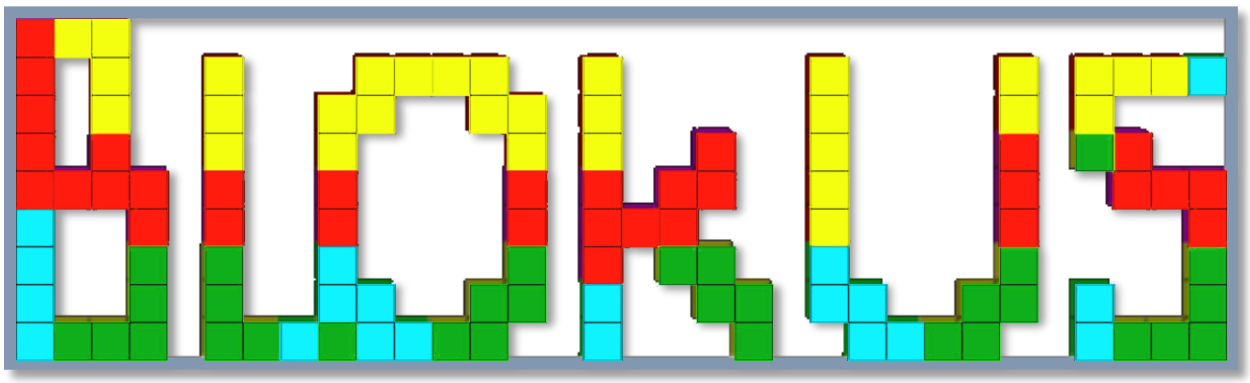
FRAGNI Dorian

BERTIN Clément

Gr1A1

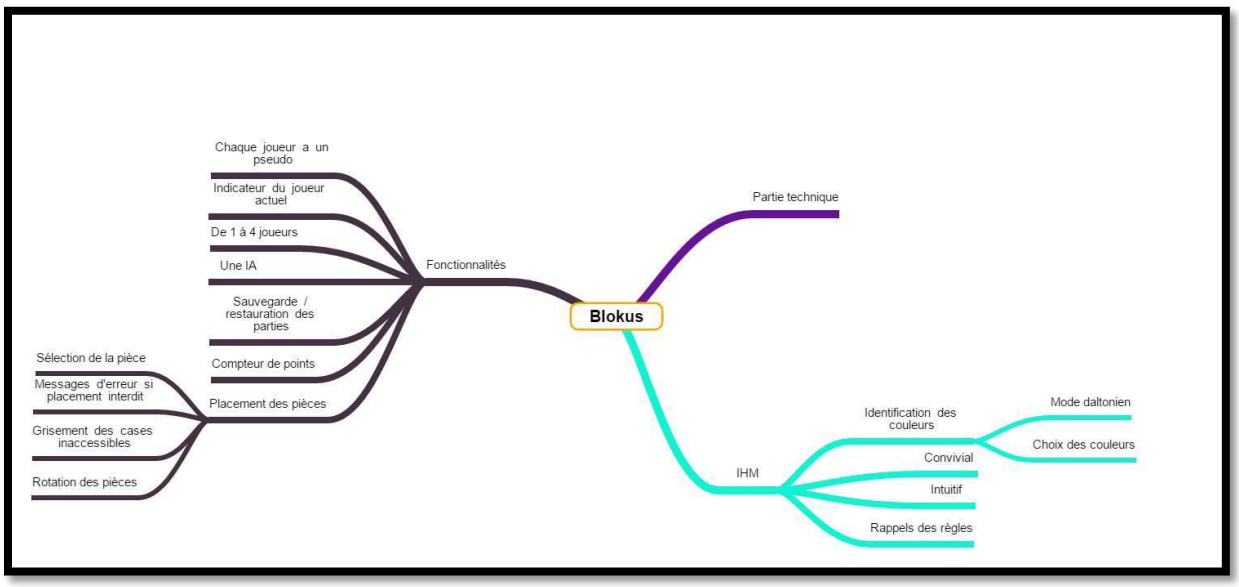


Projet de programmation

L'objectif du projet de programmation 2016 est de réaliser un BLOKUS en Java. Des équipes de quatre personnes ont été formés pour réfléchir et rédiger un rapport sur l'application en java.

