الجمـــهورية الـــــجزائرية الديمقـــــراطية الشعـــــــبية
République Algérienne Démocratique et Populaire
وزارة التعليـــــــــم العالـــــــــي و البحث العلمـــــــي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 20 Août 1955 – Skikda		جـامعة 20 أوت 1955- سكيكدة
Faculté des Sciences		كـــــلية العلـــــوم
Departement d'informatique		قسم الإعـــلام الآلـي

Mémoire De fin d'étude en vue de l'obtention du Diplôme de Master en Informatique

Option : Intelligence Artificielle

Sujet :

Conception et réalisation d'un système Hybridation du calcul neurenal avec le calcul évolutionnaire pour améliorer les stratégies de jeux combinatoires

Réalisé par l'étudiante :

- BENGGUEDAH Ferdous

Dirigé par :

Dr.BENOUDINA Lazhar

Année Universitaire 2023-2024

Remerciement

En préambule à ce mémoire, je tiens à remercier ALLAH qui m'a aidé et m'a donné la patience et le courage durant ces longues années d'étude. Je souhaite adresser mes remerciements les plus sincères à toutes les personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire. Mes premiers remerciements vont au corps professoral et administratif du département d'informatique pour la richesse et la qualité de leurs enseignements, ainsi que pour les grands efforts qu'ils déploient afin d'assurer à leurs étudiants une formation actualisée.

Je tiens à exprimer ma sincère gratitude envers Monsieur BENOUDINA Lazhar, en tant qu'encadrant de mémoire. Il a toujours fait preuve d'écoute et de disponibilité tout au long de la réalisation de ce mémoire, et je le remercie également pour l'inspiration, l'aide, et le temps qu'il a bien voulu me consacrer pour que ce mémoire voie le jour.

Je n'oublie pas de mentionner ma famille pour leur contribution, leur soutien, et leur patience. Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis qui m'ont constamment encouragé au cours de la réalisation de ce mémoire. Merci à toutes et à tous.

Dédicaces

À ma chère mère Boucherek Sihem et mon pere, pour tous leurs sacrifices, leurs amour, leurs Tendresse, leurs soutien et leurs prières tout au long de mes études, Mon cher mari pour leur appui et leur encouragement, À ma chère sœur Assala pour leur encouragements permanents, Et leur soutien moral, À mes frères Ahmed Anis, Nacer Eddine, À toute ma famille pour leur soutien tout au long de mon parcours universitaire Que ce travail soit l'accomplissement de vos vœux tant allégués, et La fuite de votre soutien infaillible Merci d'être toujours là pour moi

Résumé

Evolutionnaires a fait l'objet de nombreuses recherches. Il est possible d'utiliser ces derniers pour créer et apprendre des réseaux neuronaux, que ce soit en termes de structure ou de pondérations. La réalisation de notre projet est fondée sur l'hybridation des algorithmes évolutionnaires avec les réseaux de neurones pour faire évoluer une population de stratégies de jeu. Un algorithme évolué a appris à jouer le jeu de dames sans recourir à des caractéristiques qui nécessiteraient l'intervention humaine. Le programme évolutionnaire a amélioré les réseaux de neurones artificiels en utilisant uniquement les positions des pièces sur la table de jeu et le différentiel de pièce pour évaluer des positions alternatives dans le jeu. Le programme a été enseigné pendant plusieurs centaines de générations à jouer à un niveau compétitif avec des experts humains.

Abstract

Evolutionary algorithms have been the subject of extensive research. These algorithms can be used to create and train neural networks, both in terms of structure and weights. Our project is based on the hybridization of evolutionary algorithms with neural networks to evolve a population of game strategies. An evolved algorithm learned to play checkers without relying on features that would require human intervention. The evolutionary program improved the artificial neural networks by using only the positions of the pieces on the game board and the piece differential to evaluate alternative positions in the game. The program was trained over several hundred generations to play at a competitive level with human experts.

Table des matières

Contents

Table des figures

Table des Algorithmes

Table des Tableaux

Introduction générale

Dans notre quotidien, chacun d'entre nous a déjà participé à des jeux tels que les échecs, les dames, les cartes, ou s'est trouvé en situation de concurrence, Toutefois, peu de personnes ont pris en compte la méthode mathématique pour représenter ces instants de divertissement ou de rivalité, une perspective proposée par la théorie des jeux. Dans ce domaine mathématique, on étudie des situations où le résultat final de chaque personne dépend non seulement de ses propres choix, mais aussi de ceux de ses adversaires, ce qui offre une approche qui peut être utilisée dans les domaines des sciences sociales, économiques et même de l'informatique.

Dans le contexte actuel, Les problèmes d'optimisation jouent un rôle essentiel au sein de la communauté scientifique, encouragés par les avancées technologiques en informatique. On peut souvent observer le comportement des stratégies alternatives dans les jeux à travers des stimuli-réponses, mais la limitation à des fonctions linéaires rend les résultats discutables. Le sujet de notre mémoire est de représenter de manière efficace les stratégies de jeu et de résoudre les problèmes d'explosion combinatoire liés à la théorie des jeux combinatoires.

Afin de faire face à ces problèmes, nous utilisons une méthode innovante qui combine différentes méthodes. Plus précisément, nous étudions la combinaison des réseaux de neurones et des algorithmes évolutionnaires, en combinant les bénéfices complémentaires de ces deux approches. En évitant la convergence vers des optimaux locaux, les algorithmes évolutionnaires s'associent de manière efficace aux réseaux de neurones, ce qui permet une résolution plus solide des problèmes de sélection d'actions dans les jeux complexes.

Dans la continuité du travail mené par les chercheurs américains David B. Fogel et Rumar Chellapilla, cette étude vise à mettre en œuvre l'association du calcul neuronal et du calcul évolutionnaire afin de dénicher de nouvelles stratégies de jeu. Nous avons constaté que l'utilisation d'algorithmes évolutionnaires pour mener une recherche globale peut compenser les variations de performance des méthodes de recherche locales en élargissant la portée de la recherche vers des solutions potentiellement efficaces

Enfin, notre mémoire est structuré en cinq chapitres :

Chapitre 1 : La théorie des jeux

Ce premier chapitre posera les fondements théoriques en mettant en lumière les concepts clés de la théorie des jeux. Il explorera également les algorithmes de recherche appliqués à ces jeux, mettant en avant leur utilité dans la modélisation et la résolution de problèmes stratégiques.

Chapitre 2 : Les réseaux de neurones

Le deuxième chapitre se penchera sur l'application des réseaux de neurones, en mettant en avant le modèle du perceptron multicouches et la technique de rétro-propagation du gradient. L'accent sera mis sur la compréhension de ces outils en tant que composantes du système hybride envisagé.

Chapitre 3 : Les algorithmes génétiques et évolutionnaires

Le troisième chapitre sera dédié à l'exploration des algorithmes évolutionnaires. Il analysera leur utilisation dans le contexte des jeux combinatoires, mettant en évidence leur capacité à évoluer et à s'adapter pour trouver des solutions optimales.

Chapitre 4 : Analyse et conception

Le quatrième chapitre abordera l'aspect pratique de la conception du système. Il se concentrera sur l'utilisation de la modélisation UML et des diagrammes de flux de données pour décrire de manière formelle la structure et le fonctionnement du système hybride.

Chapitre 5 : L'implémentation

Enfin, le cinquième chapitre se consacrera à l'implémentation concrète du système en utilisant le langage de programmation Python. Il détaillera les choix spécifiques faits dans le cadre de l'implémentation, mettant en lumière les différentes fonctionnalités du système résultant de l'hybridation entre le calcul neuronal et les algorithmes évolutionnaires. Chaque fonctionnalité sera décrite succinctement, offrant une vue d'ensemble des avancées concrètes réalisées dans le cadre de ce projet. suivi d'une conclusion générale.

Chapitre 01
Théories 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 un domaine des mathématiques, de la recherche opérationnelle et de l'économie qui vise à représenter mathématiquement des situations qui sont en conflit. Elle offre des idées pratiques et des instruments formels pour l'étude du conflit et sa mise en scène.

En fait, elle consiste à analyser l'interaction dans un groupe d'agents rationnels qui a un comportement stratégique, par exemple elles nous permettent de mieux comprendre le déroulement des guerres. **[6]**

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. **[7]**

2. Historique

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 prouvé l'existence du théorème de min-max qui a joué et joue encore un rôle crucial 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.

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 trouver une solution optimale pour résoudre les conflits. **[7]**

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* ". **[8]**

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. **[8]**

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 principal défi de la théorie des jeux traditionnelle : la description des comportements rationnels et la nécessité de prévoir 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. **[7]**

5. Définition d'un jeu

Un jeu est une situation dans laquelle des personnes (joueurs) sont amenées à prendre des décisions parmi différentes options, dans un contexte préétabli. "*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. **[7]**

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*. **[9]**

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 avec un nombre limité de coups.
- Chaque joueur a un nombre fini de choix à chaque coup.

***Exemple* :** Jeu de Nim. Il est nécessaire de vider plusieurs tas de pions de taille différente - habituellement, il y a trois tas de 3, 4 et 5.). 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** se caractérise par des paiements réciproques. C'est à dire que les gains d'un joueur sont les pertes d'autres joueurs. Il n'y a aucune contribution externe.

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 parfaitement informé est un jeu qui répond aux critères suivants :

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 joueurs alternent et sont familiers avec tous les mouvements depuis le début de la partie.

- Un jeu est considéré comme **imparfait** 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 l'un des joueurs n'est pas au courant de l'organisation 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*. Il est considéré que les joueurs vont ensuite jouer ce qui a été sélectionné, c'est 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

Pour avoir une meilleure compréhension des principes de la théorie des jeux, il est essentiel de garder à l'esprit les deux caractéristiques essentielles des joueurs théoriques qui déroulent des algorithmes. Ils sont supposés :

7.1. Joueur intelligent

Ce qui signifie qu'un joueur est i**ntelligent**, c'est qu'il fera toujours le meilleur choix. 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 plusIls sont constamment en quête de réduire au maximum leurs pertes éventuelles.

