Documentation pour le développeur

Ce projet a été réalisé en utilisant les préceptes de la programmation par binôme. Pour que le code produit par chacun de nous soit compris de l'autre nous avons travaillé en binôme. Cela nous a permis également de corriger des erreurs plus rapidement car à chaque phase de développement nous étions deux, un qui code et un qui regarde et qui réfléchit sur les solutions algorithmiques ou sur les éventuelles corrections de bugs. C'est un projet qui nous a permis de réfléchir sur la structure la plus adaptée selon nous et sur des choix algorithmiques pensez par nous-mêmes.

Nous avons choisi de mettre en place la structure MVC (Modèle-Vue-Contrôleur) afin d'organiser notre code de façon à donner à chaque classe un rôle et donc des responsabilités précises. Cette architecture permet également de rendre le code plus facilement maintenable et de réduire la dépendance entre les classes (faible couplage). Les contrôleurs nous permettent de gérer les changements d'états des modèles en informant les vues des actions utilisateurs. Les vues nous permettent de présenter les données en cohérence avec l'état du modèle. Les modèles nous permettent d'implémenter le fonctionnement du système tout en gérant les accès aux données. Les vues se construisent donc à partir du modèle, les contrôleurs notifient les modèles des changements que l'utilisateur spécifie dans la vue et les contrôleurs informent les vues que les modèles ont changés et qu'elles doivent se reconstruire en prenant en compte ses changements. Cela permet une distribution des responsabilités et des rôles aux classes. C'est pour ces raisons que nous avons choisi d'utiliser MVC en architecture de notre projet.

Model

Grid :

Cette classe fait partie du modèle et elle contient tout ce qui, selon notre conception, est similaire sur Grid5D et Grid5T. Lors de notre conception, le tableau d'utilisation de la grille est différent pour ces deux là. On utilise un booléen pour marquer qu'un point est utilisé dans une direction pour la grille 5D. On utilise un Integer pour marquer qu'un point est utilisé dans une direction pour la grille 5T. Une uniformisation est possible mais dans notre conception, ces solutions étaient logiques.

Grid5D :

Cette classe fait partie du modèle et contient toutes les instances permettant de gérer la grille du morpion solitaire en mode 5D. Elle permet de mettre en place la croix de départ en stockant dans une ArrayList tous les points nécessaires à la mise en place de cette croix. Elle permet également de capter toutes les coordonnées de chaque point de la grille qui correspondent aux intersections. C'est dans cette classe que toutes les vérifications suite à un clic utilisateur sur une intersection de la grille sont mises en place. Les fonctions checkPossibleMove…() permettent de vérifier si par rapport au clic du joueur le coup est possible dans les différentes directions (Horizontale, Verticale, Diagonale Gauche, Diagonale Droite). Si un coup est valide dans plusieurs directions, cette classe propose à l'utilisateur la liste des coups possibles pour le point sélectionné par l'utilisateur. L'utilisateur fait son choix et le coup choisi apparaît sur la

grille dans la direction souhaitée par l'utilisateur. Cette classe permet également après vérification de sélectionner les points de début et les points de fin d'une ligne. Lorsqu'une ligne est choisie la classe renseigne les informations souhaitées dans ses instances de classe : la liste des mouvements possibles, la liste de toutes les intersections sur lesquelles un point est dessiné, la Map qui permet de pour chaque point de savoir dans quelle direction il est utilisé, la liste des points utilisateurs dans l'ordre. Tous ces choix sont selon nous les plus pertinents car c'est au sein même de la grille que les fonctions qui permettent de gérer la grille doivent se situer. C'est le rôle de cette classe. C'est la grille qui gère les coups en les acceptant ou pas. C'est la responsabilité de cette classe de gérer les coups car c'est elle qui a les connaissances sur les règles du jeu et qui est donc en mesure de gérer chaque coup.

Grid5T :

Cette classe fait partie du modèle et contient toutes les instances permettant de gérer la grille du morpion solitaire en mode 5T. Cependant elle à un comportement qui diffère un peu car le mode de jeu n'est pas le même. En effet nous avons ajouté 4 fonctions de sélection des points de début et des points de fin (drawMove…()) supplémentaires. Ce choix s'explique par le fait que nos fonctions qui nous permettent de sélectionner ces deux points qui permettront de tracer la ligne commencent la recherche de points par la gauche. Mais nous nous sommes aperçues qu'il fallait aussi commencer les recherches par la droite sinon certains coups ne pouvaient jamais être joués. Les fonctions drawMove…() commencent les recherches par la gauche et les fonctions drawMove...2() commencent les recherches par la droite.

