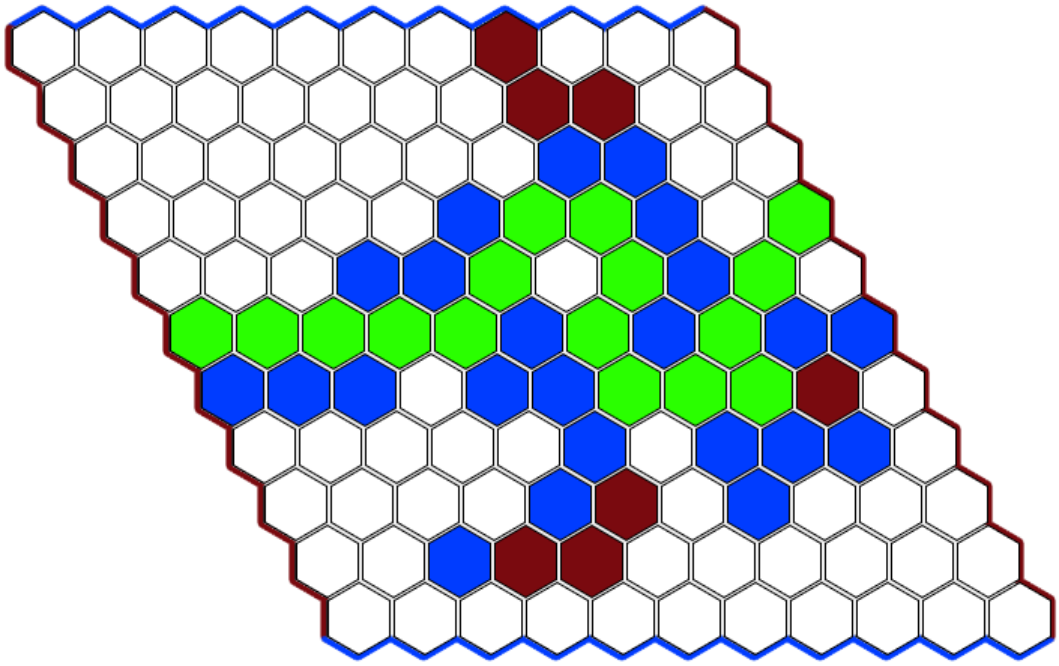
PROJET INFORMATIQUE

JEU DU HEX



2018 / 2019

DELIENCOURT Aymeric
GRATTEPANCHE Aymeric
ROZIERE Antoine

CSI3
ISEN Yncréa Méditerranée
Campus de Nîmes

Table des matières

I. Le projet : Introduction

1) Le jeu du Hex

Il se joue à 2 personnes sur un tablier en forme de losange dont les cases sont hexagonales. Les côtés face à face sont de la même couleur. Ces 2 couleurs sont représentées sur les pions et sont choisies par les joueurs au début de la partie. Le but est de faire rejoindre les 2 côtés opposés (de même couleur) avec les pions correspondants tout en empêchant l'autre joueur de le faire. Un joueur gagne s'il parvient à relier ses cotés avec ses pions avant son adversaire.

2) Cahier des Charges

Pour faire notre programme nous avions plusieurs obligations. Tout d'abord nous devons mettre à disposition 2 modes de jeu. Le premier est un mode joueur vs joueur classique. Le second est quant à lui un mode joueur vs IA (Intelligence Artificielle). Pour ce dernier nous avons donc dû créer une IA. Celle-ci était basée sur l'algorithme « Min-Max », mais elle n'a pas fonctionné alors nous nous sommes rabattus sur une IA moins « Intelligente » que les IA que nous développerons dans la partie Théorie de ce rapport.

Ensuite nous devions afficher une fenêtre ainsi que le jeu de manière graphique et non dans une console. Nous y avons fait un menu, et des paramètres réglables (mode de jeu, taille du tablier, nom du jouer, animation, etc.).

3) Description fonctionnelle du besoin

Fonction principale : Jouer au jeu "HEX"

Fonction secondaire :

- Création d'une interface graphique : Le but de cette fonction, à la demande du client, est de rendre le jeu jouable intuitivement | *Priorité Moyenne*
- Création d'un mode de jeu "Simple" (Humain contre Humain) : Permet de prendre en main facilement le jeu avant de jouer contre l'ordinateur | *Priorité Haute*
- Création d'un menu (Jouer, Option, Graphisme, etc.…) : Permet de choisir facilement le mode de jeu (H/H ou H/IA) et les options | *Priorité Basse*
- Création d'un mode de jeu "Intelligent" (Humain contre IA) : Demander par le client, il doit utiliser un algorithme de la théorie des jeux | *Priorité Haute*

4) Possibilité d'application

Pour faire ce programme, nous avions plusieurs choix.
Concernant le langage, nous en avions deux : le C/C++ et le JAVA. Nous avions commencé par le C/C++ avant de tout retransposer en Java.
Pour créer l'interface graphique, nous avions deux bibliothèques disponibles : SDL et JFrame.
La bibliothèque SDL a été utilisée en C/C++ tandis que JFrame l'a été pour Java.