Les membres de l'équipe sont : 1er binôme (Bertin Clément & Fragni Dorian) et 2nd binôme (Leboul Florian & Doré Nathan).

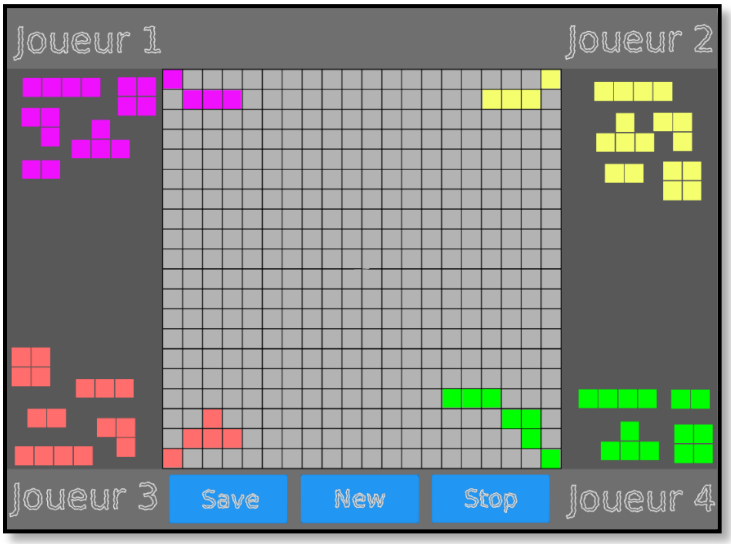
Pendant la phase de réflexion en équipe, nous avons pu faire un brainstorming sur les fonctionnalités du BLOKUS, nous avons aussi réalisé une mindMap avec toutes les fonctionnalités du Jeu :



Suite à la création de la carte mentale, nous avons réalisé un diagramme de classe qui a subi certaines mises à jour depuis le début du projet. Le diagramme représente de manière exhaustive le modèle MVC, toutes les classes et les méthodes implémentées dans le Blokus.