Graph :

La classe Graph fait partie du modèle. Elle a pour responsabilité de transformer en Point une liste d'entier. Elle sera utilisée notamment pour afficher l'historique des scores dans un même mode de jeu.

Line :

La classe Line fait partie du modèle et correspond à un coup utilisateur qui va tracer une ligne passant par 5 points. Cette classe permet donc de connaître pour chaque coups valide de récupérer les 5 points permettant de tracer la ligne. La responsabilité de cette classe est donc de stocker les points par lesquels passe la ligne.

Score :

La classe Score fait partie du modèle et correspond aux scores de la partie. Il existe deux types de score, le score du joueur et le score de l'ordinateur. La responsabilité de cette classe est donc d'augmenter le score de chaque joueur ("player", "IA") si le coup est valide. Elle gère tout ce qui se rapporte au score. Elle stocke les deux scores dans dans ses instances de classe.

Direction :

L'énumération direction permet de mettre en place toutes les directions qu'il est possible de faire dans les deux modes de jeu 5T et 5D. Cette énumération nous a été très utile dans le développement du code car elle nous a permis de faire toutes les vérifications possibles pour

valider ou non un coup dans une direction particulière. Sa responsabilité se limite à la connaissance de toutes les directions possibles que peut prendre un coup de l'utilisateur.

Vue

MenuView :

Cette classe fait partie des vues. Elle a la responsabilité d'afficher une vue selon les informations que lui envoie le contrôleur. Cette vue affiche le menu. Le menu contient les scores (joueur et ordinateur) qui sont incrémentés à chaque coup valide d'un de ces deux utilisateurs. C'est le contrôleur qui envoie l'information d'un coup valide et que le score doit être incrémenter. Ensuite cette vue affiche et propose les fonctionnalités suivantes par l'intermédiaire de boutons: sauvegarde ("Save") qui permet de sauvegarder l'état encours de la grille, importer ("Import") qui permet d'importer un fichier et de remettre en place une grille sauvegarder (redessine les coups sauvegardés et permet à l'utilisateur de continuité la partie si cette dernière n'est pas terminé), recommencer une partie ("reset"), aide ("help") si l'utilisateur est perdu et ne trouve plus de coup ce bouton lui permettra de trouver un coup, solution ("Solution") permet d'afficher une solution à l'utilisateur. La solution n'est pas la meilleure mais est souvent pas trop mauvaise et la distribution des scores obtenus au cours des différentes parties jouées. Elle possède aussi le bouton ("Distribution") qui montre la distribution des scores dans le mode de jeu actuel. Pour qu'un score s'affiche, il faut que la partie soit réinitialisée.

Grid5TView :

Cette classe fait partie des vues. Elle a pour responsabilité d'afficher l'état actuel de la grille du mode 5T en temps réel grâce aux données qu'elle possède du modèle.

Grid5DView :

Cette classe fait partie des vues. Elle a pour responsabilité d'afficher l'état actuel de la grille du mode 5D en temps réel.

GraphView :

Cette classe fait partie des vues. Elle a pour responsabilité d'afficher l'état d'un graph.

ChoiceView :

Cette classe fait partie des vues. Elle a pour responsabilité d'afficher la fenêtre du choix du mode de jeu par l'intermédiaire de deux boutons. Elle est appelée par le contrôleur Choice qui définit les actions de chacun des boutons.

Contrôleurs

Choice :

Cette classe fait partie du contrôleur. On peut parler du super contrôleur car c'est cette classe qui va permettre de lancer le contrôleur d'une partie en 5T ou en 5D. En effet cette classe à la responsabilité de de faire apparaître une Vue (ChoiceView) qui va proposer les deux modes de jeu à l'utilisateur et selon son choix le contrôleur Choice va lancer le contrôleur d'un des deux modes de jeu.

Distribution :

Cette classe fait partie du contrôleur. Cette classe a la responsabilité de lancer la vue GraphView en lui donnant les informations de Graph. Elle fait aussi le lien avec les grilles car elle reçoit directement l'historique des scores puis crée le modèle et la vue en conséquence.

Game5D :

Cette classe fait partie du contrôleur. Cette classe à la responsabilité d'informer la vue GridView5D à chaque coup pour que cette dernière se mette à jour au niveau de tous les éléments de la structure. Elle a également pour responsabilité d'informer le MenuView afin que le score s'incrémente lorsque le coup proposé est valide.

