République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 20 Août 1955 - Skikda

Faculté des Sciences

Département d'Informatique

Mémoire

En vue de l'obtention du diplôme de Master

Filière : Informatique

Spécialité : Systèmes D'information Avancés et Applications

Thème

Développement d'un jeu de Dames en utilisant une Technique D'intelligence Artificielle

Présenté par :

***Boulkenafet Cheima**

***Lokchiri Esma**

Devant le jury composé de :

Président : Mosbah

Rapporteur : Lahsasna Adel

Examinateur : Cheribat

Remerciements

Nous remercions Dieu au début pour notre réconciliation cette année et pour avoir terminé ce rapport.

Nous sommes très honoré que Monsieur : Mosbah de l'Université 20 Août 1955 - Skikda ait accepté de présider notre jury.

Nous tenons à remercier Messieurs : Chribat pour l'intérêt qu'ils ont porté à ce travail en acceptant d'en être les examinateurs.

Nous exprimons toute notre gratitude à Monsieur A. LAHSASNA de l'Université 20 Août 1955 - Skikda d'avoir dirigé et assurer avec compétence l'encadrement de cette étude.

Notre sincères remerciements vont particulièrement à de l'Etablissement de (Université de SKIKDA) pour les nombreuses discussions scientifiques que nous avons eues ensemble ainsi que les conseils valeureux qui ont permis d'avoir un travail organisé.

Nous adressons notre profonde reconnaissance à tous les enseignants et le staff administratif du département d'Informatique de l'Université 20 Août 1955 - Skikda.

Dédicace

Louange Dieu, qui nous avoir donnée la santé et la volonté d'entamer et de terminer ce mémoire et nous a permis de valoriser cette étape de notre parcours universitaire .

Dédié à la mémoire bienveillante de mon père bien-aimé, [AHMED] , dont l'amour, le soutien et les encouragements continuent de m'inspirer chaque jour. Tes valeurs, ta sagesse et ton héritage resteront à jamais gravés dans mon cœur .

رحمك الله

À ma chère mère, [FARIDA], ma plus grande source d'inspiration et ma confidente inébranlable. Ta force, ta détermination et ton amour inconditionnel m'ont poussé à atteindre de nouveaux sommets. Tu as été mon roc pendant ce parcours académique, et je suis profondément reconnaissant(e) de t'avoir à mes côtés Vos encouragements incessants ont été ma boussole tout au long de cette aventure académique..

Que cette humble réalisation soit un témoignage de mon amour éternel pour vous deux.

Je souhaite également exprimer ma gratitude envers ma famille élargie

mes frères : Ayoub, Oussama, Mouad, Mohamed Amine

et sœurs : Kaltoum, Rabab, Assila , Selma

Au plus jeune de la famille : Abd Elmodjib, Nour Elyakine, Sadjida, Abd Elmohaimin, Ahmed, Maria

et À mes amis: Iman, Esma, Amel, Rokia, Amel

qui m'ont apporté leur soutien inconditionnel tout au long de cette aventure académique.

Enfin, j'adresse mes remerciements à mes professeurs et mentors, qui ont partagé leur savoir, m'ont inspiré et ont contribué à mon développement personnel et professionnel.

CHEIMA BOULKENAFET

Dédicace

En premier lieu et avant tous, je prie ALLAH de m'avoir donné la volonté et le courage d'achever mes études.
C'est avec beaucoup de respect et tant d'amour je dédie ce travail à tous ceux qui ont participé à son accomplissement.
À ma mère Ratiba , la lumière de mes jours, la source de mes efforts, la flamme de mon cœur, ma vie et mon Bonheur , A vous cher maman , maman que j'adore qui m'a offert plus que le nécessaire pour que je puisse mener mes études dans les meilleures conditions possible.
A mon père Azzouz, merci infiniment
A ceux que j'aime beaucoup et qui mes sources d'énergie , m'a soutenue tout au long de ce projet :
Mes chères soeures : Anfel, Yasmine.
mes chères frères :Yassine, Oussama, Amjed.
A mes belles :Sabrina,Amel,Chaima, Rokia.
Un merci spécial à mon mari Hamza, mon compagnon qui m'a soutenu tout la période de mes études.
A ceux qui m'ont soutenu, m'ont encouragé durant toute ma période d'étude, et pour leurs consentis

ASMA LOKCHIRI

TABLE DES MATIÈRES

Introduction générale

Chapitre 1 : Introduction à la théorie des jeux

Chapitre 2 : Algorithmes évolutionnaire : Principes et méthodes

Chapitre 3 : Analyse et Conception

Chapitre 4 : Implémentation

Conclusion générale

Références

LISTE DES FIGURES

Chapitre 1 : Introduction à la théorie des jeux

Chapitre 2 : Algorithmes évolutionnaire : Principes et méthodes

Chapitre 3 : Analyse et Conception

Chapitre 4 : Implémentation

LISTE DES ALGORITHMES

INTRODUCTION GÉNÉRALE

1. Contexte général :

Qui parmi nous n'a jamais participé aux jeux d'échecs, jeux de dames, jeux de cartes ou tout simplement ne s'est jamais trouvé dans une situation de concurrence. Mais personne n'a jamais imaginé que ce moment de distraction ou cette situation de rivalité pouvait être présentée sous forme d'un modèle mathématique. Cela se fait par les concepts de la *théorie des jeux*.

La *théorie des jeux* est une discipline mathématique qui étudie les situations où l'état final des individus (joueurs) dépond non seulement des décisions qu'il prend mais également des décisions prises par l'adversaire. Ainsi, le choix *optimal* pour un individu dépend généralement de ce que fait son adversaire. En effet, la théorie des jeux a développé une vocation pour les sciences sociales et économiques, avec des applications disparates. Elle apparait aujourd'hui comme un paradigme très général de concepts et de techniques, dont le potentiel reste encore à exploiter en informatique. En fait, les *jeux* peuvent être chose sérieuse. Ils sont un moyen d'évaluer l'intelligence des ordinateurs par rapport à la nôtre.

2. Problématique :

Les problèmes d'*optimisations* occupent actuellement une place de choix dans la communauté scientifique. Non pas qu'ils aient été un jour considérés comme secondaires mais l'évolution des techniques informatiques a permis de dynamiser les recherches dans ce domaine. **[1]**

Le comportement des stratégies alternatives dans les jeux est définit par le tracé du stimulus-réponse de chaque individu. La limitation de ces comportements aux fonctions linéaires des conditions environnementales rend les résultats douteux. En effet, la prise des décisions efficaces dans n'importe quel environnement complexe exige presque toujours un tracé non linéaire du stimulus-réponse.

Dans le cadre de notre mémoire, l'obstacle vient dans le choix de la représentation des stratégies de jeu appropriées et de l'algorithme du recherche utiliser ainsi que sa fonction d'évaluation afin de remédier aux problèmes d'*explosion combinatoire* qui reste un défait pour les gents qui s'intéressent de la théorie des jeux combinatoire.

3. Approche adoptée :

On dirait une fusion de plusieurs méthodes afin d'exploiter les avantages de chacune, semble constituer une approche intéressante pour surmonter ces difficultés. La combinaison la plus adéquate dans notre cas est Combiner l'algorithme minimax avec un algorithme évolutionnaires (EA). La principale raison est que ces deux méthodes présentent des avantages complémentaires, elles fournissent des outils pour la résolution de problème de sélection de l'action dans les jeux.

Les EA ont aussi montré leur capacité à éviter la convergence des solutions vers des optimums locaux, aussi bien lorsqu'ils sont combinés avec des méthodes de recherche locale comme la rétro-propagation du gradient que lorsqu'ils sont seuls. [2]

Les algorithmes minimax ont aussi montré pour évaluer les états du jeu et effectuer des coups optimaux. L'algorithme minimax explore l'arbre de jeu en analysant de manière récursive les coups possibles et leurs conséquences. Il attribue une valeur à chaque état du jeu, représentant le degré de désirabilité pour le joueur actuel. Cette valeur est utilisée pour déterminer le meilleur coup.

L'arrivé des stratégies d'évolution permet d'utiliser l'ordinateur comme un outil pour découvrir des stratégies de jeu où l'heuristique peut être indisponible. De même, l'utilité de mettre en application de Minimax comme fonction d'évaluation non linéaire pour produire des stratégies dans les jeux complexes devient évidente. Alors, la combinaison des stratégies d'évolution avec les algorithme Minimax semble bien adaptée à découvrir des stratégies optimales dans les jeux où la théorie des jeux classique est incapable de fournir des décisions efficaces.

Plan du mémoire :

Le travail présenté dans ce mémoire a été organisé en quatre chapitres :

Le *premier chapitre* est consacré à l'étude de la théorie des jeux en indiquant principalement les différents types de jeu ainsi que les méthodes de recherches dans les arbres de jeu (Minimax, Alpha-Beta, Fail soft Alpha-beta, …).

Dans le *second chapitre*, nous abordons les concepts de base des algorithmes évolutionnaires en se concentrant sur les études des stratégies d'évolution.

Le *troisième chapitre* présente notre travail "Combiner l'algorithme minimax avec un algorithme génétique (GA) " pour faire améliorer les stratégies des jeux. On va montrer d'une manière détaillée les étapes de conception de notre projet. Il s'agit d'une méthode pour améliorer les stratégies dans les jeux de dames dont nous expliqueront ces règles ainsi que la description de la méthode évolutionnaire utilisée pour l'amélioration.

Le *dernier chapitre* est consacré pour la représentation de l'étape de l'implémentation de notre logiciel.

Pour terminer, une conclusion générale résumant le travail, et une exposition de quelques perceptives ,Combiner l'algorithme minimax avec un algorithme évolutionnaire (GA) dans plusieurs domaines autre que les jeux, ainsi que l'optimisation du temps d'exécution des algorithmes.

CHAPITRE I

INTRODUCTION Á LA THÉORIE DES JEUX

Ce chapitre présente les principes fondamentaux de la théorie des jeux qui permettent la modélisation mathématique des jeux, ce qui facilite la tâche de la programmation, tout en explorant les algorithmes de recherche les plus répandus pour représenter et optimiser les stratégies des jeux.

1. Introduction :

Dans de nombreuses situations de la vie quotidienne, la performance d'un acteur, qu'il soit un individu, une entreprise ou un pays, ne dépend pas uniquement de son action, mais aussi de celle prises par les autres. Cette interdépendance stratégique est le domaine de prédilection de *la théorie des jeux.*

La théorie des jeux est une branche des mathématiques, de la recherche opérationnelle et de l'économie, qui a pour objet la modélisation mathématique des situations conflictuelles. Elle propose des concepts opératoires et des outils formels pour l'analyse du conflit ainsi que sa représentation.

En fait, elle consiste à analyser l'interaction dans un groupe d'agents [1] rationnels [2] qui a un comportement stratégique, par exemple elles nous permettent de mieux comprendre le déroulement des guerres. **[3]**

Cette théorie définit une étude des comportements rationnels des individus en situation de conflit, à travers des modèles appelés *jeux*. Elle a été appliquée pour la première fois en science économique où elle a remporté un franc succès. Par la suite, on s'est aperçu que les jeux sont présents dans des domaines aussi inattendus que la biologie, la sociologie, et l'informatique. **[5]**

2. Historique : [3]

Certaines idées de base de la *théorie des jeux* aient été présentées avant même sa naissance à travers les écrits de *Cournot, Zermelo* et d'*Emile Borel*, le résultat pionnier de cette théorie revient à *John Von Newman* en 1928. Ce dernier a démontré le théorème de min-max qui a joué et joue jusqu'à présent un rôle très important dans la théorie des jeux.

C'était en 1944, dans un ouvrage très réputé intitulé « *Game Theory and Economic Behavior* » que le mathématicien *John Von Newman* et l'économiste *Oskar Morgenstern* ont donné naissance à la théorie des jeux.

---

[1] Groupe d'agents. Toute personne qui participe au conflit et capable de prendre une décision, appelé aussi joueur.

[2] Rationalité. La rationalité individuelle d'un joueur est une règle de maximisation du gain individuel.

En répondant aux insuffisances d'équilibres, utilisés dans la microéconomique traditionnelle, *John Nash* a mis en évidence la notion de l'***équilibre de Nash*** qui prend comme référence le principe de la rationalité individuelle. Cette notion peut conduire chaque individu à une situation de non regret mais elle ne peut pas lui garantir un gain optimal.

La théorie des jeux a été, enfin, consacrée par l'obtention du prix de ***Nobel d'économie***, en 1944, attribué aux trois chercheurs *J. Nash*, *C. Harsanyi* et *R. Selten*. Ces derniers ont contribué à faire avancer la science économique.

Assez rapidement, cette théorie a été considérée comme une solution éventuelle aux problèmes de formalisation que connaissaient les sciences économiques, les sciences sociales, les situations politiques, militaires et autres.

3. La théorie des jeux classique :

La théorie des jeux prend comme hypothèse principale la rationalité forte des individus "*Chaque individu cherche à maximiser ses gains personnels en prenant en considération le comportement de ses adversaires*". La théorie classique constitue une approche mathématique des différentes stratégies de chacun des individus et cherche à trouvé une solution optimale pour résoudre les conflits. **[5]**

Tout jeu comporte une liste de joueurs, un ensemble de stratégies possibles pour chacun, et des règles qui donnent les gains des joueurs. Chaque choix stratégique d'un joueur a un impact sur les gains d'un autre joueur, et on parle donc aussi de " *la théorie de la décision en interaction* ". **[6]**

4. La théorie des jeux évolutionniste :

Une branche particulière de la théorie des jeux, développée par des biologistes de l'évolution à partir des années 70, s'est détachée de la théorie initiale classique, on parle ici de la *théorie des jeux évolutionnistes*. Les biologistes ont utilisé les principes de la théorie des jeux pour modéliser certains aspects de l'évolution biologique. **[6]**

En théorie des jeux évolutionniste chaque individu cherche à améliorer non pas son gain personnel mais le gain total de la population dont il fait partie, son avantage est d'éviter le problème majeur de la théorie des jeux classique : *la caractérisation des comportements rationnels* et *la nécessité d'anticiper les actions des autres joueurs*.

Dans les jeux évolutionnaires toute idée de choix stratégique et d'anticipation - et donc de rationalité - est abandonnée, ce n'est plus la rationalité de chaque individu qui le pousse à adapter son comportement aux stratégies de ses adversaires, mais une évolution propre à l'ensemble de la population à laquelle il appartient, et dont il est simplement un acteur parmi d'autres. **[5]**

5. Définition d'un jeu :

Un jeu est une situation où des individus (joueurs) sont conduits à faire des choix parmi un certain nombre d'actions possibles, et dans un cadre défini à l'avance "*les règles du jeu*", qui permet de déterminer qui peut faire quoi et quand.

Les résultats de ces choix constituent une issue du jeu à laquelle est associé un gain pour chacun des participants. Ces résultats ne dépendent pas de la décision d'un seul joueur et ne dépendent pas non plus uniquement du hasard, bien que celui-ci puisse intervenir. **[5]**

En fait, un jeu forme un petit univers plus facile à maîtriser que certains problèmes réels, mais quand même suffisamment complexe pour que l'on puisse faire intervenir des notions typiquement humaines comme *la réflexion* et *le raisonnement*. **[7]**

6. Les types des jeux :

Les jeux sont souvent différenciés par leur appartenance, ou leur non appartenance, à une des catégories suivantes :

6.1. Jeux finis :

Un jeu est dit ***fini***, s'il satisfait aux conditions suivantes :

- Il est joué en un nombre fini de coups.
- Chaque joueur a un nombre fini de choix à chaque coup.

***Exemple* :** Jeu de Nim. (Vider plusieurs tas de pions de taille variable - traditionnellement, il y a trois tas de 3, 4 et 5 pions). On ne peut enlever qu'un nombre fini d'allumettes, et on ne peut jouer que nombre d'allumettes coups au maximum.

6.2. Jeux à somme nulle :

Un jeu à ***somme nulle*** est un jeu où les paiements sont réciproques. C'est à dire que les gains d'un joueur sont les pertes d'autres joueurs. Il n'y a pas d'apport de l'extérieur.

***Exemple* :** Jeu de Dames. Si un joueur gagne, c'est grâce à la défaite de l'autre.

6.3. Jeux à information parfaite/imparfaite :

Un jeu à ***information parfaite*** est un jeu satisfaisant aux conditions suivantes :

- Les joueurs jouent successivement.
- Chaque joueur est parfaitement renseigné sur les coups précédents des autres joueurs.

***Exemple* :** Jeu d'Echecs est à information *parfaite*. Les joueur jouent à tour de rôle et connaissent tous les mouvements depuis le début de la partie.

Un jeu est à ***information imparfaite*** si :

- Un des joueurs ne connaît pas, à un moment du déroulement du jeu, ce qu'a joué un autre joueur. Ceci peut arriver dans le cas où on cache l'information aux joueurs ou parce que les joueurs jouent simultanément.

***Exemple* :** Jeu du dilemme du prisonnier est à information *imparfaite* car les deux joueurs jouent simultanément.

6.4. Jeux à information complète/incomplète :

On dit qu'un jeu est à ***information complète*** si chaque joueur connaît lors de la prise de décision :

- Ses possibilités d'action.
- Les possibilités d'action des autres joueurs.
- Les gains résultants de ces actions.
- Les motivations des autres joueurs.

***Exemple* :** Le jeu du dilemme du prisonnier est à information *complète* car chacun des prisonniers connaît parfaitement la règle du jeu définie par le policier ainsi que l'utilité de l'autre joueur.

Le jeu est dit à ***information incomplète*** si :

- Au moins un des joueurs ne connaît pas entièrement la structure du jeu.

***Exemple* :** jeu de cartes.

6.5. Jeux coopératifs/non coopératifs :

Les jeux ***coopératifs*** sont les jeux dans lesquels on cherche la meilleure situation pour les joueurs sur des critères tels que *la justice*. On considère qu'ensuite les joueurs vont jouer ce qui aura été choisi, il s'agit d'une approche normative.

***Exemple* :** jeu de football.

On appelle jeu ***non coopératif***, tout jeu où les joueurs ne peuvent pas se regrouper en coalitions, ils peuvent être d'accord sur telle ou telle issue, à condition qu'ils ne contractent pas d'accord contraignant. Aucun joueur ne cherchera à manipuler les autres, il ne cherche qu'à maximiser son propre gain.

7. Les types des joueurs :

Afin de mieux comprendre les principes de la théorie des jeux, il faut toujours avoir à l'esprit les deux caractéristiques fondamentales de joueurs théoriques déroulant les algorithmes. Ils sont supposés :

7.1. Joueur intelligent :

Dire qu'un joueur est ***intelligent***, revient à dire qu'il jouera toujours le meilleur coup. Il ne commettra donc jamais d'erreur et jouera toujours le coup l'avantageant le plus.

7.2. Joueur prudent :

Les joueurs ***prudents*** ne prennent pas de risques dans le but de gagner plus. Ils cherchent toujours à minimiser leurs pertes potentielles.

Remarque* :** Pour plus de lisibilité, Il est nécessaire d'adopter dans la suite de ce travail, une notation commune et compréhensible pour décrire les joueurs : On prend donc arbitrairement, le joueur qui fait le premier coup est appelé ***joueur maximisant, l'autre est appelé ***joueur minimisant***.

8. Les stratégies des jeux :

8.1. Généralités :

Une stratégie est un plan d'actions complet pour chaque joueur spécifiant ce que fera ce dernier à chaque étape du jeu et face à chaque situation pouvant survenir au cours du jeu.

Elle décrit totalement le comportement d'un joueur, et peut-être représentée par une série de "*Si…alors…sinon*", prenant en considération toutes les situations possibles. Une partie de jeu se modélise donc en deux coups, c'est à dire les choix de stratégies des deux joueurs. Une fois choisie, une stratégie ne peut être changée, elle détermine le déroulement de toute la partie pour le joueur concerné.

On peut alors se demander quel est le nombre maximal de stratégies dont peut disposer un jeu. Certaines peuvent être comptées plusieurs fois mais ce dénombrement porte sur les façons dont le jeu peut se dérouler et non pas sur les stratégies.

***Exemple* :** Jeu d'échecs : on ne peut pas définir toutes les stratégies possibles.

Remarque :

- Il n'est possible de connaître l'ensemble des stratégies que pour très peu de jeux.
- Dans la plupart des cas, il est impossible de représenter une stratégie dans son intégralité sous une forme rapidement compréhensible (une suite de tests de positions est assez peu digeste à représenter).

