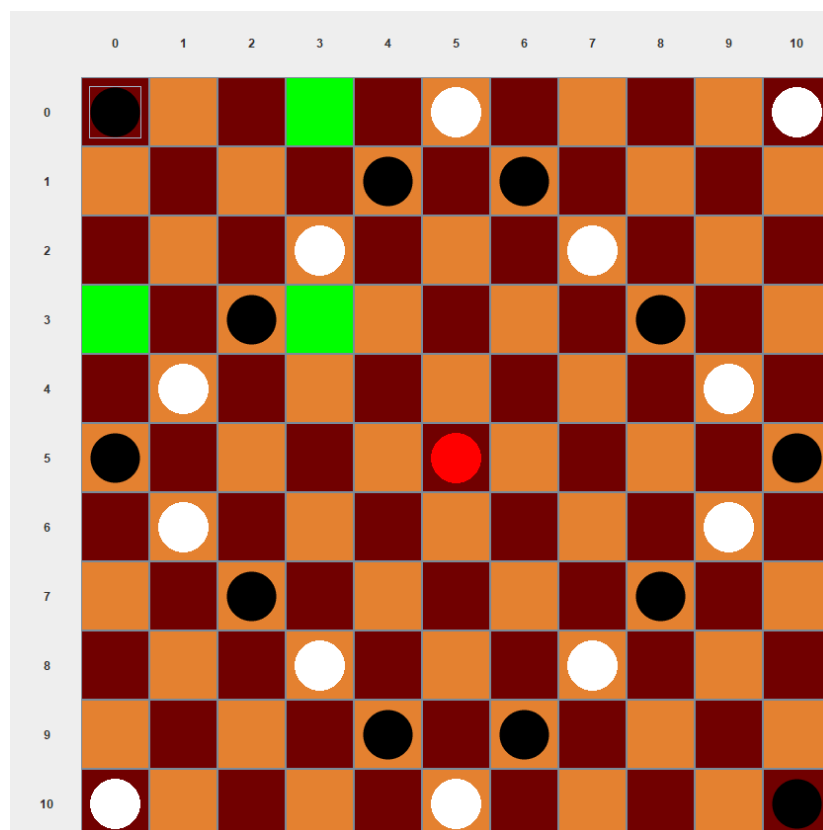


Figure : Illustration des mouvements possible pour un pion.



*Pourtant, il existe des subtilités sur la victoire d'un joueur. En effet, un match peut-être nul si un joueur en déplaçant le « zen » conclut à la connexion de la chaîne de l'adversaire. Il peut aussi être nul si un joueur capture le seul pion isolé de son adversaire, ce qui entraîne que l'adversaire a une chaîne avec ses pions restant.*

Cette fonctionnalité a été réalisée à l'aide d'une simple condition. En effet, on remarque que le jeu est nul dans tous les cas où le joueur courant entraîne la victoire de son adversaire donc une condition vérifie que le joueur courant est bien le joueur gagnant.

*Enfin, jusqu'à présent nous avons parlé de la victoire d'une manche par un joueur. Mais, il se trouve que le gagnant final est le premier joueur à atteindre 7 points ou plus. En fait, pour déterminer le gagnant, il est possible de faire plusieurs manches.*

Avant de parler de l'avancement du développement sur cette partie, il se trouve que j'ai appris que cette règle est une subtilité, c'est-à-dire qu'elle n'est pas obligatoire et c'est une option pour les joueurs confirmés, ce que je n'avais pas compris lors de l'écriture du cahier des charges. Cependant, cette fonctionnalité a été faite en mode console, c'est-à-dire qu'il est possible de faire un match en sept points en mode console. Le programme va compter les points sans problème en repérant les chaînes isolées du perdant. En revanche, cette fonctionnalité n'a pas été incorporée en mode graphique par manque de temps.

