École Centrale Lyon

UE PRO
PE 73
Rapport

Développement d'une application pour le jeu Blokus

Élèves :
Anas Fail
Gael Luguern
Jean-Charles Noirot Ferrand
Jules Coulon
Maxime Roucher
Mohamed Bouchafaa

Tuteurs :
M. Philippe Michel
M. Alexandre Saidi
M. Abdel-Malek Zine

Conseillers :
M. Cédric Gamelon
M. Simon Bachellier

27 janvier 2023

Résumé

Dans ce rapport sera présenté le travail réalisé par le groupe d'étudiants de l'École Centrale de Lyon du projet d'études numéro 73. Ce projet d'études consiste en le développement d'une application permettant de jouer au jeu de plateau Blokus en équipe avec interface et intelligence artificielle.
Les objectifs concernant à la fois la forme et le fond, l'interface graphique et l'**I.A.** ont d'abord été considérées individuellement puis liées de manière à avoir une application répondant à l'ensemble des objectifs. L'interface est constituée d'un menu et d'un tutoriel et permet de jouer au Blokus en respectant les règles et de choisir pour adversaire ou équipier une **I.A.** ainsi que sa difficulté. L'**I.A.** quant à elle repose sur deux notions : la notion de fonction d'évaluation et celle de fonction de recherche. La fonction d'évaluation permet d'attribuer un score à une position donnée tandis que la fonction de recherche permet d'obtenir le meilleur coup par recherche dans un arbre en utilisant la fonction d'évaluation.
Ce rapport met en évidence d'une part la construction des différents éléments de l'interface allant du menu à la représentation du jeu et d'autre d'une part la démarche de construction de la fonction d'évaluation par différents modèles et de choix d'algorithme pour la fonction de recherche. Il conclut sur la jonction entre l'interface et l'**I.A.** permettant d'obtenir l'application finale.
Plusieurs perspectives peuvent être considérées comme l'ajout de la possibilité de jouer en réseau, l'amélioration de la forme de l'interface et l'optimisation de l'**I.A.** par implémentation d'autres méthodes de recherche ou par modification du modèle final de la fonction d'évaluation.

Abstract

In this report will be presented the work realized by the group of ECL students from the study project number 73. This study project consists in the development of an application allowing to play the game Blokus in team with an interface and artificial intelligence. As the objectives affect both the style and the context, the graphic interface and the A.I. were firstly considered individually, then linked in order to create an application which answers all the objectives. The interface is made of a menu and a tutorial and allow the player to play Blokus while respecting the rules and to choose an A.I. as an opponent or ally and its difficulty. The A.I. is based on two concepts : the concept of evaluation function and the concept of research function. The evaluation function gives a score to a certain position in the game, while the research function provides the best move by tree search using the evaluation function.
On one hand, this report show how the different elements of the interface have been built from the menu to how the game is represented. On another hand, it shows how the evaluation function and the research function have been created using different models for the first and different algorithms for the second. It concludes on the link between the interface and the A.I. allowing to create the final application.
Some perspectives can be considered such as adding the possibility to play online, improving the style of the interface and optimizing the A.I. by implementing other search methods or by changing the final model of the evaluation function.

Remerciements

Nous tenons à remercier d'une part Philippe Michel, Alexandre Saïdi et Abdel-Malek Zine, nos tuteurs scientifiques de projet, pour leur investissement et leurs conseils qui ont permis de mener à bien la conduite de ce projet d'études. D'autre part, nous tenons à remercier Simon Bachellier, notre conseiller en communication, et Cédric Gamelon, notre conseiller en gestion de projet, qui par leur expertise dans leurs domaines respectifs nous ont permis d'améliorer la conduite de notre projet d'études aussi bien dans la forme que dans le fond.

Table des matières

Table des figures

Glossaire

Naif : Un des modèles de la fonction d'évaluation se concentrant sur la taille et la forme des pièces posées, indépendamment de leur position.

Micro : Un des modèles de la fonction d'évaluation se concentrant sur la position d'une pièce donnée, d'où son appellation.

Macro : Un des modèles de la fonction d'évaluation se concentrant sur la position générale des pièces, d'où son appellation.

I.A. : Acronyme pour Intelligence Artificielle, signifiant ici la partie de l'application qui peut jouer au jeu de manière plus ou moins bien.

Profondeur : En théorie, la hauteur de l'arbre de recherche, mais se confond ici avec le nombre de coups d'avance.

Complexité : En informatique, la complexité d'un jeu est un ordre de grandeur du nombre de parties différentes possibles.

Méthode : En informatique et plus particulièrement en orienté objet, une méthode est une routine membre d'une classe, elle permet de faire des opérations (mathématiques ou logique par exemple) sur un objet de la classe. Elle peut être vue comme une "fonction" qui prend en paramètres des attributs de l'objet de la classe.

Jeux séquentiels à informations complètes : Ce sont des jeux qui se jouent coup par coup et dont toutes les informations sont disponibles par tous les joueurs de la partie.

1 Introduction

Les jeux de réflexion, depuis le développement de la recherche dans la théorie des jeux et l'informatique, ont souvent fait l'objet d'études permettant l'implémentation d'intelligences artificielles de différentes complexités. Un exemple notable est celui des échecs qui ont révolutionné la vision du monde sur l'intelligence artificielle le 11 mai 1997 lorsque Deep Blue (**I.A.** d'IBM) a battu Garry Kasparov (meilleur joueur d'échecs à l'époque) en 19 coups. Depuis ce jour, les intelligences artificielles n'ont cessé de s'améliorer et maintenant, les **I.A.** servent de référence aux meilleurs joueurs pour s'améliorer. La création d'**I.A.** constitue donc un enjeu particulièrement important lors de la création de jeux sous forme d'application.

Blokus est un jeu de société créé en 2000 par Bernard Tavitian (Centrale Paris, promo sortante 1984). Il gagna 26 récompenses (dont l'As d'or Jeu de l'année 2001) depuis sa création et fut rapidement populaire. Il y a même eu des compétitions organisées dans plusieurs pays comme la France et les États-Unis. Dans ce jeu, chaque joueur possède des pièces de différentes tailles et formes et doit en placer un maximum sur le plateau en suivant les règles (voir 2.2) tout en bloquant ses adversaires. Sur ce principe simple, il a été jugé intéressant de faire une application Blokus et son intelligence artificielle associée. D'autant plus que Blokus satisfait les enjeux du projet d'études "Coopération Compétition" à savoir la création d'une intelligence artificielle qui peut jouer avec ou contre d'autres joueurs sur un jeu multijoueur. Il émerge ainsi la problématique suivante : comment construire une application permettant de jouer au Blokus avec ou contre une **I.A.** ?

Deux parties seront abordées. La première traite de l'interface graphique : elle montre les différents résultats de l'interface et explique comment le jeu est représenté et La deuxième traite de l'intelligence artificielle et introduit les notions de fonction d'évaluation et fonction de recherche.

2 Contexte

2.1 Le jeu du Blokus

Le jeu de Blokus est un jeu de plateau récent (créé en 2000 par Bernard Tavitian) très similaire au jeu de go en ce qui concerne le placement des pièces. Le matériel utilisé est un plateau quadrillé (20x20) composé de 400 carrés de taille 1 où 4 joueurs s'affrontent afin de pouvoir placer un maximum de pièces. Chaque pièce est composée de carrés de taille 1. Tous les joueurs commencent la partie avec le même nombre et type de pièces.

FIGURE 1 – Image montrant un plateau de jeu du Blokus.

Tandis que les règles (voir 2.2) et l'apparence du jeu Blokus paraissent simples, programmer un jeu de plateau sur ordinateur n'est pas aisé lorsque l'on n'est pas familier au développement d'application sur ordinateur.

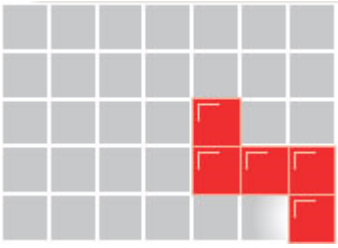
En outre, une variante au jeu originel a été implémentée. Celle-ci consiste à considérer que les joueurs en diagonales doivent coopérer pour gagner, ainsi le score n'est pas individuel mais par équipe. Chaque équipe doit occuper le maximum d'espace et pour se faire définir des stratégies pour contrer l'équipe adverse. Cette variante est un élément important qui doit être pris en compte lors de la création de l'intelligence artificielle.

2.2 Règles du Blokus en équipe

Dans un souci de clarté, cette partie présente succinctement les règles du jeu du Blokus comme vu en [1].

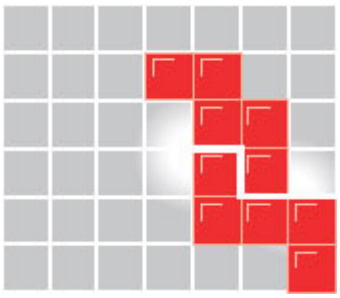
- Chaque joueur se voit assigner une couleur (vert, bleu, jaune, rouge), 21 pièces et un coin de départ. Les joueurs d'une même équipe sont diagonalement opposés.
- La première pièce posée doit avoir un carreau dans le coin du plateau, voir figure 2.
- Chaque pièce posée doit toucher une pièce de la même couleur par un ou plusieurs coins, mais jamais par les côtés, voir figure 3.
- Chaque pièce posée rapporte autant de points que le nombre de carreaux qui la composent.
- Lorsque aucun joueur ne peut placer de pièce, la partie est terminée et le score des équipes s'obtient en sommant les scores des joueurs de l'équipe.

(a) Première pièce mal positionnée

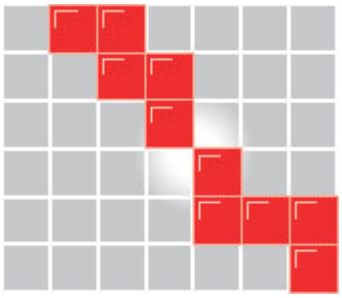


(b) Première pièce bien positionnée

FIGURE 2 – Positionnement de la première pièce



(a) Mauvais positionnement



(b) Bon positionnement

FIGURE 3 – Positionnement d'une pièce

2.3 Présentation du projet

Le projet consiste à programmer le jeu du Blokus sur ordinateur et y intégrer une intelligence artificielle qui joue de manière non aléatoire.
Le livrable, qui est une application Windows, devra satisfaire un cahier des charges qui est :

- Pouvoir jouer sur un ordinateur personnel.
- L'utilisateur ne doit pas pouvoir faire de coup interdit.
- Faire jouer l'ordinateur en respectant les règles du Blokus.
- Le logiciel doit avoir une interface intuitive.
- Le logiciel doit pouvoir être lancé sur tout type d'ordinateur.
- Le logiciel doit être fluide.
- L'**I.A.** doit jouer rapidement (temps de réponse inférieur à 5 secondes).
- Le logiciel doit contenir un tutoriel pour les nouveaux joueurs.

2.4 État de l'art

Le jeu de Blokus n'est pas assez connu pour avoir eu des projets de recherches d'intelligences artificielles spécifiques aux jeux. Sur internet, il y a seulement quelques projets pédagogiques sur le sujet comme [2]. Il y a quelques applications téléphones qui permettent de jouer au Blokus avec une **I.A.** mais aucune n'explique le fonctionnement de leurs IA.

Il y a par contre beaucoup de littérature sur des jeux similaires au Blokus. C'est-à-dire des **Jeux séquentiels à informations complètes**. Les jeux les plus connues de ce type sont les échecs, le jeu de GO ou le puissance 4. Sur la base de ces recherches, il a pu être possible de choisir de la meilleure manière possible le modèle d'**I.A.** le plus adapté au Blokus.[3]

2.5 Solution technique retenue

Dans le cadre de la réalisation d'un programme informatique, plusieurs langages de programmation peuvent convenir. Afin de choisir un langage en particulier, un tableau comparatif a été réalisé, il contient 5 langages de programmation (Python, C, Csharp, C++ et Java) ces différents langages sont ordonnés du plus haut niveau au plus bas niveau. ce tableau est présenté figure 4 :

Langage informatique	Paradigmes et niveau	Bibliothèques disponibles pour réaliser un jeu	Avantages	Inconvénients
Python	Objet, impératif et fonctionnel Langage très haut niveau	Pygame	Facile à prendre en main et maitrisé par tous les membres du groupe	chronophage et utilisation de la mémoire non optimale. Temps de calcul très élevé Interface graphique très basique
CSharp (C#)	Structuré, impératif et orienté objet Langage de moyen niveau	Monogame	Intéressant à apprendre et très répandue dans l'industrie. Temps de calcul peu élevé (si bien utilisé) Optimal pour coder un jeu video.	Langage typé et orienté objet. Les membres du groupes n'y sont pas habitués.
Java	orienté objet, impératif, structuré et orienté objet Langage de moyen niveau	LWJGL (Light Weight Java Game Library)	possibilité de faire de la programmation orientée objet, ne dépend pas du système d'exploitation.	nécessite un espace mémoire énorme difficile à installer et à prendre en main pour des débutants.
C++	générique, procédurale et orienté objet Langage bas niveau	OpenGL Allegro	Très utilisé dans l'industrie. Bonne gestion de la mémoire et temps de calcul peu élevé (si bien utilisé)	Langage très typé, difficile à maitriser en autonomie et donc nécessite beaucoup de temps d'apprentissage.
C	Impératif, procédural et structuré Langage très bas niveau	SDL (Simple DirectMedia), Allegro	Possibilité d'optimiser l'utilisation de la mémoire.	Gestion totale de la mémoire et donc demande un haut niveau de compétences. Plusieurs membres du groupe ne sont pas familiers avec.