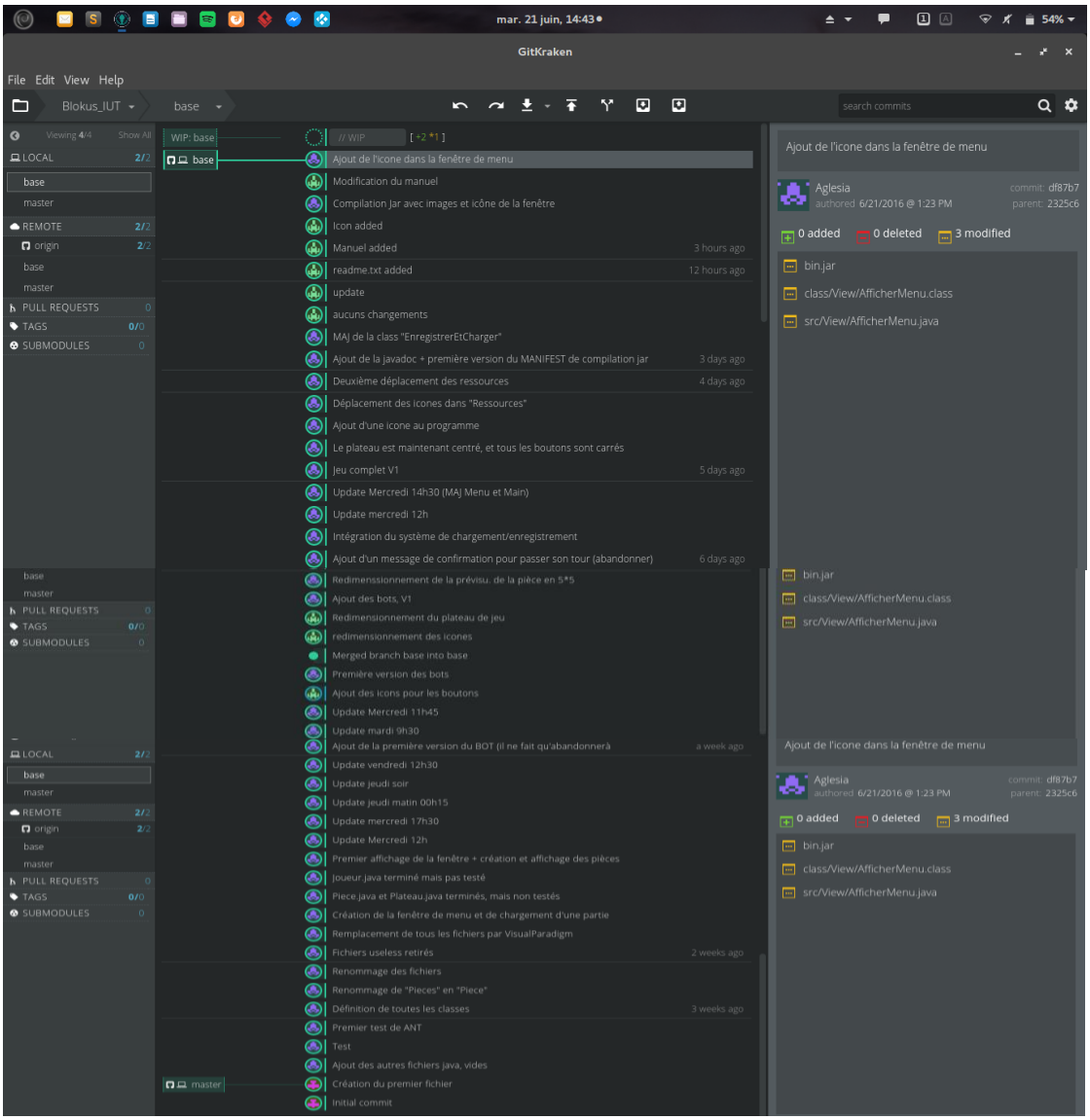
Nous avons fait différents choix sur les joueurs, par exemple, ces derniers peuvent choisir leurs couleurs. Nous essayons de laisser un large panel de couleur car le Blokus peut être utilisé par des daltoniens. Il ne faut pas les pénaliser.

Nous avons fait des maquettes de l'interface graphique du menu et du plateau de jeu (qui ont ensuite connu de grosses modifications par la suite) :



Répartition des tâches :

Lors de notre projet, nous nous sommes servis d'un GIT (GITHUB). Grâce à ses fonctionnalités, nous pouvons voir la répartition des tâches au sein du projet :



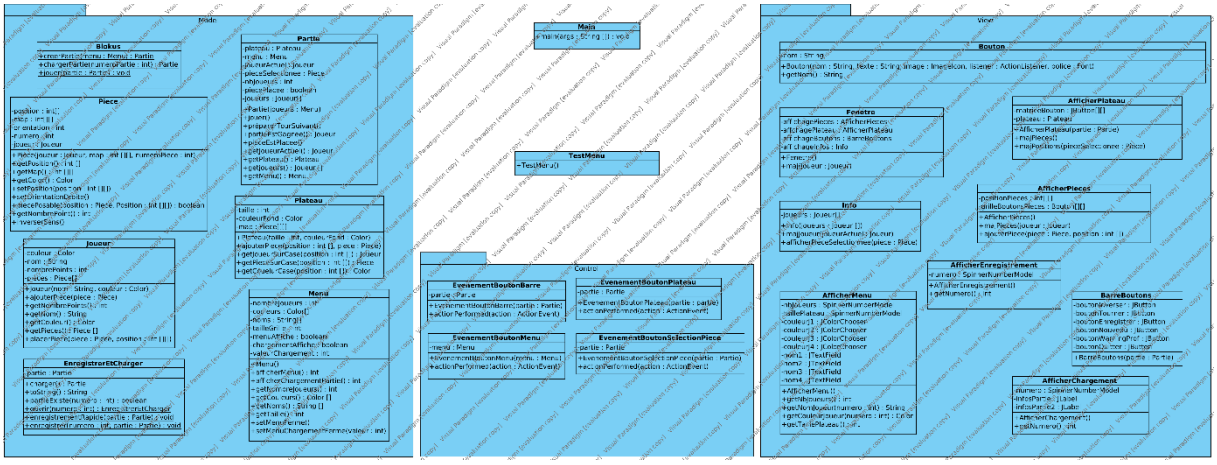
Grâce à GIT, nous pouvons aussi voir l'avancée du projet et les différents auteurs des Commits.

