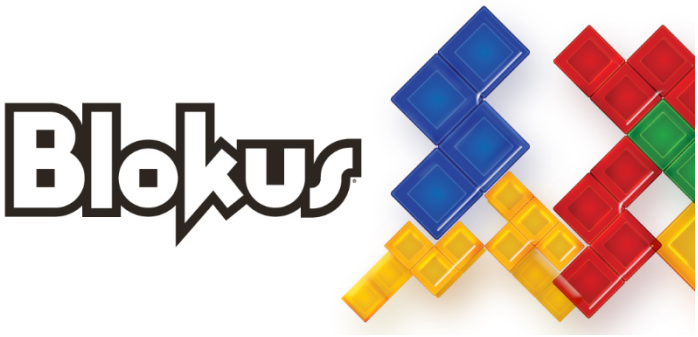

Le Mans Université

Programmation Avancée en C

Projet - Création d'un jeu - Semestre 4

Jeu du Blokus

Auteurs :
M. Florian Tourpe
M. Kevin Semamra
M^me^ Marilou Friant
M. Joris Amillard

Encadrants :
M. Loïc Barrault
M^me^ Claudine Pio-Tofollon
M^me^ Dominique Py

Dépot GitHub Final : https ://github.com/sylyako/Projet_blokus
1er dépot GitHub : https ://github.com/Deartheload/Projet_Algo_S4
(cf. Notification envoyée par Mail à Mr Loïc Barrault)

En date du Vendredi 6 Avril

Table des matières

1 Introduction

Nous voilà tous bel et bien tombés dans le grand bain...

Un projet... Un rapport... Une soutenance...

Pour ce quatrième semestre d'étude, nous avons, comme un commun accord, relevé le défi de programmer un « petit » jeu en quelques mois. Tous les membres du groupe ont pris cette mission à bras de corps avec un esprit collaboratif à n'en pas douter.

Nous avons repris un jeu assez connu créé par un Français : Bernard Tavitian. C'est un jeu de plateau où chaque joueur se voit attribuer une couleur et une série de pièces. Ces dernières sont toutes différentes et ont des formes rectangulaires. A tour de rôle chacun doit apposer une pièce sur le plateau et accumuler un maximum de points... Un petit effort... Vous ne trouvez toujours pas... ?

Le très célèbre jeu du Blokus

2 Organisation

Le groupe s'est naturellement décomposé en sous groupe de binômes.

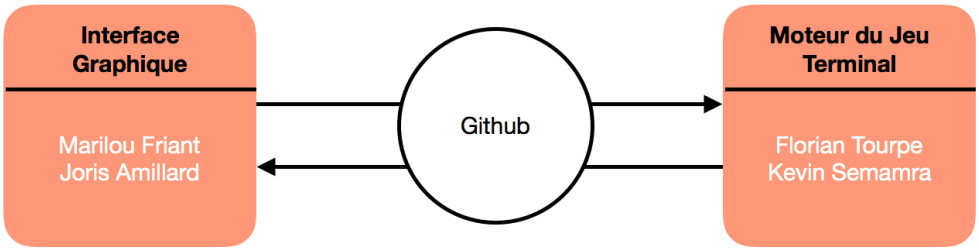

FIGURE 1 – Représentation de notre organisation.

Très simplement, c'est de cette manière que nous avons réussi à collaborer : un binôme s'occupait à plein temps de l'interface graphique, l'autre du moteur du jeu ainsi que la version terminal qui y est étroitement liée, tout en gardant la communication pour rester dans un même état d'esprit.

Ce schéma de groupe nous a semblé plutôt efficace car tout d'abord aucune personne n'a été mise à l'écart sur une tâche et deuxièmement, chaque binôme pouvait être redécoupé en deux afin d'accélérer la réalisation de tel ou tel fonctionnalité sur un programme.

De plus l'accès à Github a significativement facilité la mise en relation des données communes à la création du jeu. Les mises à jour du dépôt permettait d'accéder , presque en temps réel, au travail de l'autre binôme.

On peut donc représenter dans un tableau la répartition des tâches du groupe.

	AMILLARD	TOURPE	SEMAMRA	FRIANT
Cahier des charges - Mise en place des idées	25 %	25 %	25 %	25 %
Représentation d'une pièce	25 %	25 %	25 %	25 %
Rotation d'une pièce	25 %	25 %	25 %	25 %
Déroulement d'une partie	25 %	25 %	25 %	25 %
Developpement du Moteur	10 %	50 %	30 %	10 %
Interface Graphique	45 %	5 %	5 %	45 %
Sauvegarde	25 %	25 %	25 %	25 %
Réseau	5 %	5 %	85 %	5 %
I . A	5 %	85 %	5 %	5 %
Intégration du Moteur à L'interface	40 %	5 %	5 %	50 %
Redaction du rapport	40 %	15 %	15 %	30 %
	0,245	0,264	0,245	0,245

FIGURE 2 – Représentation de notre organisation.

Afin de représenter le planning des tâches à effectuer, nous avons réalisé un diagramme de Gantt. Au fur et à mesure du projet, nous sommes revenus dessus pour anoter la durée réelle de la tâche (symboles).

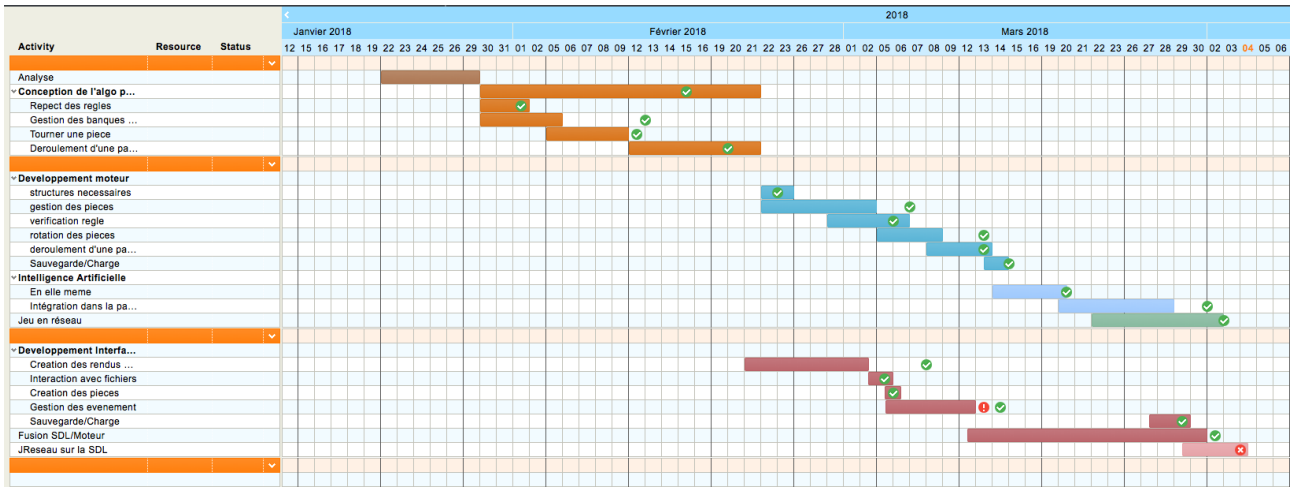

FIGURE 3 – Diagramme de Gantt.