5) Gestion du projet

Afin de mener à bien notre projet, nous avons immédiatement désigné un Chef d'équipe qui aura pour mission d'attribuer les différentes taches et étapes clés du projet. Le choix s'est rapidement porté sur Aymeric DELIENCOURT car il a déjà quelques connaissances autour du « management » dans ce domaine.

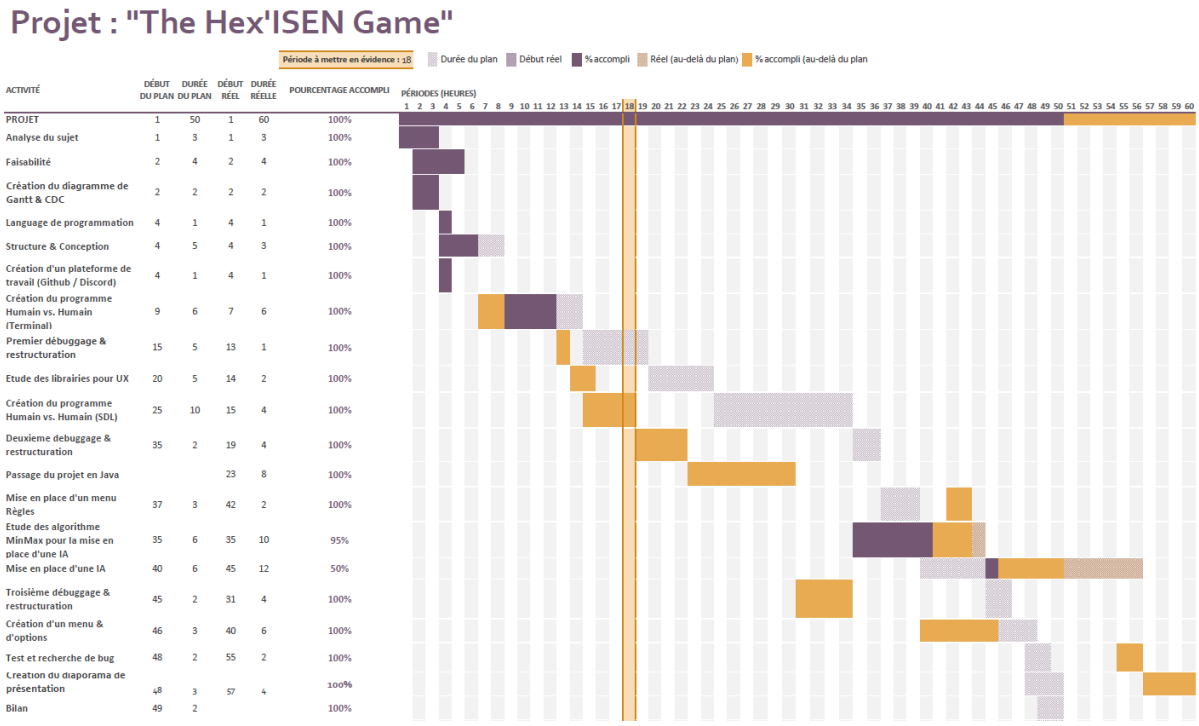
Pour faciliter la communication et le développement du jeu, nous avons utilisé deux technologie/software : *Discord* & *GitHub*.

Discord est un logiciel de communication par vocal/message, nous avons créer un serveur spécial pour le projet. Celui-ci nous a permis de travailler à distance, et de travailler en dehors des horaires de projet pour plus de flexibilité dans les tâches à accomplir.

GitHub est un site reposant sur la technologie du *Git*, c'est un système de partage et de versionning énormément utilisé dans le milieu du développement informatique. Celui-ci nous a donc permis de travailler chacun sur des parties et fonction du jeu et ensuite de les rassembler aisément. Afin de donner une idée Antoine à développer la fonction *findPath()* et la ensuite *commit* sur notre Git.

Un *repo* GitHub a également été créé lors de la création de la *library* « SDL-Menu » en collaboration avec François BARBIERO, un membre d'une autre équipe. (Equipe sur les Dames Chinoises)

Un Diagramme de Gantt a été mis en place. Le voici :



Lors de la création de ce diagramme il est à noter que le projet était prévu pour une durée de 50h, celle-ci a été allongé au-delà de 60h.

Sur ce diagramme nous pouvons voir distinctement que certaines étapes on était plus rapide que d'autres. De plus, l'étape « Passage en Java » a été rajouté seulement plus tard lors d'un besoin de flexibilité sur le jeu qui sera expliquée dans la suite de ce rapport. Cette étape étant une reconversion complète du jeu en langage orienté objet *Java*, il a été nécessaire de recommencer toutes les étapes précédemment réalisées (Version Terminal, Humain vs Humain, etc…)

Notre projet s'est donc subdivisé en trois parties majeures, nous avons donc choisi d'en faire de même pour ce rapport.