8.2. Les types des stratégies : [5]

Il existe deux types de stratégies :

8.2.1. Stratégies pures :

Une stratégie est dite pure si elle ne contient aucune notion d'aléatoire et n'utilise pas des fonctions de probabilité.

8.2.2. Stratégies mixtes :

Ce sont les stratégies qui consistent à donner une distribution de probabilité sur les différentes actions possible.

***Exemple* :** "*Il sait que je sais qu'il sait que je vais appliquer telle stratégie*".

Afin d'éviter ce type de raisonnement au nième degré, on a recours aux stratégies mixtes, dont le principe est d'affecter une probabilité d'être jouée à chaque stratégie, en privilégiant la meilleure stratégie déterminée par le minimax. **[8]**

8.3. Choix de stratégies :

Lorsqu'une *stratégie optimale* existe, elle est choisie. Lorsqu'elle n'existe pas, ou lorsque plusieurs stratégies équivalentes sont disponibles, on effectue un *choix aléatoire*.

9. Représentation des jeux :

Afin de rechercher le meilleur coup, ou la meilleure stratégie, il est nécessaire de représenter un jeu ou une partie d'une manière exploitable. Les deux méthodes de représentation sont :

9.1. Représentation matricielle :

Appelé aussi *forme normale*, ce mode de représentation se situe au niveau des stratégies, dont il modélise les effets ou les résultats.

Dans ce mode de représentation, les lignes et les colonnes représentent les stratégies ouvertes aux deux joueurs. Par convention, les lignes représentent généralement les stratégies du joueur maximisant. Par extension, les colonnes représentent celles du joueur minimisant.

Les valeurs des différentes cases représentent la valeur d'une partie issue de la confrontation des stratégies de la colonne et de la ligne correspondantes. **[8]**

***Remarque* :** Les valeurs des cases sont à considérer du point de vue du joueur maximisant.

***Exemple* :**

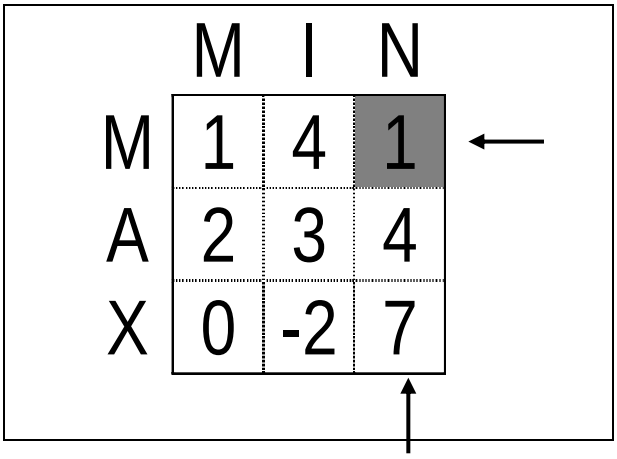

Figure 1.1 : Exemple de représentation matricielle d'un jeu.

Dans cet exemple, si le joueur maximisant choisit la stratégie correspondant à la première ligne, et le joueur minimisant celle correspondant à la troisième colonne, alors le résultat de la partie sera 1, correspondant à un gain du joueur maximisant.

9.2. Représentation arborescente :

Appelé aussi *forme extensive*, Ce mode de représentation se situe au niveau des coups et des positions produites. C'est une représentation explicite de toutes les actions possibles du jeu.

À chaque niveau on a tous les choix possibles pour un joueur, pour un coup donné.

- Les nœuds représentent *les positions* du jeu. Ainsi, la racine est la *position initiale* du jeu, et les feuilles, ou nœuds terminaux correspondent aux *positions de fin de partie*.
- Les arcs représentent *les coups* d'un joueur.
- Les nœuds du premier niveau représentent donc les positions que peut atteindre le premier joueur en un déplacement. Ainsi, chaque chemin partant de la racine vers un nœud terminal représente une partie différente, complète, du jeu.

***Exemple* :**

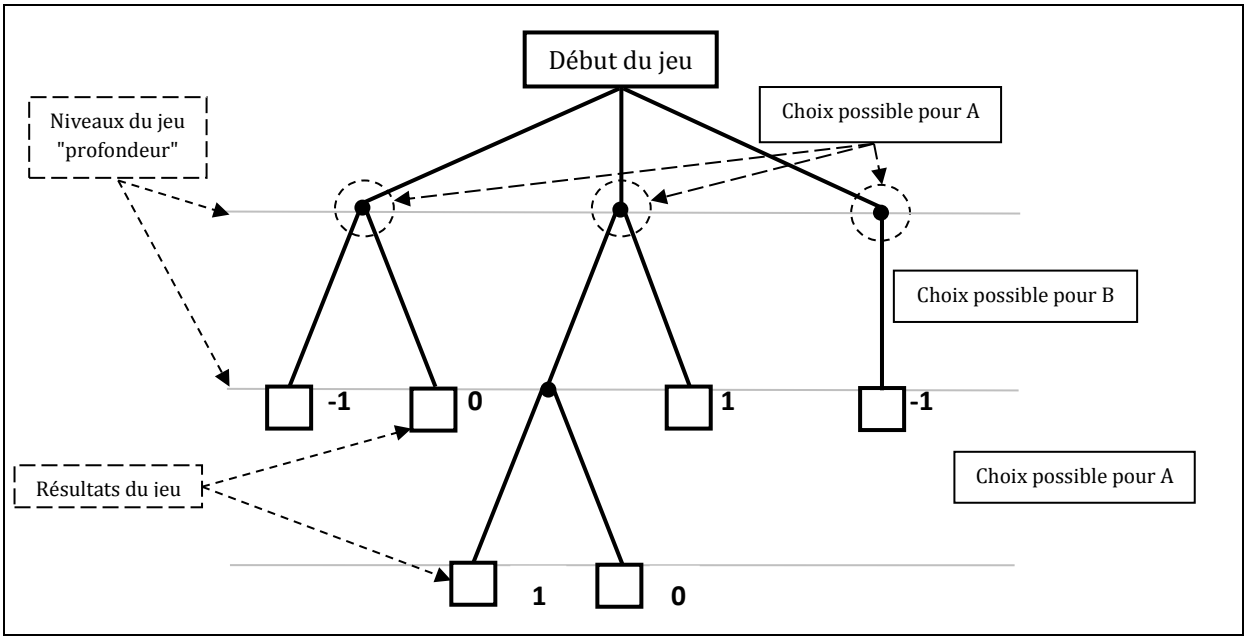

Figure 1.2 : Exemple de représentation arborescente d'un jeu.

***Remarque* :** La représentation d'un arbre de jeu complet se heurte rapidement à une *explosion combinatoire*, tout du moins pour les jeux "intéressants".

***Exemple* :** Le jeu d'échecs, le niveau 2 contient déjà 400 nœuds.

Remarque* :** dans la suite, nous allons adopter la représentation suivante: Les nœuds représentant des positions où le joueur maximisant doit jouer seront représentés par des ***carrés, les autres (joueur minimisant) seront représentés par des ***cercles***.

***Exemple* :**

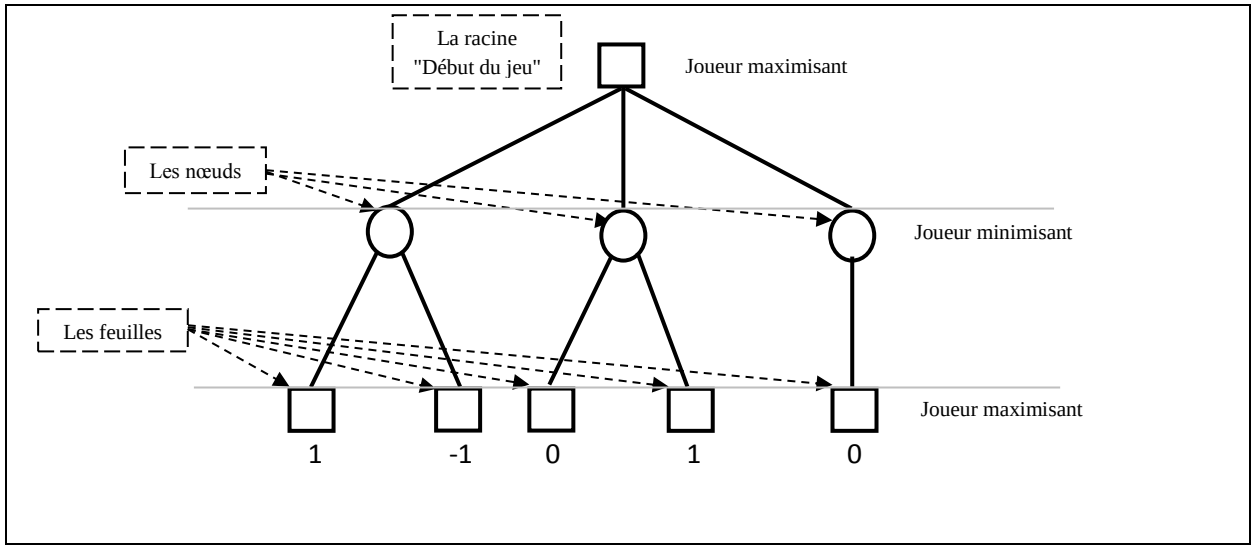

Figure 1.3 : Exemple de représentation d'un arbre de jeu avec des carrés et des cercles.

***Remarque* :** dans la représentation matricielle, les cases correspondent aux feuilles de l'arbre du jeu.

10. Fonction d'évaluation :

Une fonction d'évaluation est une fonction qui associe à une configuration de jeu et à un joueur donné, une valeur réelle. Cette valeur est sensée estimer la qualité de la configuration de jeu pour le joueur, en termes de ses chances de gagner la partie.

Les première fonctions proposées furent les plus simples. Aux jeux de dames par exemple, on peut effectuer la différence entre le nombre de pions du joueur et le nombre de pions de son adversaire.

Il est clair qu'une fonction heuristique élaborée permet de mieux jouer « *si j'inclus dans ma fonction d'évaluation des notions d'attaque et de défense de pions, j'améliore mon niveau de jeu* ». **[11]**

11. Complexité des fonctions d'évaluation :

Une bonne fonction d'évaluation est, a priori, une fonction élaborée incluant différentes stratégies de jeu. Cependant, plus la fonction d'évaluation est compliquée, plus elle met de temps à être calculée. Ce problème est important, car si la fonction est trop longue à calculer, cela peut réduire la profondeur de recherche dans l'arbre de jeu, et donc rendre l'algorithme moins performant. Il faut donc trouver un compromis entre la complexité stratégique et la complexité algorithmique de la fonction. **[26]**

12. Les algorithmes de recherche :

Pour pouvoir résoudre un problème, un *algorithme de recherche* doit réaliser une exploration de l`espace d`états d`une manière systématique et contrôlée.

12.1. L'algorithme de Minimax :

L'algorithme Minimax peut être utilisé avec la forme normale (matricielle) ou avec la forme extensive (arborescente). Voyons ces deux applications :

12.1.1. Forme normale :

Les joueurs étant prudents, ils vont chercher à *minimiser leurs pertes*. Ils vont donc commencer par chercher ce qui peut leur arriver de pire pour chaque stratégie (ligne/colonne), puis par choisir, parmi ces cas de figure, celui qui leur est le moins défavorable.

- **Joueur maximisant :**

Le joueur maximisant prend donc, pour chaque ligne la plus petite valeur algébrique. Il choisit ensuite la stratégie (ligne) correspondant à la plus grande de ces valeurs. Il va donc choisir le maximum parmi les minimums (max min).

- **Joueur minimisant :**

De même, le joueur minimisant prend, pour chaque colonne, la plus grande valeur algébrique (ses pertes correspondent aux gains du joueur maximisant). Il choisit ensuite la

stratégie (colonne) correspondant à la plus petite de ces valeurs. Il va donc choisir le minimum parmi les maximums (min max).

Exemple **:** On a ici une matrice de jeu.

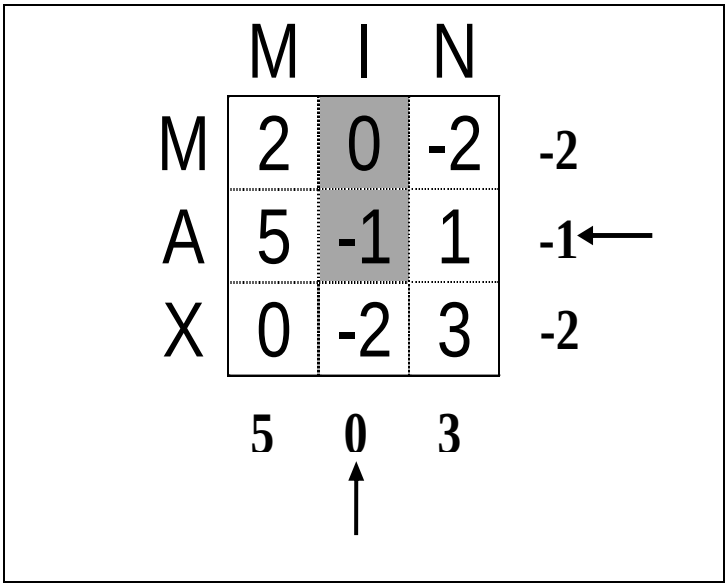

Figure 1.4 : Exemple d'application de l'algorithme Minimax avec la représentation matricielle.

Le joueur maximisant commence par calculer les pires cas pour chaque ligne (stratégies). Il choisit ensuite celle qui lui causera le moins de pertes : la stratégie 2, avec son résultat de -1.

Le joueur minimisant fait de même avec les colonnes (ses stratégies), mais de son point de vue, il cherche donc les plus grandes valeurs. Il choisit ensuite celle qui lui causera le moins de pertes : la stratégie 2 avec son résultat de 0.

Remarque **:** Lorsque l'on a maxmin = minmax, alors on est en présence d'un *point col*, et réciproquement. Les deux stratégies (celle du joueur maximisant et celle du joueur minimisant) aboutissant au point col sont des stratégies dites *pures optimales* « on ne pourra pas obtenir un résultat plus désavantageux en jouant autre chose ».

On peut avoir plusieurs points col. Mais, dès que l'on a un, on ne devra pas appliquer de stratégies mixtes, puisqu'aucun joueur ne pourra faire mieux.

Exemple **:** On est ici en présence d'un jeu avec point col.

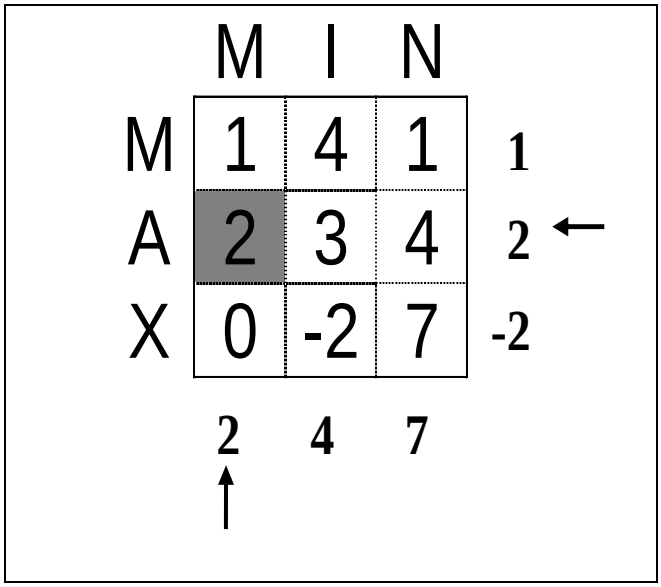

Figure 1.5 : Exemple de représentation matricielle en présence d'un jeu avec point col.

En effet, après avoir appliqué l'algorithme du Minimax, comme dans l'exemple précédent, on s'aperçoit que la valeur issue des deux stratégies optimales est la même : 2.

Si le joueur maximisant utilise une des deux stratégies 1 ou 3, il aura un gain inférieur (1 ou 0) à celui apporté par la stratégie optimale. De même, si le joueur minimisant choisit une stratégie autre que la première, il perdra encore plus (3 ou 4 au lieu de 2). Aucun des deux joueurs ne peut donc faire mieux que son mieux.

12.1.2. Forme extensive :

L'algorithme du Minimax prend en considération le fait que l'adversaire va toujours jouer son meilleur coup. Il est donc nécessaire, pour chaque joueur de rechercher le coup qui va lui assurer le minimum de perte "*stratégie sécurisante*".

Ainsi, à chaque nœud où le joueur maximisant doit prendre une décision, il va considérer la plus grande des valeurs de ses fils. Réciproquement, le joueur minimisant prendra la plus petite des valeurs de ses fils. **[8]**

***Exemple* :**

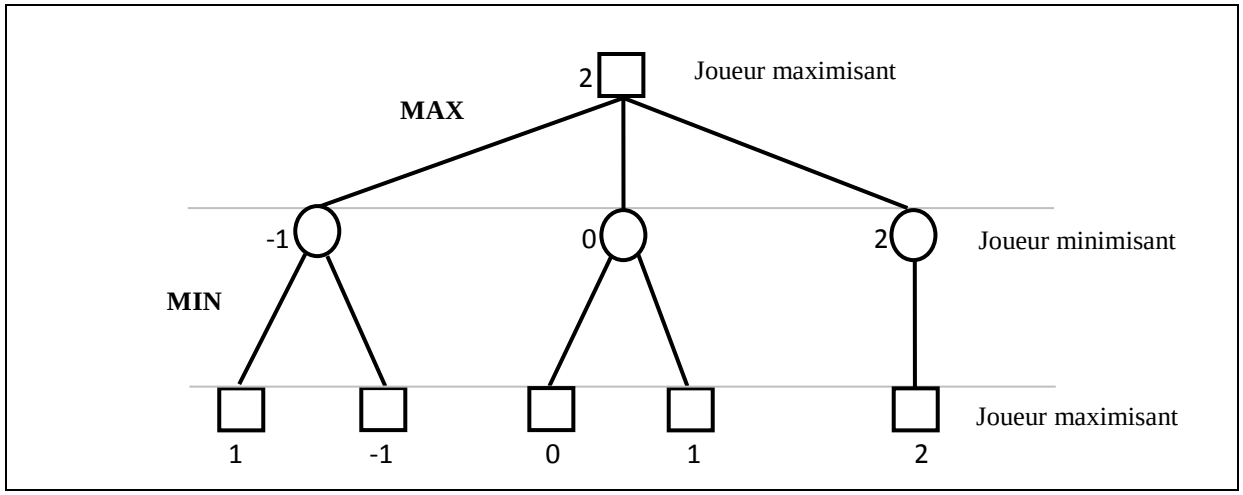

Figure 1.6 : Exemple de calcul de la valeur d'un nœud en fonction des valeurs de ses fils.

Cette notation se fait donc en remontant l'arbre, c'est à dire les feuilles les plus éloignées jusqu'aux descendants de la racine. À ce stade, le premier joueur doit jouer le coup qui maximise la note affectée aux descendants.

L`algorithme Minimax peut être décrit par le pseudo code suivant :

Minimax

```
int fonction minimax (int profondeur)
{
   Si (jeu est terminé ou profondeur = 0)
   retourner Score résultant ou eval();
int Current;
Mouvement MeilleurMouvement;
   Si (nœud == MAX) {
      Current = -INFINI;
      Pour chaque (Mouvement m) {
         Faire le mouvement m;
         int score = minimax (profondeur - 1)
         Retirer le mouvement m;
         Si (score > Current) {
             Current = score;
             MeilleurMouvement = m ;
         }
      }
   }
   Sinon {
      Current = +INFINI;
      Pour chaque (Mouvement m) {
         Faire le mouvement m;
         int score = minimax (profondeur - 1)
         Retirer le mouvement m;
         Si (score < Current) {
             Current = score;
             MeilleurMouvement = m ;
         }
      }
   }
   retourner Current;
}
```

Algorithme 1.1 : Algorithme de Minimax.

