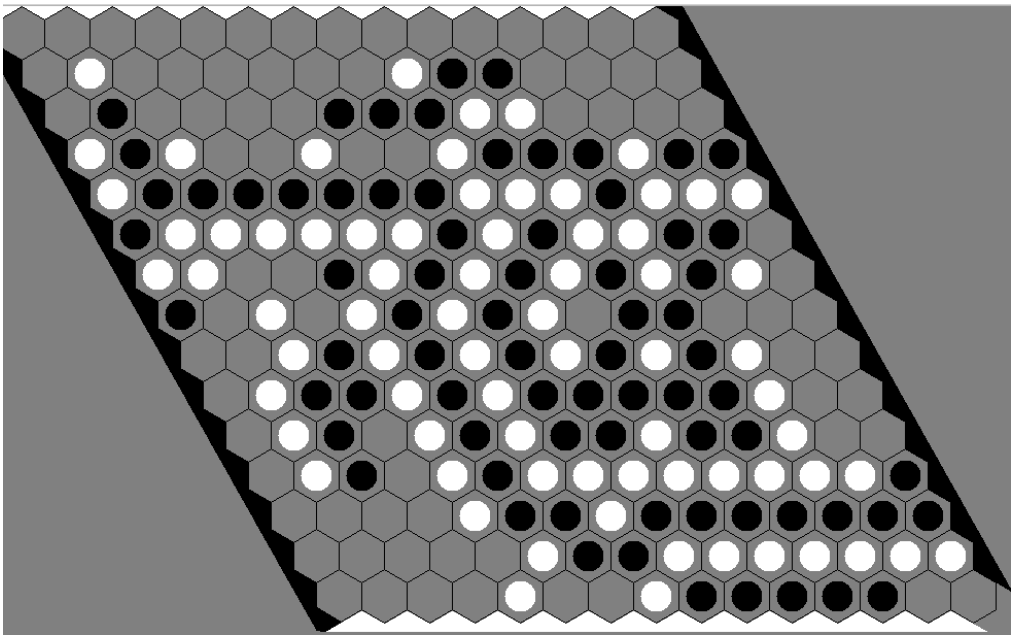
Rapport de Projet :

Jeu de Hex



Université Paul Sabatier

Matthieu PONT

Tom BLOCH

Bence LOVAS

Deuxième année de licence Informatique

Du 27 février au 22 mai 2017

Tuteur Universitaire : M. Vincent DUGAT

Sommaire

Introduction

Dans le cadre de la deuxième année de la Licence Informatique de l'Université Paul Sabatier il nous a été demandé, afin de clore le 4eme semestre, de réaliser un projet en groupe donc l'objectif est de concevoir le jeu de société Hex.

Brièvement, ce jeu se jouant à deux joueurs se déroule sur un plateau pavé d'hexagones. L'objectif de chaque joueur est de relier les deux bords opposés de sa couleur en posant tour à tour un pion sur un des hexagones.

L'étude de ce jeu est source d'inspiration en théorie des jeux, en mathématiques et en intelligence artificielle.

Le projet est divisé en trois grandes parties. Une première, codée en Java, où l'on réalisera une interface utilisateur afin que ce dernier puisse jouer au jeu de Hex. Une deuxième, codée en C, qui s'occupera de tous les calculs liés au jeu. Et enfin une dernière, en SQL, qui représentera une base de données liée au jeu.

Ce document retrace de façon synthétique toutes les activités réalisés durant ce projet et comment nous les avons menées à bien.

Réalisation du jeu de Hex

I – Organisation

Le projet est séparé en 3 grandes étapes (de la V1 à la V3). Uniquement la V1 est séparé en 3 parties (Java, SQL et C) tandis que la V2 et la V3 concernent l'intelligence artificielle du jeu et seront codées en C.

Étant donné que nous sommes trois dans la réalisation du projet nous nous sommes proposé de nous séparer la V1 de la sorte : Tom Bloch réalisera l'interface utilisateur Java, Bence Lovas réalisera la base de données et Matthieu Pont s'occupera des calculs du jeu en C. Nous nous sommes ensuite partagé la réalisation de la V2 et de la V3.

Deux semaines avant la fin du projet nous avions prévu de commencer la V3, la V2 n'étant pas totalement terminée nous avons décidé que Tom Bloch et Bence Lovas commenceraient à programmer la V3 pendant que Matthieu Pont terminerait la V2 en trouvant une fonction d'évaluation pour le minimax.

Bien que nous nous soyons répartis le travail de la sorte, avant que l'un des membres commencent sa partie nous réfléchissions tous ensemble en posant sur papier le problème et en réfléchissant comment le résoudre.

Cela permettait d'avoir différents points de vue sur le problème, de plus en faisant comme ceci nous nous assurons que chacun des membres possède une vue d'ensemble du projet et de chaque module le constituant.

Afin de partager les fichiers et que tous les membres du groupe puissent avoir accès aux différents codes, nous avons choisi d'utiliser « Google Drive » à défaut de « Github ».

En effet, nous avons fait ce choix car utiliser « Github » aurait pu être certes plus pratique, mais aucun de nous ne connaissait cet outil, sa prise en main nous aurait peut-être fait perdre du temps qui aurait pu être précieux pour le projet.

Lorsque nous avions commencé le projet, nous avions encore des cours TDs ainsi que des cours TPs. Nous nous réunissions alors tous les jeudi afin de préparer le sujet en amont :

En effet, au tout début, nous ne savions pas quelle type de structure utiliser, nous avons dû nous concerter et faire des schémas afin de mieux comprendre ces structures. Les premières entêtes de fonctions ont étés écrites pour que tous les membres du groupes puissent savoir quels paramètres étaient pris en entrée et qu'est-ce que les fonctions renvoyaient.

Nous avons également fait le MCD de la Base de Données afin de ne rien oublier et que le codage puisse se faire dans de bonnes conditions.

Au fur et à mesure que les semaines passaient, les séances en groupes et les réunions ont étés plus fréquentes, en salles de TPs particulièrement. En effet, nous pouvions nous entraider en cas de problèmes.

II – Module Java

Le module Java peut-être séparé en trois parties distinctes. Il était préférable de commencer à travailler sur l'affichage du plateau pour en évaluer la difficulté et vérifier le degré de faisabilité de notre interface.

2.1 Interface Graphique

Après avoir fait part de notre intérêt pour l'interface graphique, nous avons mené quelques recherches sur internet. Notamment, pour assimiler les fondamentaux (ouvrir une fenêtre, ajouté un composant, dessiné un graphique…).

2.1.1 Affichage du plateau

Au bout de quelques semaines, nous avons réussi à afficher un premier plateau (voir bibliographie). Pour gérer l'affichage de façon dynamique nous avons du apporter quelques modifications au code. Ici chaque hexagone se voit attribuer un numéro qui lui est propre. Cette numérotation nous permet de mettre en évidence l'hexagone survolé par la souris et de calculer les coordonnées d'un pion, lorsque l'on doit le jouer.

Attention : La dimension d'un hexagone est fixée en nombres de pixels, ce qui nous empêche de redimensionner proportionnellement la fenêtre et ses composants.

2.1.2 Affichage des options / Informations

Ensuite, nous nous sommes intéressé aux différentes « options » proposées dans le jeu. On s'est légèrement aidé du logiciel « NetBeans » pour développer le menu principal. C'est ce qui nous a permis d'être plus autonome par la suite (par exemple : comprendre et maîtriser la syntaxe des MouseListeners, le positionnement des boutons, etc.).

Pour personnaliser légèrement notre interface, nous sommes allé chercher sur internet, des icônes matérialisant les différents choix possibles lors d'une partie (sauvegarder, recommencer…). Nous avons aussi intégré ces options dans la barre des menus, en plus des rubriques « Règles » et « A propos » qui renseignent l'utilisateur sur le logiciel.