II. Théorie

En ce qui concerne la création d'intelligence artificielle, plusieurs possibilités se sont offertes à nous. La première dont on a eu connaissance est l'algorithme développé par von Neumann appelé *MinMax*.

1) MinMax

L'algorithme minimax (aussi appelé algorithme MinMax) est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle consistant à minimiser la perte maximum (c'est-à-dire dans le pire des cas).

Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser (le jeu est à somme nulle).

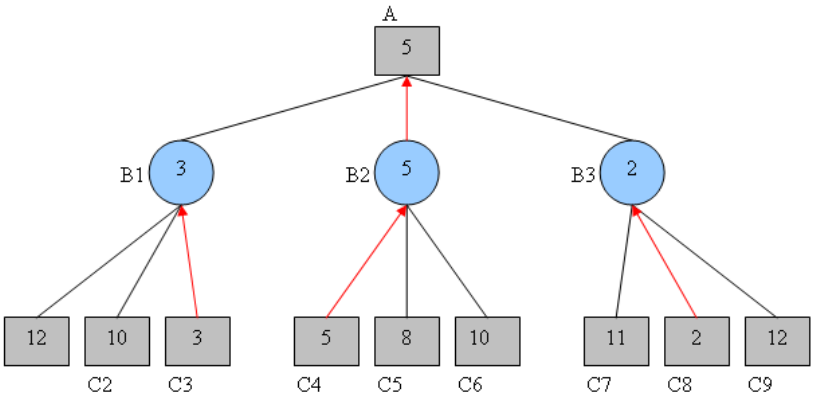


Figure 1Exemple d'arbre Algorithme MinMax

2) MinMax par élagage alpha-bêta

Cet algorithme reprend le principe du MinMax mais en arrêtant l'exploration de certaine branche car trop faible.

L'algorithme alpha bêta accélère donc la routine de cherche minimax en éliminant les cas qui ne seront pas utilisés. Cette méthode utilise le fait que tous les autres niveaux de l'arbre seront maximisés et que tous les autres niveaux seront minimisés.

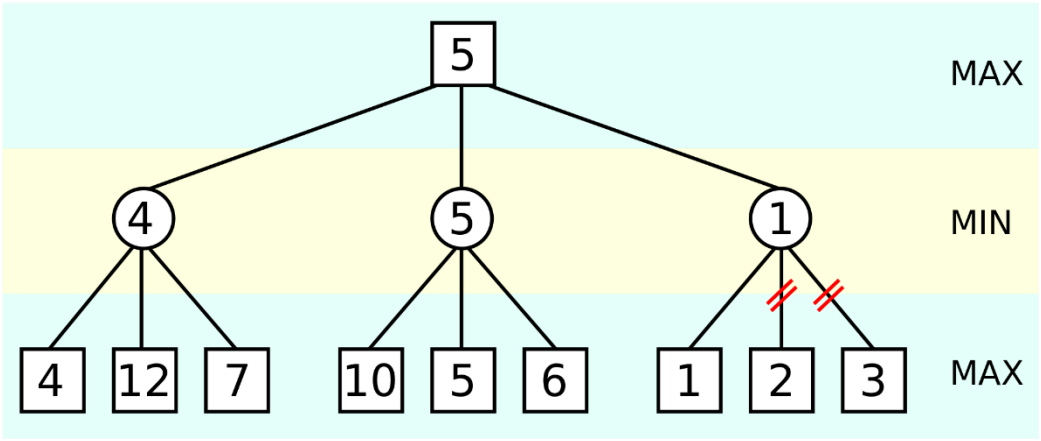


Figure 2 Exemple Arbre d'algorithme MinMax élagage Alpha-Beta

3) Intelligence artificielle : Apprentissage supervisé.

L'apprentissage supervisé (*supervised learning* en anglais) est une tâche d'apprentissage automatique consistant à apprendre une fonction de prédiction à partir d'exemples donnés. L'idée pour le jeu de Hex serait d'avoir une base de données de plusieurs parties gagnantes. Ainsi avec l'utilisation d'un algorithme utilisant *Naives Bayes* nous pourrions estimer sur quelle case jouée à l'aide d'une probabilité basée sur les coups précédents.

Pour ce type d'apprentissage, il existe d'autres types comme les régressions linéaires et le *K-Means*, et bien d'autres…

Ce type d'IA est celle utilisée pour AlphaGo, l'Intelligence Artificielle développée par Google pour jouer et gagner contre le champion du monde du jeu de Go. Le jeu de Go se rapprochant du Hex.

4) Intelligence artificielle : Apprentissage non supervisé.