3 Conception

3.1 Les règles du jeu

Les règles de jeu du Blokus ne sont pas très difficiles... « Pas très difficile » quand une personne les explique devant le plateau mais un peu plus compliquées quand il s'agit de les programmer. Nous avons donc, dans la mesure du possible, essayé de respecter au maximum les règles du jeu.

Règle N1 : « La règle d'or »

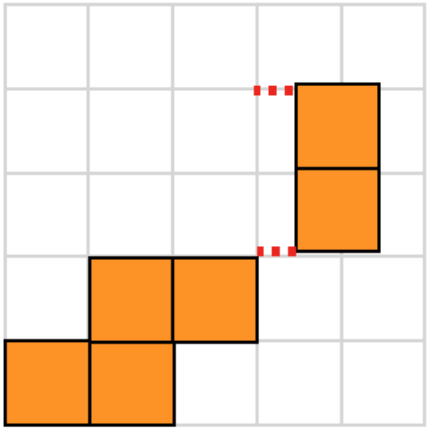

FIGURE 4 – Seul placement autorisé

Chaque pièce doit être apposée sur le plateau de façon à ce qu'elle soit en contact seulement avec un coin d'une autre pièce de la même couleur.

But du jeu :

Placer le plus de pièces possible. Les pièces restantes en main à la fin de la partie détermineront un score négatif.

Déroulement d'une partie :

Les joueurs tirent au sort leur couleur et à chacun est attribué une banque de pièces composée de 21 pièces. Le premier joueur place la première pièce de son choix sur le plateau de telle sorte que celle-ci recouvre l'angle de départ de ce joueur.

Les autres joueurs jouent à tour de rôle et placent leur première pièce de la même manière. Pour les tours suivants, chaque nouvelle pièce doit être posée en respectant la « Règle d'or » cf. Règle N1.

En revanche, les pièces de couleur différentes peuvent se toucher par les cotés.Les pièces du jeu peuvent être disposées dans n'importe quel sens sur le plateau. Une pièce posée sur le plateau reste en place jusqu'à la fin de la partie.

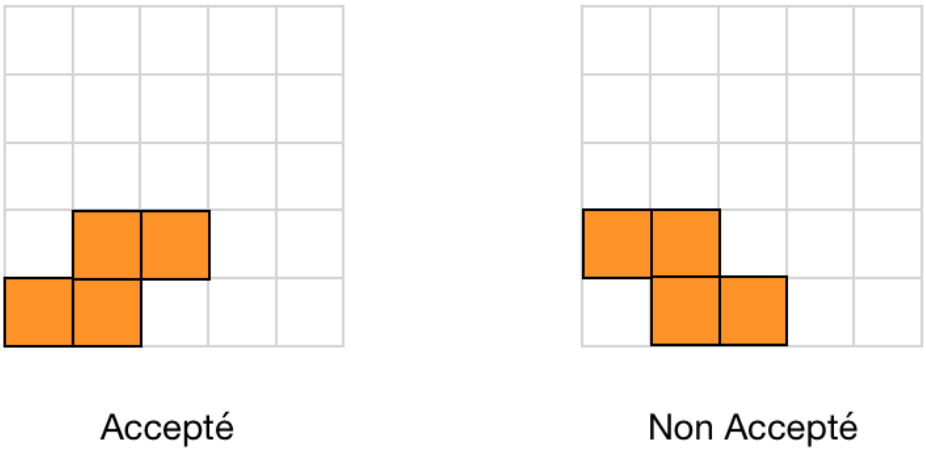

FIGURE 5 – Premier Tour de Jeu

Fin d'une partie :

Lorsqu'un joueur est bloqué et ne peut plus placer de pièce, il est obligé de passer son tour et ne peut plus jouer jusqu'à la fin du jeu. Les autres joueurs poursuivent en conservant le même ordre de jeu. Lorsque tous les joueurs sont bloqués, chacun compte le nombre de pièces qu'il n'a pu placer sur le plateau et calcule son score : -1 point par carré non posé. +15 points si les 21 pièces ont été posées. +20 points si les 21 pièces ont été posées avec le carré solitaire (pièce n1) en dernière position.

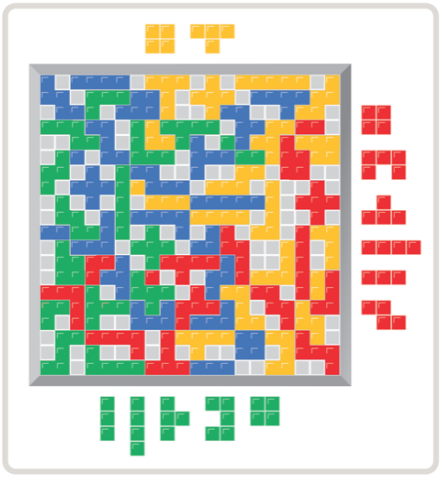

FIGURE 6 – Exemple de Fin de Partie

3.2 Fonctionnalités

Le jeu du Blokus étant simple, nous avons très vite assimilé les difficultés que l'on allait rencontrer au cours du projet. En ayant pu anticiper le travail à fournir, nous avons pu rajouter ou adapter certaines fonctionnalités, en voici la liste :

Mode 4 Joueur

Possibilité de jouer au blokus à quatre joueurs, le plateau prend une taille de 20x20. Les positions de départ des joueurs sont définies dans les quatre coins.

Mode 2 Joueur

Possibilité de jouer au blokus à deux joueurs, le plateau prend alors une taille de 14x14. Les positions de départ des joueurs sont changées : le permier joueur est placé en haut à gauche et le deuxieme en bas à droite. Le positionnement de la première piece change aussi, le premier joueur doit débuter en 4 4 et le second joueur en 10 10.

Interface Graphique

Interface graphique SDL qui permet de jouer au blokus en mode deux joueurs, quatre joueurs, avec ou sans IA et de lire les règles du jeu.

"Intelligence Artificielle"

Au lancement de la partie, le joueur a la possibilité de choisir jusqu'à 3 IA.

Jouer en Réseau

Permet de jouer au blokus en réseau à deux joueurs.

Sauvegarde une Partie

Permet à l'utilisateur de sauvegarder une partie en cours.

Charger une Partie

Permet à l'utilisateur de revenir au menu et de retrouver sa partie en cours quand il le souhaite.

Aide au Placement

Affiche les positions où le joueur peut placer ses pièces restantes.