Nous avons consulté des exemples, en explorant les logiciels installés sur notre machine. Et nous nous sommes renseigné sur les notions d'OpenSources, Copyright… Ce qui nous a emmené à déposer un Copyleft sur le site http://fr.creativecommons.org. Nous nous sommes renseigné pour apposer le caractère spécial sur notre logiciel. Petite anecdote, bien qu'il n'y ait pas en tant que tel de caractère copyleft dans le répertoire d'Unicode faute qu'il ait jamais été proposé, on peut le construire à partir des caractères lettre latine minuscule c réfléchie (U+2184) et cercle englobant (U+20dd).

2.2 Interactions homme-machine

Les interactions homme-machines définissent les moyens et outils mis en œuvre afin qu'un humain puisse contrôler et communiquer avec une machine. Nous aimerions que le jeu soit facile à prendre en main ou plus généralement adapté au contexte d'utilisation.

2.2.1 Prise en main

Par exemple : la principale contrainte que nous nous sommes fixée laisse l'opportunité au joueur d'annuler son coup. Nous avons donc choisi de faire intervenir le « clic droit » pour valider un coup et ainsi laisser le temps à l'utilisateur d'évaluer sa décision ou changer d'avis si besoin. Il était d'autant plus intéressant de mettre en évidence avec un hexagone bleu, le dernier pion placé sur le terrain. Notamment lorsqu'un joueur reprend une partie qu'il n'a pas terminé et qu'il cherche à se remettre dans le contexte (il peut finalement changer d'avis.).

Par la même occasion, nous avons clarifié la position de la souris et l'organisation du plateau: comme dit précédemment l'hexagone survolé par celle-ci est entouré d'un trait rouge lorsqu'il n'est pas déjà occupé par un pion.

2.2.2 Interactivité

Depuis le début du projet, nous cherchons à ce que l'utilisateur ait toujours le choix. On a donc cherché à développer un maximum les opportunités que nous avions. Sans oublier de laisser au joueur la possibilité de revenir sur ses décisions, en générant une boite de confirmation ou en faisant intervenir un bouton « Retour ».

Rien qu'à l'idée de pouvoir recommencer une partie en cours, nous avons choisi de créer deux classes différentes pour séparer l'IHM et l'affichage de la grille. L'affichage de la grille se contente de communiquer les coordonnées le concernant (les icônes correspondantes aux différentes options, le numéro de l'hexagone survolé, etc.) à l'IHM (un de ses attributs) qui quant à lui contient le MouseListener et les pions affichés pendant la partie. Ainsi, lorsqu'un joueur décide de recommencer la partie, il suffit de faire intervenir un removeAll() dans l'IHM (Ce qui revient à supprimer tous les composants du JPanel.).

Malheureusement, après confirmation dans la boite de dialogue il fallait sortir la souris de la fenêtre et revenir à nouveau sous peine de ralentissement lorsqu'on cliquait et qu'on cherchait à continuer la partie. C'est pendant la rédaction du bilan que nous avons compris qu'il suffisait de faire un getparent().repaint() pour renouveler les MouseListeners.

La mise en place d'un attribut IHM dans la classe « Afficher Grille », pose soucis lorsqu'on lance une partie contre l'intelligence artificielle. L'affichage du plateau se retrouvant obligé de construire l'IHM, avant de terminer son exécution, il nous oblige à attendre que l'IA joue pour voir la partie commencée. Nous avons essayé d'appelé la fonction « Initialiser_ihm » depuis la classe parent , mais rien n'y fait. Nous nous pencherons sur le problème pour la prochaine mise à jour.

2.3 Programme Principal

L'existence d'une interface graphique, l'utilisation des MouseListeners, mais aussi la multitude de scénarios possibles impliquent quelques changements dans notre manière de voir le programme principal. Il dépend finalement des initiatives prises par l'utilisateur à partir d'une seule et même classe au départ (le menu principal).

2.3.1 Organisation

Quand il ne s'agit pas d'actualiser automatiquement le contenu de la page, les différents scénarios sont illustrés dans le MouseListener et conditionnent la partie en question. Une

partie peut prendre différentes formes (Joueur Vs Joueur, Joueur Vs IA, ce qui implique aussi différents messages affichés à l'écran, avec différents contenus, etc.). Néanmoins nous avions pour objectif de les rendre finalement réalisable dans une seule et même classe « IHM ». Comme dit précédemment, certaines informations clé conditionnent le contexte d'une partie, ce sont donc ces éléments que nous avons utilisé pour paramétrer les différents constructeur. L'interface Java/C nous permet aussi d'aller chercher des valeurs en mémoire et ceci, peu importe la classe dans laquelle se situe l'utilisateur. Par exemple, c'est elle qui conserve la taille du plateau, pendant le choix de la difficulté.

2.3.2 Interface Java/C

L'interface Java/C nous permet d'appeler des fonctions codés en C, à l'intérieur d'un projet java. Nous appris à générer trois types de librairies différentes : « .so », « .dll » et « .dylib » correspondant respectivement aux différents systèmes d'exploitations: Linux, Windows, et OS X. Malheureusement, nous avons eu quelques problèmes avec l'Interface Java/C. Nous n'avons pu testé le programme principal, mais sur une seul machine uniquement. Il se peut que ce problème là nous ait retardé quelques peu sur la fin du projet.

La déclaration du plateau en tant que variable globale donne un accès direct aux dimensions du plateau ou aux informations concernant l'intelligence artificielle, et ceci peu importe la classe dans laquelle nous nous trouvons (Voir: init_plateau(), get_plat_taille(), ou encore get_plat_ia()…). Ce procédé nous a permis de réduire le nombre de paramètres dans les différents constructeurs.

2.3.3 Lex exécutables

Au final, nous étions intrigué à l'idée de générer un exécutable. Après quelques recherches sur le net, nous avons expérimenté les scripts .SH et .BAT (respectifs au Shell et à Windows). Pour plus de compatibilité, nous avons cherché à produire un .JAR qui utilise lui, la JVM et nous avons fini par télécharger un logiciel comme JExeCreator (disponible en version d'essai 30 jours) ou le plugin Launch4j sur NetBeans afin de convertir ce dernier en .EXE.

Chaque procédé avait ses propres avantages, le premier nous permet de renseigner à l'utilisateur un lien où il peut se procurer la JVM, dans le cas où elle ne serait pas installée. Tandis que le deuxième lui, nous permet de définir une icône.

III – Module C

Le module C est séparé en trois parties distinctes. Pour le bon fonctionnement du programme la première doit obligatoirement être réalisée.

Étant donné que nous allons utiliser des listes de divers éléments nous avons pris le choix d'implémenter une liste de `void *` (liste générique).

3.1 Programme de base (V1)

3.1.1 Gestion du plateau

Pour gérer le plateau de jeu nous avons décidé de créer un tableau à deux dimensions dynamique qui est initialisé lors de la création du plateau en fonction de la taille choisie. La taille du tableau reste inchangée jusqu'au moment où une autre partie débute.

Chaque case du tableau contient une valeur (0 pour vide, 1 pour pour un pion du joueur blanc et 2 pour un du joueur noir) et un pointeur vers son sommet correspondant dans le graphe réduit (cf. 2.1.2).

Nous gérons en plus un historique de partie qui garde en mémoire tous les coups joués depuis le début de la partie, cela nous sert principalement pour la sauvegarde afin de respecter la forme de fichier donné dans le sujet (un exemple est référencé en *Annexe 3.1*).

Il y a eu peu voire pas de difficultés au niveau de ce module.