Exemple **:**

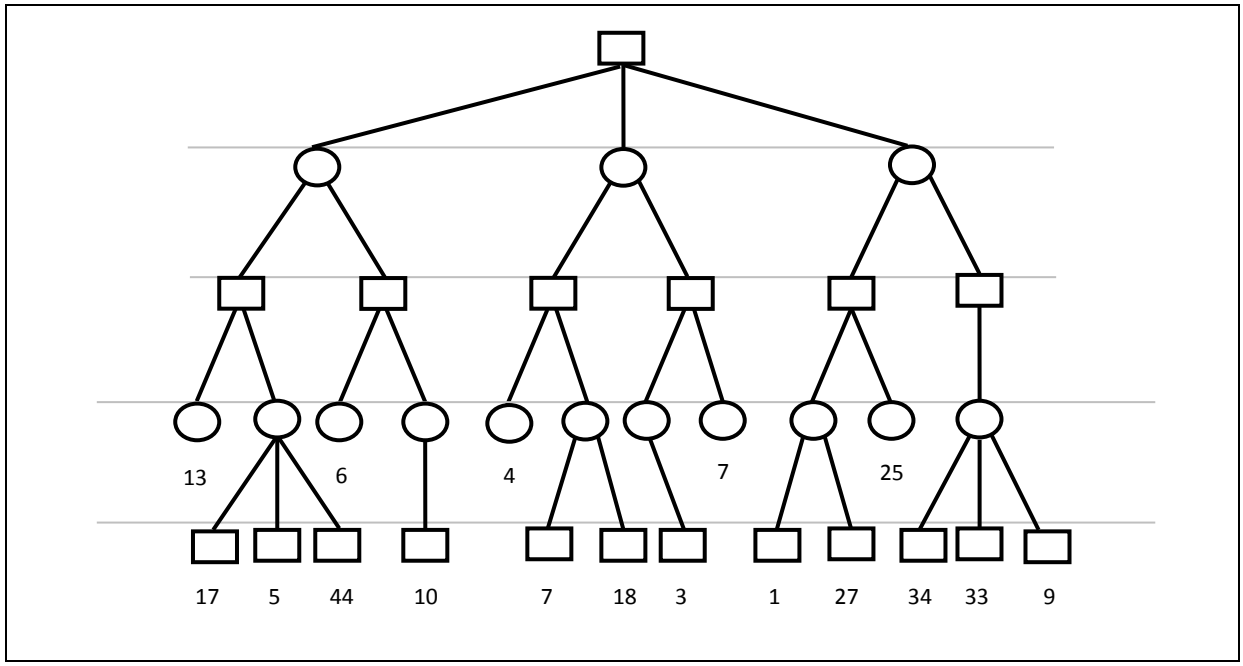

Figure 1.7 : Exemple d'arbre de jeu.

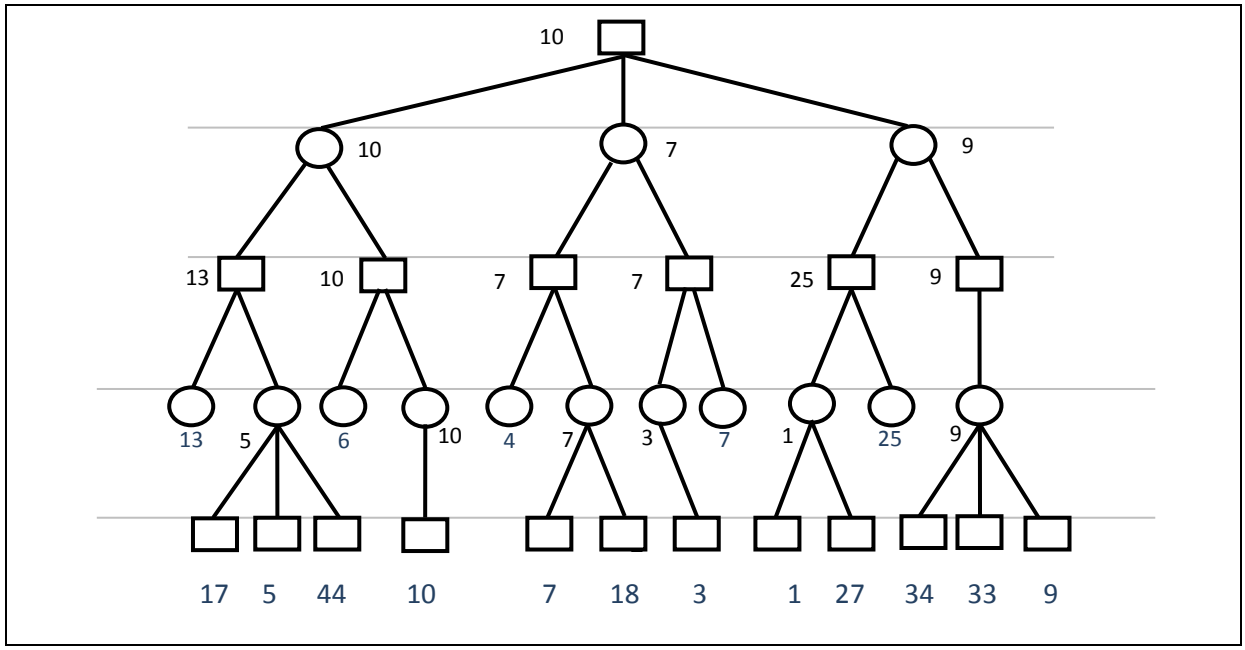

Figure 1.8 : Exemple résolu avec l'algorithme du Minimax.

12.1.3. Les limites de l'algorithme Minimax :

L'algorithme du Minimax est bien entendu, inutilisable dans la pratique, puisqu'il explore toutes les branches de l'arbre sans distinction alors que certaines branches n'apporteront rien. Son principe est néanmoins repris par tous les autres algorithmes dont les améliorations consisteront principalement à introduire des méthodes d'élagage. C'est à dire que l'on va éviter d'analyser les branches de l'arbre qui ne peuvent plus influencer le résultat à un moment donné du calcul.

On observe que l`algorithme Minimax effectue l`évaluation pour tous les nœuds de l`arbre de jeu d`un horizon donné. Mais il existe des situations dans lesquelles, pour déterminer la valeur Minimax associée à la racine, il n'est pas nécessaire de calculer les valeurs associées à tous les nœuds de l'arbre. **[8]**

12.2. L'élagage Alpha-Bêta :

Une amélioration du Minimax est de faire *un élagage* de certaines branches de l'arbre. L'idée de base est de retenir les valeurs de référence des ancêtres du nœud où l'on se trouve. Ces valeurs servent de référence, permettant ainsi de ne pas parcourir les branches inutiles car ils ne pouvant plus influencer la valeur Minimax de la racine.

En d'autres termes, l'algorithme *alpha-bêta "α–β"* utilise l'histoire de la recherche déjà effectuée lors du début de la recherche minimax pour borner les variations possibles de la notation de la racine, ce qui permet d'élaguer de façon non risquée des sous-arbres entiers de l'espace de recherche (on parle de *coupures alpha-bêta*).

12.2.1. Les limites Alpha et Bêta :

Les deux valeurs de référence conservées sont appelées α et β, d'où le nom de l'algorithme :

- **La limite α :**

La limite α est la limite inférieure de coupure pour un nœud *Min J*. Il représente la valeur courante maximale de tous les ancêtres de *J*. On arrête l'exploration d'un nœud *J* dès que la valeur courante "*cv*" vérifie : $cv \leq \alpha$. On a donc, initialement : $\alpha = -\infty$

- **La limite β :**

La limite β est la limite supérieure de coupure pour un nœud *Max J*. Il représente la valeur courante minimale de tous les ancêtres de *J*. On arrête l'exploration d'un nœud J dès que la valeur courante "*cv*" vérifie : $cv \geq \beta$. On a donc, initialement : $\beta = +\infty$

12.2.2. Avantages par rapport au Minimax :

L'avantage de cet algorithme est qu'il donne exactement les mêmes résultats que l'algorithme Minimax tout en étant beaucoup plus rapide. Si les nœuds sont correctement ordonnés la vitesse de l'algorithme est accrue du fait que les coupes surviennent plus tôt. Ainsi pour un même temps d'exécution, l'algorithme Alpha-Bêta permettra d'aller à une profondeur plus grande.

En effet, Alpha-Bêta est l'algorithme le plus utilisé dans les applications de jeux, en partie grâce à sa consommation de mémoire linéaire. Cette consommation est proportionnelle à la profondeur de l'arbre analysé, puisque l'on stocke à tout moment un couple (α,β) par ancêtre du nœud local.

Bien entendu, par consommation de mémoire, on entend consommation propre à l'algorithme, et non pas la consommation due à l'arbre, celui-ci ne variant pas en fonction de l'algorithme utilisé. **[10]**

L`algorithme de Alpha-Bêta peut être décrit par le pseudo code suivant :

Alpha-Bêta :

```
int alphabêta(int profondeur, int alpha, int bêta)
{
   Si (jeu est terminé ou profondeur <= 0)
      retourner score résultant ou eval();
   mouvement MeilleurMouvement;
   Si (nœud == MAX) {
      Pour chaque (mouvement m) {
         Faire le mouvement m;
         int score = alphabêta (profondeur - 1, alpha, bêta)
         Retirer le mouvement m;
         Si (score > alpha) {
            alpha = score;
            MeilleurMouvement = m ;
            Si (alpha >= bêta)
               break;
         }
      }
      retourner alpha ;
   }
   Sinon {
      Pour chaque (mouvement m) {
         Faire le mouvement m;
         int score = alphabêta (profondeur - 1, alpha, bêta)
         Retirer le mouvement m;
         Si (score < bêta) {
            bêta = score;
            MeilleurMouvement = m ;
            Si (alpha >= bêta)
               break;
         }
      }
      retourner bêta;
   }
}
```

Algorithme 1.2 : Algorithme d'Alpha-Bêta.

Exemple **:**

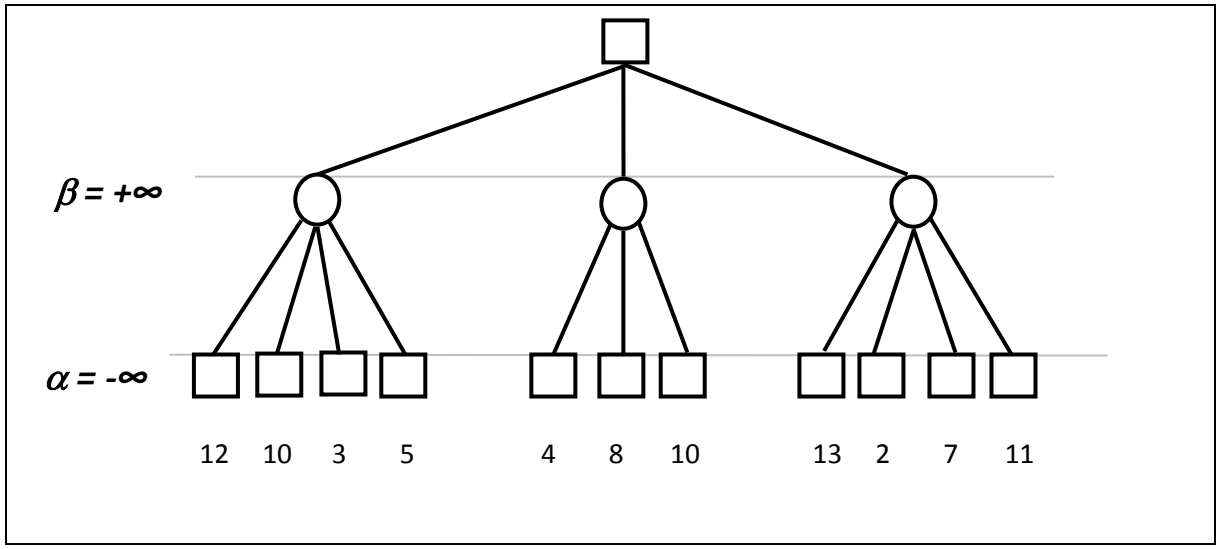

Figure 1.9 : Exemple d'arbre de jeu.

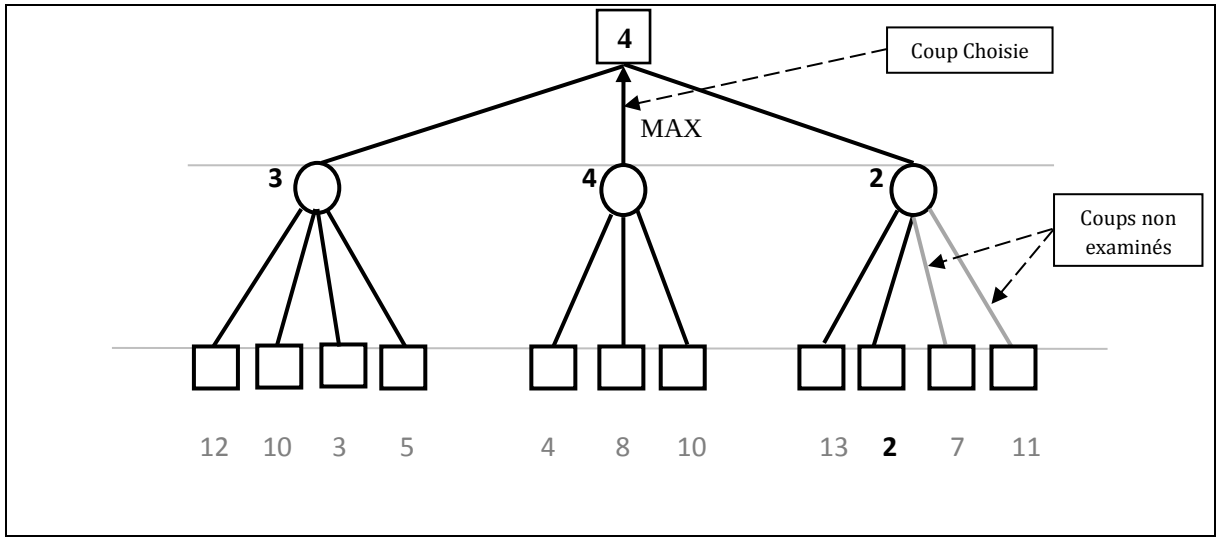

Figure 1.10 : Exemple résolu par l'algorithme α–β Sens de parcours de droite à gauche.

12.3. La convention NegaMAx :

C'est une convention simplificatrice (du moins pour le programmeur) qui consiste à considérer qu'on évalue une position non pas du point de vue d'un joueur fixe (par exemple joueur Max = programme) mais du point de vue du joueur qui a le trait sur cette position.

Pour conserver alpha et bêta pour chaque niveau il suffit de les intervertir entre les appels de procédures et d'inverser leurs valeurs. En d'autres termes, on utilise encore, implicitement, la condition de symétrie des jeux à somme nulle : **[11]**

Position.eval (Joueur) = - Position.eval (Adversaire)

L`algorithme de Alpha-Bêta, en convention NegaMax peut être décrit par le pseudo code suivant :

Alpha-Bêta, en convention NegaMax

```
int alphabêta(int profondeur, int alpha, int bêta)
{
   Si (jeu est terminé ou profondeur <= 0)
      retourner score resultant ou eval();
   mouvement MeilleurMouvement ;
   Pour chaque (mouvement m) {
      Faire le mouvement m;
      int score = -alphabêta(pronfondeur - 1, -bêta, -alpha)
      Retirer le mouvement m;
      Si (score >= alpha) {
         alpha = score ;
         MeilleurMouvement = m ;
      Si (alpha >= bêta)
         break;
      }
   }
   retourner alpha;
}
```

Algorithme 1.3 : Algorithme d'Alpha-Bêta, en convention NegaMax.

12.4. Algorithme Fail-Soft Alpha-Bêta :

L'algorithme de Fail-soft Alpha-Bêta est une amélioration de Alpha-Bêta en modifiant légèrement le pseudo-code de l'algorithme, en convention NegaMax, il est possible de ramener un supplément d'information de la recherche, "*la valeur qui a provoqué la coupure*".

L`algorithme de Fail-Soft alpha-bêta peut être décrit par le pseudo code suivant :

Fail-Soft Alpha-Bêta

```
int alphabêta(int profondeur, int alpha, int bêta)
{
   Si (jeu est terminé ou profondeur <= 0)
      retourner score résultant ou eval();
   mouvement MeilleurMouvement ;
   int current = -INFINI;
   Pour chaque (mouvement m) {
      Faire le mouvement m;
      int score = - alphabêta(profondeur - 1, -bêta, -alpha)
      Retirer le mouvement m;
      Si (score >= current) {
         current = score;
         MeilleurMouvement = m;
         Si (score >= alpha){
            alpha = score;
            MeilleurMouvement = m ;
            Si (score >= bêta)
               break;
         }
      }
   }
   return current;
}
```

Algorithme 1.4 : Algorithme de Fail-Soft Alpha-Bêta.

13. Conclusion :

Dans ce chapitre nous avons présenté les notions de base de la théorie des jeux. L'intérêt principal de cette théorie consiste à étudier les différentes interactions entre les individus. La théorie des jeux reste un modèle très fécond, qui suscite de nombreux travaux, en matière de représentation des jeux ainsi que le choix de la meilleure fonction d'évaluation tout en permettant la modélisation de la non-linéarité même sans passer par les équations du modèle mathématique.

CHAPITRE II

ALGORITHMES ÉVOLUTIONNAIRES: PRINCIPES ET MÉTHODES

Ce chapitre présente les principes de l'évolution naturelle et artificielle. Nous détaillons les grandes lignes des algorithmes évolutionnaires et plus particulièrement les algorithmes génétiques et les stratégies d'évolution. Nous présentons ensuite nos choix concernant le type d'évolution que nous allons utiliser dans la suite de ce mémoire.

1. Introduction :

Il existe une catégorie de problèmes pour lesquels il est difficile, voire impossible, de trouver une solution en un temps limité. Il est alors utile de trouver une technique permettant la localisation rapide de solutions sous-optimales, sachant que l'*espace de recherche* a une *taille* et une *complexité* suffisamment importantes pour éliminer toute garantie d'*optimalité*. Pour cela, un système qui est capable de s'auto-modifier au cours du temps, tout en améliorant sa *performance* dans l'accomplissement des tâches auxquelles il est confronté, semble ouvrir la voie à une recherche intéressante. **[12]**

L'évolution biologique a engendré des systèmes vivants extrêmement complexes. Elle est le fruit d'une altération progressive et continue des êtres vivants au cours des générations. **[4]**

Une voie de recherche qui est très active de nos jours est celle qui essaye de développer de nouvelles approches s'inspirant des principes de cette évolution pour traiter plus efficacement les différents problèmes, notamment ceux portant sur l'*optimisation*. **[13]**

« *La vie est une compétition, et seuls les mieux adaptés survient et se reproduisent* », dit *Darwin*. Cette règle, qui a engendré les organismes que nous connaissons aujourd'hui, est utile pour la résolution de problèmes par ordinateur. **[14]**

Les principes de la ***sélection naturelle*** [1] et de la ***reproduction***, présentés pour la première fois par *Darwin*, ont inspiré bien plus tard les chercheurs en informatique. Ils ont donné naissance à une classe d'algorithmes regroupés sous le nom générique d'***Algorithmes Evolutionnaires*** (Evolutionary Algorithms (EA)). **[2]**

On distingue quatre grandes familles historiques d'algorithmes évolutionnaires et les différences entre elles ont laissé des traces dans le paysage évolutionnaire actuel, en dépit d'une unification de nombreux concepts.

- Algorithmes Génétiques (*GA*) : J. Holland, 1975 et D.E. Goldberg, 1989. Michigan, USA.
- Stratégies d'évolution (*ES*) : I. Rechenberg et H.P. Schwefel, 1965. Berlin.
- Programmation évolutionnaire (*EP*) : L.J. Fogel, 1966 et D.B. Fogel, 1991, 1995. Californie, USA.
- Programmation génétique (*GP*) : J. Koza, 1990. Californie, USA. **[9]**

[1] Sélection naturelle. Les gènes les plus performants ont tendance à se diffuser dans la population tandis que ceux qui ont une performance relative plus faible ont tendance à disparaître.