## b. Demande du client

*Tout d'abord, le client nous a explicitement demandé de faire une version texte du jeu, c'est-à-dire que l'on pourra jouer depuis une console ou un terminal. Et ensuite, il sera possible de réaliser une version graphique. C'est deux versions doivent avoir les mêmes fonctionnalités.*

Nous avons bien deux versions : une version texte et une version graphique. En revanche, par manque de temps, il y a une fonctionnalité que le mode graphique n'incorpore pas : la possibilité de faire un match en sept point, cette fonctionnalité est seulement présente dans la version console. Sinon, toutes les autres fonctionnalités sont présentes dans les deux versions. Par exemple, l'impression des cases possibles est une option activable et désactivable dans les deux versions.

```
Write the mode that you want :  
- Graphical mode (Gui)  
- Command line mode (Cli)
```

Figure : Au démarrage, on demande le mode au joueur.

*Ensuite, il sera aussi possible de jouer à deux ou seul contre l'ordinateur. Pour jouer contre l'ordinateur dans la version de base, l'ordinateur fera le choix du mouvement aléatoirement parmi les mouvements possibles.*

Cette fonctionnalité a été ajoutée, il est possible de jouer à deux ou seul contre l'ordinateur avec un choix de pions et de cases finales au hasard. J'ai d'ailleurs été surpris de certains mouvements en mode aléatoire qui étaient de très bon mouvements. Cette fonctionnalité est évidemment présente en mode graphique et en mode console.

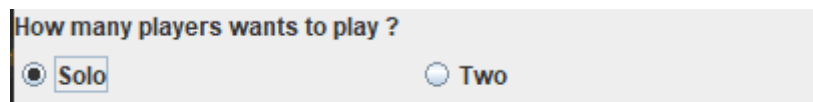


Figure : Illustration du choix du mode de jeu.

*Enfin, il y aura la possibilité d'enregistrer une partie en cours pour pouvoir la reprendre plus tard.*

La sauvegarde d'une partie en cours est possible, cette fonctionnalité est d'ailleurs très avancée. En effet, il est possible d'enregistrer une simple partie en cours et de la reprendre quand l'utilisateur le veut. Ces paramètres de jeu vont être enregistrés afin de reprendre la partie comme lors de la première fois. Ensuite, il est possible de lancer une simple partie, de l'arrêter, de la reprendre, de l'arrêter à nouveau, et de la reprendre à la dernière sauvegarde. Et c'est là toute la subtilité, il n'y aura pas de nouveau fichier de sauvegarde, on va écrire la dernière sauvegarde, le nouvel avancement dans le fichier de la partie. Lors de la fin de la partie, la sauvegarde est automatiquement supprimée. De plus, en mode console, nous avons vu qu'il est possible de faire un match en sept points. Il est possible d'enregistrer un match en cours que ce soit lors de l'entre match c'est-à-dire entre deux manches ou pendant la manche. Si l'enregistrement a été réalisé durant une manche le jeu reprendra au dernier mouvement. Cependant, il est impossible d'enregistrer l'avancement de cette partie si elle est à nouveau arrêtée. En revanche, dans les autres manches, il n'y aura aucun problème.

### c. Evolutions

*Premièrement, lorsque j'ai découvert le jeu, les règles n'avaient pas forcément de sens pour moi, non initié. De plus, je me suis douté que M. Lefevre se laisserait de jouer seul contre l'ordinateur. Mais malheureusement, le jeu n'a pas traversé le temps et aujourd'hui peu de personnes serait en mesure de jouer avec notre client. C'est pourquoi, j'ai pensé que M. Lefevre apprécierait de présenter ce jeu à ses amis, à ses enfants afin de ne plus jouer seul. Mais apprendre un jeu peut paraître fastidieux. C'est pourquoi, j'ai eu l'idée de créer un didacticiel où le débutant pourrait être guidé dans une première partie afin de découvrir les règles de ce jeu de stratégie.*

Cette fonctionnalité a été réalisée en mode console et en mode graphique. J'ai même repris le personnage du professeur Chen pour jouer le rôle du professeur. En mode console, j'ai d'ailleurs utilisé du ascii art afin de présenter les règles et de rendre les choses plus agréables. Cependant, même si le didacticiel est présent dans les deux modes, en mode console je demande aux joueurs de finir une partie, chose que je ne fais pas en mode graphique. Donc le didacticiel a plus de fonctionnalités en mode console que dans le mode graphique. A part cela, dans les deux modes, le fil conducteur est identique.