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 :

- L'ensemble des stratégies ne peut être connu 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 [7]

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. **[10]**

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

Pour trouver le coup le plus efficace, ou la meilleure stratégie, il faut représenter un jeu ou une partie de manière à pouvoir l'exploiter.. 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 cette approche, les lignes et les colonnes symbolisent les différentes stratégies ouvertes aux deux joueurs. En règle générale, les lignes symbolisent les stratégies du joueur visant à maximiser leur efficacité.. 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. **[10]**

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

***Exemple* :**

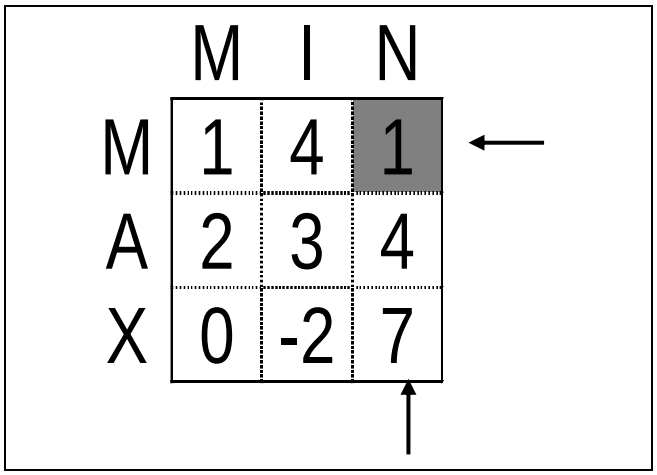


Figure 1.1 Exemple de représentation matricielle d'un jeu.

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. Il s'agit d'une illustration claire de toutes les actions envisageables 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. De cette manière, la racine correspond à la position initiale du jeu, tandis que les feuilles, ou nœuds terminaux, correspondent aux positions de fin de partie.
- Les arcs symbolisent les coups effectués par 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* :**

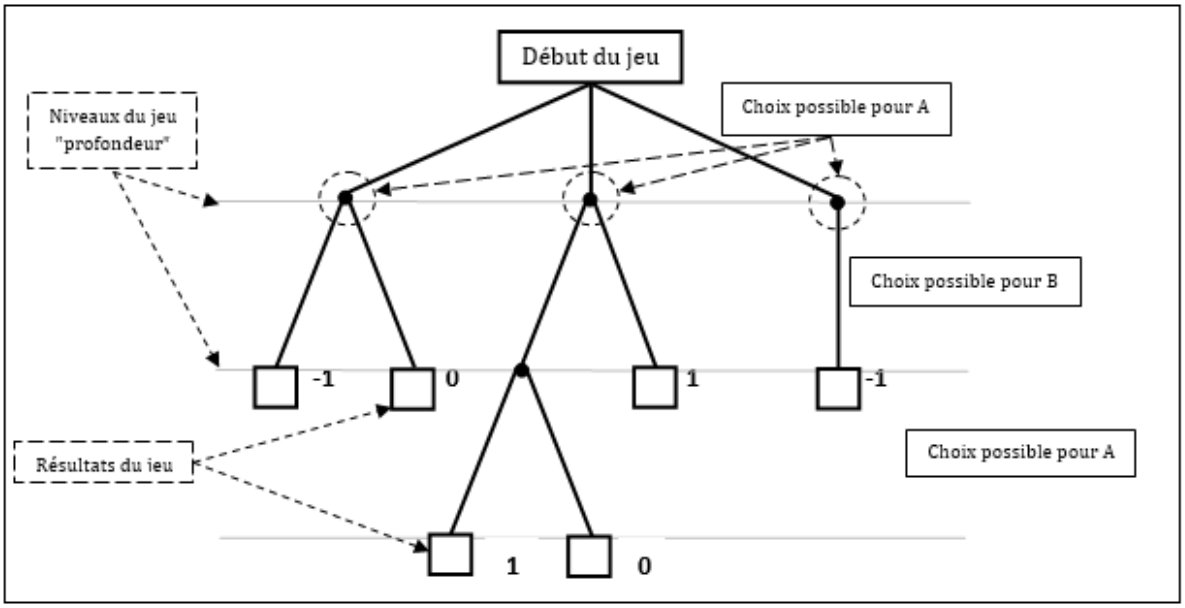


Figure 1.2 Exemple de représentation arborescente d'un jeu.

***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* :**

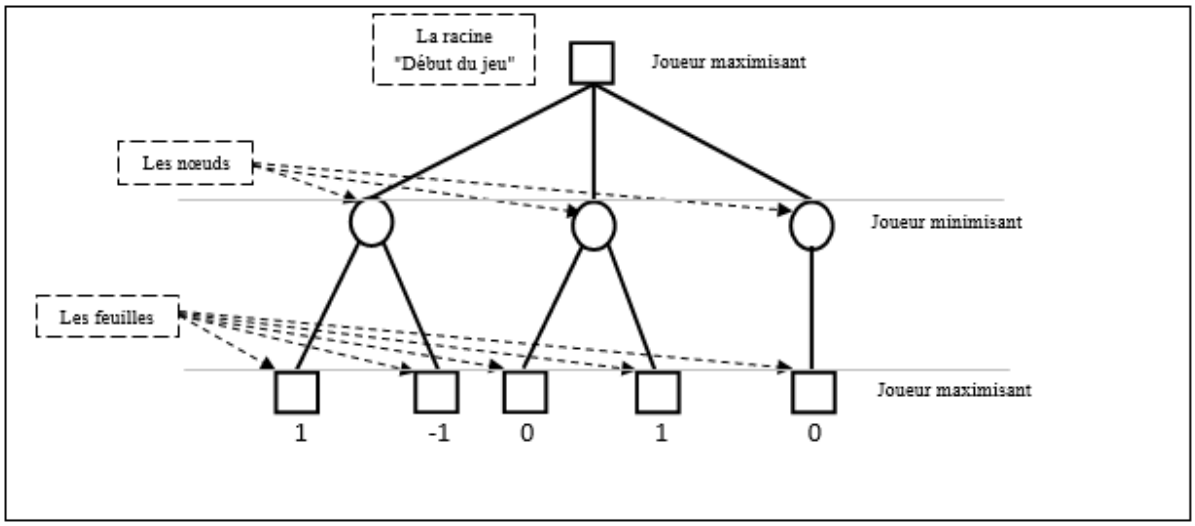


Figure 1.3 Exemple de représentation d'un arbre de jeu avec des carrés et des cercles.

***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 évident qu'une fonction heuristique bien développée permet d'améliorer le jeu. 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. **[51]**

12. Les algorithmes de recherche

Afin de résoudre un problème, un algorithme de recherche doit effectuer une exploration systématique et contrôlée de l'espace d'états.

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 qui maximise prend donc, pour chaque ligne, la valeur algébrique la plus faible. 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 la même manière, le joueur qui minimise prend la valeur algébrique la plus élevée pour chaque colonne (ses pertes sont équivalentes aux gains du joueur qui maximise). 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.

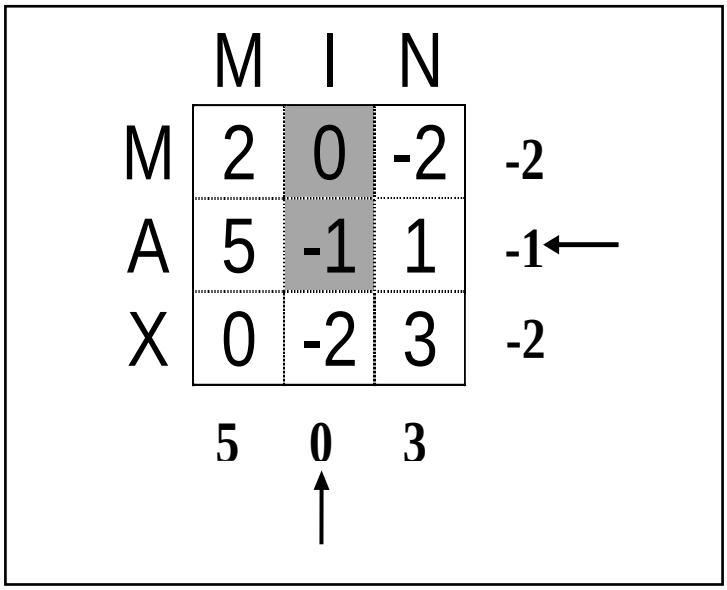


Figure 1.4 Exemple d'application de l'algorithme Minimax avec la représentation matricielle.

Le joueur maximisant commence par calculer les pires cas pour chaque ligne (stratégies). Par la suite, il opte pour celle qui lui entraînera le moins de pertes : la stratégie 2, qui a un 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.

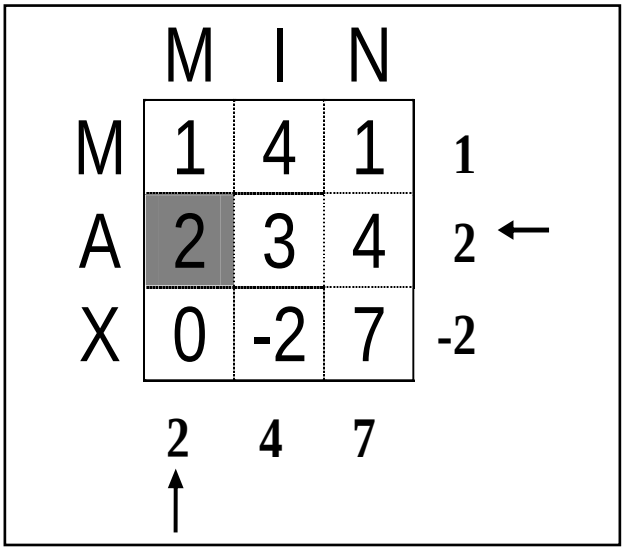


Figure 1.5 Exemple de représentation matricielle en présence d'un jeu avec point col.