3.1.2 Gestion du graphe réduit et des groupes

Un **groupe** est un ensemble d'hexagones adjacents occupés par des pions d'un même joueur. Un **graphe** est une structure de données abstraite où, dans notre cas, chaque hexagone est représenté par un sommet qui est relié à ses voisins par des arrêtes (voir *Annexe 3.2* pour un exemple de graphe). Un **graphe réduit** est un graphe où l'on remplace les groupes de chaque joueur par un sommet unique.

a) Gestion du graphe réduit

Notre sommet contient sa valeur (i.e sa couleur, identique à la/aux case(s) du tableaux correspondantes) et une liste qui représente ses sommets voisins, nous avons repris le principe des listes d'adjacence utilisées dans les graphes.

Le graphe réduit nous est utile pour détecter la victoire d'un joueur. En effet nous créons 4 sommets en plus des autres qui représentent les bords du terrain (deux bords blancs et deux noirs). Une fois qu'un sommet (donc un groupe) d'un joueur possède dans sa liste d'adjacence les deux bords de sa couleur alors ce joueur a gagné.

Le principal objectif du module Gestion du graphe réduit est donc de « fusionner » des sommets voisins qui appartiennent au même joueur. Nous appelons ce processus lors de la pose d'un nouveau pion, nous vérifions alors dans le graphe réduit si ce nouveau pion est adjacent à des sommets de la même couleur et si c'est le cas nous fusionnons tous ces sommets en un seul. Nous prenions soin d'actualiser la liste des voisins de tous les voisins des sommets en question pour remplacer le/les ancien(s) sommet(s) par le nouveau.

b) Gestion des groupes

La gestion des groupes consiste à stocker tous les groupes d'un joueur. Pour ce faire nous stockions une liste de groupe pour chaque joueur. Cette liste est constitué d'une structure contenant un sommet (qui représente un groupe d'un joueur) et les cases correspondantes à ce sommet dans le plateau.

Ceci nous servait dans un premier temps lors de la fusion de plusieurs sommets correspondants à des groupes (donc à plusieurs cases dans le plateau) afin de remplacer l'ancien sommet des cases correspondantes dans le plateau par le nouveau.

Nous nous sommes ensuite rendu compte qu'il était plus simple (notamment pour la gestion des ponts) de mettre la liste des cases correspondantes à un sommet directement dans ce sommet (au lieu de stocker les cases d'un sommet uniquement quand il devient un groupe et ceci dans une autre structure de données spéciale aux groupes). Cela a aussi simplifié le code.

3.2 Minimax (V2)

3.2.1 Principe

Le principe de l'algorithme du minimax est de passer en revue toutes les possibilités pour un nombre exhaustif ou limité de coups afin de trouver une stratégie gagnante. Cela revient en quelque sorte à prévoir à l'avance des coups afin de trouver l'enchaînement de coup qui nous fera gagner.

Nous partons du principe qu'un joueur cherche à minimiser ses pertes tout en maximisant celles de l'adversaire. Par ce principe nous pouvons conclure que lors du passage en revue des différentes possibilités nous devrons prendre en compte le « pire » coup lorsque c'est l'adversaire qui joue (il cherche à maximiser nos pertes) et le « meilleur » coup lorsque c'est nous qui jouons (on minimise nos pertes) : c'est le principe du minimax.

Pour ce faire nous attribuons une valeur à chaque nœud. Si le nœud est une feuille de l'arbre où le joueur adverse gagne on lui attribue -1, 1 si nous gagnons et sinon nous prenons le minimax le plus grand de ses fils lorsque le nœud en question est un nœud où l'adversaire joue, ou le minimax le plus petit de ses fils si c'est un nœud où l'on vient de jouer.

Étant donné que notre fonction d'évaluation ne retourne que deux valeurs (1 ou -1) la valeur maximale est donc 1 et minimale -1. Dans un nœud de l'arbre, lorsque nous cherchons la valeur maximale de ses fils nous pouvons alors nous arrêter dès que l'on trouve un 1 car nous ne pourrons pas trouver plus grand que 1 (et pareil pour -1 lorsque l'on cherche la valeur minimale). Cela reprend le principe de l'élagage Alpha-Beta, mais est beaucoup plus simplifiée étant donné que nous n'avons que deux valeurs.

3.2.2 Réalisation

Le minimax est symbolisé par un arbre de jeu où la racine correspond à la situation du jeu actuelle. Les fils d'un nœud représentent toutes les possibilités existantes à partir de ce nœud.

L'arbre de jeu peut donc avoir une profondeur maximale de NxN avec N : taille du plateau (car il peut y avoir au maximum NxN coups joués) et au maximum (NxN)! nœuds (la racine de l'arbre au début du jeu peut avoir NxN fils, chacun de ses fils NxN - 1 puis chacun de ses autres fils

NxN - 2 et ainsi de suite).

On se rend vite compte que le nombre de nœuds à visiter peut vite devenir très important, voire trop important pour pouvoir faire une recherche exhaustive, en particulier pour des plateaux 4x4 ou plus grands ($2{,}1 \times 10^{13}$ nœuds pour un plateau 4x4).

Voici l'algorithme de base (simplifié) que nous avons adopté pour le minimax :

```
minimax(noeud a, int num_fils) :
      On recherche le (num_fils+1)ème prochain coup
      On simule la pose de ce coup
      Si un des joueurs gagne
            On met à jour la valeur de minimax de ce noeud (comme dit plus haut)
      Sinon
            i = 0 ;
            tant que a.minimax != cond ET i<a.nbfils_max
                  a.minimax=minimax(a.fils[i],i)
                  ++i
            fin_tant_que
      fin_si
      retour a.minimax
fin
```

avec cond = 1 si nœud adverse
-1 sinon (si c'est un de nos nœuds)

3.2.2 Problèmes rencontrés

Il nous a été demandé de tester dans un premier temps notre algorithme sur un plateau 2x2. Nous avions au début des problèmes de mémoire, mais à l'aide des outils `gdb` et `valgrind` nous avons pu les résoudre plus ou moins facilement.

Nous nous sommes rendus compte à l'aide de ces deux outils que nous ne pensions pas à attribuer la valeur `NULL` à un pointeur après l'avoir libéré de la mémoire (`free`). Ce qui faisait que lorsque nous refaisions une allocation de mémoire et que cette dernière utilisait la zone ou une partie de la zone de l'ancien pointeur que nous avions libéré alors `valgrind` nous indiquait un « `invalid read` » ou « `invalid write` » sur la zone en question par exemple.

Nous avions eu le même problème lorsque nous faisions une allocation de mémoire d'une structure et que l'un des champs de cette structure n'était pas initialisé. Si dans le programme il venait un moment où ce champ était testé alors valgrind nous disait la même erreur (notamment pour l'élément de la sentinelle de notre liste qu'il fallait initialiser à `NULL` car l'itérateur pouvait être amené à tester cet élément).

Bien que ces erreurs ne nous provoquait pas (la plupart du temps) d'erreurs de segmentation (donc un arrêt du programme) nous avons tout de même décidé de corriger toutes les erreurs que `valgrind` nous indiquait afin d'avoir une mémoire plus optimisé et plus « propre ».

Nous avons ensuite testé notre algorithme pour un plateau 3x3, il y eut quelques soucis mineurs mais comme précédemment nous avons utilisé les deux précédents outils pour nous aider.

Le principal problème a été lorsque nous avons testé notre algorithme sur un plateau 4x4 nous avons rapidement vu que l'algorithme mettait beaucoup trop de temps. Nous avons donc décidé dans un premier temps d'arrêter l'algorithme lorsque son temps d'exécution dépassait une

constante prédéfini (5 secondes).