Developpement

4 Programmation modulaire

La programmation modulaire est très utile dans ce genre de projet. En effet, il est souvent nécessaire d'avoir des fonctions "passe-partout" qui peuvent être utilisées plusieus fois dans le projet. On se constitue donc une "boîte à outils" utilisable par tous les membres du groupe, à n'importe quel endroit du code. Pour se faire, nous avons du faire les liens entre nos fichiers avec la commande include en début de programme et regrouper les signatures des fonctions dans des fichiers .h. Afin de pouvoir compiler l'ensemble des fichiers en une fois, nous avons créé un makefile. (cf.Outils)

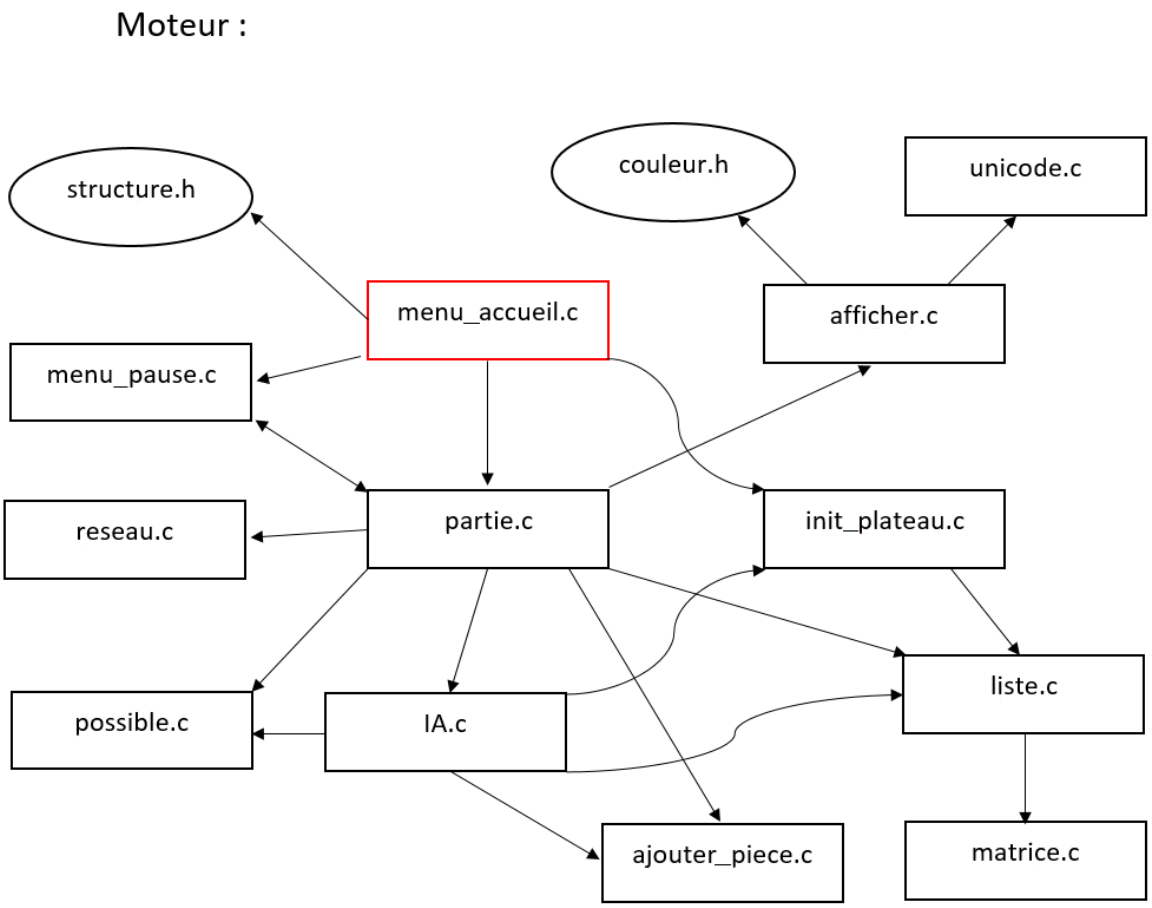

FIGURE 7 – Schéma de l'architecture de nos fichiers pour la version terminal

5 Moteur du jeu - Version terminal

Il est temps de rentrer dans le vif du sujet et d'expliquer la manière avec laquelle on a programmé le jeu du Blokus.

Revenons sur les notions de bases qui vont nous permettre d'expliquer dans le détails les prochaines fonctionnalités.

5.1 Les structures principales

Le plateau de jeu est défini comme un tableau à deux dimensions de type t_case, il est de taille 14x14 pour une partie à deux joueurs ou de taille 20x20 pour une partie à quatre joueurs. On définit chaque case de ce plateau par la couleur qui lui est associée (rouge, bleu, vert, jaune ou libre), ainsi que la possibilité qu'à chaque couleur de pouvoir placer une pièce sur cette case.

Pour ce qui est des pièces chaque joueur possède sa propre banque de pièces, elles sont stockées dans un tableau de type t_liste. Chaque liste regroupe toutes les pièces du joueur et toutes les pièces de la liste sont de type t_matrice, qui contient la taille de la pièce, son numéro d'identification ainsi que la matrice de référence de la pièce qui permet de savoir à quoi elle ressemble. Cette matrice est de taille 5x5 car les pièces sont constituées au maximum par cinq blocs.

5.2 De deux à quatre joueurs

Tout le cœur du jeu est présent ici, la plupart, si ce n'est toutes, des fonctions qui ont été créées au cours du projet sont appelées ici. Il est cependant demandé au préalable, au moment où l'on veut lancer la partie, de définir le nombre de joueurs total, (c'est-à-dire deux ou quatre joueurs) ainsi que le nombre total d'ordinateurs parmi tous ces joueurs. Le programme est capable d'adapter automatiquement son fonctionnement, selon les paramètres choisi avant le lancement. Il en est de même pour la plupart les fonctions appelées qui seraient capables de fonctionner même avec d'autres tailles de plateau ou avec un autre nombre de joueurs différent que ceux définis par les règles du blokus.

De ce fait, le fonctionnement d'une partie en mode deux joueurs et quatre joueurs ne diffèrents pas du tout, la principale contrainte aura donc été de faire un programme qui aurait dans les grandes lignes été capable de fonctionner sur d'autres modes de jeux voir même sur d'autres jeux complètement différents, autrement dit, un programme générique.

5.3 L'affichage

Il nous a semblé important de créer un rendu visuel agréable dans la version terminal. Nous avions créé au départ une matrice avec des caractères simples du clavier (_ , |) qui étaient générés. Cette matrice était peu lisible car relativement étroite. C'est pourquoi nous avons décidé d'utiliser des caractères unicodes pour créer une matrice de plus grande taille. Chaque case du plateau fait deux caractères de haut et de large dans le terminal. Dû au retour à la ligne obligatoire pour afficher la totalité d'une case, les cases sont affichées ligne par ligne, moitié (haute) par moitié (basse). De plus, nous avions entrevu l'utilisation des couleurs dans le terminal en cours d'outils de communication, nous nous en sommes resservis pour remplir les cases de la couleur de chaque joueur afin de différencier chaque pièce.