En effet, une fois que l'algorithme du Minimax a été mis en œuvre, comme dans l'exemple précédent, on constate que la valeur obtenue à partir des deux stratégies optimales est identique : 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 la même manière, si le joueur qui minimise opte pour une stratégie différente de la première, il risque de perdre davantage (3 ou 4 au lieu de 2). Il n'y a donc aucun des deux joueurs qui puissent faire mieux que son meilleur.

12.1.2. Forme extensive :

L'algorithme du Minimax tient compte de l'idée que l'adversaire va toujours faire son maximum. Il est donc nécessaire, pour chaque joueur de rechercher le coup qui va lui assurer le minimum de perte "stratégie sécurisante".

De cette manière, à chaque moment où le joueur maximisant doit prendre une décision, il prendra en compte la valeur la plus importante de ses fils. Réciproquement, le joueur minimisant prendra la plus petite des valeurs de ses fils. **[10]**

***Exemple* :**

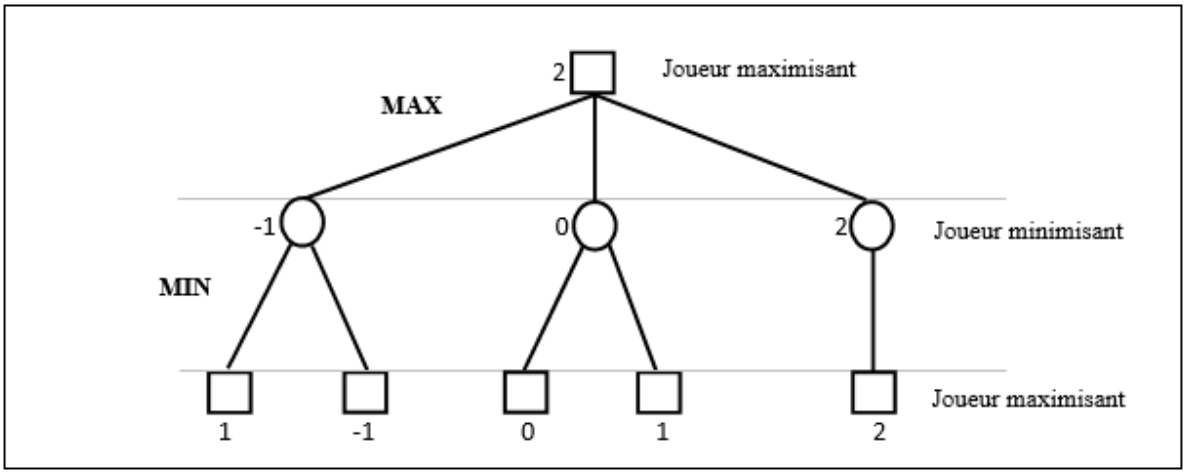


Figure 1.6 Exemple de calcul de la valeur d'un nœud en fonction des valeurs de ses fils.

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* :**

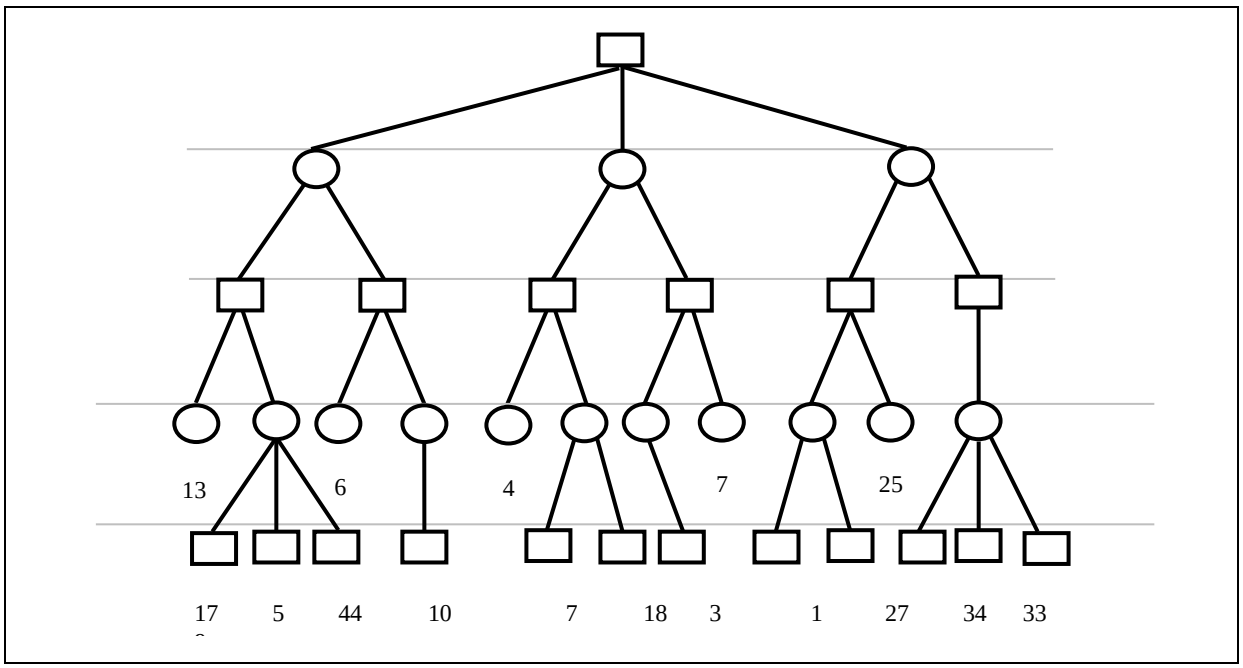


Figure 1.7 Exemple d'arbre de jeu.

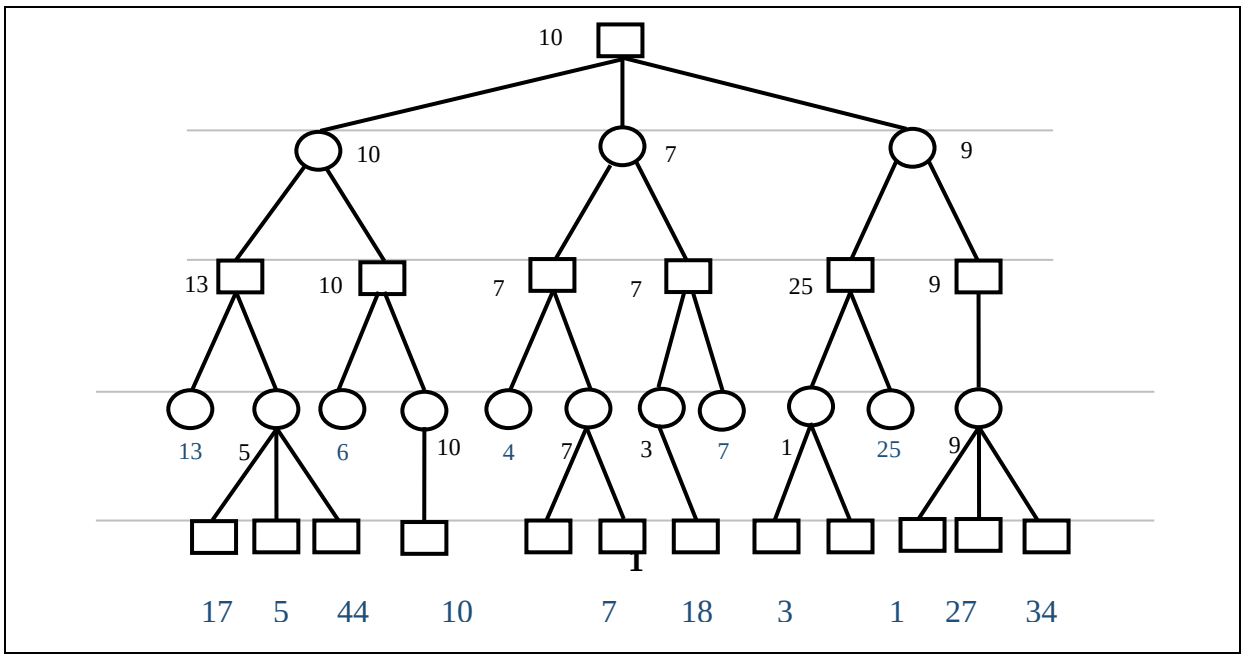


Figure 1.8 Exemple résolu avec l'algorithme du Minimax.

12.1.3. Les limite 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. Tous les autres algorithmes reprennent cependant son principe, dont les améliorations seront principalement basées sur l'introduction de méthodes d'élagage. En d'autres termes, on évitera d'examiner les branches de l'arbre qui ne peuvent plus avoir d'influence sur le résultat à un moment donné de la calculation.

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. **[10]**

12.2. L'élagage Alpha-Bêta

Il est possible d'améliorer le Minimax en élaguant certaines branches de l'arbre. Les valeurs de référence des ancêtres du nœud où l'on se trouve sont l'idée fondamentale. 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.