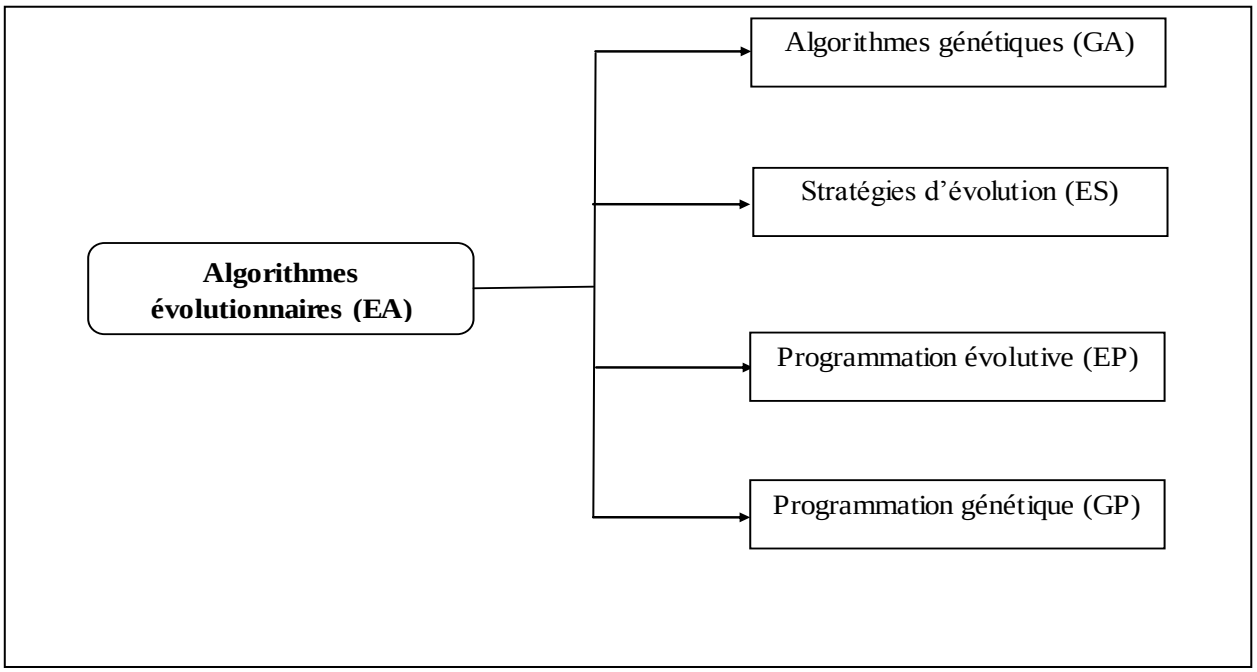

Figure 2.1: Différentes branches des algorithmes évolutionnaires.

2. Les Algorithmes évolutionnaires :

Les *EA* désignent un ensemble d'algorithmes stochastiques qui utilisent les principes de l'évolution naturelle pour résoudre des problèmes divers, très majoritairement d'*optimisation*. Leur fonctionnement repose sur la *génération aléatoire* de solutions potentielles, puis la *sélection* des meilleurs candidats selon un critère préétabli **[12]**. Ils commencent avec une population initiale, candidate, qui évolue vers la solution optimale ou du moins vers une solution proche de l'optimale.

Ces algorithmes résolvent le problème de la *recherche* des bons éléments, les meilleurs produisent plus souvent que les mauvais. En d'autres termes, si tout va bien dans l'algorithme les enfants doivent être meilleurs que les parents. **[15]**

Les *EA* ont pour but d'optimiser une fonction réelle. En effet, peu de connaissances sur la manière de résoudre ces problèmes sont nécessaires, même si certaines peuvent être exploitées afin de rendre plus efficace l'évolution (en effet, il n'est pas réaliste d'espérer obtenir une méthode d'optimisation raisonnablement efficace si aucune connaissance sur le domaine à traiter). C'est pourquoi, dans de nombreux domaines, les chercheurs ont été amenés à s'y intéresser. **[16]**

Un EA est défini par :

- **Séquence[1]/Individu/Chromosome :** une solution potentielle du problème qui correspond à une valeur codée de la variable (ou des variables) en considération.
- **Population :** un ensemble de chromosomes ou de points de l'espace de recherche (donc des valeurs codées des variables).
- **Espace de recherche :** l'environnement (caractérisé en termes de performance correspondant à chaque individu possible).
- **Fonction de performance (fitness) :** appelée aussi fonction d'évaluation, c'est la fonction - positive - que nous cherchons à maximiser car elle représente l'adaptation de l'individu à son environnement. **[17]**

3. Description détaillée des algorithmes évolutionnaires :

Voici une description plus précise d'un *EA* :

Il s'agit d'un algorithme itératif de recherche globale dont le but est d'*optimiser la fonction de performance* définie par l'utilisateur, pour atteindre cet objectif, l'algorithme travaille en parallèle sur une population d'*individus* (chromosomes[2]), distribués dans l'entièreté de *l'espace de recherche* (l'environnement).

La fonction de performance d'un individu est normalement indépendante des autres individus de la population (elle est explicitement donnée par l'utilisateur). Chaque individu ou chromosome est constitué d'un ensemble d'éléments appelés *caractéristiques* ou *gènes*[3], pouvant prendre plusieurs valeurs appelées *allèles* appartenant à un alphabet non nécessairement numérique. Dans l'algorithme de base, les allèles possibles sont 0 et 1, et un chromosome est donc une chaine binaire [23]. *Exemple* : 001001

Le but est de chercher la *combinaison optimale* de ces éléments, qui donne lieu au maximum de performance. A chaque itération, appelé *génération*, est créée une nouvelle population avec le même nombre d'individus.

Cette nouvelle génération consiste généralement en des individus mieux *adaptés* à l'environnement tel qu'il est présenté par la fonction de performance. La génération d'une nouvelle population à partir de la précédente s'effectue en trois étapes : *Evaluation*, *Sélection* et *Reproduction*.

[1] Séquence. Une séquence A de longueur L(A) une séquence A= {a1, a2, …, a $_i$} avec ∀ i∈ {1,…, L}, a_i∈ V= {0,1}.

[2] Chromosomes. Une suite de gène.

[3]Gène. Un gène est repérable par sa position (son locus) sur le chromosome en question.

4. Principes généraux des algorithmes évolutionnaires :

Les *EA* sont une classe d'algorithmes d'optimisation par recherche probabiliste. Ils modélisent une population d'individus par des points dans un espace. **[16]**

Quel que soit le type de problème à résoudre, les EA opèrent selon les principes suivants :

- La population est *initialisée* de façon dépendante du problème à résoudre (l'environnement).
- La population *évolue* de génération en génération à l'aide d'opérateurs de sélection, de recombinaison et de mutation.
- L'environnement a pour charge d'*évaluer* les individus en leur attribuant une performance. Cette valeur favorisera la *sélection des meilleurs individus*, en vue, après reproduction (opérée par la mutation et/ou recombinaison), d'améliorer les performances globales de la population. **[18]**

La Figure suivante présente l'organigramme d'un AE :

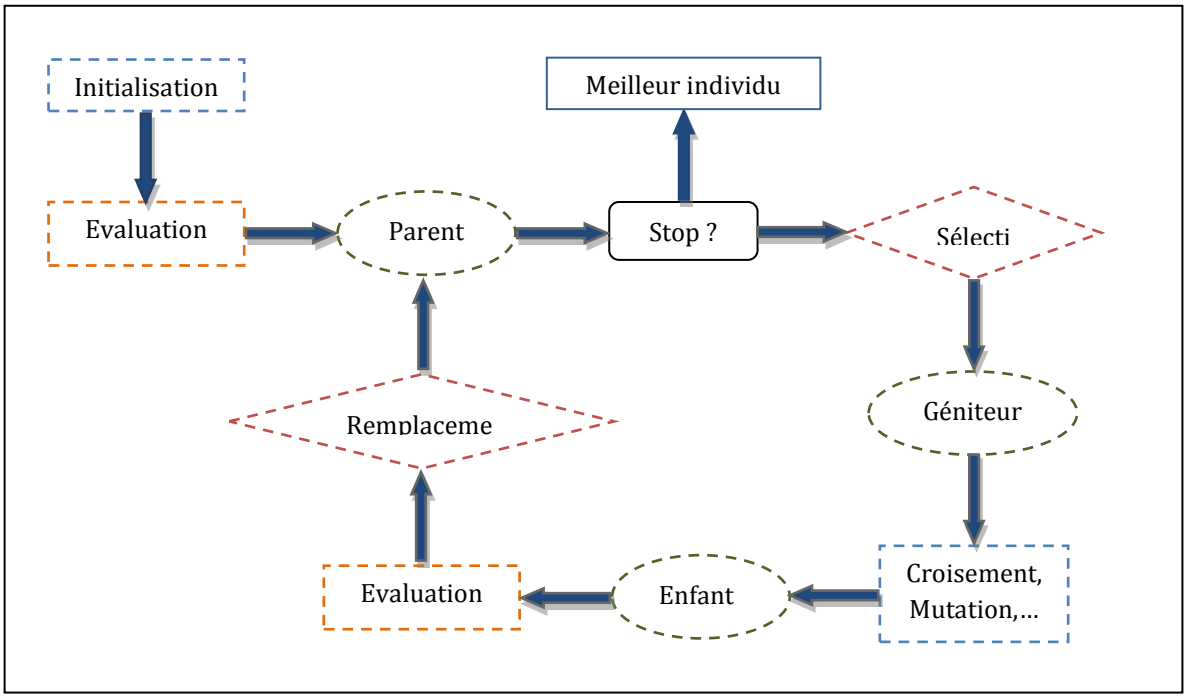

Figure 2.2: Organigramme d'un algorithme évolutionnaire.

5. Éléments de base d'un Algorithme évolutionnaire :

Pour utiliser un EA, on doit disposer des cinq éléments suivants :

5.1 Un principe de codage de l'élément de population :

Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne le succès des algorithmes évolutionnaires.

Les *codages binaires* ont été très utilisés à l'origine, ils ont donné de très bons résultats. Cependant, de nouvelles versions modifiées des algorithmes évolutionnaires originaux ne se basent plus sur le codage binaire des paramètres à optimiser, mais travaille directement sur les paramètres eux-mêmes, on parle alors des codages réels. **[19]**

Les *codages réels* ont été utilisés dans des systèmes de commande, dans l'apprentissage des réseaux de neurones, le chromosome dans ce cas est une chaîne de réels. **[15]**

5.2 Un mécanisme de génération de la population initiale :

Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.

5.3 Une fonction à optimiser :

Celle-ci retourne une valeur appelée *fonction de performance* ou d'évaluation de l'individu. Elle doit traduire correctement le problème à résoudre c'est-à-dire que l'évaluation des chromosomes qui sont les solutions candidate au problème posé doit être pertinente **[15]**. Si par exemple on avait à chercher le maximum d'une fonction (problème posé), un chromosome devrait représenter les valeurs des variables de la fonction, et une bonne fonction d'évaluation serait la fonction même à maximiser. **[19]**

5.4 Des opérateurs :

Permettant de diversifier la population au cours des générations et d'explorer l'espace d'état.

5.4.1 Opérateur de sélection :

L'algorithme génétique sélectionne les individus sur la base de leur fonction de performance, plus précisément, l'opérateur de sélection sélectionne chaque individu *i* avec une probabilité $\boldsymbol{fi/\sum fj}$ (la somme étant prise sur les *n* individus constituant la population), comme l'opérateur de sélection est appliqué n fois, l'espérance mathématique du nombre d'enfants de *i* sera $\boldsymbol{n\ fi/\sum fj}$**.** Les individus sélectionnés constituent une nouvelle population. **[20]**

Les différentes méthodes de sélection :

5.4.1.1 Sélection par roulette :

Les parents sont sélectionnés en fonction de leur performance. Lors du tirage d'un individu à la roulette, un individu *S* est sélectionné selon une probabilité proportionnelle à *S*.

5.4.1.2 Sélection par rang :

La sélection par rang trie d'abord la population par la fonction d'évaluation. Ensuite, chaque individu se voit associé un rang en fonction de sa position. L'individu le plus mauvais aura le premier rang, le suivant aura le deuxième rang, et ainsi de suite jusqu'au meilleur individu qui aura le rang *N* (pour une population de *N* individu).

5.4.1.3 Sélection par tournois :

Choix uniforme de deux individus dans une population, puis choix du meilleur de l'échantillon, c'est-à-dire sur une population de *N* chromosomes, on forme *N/2* paires chromosomes ensuite on sélectionne le meilleur de chaque paire. Dans ce type de sélection, on détermine une *probabilité de victoire* du plus fort, cette probabilité représente la chance que le meilleur chromosome de chaque paire soit sélectionné.

5.4.1.4 Elitisme :

À la création d'une nouvelle population, il y a une grande chance que les meilleurs chromosomes soient perdus après les opérations d'hybridation et de mutation. Pour éviter cela, on utilise la méthode d'élitisme. Elle consiste à copier un ou plusieurs des meilleurs chromosomes dans la nouvelle génération. Ensuite, on génère le reste de la population selon

l'algorithme de reproduction. Cette méthode améliore considérablement les algorithmes évolutionnaires, car elle permet de ne pas perdre les meilleures solutions.

5.4.2 Opérateur de mutation :

La mutation agit en modifiant aléatoirement un ou plusieurs gènes (bits) d'un chromosome. Cet opérateur est appliqué avec une probabilité fixée (*Pm*) sur chacun des individus de la population.

L'opérateur de mutation fonctionne comme suit :

Mutation

```
Pour chaque bit de l'individu
  Générer un réel aléatoire r, tel que r ∈ [0, 1]
  Si r < Pm alors
    le bit sera inversé
```

Algorithme 2.1 : Algorithme de mutation.

5.4.3 Opérateur de croisement :

Le croisement permet de créer de nouvelles chaines en échangeant de l'information entre deux chaines. Le croisement s'effectue en deux étapes :

Etape 01 **:** les nouveaux éléments produits par la reproduction appariés.

Etape 02 **:** chaque paire de chaines subit un croisement comme suit : un entier *k* représentant une position sur la chaine est choisi aléatoirement dans l'intervalle [1, (L – (L-1))], tel que *L* est la langueur de la chaine. Deux nouvelle chaine sont crées en échangeant tous les caractères compris entre les positions *k+1* et *L* inclusivement.

Le croisement peut se faire selon deux méthodes :

5.4.3.1 Croisement en deux points :

On choisi au hasard deux point de croisement et on échange les parties de chaine situées entre ces deux points. **[20]**

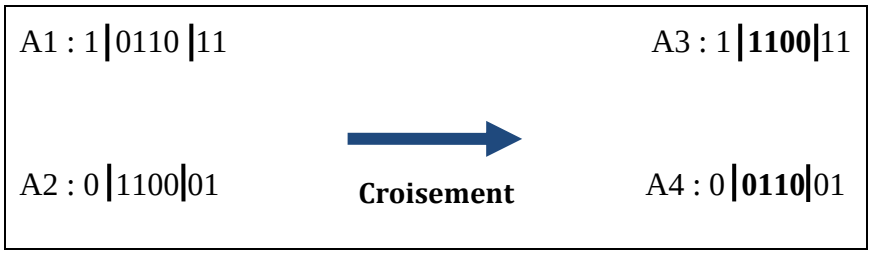

Figure 2.3 : Croisement en deux points.

5.4.3.2 Croisement uniforme (multi-points) :

Dans ce type de croisement, on utilise un *masque de croisement*, qui consiste en un vecteur généré aléatoirement, de longueur identique aux chaines parents, et composé de 0 et 1.

Lorsque le bit du masque vaut 0, l'enfant hérite le bit du premier parent, sinon il hérite du second parent. Le seconde enfant est le complémentaire du premier. Ce croisement peut être considéré comme une génération du croisement *multipoints* sans connaissance préalable du point de croisement. **[20]**

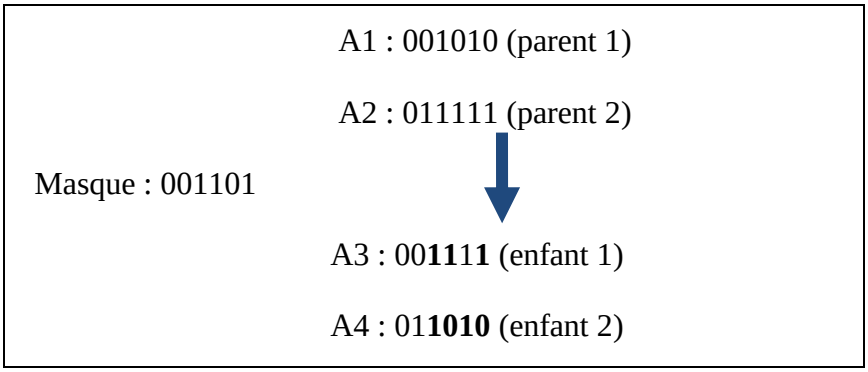

Figure 2.4 : Croisement uniforme.

5.5 Des paramètres de dimensionnement :

La *taille de la population*, le *nombre total de générations* ou *critère d'arrêt*, les *probabilités* d'application des opérateurs de croisement et de mutation.

Dans les sections suivantes, nous allons présenter deux méthodes classiques des *EA* qui sont les algorithmes génétiques (*GA*) et les stratégies d'évolution (*ES*) afin de choisir une des deux méthodes pour la réalisation de notre mémoire.

6 Algorithmes génétiques :

Développés dans les années 70 avec le travail de Holland puis approfondis par Goldberg, Les *GA* sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la *sélection naturelle* et de la *génétique* **[4]**. Ils s'inspirent des mécanismes de l'évolution biologique pour les transposer à la recherche de solutions adaptées au problème qu'on cherche à résoudre.

Ce sont des algorithmes *robustes* car ils peuvent résoudre des problèmes fortement non-linéaires et discontinus, et *efficaces* car ils font évoluer non pas une solution mais toute une population de solutions potentielles et donc ils bénéficient d'un parallélisme puissant.

Les *GA* sont certainement la branche des *EA* la plus connue et la plus utilisée. La particularité de ces algorithmes est le fait qu'ils font évoluer des populations d'individus codés par une chaîne binaire. Ils utilisent les opérateurs de mutation binaire et de recombinaison de différents types. **[17]**

6.1 Principes de fonctionnement :

A partir d'un problème qu'on cherche à résoudre, Le fonctionnement d'un *GA* est défini par les phases suivantes :

- **Initialisation:** Une population initiale de N chromosomes est tirée aléatoirement.
- **Évaluation:** Chaque chromosome est décodé, puis évalué.
- **Sélection:** Création d'une nouvelle population de N chromosomes par l'utilisation d'une méthode de sélection appropriée.
- **Reproduction:** Possibilité de croisement et de mutation au sein de la nouvelle population.
- **Retour:** A la phase d'évaluation tant que la condition d'arrêt du problème n'est pas satisfaite.

On va détailler ces principes dans ce qui suit :

6.1.1 Sélection :

La sélection proposée par Goldberg consiste à sélectionner les individus proportionnellement à leur performance. Un individu ayant une forte valeur d'adaptation a alors plus de chances d'être sélectionné qu'un individu mal adapté à l'environnement. **[4]**

6.1.2 Croisement :

Le croisement (recombinaison) consiste à sélectionner aléatoirement une position de césure et de permuter les parties droites des deux parents. **[4]**

***Exemple illustratif* :**

Un entier *k* est choisi aléatoirement entre 1 et la taille *L* des chaînes moins 1.

Les nouvelles chaînes sont créées en échangeant tous les caractères compris entre les positions *k+1* et L inclues.

Par exemples, considérons les deux chaînes binaires *A* et *B* (parents) et supposons k=2, on obtient alors les chaînes *A1* et *B1* (fils):

Parents	Fils
A : 00 | 101	A1 : **01**101
B : 01 | 011	B1 : **00**011

Recombinaison

Figure 2.5 : Application de l'opérateur de Croissement.

6.1.3 Mutation :

Une mutation consiste simplement en l'inversion d'un bit (ou de plusieurs bits, mais vu la probabilité de mutation c'est extrêmement rare) se trouvant en un locus bien particulier et lui aussi déterminé de manière aléatoire. **[4]**

***Exemple illustratif* :**

La mutation consiste juste à choisir aléatoirement un caractère d'une chaîne et à le modifier.

Par exemple, soit les chaînes binaires suivantes :

Chromosomes avant mutation	Chromosomes après mutation
00101	01101
11100	11000

Mutation

Figure 2.6 : Application de l'opérateur de Mutation.

6.1.4 Remplacement :

Le remplacement consiste à réintroduire les descendants obtenus par application successive des opérateurs de sélection, de croisement et de mutation (la population P') dans la population de leurs parents (la population P).