 : Dorian F.

 : Clément B.

Diagramme de Classe original :

Nous avons commencé par créer le diagramme de classe, puis avons généré les classes et méthodes dans leurs fichiers java ainsi que leurs javadocs, directement depuis VisualParadigm.



Version agrandie en annexe

Choix techniques :

Partie modèle du Blokus

Les pièces :

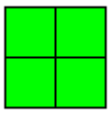
Pour la réalisation des pièces du BLOKUS nous avons utilisé des matrices d'entiers.

Nous avons remplis les matrices avec quatre valeurs différentes :

- 1 : la valeur 1 correspond à un coin de la pièce.
- 2 : la valeur 2 correspond à un bord de la pièce.
- 3 : la valeur 3 correspond à la partie « colorée » de la pièce.
- 0 : la valeur 0 correspond à la partie vide de la matrice.

Un exemple de pièce :

1 2 2 1 0 0 0
2 3 3 2 0 0 0
2 3 3 2 0 0 0
1 2 2 1 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0



1 2 2 1
2 2
2 2
1 2 2 1

Pour qu'une pièce puisse être placée sur le plateau, elle doit avoir au moins un coin de la pièce (case à 1 dans la matrice) qui se superpose à une pièce (case 3 dans la matrice) du même joueur.

La seconde condition pour que la pièce puisse être posée est qu'aucune case à 2 ou 3 de la pièce à poser ne soit sur une case 3 d'une pièce déjà posée du même joueur. La dernière condition est que toutes les cases à 3 de la pièce se posent sur une case vide du plateau.

Les joueurs :

La classe *joueur* contient toutes les propriétés du joueur : son nom, sa couleur, sa liste de pièces, ses points. Elle contient aussi la méthode qui permet de jouer.

La classe partie appelle une à une les méthodes *jouer* de chaque joueur (chacun son tour) jusqu'à la fin de la partie.

Une classe Bot peut donc être facilement intégrée : elle est sous-classe de *joueur* et ne redéfinis que la méthode *jouer*.

Lorsqu'un joueur n'a plus de pièces disponibles ou que ce dernier abandonne, il ne peut plus jouer jusqu'à la fin de la partie. La partie se termine lorsque les quatre joueurs ont abandonnés.