Autrement dit, l'algorithme alpha-bêta «α–β» se base sur l'historique de la recherche déjà réalisée au début de la recherche minimax afin de limiter les variations potentielles 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*. L'exploration d'un nœud J est interrompue dès que la valeur courante "cv" est vérifiée : $cv \leq \alpha$. On a donc, initi $\alpha = -\infty$

- **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*. L'exploration d'un nœud J est interrompue dès que la valeur courante "cv" est vérifiée : $cv \geq \beta$. On a donc, initialem $\beta = +\infty$

12.2.2. Avantages par rapport au Minimax :

Cet algorithme présente l'avantage de fournir les mêmes résultats que l'algorithme Minimax tout en étant bien plus rapide. Lorsque les nœuds sont bien organisés, la vitesse de l'algorithme est augmentée car les coupes se produisent 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. **[12]**

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* :**

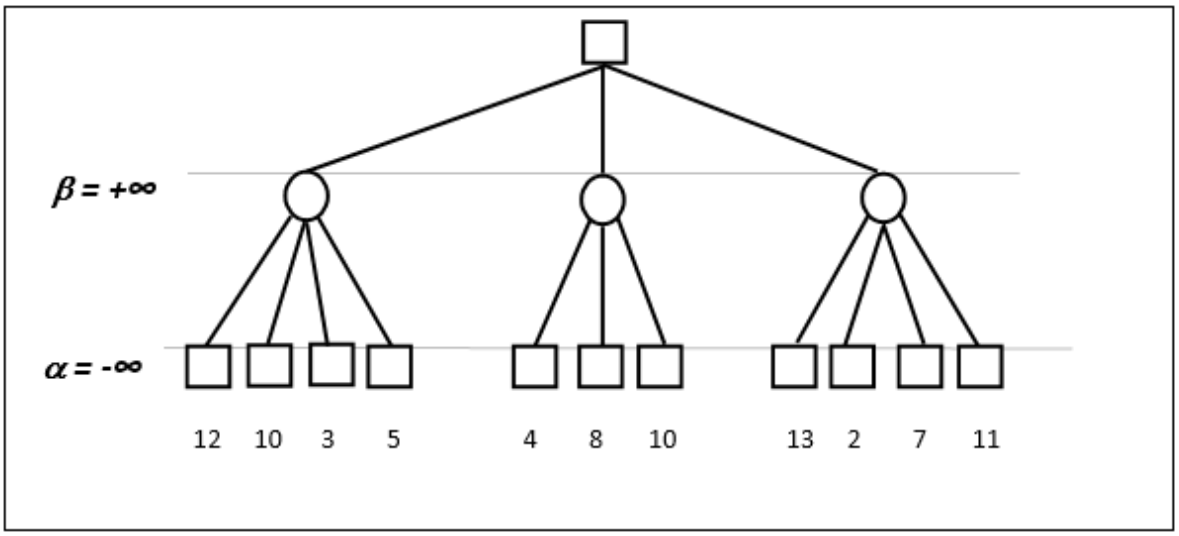


Figure 1.9 Exemple d'arbre de jeu.

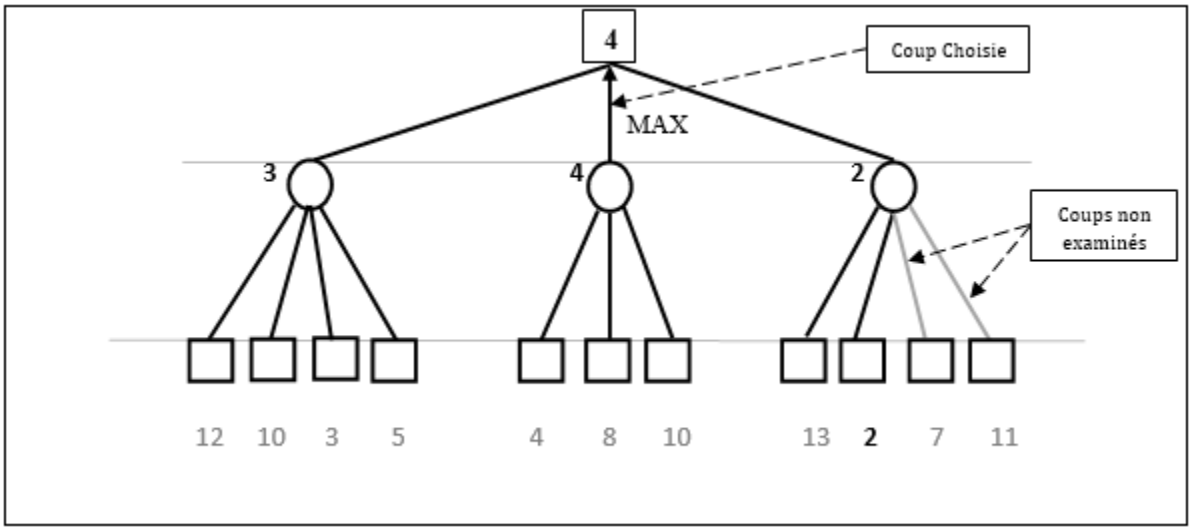


Figure 1.10 Exemple résolu par l'algorithme α–β Sens de parcours de droite à gauche.

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 : **[13]**

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

Au cours de ce chapitre, nous avons exposé les concepts fondamentaux 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 mathématiqu

Chapitre 02 :
Les Réseaux de neurones

1. Introduction

Reproduire l'intelligence de l'être humain constitue sans aucun doute le rêve le plus captivant de nombreux scientifiques de notre époque. Les réseaux de neurones sont fondés sur des modèles qui tentent d'expliquer comment les cellules du cerveau et leurs interconnexions parviennent, d'un point de vue global, à exécuter des calculs complexes. Au cours des deux dernières décennies, les réseaux de neurones ont constaté un développement fulgurant **[29].** Cet intérêt a démarré avec l'application réussie de cette technique puissante pour des problématiques très différentes, et dans des domaines aussi divers que la finance, la médecine, la production industrielle, la géologie ou encore la physique. Le succès croissant des réseaux de neurones sur la plupart des autres techniques statistiques peut s'attribuer à leur puissance, leur polyvalence et à leur simplicité d'utilisation. Ils sont des techniques extrêmement sophistiquées de modélisation et de prévision, en mesure de modéliser des relations entre des données ou des fonctions particulièrement complexes. **[29]** Nous donnerons dans ce chapitre les notions de base pour la compréhension des réseaux de neurones.

2. Historique

En 1943, le premier neurone formel a été proposé par les deux biophysiciens, *Mc Culloch* et *Pitts*. Leur but était de comprendre les propriétés des systèmes nerveux à partir de composants élémentaires. **[30]**

Grâce à des modèles à base neurones simplifiés *"Neurones formels"*, ils montrent qu'il est possible de construire des systèmes vérifiant la définition de Turing pour les machines à calculer à usage général et donc capables de calculer des fonctions logiques. **[31]**

En 1949, *Donald Hebb* s'attaque au problème de l'apprentissage, il a proposé une formulation du mécanisme d'apprentissage, sous la forme d'une règle de modification des connexions synaptiques "*règle de Hebb*". Cette règle décrit la manière dont les cellules apprennent à modifier l'intensité des connexions qui les relient. **[30]**

En 1956, a lieu une grande conférence à Datrmouth sur le thème de l'intelligence artificielle et de l'apprentissage, elle sera le point de départ de l'âge d'or des réseaux de neurones et de l'intelligence artificielle. **[31]**

En 1958, l'apparition du premier réseau de neurones artificiel, grâce aux travaux de *Rosenblatt* qui conçoit le fameux *"perceptron"*. Le perceptron est inspiré du système visuel (en termes d'architecture neurobiologique) et possède une couche de neurones d'entrée (perceptive) ainsi qu'une couche de neurones de sortie (décisionnelle). **[30]**

En 1965, *Nilsson* publie "Machine Learning" qui donne les fondements mathématiques de l'apprentissage automatique pour la reconnaissance des formes. **[31]**

En 1969, une critique violente du perceptron par *Minsky* et *Papert* qui montrent dans un livre "*Perceptrons*" toutes les limites de ce modèle, et soulèvent particulièrement l'incapacité du perceptron à résoudre les problèmes non linéairement séparables, tels que le célèbre problème du XOR (ou exclusif), et les difficultés théoriques posées par l'apprentissage dans les réseaux multicouches. **[30]**

En 1982, *Hopfield* a démontré tout l'intérêt de l'utilisation des réseaux récurrents "*feed-back*" pour la compréhension et la modélisation des processus mnésiques. Les réseaux récurrents constituent alors la deuxième grande classe de réseaux de neurones, avec les réseaux type perceptron "*feed-forward*". **[31]**

En 1986, *Werbos* conçoit son algorithme de *rétro propagation* de l'erreur, qui offre un mécanisme d'apprentissage pour les réseaux multicouches de type perceptron appelés *MLP* (Multi Layer Perceptron), fournissant ainsi un moyen simple d'entraîner les neurones des couches cachées. **[31]**

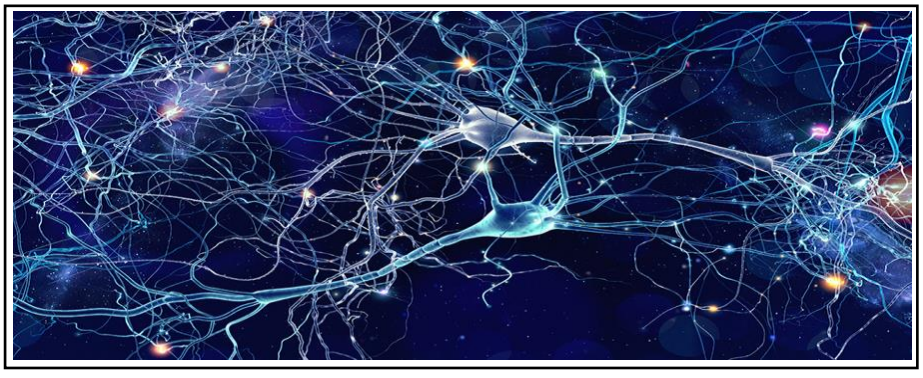


Figure 2.1 Les Réseaux de neurones

3. Le Neurone Biologique

Près de 100 milliards de neurones sont connectés dans le cerveau humain. Un neurone est composé de plusieurs parties : - **Les dendrites**: représentent les entrées du neurone. Ce sont eux qui envoient les signaux au corps cellulaire ; – **Le corps cellulaire** : traite les informations parvenues à travers les dendrites et contrôle la réaction du neurone en fonction des informations reçues en entrée ; —**L'axone**: l'information traitée est envoyée à travers l'axone. — Une dendrite d'un autre neurone peut être en contact avec l'extrémité de l'axone ; — **Les synapses :** communiquent l'information aux autres neurones et aux fibres musculaire.