Game5T :

Cette classe fait partie du contrôleur. Cette classe à la responsabilité d'informer la vue GridView5T à chaque coup pour que cette dernière se mette à jour au niveau de tous les éléments de la structure. Elle a également pour responsabilité d'informer le MenuView afin que le score s'incrémente lorsque le coup proposé est valide.

App

C'est le package qui lance le jeu.

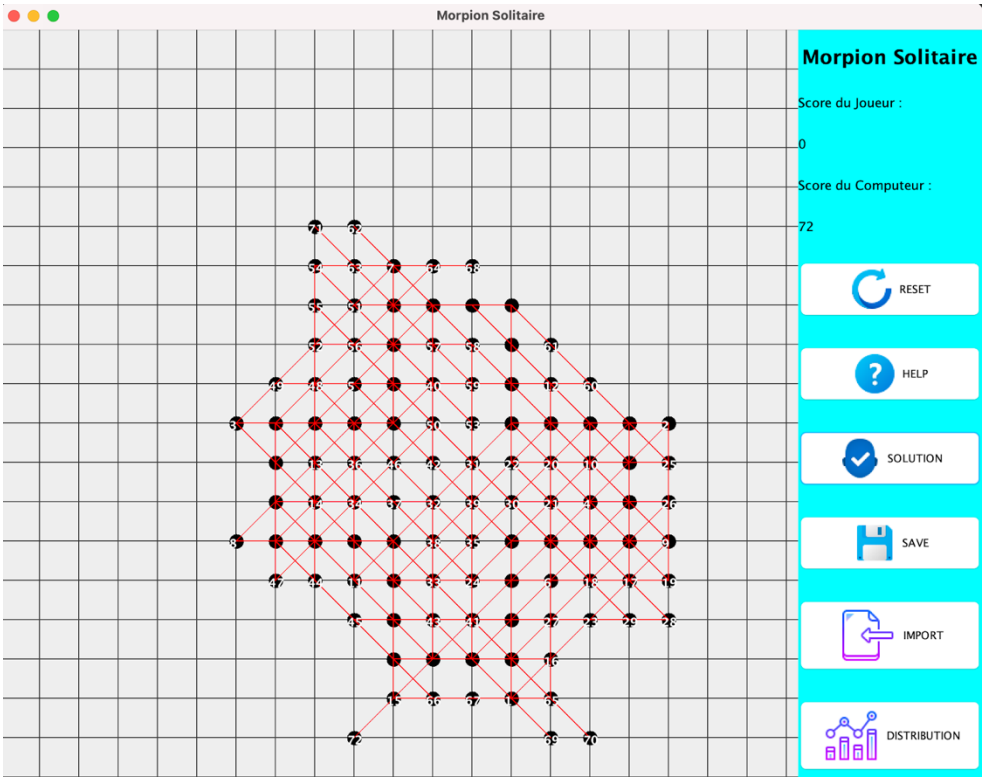
App :

App est la classe qui a pour responsabilité de lancer le jeu. La grid est un objet qui contient des structures données reflétant ce que le plateau sait de lui-même. Il correspond à la feuille de papier n'acceptant que les bons coups. La grid 5T est son pendant mais celui-ci n'acceptant que les bons coups suivant les règle 5T.

Fonctionnalités mises en place :

Afficher grille :

Pour afficher la grille nous avons fait une fonction qui initialise les instances de la classe Grid5D et Grid5T avec la hauteur de la grille, la largeur de la grille et le pas entre chaque intersection. Avec ces informations nous pouvons tracer les lignes horizontales et verticales de notre plateau de jeu. Nous n'avons cependant pas réussi à mettre en place une grille infinie. Nous avons essayé mais sans y parvenir. Notre plateau de jeu est donc d'une taille fixe mais il permet quand même de jouer une partie à plus de 72 coups au minimum.



Afficher croix :

Pour afficher la croix de départ sur la grille de jeu, nous avons choisi de ne pas sauvegarder les 36 points de la croix dans une ArrayList (ou autre structure de données). Nous avons fait le choix de faire une fonction qui va créer un point pour les 36 intersections de la croix de départ. Pour ce faire nous trouvons à l'aide d'une fonction le premier point de la croix et nous initialisons l'attribut de classe ("center") avec ce premier point. Lorsque nous avons ce point une fonction va se déplacer de façon séquentiel (de façon à former une croix) dans la grille afin de remplir un tableau qui va sauvegarder tous les points de la croix et mettre à jour le tableau 2D de boolean qui permet de savoir si un point est dessiné sur une intersection (on met true dans ce tableau quand c'est le cas. Pour dessiner cette croix la difficulté que nous avons rencontrée était de trouver une méthode qui dessine cette croix sans que cette méthode ne soit trop longue. Notre méthode est assez dense car nous dessinons chaque côté de la croix avec une boucle for. Nous n'avons pas trouvé un algorithme qui va dessiner la croix sans passer par plusieurs for.