Dans le domaine informatique et de l'intelligence artificielle, l'apprentissage non supervisé est un problème d'apprentissage automatique. Il s'agit, pour un logiciel, de trouver des structures sous-jacentes à partir de données non étiquetées. Puisque les données ne sont pas étiquetées, il n'est pas possible d'affecter au résultat de l'algorithme utilisé un score d'adéquation.

Dans notre jeu, et à notre niveau, il est très difficile de l'implémenter.

5) Fonction d'évaluation

L'algorithme MinMax et MinMax Alpha Beta nécessite une fonction d'évaluation du plateau afin de déterminer un poids à chaque case du tablier. Ainsi la case aillant un point élevé est à privilégier.

Dans le cadre du jeu Hex, la littérature indique que l'évaluation la plus efficace utilise l'algorithme de Dijkstra qui permet de déterminer le chemin le plus court. Cet algorithme fait appel à la théorie de graphe, et est très intéressant à étudier mais nous n'avons pas réussi à le mettre en œuvre pour l'Hex.

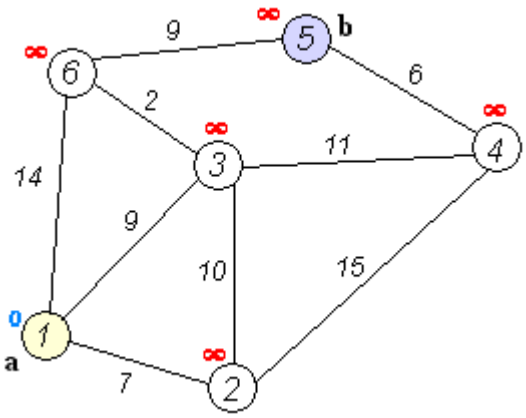


Figure 3 Exemple de graphe - Algorithme de Dijkstra

D'autres fonctions ont également été étudié par notre groupe comme celle de considérer le plus gros groupement de pion. Celle-ci s'est avérée inefficace.

III. Première partie : « Version C-Console »

Nous avons commencé le projet informatique en utilisant un codage de type langage C celui que nous étudions à ce moment de l'année.

En utilisant des logiciels tel que discord pour communiquer ou encore GitHub pour avancer le code simultanément, la première version joueur vs joueur de HEX a été finalisée fin octobre.

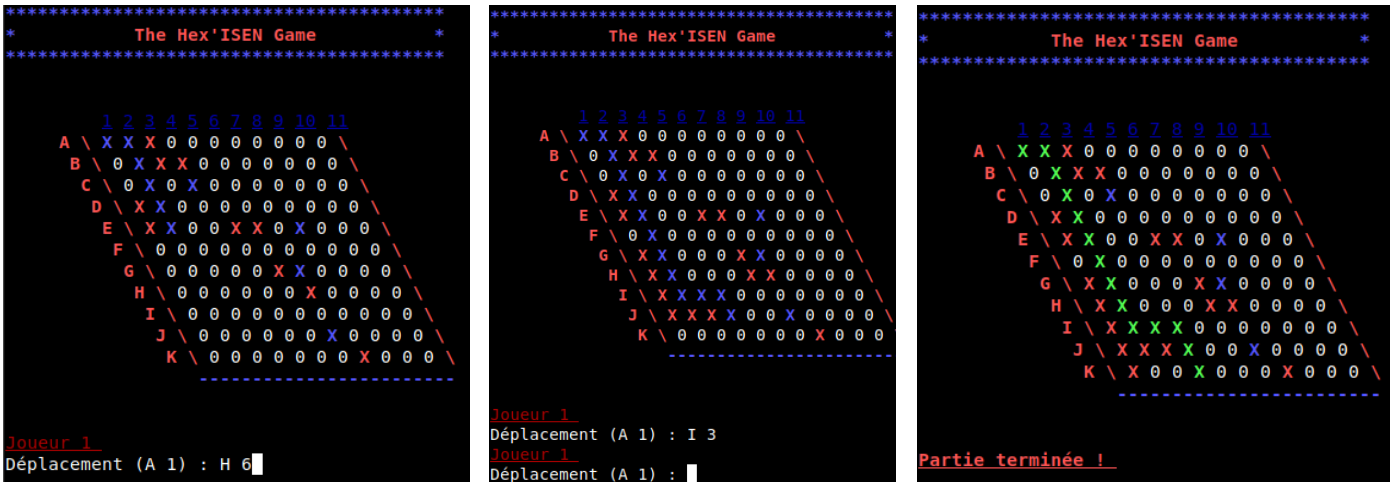


Figure 4 Version Terminal (Demande position à l'utilisateur | Position déjà prise | Utilisateur qui gagne)