Un GA générique à la forme suivante :

Algorithme Génétique :

```
1. Initialiser la population initiale P.
2. Evaluer P.
3. TantQue (Pas Convergence) faire :
     a. P ' = Sélection des Parents dans P ;
     b. P ' = Appliquer Opérateur de Croisement sur P ' ;
     c. P ' = Appliquer Opérateur de Mutation sur P ' ;
     d. P = Remplacer les Anciens de P par leurs Descendants
        de P ' ;
     e. Evaluer P ;
   FinTantQue
```

Algorithme 2.2 : Algorithme génétique.

***Remarque*:** Le critère de convergence peut être de nature diverse, par exemple :

- Un taux minimum qu'on désire atteindre d'adaptation de la population au problème.
- Un certain temps de calcul à ne pas dépasser.
- Une combinaison de ces deux points.

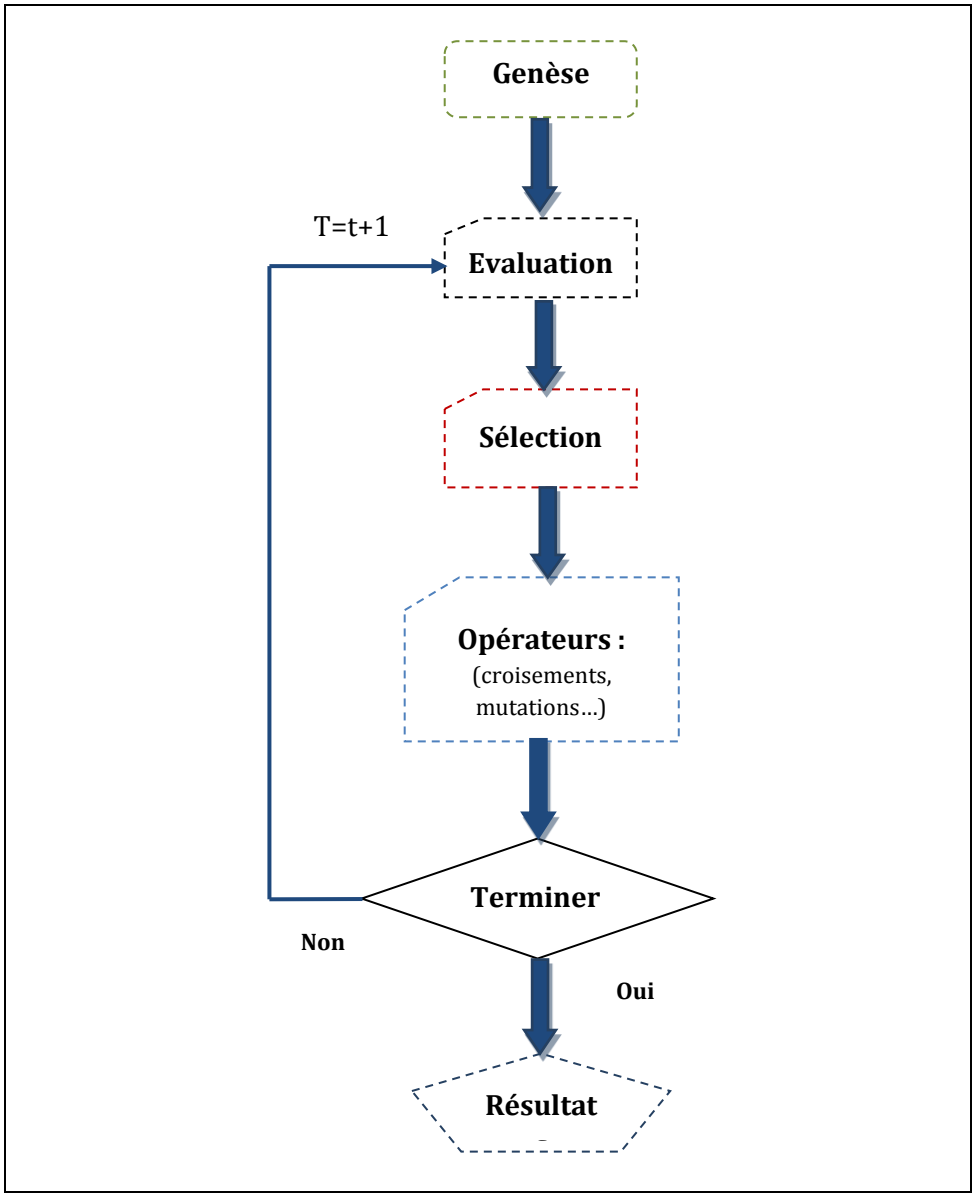

Figure 2.7: Organigramme d'un algorithme génétique.

7 Stratégies d'évolution :

Les *ES* sont apparues dans les années 70 avec les travaux de Ingo Rechenberg, ensuite ces travaux ont été poursuivis par Hans-Paul Schwefel. **[21]**

La première particularité de ces méthodes est de coder les paramètres du problème à résoudre en nombres réels. La seconde est d'effectuer une sélection déterministe des individus en ne choisissant que les *n* individus classés selon leur performance. La troisième, enfin, est d'encoder les paramètres d'évolution directement dans le génotype afin de les faire évoluer au même titre que les valeurs des paramètres solutions du problème. **[4]**

Les ES favorisent la mutation plutôt que la recombinaison. Travaillant sur des réels, la mutation suit une loi généralement gaussienne avec des écarts-types généralement codés dans le génotype.

7.1 Principes de fonctionnement :

Le fonctionnement des ES est défini comme suit :

7.1.1 Sélection :

La sélection des individus est déterministe. Deux types de sélection existent, qui sont les sélections (μ, λ) et $(\mu+\lambda)$.

- La première (μ, λ) : consiste à sélectionner les μ meilleurs parmi les λ enfants.
- La seconde $(\mu+\lambda)$: sélectionne les μ meilleurs individus parmi les μ parents de la génération précédente et les λ enfants créés (chaque parent créant λ/μ enfants avec $\lambda>\mu$).

Cette dernière méthode permet de ne pas perdre les meilleurs individus d'une génération à une autre mais accroît les possibilités que la population converge prématurément vers une solution qui ne peut pas être optimale mais qui représente un minimum local.

7.1.2 Recombinaison :

La recombinaison opère ici rarement sur le génotype contenant les variables du problème. Cependant, elle semble très utile pour l'évolution des paramètres de mutation.

7.1.3 Mutation :

Le codage étant réel, le problème se pose quant à la réalisation de la mutation. Les stratégies d'évolution proposent d'utiliser un modèle basé sur des distributions normales, avec des écarts-types qui peuvent être directement codés dans le génotype.

En considérant la représentation simplifiée (sans direction d'évolution) définie plus haut, la mutation est alors : $\forall i \in \{1, \ldots, k\}$ **indice des gènes de** $\vec{x}$ **et** $\vec{\sigma}$**:**

$$\begin{cases} \sigma i' = \sigma i \,.\, Exp\,(\tau' \,.\, N(0,1) + \tau \,.\, Ni(0,1)) \\ xi' = xi + \sigma i' \,.\, Ni(0,1) \end{cases}$$

Ni(0, 1): représente la réalisation d'une variable suivant une loi normale d'espérance 0 et d'écart type 1 calculée pour chaque indice i.

N(0, 1): une variable de même type calculée une seule fois par individu.

τ et τ': considérés comme des taux d'apprentissage.

σ: une variable aléatoire gaussienne centrée en zéro et d'écart-type σ (ajustée au cours de l'évolution).

Hans-Paul Schwefel propose pour des résultats robustes l'initialisation des paramètres τ et τ' suivante :

$$\begin{cases} \tau' = 1/SQRT\,(2K)\,; \\ \tau = 1/SQRT\,(2SQRT\,(K))\,; \end{cases}$$

8 Comparaison entre les GA et les ES : [22]

- **Les algorithmes génétiques :**

Ce sont les algorithmes évolutionnaires les plus usuels. Les génotypes évolués sont des vecteurs de valeur (souvent binaires), qui correspondent à un phénotype et dont on évalue la capacité à résoudre un problème. A chaque génération, tous les parents disparaissent et laissent place à une population totalement composée de leurs descendants.

- **Les stratégies d'évolution :**

Elles sont très similaires à un algorithme génétique, mais cette fois la population est recréée à partir des parents et des enfants. Autrement dit, la sélection se base sur un classement regroupant parents et enfants. Ces derniers ne sont sélectionnés que s'ils sont mieux adaptés

que leurs parents. Les ES utilisent directement un vecteur de nombres réels. Le croisement n'est pas utilisé dans les premières versions, mais des variantes plus élaborées inspirées des AG l'utilisent.

9 Les Algorithmes évolutionnaires et la théorie des jeux :

Dans le domaine de la théorie des jeux, l'exemple le plus connu de l'utilisation des algorithmes évolutionnaires (*EA*) pour résoudre un problème standard est le travail d'*Axelrod* sur l'émergence de la coopération dans le jeu du dilemme du prisonnier répété [1].

Axelrod (1987) utilise les *EA* pour faire évoluer une population de stratégies qui jouent au dilemme du prisonnier répété à deux joueurs contre toutes les autres stratégies dans la population. **[25]**

Dans cette approche, chaque chromosome représente l'histoire récente des choix et des observations de chaque joueur. La performance de chaque stratégie est alors évaluée dans ce jeu. L'environnement de chaque stratégie est formé par la population des autres stratégies dans la population. Comme cette population évolue dans le temps, l'environnement de chaque stratégie évolue aussi.

10 Conclusion :

Dans ce chapitre, nous avons présenté un aperçu sur deux méthodes classiques des EA. Nous en avons présenté les principes généraux afin d'en étudier les caractéristiques.

« La principale conclusion est que les EA ont déjà dépassé le cadre seul de leur applications techniques aux systèmes artificiels, ils sont prêts à entrer dans le monde réel des systèmes vivants et ils ont même déjà fait leurs premiers pas dans cette direction ».

Kosorukoff et Goldberg (2001)

[1]

ALGORITHMES ÉVOLUTIONNAIRES

Les organismes vivants résolvent superbement de nombreux problèmes, et leurs capacités d'adaptation en remontrent aux meilleurs programmes informatiques. De surcroît, les informaticiens passent des mois, voire des années, à construire des algorithmes, alors que les organismes s'élaborent spontanément grâce aux mécanismes de sélection naturelle et d'évolution.

La sélection naturelle élimine l'un des obstacles majeurs à la conception des programmes : la spécificité préalable de toutes les caractéristiques d'un problème et des tâches précises qu'un programme doit effectuer pour résoudre ce problème. En reproduisant informatiquement des mécanismes d'évolution, on « élève » aujourd'hui des programmes qui résolvent des problèmes dont personne ne comprend complètement la structure.

Les ***algorithmes évolutionnaires*** *explorent des espaces de solutions beaucoup plus vastes que les programmes classiques. En outre, l'étude de l'effet de la sélection naturelle sur les programmes, dans des conditions contrôlées et bien comprises, pourrait montrer comment la vie et l'intelligence ont évolué sur la terre.*

La plupart des organismes évoluent par deux mécanismes principaux : la ***sélection naturelle*** *et la* ***reproduction sexuée****. La sélection naturelle détermine quels membres d'une population survient et se reproduisent, la reproduction sexuée assure le brassage et la recombinaison des gènes parentaux, pour former des descendants aux potentialités nouvelles. Lorsqu'un spermatozoïde et un ovule fusionnent, leurs chromosomes s'apparient et s'échangent des segments. Ce mélange permet une évolution beaucoup plus rapide qui si les individus recevaient tous leurs gènes d'un même parent et que si les mutations étaient les seules causes de variations génétiques (bien que les organismes unicellulaire ne s'accouplent pas comme les animaux, ils échangent également du matériel génétique, et leur évolution peut se décrire de la même façon).* [14]

John Holland

CHAPITRE III

ANALYSE ET CONCEPTION

Pour mener à bien le projet, nous devons tout naturellement avoir recours à un formalisme de conception. Cette partie est consacrée aux étapes fondamentales pour le développement de notre système.

1. Introduction :

À partir de tous ce que nous avons déjà vu dans les chapitres précédents, nous pouvons maintenant faire du calcul évolutionnaire afin d'aboutir à une méthode efficace et optimale pour développer et améliorer les stratégies des jeux combinatoires. Pour appliquer cette approche, on a choisi le jeu de dames.

Pour la réalisation de notre logiciel, on a suivi la démarche illustrée dans le schéma suivant :

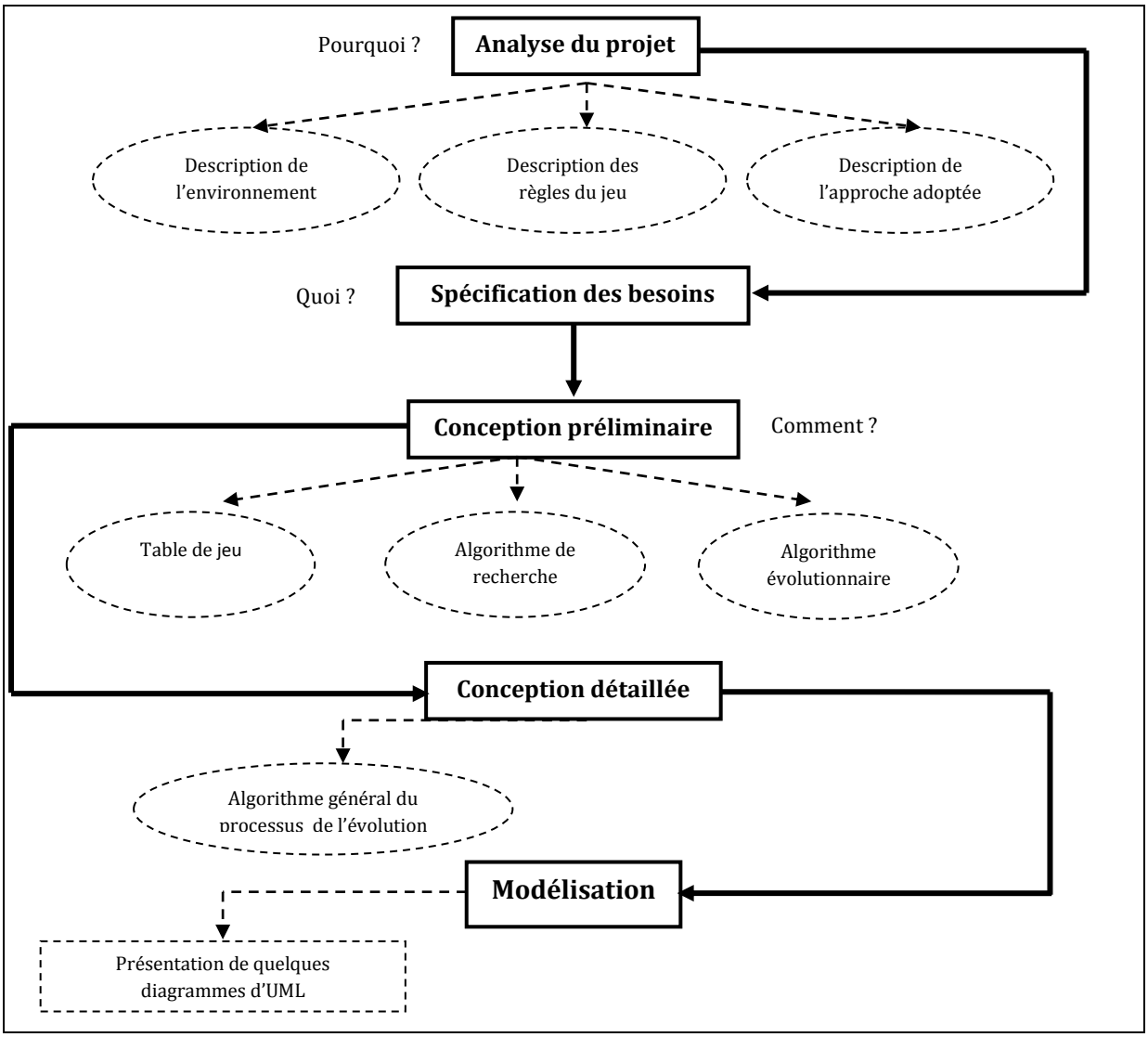

Figure 4.1 : Plan générale du chapitre de conception

2. Analyse du projet :

2.1 Description de l'environnement :

Le *système* que l'on va concevoir est un environnement de création et d'amélioration des stratégies du jeu de dames. Il pourra donc *interagir* avec un *administrateur* pour configurer et créer les stratégies employées par les différents adversaires machines ou bien tout simplement avec un *joueur* pour jouer une partie de jeu de dames contre la machine, ce qui constitue l'*environnement* lié au système.

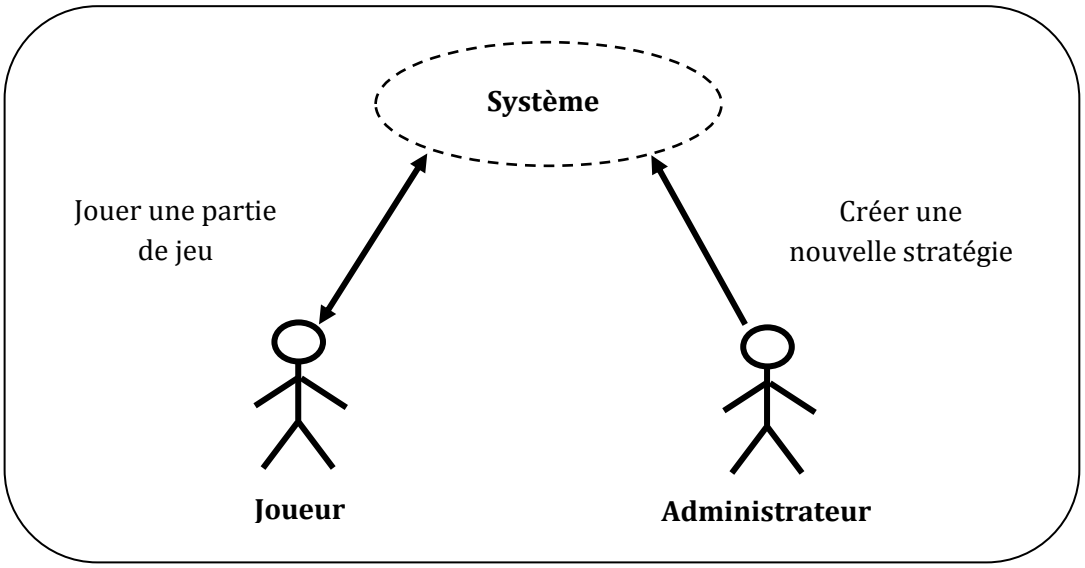

Figure 4.2 : Schéma de l'environnement.

2.2.Description des règles du jeu :

2.2.1. But du jeu :

Capturer ou immobiliser les pièces de son adversaire.

2.2.2. Matériel :

Le jeu de dame est traditionnellement joué sur une table huit par huit (8x8) avec des cases de couleurs alternatives.

Il y a deux joueurs, dénotés par les pièces noires et blanches respectivement. Chaque coté (joueur) a 12 pions qui sont placés dans les 12 premières cases alternatives de la même couleur, avec la case extrême droite sur la première rangée de chaque joueur (coté) étant

laissée ouverte, c'est-à-dire cette case appartient à l'ensemble de cases où on ne peut pas y mettre les pièces, comme le montre la figure suivante :

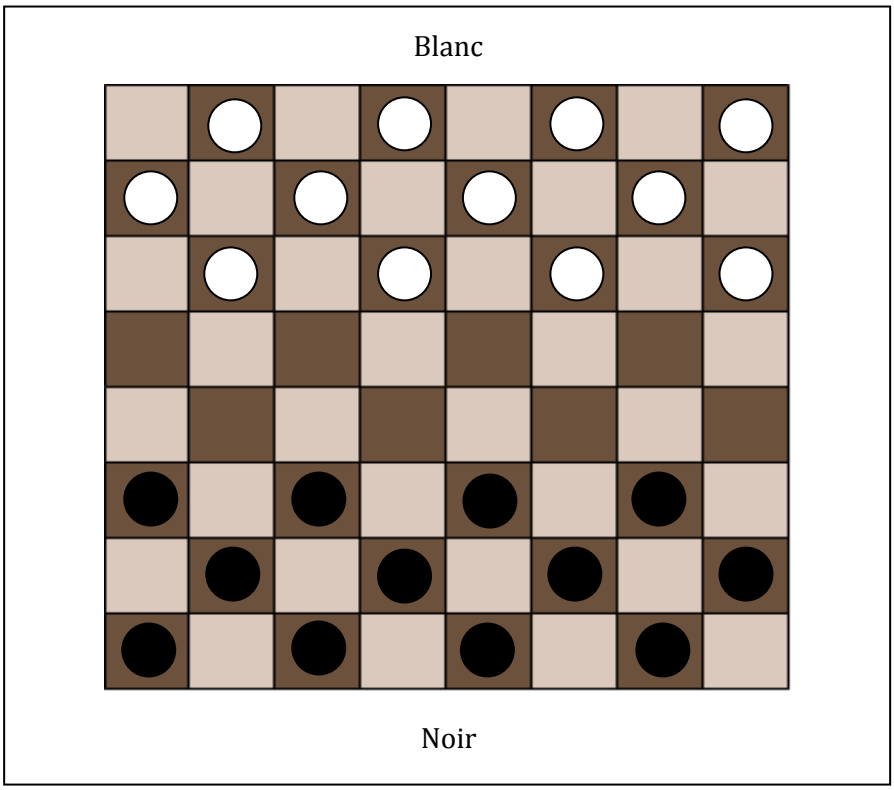

Figure 4.3 : Position initiale du jeu de dames.