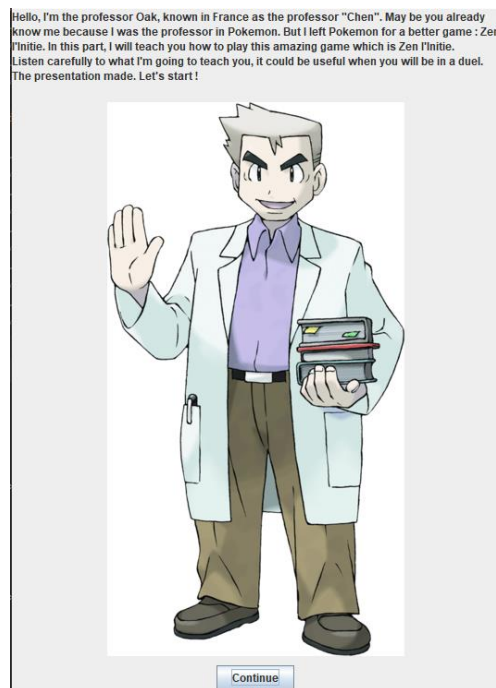


Figure : Illustration du tutoriel en mode graphique.

```

Hello, I'm the professor Oak, known in France as the professor "Chen". May be you already
know me because I was the professor in Pokemon. But I left Pokemon for a better game : Zen
l'Initie.
In this part, I will teach you how to play this amazing game which is Zen l'Initie.
Listen carefully to what I'm going to teach you, it could be useful when you will be in a duel.
The presentation made. Let's start !

-----\ /-----
      \ /
       V

      (      %&/
      .      (
      , , , , ,      , , , , ,
      /( * * #      * * ,      , , , *
      % * * #      , *      , , ( * *
      . * * % % %      % % % % . // &
      . . (      & & /      & & # * ( . . #
      , , , (      &      * #      * / . ( . *
      //      , , , , ( ,
      /      , , , , , (
      .      , , , , , / * * * * * , , , , ,
      ( % , (      ( . , # # ( / . , #
      * # . *      # ( , * # , , , , , * * #
      , # # # #      %      * , , , % / , & * * * * * % . / . . % /
      / * / ( # #      *      * % , , / * * @ * * / * * * * ( . . # . . . . . #
      * * * * /      %      / * ( * * * * * # * * . . . . .
      . / . #      . # ( , , , , , * * * * * % * , , , , , . . /
      * / ,      #      , , , , , , / , , , , , # , , , , , %
      % # . * *      ( # , , , , , , , , , , *      * * . . . ,
      & & . . . * # . . . . . ( ( , , , , , , , , , , (      , , , , , .
      % . . . , , * . . . . . , , , , , , , , , , , (      *      % . . . . .
      . . . # . . . . . & . . . , , # , , , , , , , , , , /      /      * . . . . . #
      / . . . . % . . . . . # , , , , , , , , , , , /      , , , , , . . . . . %
      * . / . . . . . , , , , , , , , , , , ( , , , , , , , , , , , @ & % % % % ( ( ( ( ( ( ( ( ( ( ( ( & # & . . . ,
      . . . . . ( . % . . . . . ( % , , , , , , , , , , , # . . & % # * , , , , , # / , * , * ( & / % / # .

```

Figure : Illustration du tutoriel en mode console.