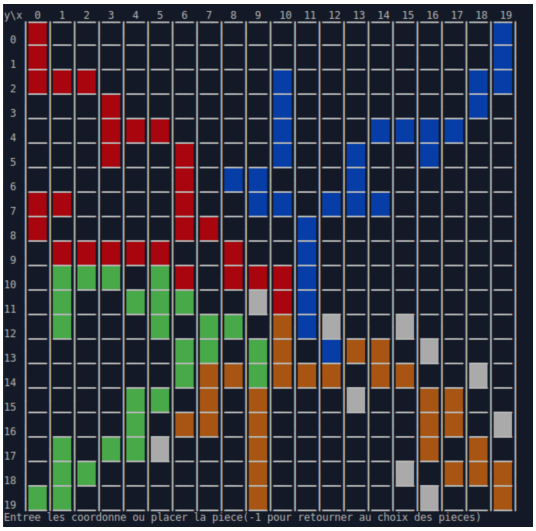

FIGURE 8 – Affichage du Jeu en Version terminal

5.4 Positionnement et test

L'une des premières fonctionnalités du Blokus est de pouvoir poser des pièces sur un plateau ainsi que de pouvoir appliquer une rotation à ces pièces. Les rotations qui sont disponibles dans le terminal, correspondent à toutes celles qui sont possibles de faire dans la réalité, c'est-à-dire rotation droite, gauche ainsi que la rotation miroir horizontale et verticale. Ces algorithmes ont pu être facilement mis en place grâce au cours de mathématiques du S3. Pour ce qui est du placement, la principale difficulté est l'utilisateur, qui doit déterminer mentalement là où la pièce doit être positionnée. En effet celui-ci doit déterminer en fonction du centre de la matrice de la pièce qu'il a choisi, la coordonnée à laquelle il veut poser la pièce. Il nous a donc semblé évident de devoir créer la fonction qui pose

la pièce dans le plateau comme une fonction de test qui vérifie si les coordonnées rentrées permettent bien de placer une pièce.

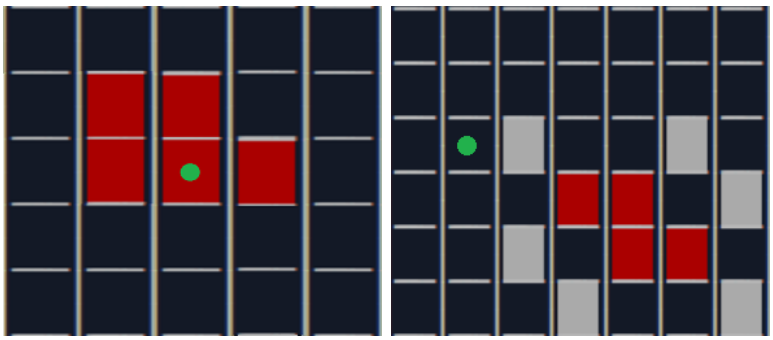

FIGURE 9 – A gauche : Pièce à placer. A droite : A quelle position la placer

Pour ce qui est de la gestion des cases où le joueur peut jouer, le programme est capable de déterminer s'il reste des coups possibles au joueur, en considérant la liste de pièce du joueur et les cases définies comme possibles après avoir posé la pièce. Une fois la pièce posée sur le plateau, le programme détermine toutes les cases où le joueur pourra jouer et entre les coordonnées correspondantes dans un tableau dynamique. Ceci permet de garder un tableau avec un minimum de place mémoire. On traite ensuite ces coordonnées pour savoir s'il existe une possibilité d'y insérer une pièce, on y retire alors les possibilités incorrectes si on en trouve.

5.5 L'intelligence artificielle

L'intelligence artificielle étant l'un des principaux points d'intérêts du « pourquoi » nous avions choisi le blokus, elle a très vite été implémentée après avoir terminé de créer le jeu en lui-même. Cette dernière a été conçue de façon à simuler n = trois tours de jeu, tout en prenant en compte le placement des pièces pour les autres joueurs qui eux vont donc simuler n-1 tours de jeux. Tout ceci avec différentes pièces de chaque joueur, avec les huit rotations de pièces possibles et bien sûr les différentes cases où il est possible de jouer. Pour vous donner une idée, cela représente environ 73 000 itérations pour un milieu de partie, ce qui n'est pas acceptable niveau temps de traitement. Pour compenser, l'algorithme a très vite été tourné vers un MinMax à élagage Alpha-Bêta. Celui-ci cherchant à optimiser ses placements en fonction du nombre de possibilités que lui fournira la pièce, la taille de la pièce ainsi que du positionnement vers le centre au début de la partie.

Ce qui a été le plus long n'a pas été la création de l'algorithme en lui-même, mais plutôt la phase de test et de complétion de l'algorithme afin d'obtenir une intelligence artificielle suffisamment compétante, tout en gardant un temps de réponse très bas. On notera notamment des possibilités d'amélioration pour prendre en compte le taux de « blocage » d'une pièce.

6 Interface Graphique - Version SDL

Cette première approche du jeu en version terminal fut poussée à son maximum. L'affichage de la matrice avec des caractères unicodes, l'utilisation des couleurs pour différencier les joueurs, le menu pour accéder aux différentes fonctionnalités. Pour une approche de programmeur tout ceci peut paraitre plutôt aboutit mais du point de vue d'un utilisateur lambda ? Est ce qu'il ne serait pas plus facile d'utiliser la souris afin de manipuler concrètement les pièces ? Est ce qu'il ne serait pas plus esthétique de voir les pièces qu'il nous reste en main ? C'est là qu'intervient l'interface graphique, un complément visuel venant suppléer le moteur du jeu.

On ne parle ici que d'une option, d'une simple amélioration visuelle pour l'utilisateur. En effet, l'interface graphique réalise des actions à l'écran qui elles, sont récupérées par le moteur du jeu tournant en tâche de fond. C'est donc à nous, programmeur, de faire le lien entre le back et le front.

6.1 Le langage

Le langage C étant un langage de bas niveau, les modules qui le composent le sont aussi... Nous avons la chance, que dis-je le privilège de découvrir la SDL (Simple Directmedia Layer). Bien que le côté graphique peut paraître plus ludique et plus agréable à utiliser, cependant on ne peut qualifier sa programmation de la même manière. Le langage SDL reprend le langage C dans sa globalité mais les termes et expressions sont tous nouveaux et biens différents les uns des autres. De plus, les langages étant sans cesse renouvelés et donc versionnés, il était primordial d'utiliser la bonne syntaxe en fonction de la bonne version. Pour ce qui est du projet, nous avons utilisé la version SDL2.

6.2 Jouer sur l'interface graphique

La fenêtre de jeu peut être décomposée en trois parties. La première à gauche, est composée de 4 boutons, qui sont les déclencheurs pour envoyer les informations au moteur. La deuxième au milieu est le plateau de jeu et la dernière à droite est la banque de pièces du joueur qui est en train de jouer.