2.2.3. Comment joué :

2.2.3.1. Le déplacement :

Le joueur noir se déplace d'abord, ensuite le jeu s'alterne entre les côtés. On peut faire avancer diagonalement les pièces une case à la fois.

2.2.3.2. L'enlèvement :

Lorsqu'une pièce se positionne à côté d'une pièce d'opposition (adversaire), et il y a une case vide derrière cette dernière, il faut sauter diagonalement au-dessus de la pièce d'opposition. Dans ce dernier cas, le pion d'opposition est pris et supprimé du jeu. Une prise peut s'effectuer vers l'avant ou vers l'arrière.

Si un saut placerait à leur tour la pièce sautant en une position pour un autre saut, ce saut doit également être joué, et ainsi de suite jusqu'aucun saut ne soit disponible pour cette pièce.

Toute les fois qu'un saut est disponible, il doit être joué de préférence à un mouvement qui ne saute pas, cependant quand des mouvements multiples sont disponibles, le joueur a le choix d'un saut à faire. Le mieux choix est le saut qui offre l'enlèvement de plusieurs pièces de l'adversaire (c'est-à-dire un double saut contre un saut simple).

Quand une pièce avance à la dernière rangée de la table, elle devient un **Roi**, et peut ensuite se déplacer diagonalement à n'importe quelle direction (c'est-à-dire en avant ou vers l'arrière).

2.2.4. Résultat du jeu :

Le jeu se termine quand un joueur n'a plus de mouvements disponibles, qui se produit le plus souvent en faisant enlever leur dernière pièce de la table. Mais il peut également se produire quand toutes les pièces existantes sont emprisonnées. Ceci implique la perte pour ce joueur sans mouvement restant et une victoire pour l'adversaire (l'objectif du jeu). Le jeu peut également se terminer lorsque la partie est nulle.

3. Spécification des besoins :

C'est une étape primordiale au début de chaque démarche de développement. Son but est de veiller à développer un logiciel adéquat, sa finalité est la description générale des fonctionnalités du système, en répondant à la question : Quelles sont les fonctions du système ?

Les fonctions principales que le logiciel devra satisfaire sont les suivantes :

- Permettre à un utilisateur de créer une nouvelle stratégie de jeu en passant par les étapes qu'on va discuter dans la suite de ce chapitre.
- Permettre à un utilisateur de jouer une partie de jeu de dames contre l'ordinateur, c'est-à-dire contre la meilleure stratégie sélectionnée expérimentalement.

Nous avons identifié les problèmes suivants :

- L'explosion combinatoire qui reste un défi pour les gens intéressés de la théorie des jeux.
- La difficulté de déterminer une fonction d'évaluation d'une situation possible dans le jeu.
- La difficulté de trouver une meilleure représentation des stratégies du jeu.

4. Conception préliminaire :

La conception préliminaire permet de déterminer l'architecture informatique globale du système.

4.1 Table du jeu :

La classe *Board* représente le plateau de jeu et contient toutes les fonctionnalités nécessaires pour gérer le plateau, les pièces et les mouvements.

La méthode *__init__* initialise le plateau avec les pièces et les compteurs.

La méthode *draw_squares* dessine les cases du plateau.

La méthode *evaluate* calcule une évaluation du plateau.
La méthode *get_all_pieces* retourne toutes les pièces d'une couleur donnée.
La méthode *move* effectue un mouvement d'une pièce à une position donnée.
La méthode *get_piece* retourne la pièce située à une position donnée.
La méthode *create_board* crée le plateau avec les pièces initialisées.
La méthode *draw* dessine le plateau et les pièces sur la fenêtre de jeu.
La méthode *remove* enlève les pièces spécifiées du plateau.
La méthode *winner* détermine le gagnant du jeu.
La méthode *get_valid_moves* retourne tous les mouvements valides pour une pièce donnée.
Les méthodes privées *_traverse_left* et *_traverse_right* sont utilisées pour calculer les mouvements valides en parcourant les cases vers la gauche et vers la droite respectivement.
La classe *Piece* représente une pièce sur le plateau.
La méthode *__init__* initialise les attributs de la pièce.
La méthode *make_king* transforme la pièce en dame.
La méthode *move* met à jour la position de la pièce.
La méthode *draw* dessine la pièce sur la fenêtre de jeu.

Le code utilise également des constantes définies dans le module constants. Ces constantes spécifient les couleurs, les dimensions du plateau, la taille des cases, etc.

4.2. Algorithme évolutionnaire :

Dans le chapitre 02 «*Les algorithmes évolutionnaires : principes et méthodes* », nous avons présenté deux méthodes classiques des EA, on a étudié leurs caractéristiques afin de choisir une méthode pour la réalisation de notre projet, notre choix c'est tombé sur les *Stratégies d'évolution*.

La représentation d'un réseau connexionniste dans un génotype a suscité un certain nombre de travaux. A un extrême, le réseau est codé littéralement dans le génotype, dont chaque poids est codé d'une manière précise. Dans ce cas, les Stratégies d'évolution se réduisent à un problème d'optimisation multicritères standard.

4.3. Algorithme de recherche (MiniMax):

Le principe des algorithmes de recherche dans les jeux est d'évaluer les différents coups à jouer pour un joueur donné, et de retourner le meilleur. Ceci implique de connaître l'ensemble des coups jouables par le joueur en construisant ce qu'on appelle un arbre de jeu.

Dans cette optique, et pour l'accomplissement de notre projet, nous avons utilisé l'algorithme de recherche qui est décrit dans le premier chapitre « *Introduction à la théorie des jeux* », à savoir le MiniMax

5. Combinaison de l'algorithme Minimax avec l'algorithme génétique :

Combiner l'algorithme minimax avec un algorithme génétique (GA) dans un jeu de dames peut être réalisé grâce à une approche hybride qui intègre les points forts des deux techniques. Voici un aperçu général de la façon dont vous pouvez les combiner :

1. Représentation : Définissez une représentation appropriée de l'état du jeu dans le contexte de l'algorithme minimax et du GA. Cette représentation devrait prendre en compte la configuration actuelle du plateau, le tour du joueur et toute autre information pertinente.

2. Algorithme minimax : Implémentez l'algorithme minimax pour évaluer les états du jeu et effectuer des coups optimaux. L'algorithme minimax explore l'arbre de jeu en analysant

de manière récursive les coups possibles et leurs conséquences. Il attribue une valeur à chaque état du jeu, représentant le degré de désirabilité pour le joueur actuel. Cette valeur est utilisée pour déterminer le meilleur coup.

3. Algorithme génétique : Utilisez un GA pour faire évoluer une population de stratégies de jeu. Représentez chaque stratégie sous la forme d'un ensemble de paramètres ou de caractéristiques qui influencent le processus de prise de décision. Appliquez des opérateurs génétiques tels que la sélection, le croisement et la mutation pour générer de nouvelles stratégies à chaque génération.

4. Fonction d'évaluation : Définissez une fonction d'évaluation qui peut évaluer la qualité d'une stratégie de jeu. Cette fonction devrait prendre en compte des facteurs tels que la position sur le plateau, le nombre de pièces et les pièces de dame. Utilisez cette fonction pour évaluer la performance de chaque stratégie dans le GA.

5. Algorithme génétique + minimax : Intégrez la fonction d'évaluation dans le calcul de fitness du GA. Après chaque génération, sélectionnez les stratégies les plus performantes et appliquez l'algorithme minimax pour évaluer leur performance contre un ensemble fixe d'adversaires. Utilisez les résultats du minimax comme mesure de fitness supplémentaire lors de la sélection et de l'évolution du GA.

6. Processus évolutif : Itérez le processus du GA, permettant aux stratégies d'évoluer sur plusieurs générations. Suivez les performances des stratégies et ajustez les paramètres du GA, tels que la taille de la population, la pression de sélection et le taux de mutation, pour obtenir de meilleurs résultats.

En combinant l'algorithme minimax avec un GA, on peut améliorer les capacités de recherche et de prise de décision de l'IA jouant au jeu de dames, ce qui conduit à un gameplay amélioré et à des choix stratégiques plus pertinents.

6. Conception détaillée :

Cette partie traite de l'application des algorithmes évolutionnaires sur l'algorithme MiniMax pour la création de nouvelles stratégies de jeu.

6.1 Algorithme général du processus d'évolution :

L'algorithme qui génère le processus illustré dans les sections précédentes est donné par le code suivant :

```
RED = (255,0,0)
WHITE = (255, 255, 255)
def minimax(position, depth, max_player, game):
    if depth == 0 or position.winner() != None:
      return position.evaluate(), position
       if max_player:
        maxEval = float('-inf')
        best_move = None
        for move in get_all_moves(position, WHITE, game):
            evaluation = minimax(move, depth-1, False, game)[0]
            maxEval = max(maxEval, evaluation)
            if maxEval == evaluation:
                best_move = move
           return maxEval, best_move
    else:
        minEval = float('inf')
        best_move = None
        for move in get_all_moves(position, RED, game):
         evaluation = minimax(move, depth-1, True, game)[0]
            minEval = min(minEval, evaluation)
            if minEval == evaluation:
                best_move = move
                return minEval, best_move
def simulate_move(piece, move, board, game, skip):
    board.move(piece, move[0], move[1])
    if skip:
      board.remove(skip)
    return board
def get_all_moves(board, color, game):
    moves = []
   for piece in board.get_all_pieces(color):
        valid_moves = board.get_valid_moves(piece)
        for move, skip in valid_moves.items():
            draw_moves(game, board, piece)
            temp_board = deepcopy(board)
            temp_piece = temp_board.get_piece(piece.row, piece.col)
            new_board = simulate_move(temp_piece, move, temp_board,
game, skip)
            moves.append(new_board)
        return moves
def draw_moves(game, board, piece):
    valid_moves = board.get_valid_moves(piece)
    board.draw(game.win)
```

```
  pygame.draw.circle(game.win, (0,255,0), (piece.x, piece.y), 50, 5)
game.draw_valid_moves(valid_moves.keys())
    pygame.display.update()
   #pygame.time.delay(100)
```

Algorithme 4.1 : Algorithme général de l'application.

*La ligne draw_moves(game, board, piece) est mal indentée. Assurez-vous que cette ligne est correctement indentée pour qu'elle soit à l'intérieur de la boucle for move, skip in valid_moves.items():.

*La ligne return moves à la fin de la fonction get_all_moves(board, color, game) est mal indentée. Assurez-vous que cette ligne est correctement indentée pour qu'elle soit en dehors de la boucle for piece in board.get_all_pieces(color):.

*La ligne pygame.draw.circle(game.win, (0,255,0), (piece.x, piece.y), 50, 5) est mal indentée et semble être en dehors de la fonction draw_moves(game, board, piece). Assurez-vous que cette ligne est correctement indentée pour qu'elle soit à l'intérieur de la fonction draw_moves(game, board, piece).

*Il n'est pas clair d'où proviennent les valeurs de RED, WHITE, position, et game. Assurez-vous que ces variables sont définies et initialisées correctement avant d'appeler la fonctionminimax().

*Une fois que ces problèmes seront corrigés et que les dépendances nécessaires seront importées (comme pygame), l'algorithme devrait être en mesure de fournir un résultat, qui serait une évaluation du meilleur mouvement possible dans le jeu à une certaine profondeur.

7. Modélisation :

La motivation fondamentale de la modélisation est de fournir une démarche antérieure afin de réduire la complexité du système étudié lors de la conception et d'organiser la réalisation du projet en définissant les modules et les étapes de la réalisation. Plusieurs démarches de modélisation sont utilisées. Nous adoptons dans notre travail une approche objet basée sur un outil de modélisation *UML*.

7.1. Présentation de l'UML :

Le langage *UML* **«** *Unified Modeling Language* » est un langage de modélisation qui a pour but de faciliter les transitions, lors du développement d'un projet. Il permet de structurer un projet et de le matérialiser graphiquement sous forme de diagrammes compréhensibles par les non informaticiens. Aucune connaissance de langage informatique n'est pré-requise.

Cette modélisation permet dans un second temps de développer le code informatique, le plus souvent à l'aide d'un langage orienté objet. La description de projets en UML est une étape nécessaire qui permet de gagner beaucoup de temps dans le développement d'une application car la mise au point du code en est moins fastidieuse et le risque d'erreurs de conception ou de réalisation est plus limité. Bien que conçue pour la gestion de projets de grande envergure, l'utilisation de cette méthodologie est bénéfique même pour les projets les plus modestes.

7.2. Historique : [22]

En 1994, naissance de l'UML chez Rational Software Corporation à l'initiative de G. Booch et de J. Rumbaugh, il est le fruit d'un travail d'unification et de standardisation de trois méthodes de modélisation dominantes développées dans les années 90 : OMT, Booch et OOSE.

En 1997, UML 1.1 a été standardisé par l'OMG (Object Management Group), suite à la demande émanant de la collaboration de plusieurs entreprises (Hewlett-Packard, IBM, i-Logix, ICON Computing, IntelliCorp, MCI Systemhouse, Microsoft, ObjecTime, Oracle, Platinum Technology, Ptech, Rational Software Corporation, Reich Technologies, Softeam, Sterling Software, Taskon et Unisys).

Depuis 1999, la version actuelle est UML **1.3** (la version 1.4 sera bientôt prête, afin de préparer la prochaine version 2.0).

7.3. Objectifs de l'UML : [23]

Au final, le langage UML est une synthèse de tous les concepts et les formalismes méthodologiques les plus utilisés, pouvant être utilisé, grâce à sa simplicité et ça son universalité, comme langage de modélisation pour la plupart des systèmes ils nécessiteraient le développement.

Le langage UML permet ainsi d'apporter des solutions lors du développement des systèmes informatisé :

- Décomposer le processus de développement en distinguant la phase d'analyse (aspects fonctionnels) de la phase de réalisation (aspects technologiques et architecturaux).
- Décomposer le système en sous-systèmes plus facilement abordables : réduction de la complexité, répartition du travail, réutilisation des sous-systèmes.
- Utiliser une technologie de haut niveau proche de la réalité pour aborder le développement.

7.4. Les différentes vues d'UML : [24]

La modélisation proposée par le langage UML se réalise principalement sous forme graphique, en usant de divers types de diagrammes spécifiques, répartis en trois groupes :

- **Vue fonctionnelle :**

Interactive, qui est représentée à l'aide de *diagrammes de cas d'utilisation*, *diagrammes des séquences, et* les *diagrammes de collaboration*. Elle cherche à appréhender les interactions entre les différents acteurs/utilisateurs et le système, sous forme d'objectif à atteindre d'un côté et sous forme chronologique de scénarios d'interaction typiques de l'autre.

- **Vue structurelle :**

Appelée aussi *statique*, réunit les *diagrammes de classes* et les *diagrammes de packages*. Les premiers favorisent la structuration des données et tentent d'identifier les objets/composants constituant le programme, leurs attributs, opérations et méthodes, ainsi que les liens ou associations qui les unissent. Les seconds s'attachent à regrouper les classes fortement liées entre elles en des composants les plus autonomes possibles. A l'intérieur de chaque package, on trouve un diagramme de classes.

- **Vue dynamique :**

Qui est exprimée par les *diagrammes d'états*. Cette vue est plus algorithmique et orientée « traitement», elle vise à décrire l'évolution (la dynamique) des objets complexes du

programme tout au long de leur cycle de vie. De leur naissance à leur mort, les objets voient leurs changements d'états guidés par les interactions avec les autres objets. Le *diagramme d'activité* est une sorte d'organigramme correspondant à une version simplifiée du diagramme d'états. Il permet de modéliser des activités qui se déroulent en parallèle les unes des autres, quand ce parallélisme peut poser problème. En général, les diagrammes d'états à eux seuls ne permettent pas de faire apparaître les problèmes spécifiques posés par la synchronisation des processus en concurrence, pour assurer la cohérence du comportement et l'absence d'inter-blocage. Etablir un diagramme d'activité peut aider à mettre au point un diagramme d'états.

Certains de ces diagrammes sont indépendants, alors que d'autre servent de base de travail ou bien sont la continuité d'autres diagrammes.

Afin de développer notre application, on s'intéresse aux diagrammes suivants :

- **Diagramme de cas d'utilisation :**

Un cas d'utilisation (use case) modélise une interaction entre le système informatique à développer et un utilisateur ou acteur interagissant avec le système. Plus précisément, un cas d'utilisation décrit une séquence d'actions réalisées par le système qui produit un résultat observable pour un acteur.

- **Diagramme des classes :**

Un diagramme des classes décrit le type des objets ou données du système ainsi que les différentes formes de relation statiques qui les relient entre eux. Le diagramme de classes qui est unique, se construit en partie à l'aide des informations issues des différents de séquence. Il permet d'obtenir le squelette du code par génération automatique de code ; il s'agit donc de la dernière étape d'analyse juste avant le codage proprement dit.

- **Diagramme de séquence :**

Les diagrammes de séquences mettent en valeur les échanges de messages (déclenchant des événements) entre acteurs et objets (ou entre objets et objets) de manière chronologique, l'évolution du temps se lisant de haut en bas. Un diagramme de séquences est un moyen semi-formel de capturer le comportement de tous les objets et acteurs impliqués dans un cas d'utilisation. On peut indiquer un type de message particulier : les retours de fonction qui, bien entendu, ne concernent aucun message mais signifient la fin de l'appel de l'objet appelé. Ils permettent d'indiquer la libération de l'objet appelant (ou de l'acteur). Un emploi abusif de retours de fonction peut alourdir considérablement le diagramme, aussi un usage parcimonieux est-il conseillé.