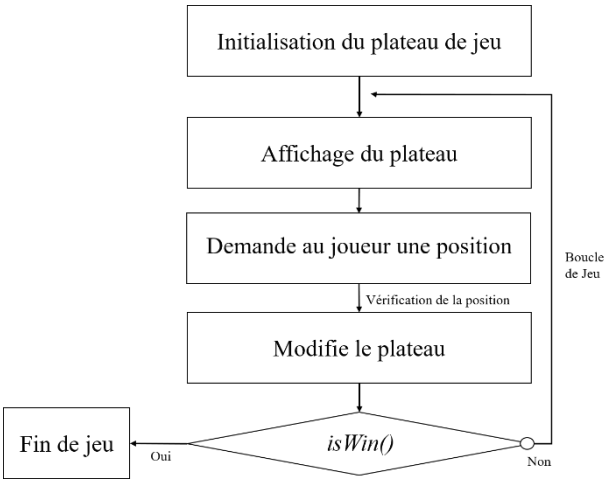
1) Gameplay

Pour cette première version nous avons décidé de faire les choses simples, et de nous concentrer sur un jeu fonctionnel avant tout. Aucun aspect graphique, aucune intelligence artificielle. Des variables en français, un affichage sous forme de console (et non pas de plateau graphique).

2) Conception et organisation

Ce code se divise en 5 fichiers :

- *Define.h* va nous permettre de définir les couleurs de chaque joueur (rouge ou bleue) ainsi que toutes les constantes du jeu.
- *Affichage.h* contient majoritairement des *printf* pour afficher le tableau tout au long du jeu.
- *Jeu.h* va déterminer la victoire à l'aide de la fonction *findPath()* et *isWin()* d'un des concurrents. Ce fichier va également déterminer les fonctions mécaniques du jeu de Hex.
- *Game.c* notre fonction principale qui va définir le plateau ainsi que la boucle de jeu
- *Prototype.h* pour le prototypage.



Tout d'abord on initialise le tableau de 11*11 (un plateau classique de Hex ne pouvant être redimensionné ici).
On affiche ce dernier.
Puis on lance la boucle de jeux permettant aux deux joueurs de placer un pion chacun leur tour testant chaque fois la condition de victoire.
Enfin un des joueurs gagne une fois la condition de victoire atteinte.

IV. Deuxième partie : « Version C-Graphique »

Pour cette seconde version, après avoir réussi une version « *Clean and Simple* », nous nous sommes penchés sur un aspect important du cahier des charges, l'interface graphique. La première difficulté a été de choisir quelle librairie C utilisée. Trois choix ont été possible : la librairie GTK, la librairie TKinter ou la SDL. Cette dernière a été privilégiée pour sa capacité d'adaptation à un jeu vidéo 2D.

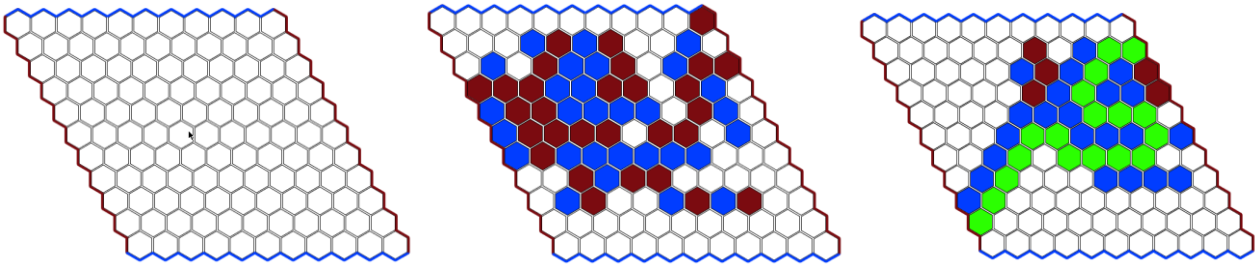


Figure 5 Interface Graphique SDL (Plateau vide | Partie en cours | Partie terminé)

1) Gameplay

L'idée de cette version graphique est d'avoir un jeu plus facile d'accès à une utilisation « lambda ». En effet, une version se jouant dans un terminal en ligne de commande n'est pas accessible et pas très ludique. Nous avons donc, avec la librairie SDL créer une fenêtre avec un plateau de jeu cliquable. Un simple : *Point and Click*

2) Conception & Organisation

Pour réaliser une interface graphique, il a d'abord fallu créer et imaginer comment rendre une expérience utilisateur agréable. Pour cela, nous avons réalisé une maquette que l'on a ensuite découpée pour l'intégration des images dans le jeu.