(Pour des réponses nerveuses potentielles). Au bout de l'axone se trouvent les synapses.

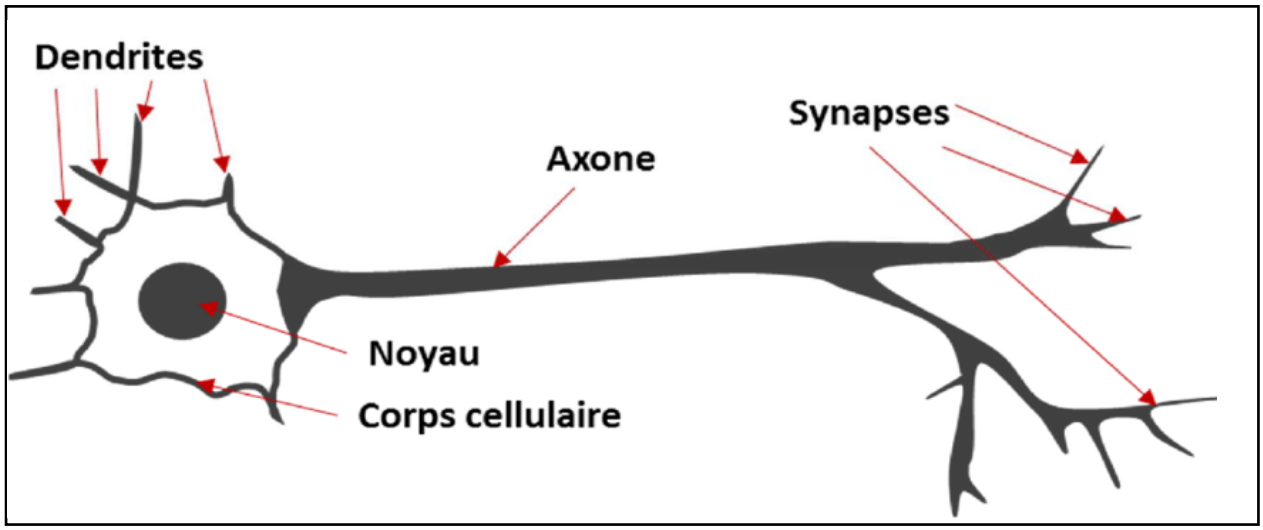


Figure 2.2 Schématisation d'un neurone biologique

Ce sont les connexions synaptiques qui permettent aux humains de faire des choses comme

- reconnaître des formes ;

- mémoriser des ;

- apprendre par l'exemple.

Ainsi, chaque neurone n'agit pas indépendamment. Ces actions sont possibles grâce au comportement global de l'ensemble du réseau.

4. Le Neurone Formel

Le neurone formel (artificiel) est l'unité de traitement élémentaire dans un réseau de neurones artificiels, créé par McCulloch et Pitts (1943) en raison de leur similitude superficielle avec les neurones biologiques.

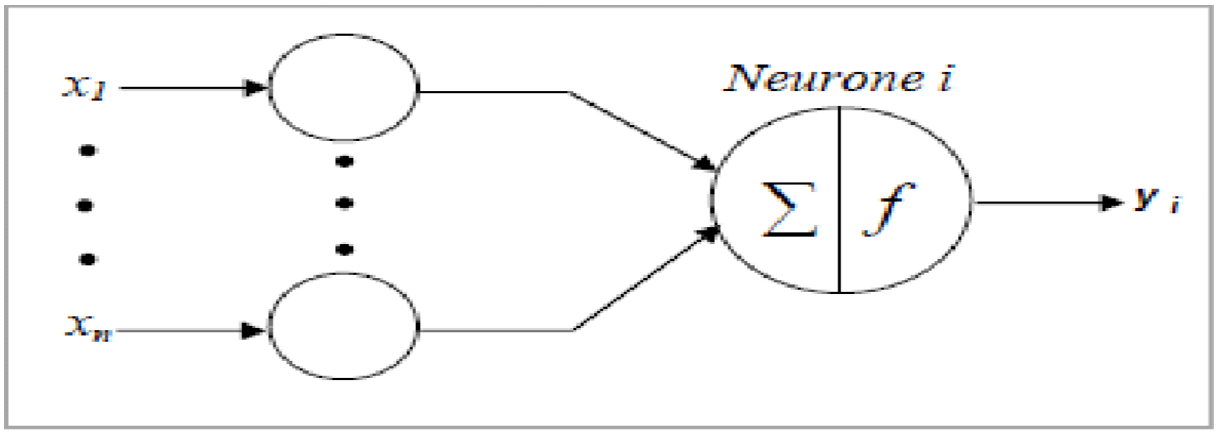


Figure 2.3 Schéma general d'un neurone formel

Un neurone formel, en comparaison avec son homologue biologique, est composé de :

– **des synapses** représentées par des flèches pondérées par des poids ;

– **un centre** qui est l'équivalent du corp cellulaire où se déroulent des activités mathématiques telles que la sommation des entrées ;

– **une fonction d'activation** qui s'applique à la sommation des entrées pour la comparer au seuil d'activité ;

– **une sortie.**

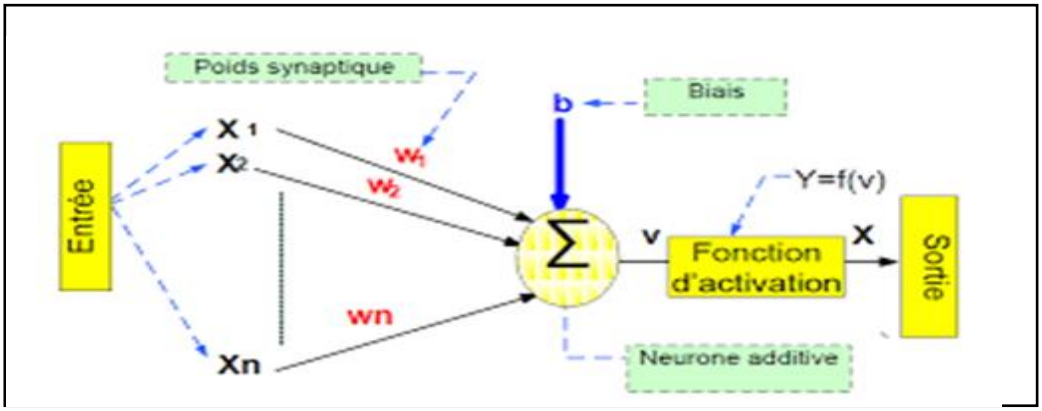


Figure 2.4 Schéma des composants d'un neurone formel

Les deux équations suivantes peuvent être utilisées pour décrire mathématiquement un neurone formel k :

$v = \sum_{j=1}^{n} w_j x_j \quad y = \varphi(u - \theta)$

Les entrées sont : $x_1, x_2, \dots, x_n$

- les poids synaptiques du neurone k sont : $w_1, w_2, \dots, w_n$
- La sortie de l'unité de sommation est : v
- Le seuil est θ
- la fonction d'activation est $\varphi(\,.\,)$
- le signal de sortie du neurone k est y

Trois idées caractérisent un neurone formel :

– **L'état** est une valeur scalaire qui détermine si un neurone s'active ou non ;

–**Les connexions** ont un poids qui indique leur importance ;

– **La fonction d'activation** (transfert) calcule la valeur de l'état du neurone qui sera transmise aux neurones avals.

5. Les Réseaux de neurones artificielle

Réseau artificiel de neurones (RNA) est une structure de neurones formels disposés en couches et qui fonctionnent en même temps. Dans un réseau, chaque couche réalise un traitement autonome des autres, puis transmet le résultat de son analyse à la couche suivante. De cette façon, l'information passe successivement de la couche d'entrée à la couche de sortie, éventuellement en passant par une ou plusieurs couches intermédiaires connues sous le nom de « couches cachées ». D'après l'algorithme d'apprentissage employé, il est aussi envisageable de propager l'information dans le sens inverse, connue sous le nom de rétropropagation. Dans la plupart des cas, sauf pour les couches d'entrée et de sortie, chaque neurone d'une couche est relié à tous les neurones de la couche précédente et de la couche suivante.

Les RNA peuvent envelopper les connaissances empiriques et les rendre opérationnelles. Les connaissances du réseau sont enregistrées dans les poids synaptiques, qui sont modifiés par des processus d'adaptation ou d'apprentissage. Ainsi, les RNA sont semblables au

fonctionnement du cerveau, où la connaissance est obtenue par l'apprentissage et conservée dans les connexions synaptiques entre les neurones.