FIGURE 4 – Tableau de choix de langage

Le langage de programmation CSharp (C#) se démarque par sa facilité d'apprentissage et son utilité. Ce langage est aussi très présent dans l'industrie du jeu vidéo, ce qui en fait un candidat idéal pour ce projet. En outre, plusieurs membres du groupe sont familiers avec ce langage. Donc, il a été retenu de réaliser ce projet en CSharp (C#).

Afin de pouvoir réaliser le projet, le groupe s'est formé à travers des outils en ligne comme Openclassroom [4]. Mais aussi en ayant recours à des forums spécialisés en ligne comme Stack Overflow [5]

3 Interface graphique

3.1 Introduction

L'objectif de cette partie est de détailler le fonctionnement de l'interface. En effet, une majorité des points du cahier des charges portent sur l'interface. Elle doit être facile à prendre en main et avoir un affichage fluide. Une des grandes difficultés est de faire jouer aussi bien les utilisateurs que les **I.A.**, bien que leurs interactions avec l'interface soient très différentes. Les parties suivantes précisent les solutions apportées ainsi que leur implémentation.

3.2 Menu

Le menu est le premier écran que l'utilisateur voit en laçant l'application. Il comporte 3 boutons. Le bouton Play permet de lancer la partie, le bouton Help permet de lancer le tutoriel et le bouton Leave ferme l'application.

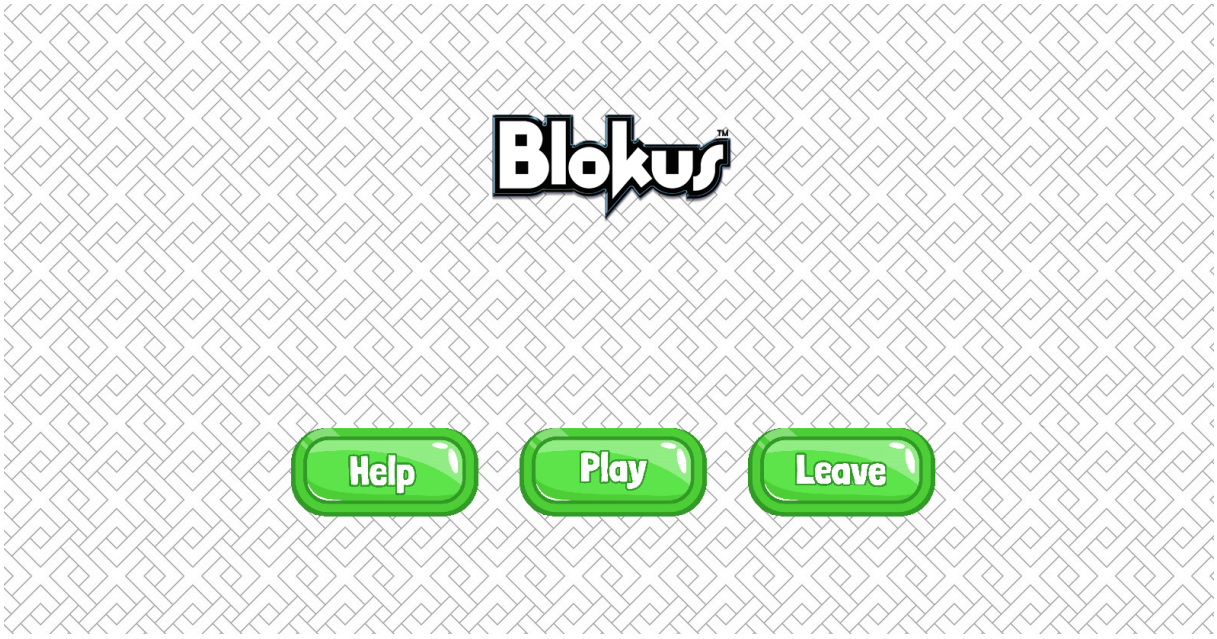


FIGURE 5 – Capture d'écran du menu

Une fois le bouton Play cliqué, un nouvel écran s'affiche. Il comporte 6 boutons. 4 permettent de choisir pour chaque joueur, s'il est utilisateur ou **I.A.** ainsi que son niveau le cas échéant. Un bouton Back permet de revenir au premier écran et un bouton Play permet de lancer la partie.

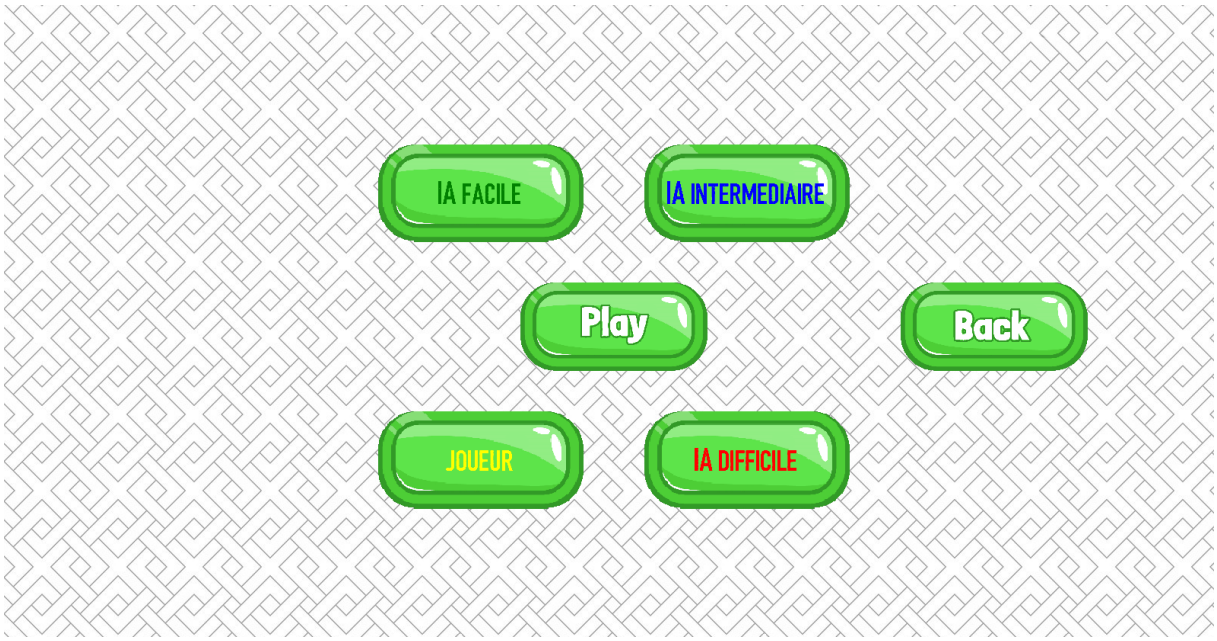
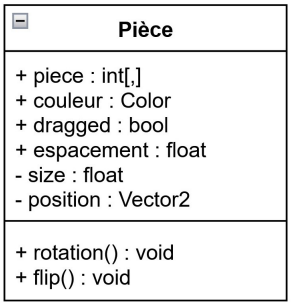


FIGURE 6 – Capture d'écran du menu

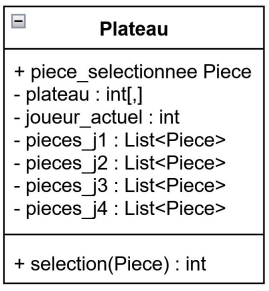
3.3 Représentation du jeu

Puisque le jeu est un jeu sur un plateau carré, il est apparu adapté d'utiliser une représentation matricielle des pièces et du plateau. Ainsi, le placement, la rotation et le renversement des pièces sont de simples opérations sur leurs matrices De plus, afin de manipuler plus facilement ces matrices, plusieurs classes ont été utilisées, afin de stocker une pièce, la liste de toutes les pièces du jeu et un plateau où l'on peut poser ces pièces.

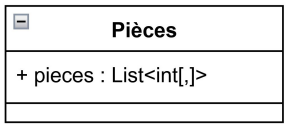
Les diagrammes suivants détaillent les classes ainsi que leurs interactions.



(a) Diagramme UML de la classe Pièce



(b) Diagramme UML de la classe Plateau



(a) Diagramme UML de la classe Pièces

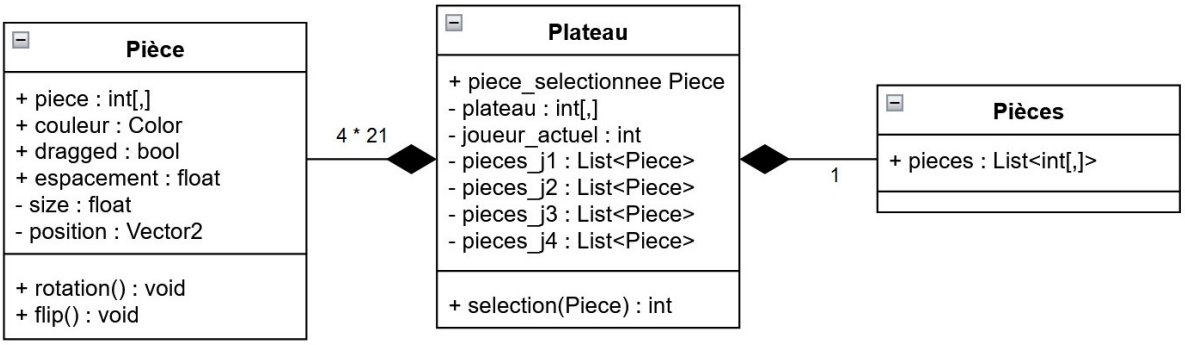


FIGURE 9 – Diagramme UML de l'interaction entre les trois classes Pièce, Pièces et Plateau

Le plateau est composé de 4 listes de 21 pièces identiques, dont les matrices sont stockées dans Pièces.

3.4 Fonctionnement du jeu

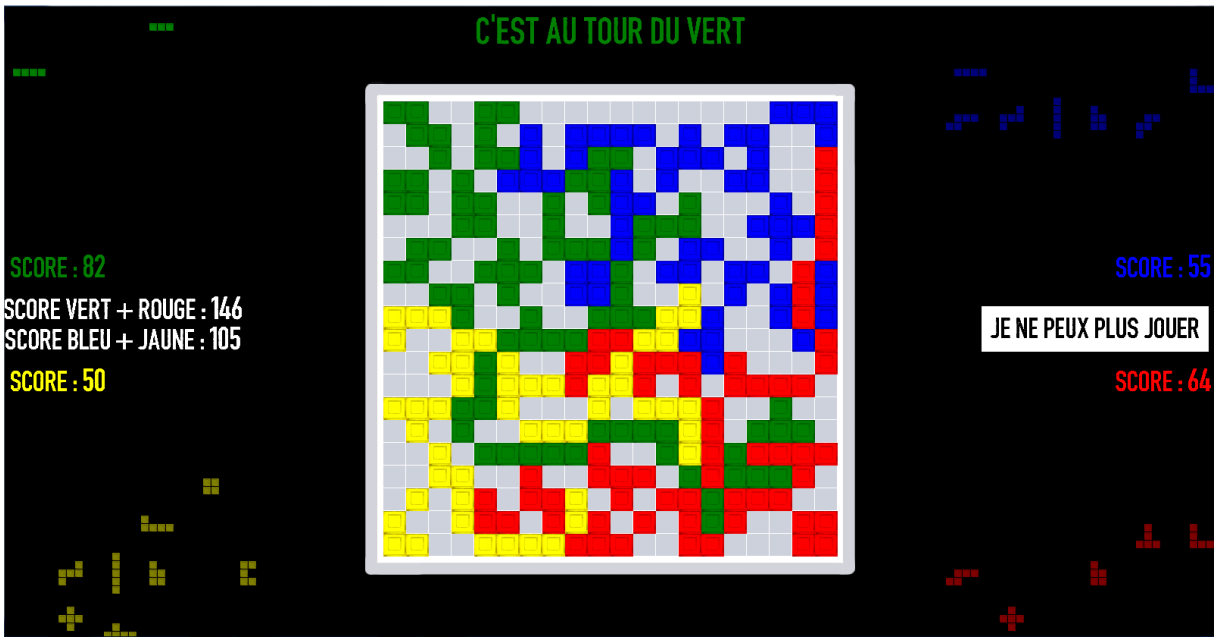


FIGURE 10 – Capture d'écran d'une partie en cours

L'image ci-dessus est une capture d'écran de l'interface au cours d'une partie, elle montre la mise en place de chacun des éléments

Afin de faire jouer les joueurs les uns après les autres, un entier stockant le numéro du joueur est mis à jour après chaque coup, les pièces d'un joueur ne sont interactives que si cet entier est celui du joueur, ce que le texte en haut traduit.

Quand un joueur peut jouer, il utilise la souris pour cliquer sur une pièce, qui est donc sélectionnée, il peut ensuite placer cette pièce sur le plateau en déplaçant la souris. Le logiciel vérifie en direct que positionner la pièce à cet endroit respecte toutes les règles du jeu et l'affiche en violet si ce n'est pas le cas. Le joueur peut également faire tourner la

pièce de 90° en appuyant sur les flèches droite et gauche du clavier et la renverser avec les flèches bas et haut du clavier. Une fois la pièce en place, il suffit à l'utilisateur de cliquer pour la poser. L'application passe alors au joueur suivant.
Les scores de chaque joueur ainsi que les scores par équipes sont affichés de part et d'autre du plateau pour permettre à chaque joueur de connaître l'état de la partie.
Pour les **I.A.**, le coup à jouer est donné par les algorithmes détaillés ci-dessous (voir 4 . Une classe Coup prenant en argument une pièce et une position a été créée pour ce faire.