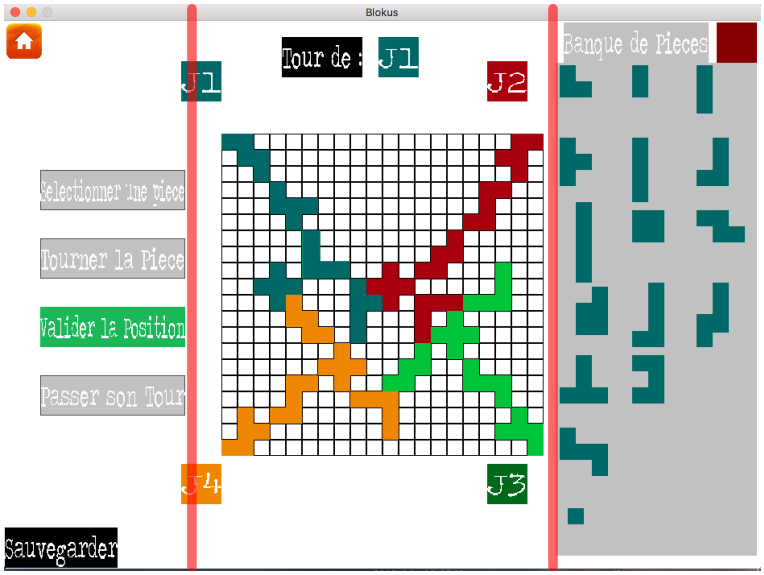

FIGURE 10 – Les éléments qui composent la fenêtre de jeu

6.3 Nos importations

6.3.1 Le texte

L'affichage de texte a été aussi une étape que nous n'avions pas imaginé au départ. Encore une fois, étant dans un langage de base niveau aucune fonction prédéfinie n'existe vraiment... Si l'on veut insérer du texte et bien nous devons aller le chercher et c'est le cas de le dire. L'utilisation des chaînes de caractères fait appel à la ressource SDL_ttf qui gère les polices et qu'il faut télécharger ainsi que les fonctions qui permettent de manipuler du texte dans une fenêtre.

6.3.2 Les images

Après avoir téléchargé la bibliothèque de fonction prévue à cette effet, nous avons pu utiliser des images au format compressé (jpg,png) dans une fenêtre. Il est vrai que la SDL permet, sans cet apport, de manipuler des images mais celles-ci doivent être en format bmp (format d'image très lourd). Afin de pouvoir utiliser l'image, cette dernière doit être texturée mais aussi dimensionnée grâce à un « Rect » implicitement créer afin de lui attribuer une taille et une position puis il faut copier cette texture sur le Render aux dimensions du « Rect ».

6.3.3 Les outils d'image

En ayant pris la décision d'utiliser des images pour générer les pièces de jeu, il nous fallait des outils pour les manipuler. Pour prendre un cas concret, la rotation des images a demandé l'importation d'une nouvelle bibliothèque : SDL2_gfx.

FIGURE 11 – Exemple d'image représentant une des pièces du blokus

6.4 L'organisation d'une fenêtre

En SDL2, une fenêtre ne peut avoir un, et un seul « Renderer », autrement appelé moteur de rendu. C'est sur celui-ci que nous y déposons nos différents éléments . Ces éléments en question sont dans notre cas des « Surfaces » ou des « Rect » et permettent d'agencer, de mettre en forme une fenêtre. Les image elles, nécessitent l'utilisation des « Textures », une fois que l'image a été insérée dans une texture on peut ensuite la copier sur le moteur de rendu. Il est vrai que tout ceci n'est pas vraiment intuitif et c'est pourquoi la prise en main peut-être plus ou moins longue.

Voici le schéma qui nous aurait fait gagner beaucoup de temps au commencement du projet :

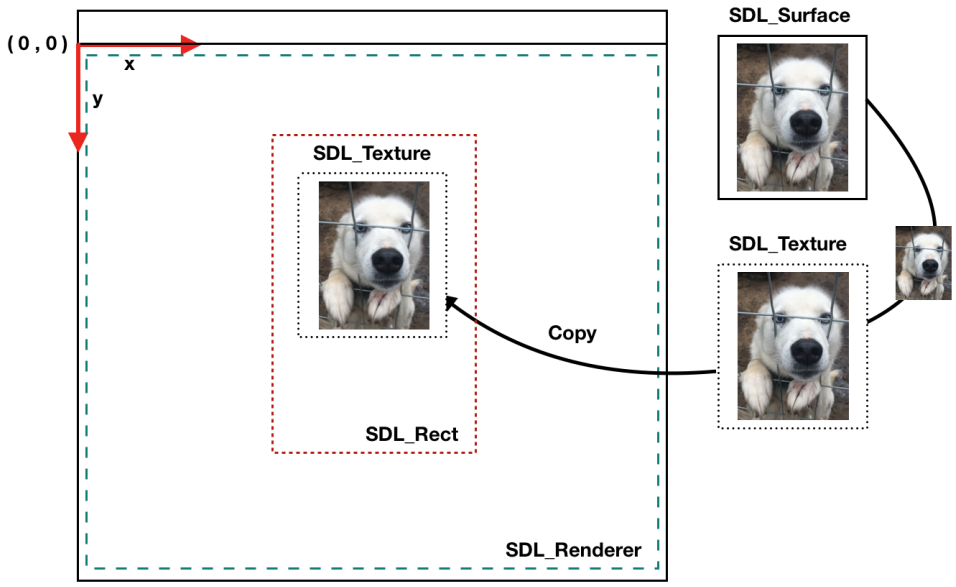

FIGURE 12 – Représentation d'un élément sur une fenêtre en SDL2

L'image est d'abord insérée dans une surface qui, pour pouvoir être manipulée, doit être convertie en Texture. Cette texture est ensuite copiée dans le renderer de la fenêtre aux positions et aux dimensions du rect défini plus haut dans le programme. Toutes ces étapes sont indispensables pour intégrer des éléments dans une fenêtre en SDL2.

6.5 Menus

Nous avons opté pour un menu très simple afin d'axer nos efforts sur les fonctionnalités du jeu. La fenêtre principale du jeu est un menu permettant l'accès à :
- une nouvelle partie
- une partie sauvegardée
- règles du jeu

FIGURE 13 – Menu du jeu

Pour permettre d'accéder à différentes fenêtres du jeu, il a fallu comprendre une autre notion indispensable en SDL

6.5.1 La gestion des événements

Afin de gérer les actions que peut générer l'utilisateur (souris, clavier et d'autres moins connues) la SDL fait intervenir une nouvelle variable : la variable « event » Grâce à celle-ci nous avons la possibilité d'intéragir avec les différents INPUT mais aussi de manière plus précise : clic droit, gauche, la touche « espace », « entrer » ou les deux... Nous avons le moyen de tout récupérer afin d'y attribuer telle ou telle action. En rajoutant à tout ceci une action bloquante (« WaitEvent ») ou non bloquante (« PollEvent ») nous pouvons créer l'illusion d'un bouton intégrant une action.

Ainsi, pour qu'une piece suive la souris de l'utilisateur, on récupère constamment la position de la souris, et on affecte cette position à l'image.