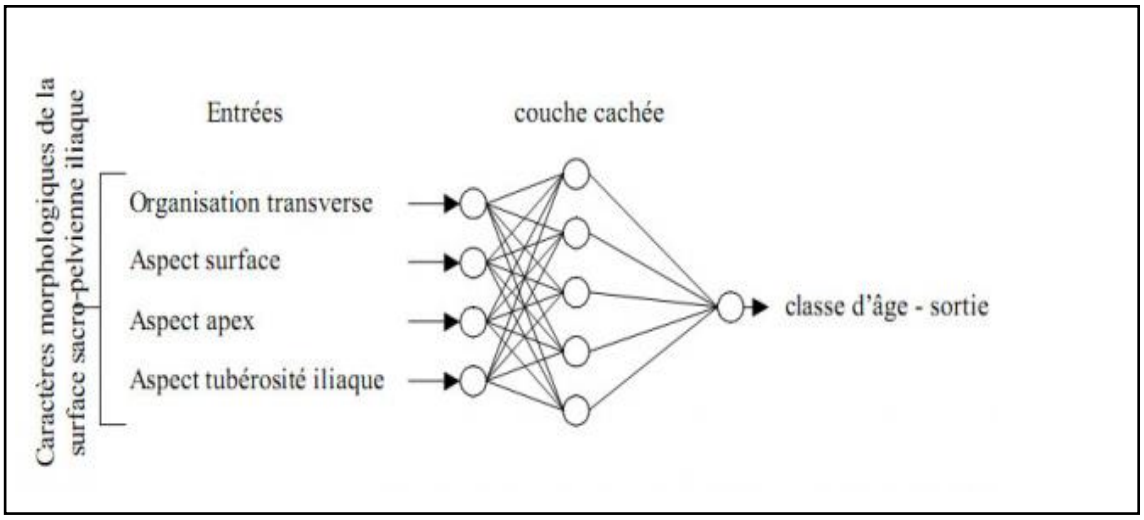
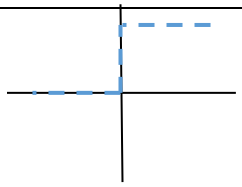
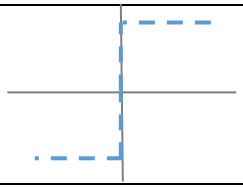
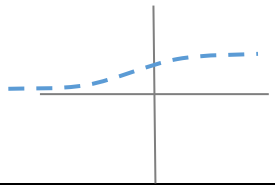


Figure 2.3 Schéma des réseaux de neurones artificiels

6. Les Fonctions d'activation

Appelée aussi *Fonction de transfert*, le choix d'une fonction d'activation est un élément constitutif important des réseaux de neurones. Ainsi, l'identité n'est pas toujours suffisante, bien au contraire, et le plus souvent des fonctions non linéaires et plus évoluées seront nécessaires. **[34]**

À titre illustratif, le tableau suivant (tableau 2.1) présente quelques fonctions couramment utilisées comme fonctions d'activation, avec : $y = f(n)$.

Nom	Graphe	Relation entré/sortie	Étendue
Seuil		$\begin{cases} y = 0, & n < 0 \\ y = 1, & n \geq 0 \end{cases}$	**{0, 1}**
Seuil symétrique		$\begin{cases} y = -1, & n < 0 \\ y = 1, & n \geq 0 \end{cases}$	**{-1, 1}**
Sigmoïde		$y = 1 / (1 + \mathrm{Exp}(-n))$	**{0; 1}**

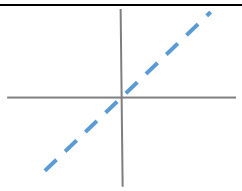
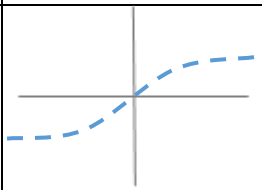
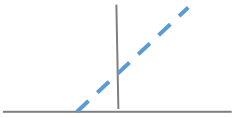
Linéaire		$y = n$	$]-\infty, +\infty[$
Fonction tanh		$y = 2 / (1 + \text{Exp}(-2n)) - 1$	$\{-1, 1\}$
Gaussienne		$y = \text{Exp}(-(n^2) / 2)$	$]\ 0, +\infty[$
Linéaire positive		$\begin{cases} y = 0, & n < 0 \\ y = n, & n \geq 0 \end{cases}$	$\{0, +\infty[$

Tableau 2.1 : Fonctions de transfert.

7. Architecture des réseaux de neurones

Architecture est le terme le plus général pour désigner la façon dont sont disposés et connectés les différents neurones qui composent un réseau. Les RNA peuvent être divisés en deux catégories principales :

7.1. Les réseaux Feed-Forward :

Ces réseaux, également connus sous le nom de réseaux de type perceptron, sont des réseaux où l'information se propage de couche en couche sans possibilité de retour en arrière.

7.1.1. Les perceptrons :

7.1.1.1. Le perceptron monocouche :

C'est historiquement le premier RNA, proposé par Rosenblatt. C'est un réseau simple, puisque il ne se compose que d'une couche d'entrée et d'une couche de sortie.

Il a été conçu dans un but premier de reconnaissance des formes. Toutefois, il peut également servir à la classification et à la résolution de simples opérations logiques comme "ET", "OU". Sa principale limite est qu'il ne peut résoudre que des problèmes linéairement

séparables. En général, il suit une formation supervisée en suivant la règle de correction de l'erreur.

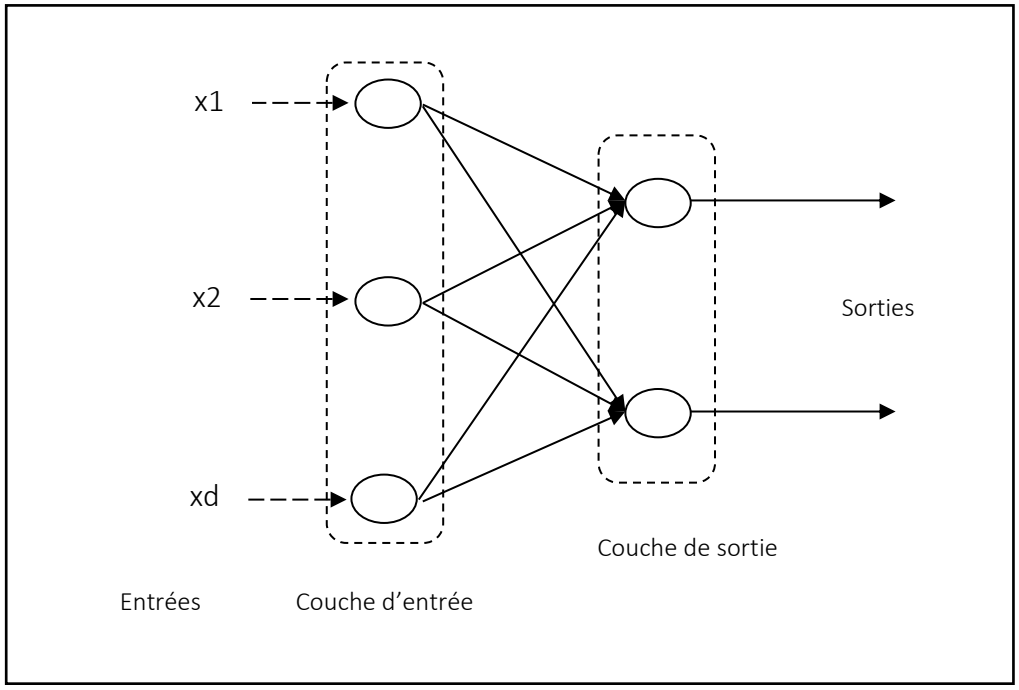


Figure 2.5 Schéma d'un perceptron mono-couche.

7.1.1.2. Perceptron multicouches :

C'est une extension du précédent, avec une ou plusieurs couches cachées entre l'entrée et la sortie. Chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et la couche suivante (excepté pour la couche d'entrée et de sortie) et il Il n'existe pas de liens entre les cellules d'une couche commune. Dans ce genre de réseaux, les fonctions d'activation sont principalement des fonctions à seuil ou sigmoïdes.

Il peut résoudre des problèmes non linéairement séparables et des problèmes logiques plus compliqués. Il reçoit également un apprentissage supervisée en suivant la règle de correction des erreurs.

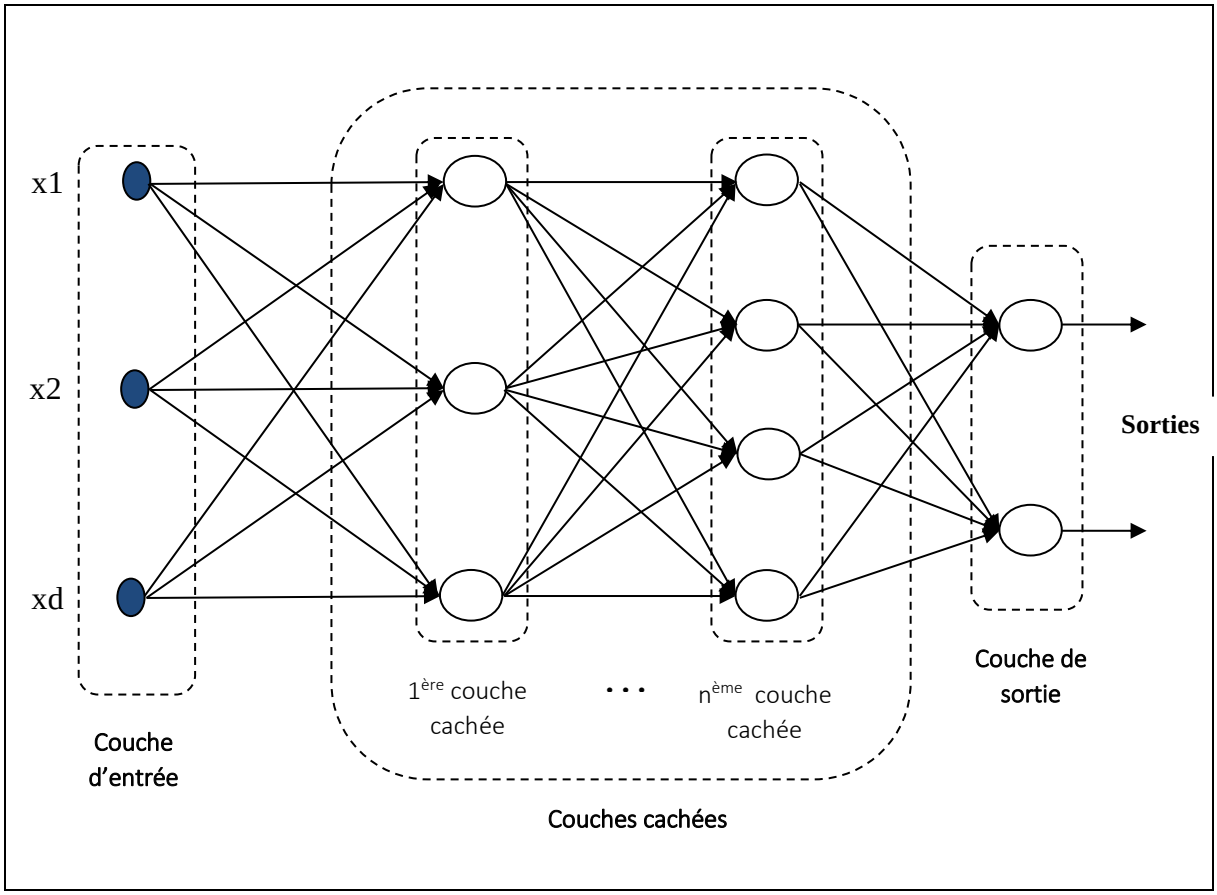


Figure 2.6 Schéma d'un perceptron multi couches.