De plus nous avons réfléchi à un moyen permettant au minimax de chercher directement dans ses fils les plus rentable. Notre première version de la fonction qui recherche le prochain coup (le coup joué que symbolise le nœud actuel de l'arbre)

Une autre option se propose à nous, elle consiste à visiter l'arbre du minimax mais uniquement pour une profondeur maximale donnée et d'attribuer une valeur de minimax aux feuilles de l'arbre qu'un joueur ait gagné ou non, pour ce faire nous avons besoin d'une fonction d'évaluation qui nous permet de dire si le plateau actuel non gagnant est plus favorable à tel ou tel joueur.

3.2.3 Fonction d'évaluation du jeu de Hex

Nous avons réfléchi fort longtemps afin de trouver des calculs qui par rapport aux groupes, aux ponts etc. permettraient de déterminer si le plateau est plus favorable pour un joueur ou pour l'autre.

On peut penser qu'un joueur est plus favorable de gagner si son nombre de pions à poser afin de compléter une chaîne gagnante (chaîne d'hexagones reliant ses deux bords) est plus petit que celui de l'adversaire. Nous avons au départ chercher un moyen de trouver une chaîne semi-gagnante (presque gagnante) puis de calculer le nombre de pions restants avant qu'elle ne devienne réellement gagnante.

Le problème d'une telle technique est qu'elle ne prend pas en compte les autres chemins possibles, et perd donc tout son sens lorsque l'adversaire peut bloquer la réalisation de cette chaîne gagnante avec la pose d'un seul pion par exemple (ce qui fausserait donc tout calcul car la chaîne semi-gagnante peut être trop facilement contrée).

a) Réseau de résistances de Shannon

Nous avons fait beaucoup de recherche afin de trouver une fonction d'évaluation correcte et nous sommes tombés sur la machine analogique de Claude Shannon inventée pour le jeu de Hex. Cette machine consiste en un réseau de résistances électriques qui représente le plateau du jeu de Hex. Elle est capable de déterminer un coup à jouer en fonction des potentiels électriques au sein du circuit.

Nous ne sommes pas lancés dans la réalisation d'une telle machine mais nous en avons repris les principes pour concevoir une fonction d'évaluation.

Plusieurs documents nous ont donnés certaines explications quant à la réalisation d'une fonction d'évaluation reprenant ce principe. Hélas nous n'avons trouvé à aucun endroit une méthode claire et précise de ce qu'il fallait vraiment faire afin de concevoir un tel système. Tous les documents parlant de cette méthode en parlaient brièvement et n'expliquaient pas tous les détails des calculs.

Nous avons donc regrouper tout ces documents (disponible en bibliographie) et fait plusieurs simulations (notamment des calculs sur papier) afin de développer une méthode pour réaliser un tel système. Ci-dessous nous faisons une synthèse de toutes les recherches que nous avons faites et de la méthode qui en est ressortie.

Nous allons considéré deux réseaux de résistance (un pour chaque joueur) auxquels on applique une tension sur les deux bords d'un joueur. L'objectif est de déterminer la résistance totale du circuit de chaque joueur. En fonction des deux valeurs de résistance nous pourrons déterminer le joueur le plus favorable sur ce plateau (la valeur de la résistance totale représente en fait à quel point le courant circule « bien » entre les deux bornes).

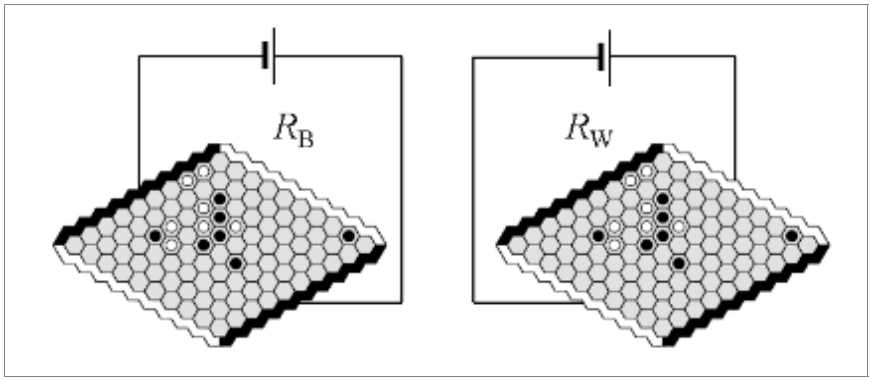


Figure 3.1 : Circuits électriques des joueurs noirs et blancs

Dans notre programme les bords sont inversés par rapport à cette image.

Selon les lois de Kirchhoff sur les courants électriques la résistance totale prend en compte non seulement le chemin le plus court mais aussi tous les autres chemins connectant les bords, leurs longueurs et leur intersections.

Pour expliquer la mise en place d'un tel réseau de résistance nous prenons le circuit du joueur noir pour exemple mais la méthode est la même pour l'autre circuit.

Nous allons maintenant établir le réseau de résistance. Pour chaque case du plateau nous associons une valeur de résistance de la manière suivante :

r(c) = 1 si la case est vide

r(c) = 0 si la case est occupée par un de nos pions

r(c) = +∞ si la case est occupée par un pion adverse

Quelques petits rappels d'électroniques, une résistance de valeur +∞ est considéré comme un fil ouvert, c'est à dire un fil où le courant ne passe pas. Ce qui corrobore avec le jeu de Hex, nous n'allons pas créer de chaîne gagnante à l'aide des pions adverses, le courant qui détermine les chaînes gagnante ne doit donc pas passer par ces nœuds du circuit.

De plus une résistance de valeur 0 est un fil qui laisse passer totalement le courant. Cela corrobore une fois de plus avec le jeu de Hex, car nous allons créer une chaîne gagnante à l'aide de nos pions, donc le courant doit passer aisément au travers des nœuds représentant nos pions.

Nous pouvons définir une matrice qui contient toutes ces valeurs (il n'est pas forcément nécessaire de stocker une matrice pour chaque circuit car il suffit de remplacer les 0 et +∞ d'un circuit pour trouver le circuit du joueur opposé). Pour représenter la valeur de +∞ nous avons décidé d'utiliser la valeur -1 et de faire nos traitements en fonction.

Un tel circuit électrique est beaucoup trop complexe pour calculer sa résistance totale de

manière usuelle (avec les lois sur les résistances en parallèles, en séries, en connexion delta etc.). L'objectif sera donc de calculer le courant sortant du circuit afin de déterminer avec la loi d'Ohm la résistance totale.

Loi d'Ohm :

U = RI où U est la tension appliquée aux bornes

R la résistance totale

I le courant du circuit

Dans un premier temps nous allons devoir calculer la tension associée à chaque nœud du réseau de résistance.

La loi des nœuds de Kirchhoff stipule que la somme des courants entrant dans un nœud est égale à la somme des courants sortants de ce nœud. Nous pouvons voir en *figure 3.2* comment circule le courant pour un hexagone du jeu de Hex.

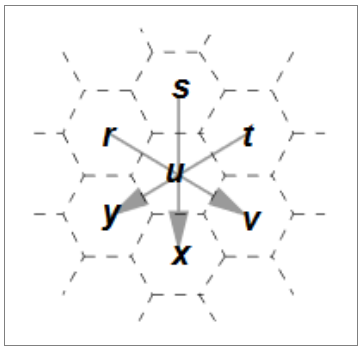


Figure 3.2 : Illustration du courant pour un hexagone du jeu de Hex.