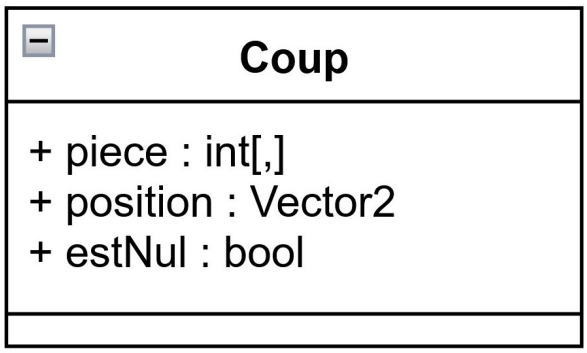


FIGURE 11 – Diagramme UML de la classe Coup

Si ces algorithmes retournent un coup jouable (si le booléen estNul est faux), la matrice de la pièce est alors positionnée à la position donnée, sinon l'IA abandonne et passe définitivement ces tours. L'application passe alors au joueur suivant.

3.5 Tutoriel

Le tutoriel est une suite de pages accessible depuis un bouton dans lesquelles une image illustre une explication données par le texte. Ce tutoriel sert à expliquer tout le fonctionnement de l'application ainsi que les règles du jeu.

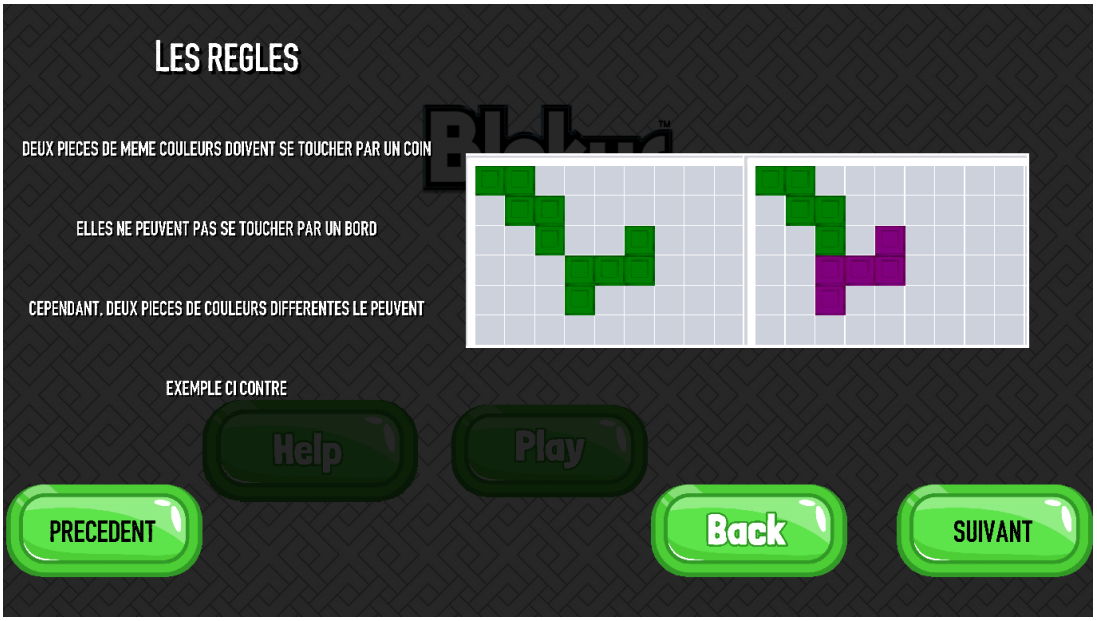


FIGURE 12 – Exemple d'une page du tutoriel

3.6 Conclusion

L'interface du jeu avait pour objectif principal de permettre à toute personne de jouer de façon intuitive et fluide en respectant les règles. Les essais du jeu par de nombreuses personnes hors du projet ont pu montrer que tous les objectifs ont bien été respectés.

4 Intelligence Artificielle

4.1 Introduction

Le premier choix à faire pour l'**I.A.** est le choix du modèle. Pour ce faire, il a fallu mesurer la **Complexité** du Blokus pour le comparer aux différents jeux de même type dont la théorie sur leur **I.A.** est connue. La **Complexité** du Blokus est donc de l'ordre de 10^{500} ce qui se situe entre les échecs (10^{120}) et le jeu de go (10^{600}). Pour ce type de jeu, il y a différents types d'**I.A.**([6]).

La première qui paraît la plus naturelle est le système expert. C'est une **I.A.** qui essaye de recopier un expert dans le jeu en utilisant ses techniques. Cette méthode n'a pas été retenu, car il faudrait avoir accès à des parties de professionnels. De plus cette technique utilise de nombreuses disjonctions de cas ce qui complique la programmation vu la **Complexité** du jeu. Cette méthode a quand même été utilisé pour l'optimisation de l'**I.A.** (voir 4.4.1)

Le second type d'**I.A.** utilise le parcours d'arbre. C'est-à-dire que l'**I.A.** simule de nombreux coups en avance pour en déduire le meilleur coup à jouer. C'est par ce type de méthode que l'on a résolu le puissance 4 et que le premier ordinateur a battu le champion du monde d'échec (Garry Kasparov). Cette technique peut vite être très chronophage quand la **Complexité** du jeu est trop importante, mais elle est relativement simple à implémenter et il existe des optimisations pour la rendre plus rapide. C'est donc ce modèle d'**I.A.** qui a été choisi et qui va être explicité dans la suite du rapport.

Les derniers types d'**I.A.** sont le deep learning et le machine learning. Ces techniques utilisent de nombreuses données de parties pour en déduire les meilleurs coups à jouer. Le deep learning permet ensuite à l'ordinateur de s'améliorer par lui-même. C'est à l'aide de ces techniques que l'on obtient les meilleurs **I.A.** d'échecs et que le champion du monde du jeu de go a été battu. Bien que meilleures, ces techniques sont beaucoup trop ambitieuses pour le projet et il faudrait un cours complet sur ces techniques pour pouvoir se familiariser avec.

4.2 Fonction d'évaluation

Dans cette partie, il s'agit d'étudier la fonction d'évaluation, un des constituants principaux de l'intelligence artificielle. En effet, cette fonction, comme son nom l'indique, permet d'évaluer (c'est-à-dire donner un score) à une personne pour une position donnée dans le jeu. Ici, de nombreux paramètres peuvent donc être pris en compte pour affiner ce résultat, par exemple les pièces posées (forme, taille, position) et l'allure générale du plateau.

Dans toute la suite, le formalisme mathématique suivant sera utilisé.
À chaque pièce est associé un indicateur $B^n_{i,j}$ où n correspond au numéro du joueur (n=1, 2,

3 ou 4) , i au nombre de cases que prendre la pièce et j et i entiers suivants la numérotation figure 13.

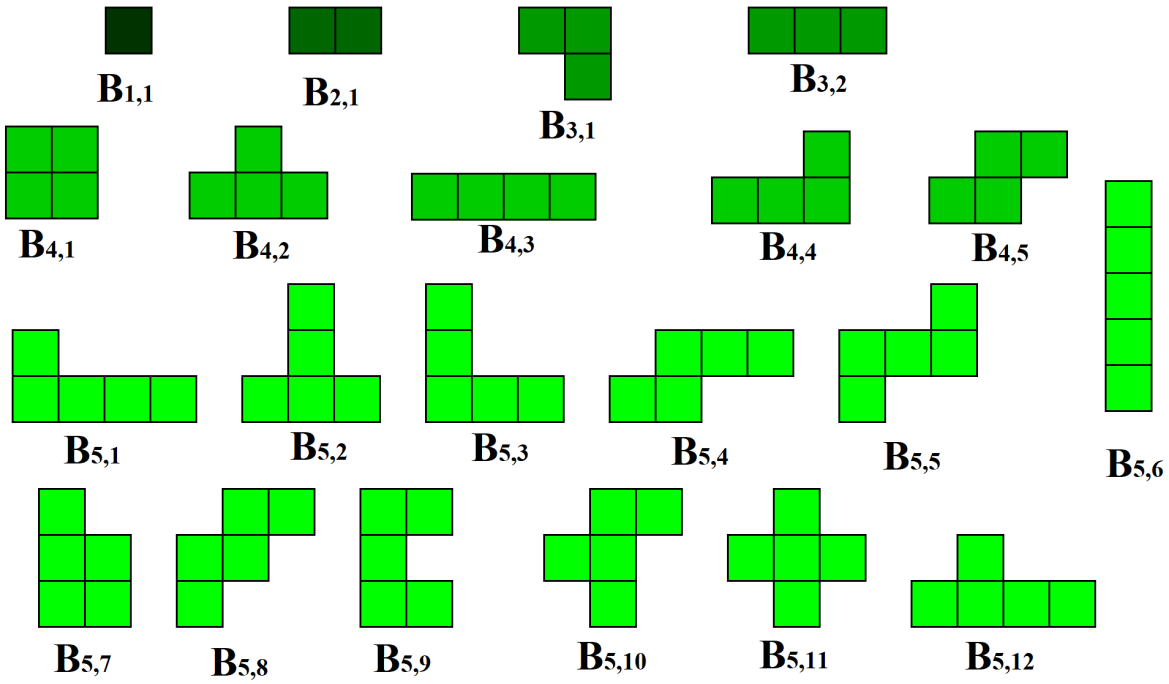


FIGURE 13 – Numérotation des pièces

4.2.1 Modèle Naïf

Dans ce modèle, ce n'est pas la position des pièces placées qui est utilisée, mais seulement nombre de pièces posées. Une analogie peut alors être faite avec une fonction d'évaluation classique pour les échecs consistant en une somme linéaire pondérée de caractéristiques. Dans le jeu Blokus, il est toujours préférable de poser les pièces prenant le plus de cases en premier. La fonction d'évaluation pour les joueurs 1 et 3 sera donc de la forme : $\sum_{i,j} P_{i,j}(B_{i,j}^{(1)} - B_{i,j}^{(2)} + B_{i,j}^{(3)} - B_{i,j}^{(4)})$ où les $P_{i,j}$ sont les poids associés à chaque type de pièce.

Choix des $P_{i,j}$ Compte tenu de la structure du jeu, il faut valoriser le fait de poser les pièces les plus difficiles à poser, donc en particulier les plus grandes. Ainsi, si j est fixé, $P_j : i \in [\![1, 5]\!]$ est une fonction croissante de i. Une première version des poids peut être trouvée en fixant $P_{i,j} = i$ et en déduisant le tableau des poids simples figure 14.

i	j	P	i	j	P	i	j	P
1	1	**1**	4	4	**4**	5	6	**5**
2	1	**2**	4	5	**4**	5	7	**5**
3	1	**3**	5	1	**5**	5	8	**5**
3	2	**3**	5	2	**5**	5	9	**5**
4	1	**4**	5	3	**5**	5	10	**5**
4	2	**4**	5	4	**5**	5	11	**5**
4	3	**4**	5	5	**5**	5	12	**5**

FIGURE 14 – Table des poids simples

Néanmoins, la géométrie des pièces doit être prise en compte pour une meilleure finesse. Une façon de prendre en compte cette géométrie serait de calculer l'aire du plus petit rectangle occupé par la pièce notée $A_{i,j}$ (voir zones rouges, figure 15). Elle est différente selon la pièce et certaines pièces peuvent avoir la même aire sans être de la même taille. Par exemple $A_{4,1} = 4$ mais $A_{4,4} = 3 \cdot 2 = 6$ et $A_{5,9} = 6$.

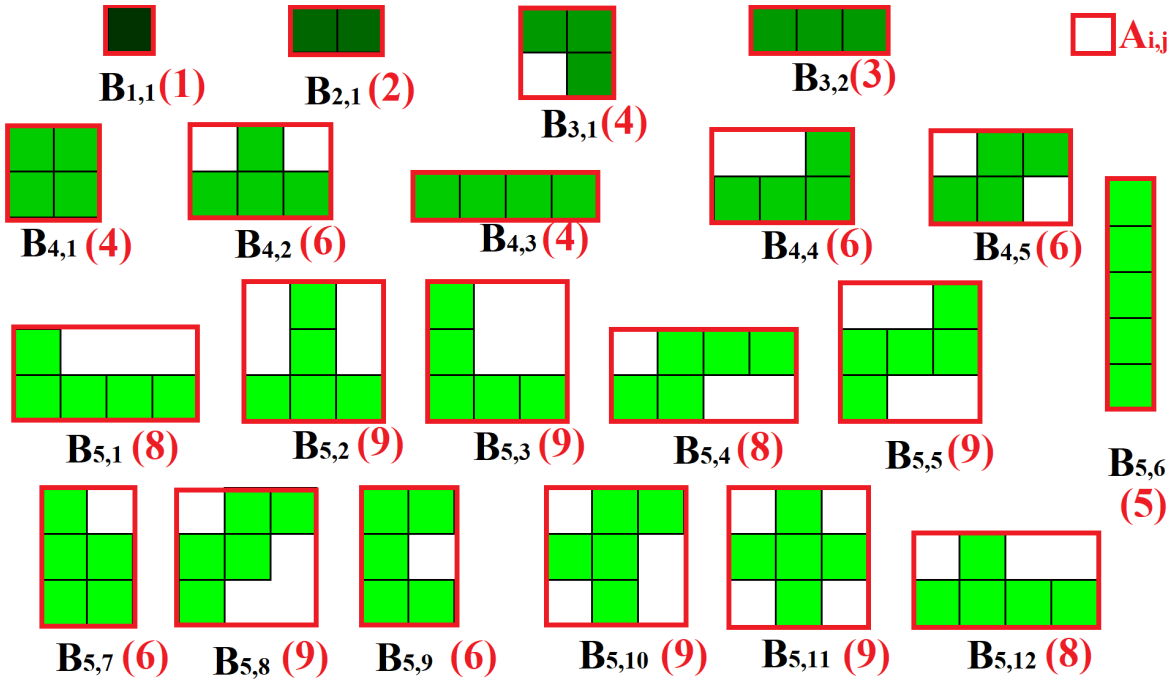


FIGURE 15 – Visualisation des $A_{i,j}$ en rouge

Chaque poids simple est alors légèrement modifié par cette différence de géométrie et ainsi une version plus fine des poids est retenue pour la suite $P_{i,j} = i + \frac{A_{i,j}}{\max\limits_{k} A_{i,k}}$ de sorte que $i \leq P_{i,j} \leq i+1$. Le tableau des poids est donc modifié et peut être visualisé figure 16.

i	j	P	i	j	P	i	j	P
1	1	**2**	4	4	**5**	5	6	**5.56**
2	1	**3**	4	5	**5**	5	7	**5.67**
3	1	**4**	5	1	**5.89**	5	8	**6**
3	2	**3.75**	5	2	**6**	5	9	**5.67**
4	1	**4.67**	5	3	**6**	5	10	**6**
4	2	**5**	5	4	**5.89**	5	11	**6**
4	3	**4.67**	5	5	**6**	5	12	**5.89**

FIGURE 16 – Table des poids avec considération de la géométrie

4.2.2 Modèle Micro

Ce modèle ne s'intéresse pas à la dynamique générale et au poids des pièces. L'objectif est d'optimiser le nombre de coups possibles. Ainsi la fonction réalisée prend en attribut la situation du plateau à l'instant t et les différentes pièces disponibles (le plateau est une matrice d'entiers et les pièces sont stockées dans une liste d'objets de type *pièce*).