7.1.2. Les réseaux à fonction radiale "RBF" :

L'architecture de ces réseaux est la même que pour les perceptrons multicouches (PMC) cependant, les fonctions de base utilisées ici sont des *fonctions gaussienne*. Les RBF seront donc employés dans les mêmes types de problèmes que les PMC à savoir, en classification et en approximation de fonctions, particulièrement.

Le mode hybride est le mode d'apprentissage le plus couramment employé pour les RBF, et les règles utilisées peuvent être soit la règle de correction de l'erreur, soit la règle d'apprentissage par compétition.

7.1.3. Les réseaux convolutifs (CNN) :

Les réseaux convolutifs, ou Convolutional Neural Networks (CNN), sont une classe de réseaux de neurones spécialement conçus pour traiter les données structurées en grille, comme

les images. Les CNN sont composés de couches de convolution, de sous-échantillonnage (pooling) et de couches entièrement connectées.

- Les couches de convolution appliquent des filtres pour détecter des caractéristiques locales des données d'entrée, comme les bords et les textures dans les images.
- Les couches de pooling réduisent la dimensionnalité des données, ce qui permet de diminuer le nombre de paramètres et de contrôler le surapprentissage.
- Les couches entièrement connectées, situées en fin de réseau, sont similaires à celles des perceptrons multicouches et sont utilisées pour la classification finale. Les CNN sont particulièrement efficaces pour les tâches de vision par ordinateur, comme la reconnaissance d'images et la détection d'objets.

7.2. Les réseaux Feed-Back :

Aussi connus sous le nom de réseaux récurrents, il s'agit de réseaux où l'information est renvoyée en arrière.

7.2.1. Les cartes auto-organisatrices de Kohonen :

Ce sont les réseaux à apprentissage *non supervisé* qui établissent une carte discrète, ordonnée topo logiquement, en fonction de patterns d'entrée. Le réseau forme ainsi une sorte de treillis dont chaque nœud est un neurone associé à un vecteur de poids. Chaque entrée est calculée en fonction de la correspondance entre chaque vecteur de poids. Ensuite, on va modifier le vecteur de poids avec la corrélation la plus élevée, ainsi que certains de ses voisins, pour accroître encore cette corrélation.

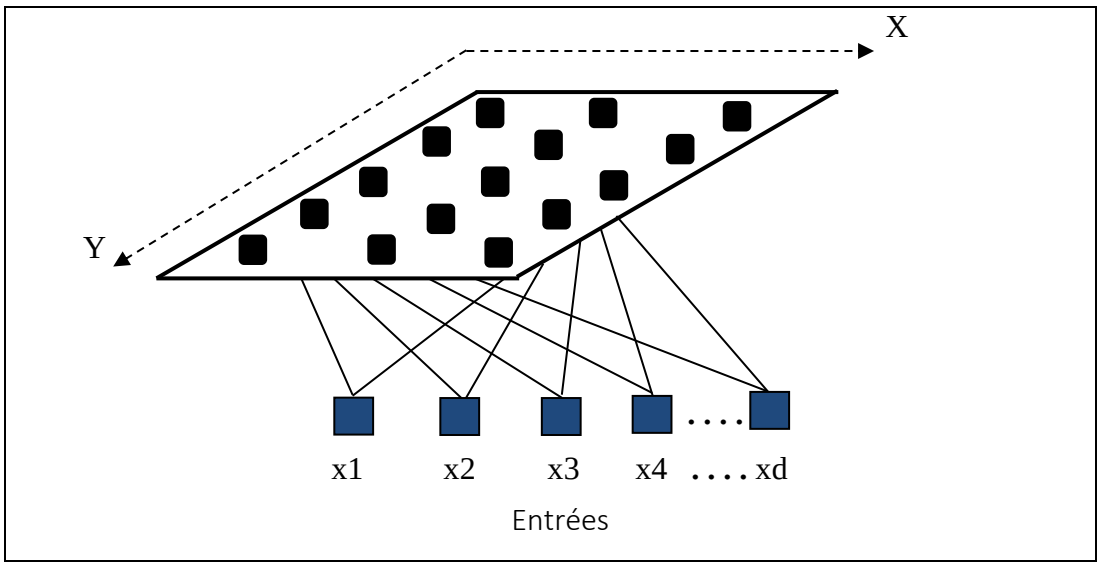


Figure 2.7 carte auto-organisatrice à deux dimensions.

7.2.2. Les réseaux de Hopfield :

Les Hopfield sont des réseaux récurrents et entièrement connectés. Dans ce type de réseau, tous les neurones sont reliés à tous les autres neurones et il n'existe aucune distinction entre les neurones d'entrée et de sortie. Ils sont à la manière d'une mémoire associative non-linéaire et peuvent repérer un objet stocké en se basant sur des représentations partielles ou bruitées. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Le mode d'apprentissage utilisé ici est le mode *non supervisé.*

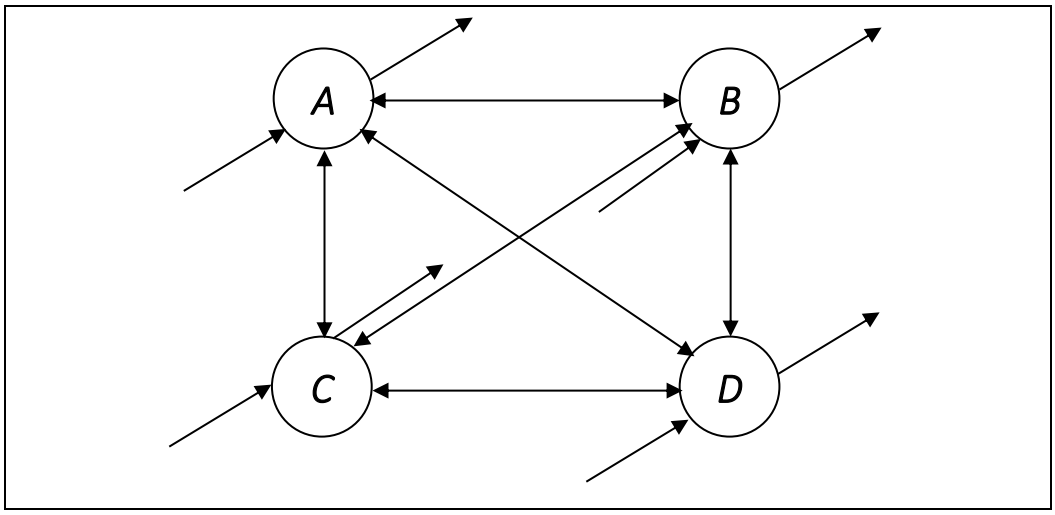


Figure 2.8 Réseau de Hopfield à 4 neurones.

7.2.3. Les réseaux ART :

Les réseaux ART "Adaptative Resonnance Theory" sont des réseaux à apprentissage par compétition. Le principal défi dans ce genre de réseaux réside dans la question de la "stabilité/plasticité". En effet, dans un apprentissage par compétition, rien ne garantit que les catégories formées restent stables. La seule possibilité, pour assurer la stabilité, serait que le coefficient d'apprentissage tende vers le zéro, mais le réseau perdrait alors sa plasticité.

Les ART ont été spécialement élaborés pour éviter cette problématique. Les vecteurs de poids ne seront appropriés dans ce type de réseau que si l'entrée fournie est assez proche d'un prototype déjà connu par le réseau. On parlera alors de résonnance. A l'inverse, si l'entrée s'éloigne trop des prototypes existants, une nouvelle catégorie va se créer, avec prototype, l'entrée qui a engendrée sa création.

Il convient de souligner qu'il y a deux catégories principales de réseaux ART : les ART-1 pour les entrées binaires et les ART-2 pour les entrées simultanées. Le mode d'apprentissage des ART peut être *supervisé ou non*.