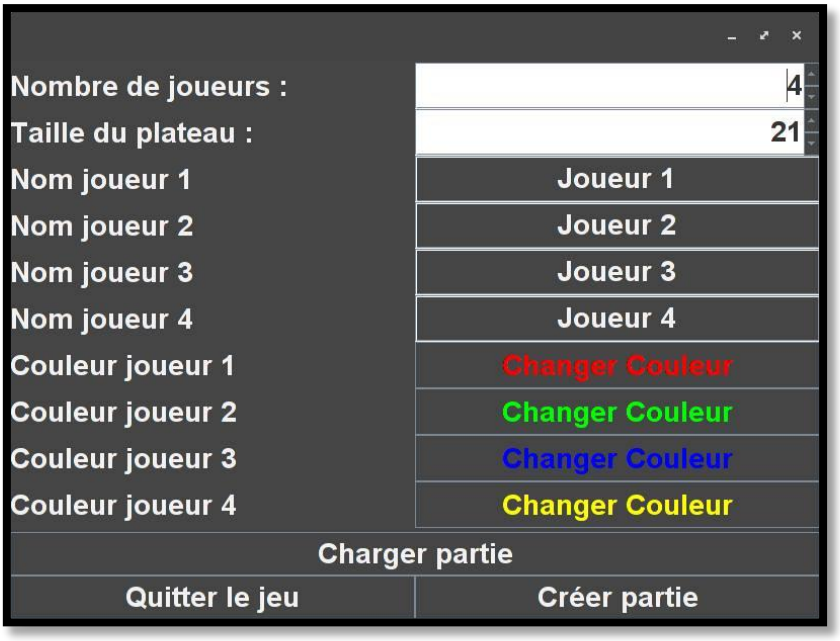
Les Bots :

Lorsque l'on commence une partie, nous choisissons le nombre de joueurs réels. Comme expliqué précédemment, les joueurs manquants sont remplacés par des bots (nous ne proposons pas de mode 2 joueurs).

Les bots vont chercher à placer les plus grosses pièces en premier : ils vont chercher toutes les possibilités pour placer toutes les pièces de 5 cases. Une fois toutes les possibilités listées (numéro pièce + position) ils en choisissent une de manière aléatoire dans la liste. Si la liste est vide, ils recommencent avec des pièces de 4 cases puis 3, puis 2 etc.

Partie visuelle du Blokus

Menu de la partie :



Avant la création de la partie, on demande au joueur les différents paramètres de la partie : le nombre de joueurs, la couleur des joueurs/pièces et la taille du plateau.

Nous avons opté pour un menu sobre, et le plus intuitif possible. Chaque joueur peut changer son nom et sa couleur. Le menu permet de sélectionner la taille du plateau, plus le plateau est petit, plus la difficulté s'accroit.

Déroulement d'une partie :

Une fois le menu créé, la partie est créée en fonction des paramètres de ce menu. Si il y a moins de 4 joueurs, les joueurs manquants sont remplacés par des bots. Ensuite, tant que la partie n'est pas terminée, On répète un cycle :

- On prépare le tour du prochain joueur (changement du joueur actuel, listage des pièces restantes, mise à jour de l'affichage).
- Le joueur peut enfin jouer, 2 choix s'offrent à lui : il sélectionne une pièce, change son sens (si besoin) et la place sur le plateau ou Il peut choisir d'interagir avec les boutons, comme *quitter*, *enregistrer* ou *abandonner*.
- Le passage au joueur suivant s'effectue.

Pendant la partie, il est possible d'enregistrer la progression de la partie dans une des dix fichiers de sauvegarde (*0.save -> 9.save*). Il est possible de retrouver sa partie grâce au bouton « *charger partie* » sur le menu. (Capture d'écran ci-dessus).