6.5.2 Les pièces du jeu

Nous avons fait le choix d'importer des images et de les manipuler directement pour jouer. Nous aurions pu aussi créer les pièces avec des surfaces y insérer de la couleur et les positionner de la même manière. Cependant il nous a paru intéressant aussi bien au niveau esthetique qu'au niveau de la programmation d'utiliser des images.

6.5.3 Le plateau

La grille de jeu est constituée de rectangles mis bout à bout, nous avons fait ce choix afin de pouvoir récupérer le numéro de chaque case. Le reste des éléments ne sont que des « Rect » dans lesquels on a incorporé du texte.

6.6 Fonctionnalités de l'interface

6.6.1 Le magnétisme de la grille

La pièce étant déplacée et positionnée à la souris, l'utilisateur ne peut déposer la pièce de manière parfaite au pixel près, et quand bien même il le ferait, il n'a pas à gérer les contraintes de la programmation. Il était donc nécessaire pour nous de créer une fonction qui puisse repositionner la pièce correctement à l'endroit voulu.

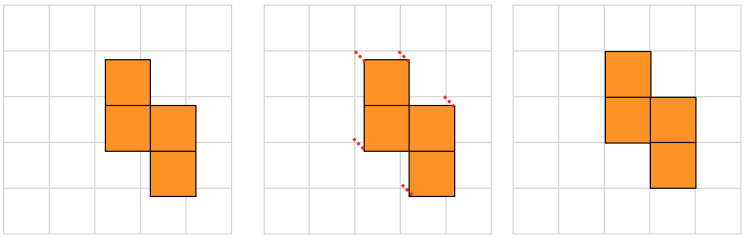

FIGURE 14 – Représentation du magnétisme sur la Grille

6.6.2 Rotation d'une pièce

Pour coller au plus près des fonctionnalités initiales du Blokus, le principe de rotation devait être intégré afin que l'utilisateur puisse modifier la pièce initialement positionnée dans la banque de pièces. Cette fonctionnalité requière un nouveau package à ajouter à nos bibliothèques de fonctions : SDL2_gfx.

Il est donc possible de définir un angle en degré qui sera appliqué à une nouvelle surface grâce à la fonction rotation90Degrees( ) (dans SDL2_rotozoom.h). Cette fonction prend en paramètre : un surface à pivoter et le nombre de rotation à 90 degrés que l'on veut faire.

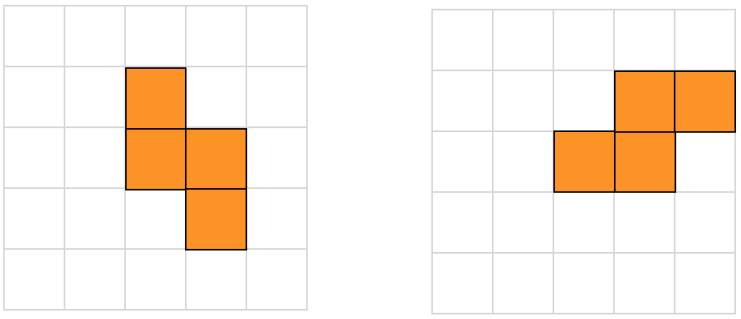

FIGURE 15 – Représentation d'une rotation à 90 degrés

7 Nos Outils

Ce genre de projet se doit d'utiliser des outils adaptés et beaucoup plus performants que ceux que l'on a l'habitude d'utiliser.

Notre premier outil a facilité la collaboration et la transmission des données à travers le groupe.

7.1 GitHub

GitHub est une plateforme permettant de regrouper des fichiers afin qu'ils soient accessibles depuis une machine connectée à internet.

De plus, ce logiciel permet de créer sur chaque machine du groupe un dépot local facilitant l'accès et la mise à jour des fichiers. Ces fonctionnalités spécifiques nécessitent l'utilisation d'instructions particulières en ligne de commande (git push, git pull , git commit...)

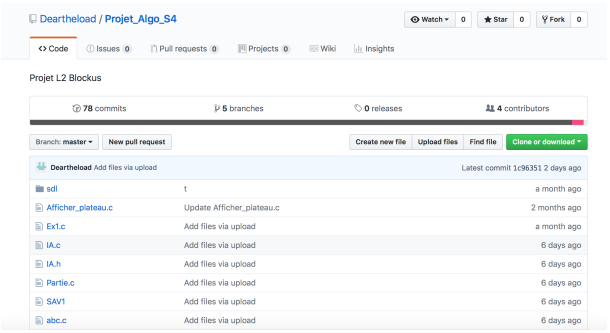

FIGURE 16 – Page GitHub en ligne

7.2 Le makefile

Il est primordial, pour un projet de cette ampleur, de créer un fichier de type Makefile afin d'indiquer les liens entre les différents fichiers indispensables à la compilation du programme, ainsi que les chemins d'accès aux differentes ressources de la SDL.

```
PROG = partie
OBJ = menu_accueil.o Partie.o menu_pause.o matrice.o afficher.o liste.o
CC = gcc
CFLAGS = -g -W



prog: ${OBJ}
	${CC} ${FLAGS} ${OBJ} -o ${PROG}

menu_accueil.o : menu_accueil.c
	${CC} ${FLAGS} -c menu_accueil.c

Partie.o : Partie.c
	${CC} ${FLAGS} -c Partie.c

menu_pause.o : menu_pause.c
	${CC} ${FLAGS} -c menu_pause.c

matrice.o : matrice.c
	${CC} ${FLAGS} -c matrice.c

afficher.o : afficher.c
	${CC} ${FLAGS} -c afficher.c
```

FIGURE 17 – Extrait de Makefile du projet

7.3 Doxygen

Doxygen est un outils assez puissant permettant de générer, dans un fichier externe au code principal, tous les commentaires associées aux différentes fonctions du programme.

Les commentaires doivent être accompagnés d'une syntaxe particulière afin que Doxygen puisse parcourir le code et remette en forme, dans une page HTML, l'ensemble des commentaires de l'ensemble du programme.

```
\fn int piece_dispo(int piece, int* nb_piece, int tab_piece[], t_liste* liste, t_matrice* copie)
\brief Verifie si la piece choisie (int piece) est bien une piece disponible. Si c'est le cas on copie tout les information
\param piece Numero de la piece concernée
\param tabpiece Tableau contenant les numeros des pieces disponibles
\param liste Pointeur de type t_liste contenant toutes les pieces d'un joueur
\param copie Pointeur de type t_matrice recevant la piece disponible
```

FIGURE 18 – Exemple d'utilisation des balises Doxygen

7.4 Debugage

Le debugage est une partie très importante dans le déroulement d'un projet, à laquelle nous sommes souvent confronté. En effet, lors de la compilation, nous sommes sans arret soumis à des erreurs. Celles-ci peuvent être repérées et modifiées en parcourant le code de visu. Cependant dans la plupart des cas, l'erreur est très difficilement repérable, il est donc necessaire d'utiliser des outils de débugages. Un d'entre eux est particulièrement efficace et peut s'utiliser directement dans le terminal : le GDB.