Ainsi nous pouvons déduire **l'équation** suivante **pour un nœud *u*** :

$$\frac{V(r)-V(u)}{R(r,u)}+\frac{V(s)-V(u)}{R(s,u)}+\frac{V(t)-V(u)}{R(t,u)}+$$
$$\frac{V(u)-V(v)}{R(u,v)}+\frac{V(u)-V(x)}{R(u,x)}+\frac{V(u)-V(y)}{R(u,y)} = 0,$$

où V(*i*) est la tension au nœud *i* et R(*i*,*j*) la résistance entre le nœud *i* et le nœud *j*. Cette résistance se calcule avec la formule suivante :

R(*i*,*j*) = r(*i*) + r(*j*)

Cette équation est le cœur de la méthode. Avec la loi d'Ohm nous avons I = U/R :

La première ligne symbolise donc les 3 courants entrants au nœud *u* et la deuxième représente l'opposé des 3 courants sortants du nœud (selon les lois de Kirchhoff leur somme est égale à 0).

Les 6 membres de l'équation représente la relation du nœud *u* avec ses 6 nœuds voisins. Par exemple :

- *r* entre dans *u* donc la formule associée est : $\frac{V(r)-V(u)}{R(r,u)}$

- *y* sort de *u* donc la formule associée est : $\frac{V(u)-V(y)}{R(u,y)}$ (en fait on peut dire que *u* entre dans *y*, cela revient au même)

Nous devons établir ce type d'équation pour chaque nœud représentant une case vide du plateau, c'est à dire chaque case qui a pour valeur de résistance r(*c*) = 1.

Nous devons faire attention au sens du courant en fonction du circuit que l'on traite. Nous nous basons sur un plateau comme en *Annexe 3.3*.

Nous traitons le circuit noir dont les deux bornes sont les bords gauche et droite du plateau avec un courant circulant de la gauche vers la droite comme sur la *figure 3.3*. Le sens du courant pour le circuit blanc va de haut en bas et est présenté sur la *figure 3.4*. Nous devons appliquer les formules ci-dessus en respectant le même sens de courant pour chaque nœud.

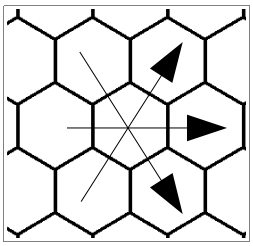


Figure 3.3 : Sens du courant pour le circuit noir

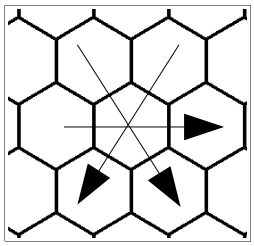


Figure 3.4 : Sens du courant pour le circuit blanc

Lorsque l'on détermine les équations pour un nœud *u* :

- Si ce nœud est adjacent à un nœud *c* ayant pour résistance r(*c*) = +∞ nous ne devons pas faire apparaître dans l'équation du nœud *u* la relation entre *u* et *c* car le courant ne passe pas à travers la résistance de +∞ (la valeur de résistance R(*u*,*c*) = r(*u*) + r(*c*) = +∞ donc cela revient à faire une division par +∞ donc le terme tend vers 0).
- Si le nœud *u* est adjacent à un bord représentant les bornes du circuit alors nous ne devons faire apparaître qu'une seule fois dans l'équation la relation entre *u* et le bord (même si la *figure 3.3* ou *3.4* indique que le nœud serait en quelque sorte « deux fois » adjacent au même bord). On donne la tension V(*bord*) du bord :

  - V(*bord*) = *tension* si cette borne est le bord haut ou le bord gauche du plateau (nous avons pris une tension de 5V, mais à priori, peu importe la valeur tant qu'elle est la même pour les deux circuits, cette valeur représente la borne + du circuit).

  - V(*bord*) = 0 si cette borne est le bord bas ou le bord droit du plateau (représente la borne - du circuit, la masse).
- Si le nœud *u* est adjacent à un nœud *c* ayant pour résistance r(*c*) = 0 et si il existe une chaîne de nœud(s) *i* ayant chacun pour résistance r(*i*) = 0 reliant le nœud *c* et une borne du circuit alors nous prenons

- V(c) = *tension* si la chaîne relie la borne +

- V(c) = 0 si la chaîne relie la borne –

La chaîne ne peut pas relier les deux bornes sinon cela veut dire qu'elle est gagnante. Si la chaîne ne relie aucune borne nous lui attribuons aussi V(c) = 0.

– Il n'y a pas de règle particulière si le nœud u est adjacent à un nœud c ayant pour résistance r(c) = 1.

Une fois toutes les équations trouvées nous avons n équations à n inconnues (où n représente le nombre de case vide du plateau). Chaque inconnue V(i) $0 \leq i < n$ représente la tension à un nœud i.

Nous pouvons modéliser le système d'équation par une matrice correspondant aux coefficients V(i) et une matrice correspondant aux constantes de chaque équation.

Pour résoudre ces équations nous avons utilisé un algorithme utilisant la méthode du pivot de Gauss. Là est une faille du système que nous avons développé, nous utilisons un algorithme naïf qui ne résout pas le problème dû à un 0 sur la diagonale (cela amène à une division par 0). Ce qui signifie que notre calcul marche rarement lorsque le nombre d'équation devient grand (car il y a beaucoup plus de chance d'avoir un seul 0 sur la diagonale).

Nous avons conçu un léger programme tentant de ne laisser aucun 0 sur la diagonale. Malheureusement nous n'avons pas eu le temps de concevoir un programme très performant donc il arrive certains cas où la résolution des équations ne marche pas.

Une fois le système d'équations résolu nous avons fait le choix de vérifier la validité de la loi des nœuds en vérifiant qu'à chaque nœud le courant entrant est bel et bien égal au courant sortant. Cela permet de s'assurer que le principe de base de ce système est validé. Il faut ensuite prendre la valeur maximale du courant parmi tous les nœuds en tant que courant du circuit.

Nous avons maintenant tous les éléments en main pour utiliser la loi d'Ohm afin de calculer la résistance totale du circuit. La formule, tirée de la loi d'Ohm, est la suivante:

$$R = U/I$$

Le joueur qui a la valeur de résistance la plus faible (donc qui a le circuit où le courant passe le mieux entre ses deux bornes) est celui qui est le plus favorable sur le plateau.

Sur plusieurs plateaux de tests nous nous sommes rendus compte que le calcul était correcte et semblait bel et bien déterminer quel joueur est le plus favorable. Il va sans dire que nous sommes fière d'avoir réussi à concevoir une fonction d'évaluation de la sorte, même si certains points doivent encore être améliorés.

Cette méthode est expliquée fort vaguement sur les différents documents trouvés, il a donc dû faire un très gros travail de recherche et de mise sur papier du problème afin de déterminer la méthode exacte pour arriver à ce résultat (il aurait été contre-productif de commencer à programmer un système dont nous n'étions pas sur de sa validité, surtout pour un système de cette ampleur).

Ce système peut être amélioré en trouvant un meilleur algorithme de résolution de système d'équations linéaires. Nous n'avons pas eu le temps d'approfondir plus là-dessus.

De plus nous avons remarqué lors de nos tests que certains nœuds du réseau de résistances pouvaient être ignorés car ils étaient trop isolés dans le circuit des principales chaînes gagnante. Leurs équations pouvaient donc ne pas être prises en compte et ainsi simplifier le système.

Nous pensons avoir respecté les principes de l'électricité dans la réalisation de cette méthode. De plus la vérification de la loi des nœuds après la résolution du système d'équations appuie le fait que la méthode est correct. Néanmoins si vous remarquez des incohérences, si vous avez des remarques, des questions ou des idées pour améliorer les performances de ce système n'hésitez pas à nous contacter à l'adresse suivante: *matthieu.pont@hotmail.fr*.

Don't hesitate to send us an e-mail if you need an english translation of this method.