*Dans un second temps, étant en train d'apprendre à jouer aux échecs, j'ai expérimenté différents jeux d'échecs sur pc. Et je remarquais qu'au début, il est courant de se tromper dans le déplacement des pions. C'est pourquoi, pour les joueurs débutants, quand on clique sur une pièce, il est possible de distinguer les cases sur lesquelles il peut se déplacer. Le principe serait le même sur le jeu de Zen l'Initié, on clique sur un pion et on est capable de distinguer les cases sur lesquelles le joueur peut aller.*

Cette petite fonctionnalité a été incorporée en mode console et en mode graphique. En mode console, l'utilisateur rentre la position d'un pion et le programme lui retourne les positions possibles. En mode graphique, il suffit de cliquer sur un de nos pions et en vert on peut observer les cases possibles. Si l'utilisateur ne clique pas sur une des cases en vert alors le pion est désélectionné et les cases vertes disparaissent. Il s'agit d'une option c'est-à-dire que l'utilisateur s'il le décide peut retirer cette fonctionnalité et alors le programme va arrêter de lui donner les cases possibles pour un pion.

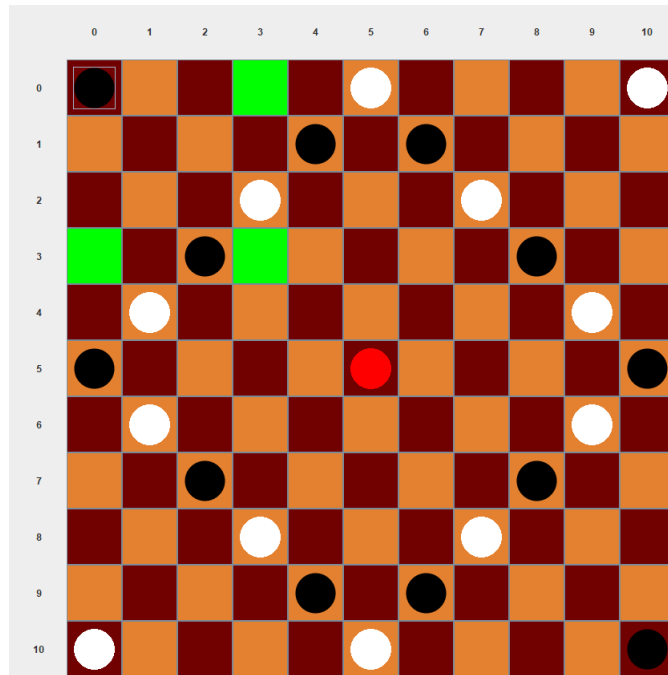


Figure : Illustration en mode console de l'affichage des possibles cases.

```

0 | x | | | | | o | | | | |
1 | | | | | x | | x | | | |
2 | | | | o | | | o | | | |
3 | | | x | | | | | x | | |
4 | | o | | | | | | | o | |
5 | x | | | | | z | | | | x |
6 | | o | | | | | | | o | |
7 | | | x | | | | | x | | |
8 | | | o | | | | o | | | |
9 | | | | x | | x | | | | |
10 | o | | | | | o | | | | x |

rgr plays with the blue cross

Which pawn do you want to move ?
Please enter the x position of the pawn that you want to move
0
Please enter the y position of the pawn that you want to move
0
You can see the possible square from this pawn :

(3 ; 0)
(0 ; 3)
(3 ; 3)

Please, enter the final position of the selected pawn
Please enter the x position of the final position of the pawn

```

Figure : Illustration en mode console de l'affichage des possibles cases.

*Enfin, la dernière fonctionnalité qui serait intéressant d'ajouter est la création de plusieurs niveaux de difficulté quand on joue contre l'ordinateur. Cela permettrait d'ajouter un peu de difficulté quand M. Lefevre jouera contre l'ordinateur. Et puis*

*cela pourrait permettre d'aider les amis de M. Lefevre à s'améliorer et s'entraîner pour un jour espérer le battre.*

Cette évolution n'a pas été faite, je n'ai même pas eu le temps de me documenter ou même de commencer à chercher comment faire.

Pour conclure, l'application répond donc aux normes décrites dans le cahier des charges. Cependant, il reste encore quelques détails pour que le jeu soit parfait. Par exemple, il serait nécessaire de gérer le cas où un des joueurs s'est fait capturer tous ses pions, ou encore rendre l'interface un peu plus jolie.