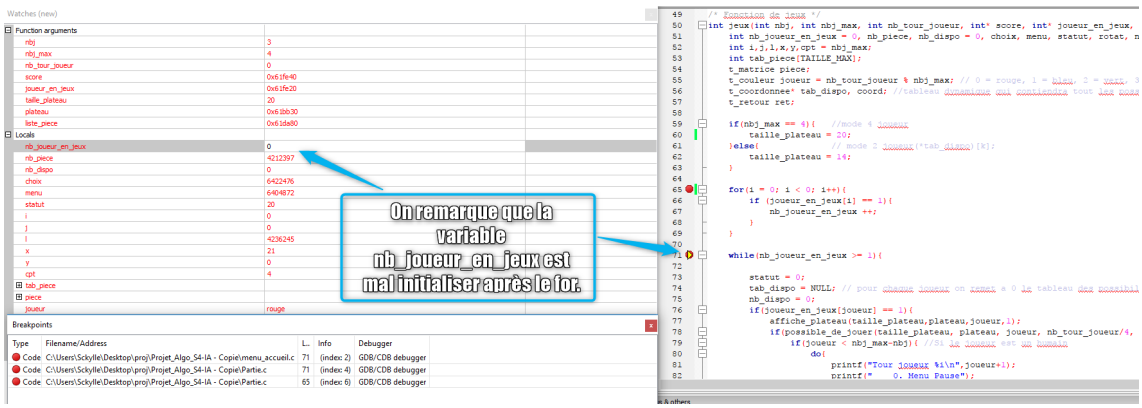

FIGURE 19 – Exemple de debugage au GDB sur Code : :Blocks

Sous Code : :Blocks, lors du debugage au GDB les variables sont visibles en temps réel. En plaçant des break-points aux endroits judicieux, nous avons pu trouver l'erreur qui était présente dans le programme.

7.5 Test

Les tests, comme leur nom l'indique, servent à fournir aux fonctions ou aux programmes eux-mêmes un jeu de données très varié afin de générer le plus d'erreurs possible pour les anticiper. De manière générale si la compilation d'un programme ne nous retourne aucune erreur cela ne veut pas dire qu'il fonctionne.

Dans notre cas, il était important de tester les différentes valeurs que peut contenir la matrice de jeu. Pour se faire nous avons créé un programme de test permettant de tester différentes valeurs aussi diverses que variées.

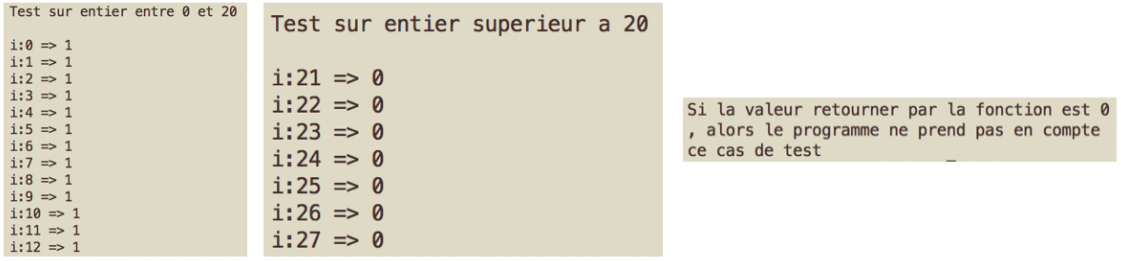

FIGURE 20 – Exemple pour des entiers

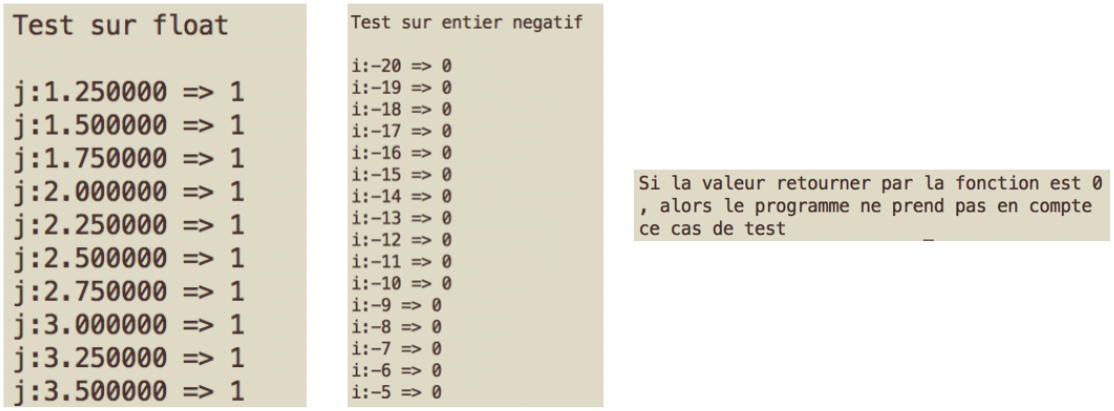

FIGURE 21 – Exemple pour des floats

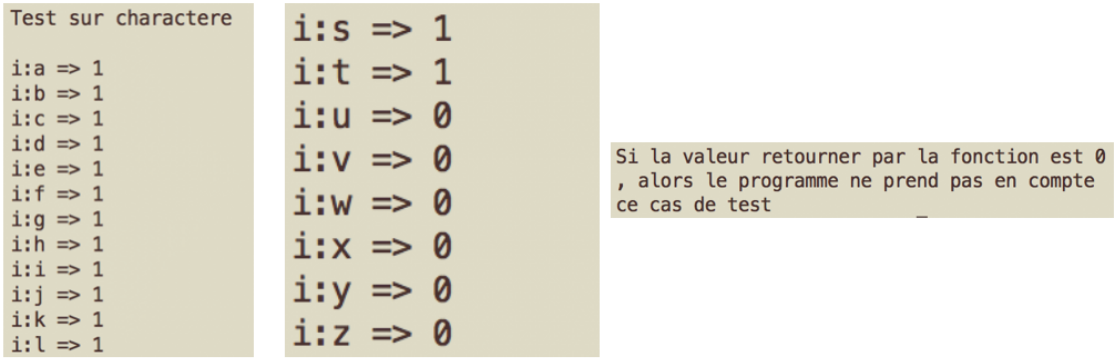

FIGURE 22 – Exemple pour des caractères

8 Nos Résultats

8.1 Version terminal

Nous sommes très satisfait de la version terminal, celle-ci correspond parfaitement avec l'idée que l'on avait d'elle aux prémices du projet. Toutes les fonctionnalités prévues au niveau du moteur ont étés intégrées. De plus, une grande partie des fonctionnalités optionnelles ont été implémentées tel que le mode 2 joueurs, la vérification des possibilités restantes, plusieurs types de rotation et l'interface graphique.

Pour ce qui est de l'intelligence artificielle, celle-ci fonctionne pour le mode local. On a la possibilité de choisir le nombre d'IA pour une partie (de zéro à trois). La difficulté est moyenne, car la profondeur de l'elagage est fixée à trois. Elle ne possède pas de coups aléatoires, ce qui la rend donc prévisible sur le long terme. De même, elle n'a pas été pensé pour bloquer les autres joueurs mais elle cherche principalement à se trouver un maximum d'ouvertures.