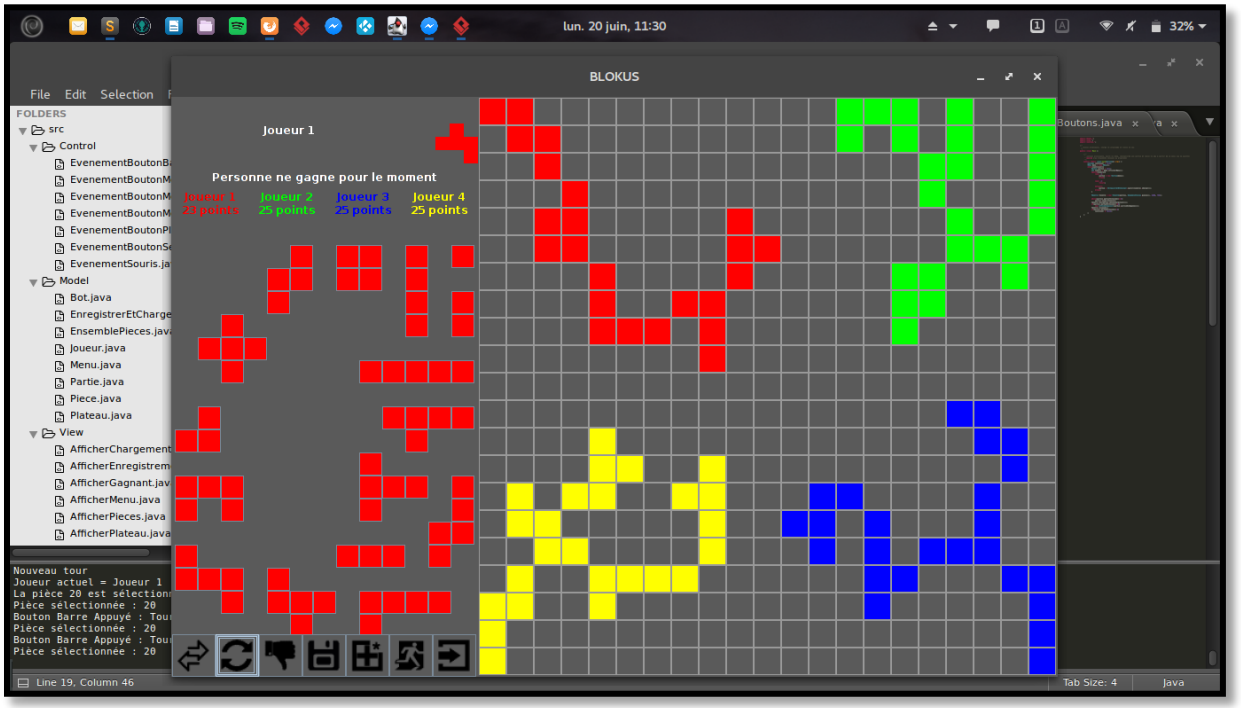
Représentation des pièces :

Que ce soit sur le plateau ou sur la grille de sélection des pièces, nous utilisons des JButton.

- Sur le plateau lui-même, les boutons pointent vers la pièce à placer ainsi que sa position sur le plateau, lors d'un clic sur ce bouton. La liste des possibilités sur chaque bouton (pièce / position) est actualisée lors d'une sélection ou d'une rotation d'une pièce. Si un bouton ne pointe vers aucune possibilité, il est grisé.
- Du côté de la sélection des pièces, les boutons pointent vers le numéro de la pièce correspondante et sont grisés si la pièce est déjà utilisée.

La pièce actuellement sélectionnée est, elle, représentée par un tableau de Jlabel (5*5) contenant des espaces et changeant de *background* selon la pièce et sa couleur.

Ci-dessous, un exemple : nous voyons bien la grille de sélection (à gauche) et le plateau avec les pièces posées (à droite). La représentation de la pièce sélectionnée et de sa rotation se trouve en haut à droite de la partie gauche.



Utilité des boutons :

(1) (2) (3) (4) (5) (6) (7)

Le bouton (1) inverse la matrice la pièce sur l'axe X.

Le bouton (2) tourne la pièce de 90°.

Le bouton (3) permet au joueur d'abandonner la partie en cours (quand, par exemple, il ne peut plus placer de de pièces).