3.3 Gestion des ponts (V3)

Pour gérer les ponts de l'IA dans le jeu, nous avons décidé d'implémenter une liste de pont pour chaque joueur, nous avons déclaré cette liste en variable globale. Chaque pont de la liste des ponts contient des pointeurs vers 2 sommets ainsi que le nombre de cases constituant ce pont . Le comportement de l'IA va être influé par ces ponts lors de la V3.

/* Le comportement de L'IA avec les ponts est le suivant :

Si l'adversaire essaie de casser un pont de l'IA, le nombre de cases de son pont dans la liste_pont passera à 1 et l'IA rejoindra ses 2 sommets.

Si l'adversaire n'essaie pas de casser un pont de l'IA, l'IA regarde les ponts de son adversaire si jamais il y a un pont à 1, si oui il joue , sinon il applique le comportement par défaut!

*/

Deux sommets donnent naissance à un « pont » lorsqu'ils sont séparés par deux chemins différents (un chemin étant représenté par un seul et même hexagone qui lui est propre.). Ils laissent l'opportunité au joueur propriétaire du pont d'anticipé d'autres coups, sachant qu'il est sur de toujours pouvoir connecté ces deux sommets à partir du moment ou le joueur adverse commencera à prendre un des deux chemin possibles.

3.3.1 Création des ponts

Pour calculer les ponts, le plus intéressant semblait être de parcourir la liste des groupes et de comparer à chaque tour de boucle, la totalité des voisins avec les voisins du sommet donné. Nous voulions implémenter plusieurs fonctions. Une pour gérer la création et l'actualisation d'éventuelles anciens ponts après la fusion de deux sommets. Puis une pour créer des ponts si un pion posé y donne naissance sans avoir fusionner avec d'autres sommets. Ce qui nous évite de parcourir la liste des ponts inutilement dans ce cas.

Nous avons choisi de recalculer la totalité des ponts pour un « joueur1 », lorsqu'il fusionne deux sommets parce que le nombre de cases pouvant relier les deux sommets à d'anciens ponts peut

toujours changer et être supérieur à deux. A l'inverse, pour vérifier que son adversaire n'ait pas pris une case correspondante à l'un de ses ponts, nous consultons à chaque fois qu'il joue les voisins du pion en question pour identifier différentes paires de sommets et décrémenter le nombre de cases dans le pont correspondant (et aussi supprimer les coordonnées du chemin obstrué). On va par conséquent le chercher dans la liste des ponts du « joueur1 ».

Bien sûr, lorsque le nombre de case arrive à 0 pour un pont donné, il est supprimé de la liste et nous libérons l'espace mémoire précédemment occupé.

3.3.2 Comportement de l'IA

L'intelligence artificielle utilise la liste des ponts assez simplement, elle joue dans un pont quand il n'est plus composé que d'une seul case. Il peut arrivé qu'elle joue dans les ponts de l'adversaire. Néanmoins, elle consulte tout d'abord la liste des ponts la concernant. Cette démarche, répond parfois a des cas très particuliers. Notamment lorsque deux joueurs, jouent leur pions à coté les uns des autres.

De part cette implémentation, nous nous sommes rendu compte qu'elle pouvait détecter une stratégie gagnante, dans certains cas. Il nous revient de la calculer.

3.3.3. Stratégie gagnante

Lorsque qu'un sommet est relié à sa base, directement ou par l'intermédiaire d'un pont mais qu'il est aussi séparé de l'autre base par l'intermédiaire d'un ou plusieurs ponts, le joueur tiens une ou plusieurs stratégies gagnante, avec un nombre de coups déterminé. L'adversaire, lui ne peut normalement pas remporter la partie. Sur un plateau 5x5 ou dans une partie relativement avancée le MiniMax serait capable de répondre à certaines de ces situations. Mais sur un plateau plus grand, ça devient tout de suite plus compliqué.

Nous avons eu l'idée d'implémenter une fonction récursive pour parcourir les éventuelles chemins possibles « de sommets séparés par des ponts » à partir d'une base et vérifier si l'un d'eux nous mène à la base du coté opposé. Ce qui vérifierait la situation décrite ci-dessus et permettrait à l'IA de remplir les ponts le moment venu.

Le premier sommet, appel la fonction récursivement sur tous les sommets avec lesquelles il entretient des ponts. Ainsi de suite, si personne ne trouve la base recherchée, tous le monde renvoit 0. Par contre si un chemin mène jusqu'à la base en question, l'appel de fonction généré sur le pont final correspondant renverra 1. En effectuant un test sur la valeur de retour, après chaque appel récursif, l'appel original prendra automatiquement cette valeur lui aussi. Afin d'éviter les boucles infinis et de ne pas générer de nouveaux appels récursifs sur un sommet que l'on a déjà visité, la fonction prend une liste de sommet en paramètre qu'elle remplit au fur et à mesure.

Une deuxième fonction, se contente de chercher les pions connectés avec la base (directement, ou avec un pont). Elle appel à chaque paire de sommets trouvée la première fonction décrite précédemment. Si le scénario décrit a lieu, elle renverra 0 et modifiera les coordonnées du « **int** coup[3] » placé en paramètre, avec celles du premier coup à jouer.

3.4 Comportement par défaut

En plus du minimax et de la gestion des ponts nous avons décidé de réaliser un comportement par défaut de l'IA au cas où aucune des deux dernières fonctions ne trouvaient de coup rentable à jouer.

Après avoir pris connaissance de l'existence des ponts et compris qu'ils étaient essentiels dans le jeu de Hex nous avons tenté de faire créer des ponts à notre IA. Après plusieurs recherches et parties jouées nous nous sommes rendus compte que le premier coup était extrêmement rentable lorsqu'il était joué vers le milieu du plateau. Le comportement par défaut s'occupe d'une telle tache puis essaye ensuite de créer des ponts à partir du dernier pion joué vers les bases du joueur.

Lorsque « le Minimax » et la « Gestion des ponts » ne trouvent pas de coup plus intéressant à jouer que les autres, il faut mettre en œuvre un comportement par défaut. En partant du principe, qu'un joueur joue dans la direction du dernier pion adverse joué, nous avons remarqué des cas particulier, induit directement par la composition du plateau de Hex. Il est parfois intéressant de créer un pont, dans n'importe qu'elle direction. Néanmoins, si l'on se trouve un peu plus prêt des bases adverses et donc plus proche des extrémités du plateau, on remarque que certains ponts ne peuvent pas nous emmener jusqu'à la victoire.

Si aucun pont n'est assez intéressant, il devient alors plus intelligent de jouer un pion en direction de la base, juste à coté du dernier coup joué. Là encore, certains cas ne sont pas intéressant si l'on se trouve dans la mauvaise partie du plateau. Nous pouvons alors anticiper la construction d'un pont adverse, ou bien repartir vers la base opposée.

Si il cherche à construire un pont les formules ci-dessous, déterminent les diagonales intéressantes pour un pion (Blanc) de coordonnées (x,y) :

Si on se déplace vers le haut et que x<=get_plat_taille()/2:

- Jouer le pont de droite ou celui d'en face, si y<x-1
- Jouer le pont de gauche ou celui d'en face, si y>nb-(x*2)

Si on se déplace vers le bas et que x>=get_plat_taille()/2:

- Jouer le pont de gauche ou celui d'en face, si y>(nb-(nb-x))+1
- Jouer le pont de droite ou celui de gauche, si y<=((nb-2)-x)*2

Les formules pour un pion noir, peuvent être calculées réciproquement. Si on se trouve dans la partie nord du plateau, mais qu'on souhaite se diriger vers le bas, aucune des formules ci-dessus ne peut être appliquée. Dans ce cas, nous essayerons de prendre le pont en ligne droite, puis l'autre du coté opposé à celui où nous nous trouvons dans le plateau. (Il suffit de faire un test sur la coordonnée Y).