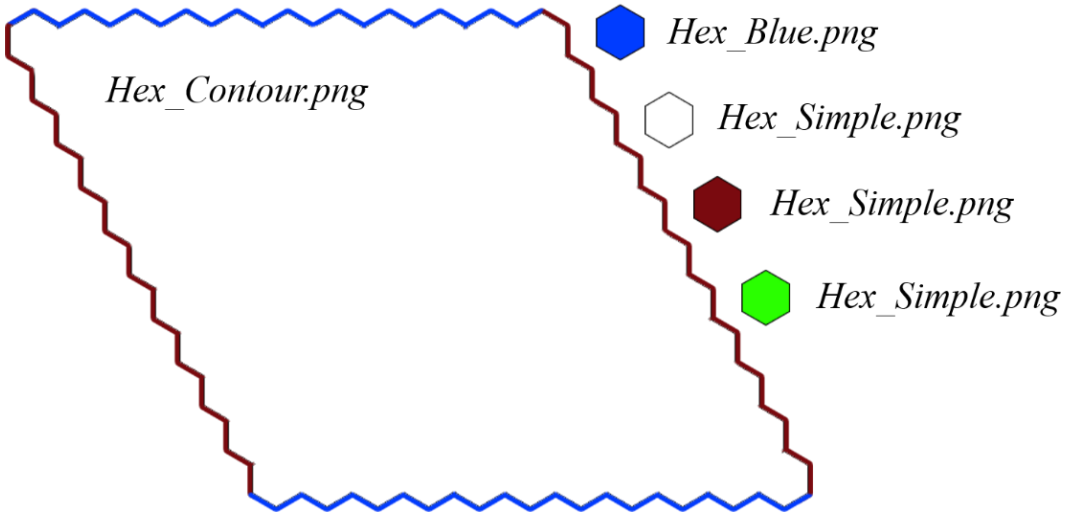


Figure 6 Création des images nécessaire pour le jeu

Afin de faciliter l'intégration de la librairie SDL, une structure a été mise en place. Celle-ci faisant écho avec les langages objets. Intéressons-nous au prototypage de notre jeu.

```
1  typedef struct {
2    SDL_Surface *hex;
3    SDL_Rect pos;
4    int joueur;
5  } Hex_Case;
6
7  typedef struct {
8    Hex_Case **board;
9    int sizeBoard;
10 } Hex;
11
12 Hex initHex(int sizeTab); // Initialise le plateau 2D en allocation dynamique (malloc etc)
13 Hex_Case **initBoard(int n);
14 void freeBoard(Hex_Case **board);
   void hexClick(Hex jeu, int n);
15 int winHex(Hex hex); // Utilisation de findPath et retourne un booléen si le jeu est WIN
16 int findPath(Hex hex, int x, int y, int n); // Recherche d'un chemin dans un labyrinthe
17
18 SDL_Rect click(Hex jeu);
19 void updateBoard(Hex jeu, SDL_Surface *ecran);
20 void updateScreen(SDL_Surface *ecran);
21 void displayContour(SDL_Surface *ecran);
```

Tout d'abord, notre jeu est composé de deux structures principales. La première celle de type *Hex*, elle définit une sorte d'instanciation de notre jeu, elle contient un tableau dynamique à deux dimensions de type *Hex_Case*, ainsi qu'une caractéristique majeure de notre jeu, la taille du plateau. La structure *Hex_Case* est la plus importante, celle-ci regroupe un plateau à la fois simple, et graphique. En effet, *joueurs* correspond uniquement à *0=EMPTY ; 1=RED PLAYER ; 2=BLUE PLAYER* mais *SDL_Surface *img* représente l'image d'une case du tableau et *SDL_Rect pos* représente les coordonnées de l'image, ainsi avec la libraire SDL nous pouvons coller notre image *BlitSurface()* et actualiser notre écran *Flip()*.

Après la réalisation du plateau graphique, notre chef d'équipe s'est attelé avec François BARBIERO à la réalisation d'une libraire SDL pour faciliter l'implémentation et la création de menu au sein d'un jeu SDL.

3) Problème rencontré

Lors du développement de l'interface graphique, nous nous sommes heurtés à un problème de gestion de la mémoire RAM. En effet, chaque nouvelle case (allocation dynamique) prenait un espace mémoire et ne se libérait pas. Après avoir joué sur tout le plateau nous étions montés à 3Go de RAM utilisé par notre programme.

Pour régler ce problème, nous avons du revoir la structure du programme afin d'inclure les *FreeSurface()* permettant de libérer la mémoire. Après modification la mémoire était stable et ne dépassait pas les 60 Mo de RAM.

4) SDL-Menu

Au départ, l'idée de cette librairie était de créer un code simple, beau et efficace pour faciliter l'implémentation de menu graphique. Ainsi, à la fin nous pourrions simplement fournir celle-ci ainsi qu'une documentation aux autres groupes pour leurs jeux.

Les détails techniques de celle-ci ne sont pas réellement intéressants et ne seront donc pas explicités dans ce rapport car nous avons passé notre jeu en Java et nous n'avons donc finalement pas utilisé la librairie contrairement à d'autres groupes.

Plus d'informations sur cette librairie sont sur la page : https://github.com/DES69U-FR/SDL-Menu ou sur le rapport du groupe de François BARBIERO.