La première étape est de trouver tous les coins où une pièce carrée de taille 1x1 est jouable (c'est-à-dire, peut être posée). Cette première étape est codée en deux temps ;

Premièrement, une **Méthode** appelée *IsUnCoin() :* qui vérifie si la case est un coin jouable est réalisée (au sens : un carré de taille 1x1 en respectant les règles peut y être posé). Un coin et défini comme étant une case (de taille 1x1) collée diagonalement à une case occupée par un joueur (c'est-à-dire qui a une valeur égale à $J = 1, 2, 3, 4$). Pour trouver ces coins, il faut se placer dans la case (de coordonnée i, j) occupée par le joueur J. Ainsi :

- si les cases (i+1 , j+1) et (i , j+1) et (i+1 , j) ont **chacune** une valeur différente de J, alors la case (i+1 , j+1) (qui correspond au coin haut droit) est un coin jouable.
- si les cases (i-1 , j-1) et (i-1 , j) et (i , j-1) ont **chacune** une valeur différente de J, alors la case (i , j-1) (qui correspond au coin haut gauche) est un coin jouable.
- si les cases (i+1 , j-1) et (i , j-1) et (i+1 , j) ont **chacune** une valeur différente de J, alors la case (i+1 , j-1) (qui correspond au coin bas gauche) est un coin jouable.
- si les cases (i-1 , j+1) et (i , j+1) et (i-1 , j) ont **chacune** une valeur différente de J, alors la case (i+1 , j+1) (qui correspond au coin bas droit) est un coin jouable.

un exemple est présenté dans la figure ci-dessous :

(a) Coin haut gauche

(b) Coin haut droit

(a) Coin bas gauche

(b) Coin bas droit

FIGURE 18 – Illustration de l'algorithme de détection de coins de pièces exploitables.

Ainsi, dans l'exemple ci-dessus, le coin haut droit (Figure 18) est un coin exploitable puisqu'il n'y a aucune case appartenant au joueur 1. De même pour le coin bas droit. En revanche, le coin bas gauche et le coin haut gauche ne sont pas admissibles puisque poser un carré de taille 1x1 à cet endroit ne respecte pas les règles du jeu.

Dans un second temps, une seconde méthode appelée *TousLesCoins()*est implémentée, cette méthode prend en paramètre le plateau P, le joueur J et fait appel à la **Méthode** *IsUnCoin()*. Celle-ci parcourt toutes les cases occupées par le joueur J et vérifie si oui ou non cette case est un coin à l'aide de la **Méthode** *IsUnCoin()*. Si la case est en effet un coin, alors cette case (d'indice (i,j)) est stockée dans un vecteur de tuple.
Enfin, cette méthode *TousLesCoins()* rend la liste des indices des cases qui sont des coins, la longueur de cette liste représente le nombre de cases où un carré de taille 1x1 est jouable pour le joueur J. Ainsi, la taille de cette liste est un indicateur de la situation du joueur J plus cette liste est de taille importante plus le joueur J occupe une position favorable sur le plateau et donc plus il a de chance de remporter la victoire.

4.2.3 Modèle Macro

Ainsi, en observant l'exemple ci-dessus, Dans ce modèle, le point principal est le placement des pièces. En effet, selon la dynamique générale d'une partie, un joueur peut être plus rapidement bloqué. L'idée du modèle est donc de prendre en compte cette dynamique générale.

Une idée serait donc d'utiliser les barycentres. En effet, en considérant les barycentres des pièces pour chaque joueur, il est possible de mettre en avant une stratégie plus agressive en voulant minimiser le barycentre allié et le barycentre ennemi

4.2.4 Modèle final

Maintenant que les différents modèles composant l'intelligence artificielle ont été expliqués, il convient d'établir la fonction d'évaluation finale qui est une fonction des trois

modèles précédemment expliqués. Pour construire cette fonction, les fonctions associées aux modèles sont normalisées en divisant par des majorations proches du maximum de chaque fonction.
Dans la suite, f_{Naif}, f_{Micro} et f_{Macro} correspondent aux fonctions des différents modèles. Elles prennent en argument un plateau, une couleur de joueur et la liste des pièces jouables pour chaque joueur.

Majoration des fonctions des modèles

Naif : En considérant le cas extrême où toutes les pièces de l'équipe alliée sont posées et aucune pièce de l'équipe ennemie n'est posée, il est possible d'obtenir un majorant de f_{Naif} : il suffit donc de prendre le double de la somme des poids issus de la figure 16. Ainsi, $M_{Naif} = 2\sum_{i,j} P_{i,j} = 215.32$

Micro : Pour obtenir une majoration, le cas extrême où chaque coin de chaque pièce est utilisable est considéré, la somme de ces coins permet d'obtenir la majoration. En sommant les coins de chaque pièce (voir figure 19), il vient $M_{Micro} = 112$

i	j	Coins	i	j	Coins	i	j	Coins
1	1	**4**	4	4	**5**	**5**	**6**	**4**
2	1	**4**	4	5	**6**	**5**	**7**	**5**
3	1	**5**	5	1	**5**	**5**	**8**	**7**
3	2	**4**	5	2	**6**	**5**	**9**	**5**
4	1	**4**	5	3	**5**	**5**	**10**	**7**
4	2	**6**	5	4	**6**	**5**	**11**	**8**
4	3	**4**	5	5	**6**	**5**	**12**	**6**

FIGURE 19 – Nombre de coins pour chaque pièce

Macro : Puisque la fonction f_{Macro} est égale à l'écart entre les barycentres, il est facile de majorer f_{Macro} par la longueur du plateau, soit $M_{Macro} = 20$.

Synthèse : Ainsi, en appelant le modèle finale f_{Eval}, f_{Eval} est une fonction des différents modèles normalisée par leur majorant respectif calculés précédemment, soit $f_{Eval}(\frac{f_{Naif}}{M_{Naif}}, \frac{f_{Micro}}{M_{Micro}}, \frac{f_{Macro}}{M_{Macro}})$.
Dû à un manque d'essais, cette partie de la fonction d'évaluation n'a pas pu être optimisée. Le modèle final a donc été choisi par défaut et $f_{Eval} = \frac{f_{Naif}}{M_{Naif}} + \frac{f_{Micro}}{M_{Micro}} + \frac{f_{Macro}}{M_{Macro}}$.
Ici les 3 modèles ont la même importance, mais des essais avec différentes combinaisons de modèles auraient pu permettre de trouver un meilleur modèle final (c'est-à-dire permettant d'obtenir un meilleur taux de victoire).

4.2.5 Fonctionnement du code

Le code associé à la fonction d'évaluation est plutôt simple. On se contente simplement de créer une fonction qui pour une position donnée (c'est-à-dire un plateau, une couleur de joueur et des listes de pièces des joueurs donnés) renvoie le score associé pour chaque modèle (une fonction par modèle) en utilisant les calculs expressions précédents. Le code associé à la fonction d'évaluation est disponible annexe A.1

4.3 Fonction de recherche

Les fonctions de recherche sont les fonctions au cœur de l'**I.A.** puisque c'est cette fonction qui va essayer de renvoyer le meilleur coup possible pour une partie donnée.

4.3.1 L'algorithme MinMax

Cet algorithme est la base de notre IA. Le principe est simple, l'objectif est de jouer le coup qui va mettre l'**I.A.** dans la meilleure position possible avec un nombre défini de coups d'avance (appelé la **Profondeur**).

Pour cela, tous les coups calculés sont modélisés en avance par un arbre avec les branches représentants les coups jouées, les nœuds représentants le plateau. Une branche entre deux nœuds représente le coup qui différencie les deux plateaux représentés par les nœuds (voir figure 20).

On évalue tous les plateaux de **Profondeur** maximale (en bas de l'arbre) à l'aide de la fonction d'évaluation. Cette fonction donne pour chaque plateau simulé un score qui montre quelle équipe est le mieux dans la partie en fonction des coups joués.

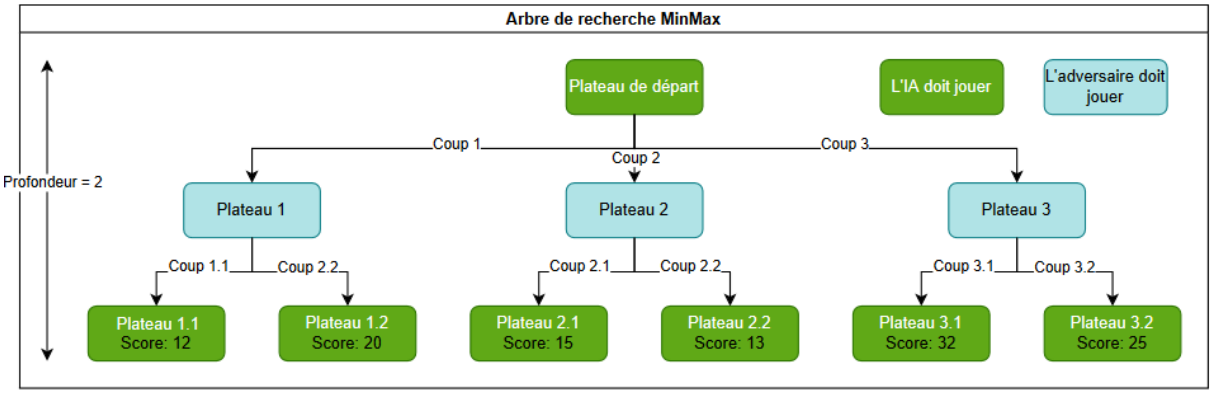


FIGURE 20 – Arbre de recherche (1/2)

Maintenant que l'arbre est complété, il ne reste plus qu'à choisir le meilleur coup à jouer. C'est là que l'algorithme MinMax rentre en jeux.

En supposant que l'adversaire va jouer le meilleur coup possible, c'est-à-dire qu'il va jouer le coup qui minimise le score de l'**I.A.**. Il va donc choisir le chemin du minimum des scores des plateaux d'en dessous. Cette valeur est alors mise dans le nœud du plateau comme montrer sur la figure 21.

C'est ensuite au tour de l'**I.A.** qui veut jouer le meilleur coup possible. L'**I.A.** va donc choisir le coup qui l'amène à un plateau le plus avantageux pour elle. Elle va donc prendre le chemin qui prend le maximum de tous les plateaux possibles.

En alternant comme cela pour chaque coup jusqu'au plateau de départ, le meilleur coup de l'**I.A.** est trouvé.

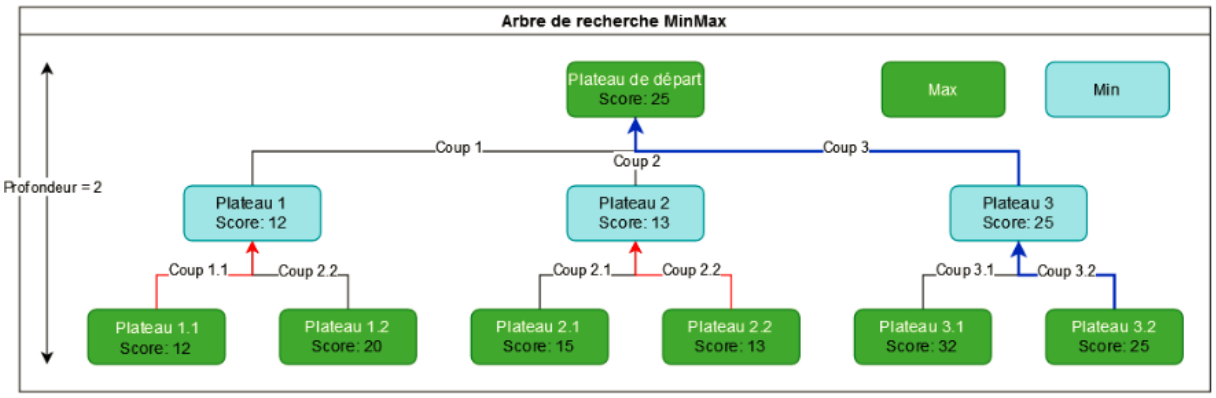


FIGURE 21 – Arbre de recherche (2/2)

4.3.2 Fonctionnement du code

Cet algorithme reprend le fonctionnement décrit ci-dessus. La difficulté vient plutôt des fonctions auxiliaires utilisées.

En effet, cet algorithme a besoin d'une fonction qui trouve tous les coups possibles pour un plateau donné.

Cette fonction commence par chercher tous les coins possibles pour un joueur donné en regardant tous les coins de toutes les cases possibles et en gardant les ceux qui respectent les règles (voir section 4.2.2) Ensuite cette fonction prend toutes les pièces et essaye de les poser sur chaque coin et dans toutes les directions possibles. Elle garde alors toutes les positions qui respectent les règles.