## VI. Synthèse des difficultés rencontrées et des solutions apportées

Le plus gros problème de la réalisation de ce projet était le temps. En effet, nous ne pouvions commencer la programmation qu'à partir du moment où nous avons rendu le cahier d'analyse et de conception le 24 mai. Ce qui nous laissait normalement un mois pour coder l'application. Mais il m'était impossible de coder avant étant donnée la surcharge de travail durant les deux premières semaines et les partiels pendant quatre jours. Ce qui nous laisse un peu moins de deux semaines pour programmer sans contrainte. Afin de compenser, il a fallu travailler très dur durant cette semaine et demie de programmation. Le résultat obtenu est le fruit de huit jours (un jour de plus que prévu) de travail à raison de dix à douze heures de travail par jour, il a été possible d'arriver à ce résultat. En effet, avec le compte rendu final du projet et un compte rendu de communication où nous avons été prévenus une semaine avant la fin du projet, nous avons été très limités en termes de temps et tout n'a pas pu être compenser par des horaires que seuls les grands patrons et chefs d'états font.

Nous avons aussi rencontré un problème lors de la capture et du mouvement d'un pion. En effet, pour capturer ou déplacer un pion, je changeais la valeur des attributs de l'attribut de la classe « Pawn » de type Square, par exemple :

```
pawn.getSquare().setY(-1);  
pawn.getSquare().setX(-1);
```

Ce qui était une très mauvaise idée puisque je modifiais la valeur des attributs de l'objet Square alors que je devais modifier l'attribut de type Square dans la classe « Pawn ». C'est pourquoi, maintenant, le programme n'utilise plus cette méthode, il modifie directement l'objet Square qui est en attribut :

```
pawn.setSquare(aSquare);
```

Quelques heures avant le rendu final, nous nous sommes rendus compte d'une erreur à l'exécution. En effet en mode humain contre l'ordinateur, le programme affiche le coup de l'ordinateur avec un coup de retard. Le problème venait de la grille qui ne mettait pas à jour le coup de l'ordinateur. Pour remédier au problème, au lieu d'afficher la grille après que le joueur humain ait joué, le programme l'affiche après le coup du joueur humain et de l'ordinateur.

Enfin, le dernier problème rencontré est arrivé à cause du jar. En effet, jusqu'à présent nous utilisons des chemins relatifs pour accéder aux fichiers. Or, lors de la création du jar, nous nous sommes rendu compte que ça posait un problème, et que le jar ne comprenait que les chemins d'accès absolu. Pour cela, nous avons créé une méthode qui permet de trouver le chemin absolu pour accéder à la classe grâce à la classe URLDecoder du package java.net. On enlève les deux derniers noms de fichiers qui ne nous intéressent pas, et on arrive au même niveau que le fichier qui contient le jar. Et, nous avons ajouté le fichier data à ce niveau.

De plus, pour les images, nous les plaçons dans le jar et le programme y accède grâce à la méthode `getResource()`.

```
getClass().getClassLoader().getResource("NomDel'Image.png")
```

## VII. Bilan personnel du travail réalisé

Il est vrai que travailler sur ce projet n'a pas été de tout repos, mais il est vrai qu'il a été très enrichissant et a servi de très bon bilan de fin d'année où il a été possible de réunir toutes les connaissances apprises cette année. J'ai pris beaucoup de plaisir à le coder malgré les journées à rallonge. C'est d'ailleurs sur cette partie que je sais que je devrais m'améliorer à l'avenir : mieux gérer mon temps de travail. En effet, il est vrai que je garde une certaine frustration de ne pas avoir pu terminer toutes les choses que j'avais planifié lors du cahier des charges. Cependant, je suis déjà très fier du résultat, au bout de seulement une semaine nous avons un jeu fonctionnel, incluant des fonctionnalités qui le rendent plus agréable alors qu'il y a quelques mois je ne connaissais ni le java ni la programmation orientée objet. Ce fût un travail qui m'a fait dire que j'aimais ce que je faisais même si c'était dur et que je voulais continuer sur cette voix dans ma vie professionnelle future.