Il était important pour nous d'aborder toutes les notions que pouvait proposer le jeu du blokus, ce qui aussi inclut la possibilité de jouer en réseau. En réseau, le mode 4 joueurs n'est pas disponible car plus compliqué à mettre en place que prévu. Tout d'abord, le réseau a été une des dernières fonctionnalités implémentées et par conséquent le temps pour le mettre en place était réduit. De plus, l'implémentation du réseau se faisait sous deux systèmes d'exploitation différents qui ne possèdent pas les mêmes bibliothèques. A partir de là, les fonctions selon le système d'exploitation ne sont pas toutes les mêmes et certaines ont des différences subtiles. Par exemple, la fonction d'envoi de données send permet d'envoyer n'importe quel type de donnée sous linux, mais uniquement des chaînes de caractères sous Windows. Enfin, l'idée pour pouvoir gérer 4 clients était de leur attribuer un numéro lorsqu'ils se connectaient au serveur et de leur envoyer pour savoir quand ils pouvaient jouer. Cependant, les clients ne recevaient pas leur numéro et passaient directement à la suite du programme, peu importe le nombre de tentatives d'envoi du numéro.

8.2 Version interface graphique

Si l'on vient mettre en comparaison le projet initial avec celui d'aujourd'hui, il est vrai que tout est bien différent. Cette nette différence n'est pas forcement gage de non-réussite mais il est important de comprendre pourquoi et surtout qu'est ce qui a entraîné ce résultat.

Tout d'abord, il est indéniable que nous avons mal anticipé le temps nécessaire à la compréhension et à la manipulation des éléments qui composent ces bibliothèques. Il est vrai qu'une fois bien assimilé, l'organisation d'une fenêtre et la mise en place des éléments n'est pas très compliqué mais cette période de prise en mains aurait dû être prise en compte plus tôt pour se dégager plus de temps sur la fin du projet.

Il est une fois de plus important de rappeler que l'interface graphique est optionnelle, elle n'est présente que pour donner à l'utilisateur une meilleure expérience de jeu. Nous avons notre grille, nous déplaçons des images à l'aide de la souris mais comment gérer l'affectation des contraintes ? En effet, un jeu est caractérisé par ses règles, le blokus a aussi les siennes qu'il est impératif d'intégrer également dans l'interface. Nous avions donc pour mission de fusionner le moteur du jeu avec les fonctions de la SDL afin que la pièce puisse être apposée ou non à tel endroit. L'association de ces deux parties a demandé beaucoup de travail pour notamment pour une raison : ces deux travaux ont été réalisés séparemment et par des personnes différentes. Ces deux programmes qui ont été pensés, à tort, pour fonctionner indépendamment l'un de l'autre. Cela a donc entrainé une phase de restructuration des deux programmes pour qu'ils puissent être complémentaires en les fusionnant.

Pour ce qui est du résultat dans son ensemble, si nous avions à donner un avis sur celui-ci, nous sommes plutôt satisfait de ce que l'on a réalisé avec l'interface graphique. Il est vrai que le graphisme en lui-même n'est pas très abouti mais il répond efficacement aux attentes que l'on a de lui. Nous aurions aimé cependant présenter quelques fonctions que possède la version terminal comme la rotation miroir ou la possibilité de jouer en réseau.

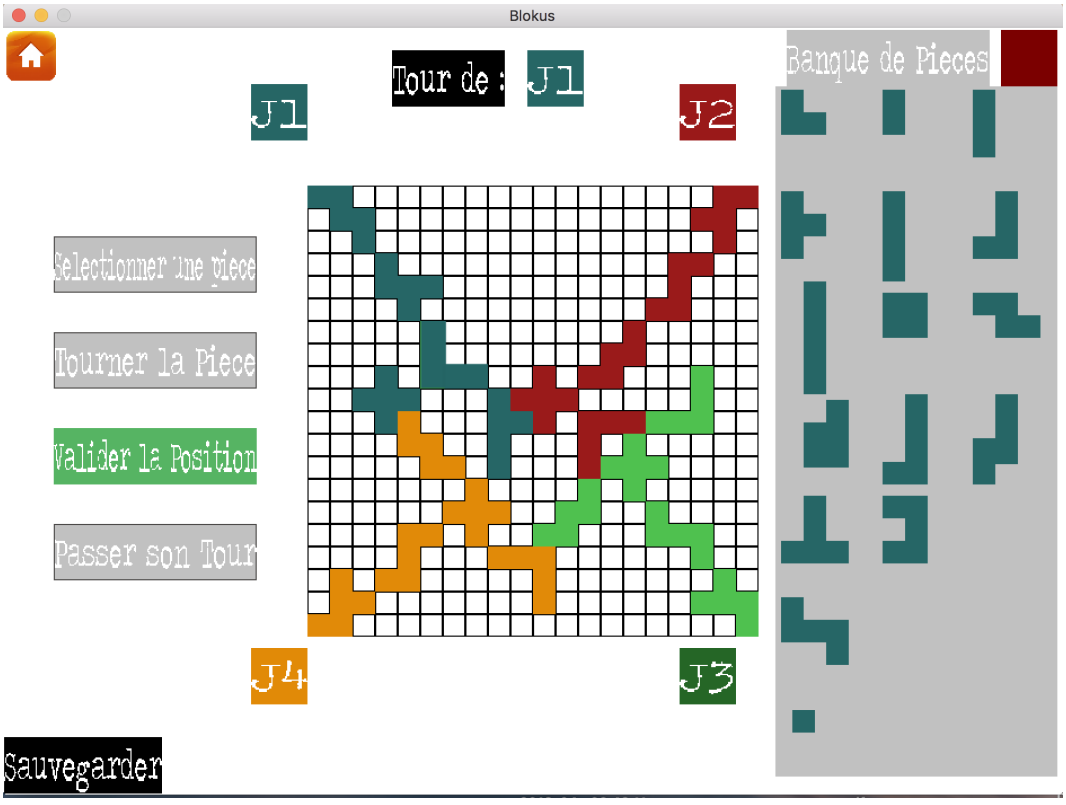

FIGURE 23 – Partie en cours - Jeu du blokus - Version Interface Graphique

9 Conclusion

Le Blokus a été l'opportunité pour nous de révéler concrètement nos compétences, de prendre conscience de nos acquis et de ce que nous pouvions réaliser. Les problèmes rencontrés au fur et à mesure ont développés notre capacité à comprendre un problème et à chercher sa résolution, que ce soit seul sur les documentations ou par le biais de la collaboration dans l'équipe. En d'autres termes, à travers ce projet, nous avons aussi bien appris l'autonomie que la collaboration.

Nous trouvons tous, personnellement, que construire un projet de A à Z est très enrichissant, de par toutes les notions différentes auxquelles on doit s'interesser, les champs de connaissances que cela implique et surtout, la satisfaction que l'on a eu à la fin de ce projet d'avoir abouti à un jeu fonctionnel.