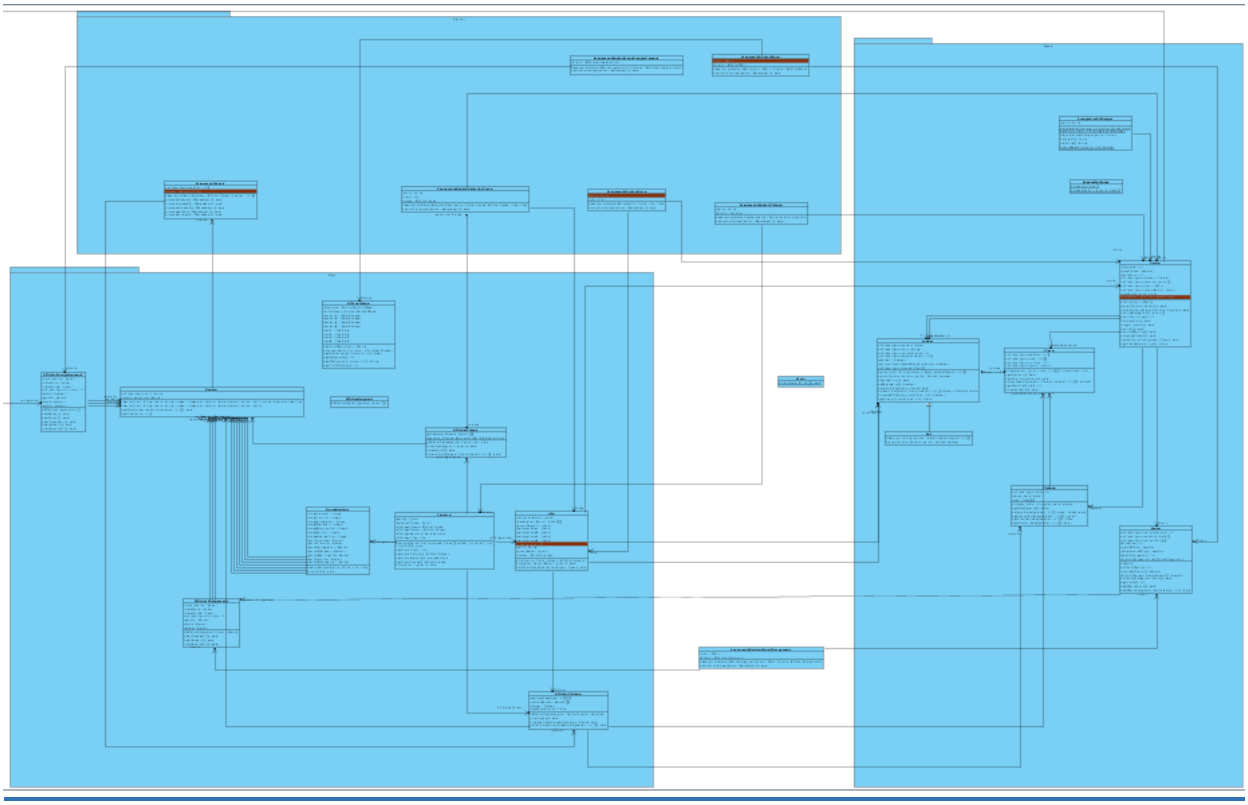
Le Bouton (4) permet de sauvegarder la partie en cours pour permettre de la reprendre plus tard.

Le Bouton (5) permet de recommencer une nouvelle partie.

Le Bouton (6) permet d'enregistrer et quitter très rapidement le jeu quand un professeur arrive.

Le bouton (7) permet de quitter le jeu sans sauvegarde automatique.

Diagramme de Classe obtenu par rétro-conception :



Version plus grande en annexe

Campagne de tests :

Nous avons fait très peu de test : nous avons codé le Blokus block par block (menu, plateau, joueur …), puis testé chaque block avant de les assembler. Nous n'avons pas utilisé JUnit, nous avons fait nos tests nous-même.

État d'avancement du projet :

Le projet est terminé dans sa globalité, nous avons fait la majorité du projet et terminé la première version sur les 2 premières semaines, puis nous avons peaufiné le programme lors de la dernière semaine.

Nous avons respecté, au maximum, les objectifs fixés par le cahier des charges (dont le modèle MVC).

Nous avons rencontré très peu de problème ; le point le plus compliqué pour nous a été d'intégrer des ressources dans une archive .jar (image, son …). Les rares problèmes ont été résolu grâce au forum et à l'API java.