S'il joue juste à coté en direction de la base, les formules ci-dessous, déterminent les cases intéressantes pour un pion (Blanc) de coordonnées (x,y) :

Si on se déplace vers le haut :

- Jouer la case de droite, si y<nb-x
- Sinon, jouer la case de gauche.

Si on se déplace vers le bas :

- Jouer la case de gauche, si y>=(nb-1)-x
- Sinon, jouer la case de droite.

Si aucun des ces quatre cas n'est possible, on utilise les coordonnées du dernier pion joué par l'adversaire pour défendre et on essaye d'anticiper la création de nouveaux ponts (Au hasard, pour l'instant.).

Concevoir une intelligence artificielle pour le jeu de Hex est extrêmement complexe car il n'y a pas de procédure exacte à adopter afin de s'assurer la victoire, beaucoup trop de cas rentre en compte. En fait si nous voulions réaliser une véritable IA nous aurions dû faire de l'apprentissage automatique (machine learning). Malheureusement le temps à notre disposition et nos connaissances étaient insuffisants afin de réaliser un tel procédé. Cela aurait permis que l'IA apprenne d'elle même comment jouer au jeu de Hex, on aurait pu la faire jouer contre elle même énormément de fois afin qu'elle s'améliore toute seule.

IV – Module Base de Données

Nous avons décidé de coder la base de données en langage SQL car c'est ce que nous avions vu en cours TD et en cours TP. Nous pensions connecter la base de donnée avec notre programme du jeu de Hex mais nous n'avons pas eu le temps.

Le module Base de Données (bien que petit) est séparé en 3 parties distinctes :

4.1 MCD

Nous avons mis en place le Modèle Conceptuel de Donnée en groupe. En effet, nous nous sommes concertés afin de ne pas commettre d'erreurs, il fallait que tout les cas demandés puissent être traités et que le codage puisse se faire dans de bonnes conditions.

Le MCD est joint dans le document qui concerne la BDD. Nous avons rencontrés un problème : Pour le joueur gagnant, au tout début nous avions mis un attribut « gagnant » dans la classe partie. Mais nous nous sommes rendu compte qu'il fallait que ce soit une clé étrangère, et donc créer une classe d'association « gagner » entre partie et joueur avec les cardinalités adéquates. En effet, maintenant « gagnant » est obligé d'être un joueur.

4.2 MLD

Le Modèle Logique de Donnée est tiré du MCD. Le MLD est joint dans le document qui concerne la BDD.

4.3 Langage SQL

Lors de la mise en place du code SQL, aucun soucis particulier n'a été à déclaré. La mise en place de contraintes sur certains attributs à été nécessaire afin de respecter la cohérence de ces derniers.

V – Bilan

Nous avons au final dépassé notre objectif. Nous nous étions fixé la V2 et le début de la V3 et nous avons finalement réalisé jusqu'à la V3. Nous avons aussi

Pour conserver une trace du projet et faire profiter les intéressés, nous avons créé un site internet. Il résume les principales fonctionnalités du jeu, donne un aperçu de l'interface graphique et un lien pour éventuellement le télécharger. Nous mettrons aussi notre bilan à disposition, si jamais quelqu'un s'intéresse à notre réflexion ou bien même, cherche à comprendre le code source pour y

apporté ses modifications.

Nous avions prévus d'automatiser le téléchargement des mises à jours. D'ailleurs, dans la rubrique « A propos », nous avons dors et déjà réservé un emplacement pour informer l'utilisateur sur la dernière mises à jour téléchargée. Nous n'avons pas encore initialisé les différents niveaux de difficultés pour l'intelligence artificiel. Compte tenue du travail effectué, le comportement par défaut décris dans le bilan n'est pas totalement terminée.

On aimerait connecter le jeu et la base de données. Ainsi le joueur pourrait partager le résultat de sa partie à d'autres utilisateurs. Sans parler, du mode « Spectateur » qu'on aurais bien aimé implémenter avant la soutenance. Ce mode nous permettrais de visionner une partie dans son intégralité, seulement à l'aide du fichier .txt.

Nous pensons continuer le Projet même après la date limite comme Projet personnel.

Bilans personnels

Bilan Tom Bloch

J'ai apprécié le déroulé du projet vue l'organisation dont nous avons fait preuve. L'intelligence artificiel et les nombreuses difficultés que nous avons pue rencontrer nous poussaient à aller toujours plus loin. C'était la première fois que je me penchais sur ce genre de problème, d'ailleurs je souhaite qu'on termine le comportement par défaut que nous avons imaginé.

Je ne suis pas surpris de l'organisation nécessaire (en structure de données particulièrement), mais c'est la première fois que je me confronte à un projet aussi conséquent. Nous nous sommes appliqués sur la répartition des taches, la tenue d'un calendrier, il était essentiel de décomposer chaque programme pour identifier les différents outils nécessaires, les fonctions à implémenter, leurs paramètres et les type de retour correspondants sans oublier, les noms de variables !

J'ai aussi profité de cette occasion pour me familiariser avec les interfaces graphiques en Java, ce qui n'as fait qu'agrandir ma motivation. Nous nous sommes mis à la place de l'utilisateur, par exemple lorsque nous développions le menu principal. Ce moment est primordial, car il nous à permis d'anticiper toutes les modifications possibles dans le programme. On a évité de se fermer des portes, où nous l'avons fait en connaissance de cause. Par exemple, lorsqu'on imaginais un mode « Entraînement » . Une partie où l'utilisateur pourrait revenir en arrière dans l'ordre des coups, autant de fois qu'il le souhaite. Il seras peut-être implémenter dans une autre mise à jour.

Personnellement, j'étais charmé par l'ampleur du défi. J'ai fait beaucoup de recherches sur internet, pour enrichir la documentation et réfléchir sur de nouveaux processus. J'ai approfondis mes connaissances dans plusieurs domaines différents. J'ai trouvé ce sujet très intéressant, d'autant plus qu'on ne connaissais pas spécialement le jeu de Hex et ses stratégies.

Bilan Bence Lovas

Dans ce projet, nous avons tous participé sur une version papier, ensemble, en groupe, afin de nous mettre d'accord sur la réalisation des tâches, du planning, et dans l'organisation toute entière de ce projet.

Nous nous sommes mis d'accord sur la réalisation des tâches et nous en avons convenu que je ferais la Base de Donnée, la gestion des ponts ainsi que la réalisation d'un site internet afin de pouvoir télécharger le jeu . Dans l'idée, ce site devait aussi servir de base de donnée connectée avec le jeu , mais nous n'avons pas pu le faire dans les délais impartis (nous pensons continuer ce projet après la date limite).

La Base de donnée à été un module assez-simple. En effet, nous nous sommes convenus sur le MCD de base, il me restait juste à faire le MLD et à implémenter la Base de Donnée en langage SQL. Peu de problèmes ont étés rencontrés, en effet, nous avions eu les enseignements adaptés en cours TD , peu de choses nouvelles. J'ai eu quelques problèmes pour générer le MCD automatique avec le site mocodo, mais ils ont étés réglés.

La Gestion des ponts a été beaucoup plus complexe. Nous devions créer un système de détection des ponts afin que l'IA puisse jouer de façon optimisée. En effet, elle devait pouvoir jouer là où il y avait des ponts, mais aussi détecter ceux de l'adversaire et jouer en conséquence. Beaucoup de fonctions ont étés implémentées. Les structures de données ne sont pas mon point fort.

J'ai eu de nombreuses erreurs sur les pointeurs de fonctions, mon groupe ainsi que des recherches sur internet, et sur des sites de programmation m'ont aidés pour cela.