L'algorithme MinMax a aussi besoin d'une fonction qui prend un plateau et un coup donné et renvoie le nouveau plateau. Cette fonction a été implémentée dans la partie graphique.

La dernière fonction auxiliaire utile est la fonction d'évaluation. Plus cette fonction est précise et représente au mieux le plateau, plus l'**I.A.** sera performante et fera les meilleurs choix.

4.4 Optimisation

4.4.1 Ouverture

Une des optimisations pour l'intelligence artificielle a été d'implémenter une ouverture (à l'image du jeu d'échecs) qui fait partie de la stratégie de l'équipe. En effet, étant donné que les débuts de parties sont très calmes, c'est-à-dire que les pièces des joueurs sont encore très éloignées. Ainsi, il est possible de prédéfinir certains coups optimaux sur l'**I.A.** sans que cela puisse être puni par les autres joueurs. Une ouverture portant le nom du joueur Jeff Barasona qui l'a trouvé repose sur deux principes : s'étendre rapidement vers le centre et rendre l'attaque de l'adversaire plus difficile[7]. Cette ouverture peut être visualisée figure 22. Puisqu'aucune autre réelle ouverture n'existe, c'est celui-ci qui sera utilisé.

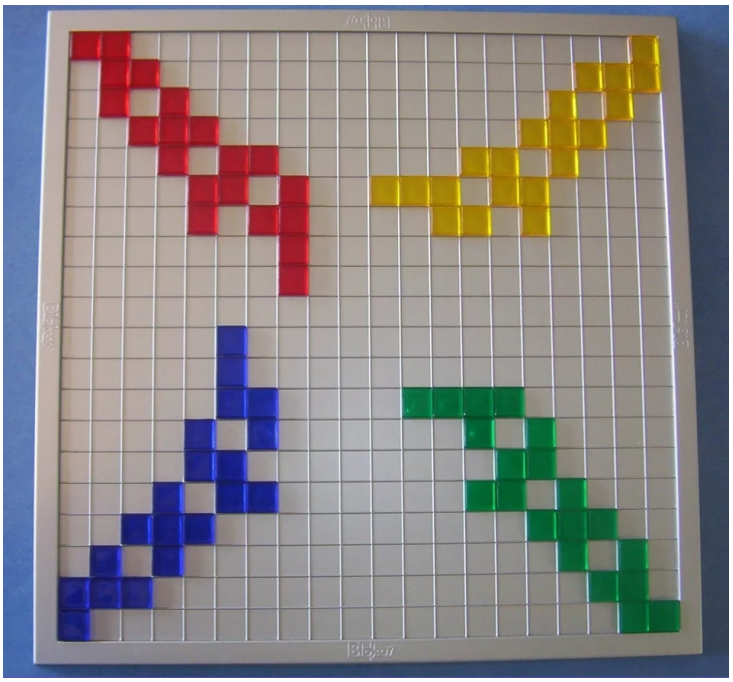


FIGURE 22 – Illustration de l'ouverture Barasona

4.4.2 AlphaBeta

L'algorithme AlphaBeta est une optimisation de l'algorithme MinMax. Il permet à l'ordinateur de ne pas tester les coups qui ne seront pas choisis à coup sûr.

Par exemple, en reprenant la figure 21, en supposant que l'algorithme parcoure l'arbre de droite à gauche. Le plateau 3 a un score de 25 donc le plateau de départ (qui prend le max) aura un score supérieur à 25. Donc après avoir évalué le plateau 2.2 de score 13, il peut être déduit que le plateau 2 (qui prend le min) est inférieur à 13. Donc dans tous les cas, le plateau de départ ne choisira pas le coup 2 (car $13<25$). Cela ne sert donc à rien d'évaluer les autres branches du plateau 2. Généralisé à tout l'arbre, cette méthode diminue considérablement le temps de calcul.

Cet algorithme étant une optimisation du MinMax, il a été facile de l'implémenter en modifiant quelques lignes du MinMax.

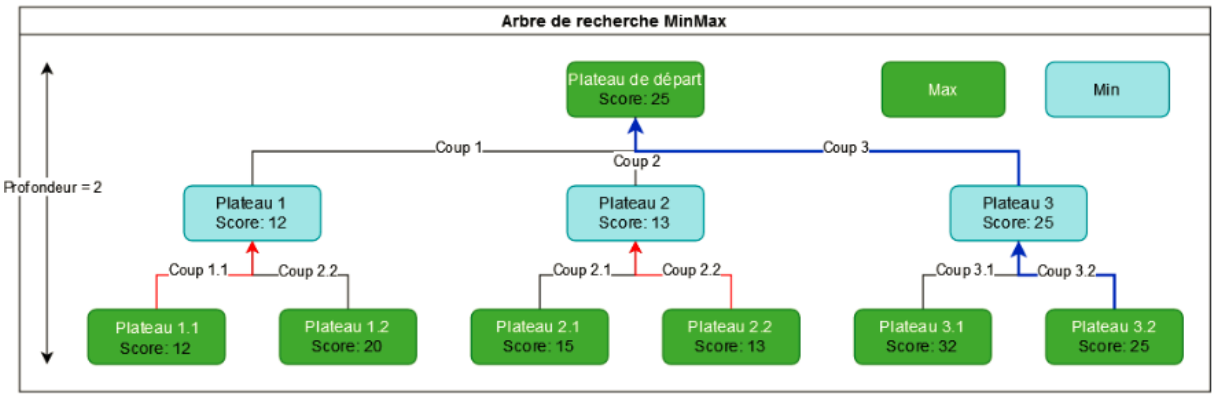


FIGURE 23 – Arbre de recherche

4.4.3 Monte-Carlo

La méthode de Monte-Carlo est une autre méthode utilisant un parcours d'arbre. Cette méthode a permis une révolution pour l'amélioration des **I.A.** pour les jeux avec une **Complexité** importante comme le jeu de GO avant l'apparition du deep learning. Le fonctionnement est simple, l'algorithme génère des coups aléatoires jusqu'à la fin de la partie. Si l'**I.A.** gagne cette partie alors la probabilité de générer les coups joués dans la partie va augmenter sinon elle diminue. En simulant ainsi de nombreuses parties, les coups avec la plus grande probabilité d'être généré indique le chemin optimal pour gagner.

Avec plus de temps, cette méthode aurait pu être implémenté. En la faisant jouer contre l'autre IA, il aurait été possible de voir si cet algorithme est plus efficient pour le jeu du Blokus.

4.5 Conclusion

L'intelligence artificielle est un point crucial dans un jeu vidéo. Il est nécessaire de veiller à ce que cette dernière soit bien implémentée et fonctionnelle au sens où elle répond au cahier des charges. Dans ce but, l'**I.A.** implémentée possède plusieurs niveaux de difficultés grâce aux différents modèles utilisés et à leurs combinaisons.

Le premier niveau de difficulté est une **I.A.** qui joue des coups aléatoires en respectant les règles. La seconde joue des coups aléatoire par ordre de taille (les pièces les plus grosses en première) et la dernière utilise la fonction de recherche avec la fonction d'évaluation. La réalisation de l'algorithme de recherche a été conçu en considérant le temps de réponse, dans le but de ne pas dépasser le délai de 5 secondes. L'**I.A.** respecte donc bien le cahier des charges défini au début du projet.

5 Conclusion

En conclusion, l'ensemble des objectifs minimaux du cahier des charges a été respecté. L'application permet effectivement de jouer au Blokus en respectant les règles, possède un menu, un tutoriel et une intelligence artificielle capable de jouer avec et contre d'autres joueurs de manière efficiente et en respectant la contrainte de temps imposée dans le cahier des charges, soit 5 secondes par coup.

Plusieurs perspectives d'amélioration peuvent être considérées. Premièrement, il aurait pu être intéressant d'ajouter la possibilité de jouer en réseau, ce qui était un objectif non nécessaire du cahier des charges. Ensuite, l'interface graphique pourrait être améliorée dans la forme. Néanmoins, certaines améliorations de forme nécessitent un remaniement de la représentation du jeu, ce qui peut facilement être chronophage. De plus, l'intelligence artificielle peut encore être optimisée de deux manières différentes : une première consiste en l'implémentation de la méthode de Monte-Carlo mentionné dans la partie optimisation, une autre consiste en l'optimisation du modèle final de la fonction d'évaluation en trouvant la combinaison de modèles qui permet d'obtenir le meilleur taux de victoire.

Une dernière perspective d'amélioration du projet serait l'étude de la différence entre l'intelligence artificielle reposant sur les fonctions d'évaluation et de recherche et une intelligence artificielle basée sur un réseau de neurones dans le cadre du jeu Blokus. En effet, les deux types d'intelligence artificielle ont chacun leurs avantages et leurs inconvénients dont l'appréhension est intéressante pour choisir le type utilisé dans d'autres projets axés sur l'intelligence artificielle.

6 Bibliographie

Références

[1] : Blokus : Règles du jeu. Disponible sur https://www.regledujeu.fr/blokus/ (consulté le 24/11/2021)

[2] : Clément Dussieux, Ismael Attoumani, Anthony Gautier, Elorri Sagardia, Imen Mhamdi, Projet S6 Blokus , Rapport ENSEIRB MatMeca Bordeaux Disponible sur https://imenmhamdi.files.wordpress.com/2014/03/rapport2.pdf (consulté le 13/10/2021)

[3] : BOUZY Bruno, Cours Intelligence artificielle https://helios2.mi.parisdescartes.fr/~bouzy/Doc/IAL3/04_IA_jeux_BB.pdf (consulté le 15/12/2021)

[4] : Learn Programming With C#, Openclassroom Disponible sur https://openclassrooms.com/fr/courses/5670356-learn-programming-with-c (consulté le 01/12/2021)

[5] : Stack Overflow disponible sur https://stackoverflow.com/ (consulté le 01/12/2021)

[6] : BUSTROS Alexandre, Un système d'intelligence artificielle pour le jeu de plateau axis et allies, mémoire, Université du Québec à Montréal, 2009. Disponible sur https://core.ac.uk/download/pdf/9539022.pdf (consulté le 12/12/2021)

[7] : Forum Board Game Geek disponible sur https://boardgamegeek.com/thread/91988/barasona-opening (consulté le 13/04/2022)

A Code C#

Cette section met en évidence une liste non exhaustive de classes du code en C# de l'application qui forme la partie la plus importante de celui-ci.