Afficher point de l'utilisateur :

Cette fonctionnalité nous permet, lors d'un clique utilisateur pour jouer un coup, de capter les coordonnées de l'intersection sur laquelle le joueur a cliqué. On s'est rendu compte que lors d'un clique de l'utilisateur sur une intersection les résultats retournés par e.getY() n'étaient pas très précis. Nous avons donc dû appliquer retraitement sur le résultat retourné par cette fonction afin d'affiner la précision. Avec ce retraitement nous avons des données plus précises sur les coordonnées et nous créons donc un point avec ces coordonnées ensuite nous comparons la distance de ce point avec tous les points possibles de la grille (ArrayList tabCoordonnee) et nous retournons le point le plus proche du point que nous générons avec les coordonnées retournées par e.getX() et e.getY(). La difficulté ici a été de se rendre compte de ce qu'il n'allait pas avec le click au début car lorsque nous voulions placer un point sur une intersection dans la majeure partie des cas le point était placé avec une intersection d'écart. C'est la raison pour laquelle nous appliquons un retranchement à ce que nous retourne e.getY(). La distance que nous avons choisi d'utiliser est la distance euclidienne qui est la référence dans ce genre de problèmes.

Afficher un coup correct :

Pour afficher un coup correct, on a mis en place le concept de "Line" qui est une suite de 5 points. Etant donné que pour afficher une ligne, nous n'avons besoin que de deux points, dans les faits, cette classe n'est souvent remplie qu'à ses extrémités. Les classes Grid contiennent donc une liste de lignes. Celle-ci sera utilisée par la vue pour l'afficher d'une couleur rouge. Le point affiché doit aussi contenir le numéro du coup. Il est donc stocké dans une structure qui contient le point et la direction utilisés. Dans un premier temps, la direction n'était pas stockée, puis dans un second temps, les directions étaient stockées mais nous ne gardions qu'horizontal et non la manière dont l'horizontal a été fait car dans certain cas ils peuvent être fait de deux manières possibles. Nous avons eu des difficultés pour réinitialiser un coup car un coup est lié à plein de structures de données distinctes et qu'il faut pour chacun travailler sur une copie par valeur de ces structures de données. Nous avons eu du mal à voir ça sur certaines structures notamment nos HashMap de HashMap. Pour valider que le coup soit correct, nous avons dû mettre en place des fonctions regardant si le coup était valide selon une direction (4 fonctions check, 1 pour chaque direction). Ensuite, il a fallu dessiner ces lignes. Il existe 8 fonctions draw car nous nous sommes rendu compte que dans nos fonctions nous commencions tout le temps à faire la recherche de points utilisables dans un coup vers la gauche et vers le haut. Maintenant nous avons également des fonctions qui commencent la recherche de points utilisables dans un coup vers la droite et vers le bas. L'adaptation du moteur 5D au moteur 5T a été aussi compliquée pour nous. Car nous n'avions pas compris au début qu'il fallait coder les deux moteurs de jeu. Il a donc fallu que nous comprenions correctement les règles du jeu en 5T pour adapter le code du moteur de jeu 5D en 5T.

Afficher un score :

Pour pouvoir afficher un score nous avons une méthode qui va incrémenter le score d'un joueur ou de l'ordinateur en fonction d'un String passer en paramètre (player ou IA). Si "IA" est passé en paramètre c'est le score de l'ordinateur qui sera incrémenter sinon c'est le score du joueur. Cette méthode qui incrémente le score est appelée dans les deux fonctions qui vont mettre à jour la grille en fonction des coups joués par l'utilisateur ou par l'ordinateur. Si le coup est valide le score s'incrémente.

Réinitialiser une grille :

Cette fonctionnalité permet de recommencer une partie. La méthode associée à cette fonctionnalité réinitialise les structures de données liées à la grille et stock le score dans une ArrayList pour garder l'historique des scores et mettre en place un graphique de la distribution des scores.

Sauvegarder une grille :

Cette fonctionnalité permet à l'utilisateur de pouvoir sauvegarder l'état courant d'une grille dans son système de fichier sous la forme d'un fichier contenant les informations suivantes sur les coups joués (chaque point dessiné) : coordonnées de x ; coordonnées de y ; Direction dans laquelle le point est utilisé.