Enfin, le site internet du projet a été implémenté en dernier, nous ne pensions pas en avoir besoin (ou en avoir le temps). Il à été codé en langage HTML puis hébergé sur internet.
Sur ce site vous pourrez télécharger le jeu, ainsi qu'en connaître plus sur ses créateurs ainsi que les règles.

En guise de conclusion, Ce projet a été l'aboutissement de mois de travail et de réflexion personnelle ainsi que collective qui m'a permis de me perfectionner dans les langages que je ne maîtrisait pas forcement. Le projet « Jeu de Hex » a développeé ma capacité à m'organiser et à gérer un projet dans une limite de temps. Celui-ci fût à la fois ludique et technique , une belle expérience à titre personnel.

Bilan Matthieu Pont

Ce projet m'a beaucoup fait avancer dans ma manière de mener à bien un projet. Je retiens notamment la méthode que nous nous sommes donnée, c'est à dire de répartir la réalisation des modules entre chaque membre du groupe mais de réfléchir tous ensemble sur comment les concevoir. En plus de donner une vue d'ensemble du projet à l'équipe cela permet à chacun des membres de commencer la réalisation dans la bonne voie. Néanmoins je pense que nous ne pouvons appliquer une telle méthode sur des projets de plus grande ampleur.

De plus je me dois pour les prochains projets de mieux prévoir à l'avance les différents aléas, en effet je n'ai pas assez bien pris en compte le travail hors-projet (le partiel etc.) ce qui nous a retardé par rapport au planning que nous nous étions fixé.

J'ai tout de même était déçu du comportement de notre IA finale, nous n'avons pas réussi à programmer une intelligence artificielle correcte qui peut se débrouiller et jouer au jeu de Hex correctement.

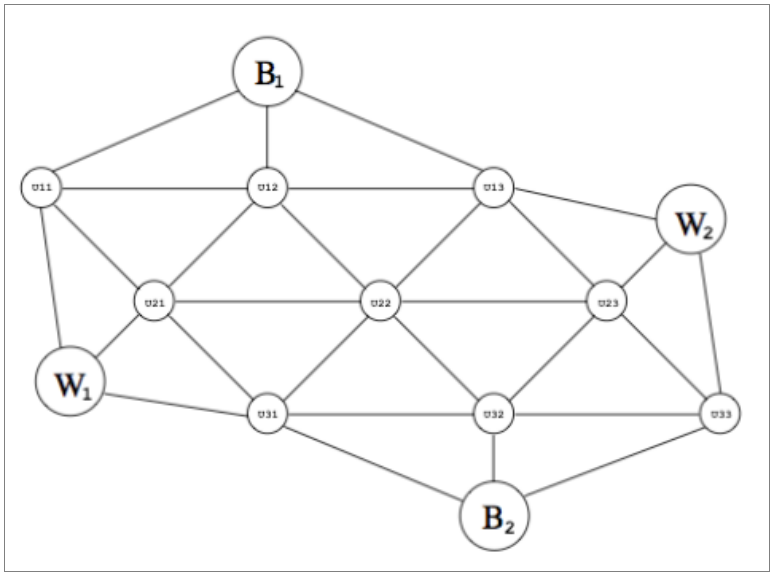
Vers la fin du projet, la V2 étant presque terminé Tom Bloch et Bence Lovas ont commencé la V3 afin de ne pas perdre de temps. J'étais donc seul pour terminer la V2, notamment pour la conception de la fonction d'évaluation basé sur le réseau de résistances. Mes connaissances en électronique m'ont aidé à mieux appréhender ce problème. C'est à mon avis le point dont je suis le plus fière pour ce projet.

III – Module C

3.1 Programme de base (V1)

```
\hex
\dim 4
\board
* * * *
. o . .
. . . .
. . . .
\endboard
\game
\play o 1 1
\play * 0 0
\play * 0 1
\play * 0 2
\play * 0 3
\endgame
\endhex
```

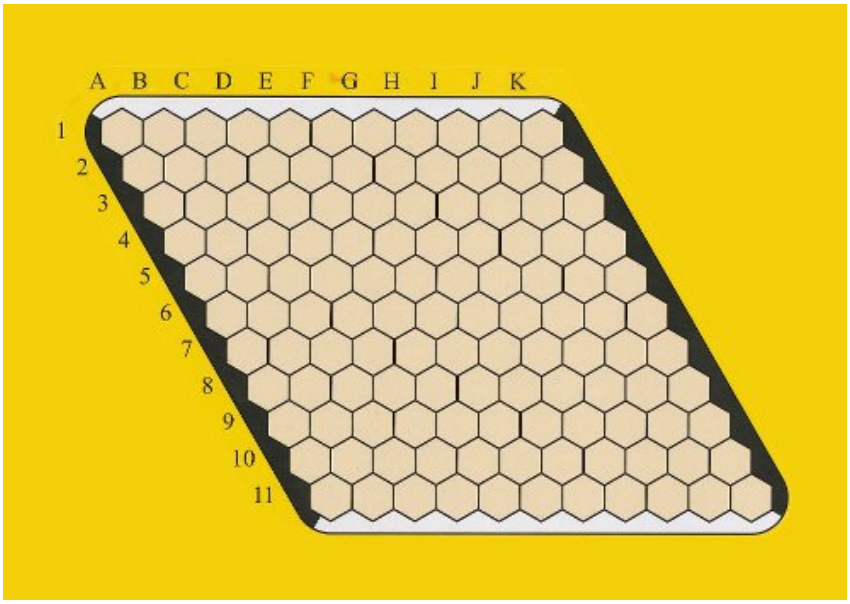
Annexe 3.1 : Exemple de fichier de sauvegarde



Annexe 3.2 : Exemple de graphe 3x3

Note: B1, B2, W1 et W2 représentent les bords du terrain et seront des sommets qui nous permettront de détecter la victoire d'un joueur. Dans notre programme les bords sont inversés par rapport à cette image.

3.2 Minimax (V2)



Annexe 3.3 : Plateau sur lequel nous nous basons

Bibliographie

II – Module Java

2.1 Interface Graphique

http://codes-sources.commentcamarche.net/source/43904-grille-hexagonale

https://openclassrooms.com/courses/apprenez-a-programmer-en-java/notre-premiere-fenetre

https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-champs-de-formulaire

http://codes-sources.commentcamarche.net/source/15687-ajouter-une-barre-de-menu-a-votre-application-jmenubar

https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html

2.2 Programme Principal

http://baptiste-wicht.developpez.com/tutoriels/java/outils/executables/

https://effingo.be/2007/10/03/creation-sous-copyleft/

http://fr.creativecommons.org

https://fr.wikipedia.org/wiki/Copyleft#Le_symbole_copyleft_et_Unicode

III – Module C

3.1 Programme de base (V1)

3.1.2 Gestion du graphe réduit et des groupes

https://fr.wikipedia.org/wiki/Liste_d%27adjacence

3.2 Minimax (V2)

http://www.ms.uky.edu/~lee/ma415fa11/gardner-hex.pdf

http://www.cs.cornell.edu/~adith/docs/y_hex.pdf

http://www.lamsade.dauphine.fr/~cazenave/papers/hex-ria.pdf

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.297.2930&rep=rep1&type=pdf

https://en.wikipedia.org/wiki/Minimax

https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

https://en.wikipedia.org/wiki/Horizon_effect

a) Réseau de résistances de Shannon

https://pdfs.semanticscholar.org/9947/d6edcdd12c294f847fb516d6e1c30e6545bd.pdf

https://www.aaai.org/Papers/AAAI/2000/AAAI00-029.pdf

http://library.msri.org/books/Book42/files/anshel.pdf