A.1 Fonction d'évaluation

```
1     public class FE
2     {
3         //MODELE NAIF
4         private double Naif(int id_joueur, List<Piece> pj1, List<Piece>
              pj2, List<Piece> pj3, List<Piece> pj4)
5         {
6             List<double> poids = new List<double>() { 2, 3, 4, 3.75,
                  4.67, 5, 4.67, 5, 5, 5.89, 6, 6, 5.89, 6, 5.56, 5.67, 6,
                  5.67, 6, 6, 5.89 };
7             double score = 0;
8             List<int [,]> pj1_piece = new List<int[,]>();
9             foreach(Piece item in pj1)
10            {
11                pj1_piece.Add(item.piece);
12            }
13            List<int[,]> pj2_piece = new List<int[,]>();
14            foreach (Piece item in pj2)
15            {
16                pj2_piece.Add(item.piece);
17            }
18            List<int[,]> pj3_piece = new List<int[,]>();
19            foreach (Piece item in pj3)
20            {
21                pj3_piece.Add(item.piece);
22            }
23            List<int[,]> pj4_piece = new List<int[,]>();
24            foreach (Piece item in pj4)
25            {
26                pj1_piece.Add(item.piece);
27            }
28            List<int[,]> pj1_eff = new List<int[,]>();
29            List<int[,]> pj2_eff = new List<int[,]>();
30            List<int[,]> pj3_eff = new List<int[,]>();
31            List<int[,]> pj4_eff = new List<int[,]>();
32            foreach(int [,] item in Pieces.pieces)
33            {
34                if (!pj1_piece.Contains(item)) {
35                    pj1_eff.Add(item);
36                }
37                if (!pj2_piece.Contains(item))
38                {
39                    pj2_eff.Add(item);
40                }
41                if (!pj3_piece.Contains(item))
42                {
43                    pj3_eff.Add(item);
44                }
45                if (!pj4_piece.Contains(item))
46                {
```

```
47                      pj4_eff.Add(item);
48                  }
49              }
50
51
52
53              for (int i = 0; i < pj1_eff.Count; i++)
54              {
55                  int forme_ind = Pieces.pieces.IndexOf(pj1_eff[i]);
56                  score += poids[forme_ind] * Math.Pow(-1, id_joueur + 1);
57
58              }
59              for (int i = 0; i < pj2_eff.Count; i++)
60              {
61                  int forme_ind = Pieces.pieces.IndexOf(pj2_eff[i]);
62                  score += poids[forme_ind] * Math.Pow(-1, id_joueur);
63              }
64              for (int i = 0; i < pj3_eff.Count; i++)
65              {
66                  int forme_ind = Pieces.pieces.IndexOf(pj3_eff[i]);
67                  score += poids[forme_ind] * Math.Pow(-1, id_joueur + 1);
68              }
69              for (int i = 0; i < pj4_eff.Count; i++)
70              {
71                  int forme_ind = Pieces.pieces.IndexOf(pj4_eff[i]);
72                  score += poids[forme_ind] * Math.Pow(-1, id_joueur);
73              }
74              return (float)score;
75          }
76
77          //MODELE MICRO
78          static int Micro(int id_joueur, int[,] plateau)
79          {
80              return (int)TousLesCoins(plateau, id_joueur).Count();
81          }
82
83          //MODELE MACRO
84
85          private double Macro(int id_joueur, int [,] plateau)
86          {
87              double score = 0;
88              List<Tuple<double, double>> bary = new List<Tuple<double,
                    double>>();
89              for (int k = 1; k < 5; k++)
90              {
91                  double bar1 = 0;
92                  double bar2 = 0;
93                  var SquareList = new List<Tuple<int, int>>();
94
95                  for (int i = 0; i < 20; i++)
96                  {
97                      for (int j = 0; j < 20; j++)
98                      {
99                          if (plateau[i, j] == k)
100                         {
101                             SquareList.Add(new Tuple<int, int>(i, j));
102                         }
```

```
103                     }
104                 }
105 
106                 foreach (Tuple<int, int> t in SquareList)
107                 {
108                     bar1 += t.Item1 / SquareList.Count;
109                     bar2 += t.Item2 / SquareList.Count;
110                 }
111                 bary.Add(new Tuple<double, double>(bar1,bar2));
112             }
113             score = Math.Min(Math.Sqrt(Math.Pow(bary[id_joueur].Item1-
                    bary[(id_joueur+1)%4].Item1,2)+Math.Pow(bary[id_joueur].
                    Item2 - bary[(id_joueur + 1) % 4].Item2,2)), Math.Sqrt(
                    Math.Pow(bary[id_joueur].Item1 - bary[(id_joueur + 3) %
                    4].Item1, 2) + Math.Pow(bary[id_joueur].Item2 - bary[(
                    id_joueur + 3) % 4].Item2, 2)) / Math.Sqrt(2 * 20^2));
114             return (float)score;
115         }
116 
117         //MODELE FINAL
118 
119         private double Eval(int id_joueur,int [,] plateau, List<Piece>
                pj1, List<Piece> pj2, List<Piece> pj3, List<Piece> pj4)
120         {
121             double score = 0;
122             double naif = Naif(id_joueur, pj1, pj2, pj3, pj4);
123             double micro = Micro(id_joueur, plateau);
124             double macro = Macro(id_joueur, plateau);
125             score = naif * micro * macro;
126             return (float)score;
127         }
128 
129         //Fonctions secondaires (IsUnCoin, TousLesCoins)
130         public static List<Tuple<int, int>> IsUnCoin(int i, int j, int
                [,] plateau)
131         {
132             int s1;
133             int s2;
134             int s3;
135             int s4;
136             var res = new List<Tuple<int, int>> { };
137             //cas limite sur les coins
138             if (i == 0 && j == 0)//coin haut gauche
139             {
140                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
141                 if (s3 == 0)/*coin bas droite*/
142                 {
143                     res.Add(Tuple.Create(i + 1, j + 1));
144                 }
145                 return res;
146             }
147 
148             if (i == 0 && j == 19)//coin haut droite
149             {
150                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
```

```
151                 if (s2 == 0)/*coin bas gauche*/
152                 {
153                     res.Add(Tuple.Create(i + 1, j - 1));
154                 }
155                 return res;
156             }
157 
158             if (i == 19 && j == 0)//coin bas gauche
159             {
160                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
161                 if (s4 == 0)/*coin haut droite*/
162                 {
163                     res.Add(Tuple.Create(i - 1, j + 1));
164                 }
165                 return res;
166             }
167 
168             if (i == 19 && j == 19)//coin bas droite
169             {
170                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
171                 if (s1 == 0)/*coin haut droite*/
172                 {
173                     res.Add(Tuple.Create(i - 1, j - 1));
174                 }
175                 return res;
176             }
177 
178             //cas limite sur les bords
179             if (i == 0)//cas limite oÃ¹ on est sur le bord en haut
180             {
181                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
182                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
183                 if (s2 == 0)/*coin bas droite*/
184                 {
185                     res.Add(Tuple.Create(i + 1, j - 1));
186                 }
187                 if (s3 == 0)/*coin bas droite*/
188                 {
189                     res.Add(Tuple.Create(i + 1, j + 1));
190                 }
191                 return res;
192             }
193 
194             if (i == 19)//cas limite oÃ¹ on est sur le bord en bas
195             {
196                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
197                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
198                 if (s1 == 0)/*coin haut droite*/
199                 {
200                     res.Add(Tuple.Create(i - 1, j - 1));
201                 }
```

```
202                 if (s4 == 0)/*coin haut droite*/
203                 {
204                     res.Add(Tuple.Create(i - 1, j + 1));
205                 }
206                 return res;
207             }
208 
209             if (j == 0)//cas limite oÃ on est sur le bord droite
210             {
211                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
212                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
213                 if (s3 == 0)/*coin bas droite*/
214                 {
215                     res.Add(Tuple.Create(i + 1, j + 1));
216                 }
217                 if (s4 == 0)/*coin haut droite*/
218                 {
219                     res.Add(Tuple.Create(i - 1, j + 1));
220                 }
221                 return res;
222             }
223 
224             if (j == 19)//cas oÃ on est sur le bord ) droite
225             {
226                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
227                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
228                 if (s1 == 0)/*coin haut droite*/
229                 {
230                     res.Add(Tuple.Create(i - 1, j - 1));
231                 }
232                 if (s2 == 0)/*coin bas droite*/
233                 {
234                     res.Add(Tuple.Create(i + 1, j - 1));
235                 }
236                 return res;
237             }
238             //hors cas limite:
239 
240             s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau[i,
                    j - 1];
241             s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau[i +
                     1, j];
242             s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau[i +
                     1, j];
243             s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau[i,
                    j + 1];
244 
245             if (s1 == 0)/*coin haut droite*/
246             {
247                 res.Add(Tuple.Create(i - 1, j - 1));
248             }
249             if (s2 == 0)/*coin bas droite*/
250             {
```

```
251                 res.Add(Tuple.Create(i + 1, j - 1));
252             }
253             if (s3 == 0)/*coin bas droite*/
254             {
255                 res.Add(Tuple.Create(i + 1, j + 1));
256             }
257             if (s4 == 0)/*coin haut droite*/
258             {
259                 res.Add(Tuple.Create(i - 1, j + 1));
260             }
261             return res;
262         }
263 
264         static List<Tuple<int, int>> TousLesCoins(int[,] P, int
                CouleurJoueur)
265         {
266             int[,] plateau = new int[20, 20];
267             for (int k = 0; k < 20; k++)
268             {
269                 for (int l = 0; l < 20; l++)
270                 {
271                     plateau[k, l] = P[k, l];
272                     if (P[k, l] != CouleurJoueur)
273                     {
274                         plateau[k, l] = 0;
275                     }
276                 }
277             }
278             var res = new List<Tuple<int, int>> { };
279             for (int i = 0; i < 20; i++)
280             {
281                 for (int j = 0; j < 20; j++)
282                 {
283                     if (plateau[i, j] == CouleurJoueur)
284                     {
285                         List<Tuple<int, int>> liste_coins = IsUnCoin(i,
                            j, plateau);
286                         if (liste_coins.Count > 0)
287                         {
288                             for (int k = 0; k < liste_coins.Count; k++)
289                             {
290                                 res.Add(liste_coins[k]);
291                             }
292                         }
293                     }
294 
295                 }
296             }
297             return res;
298         }
299 
300 
301     }
```

A.2 Classes Piece et Pieces

```
1     using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using Microsoft.Xna.Framework.Input;
4 using System;
5 using System.Collections.Generic;
6 using System.Diagnostics;
7 using System.Linq;
8 using System.Text;
9 using System.Threading.Tasks;
10
11 namespace Blokus
12 {
13     public class Piece
14     {
15         // La matrice de la piece
16         public int[,] piece;
17         // La couleur
18         public Color couleur;
19         // La taille de la piece
20         private float size;
21         // La position
22         private Vector2 position;
23         // Si la piece doit suivre la souris
24         public bool dragged;
25         // L'espacement pour un meilleur affichage
26         public int espacement;
27         // L'identifiant de la piece
28         public int ID;
29         // Le nombre de points de la piece (la somme des carres
               composant la piece)
30         public int points;
31
32         public Piece(int[,] p, Color _couleur, int _id)
33         /*
34         * Constructeur de la classe Piece
35         */
36         {
37             points = 0;
38             ID = _id;
39             espacement = 10;
40             position = new Vector2();
41             couleur = _couleur;
42             piece = p;
43             size = 0.5f;
44             dragged = false;
45
46             for (int i = 0; i < 5; i++)
47                 for (int j = 0; j < 5; j++)
48                 {
49                     points += piece[j, i];
50                 }
51         }
52
53         public void Update(GameTime gameTime, Func<Piece, int> selection
               , Func<Piece, int> deselection, Piece piece_selec)
```

```
54          /*
55          * Met Ãă jour la piece
56          */
57          {
58              // On recupere la position de la souris
59              var Mstate = Mouse.GetState();
60              Vector2 Mpos = new Vector2(Mstate.X, Mstate.Y);
61              // Si la souris est sur la piece
62              if (Mpos.X >= position.X && Mpos.X <= position.X + 5 * 11 &&
                    Mpos.Y >= position.Y && Mpos.Y <= position.Y + 5 * 11 &&
                    !dragged)
63              {
64                  // On grossit la piece
65                  size = 0.7f;
66                  // Si on clique sur la piece
67                  if (Mstate.LeftButton == ButtonState.Pressed && !Game1.
                        isClicked)
68                  {
69                      // On indique que la souris Ãă selectionnee une
                            piece
70                      Game1.isClicked = true;
71                      // On copie la piece pour la faire suivre la souris
72                      Piece np = new Piece(piece, couleur, ID);
73                      // On augmente la taille de la piece pour qu'elle
                            rentre dans le plateau
74                      np.espacement = 20;
75                      np.size = 1;
76                      // On selectionne la piece
77                      if (piece_selec != null)
78                      {
79                          deselection(piece_selec);
80                      }
81
82                      selection(np);
83                      dragged = true;
84                  }
85              }
86              // Sinon, si la piece ne doit pas suivre la souris, on la
                    redimensionne normalement
87              else
88              {
89                  if (!dragged)
90                  {
91                      size = 0.5f;
92                  }
93              }
94
95
96          }
97
98          public void Draw(SpriteBatch spriteBatch, Vector2 pos, bool gray
                = false)
99          /*
100          * Affiche la piece
101          */
102         {
103             position = pos;
```

```
104             // Si la piece doit suivre la souris, on ne l'affiche pas
                    ici
105             if (!dragged)
106                 // Pour chaque carre de la piece
107                 for (int x = 0; x < 5; x++)
108                 {
109                     for (int y = 0; y < 5; y++)
110                     {
111                         // On calcule la position du carre
112                         Vector2 p = new Vector2(position.X + x * (
                                espacement + 1), position.Y + y * (espacement
                                 + 1));
113                         // Si ce carre est dans la piece (donc vaut 1)
114                         if (piece[y, x] != 0)
115                         {
116                             // On affiche le carre
117                             if (!gray)
118                                 spriteBatch.Draw(Art.piece, p, null,
                                        couleur, 0, new Vector2(0, 0), size,
                                        0, 0);
119                             else
120                                 spriteBatch.Draw(Art.piece, p, null,
                                        couleur * 0.5f, 0, new Vector2(0, 0),
                                         size, 0, 0);
121                         }
122                     }
123                 }
124         }
125 
126         public void rotation() {
127             /*
128              * Fait une rotation de la piece
129              */
130             int[,] new_piece = new int[5, 5];
131             for (int i = 0; i < 5; i++)
132                 for (int j = 0; j < 5; j++)
133                 {
134                     new_piece[j, 4 - i] = piece[i, j];
135                 }
136             piece = new_piece;
137         }
138 
139         public void flip() {
140             /*
141              * Fait une symetrie horizontale de la piece
142              */
143             int[,] new_piece = new int[5, 5];
144             for (int i = 0; i < 5; i++)
145                 for (int j = 0; j < 5; j++)
146                 {
147                     new_piece[4 - i, j] = piece[i, j];
148                 }
149             piece = new_piece;
150         }
151     }
152 }
153 
```

```
154 using Microsoft.Xna.Framework;
155 using Microsoft.Xna.Framework.Graphics;
156 using Microsoft.Xna.Framework.Input;
157 using System;
158 using System.Collections.Generic;
159 using System.Diagnostics;
160 using System.Linq;
161 using System.Text;
162 using System.Threading.Tasks;
163
164 namespace Blokus
165 {
166     public static class Pieces
167     {
168         public static List<int[,]> pieces; // Listes des pieces
169         public static void Initialise()
170         /*
171          * Initialise les pieces
172          */
173         {
174             pieces = new List<int[,]>();
175             pieces.Add(new int[5, 5]
176             {
177                 {0,0,0,0,0},
178                 {0,0,0,0,0},
179                 {0,0,1,0,0},
180                 {0,0,0,0,0},
181                 {0,0,0,0,0}
182             });
183             pieces.Add(new int[5, 5]
184             {
185                 {0,0,0,0,0},
186                 {0,0,0,0,0},
187                 {0,0,1,1,0},
188                 {0,0,0,0,0},
189                 {0,0,0,0,0}
190             });
191             pieces.Add(new int[5, 5]
192             {
193                 {0,0,0,0,0},
194                 {0,0,0,0,0},
195                 {0,0,1,1,0},
196                 {0,0,0,1,0},
197                 {0,0,0,0,0}
198             });
199             pieces.Add(new int[5, 5]
200             {
201                 {0,0,0,0,0},
202                 {0,0,0,0,0},
203                 {0,1,1,1,0},
204                 {0,0,0,0,0},
205                 {0,0,0,0,0}
206             });
207             pieces.Add(new int[5, 5]
208             {
209                 {0,0,0,0,0},
210                 {0,0,0,0,0},
```

```
211                 {0,0,1,1,0},
212                 {0,0,1,1,0},
213                 {0,0,0,0,0}
214             });
215              pieces.Add(new int[5, 5]
216             {
217                 {0,0,0,0,0},
218                 {0,0,1,0,0},
219                 {0,1,1,1,0},
220                 {0,0,0,0,0},
221                 {0,0,0,0,0}
222             });
223              pieces.Add(new int[5, 5]
224             {
225                 {0,0,0,0,0},
226                 {0,0,0,0,0},
227                 {0,1,1,1,1},
228                 {0,0,0,0,0},
229                 {0,0,0,0,0}
230             });
231              pieces.Add(new int[5, 5]
232             {
233                 {0,0,0,0,0},
234                 {0,0,0,1,0},
235                 {0,1,1,1,0},
236                 {0,0,0,0,0},
237                 {0,0,0,0,0}
238             });
239              pieces.Add(new int[5, 5]
240             {
241                 {0,0,0,0,0},
242                 {0,0,0,0,0},
243                 {0,0,1,1,0},
244                 {0,1,1,0,0},
245                 {0,0,0,0,0}
246             });
247              pieces.Add(new int[5, 5]
248             {
249                 {0,0,0,0,0},
250                 {1,0,0,0,0},
251                 {1,1,1,1,0},
252                 {0,0,0,0,0},
253                 {0,0,0,0,0}
254             });
255              pieces.Add(new int[5, 5]
256             {
257                 {0,0,0,0,0},
258                 {0,0,0,0,0},
259                 {0,0,1,0,0},
260                 {0,0,1,0,0},
261                 {0,1,1,1,0}
262             });
263              pieces.Add(new int[5, 5]
264             {
265                 {0,0,0,0,0},
266                 {0,0,0,0,0},
267                 {0,0,1,0,0},
```

```
268                 {0,0,1,0,0},
269                 {0,0,1,1,1}
270             });
271              pieces.Add(new int[5, 5]
272             {
273                 {0,0,0,0,0},
274                 {0,0,0,0,0},
275                 {0,1,1,1,0},
276                 {1,1,0,0,0},
277                 {0,0,0,0,0}
278             });
279              pieces.Add(new int[5, 5]
280             {
281                 {0,0,0,0,0},
282                 {0,0,0,1,0},
283                 {0,1,1,1,0},
284                 {0,1,0,0,0},
285                 {0,0,0,0,0}
286             });
287              pieces.Add(new int[5, 5]
288             {
289                 {0,0,1,0,0},
290                 {0,0,1,0,0},
291                 {0,0,1,0,0},
292                 {0,0,1,0,0},
293                 {0,0,1,0,0}
294             });
295              pieces.Add(new int[5, 5]
296         {
297                 {0,0,0,0,0},
298                 {0,1,0,0,0},
299                 {0,1,1,0,0},
300                 {0,1,1,0,0},
301                 {0,0,0,0,0}
302         });
303              pieces.Add(new int[5, 5]
304         {
305                 {0,0,0,0,0},
306                 {0,0,0,0,0},
307                 {0,0,1,1,0},
308                 {0,1,1,0,0},
309                 {0,1,0,0,0}
310         });
311              pieces.Add(new int[5, 5]
312         {
313                 {0,0,0,0,0},
314                 {0,1,1,0,0},
315                 {0,1,0,0,0},
316                 {0,1,1,0,0},
317                 {0,0,0,0,0}
318         });
319              pieces.Add(new int[5, 5]
320         {
321                 {0,0,0,0,0},
322                 {0,0,1,1,0},
323                 {0,1,1,0,0},
324                 {0,0,1,0,0},
```

```
325                     {0,0,0,0,0}
326         });
327                 pieces.Add(new int[5, 5]
328         {
329                     {0,0,0,0,0},
330                     {0,0,1,0,0},
331                     {0,1,1,1,0},
332                     {0,0,1,0,0},
333                     {0,0,0,0,0}
334         });
335                 pieces.Add(new int[5, 5]
336         {
337                     {0,0,0,0,0},
338                     {0,0,0,0,0},
339                     {0,0,0,0,0},
340                     {0,0,1,0,0},
341                     {0,1,1,1,1}
342         });
343          }
344     }
345 }
```