V. Troisième partie : « Version Java-Graphique »

Dans cette dernière version du projet, nous avons dû reprendre toutes les étapes à zéro pour le plus grand bien du projet. Pourquoi être passer en Java ? Pour plusieurs raisons, tout d'abord le *cross-platform*, en effet, installer la SDL sur Windows n'a pas était chose facile pour l'ensemble des membres du groupe, de plus une SDL Linux n'est pas parfaitement identique à une SDL Windows. La version graphique du jeu était faite sur un Linux (Distribution Ubuntu), une compilation pour exporter un exécutable Windows était tout bonnement impossible. Java était une alternative.

De plus le Java gère extrêmement bien les *memory leaks* (fuites mémoires) et c'est un langage orienté objet, ce qui simplifie la première approche *casi*-objet que l'on avait sur le projet en *C-SDL*.

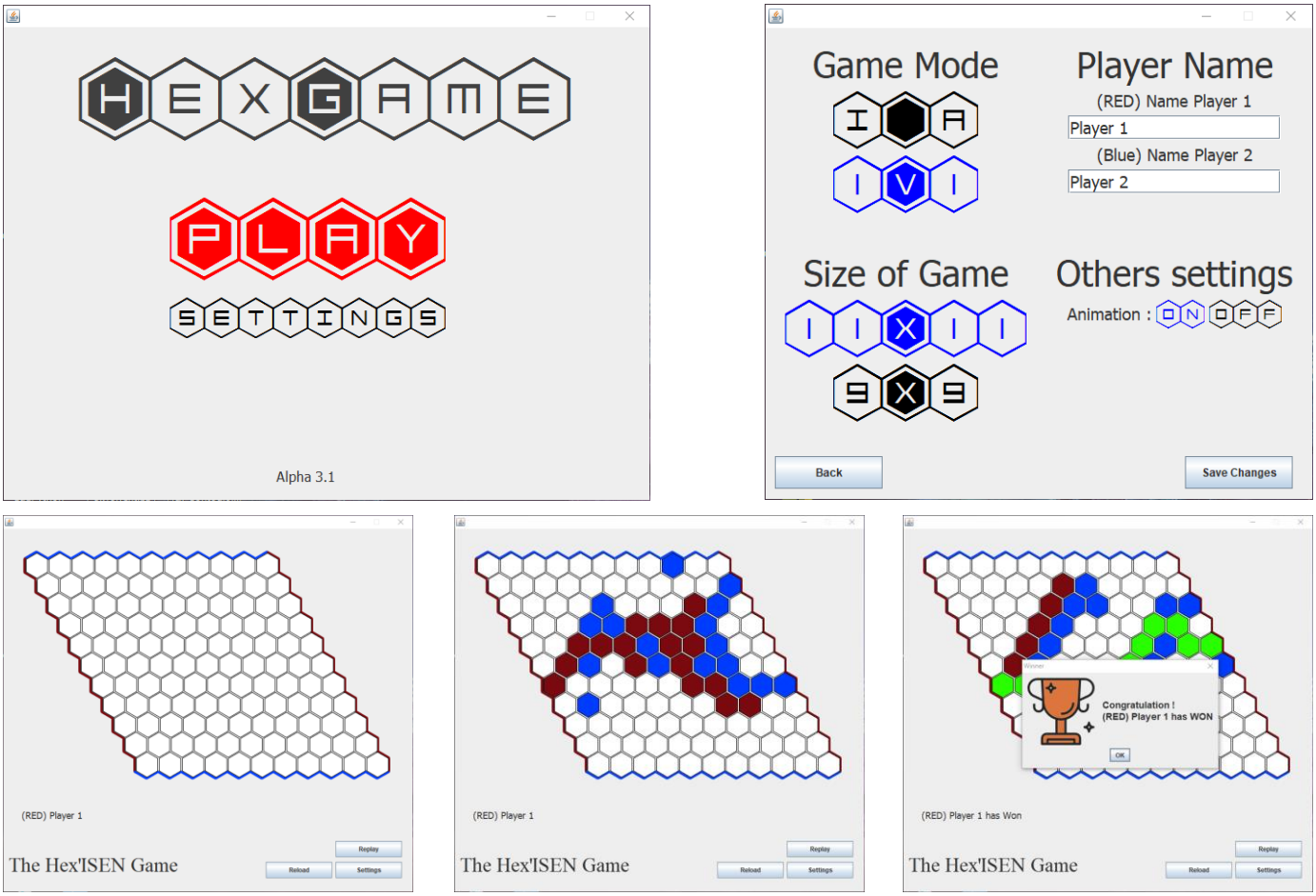


Figure 7 Interface Grpahique Java (Menu Home | Menu Settings | Jeu vide | Jeu en Cours | Partie terminé)

1) Gameplay & Fonctionnalité

En ce qui concerne le Gameplay, nous avons gardé le même principe du *Point and Click* de la version précédente. Ensuite nous avons rajouté quelques fonctionnalités :

- Possibilité de changer de mode de jeu : 1v1 ou I.A
- Possibilité de changer la taille du plateau
- Possibilité de nommer le *Player_1* et le *Player_2*
- Possibilité d'activer ou non les animations

2) Conception & Organisation

Une possibilité et une amélioration aurait été de construire notre jeu sur une structure MVC *(Model-View-Controller)*, le java étant particulièrement adapté à cela. Mais pour simplifier le développement et par manque de temps, nous avons fait autrement.