Dorian : Ayant des connaissances assez poussées dans la programmation, j'ai réussi à apporter mon aide à d'autres binômes.

Bilan personnel du travail réalisé :

Clément :

Ma part de participation au projet est clairement inférieure à celle de Dorian, la première semaine n'a pas été productive pour moi en raison de mon accident au coude, l'utilisation d'un PC m'était difficile j'ai donc assisté Dorian tant que je le pouvais avec des conseils principalement esthétique et ergonomique pour l'application.

La seconde semaine Dorian avait déjà fini la plus grande partie du projet et mes compétences en programmation ne me permettaient pas d'avancer aussi vite que lui.

Mon principal apport fut au niveau graphique de l'application et la réalisation des icônes et à rapporter les éventuelles bugs de l'application en fonctionnement afin que Dorian puisse les corriger.

Néanmoins, le projet m'a permis de prendre en mains de nouveaux outils (GIT et Visual Paradigm) qui se sont avérés très utiles pour partager les fichiers via GitHub, et la conception des diagrammes UML avant et après la conception du code Java.

Cependant certains des outils imposés tel que ANT apache n'ont pas été utilisés car nous ne connaissions pas leurs fonctionnements et nous n'avions eu aucun de cours dessus.

Dorian :

Ayant des connaissances relativement avancées dans le domaine de la programmation (orienté objet ou non), je n'ai rencontré que très peu de difficulté lors du codage. De plus, j'ai mis très peu de temps à visualiser l'architecture du futur programme, et à élaborer une première version d'un diagramme de classe pour le projet. Ne rencontrant que très peu de problème, et voyant déjà comment le programme fonctionnerait, j'avançais très vite, même en aidant les autres binômes à côté. Lors de la première semaine, j'avais déjà fini 80% du projet, le cœur du programme était terminé et il ne restait plus que des classes annexes (Comme l'IA) à implémenter ainsi que des bugs mineurs à résoudre.

Pendant ce temps, Clément ayant le bras dans le plâtre ne pouvais pas coder. Il n'a donc pas pu avancer de son côté, sur le projet, si ce n'est le Readme et la notice d'utilisation du jeu.

Visualisant la structure globale de mon code, je n'ai pas pris le temps de faire des tests unitaires pour chaque classe. J'ai commencé par coder le menu, puis testé son retour (Le menu nous retourne juste les paramètres de la partie à créer). Ensuite, je suis parti sur le cœur de la partie, que j'ai testé indépendamment. Par la suite, je n'ai fait qu'implémenter les classes petit à petit, en les intégrant directement dans le programme final. Les tests se faisaient donc sur le programme global, qui s'enrichissait petit à petit.

Le plus gros souci que j'ai rencontré a été pour la compilation finale, lors de a création de l'archive java : Je pouvais compiler mon programme et l'intégrer dans une archive java (.jar), mais je n'arrivais pas à y intégrer les ressources nécessaires tel que les images, et y accéder ensuite dans mon programme. Pour résoudre ce problème, les forums, la documentation java ainsi que les autres binômes m'ont beaucoup aidé.

Ce projet m'a permis de découvrir et de prendre en main « VisualParadigm », un outil plutôt puissant qui m'a permis de concevoir toute ma javadoc à partir du diagramme de classe, et ensuite de recréer

mon nouveau diagramme de classe à la fin du projet, à partir de mon code et des différents ajouts effectués.

Concernant ANT, nous ne l'avons pas utilisé car nous n'avons eu aucun cours dessus, et n'ai pas compris l'intérêt de cet outil. De même pour JUnit, dont je n'ai entendu parler qu'à la dernière semaine de projet.

Annexe : Les images de visualParadigm étant trop grandes pour entrer sur une feuille A4, nous ne pouvons mettre que les liens vers ces images.

Voici donc le lien vers notre Git, contenant nos sources, notre programme fini, et nos documents (dont ce dossier).

Lien vers le Git : *https://github.com/Aglesia/Blokus_IUT/tree/base*