A.3 Autres classes

```
1
2  using Microsoft.Xna.Framework;
3  using Microsoft.Xna.Framework.Graphics;
4  using Microsoft.Xna.Framework.Input;
5  using System;
6  using System.Collections.Generic;
7  using System.Diagnostics;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11
12 namespace Blokus
13 {
14     public class Coup
15     {
16         // La matrice de la piece
17         public Piece piece;
18         // La position
19         public Vector2 position;
20         // Si le coup est un coup Ãă jouer
21         public bool estNul;
22
23         public Coup(Piece _piece, Vector2 _position, bool _estNul =
               false)
24         /*
25          * Constructeur de la classe Coup
26          */
27         {
28             piece = _piece;
29             position = _position;
30             estNul = _estNul;
```

```
31          }
32      }
33  }
34  using System;
35  using System.Collections.Generic;
36  using System.Linq;
37
38  namespace Blokus
39  {
40      public static class Coin
41      {
42          public static void Main2(string[] args)
43          {
44              int[,] P = new int[20, 20]
45              {
46                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
47                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
48                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
49                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
50                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
51                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
52                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
53                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
54                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
55                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
56                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
57                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
58                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
59                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
60                  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
61                  {2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
62                  {0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
63                  {0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
64                  {0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
65                  {1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
66              };
67              //Liste des coins du joueurs 1 :
68
69
70              List<Tuple<int, int>> list_j1 = TousLesCoins(P, 1);
71              Console.WriteLine("le joueur 1 a");
72              Console.WriteLine(list_j1.Count);
73              for (int i = 0; i < list_j1.Count; i++)
74              {
75                  Console.WriteLine(list_j1[i]);
76              }
77              //liste des coins du joueurs 2 :
78              List<Tuple<int, int>> list_j2 = TousLesCoins(P, 2);
79              Console.WriteLine("le joueur 2 a");
80              Console.WriteLine(list_j2.Count);
81              for (int i = 0; i < list_j2.Count; i++)
82              {
83                  Console.WriteLine(list_j2[i]);
84              }
85
86              if (list_j1.Count > list_j2.Count)
87              {
```

```
88                  Console.WriteLine("le joueur 1 a le plus de coins");
89              }
90              else
91              {
92                  Console.WriteLine("le joueur 2 a le plus de coins");
93              }
94
95          }
96          public static List<Tuple<int, int>> IsUnCoin(int i, int j, int
                [,] plateau)
97          {
98              int s1;
99              int s2;
100             int s3;
101             int s4;
102             var res = new List<Tuple<int, int>> { };
103             //cas limite sur les coins
104             if (i == 0 && j == 0)//coin en haut Ãă gauche
105             {
106                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
107                 if (s3 == 0)/*coin en bas Ãă droite*/
108                 {
109                     res.Add(Tuple.Create(i + 1, j + 1));
110                 }
111                 return res;
112             }
113
114             if (i == 0 && j == 19)//coin en haut Ãă droite
115             {
116                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
117                 if (s2 == 0)/*coin en bas Ãă gauche*/
118                 {
119                     res.Add(Tuple.Create(i + 1, j - 1));
120                 }
121                 return res;
122             }
123
124             if (i == 19 && j == 0)//coin en bas Ãă gauche
125             {
126                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
127                 if (s4 == 0)/*coin en haut Ãă droite*/
128                 {
129                     res.Add(Tuple.Create(i - 1, j + 1));
130                 }
131                 return res;
132             }
133
134             if (i == 19 && j == 19)//coin en bas Ãă droite
135             {
136                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
137                 if (s1 == 0)/*coin en haut Ãă gauche*/
138                 {
139                     res.Add(Tuple.Create(i - 1, j - 1));
```

```
140                 }
141                 return res;
142             }
143 
144             //cas limite sur les bords
145             if (i == 0)//cas limite oÃź on est sur le bord en haut
146             {
147                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
148                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
149                 if (s2 == 0)/*coin en bas Ãă gauche*/
150                 {
151                     res.Add(Tuple.Create(i + 1, j - 1));
152                 }
153                 if (s3 == 0)/*coin en bas Ãă droite*/
154                 {
155                     res.Add(Tuple.Create(i + 1, j + 1));
156                 }
157                 return res;
158             }
159 
160             if (i == 19)//cas limite oÃź on est sur le bord en bas
161             {
162                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
163                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
164                 if (s1 == 0)/*coin en haut Ãă gauche*/
165                 {
166                     res.Add(Tuple.Create(i - 1, j - 1));
167                 }
168                 if (s4 == 0)/*coin en haut Ãă droite*/
169                 {
170                     res.Add(Tuple.Create(i - 1, j + 1));
171                 }
172                 return res;
173             }
174 
175             if (j == 0)//cas limite oÃź on est sur le bord Ãă gauche
176             {
177                 s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau
                        [i + 1, j];
178                 s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau
                        [i, j + 1];
179                 if (s3 == 0)/*coin en bas Ãă droite*/
180                 {
181                     res.Add(Tuple.Create(i + 1, j + 1));
182                 }
183                 if (s4 == 0)/*coin en haut Ãă droite*/
184                 {
185                     res.Add(Tuple.Create(i - 1, j + 1));
186                 }
187                 return res;
188             }
189 
190             if (j == 19)//cas oÃź on est sur le bord ) droite
```

```
191             {
192                 s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau
                        [i, j - 1];
193                 s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau
                        [i + 1, j];
194                 if (s1 == 0)/*coin en haut Ãă gauche*/
195                 {
196                     res.Add(Tuple.Create(i - 1, j - 1));
197                 }
198                 if (s2 == 0)/*coin en bas Ãă gauche*/
199                 {
200                     res.Add(Tuple.Create(i + 1, j - 1));
201                 }
202                 return res;
203             }
204             //hors cas limite:
205 
206             s1 = plateau[i - 1, j - 1] + plateau[i - 1, j] + plateau[i,
                    j - 1];
207             s2 = plateau[i, j - 1] + plateau[i + 1, j - 1] + plateau[i +
                     1, j];
208             s3 = plateau[i, j + 1] + plateau[i + 1, j + 1] + plateau[i +
                     1, j];
209             s4 = plateau[i - 1, j] + plateau[i - 1, j + 1] + plateau[i,
                    j + 1];
210 
211             if (s1 == 0)/*coin en haut Ãă gauche*/
212             {
213                 res.Add(Tuple.Create(i - 1, j - 1));
214             }
215             if (s2 == 0)/*coin en bas Ãă gauche*/
216             {
217                 res.Add(Tuple.Create(i + 1, j - 1));
218             }
219             if (s3 == 0)/*coin en bas Ãă droite*/
220             {
221                 res.Add(Tuple.Create(i + 1, j + 1));
222             }
223             if (s4 == 0)/*coin en haut Ãă droite*/
224             {
225                 res.Add(Tuple.Create(i - 1, j + 1));
226             }
227             return res;
228         }
229 
230         public static List<Tuple<int, int>> TousLesCoins(int[,] P, int
                CouleurJoueur)
231         {
232             int[,] plateau = new int[20, 20];
233             for (int k = 0; k < 20; k++)
234             {
235                 for (int l = 0; l < 20; l++)
236                 {
237                     plateau[k, l] = P[k, l];
238                     if (P[k, l] != CouleurJoueur)
239                     {
240                         plateau[k, l] = 0;
```

```
241                     }
242                 }
243             }
244             var res = new List<Tuple<int, int>> { };
245             for (int i = 0; i < 20; i++)
246             {
247                 for (int j = 0; j < 20; j++)
248                 {
249                     if (plateau[i, j] == CouleurJoueur)
250                     {
251                         List<Tuple<int, int>> liste_coins = IsUnCoin(i,
                            j, plateau);
252                         if (liste_coins.Count > 0)
253                         {
254                             for (int k = 0; k < liste_coins.Count; k++)
255                             {
256                                 res.Add(liste_coins[k]);
257                             }
258                         }
259                     }
260 
261                 }
262             }
263             return res;
264         }
265     }
266 }
267 
268 using Microsoft.Xna.Framework;
269 using Microsoft.Xna.Framework.Graphics;
270 using Microsoft.Xna.Framework.Input;
271 using System.Collections.Generic;
272 using System;
273 
274 namespace Blokus
275 {
276     public static class GFG
277     {
278         public static int rotation_joueur(int j)
279         {
280             int rj = j;
281             if (j == 2)
282             {
283                 rj = 4;
284             }
285             else if (j == 4)
286             {
287                 rj = 3;
288             }
289             else if (j == 3)
290             {
291                 rj = 1;
292             }
293             else
294             {
295                 rj += 1;
296             }
```

```
297             if (j == 5)
298             {
299                 rj = 1;
300             }
301             return rj;
302         }
303 
304         public static Tuple<double, Coup> Minmax(int depth, Plateau
                board, bool isMax, int h, Coup meilleurcoupsauvegarde, int
                joueur_actuel, int joueurPermanent)
305         {
306             if (depth == 0)
307             {
308                 //return new Tuple<double, Coup>(FE.Naif(joueur_actuel,
                        board.pieces_j1, board.pieces_j2, board.pieces_j3,
                        board.pieces_j4), meilleurcoupsauvegarde);
309                 int eval = board.coupPossible(joueur_actuel, board.
                        plateau).Count;
310                 Console.WriteLine(eval);
311                 return new Tuple<double, Coup>(eval,
                        meilleurcoupsauvegarde);
312             }
313             if (isMax)
314             {
315                 double best = -1000000;
316                 Coup meilleurcoup = new Coup(new Piece(Pieces.pieces[0],
                        Color.White, 0), Vector2.Zero, true);
317                 List<Coup> coups = board.coupPossible(joueur_actuel,
                        board.plateau);
318                 foreach (Coup coup in coups)
319                 {
320                     board.jouerCoup(coup);
321                     Tuple<double, Coup> temp = Minmax(depth - 1, board,
                            false, h, meilleurcoup, rotation_joueur(
                            joueur_actuel), joueurPermanent);
322                     board.annulerCoup(joueur_actuel, coup);
323                     if (temp.Item1 > best)
324                     {
325                         best = temp.Item1;
326                         meilleurcoup = new Coup(board.copyPiece(coup.
                                piece), coup.position);
327                     }
328                 }
329                 return new Tuple<double, Coup>(best, meilleurcoup);
330             }
331             else
332             {
333                 double best = 1000000;
334                 Coup meilleurcoup = new Coup(new Piece(Pieces.pieces[0],
                        Color.White, 0), Vector2.Zero, true);
335                 List<Coup> coups = board.coupPossible(joueur_actuel,
                        board.plateau);
336                 foreach (Coup coup in coups)
337                 {
338                     board.jouerCoup(coup);
339                     Tuple<double, Coup> temp = Minmax(depth - 1, board,
                            true, h, meilleurcoup, rotation_joueur(
```

```
                        joueur_actuel), joueurPermanent);
340                     board.annulerCoup(joueur_actuel, coup);
341                     if (temp.Item1 < best)
342                     {
343                         best = temp.Item1;
344                         meilleurcoup = new Coup(board.copyPiece(coup.
                            piece), coup.position);
345                     }
346                 }
347                 return new Tuple<double, Coup>(best, meilleurcoup);
348             }
349         }
350     }
351 }
352
353
354 using Microsoft.Xna.Framework;
355 using Microsoft.Xna.Framework.Graphics;
356 using Microsoft.Xna.Framework.Input;
357 using System;
358 using System.Collections.Generic;
359 using System.Diagnostics;
360 using System.Linq;
361 using System.Text;
362 using System.Threading.Tasks;
363
364 namespace Blokus
365 {
366     public class Button
367     {
368         public string effet;
369         public Vector2 position;
370         public Texture2D image;
371         public Texture2D imageTXT;