7.5. Présentation des diagrammes :

Dans la section suivante, nous allons identifier les trois diagrammes illustrés précédemment afin de mettre en œuvre l'approche choisie pour le développement de notre système :

- **Joueur :**

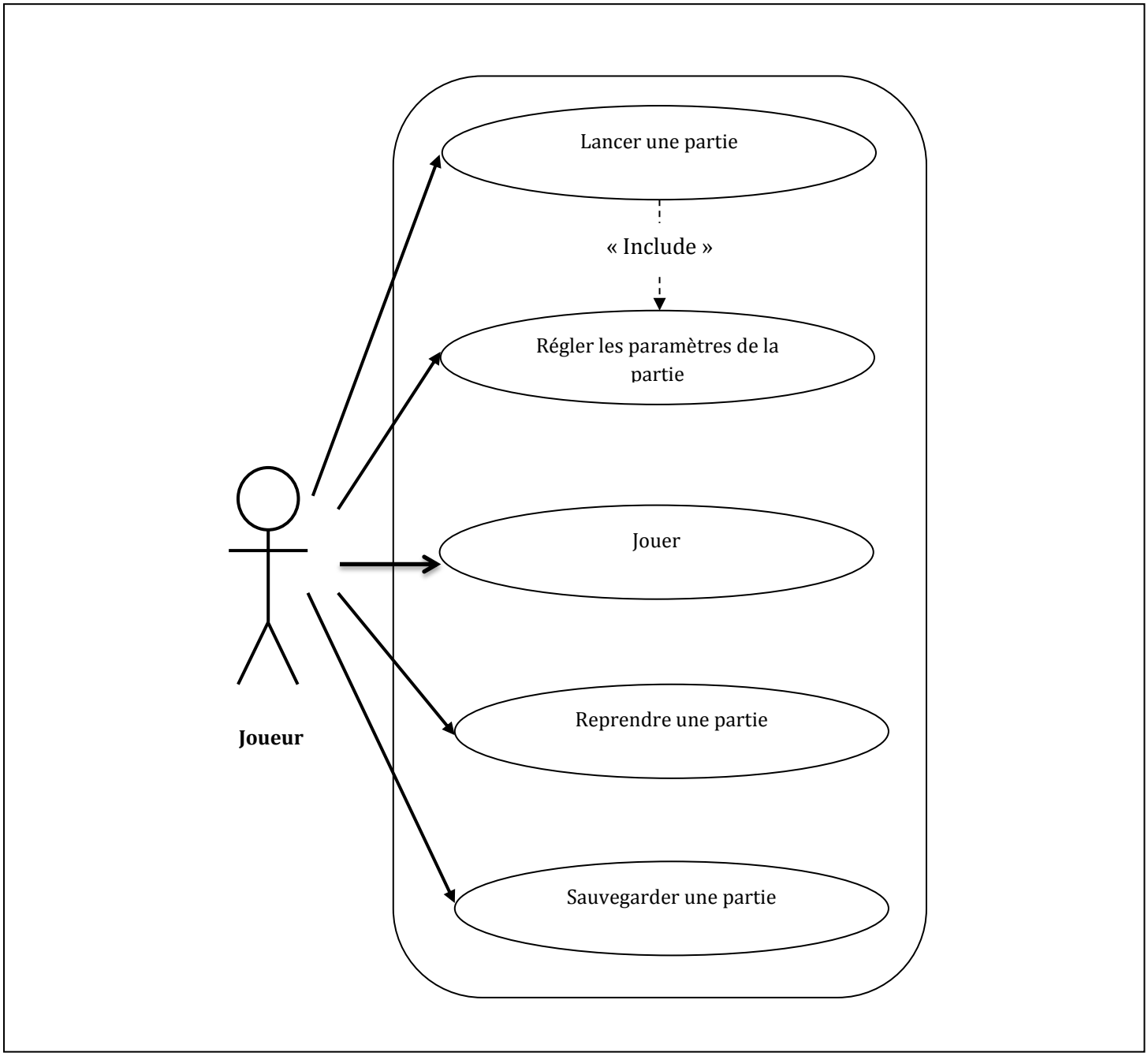

Figure 4.11 : Diagramme de cas d'utilisation « Joueur».

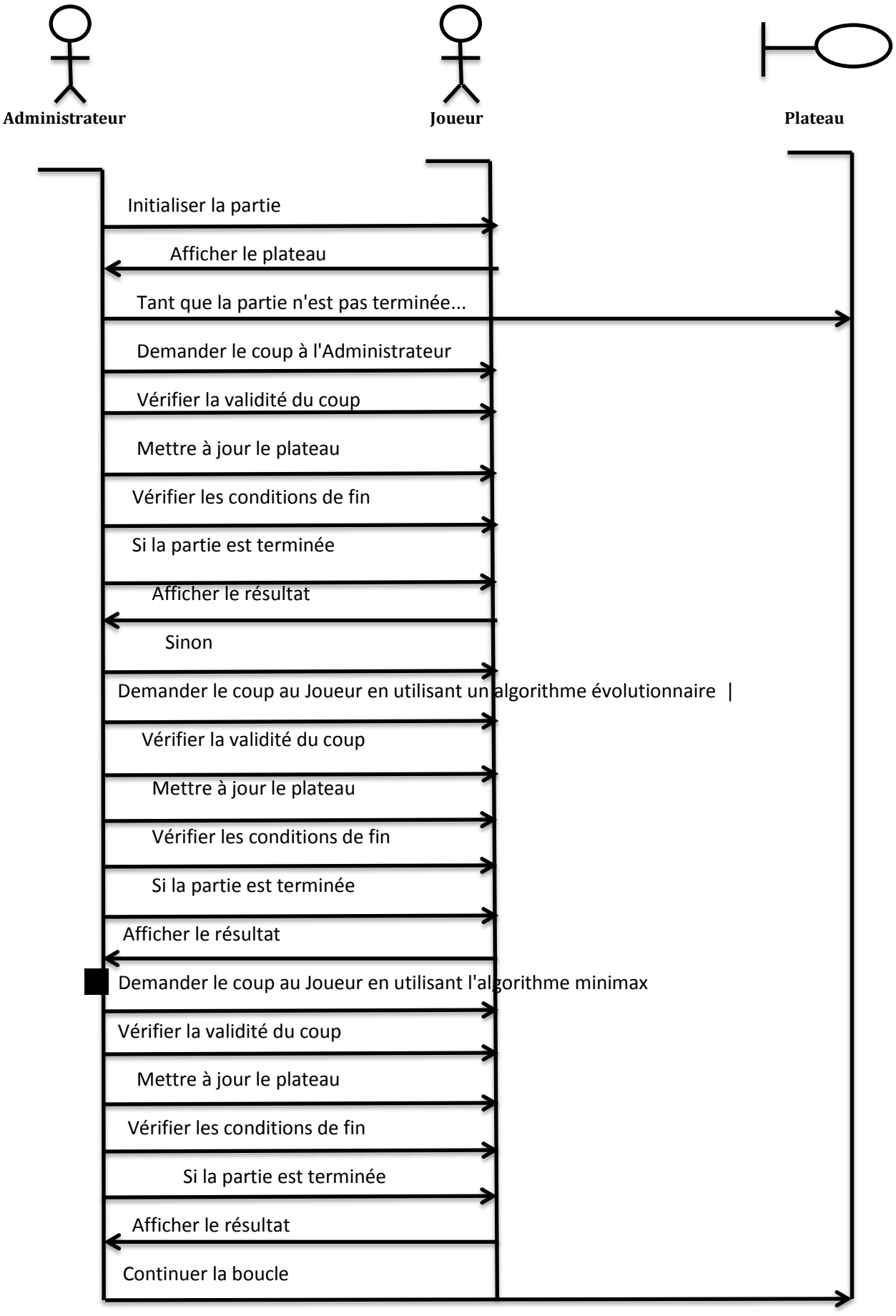

Figure 4.12 : Diagramme de séquence

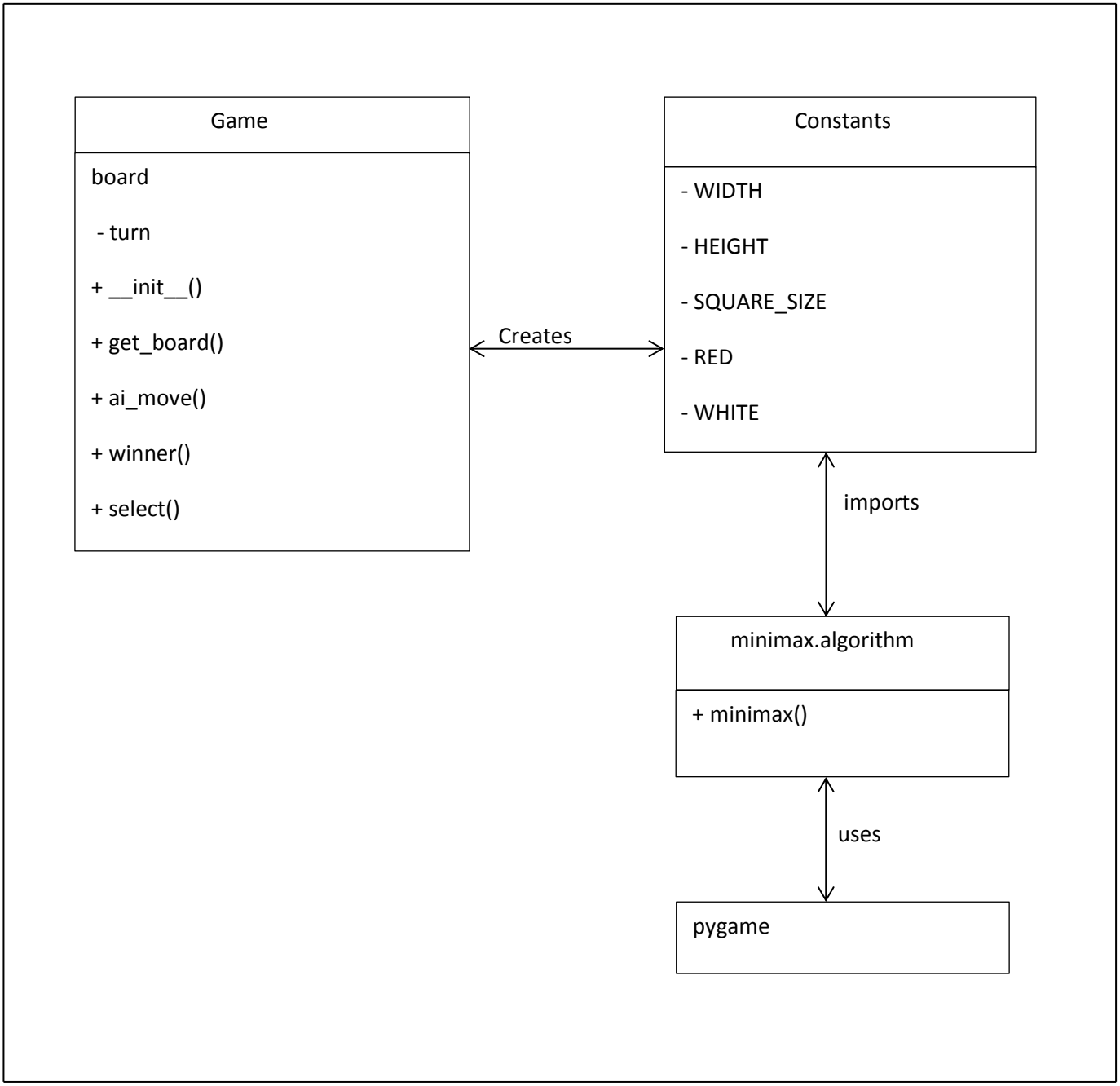

Figure 4.13 : **Diagramme de classes : UML**

8.Conclusion :

Combiner les différents types de diagrammes offrent une vue complète des aspects statiques et dynamiques des systèmes. Comme nous pouvons le constater, l'activité de la conception a facilité la compréhension de notre système, qui ébauche vers l'activité d'implémentation.

Dans le chapitre suivant nous allons présenter l'implémentation et la réalisation de notre application. Pour montrer l'efficacité de l'approche illustrée dans les sections précédentes, on a besoin de réaliser un logiciel sous forme d'un prototype pédagogique qui permet d'évoluer un jeu combinatoire (jeu de dames) selon les points abordés dans ce présent chapitre.

CHAPITRE IV

IMPLÉMENTATION

Après avoir détaillé la conception de notre application, nous allons présenter dans ce chapitre la partie de réalisation ainsi que les différents outils pour le développement du logiciel.

1. Introduction :

Dans ce chapitre, nous présentons l'environnement sur lequel nous avons développé notre application, les différents outils utilisés ainsi que les composantes applicatives réalisées. Enfin nous présentons les principales interfaces et fenêtres de l'application.

2. Présentation de l'application :

Notre application a pour objectifs d'évaluer des stratégies de jeu combinatoire (jeu de dames) on se basant sur l'idée combinaison des mini max avec les algorithmes évolutionnaires. La programmation de ces stratégies se fait par le déroulement d'un processus évolutionnaire qui contient au début un ensemble de stratégies comme l'analysant de manière récursive les coups possibles et leurs conséquences. Cette application permet de créer de nouvelles générations, de les importer et de les enregistrer, parce que le parcours d'une seule génération devient très long.

3. Outils de développement :

3.1. Environnement matériel de développement :

Afin de réaliser notre projet, nous avons utilisé un pc portable ayant les caractéristiques suivantes :

- Fabriquant : Lenovo
- Processeur : Intel ® Core (TM) CPU B970 @ 1,90 GHz 1,90 GHz
- Mémoire installée (RAM) : 4,00 Go
- Type du système : Système d'exploitation 46 bits, Processeur *64.
- Windows 8.1

3.2. Environnement logiciel de développement :

Après avoir passé par l'étape de l'analyse et conception, il nous reste de choisir une plateforme de développement appropriée pour mettre en place tous les efforts qu'on a fournit tout au long de ce thème. Pour la mise en œuvre de ce prototype, nous avons choisi l'environnement de développement PYTHON .

Python est un langage de programmation polyvalent, interprété et de haut niveau, apprécié pour sa lisibilité et sa simplicité syntaxique. Il est largement utilisé pour le développement de

logiciels, l'automatisation de tâches, le traitement des données, l'apprentissage automatique, l'analyse de données, la création de sites web et bien d'autres applications. les instructions et

fonctions de ce langage sont exécutées par un interpréteur Python qui convertit le code source en instructions exécutables compréhensibles par l'ordinateur. L'écosystème Python propose également une large gamme de bibliothèques et de modules prêts à l'emploi, ce qui facilite le développement de logiciels en fournissant des fonctionnalités préconstruites pour diverses tâches.

4. **Présentation de quelques méthodes :**

La création de notre application a nécessité le passage par plusieurs méthodes, nous allons présenter les plus importantes :

Création d'un nouveau fichier Python nommé "main.py" dans le dossier du tutoriel du jeu de dames

Importation du module PI (Package/Module) du jeu de dames

```
import mon_package_dames.PI as PI
# Vérification du bon fonctionnement de l'importation
print("Bonjour, de haut niveau")
PI.game()  # Exemple d'appel à une fonction du module PI

# Construction d'une interface de programmation d'application (API)
autour du jeu de dames
#  (Détails de l'API à ajouter au fur et à mesure de l'avancement)

# Création du dossier "dames" pour le package
# Note : Ce bloc de code peut être placé à un niveau supérieur selon
votre structure de dossiers
# pour permettre l'importation depuis le package
import os
if not os.path.exists("dames"):
    os.makedirs("dames")

# Création du fichier "__init__.py" pour initialiser le package
with open("dames/__init__.py", "w") as init_file:
    pass

# Importation du module PI à partir du package
import dames._._knit._._PI as PI

# Suite du code (ajoutez vos propres fonctionnalités ou API ici)

# Importation des modules nécessaires
import pygame
from pygame.locals import *

# Initialisation de Pygame
pygame.init()
```

*Dans cette implémentation, j'ai ajouté une importation du fichier "constantes.py", qui contient les valeurs constantes telles que "largeur" et "hauteur". Ces constantes sont utilisées pour définir la taille de la fenêtre d'affichage. Vous pouvez définir ces constantes dans le fichier "constantes.py" comme suit :

```
# constantes.py

# Dimensions de la fenêtre d'affichage
largeur = 800
hauteur = 800

# Nombre de lignes et de colonnes sur l'échiquier
nombre_lignes = 8
nombre_colonnes = 8

# Taille d'un carré de l'échiquier
taille_case = largeur // nombre_colonnes
```

Veuillez noter que la fonction draw_circle doit être implémentée séparément pour effectuer le dessin réel. J'ai ajouté des commentaires pour indiquer où cette fonction doit être implémentée. Assurez-vous de la personnaliser en fonction de votre environnement de développement ou de votre plateforme de dessin.

```
class Piece:
    def __init__(self, couleur, ligne, colonne, taille_carré):
        self.couleur = couleur
        self.ligne = ligne
        self.colonne = colonne
        self.taille_carré = taille_carré
        self.point_r = 255
        self.point_b = 255

    def est_negatif(self):
        if self.couleur == self.point_r:
            self.direction = "négative"
        else:
            self.direction = "positive"

    def calcule_position(self):
        self.point_x = self.taille_carré * self.ligne
        self.point_y = self.taille_carré * self.colonne

  def dessine(self):
```

```
        self.calcule_position()

        # Dessiner le cercle
        draw_circle(self.couleur, self.point_x, self.point_y,
self.taille_carré)

        # Dessiner le contour
        contour = 2
        draw_circle(self.couleur, self.point_x, self.point_y,
self.taille_carré + contour)

        # Dessiner la couleur de la pièce
        couleur_interieure = (28, 28, 28)  # Gris
        draw_circle(couleur_interieure, self.point_x, self.point_y,
self.taille_carré)

    def draw_circle(self, couleur, x, y, rayon):
        # Implémentation de la fonction de dessin du cercle
        Pass
```

Algorithme MiniMax :

```
RED = (255,0,0)

WHITE = (255, 255, 255)

def minimax(position, depth, max_player, game):

    if depth == 0 or position.winner() != None:

      return position.evaluate(), position

       if max_player:

        maxEval = float('-inf')

        best_move = None

        for move in get_all_moves(position, WHITE, game):

            evaluation = minimax(move, depth-1, False, game)[0]

            maxEval = max(maxEval, evaluation)

            if maxEval == evaluation:

                best_move = move

           return maxEval, best_move

    else:

        minEval = float('inf')
```

```
        best_move = None
        for move in get_all_moves(position, RED, game):
         evaluation = minimax(move, depth-1, True, game)[0]
            minEval = min(minEval, evaluation)
            if minEval == evaluation:
                best_move = move
                return minEval, best_move
def simulate_move(piece, move, board, game, skip):
    board.move(piece, move[0], move[1])
    if skip:
      board.remove(skip)
    return board
def get_all_moves(board, color, game):
    moves = []
   for piece in board.get_all_pieces(color):
        valid_moves = board.get_valid_moves(piece)
        for move, skip in valid_moves.items():
            draw_moves(game, board, piece)
            temp_board = deepcopy(board)
            temp_piece = temp_board.get_piece(piece.row, piece.col)
            new_board = simulate_move(temp_piece, move, temp_board,
game, skip)
            moves.append(new_board)
        return moves
def draw_moves(game, board, piece):
    valid_moves = board.get_valid_moves(piece)
    board.draw(game.win)
  pygame.draw.circle(game.win, (0,255,0), (piece.x, piece.y), 50, 5)
game.draw_valid_moves(valid_moves.keys())
    pygame.display.update()
```

```
    #pygame.time.delay(100)
```

Cet algorithme implémente l'algorithme du minimax avec élagage alpha-bêta pour une certaine forme de jeu. Voici une explication détaillée de chaque partie de l'algorithme :

1. `from copy import deepcopy` : Cette instruction importe la fonction `deepcopy` du module `copy`. Elle est utilisée pour effectuer des copies profondes des objets, ce qui est important pour éviter les modifications indésirables lors de la simulation des mouvements dans le jeu.

2. `RED = (255,0,0)` et `WHITE = (255, 255, 255)` : Ces deux lignes définissent les couleurs utilisées dans le jeu. Ici, le rouge (RED) est défini comme (255,0,0) et le blanc (WHITE) est défini comme (255, 255, 255).

3. `minimax(position, depth, max_player, game)` : C'est la fonction principale de l'algorithme du minimax. Elle prend en paramètres la position actuelle dans le jeu, la profondeur actuelle de l'arbre de recherche, un drapeau indiquant si c'est au tour du joueur maximisant ou du joueur minimisant, et l'objet `game` qui représente le jeu lui-même.

- Si `depth == 0` ou si la méthode `winner()` de l'objet `position` renvoie une valeur différente de `None`, cela signifie que la profondeur maximale a été atteinte ou qu'un joueur a gagné. Dans ce cas, la fonction renvoie l'évaluation de la position actuelle et la position elle-même.

- Si `max_player` est `True`, cela signifie que c'est au tour du joueur maximisant. La fonction effectue alors une boucle sur tous les mouvements possibles pour le joueur maximisant dans la position actuelle. Pour chaque mouvement, elle appelle récursivement la fonction `minimax` avec le drapeau `max_player` inversé et une profondeur réduite de 1. Elle met à jour `maxEval` avec la valeur maximale entre `maxEval` et l'évaluation retournée par la fonction récursive. Si la valeur de `maxEval` est mise à jour, cela signifie qu'un meilleur mouvement a été trouvé, et donc `best_move` est mis à jour avec ce mouvement. Enfin, la fonction renvoie `maxEval` et `best_move`.

- Si `max_player` est `False`, cela signifie que c'est au tour du joueur minimisant. Le processus est similaire à celui décrit ci-dessus pour le joueur maximisant, mais avec des valeurs inversées. La fonction effectue une boucle sur tous les mouvements possibles pour le joueur minimisant dans la position actuelle. Pour chaque mouvement, elle appelle récursivement la fonction `minimax` avec le drapeau `max_player` inversé et une profondeur réduite de 1. Elle met à jour `minEval` avec la valeur minimale entre `minEval` et l'évaluation retournée par la fonction récursive. Si la valeur de `minEval` est mise à jour, cela signifie qu'un meilleur mouvement a été trouvé, et donc `best_move` est mis à jour avec ce mouvement. Enfin, la fonction renvoie `minEval` et `best_move`.