Importer une grille :

Cette fonctionnalité permet d'importer un fichier sauvegardé d'une grille et de pouvoir continuer à jouer à partir de cette grille si des coups sont encore possibles. Cet import se fait par la lecture de chaque point enregistré dans un fichier de sauvegarde avec la direction qui leur est associée. Pour chaque point du fichier de sauvegarde, on appelle la méthode de mise à jour de la grille. Il est donc possible d'importer une grille sauvegardée et de continuer à jouer dessus.

Donner un coup possible :

Cette fonctionnalité permet d'aider l'utilisateur en cas de blocage. Si le joueur n'arrive pas à trouver un autre coup alors qu'il en reste, cette fonctionnalité va jouer un coup pour l'utilisateur. Pour afficher un coup nous avons mis en place une fonction qui va retourner, parmi tous les points contenus dans la liste des points disponibles et possibles, celui pour lequel il va rester le plus de coups possibles après l'avoir joué. La difficulté que nous avons rencontrée ici c'est le fait que notre algorithme, au début, choisissait le premier point pour lequel il rencontrait un maximum. Ceci était dû à la recherche qui cherchait le maximum de points possible en jouant les points de la liste des points possibles. En effet il peut y avoir plusieurs max dans cette liste mais notre méthode prenait le premier Point qui avait le résultat maximum à cause de notre condition qui nous permet de récupérer le maximum de coups possibles si on joue un coup possible (if(x>max)). Nous avons donc essayé de comprendre l'algorithme NMCS et nous avons aussi essayé de réfléchir sur un algorithme qui permettrait de trouver la meilleure solution. Pour l'instant nous vérifiions s'il y a plusieurs "maximums" égaux si c'est le cas nous stockons tous les points pour lesquels le maximum est atteint dans une ArrayList tampon, nous la mélangeons aléatoirement et nous prenons le premier point de ce tableau.

Donner une solution possible :

Plusieurs algorithmes ont été mis en place. Ils sont détaillés dans la documentation rapport d'expérimentations.

Donner la distribution des scores existants :

Lié à la fonctionnalité reset. A chaque fois que la grille est réinitialisée, le score est ensuite stocké dans une liste qui conserve l'historique. Ensuite cet historique est confié au contrôleur

Distribution qui va instancier le Graph et la GraphView. Le graph va transformer cette liste en liste de point et la vue va la montrer. La difficulté a été de gérer l'axe Y et faire que la fenêtre ressemble au graphique sans utiliser d'autres librairies. Elle a pu être gérée via l'objet Graph.

Donner la possibilité au joueur de choisir la direction de son coup si ce dernier est possible dans plusieurs directions :

Nous avions un moteur qui avait une vision à l'instant T. Nous avions des checks qui étaient performants mais deux manières de dessiner possibles. Nous avons donc entrepris de créer une fonction qui remplirait l'objet direction potentiels pour un point donnée. Plutôt que de dupliquer les checks, on a simulé l'insertion du point en utilisant le draw1 puis le draw2 puis ensuite nous avons comparé les lignes créées. Si elles sont les mêmes, pas de choix, sinon un choix est offert si l'utilisateur n'est pas IA. Ici, malheureusement, le modèle est obligé d'afficher un écran. C'est le seul endroit où la vue se retrouve dans le modèle mais cela est dû au fait qu'elle est nécessaire dans la transformation des données.

Algorithmes de solutions :

Plusieurs algorithmes ont été mis en place. Ils sont détaillés dans la documentation rapport d'expérimentations.

Fonctionnalités non mises en place :

NMCS :

Il faut définir un score et c'est dans cela que nous n'avons pas eu l'intuition de comment le calculer. Vous pourrez avoir plus de détails dans notre documentation sur nos expérimentations.

Grille 5D# et 5T# :

Elle peut être implémenter en changeant la fonction initialiser point. On peut imaginer la possibilité de laisser mettre 36 points n'influencent pas le score et ne mettant à jour que point et tabCross. Le souci est la gestion de sa sauvegarde. Mais c'est une implémentation possible via notre architecture.

Multithreading :

Cette fonctionnalité devrait permettre d'accélérer nos algorithmes de recherches néanmoins, elle n'a pas été implanté dans la version actuelle car dans nos algorithmes nous avons besoin de lui passer l'état de la grille actuelle et nous n'avons pas trouvé le moyen par faute de temps.

Temps de travail : Environ 50h-65h