372         public Vector2 origin;
373         public float size;
374         public float sizeBase;
375         public float sizeAnim;
376         public bool isFlipped;
377         public bool isSoundPlayed;
378         public Text texte;
379         public int joueur;
380         public Button(Vector2 pos, Texture2D img, Texture2D imgTXT,
                string effect, float sz, float szAnim, string txt = "", float
                 sizetxt = 1, int _joueur = 0)
381         {
382             joueur = _joueur;
383             texte = new Text(sizetxt, pos + new Vector2(0, 5), txt,
                    Color.Black, Color.Transparent, 25000);
384             isSoundPlayed = false;
385             position = pos;
386             effet = effect;
387             image = img;
388             imageTXT = imgTXT;
389             size = sz;
390             sizeBase = sz;
391             sizeAnim = szAnim;
```

```
392             origin = new Vector2(image.Bounds.Center.X, image.Bounds.
                    Center.Y);
393             isFlipped = false;
394 
395         }
396         private void ApplyEffect()
397         {
398             int player = 0;
399             switch (effet)
400             {
401                 case "suivant":
402                     Game1.Instance.avancementTuto += 1;
403                     if (Game1.Instance.avancementTuto >= Game1.Instance.
                            imgTuto.Count)
404                     {
405                         Game1.Instance.isTutorial = false;
406                         Game1.Instance.avancementTuto = 0;
407                     }
408                     break;
409                 case "precedent":
410                     Game1.Instance.avancementTuto -= 1;
411                     if (Game1.Instance.avancementTuto < 0)
412                     {
413                         Game1.Instance.avancementTuto = Game1.Instance.
                                imgTuto.Count - 1;
414                     }
415                     break;
416                 case "quitselection":
417                     Game1.gameState = GameState.menu;
418                     break;
419                 case "tuto":
420                     Game1.Instance.isTutorial = true;
421                     break;
422                 case "quit":
423                     if (Game1.Instance.isTutorial)
424                     {
425                         Game1.Instance.isTutorial = false;
426                         Game1.Instance.avancementTuto = 0;
427                     }
428                     else
429                         Game1.Instance.quitGame();
430                     break;
431                 case "play":
432                     Game1.Instance.resetSelection();
433                     Game1.gameState = GameState.selection;
434                     break;
435                 case "playSelection":
436                     Game1.gameState = GameState.play;
437                     break;
438                 case "skip":
439                     Game1.Instance.skip_turn();
440                     break;
441                 case "rejouer":
442                     Game1.Instance.ResetGame();
443                     Game1.gameState = GameState.menu;
444                     break;
445                 case "J1":
```

```
446                     player = 0;
447                     if (texte.text == "joueur")
448                     {
449                         texte.text = "IA facile";
450                         Game1.Instance.plateau.WhatAI[player] = 0;
451                         Game1.Instance.plateau.isAI[player] = true;
452                     }
453                     else if (texte.text == "IA facile")
454                     {
455                         texte.text = "IA intermediaire";
456                         Game1.Instance.plateau.WhatAI[player] = 1;
457                         Game1.Instance.plateau.isAI[player] = true;
458                     }
459                     else if (texte.text == "IA intermediaire")
460                     {
461                         texte.text = "IA difficile";
462                         Game1.Instance.plateau.WhatAI[player] = 2;
463                         Game1.Instance.plateau.isAI[player] = true;
464                     }
465                     else
466                     {
467                         texte.text = "joueur";
468                         Game1.Instance.plateau.isAI[player] = false;
469                     }
470                     break;
471                 case "J2":
472                     player = 1;
473                     if (texte.text == "joueur")
474                     {
475                         texte.text = "IA facile";
476                         Game1.Instance.plateau.WhatAI[player] = 0;
477                         Game1.Instance.plateau.isAI[player] = true;
478                     }
479                     else if (texte.text == "IA facile")
480                     {
481                         texte.text = "IA intermediaire";
482                         Game1.Instance.plateau.WhatAI[player] = 1;
483                         Game1.Instance.plateau.isAI[player] = true;
484                     }
485                     else if (texte.text == "IA intermediaire")
486                     {
487                         texte.text = "IA difficile";
488                         Game1.Instance.plateau.WhatAI[player] = 2;
489                         Game1.Instance.plateau.isAI[player] = true;
490                     }
491                     else
492                     {
493                         texte.text = "joueur";
494                         Game1.Instance.plateau.isAI[player] = false;
495                     }
496                     break;
497                 case "J3":
498                     player = 3;
499                     if (texte.text == "joueur")
500                     {
501                         texte.text = "IA facile";
502                         Game1.Instance.plateau.WhatAI[player] = 0;
```

```
503                         Game1.Instance.plateau.isAI[player] = true;
504                     }
505                     else if (texte.text == "IA facile")
506                     {
507                         texte.text = "IA intermediaire";
508                         Game1.Instance.plateau.WhatAI[player] = 1;
509                         Game1.Instance.plateau.isAI[player] = true;
510                     }
511                     else if (texte.text == "IA intermediaire")
512                     {
513                         texte.text = "IA difficile";
514                         Game1.Instance.plateau.WhatAI[player] = 2;
515                         Game1.Instance.plateau.isAI[player] = true;
516                     }
517                     else
518                     {
519                         texte.text = "joueur";
520                         Game1.Instance.plateau.isAI[player] = false;
521                     }
522                     break;
523                 case "J4":
524                     player = 2;
525                     if (texte.text == "joueur")
526                     {
527                         texte.text = "IA facile";
528                         Game1.Instance.plateau.WhatAI[player] = 0;
529                         Game1.Instance.plateau.isAI[player] = true;
530                     }
531                     else if (texte.text == "IA facile")
532                     {
533                         texte.text = "IA intermediaire";
534                         Game1.Instance.plateau.WhatAI[player] = 1;
535                         Game1.Instance.plateau.isAI[player] = true;
536                     }
537                     else if (texte.text == "IA intermediaire")
538                     {
539                         texte.text = "IA difficile";
540                         Game1.Instance.plateau.WhatAI[player] = 2;
541                         Game1.Instance.plateau.isAI[player] = true;
542                     }
543                     else
544                     {
545                         texte.text = "joueur";
546                         Game1.Instance.plateau.isAI[player] = false;
547                     }
548                     break;
549             }
550         }
551 
552         public void Update(GameTime gameTime, bool TrueScreenSize =
                false)
553         {
554             var Mstate = Mouse.GetState();
555             Vector2 mousePosition = new Vector2(Mstate.X, Mstate.Y);
556             if (!TrueScreenSize)
557                 if (MathOperation.Collision(position /*+ (Game1.
                        TrueScreenSize - Game1.ScreenSize) / 2*/,
```

```
                    mousePosition, origin.X, origin.Y))
558                 {
559                     if (!isSoundPlayed)
560                     {
561                         isSoundPlayed = true;
562                     }
563                     size += 0.05f;
564                     if (Mstate.LeftButton == ButtonState.Pressed &&
                            Game1.isClicked == false)
565                     {
566                         ApplyEffect();
567                         Game1.isClicked = true;
568                     }
569                 }
570                 else
571                 {
572                     size -= 0.05f;
573                     isSoundPlayed = false;
574                 }
575             else if (MathOperation.Collision(position, mousePosition,
                    origin.X, origin.Y))
576             {
577                 if (!isSoundPlayed)
578                 {
579                     isSoundPlayed = true;
580                 }
581                 size += 0.05f;
582                 if (Mstate.LeftButton == ButtonState.Pressed && Game1.
                        isClicked == false)
583                 {
584                     ApplyEffect();
585                     Game1.isClicked = true;
586                 }
587             }
588             else
589             {
590                 size -= 0.05f;
591                 isSoundPlayed = false;
592             }
593             size = MathHelper.Clamp(size, sizeBase, sizeAnim);
594         }
595
596         public void Draw(SpriteBatch spriteBatch)
597         {
598             Color couleur = Color.Black;
599             switch (joueur)
600             {
601                 case 1:
602                     couleur = Color.Green;
603                     break;
604                 case 2:
605                     couleur = Color.Blue;
606                     break;
607                 case 3:
608                     couleur = Color.Red;
609                     break;
610                 case 4:
```

```
611                     couleur = Color.Yellow;
612                     break;
613             }
614             if (isFlipped)
615             {
616                 spriteBatch.Draw(image, position, null, Color.White, (
                        float)Math.PI, origin, size, 0, 0);
617             }
618             else
619             {
620                 spriteBatch.Draw(image, position, null, Color.White, 0,
                        origin, size, 0, 0);
621             }
622             if (imageTXT != null)
623             {
624                 spriteBatch.Draw(imageTXT, position, null, Color.White,
                        0, new Vector2(imageTXT.Bounds.Center.X, imageTXT.
                        Bounds.Center.Y), sizeBase, 0, 0);
625             }
626             else
627             {
628                 texte.couleurInterieur = couleur;
629                 texte.Draw(spriteBatch);
630             }
631         }
632     }
633 }
634
635 using System;
636 using System.Collections.Generic;
637 using System.Linq;
638 using System.Text;
639 using System.Threading.Tasks;
640 using Microsoft.Xna.Framework.Content;
641 using Microsoft.Xna.Framework.Graphics;
642
643 namespace Blokus
644 {
645     public static class Art
646     {
647         public static Texture2D logo { get; private set; }
648         public static Texture2D backGround { get; private set; }
649         public static Texture2D boutonPlay { get; private set; }
650         public static Texture2D piece { get; private set; }
651         public static Texture2D bouton_abandonner { get; private set; }
652         public static Texture2D bouton { get; private set; }
653         public static Texture2D piece2 { get; private set; }
654         public static Texture2D plateau { get; private set; }
655
656         public static Texture2D playButton { get; private set; }
657         public static Texture2D playtexte { get; private set; }
658         public static Texture2D quitTexte { get; private set; }
659         public static Texture2D tutoTexte { get; private set; }
660         public static Texture2D leaveTexte { get; private set; }
661         public static Texture2D imgTuto { get; private set; }
662         public static Texture2D tuto1 { get; private set; }
663         public static Texture2D tuto2 { get; private set; }
```

```
664         public static Texture2D tuto3 { get; private set; }
665         public static Texture2D tuto4 { get; private set; }
666         public static Texture2D tuto5 { get; private set; }
667         public static void Load(ContentManager Content)
668         {
669             tuto5 = Content.Load<Texture2D>("Graph/tuto5");
670             tuto1 = Content.Load<Texture2D>("Graph/1tuto");
671             tuto2 = Content.Load<Texture2D>("Graph/2tuto");
672             tuto3 = Content.Load<Texture2D>("Graph/3tuto");
673             tuto4 = Content.Load<Texture2D>("Graph/4tuto");
674             imgTuto = Content.Load<Texture2D>("Graph/imgTuto");
675             leaveTexte = Content.Load<Texture2D>("Graph/leaveTexte");
676             quitTexte = Content.Load<Texture2D>("Graph/quitTexte");
677             tutoTexte = Content.Load<Texture2D>("Graph/tutoTexte");
678             plateau = Content.Load<Texture2D>("Graph/plateau");
679             playtexte = Content.Load<Texture2D>("Graph/playtexte");
680             playButton = Content.Load<Texture2D>("Graph/playButton");
681             bouton = Content.Load<Texture2D>("Graph/bouton");
682             bouton_abandonner = Content.Load<Texture2D>("Graph/
                    bouton_abandonner");
683             piece2 = Content.Load<Texture2D>("Graph/piece2");
684             piece = Content.Load<Texture2D>("Graph/piece");
685             boutonPlay = Content.Load<Texture2D>("Graph/boutonPlay");
686             logo = Content.Load<Texture2D>("Graph/Blokus");
687             backGround = Content.Load<Texture2D>("Graph/backGround");
688         }
689     }
690 }
```

B Check-list de rapport de Projet d'études

Contenu	Vérification présence	Vérification qualité
Résumé en français		
Résumé en anglais		
Table des matières		
Table des figures		
Introduction		
Conclusion générale		
Bibliographie		
Citation des références dans le texte		

Forme	Vérification présence	Vérification qualité
Vérification orthographe		
Pagination		
Homogénéité de la mise en page		
Lisibilité des figures		