4. `simulate_move(piece, move, board…)`

-

L'algorithme général de l'application :

```
import pygame

from checkers.constants import WIDTH, HEIGHT, SQUARE_SIZE, RED,
WHITE

from checkers.game import Game

from minimax.algorithm import minimax

FPS = 60

WIN = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption('Checkers')

def get_row_col_from_mouse(pos):

    x, y = pos

    row = y // SQUARE_SIZE

    col = x // SQUARE_SIZE

    return row, col

def main():

    run = True

    clock = pygame.time.Clock()

    game = Game(WIN)

    while run:

        clock.tick(FPS)

                if game.turn == WHITE:

            value, new_board = minimax(game.get_board(), 4, WHITE,
game)

            game.ai_move(new_board)

         if game.winner() != None:

            print(game.winner())

            run = False

       for event in pygame.event.get():

            if event.type == pygame.QUIT:
```

```
                    run = False

                            if event.type == pygame.MOUSEBUTTONDOWN:

                    pos = pygame.mouse.get_pos()

                    row, col = get_row_col_from_mouse(pos)

                    game.select(row, col)

            game.update()

            pygame.quit()


main()
```

Ce code est un programme de jeu de dames utilisant la bibliothèque Pygame. Voici un aperçu détaillé du fonctionnement du code :

1. Les modules nécessaires sont importés : pygame, constants du package checkers, et minimax de minimax.algorithm.

2. Les constantes du jeu sont définies, telles que la largeur et la hauteur de la fenêtre, la taille d'une case, et les couleurs rouge (RED) et blanc (WHITE).

3. La fenêtre de jeu est créée avec les dimensions spécifiées.

4. La fonction get_row_col_from_mouse est définie pour convertir les coordonnées de la souris en indices de ligne et de colonne sur le plateau de jeu.

5. La fonction principale main est définie. Elle contient une boucle principale qui s'exécute tant que la variable run est True.

6. À l'intérieur de la boucle principale, la fréquence d'images est fixée à 60 images par seconde en utilisant un objet Clock de Pygame.

7. Si c'est le tour du joueur blanc (WHITE), la fonction minimax est appelée pour obtenir le meilleur mouvement pour l'ordinateur. La profondeur de recherche est fixée à 4, la couleur du joueur maximisant est WHITE, et l'objet game est passé en tant que paramètre. La fonction minimax renvoie une valeur d'évaluation et un nouveau plateau de jeu. Ensuite, la méthode ai_move de l'objet game est appelée pour effectuer le mouvement de l'ordinateur.

8. La condition if game.winner() != None vérifie si un joueur a gagné la partie. Si c'est le cas, le nom du gagnant est affiché à la console et la boucle principale est arrêtée en affectant False à la variable run.

9. Une boucle parcourt tous les événements pygame. Si l'événement est de type QUIT, cela signifie que l'utilisateur a demandé à quitter le jeu, donc la variable run est mise à False pour arrêter la boucle principale.

10. Si l'événement est de type MOUSEBUTTONDOWN, cela signifie que l'utilisateur a cliqué sur la souris. Les coordonnées du clic sont obtenues à l'aide de la méthode get_pos de pygame, puis converties en indices de ligne et de colonne à l'aide de la fonction get_row_col_from_mouse. Ensuite, la méthode select de l'objet game est appelée pour gérer la sélection d'une pièce sur le plateau de jeu.

11. La méthode update de l'objet game est appelée pour mettre à jour l'affichage du jeu.

12. Enfin, une fois que la boucle principale est terminée, Pygame est fermé en appelant la méthode quit.

13. La fonction main est exécutée pour démarrer le jeu.

En résumé, ce code met en place une interface de jeu de dames utilisant Pygame. Il gère les événements utilisateur, tels que les clics de souris, et utilise l'algorithme minimax pour permettre à l'ordinateur de jouer contre l'utilisateur.

5. Création des interfaces :

Voici l'enchaînement de quelques interfaces :

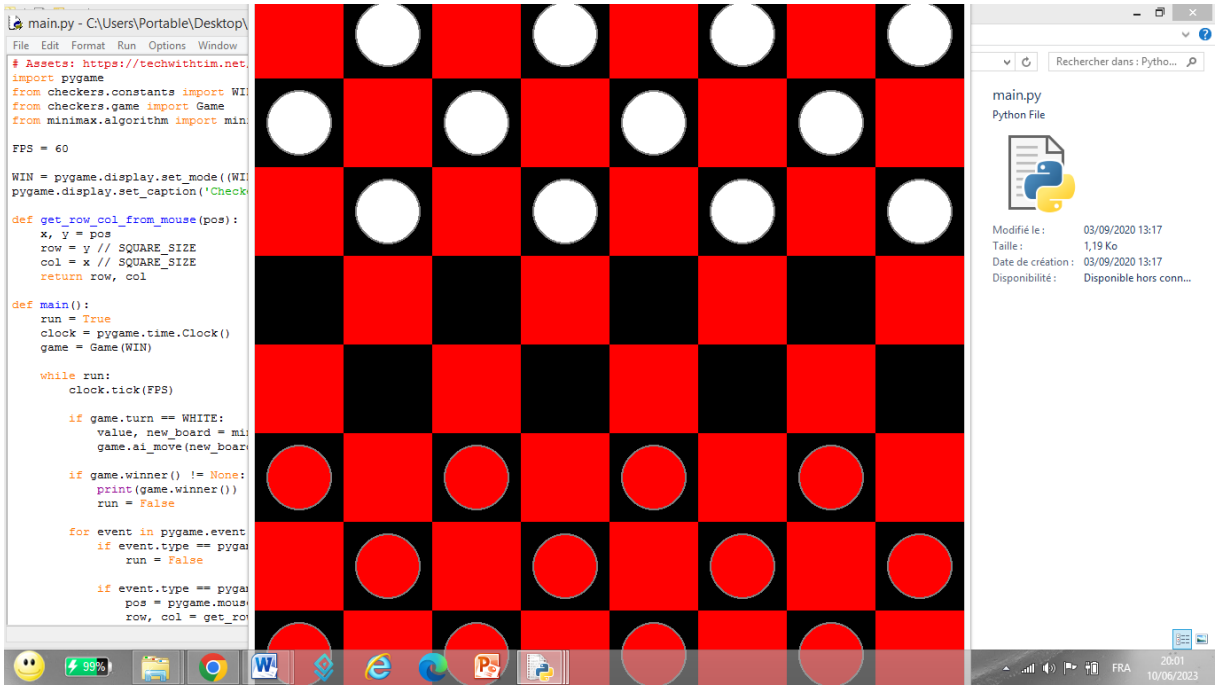

Figure 5.1 : Interface d'accueil.

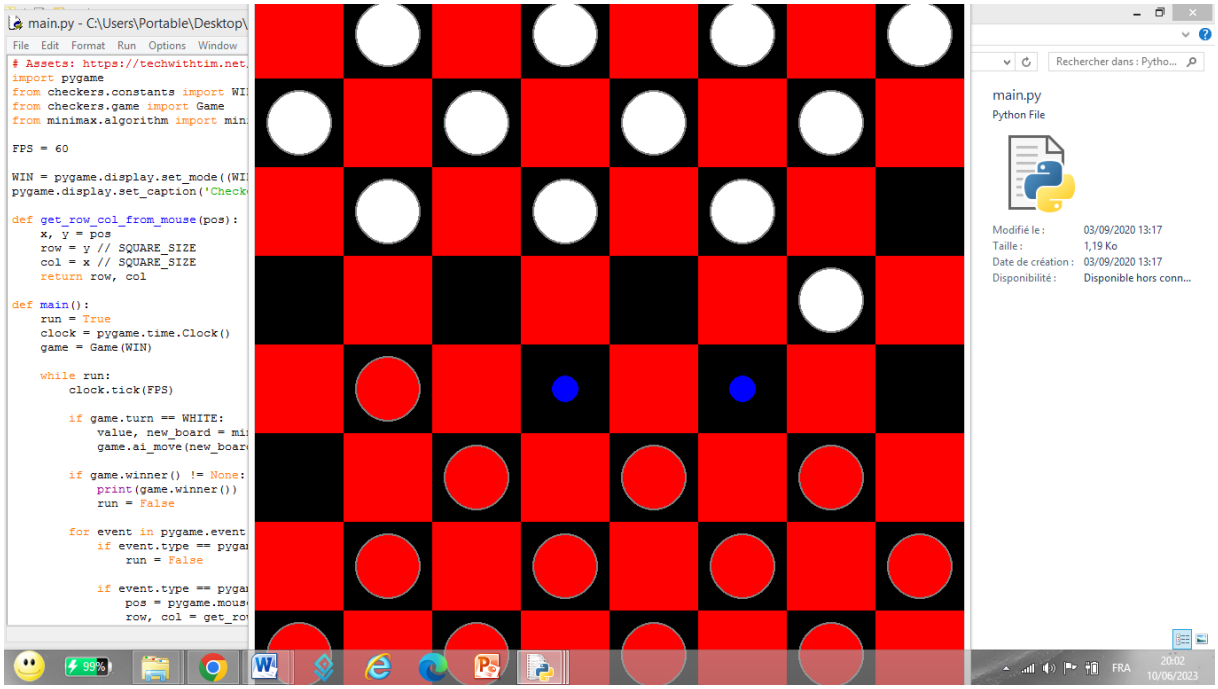

Figure 5.2 : Interface montrant la sélection de mouvements

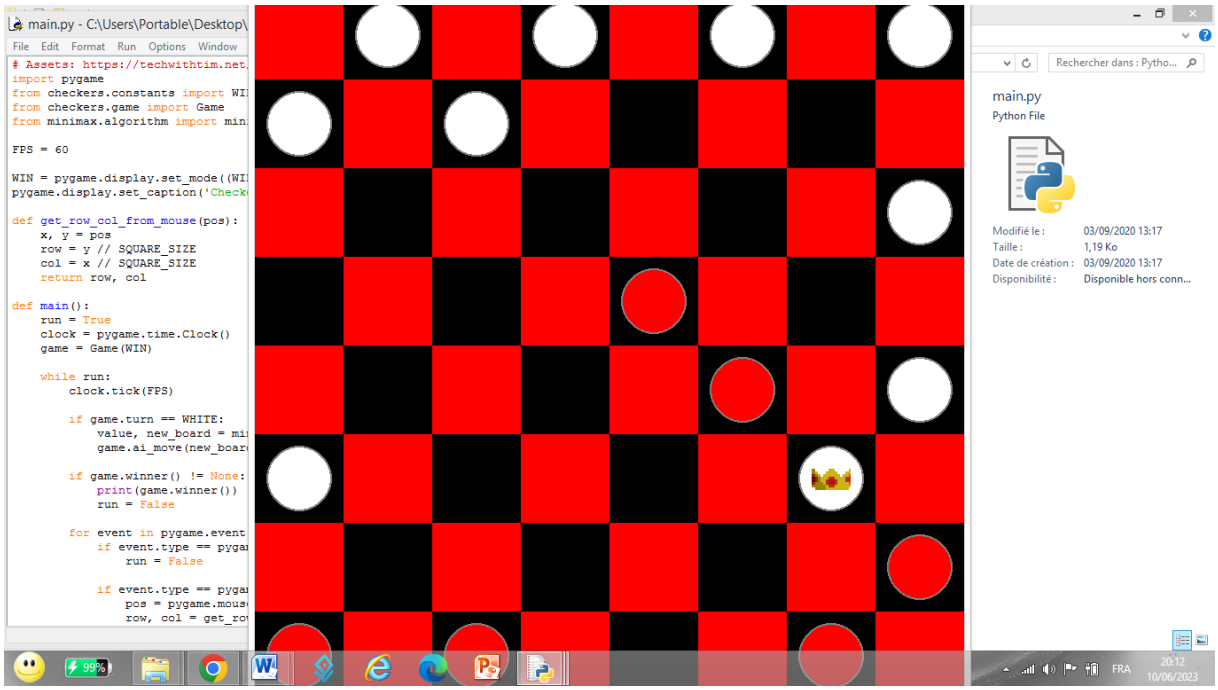

Figure 5.3 : Interface montrant la pièce de roi

6.Conclusion :

Dans le dernier chapitre, nous avons examiné les étapes nécessaires pour développer un logiciel répondant aux besoins et objectifs définis dans ce projet. Tout d'abord, il est essentiel d'avoir une compréhension approfondie du problème à résoudre. Ensuite, nous avons présenté l'environnement dans lequel nous avons travaillé, ainsi que quelques-unes des méthodes que nous avons utilisées. En résumé, la réalisation du logiciel nécessite une compréhension approfondie du problème et l'utilisation de méthodes spécifiques que nous avons abordées.

Le code commence par importer les modules nécessaires et définir certaines constantes, telles que la largeur et la hauteur de la fenêtre du jeu, la taille des cases du plateau, et les couleurs utilisées :

Ensuite, la fonction get_row_col_from_mouse est définie pour obtenir la ligne et la colonne correspondantes à une position donnée de la souris sur la fenêtre du jeu.

La fonction principale main est ensuite définie. Elle initialise les variables nécessaires, crée une instance de la classe Game (qui représente l'état actuel du jeu), et commence la boucle principale du jeu.

À chaque itération de la boucle, le temps est mis à jour, et si c'est le tour du joueur blanc, la fonction minimax est utilisée pour déterminer le meilleur coup à jouer. Ensuite, le coup est effectué par l'IA en appelant la méthode ai_move de l'objet game.

Le code vérifie également s'il y a un gagnant à chaque itération en appelant la méthode winner de l'objet game. Si un gagnant est trouvé, le nom du gagnant est affiché, et la boucle principale est interrompue.

Le code gère également les événements de la fenêtre, tels que la fermeture de la fenêtre et les clics de souris. Lorsqu'un clic de souris est détecté, la position est convertie en une ligne et une colonne en appelant la fonction get_row_col_from_mouse, puis la méthode select de l'objet game est appelée pour gérer la sélection d'une pièce par le joueur.

Enfin, la méthode update de l'objet game est appelée pour mettre à jour l'affichage du jeu, et lorsque la boucle principale est terminée, la bibliothèque Pygame est arrêtée avec pygame.quit().

En conclusion, ce code implémente un jeu de dames en utilisant Pygame et utilise l'algorithme minimax pour l'IA. Il gère les interactions du joueur avec la souris et affiche le gagnant à la fin du jeu.

CONCLUSION GÉNÉRALE

Dans cette étude, nous avons cherché à repousser les limites de trois domaines très différents : la théorie des jeux, les algorithmes évolutionnaires et le minimax.

Dans le contexte de la théorie des jeux, nous avons considéré les stratégies de jeu comme des individus devant s'adapter à un environnement changeant. Nous avons utilisé des algorithmes évolutionnaires pour faire évoluer une population de stratégies et sélectionner les individus les mieux adaptés. L'application des algorithmes évolutionnaires au minimax présente plusieurs avantages. Cela nous permet non seulement d'améliorer la vitesse de convergence du minimax, mais aussi d'éviter de converger vers des états correspondant à des minima locaux qui ne fourniraient pas de solutions optimales. Nous avons expérimenté différentes méthodes évolutionnaires pour adapter des stratégies de jeu basées sur le minimax, et avons identifié les stratégies d'évolution comme la méthode la plus appropriée pour découvrir de nouvelles stratégies optimales dans les jeux combinatoires. Cette méthode a démontré des résultats stables dans le temps et lors de la répétition d'événements similaires. Le type de minimax que nous avons utilisé était hautement optimisé pour éviter toute convergence prématurée, et particulièrement efficace pour prendre des décisions optimales dans des situations de jeu possibles. Bien que les résultats obtenus ne répondent pas toujours à nos attentes en raison de la complexité extrême de notre problème et des limitations des capacités des ordinateurs actuels, certaines recherches ont été axées sur la fusion des calculs évolutionnaires avec le minimax. On trouve des applications dans des domaines tels que la robotique pour la recherche de trajectoires optimales ou l'évitement d'obstacles, et la vision pour la détection ou la reconnaissance d'images. En combinant l'algorithme minimax avec un algorithme génétique, on peut améliorer les capacités de recherche et de prise de décision de l'intelligence artificielle jouant aux dames, ce qui conduit à un gameplay amélioré et à des choix stratégiques plus pertinents.

En conclusion, à travers cette étude, nous avons souhaité contribuer à l'exploration de nouvelles voies de recherche dans le domaine de la théorie des jeux, en particulier en fusionnant les algorithmes évolutionnaires avec le minimax pour résoudre des problèmes où la prise de décision optimale est une tâche cruciale dans notre vie quotidienne. Nous avons également cherché à améliorer le temps d'exécution des algorithmes, afin d'optimiser leur utilisation dans différents contextes.

Bibliographie **:**

[1]. https://www.google.dz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCwQFjAA&url=http%3A%2F%2Fwww.lania.mx%2F~ccoello%2FEMOO%2Fthesis_berro.pdf.gz&ei=JB50UZGAO6i54ASl44HIBw&usg=AFQjCNGfczh_WhWDR9SsnnZ8H40k7AkQUQ&bvm=bv.45512109,d.ZWU

[2]. Redouane TLEMSANI, Nabil NEGGAZ et Abdelkader BENYETTOU " *Amélioration de l'Apprentissage des Réseaux Neuronaux par les Algorithmes Evolutionnaires : application à la classification phonétique*", Mars 2005.

[3]. Anne Spalanzani, " *Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole*", Octobre 1999.

[4]. Fatiha Kacher & Karima Bouibed "*La théorie des jeux*", 2012.

[5]. Ouassila Labbani, " *Comparaison des théories des jeux pour l'étude du comportement d'agents*", Juillet 2003.

[6]. http://www.lecactusheuristique.com/article-24209600.html

[7]. Didier Müller, EPFL-DMA, CH-1015 Lausanne, " *Utilisation d'un réseau de neurones artificiels comme fonction d'évaluation d'un jeu* ", Novembre 1992.

[8]. http://frostiebek.free.fr/docs/Theorie%20des%20jeux/Games.doc

[9]. Tuomas W. Sandholm and Robert H. Crites. "*Multiagent reinforcement learning in the iterated prisoner's dilemma"*. BioSystems, 37(1,2) :147{66, 1996.

[10]. http://fluminis.free.fr/Rapport_Echecs.pdf

[11]. http://www.ffothello.org/info/algos.php

[12]. Claude Perdigou, " *Caractérisation de comportement dynamique en robotique mobilen & application de la robotique évolutionniste*", Mars 2011.

[13]. Hichem talbi, " *Algorithmes évolutionnaires quantiques pour le recalage et la*

segmentation multi-objectif d'images", 2009.

[14]. J.Greenstette. « genetic algorithms" IEEE. Octobre 1993. 5-8

[15]. DAV92 b LDAVALO. "*Handbook of genetic algorithms*". ED. VNR New York 1992.

[16]. http://www.memoireonline.com/04/11/4389/m_Optimisation-de-lenergie-reactive-dans-un-reseau-denergie-electrique18.html

[17]. Thomas Vallée, Murat Yıldızoğlu∗, " *Présentation des algorithmes génétiques et de leurs applications en économie* ", Décembre 2003.

[18]. http://ar.scribd.com/doc/94822193/memoireouali

[19]. Z.MICHALEWICZ. "*genetic algorithms + data structures=Evolution programs*". ED.spring-Verlag. New York 1992.

[20]. DREDI Leila. "*Les algorithmes génétiques*". Université de Constantine, 2005.

[21]. Schwefel H.-P., "*Evolution Strategies: A Family of Non-Linear Optimization Techniques Based on Imitating Some Principles of Organic Evolution*", Annals of Operations Research, vol. 1, pp. 165-167, 1984.

[25]. Axelrod, R. (1987), The evolution of strategies in the iterated prisoner's dilemma, *in* L. D. Davis, ed., "Genetic algorithms and simulated annealing", Morgan Kaufmann.

[26]. F.Foucaud, J.Terral A.Parant, J.Radanielina, " *Implémentation du jeu de Dames Chinoises*", Avril 2008.