Le code est divisé comme suit. Trois packages distincts : *game*, *graphical* et *main*.

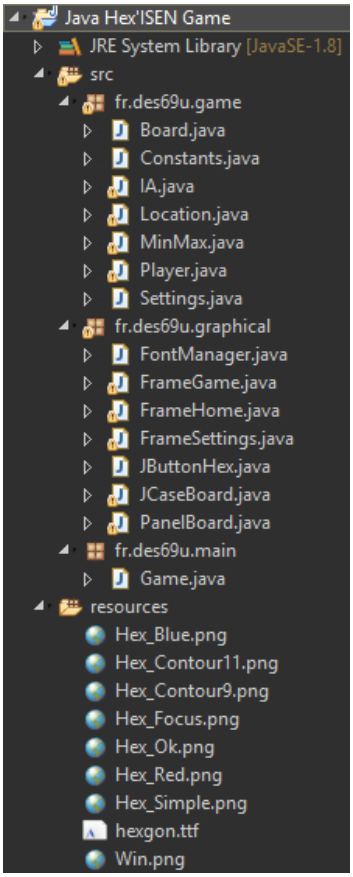


Figure 8 Arborescence Java

Le premier *game* va regrouper l'ensemble des classes lier à la mécanique du jeu. *Player, Board, IA,* etc… *Board* est la classe qui s'occupe de la gestion du plateau de jeu, celui-ci est déclaré en *public* dans la classe principale du main (*fr.des69u.main.Game*). Ainsi nous avons accès à tout moment au plateau et au fonctionnalité liée en faisant *Game.board.<function>*.

Le deuxième package gère entièrement l'aspect graphique du jeu à l'aide de la librairie Java *Swing* également appelé *JFrame*. L'objet *JCaseBoard* est une extension de *JButton* de la librairie *JFrame*, cet objet correspond à un bouton du plateau. Le plateau graphique est géré par l'objet *PanelBoard* qui est étendu de *JPanel*.

Ainsi sur notre *JFrame* principale nous affichons un *PanelBoard (extends JPanel)* contenant un affichage de 11x11 ou 9x9 *JCaseBoard (extends JButton)*.

JButtonHex est également une extension de *JButton* et permet une implémentation facile, rapide et intuitive d'un bouton aillant la police du Jeu. Cette dite police est contenue dans le fichier *JAR* afin de pouvoir l'afficher même si l'utilisateur ne l'a pas sur son PC. Elle est gérée par l'objet *FontManager*.

3) Mémoire

La java était plus lourde que le C, notre programme utilise environ 160 Mo de mémoire vive et moins de 1 Mo de mémoire morte. La java gère automatiquement la mémoire dynamique.

4) Piste d'amélioration

Système de MAJ automatique, plusieurs IA et les faires jouer entre elles, jeux en réseau (client/ serveur), etc …

VI. Bilan du projet

En conclusion, ce projet nous a permis de fortement de développer nos connaissances à la fois en programmation mais également en gestion de projet et management d'équipe.

Tout le long du projet, nous avons dû faire fasse à de nombreuses complications et « bugs » ce qui nous a permis de développer notre capacité d'adaptation : Qualité indispensable dans nos avenirs professionnels.

Sources

Stephan K. Chalup, Drew Mellor, and Fran Rosamond (2015), The Machine Intelligence Hex Project, *Technical Report*.

Arneson, B., Hayward, R. B., & Henderson, P. (2010). Monte Carlo tree search in Hex. IEEE Transactions *on Computational Intelligence and AI in Games*, *2*(4), 251-258.

Anshelevich, V. V. (2002). A hierarchical approach to computer Hex. Artificial, Intelligence, 134:101–120.

Browne, C. (2000). Hex Strategy, Making the Right Connections. Natick, MA, A. K. Peters.

Henderson, P. T. (2010). *Playing and solving the game of Hex*. University of Alberta.

Van Rijswijck, Jack. (2002). Search and evaluation in Hex.

Sciences Etonnante, (2018) Intelligence Artificiel, *Youtube.com*.

Gardner, M. (1959). The Game of Hex. Ch. 8 in Hexaflexagons and Other, *Mathematical Diversions: The First Scientific American Book of Puzzles and Games.* Simon and Schuster, New York, pages 73–83.

Bryce Wiedenbeck (2015), https://www.cs.swarthmore.edu/~bryce/cs63/s16/labs/hex.html

Wikipedia : Algorithme minimax, Apprentissage non supervisé, Apprentissage supervisé

Openclassroom : Algorithme MinMax