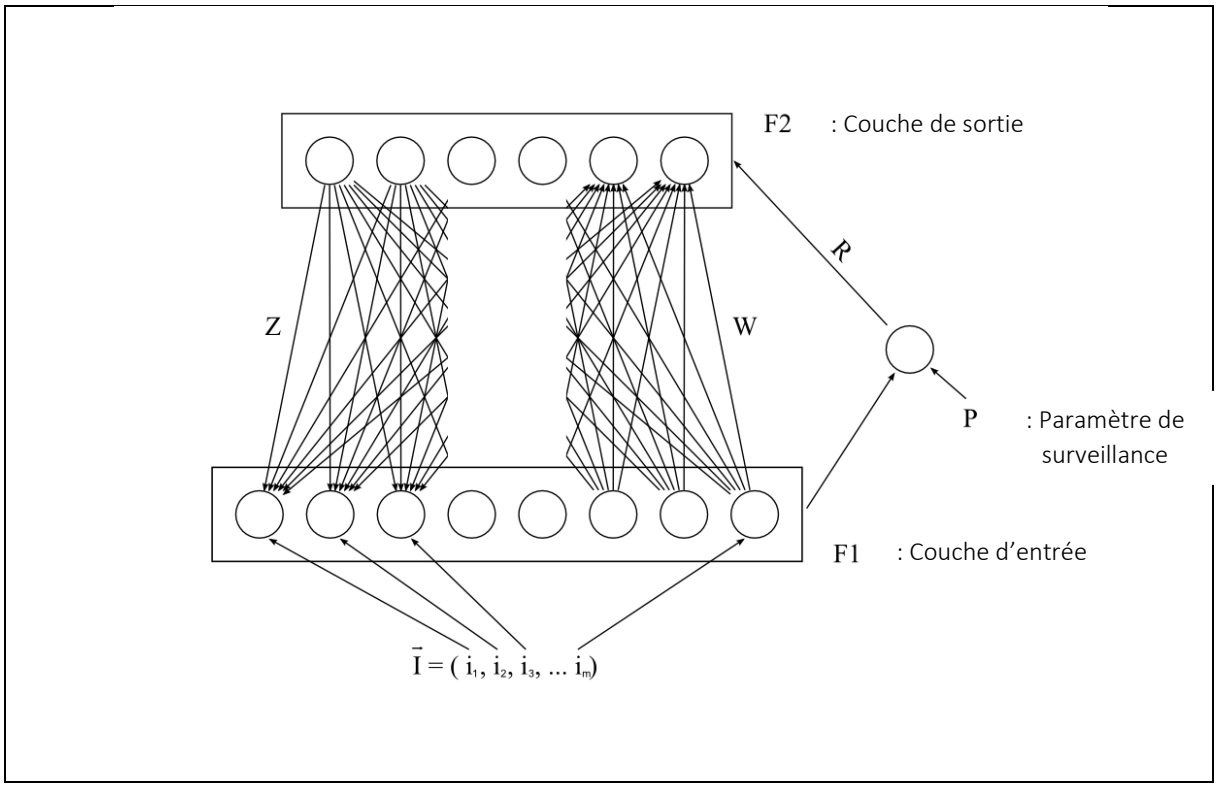


Figure 2.9 Réseau ART.

8. Domaines d'application des réseaux de neurones

Le tableau suivant (tableau 2.2) résume, en général, la correspondance entre le type de réseaux et les applications les plus appropriées à ce dernier. **[35]**

Caractéristiques fonctionnelles	***Type des réseaux de neurones***
Reconnaissance de formes	MLP, Hopefield, Kohonen
Mémoires associatives	MLP, Hopefield, Kohonen
Optimisation	Hopefield, ART
Approximation de fonction	MLP, RBF
Modélisation et Control	MLP
Traitement d'image	Hopefield
Classification et clustrering	MLP, ART, RBF, Kohonen

Tableau 2.2 : Correspondance Réseaux de neurones – Domaines d'application.

9. L'Apprentissage dans les réseaux de neurones

L'apprentissage pour un RNA peut être perçu comme le défi de la mise à jour des poids des connexions au sein du réseau, dans le but de réaliser la tâche qui lui est confiée. L'apprentissage est la caractéristique principale des RNA et il peut se faire de différentes manières et selon différente règles. **[30]**

9.1. Types d'apprentissage :

9.1.1. Le mode supervisé :

Dans ce type d'apprentissage, le réseau s'adapte par comparaison entre le résultat qu'il a calculé, en fonction des entrées fournies, et la réponse attendue en sortie. Ainsi, le réseau va se modifier jusqu'à ce qu'il trouve la bonne sortie, c'est-à-dire celle attendue, correspondant à une entrée donnée.

9.1.2. Le renforcement :

En réalité, le renforcement est une forme d'apprentissage supervisé et certains auteurs le qualifient d'un mode supervisé. Dans cette méthode, le réseau doit acquérir des connaissances

sur la relation entre l'entrée et la sortie en estimant son erreur, c'est-à-dire le rapport entre l'échec et le succès. Ainsi, le réseau cherchera à optimiser un indicateur de performance qui lui est attribué, connu sous le nom de signal de renforcement. Étant donné que le système est capable de déterminer si la réponse qu'il fournit est correcte ou non, il ne sait pas la bonne réponse.

9.1.3. Le mode non-supervisé :

Appelé aussi auto-organisationnel, dans ce cas, l'apprentissage est basé sur des probabilités. Selon les régularités statistiques de l'entrée, le réseau va évoluer et créer des catégories en attribuant et en optimisant une valeur de qualité aux catégories identifiées.

9.1.4. Le mode hybride :

Le mode hybride reprend en fait les deux autres approche, puisque une partie des poids va être déterminée par apprentissage supervisé et d'autre partie par apprentissage non-supervisé.

9.1.5. L'apprentissage profond (Deep Learning) :

L'apprentissage profond est une sous-catégorie de l'apprentissage automatique qui utilise des réseaux de neurones artificiels avec de nombreuses couches cachées pour modéliser des représentations complexes des données. Ces réseaux de neurones profonds sont capables d'apprendre des caractéristiques hiérarchiques et abstraites à partir de grandes quantités de données. Le deep learning peut être utilisé dans différents modes d'apprentissage, notamment supervisé, non-supervisé et par renforcement, selon le contexte et les objectifs de la tâche.

9.2. Règles d'apprentissage :

9.2.1. Règle de correction d'erreurs :

Cette règle s'inscrit dans le paradigme d'apprentissage supervisé, c'est-à-dire dans le cas où l'on fournit au réseau une entrée et la sortie correspondante.

En considérant « y » comme la sortie calculée par le réseau et « d » comme la sortie souhaitée, le principe de cette règle consiste à utiliser l'erreur (d-y). afin de modifier les connexions et de diminuer ainsi l'erreur globale du système. Le réseau va donc s'adapter jusqu'à ce que "*y*" soit très proche de "*d*". On retrouve ce principe en particulier dans le modèle du perceptron simple.

9.2.2. Apprentissage de Boltzmann :

Les réseaux de Boltzmann sont des réseaux symétriques récurrents. Ils possèdent deux sous-groupes de cellules, le premier étant relié à l'environnement "*cellules visibles*" et le second ne l'étant pas "*cellules cachées*". Cette règle d'apprentissage est de type stochastique (qui relève partiellement du hasard) et elle consiste à ajuster les poids des connexions, de telle sorte que l'état des cellules visibles satisfasse une distribution probabiliste souhaitée.

9.2.3. Règle de Hebb :

Cette règle repose sur des informations biologiques et illustre le fait que si des neurones, de part et d'autre d'une synapse, sont activés de manière synchrone et répétée, cela entraîne une activation synchrone et répétée, la force de la connexion synaptique va aller croissant. Il est à noté que l'apprentissage est localisé, c'est-à-dire que la modification d'un poids synaptique "*Wij*" ne dépend que de l'activation d'un neurone "*i*" et d'un autre neurone "*j*".

9.2.4. Règle d'apprentissage par compétitions :

La particularité de cette règle, c'est qu'ici l'apprentissage ne concerne qu'un seul neurone. Cet apprentissage consiste à regrouper les données en différentes catégories. Ainsi, les mêmes patrons seront classés dans une même classe en fonction des corrélations des données et seront représentés par un seul neurone, ce qui est appelé "winner-take-all".

Chaque neurone de sortie d'un réseau à compétition simple est relié aux neurones de la couche d'entrée, aux autres cellules de la couche de sortie (connexions inhibitrices) et à lui-même (connexion excitatrice). La sortie va donc dépendre de la compétition entre les connexions inhibitrices et excitatrices. **[30]**

10. Les méthodes d'apprentissage :

Les connaissances de l'expert sont énumérées dans les systèmes experts : elles sont définies par des règles. En ce qui concerne les réseaux de neurones, les informations sont distribuées : elles sont enregistrées dans les poids des connexions et la structure du réseau, les fonctions de transfert de chaque neurone, le seuil de ces fonctions, la méthode d'apprentissage utilisée. **[36]**

Il existe un certain nombre de méthodes d'apprentissage, et dans ce travail nous nous intéressons à la méthode de *rétro-propagation du gradient de l'erreur*.

10.1. La rétro-propagation du gradient de l'erreur :

10.1.1. Représentation :

Cet algorithme est utilisé dans les réseaux de type Feed-Forward, c'est l'outil le plus utilisé actuellement dans le domaine des réseaux de neurones. Le principe de la rétro-propagation implique de présenter au réseau un vecteur d'entrées, puis de calculer la sortie en passant par les couches cachées, de la couche d'entrée à la couche de sortie. Cette sortie obtenue est comparée à la sortie désirée, une *erreur* est alors obtenue.

On calcule le gradient de l'erreur en se basant sur cette erreur, qui est ensuite propagé de la couche de sortie vers la couche d'entrée, d'où le concept de rétro-propagation. Cela permet de changer les poids du réseau, ce qui favorise l'apprentissage. L'opération est réitérée pour chaque vecteur d'entrée et cela jusqu'à ce que le critère d'arrêt soit vérifié. **[36]**

L'algorithme de rétro-propagation du gradient de l'erreur est basé sur la *minimisation de l'erreur quadratique* "E" calculée en fonction des *n* sorties désirées $\mathbf{yd_i}$ et des *n* sorties effectivement données **yi** par le réseau **[37]:**

$$\mathbf{E_i = \sum_{i=1}^{n} (yi - ydi)^2}$$

Minimiser cette énergie revient alors à modifier les poids des connexions de la manière suivante :

$$\Delta W_{kh}^{(j)} = - \mathbf{a}. \delta_{k}^{(j)} . y_{h}^{(j-1)}$$

Avec *a* le gain d'adaptation, $\delta_{K}^{(j)}$ l'erreur du neurone *k* de la couche *j* et $y_{h}^{(j-1)}$ la sortie du neurone *h* de la couche *j-1*.

Pour les neurones de la couche supérieure :

$$\delta_{K}^{(j)} = y_{k}^{(j)} - yd_{k}$$

Pour les neurones des couches internes :

$$\delta_k^{(j)} = [\sum_{i \in couche\ (j+1)} \delta_i^{(j+1)} . w_{ik}^{(j+1)}] . \sigma' p_k^{(j)}$$

Avec :

$$p_k^{(j)} = \sum_i w_{ki} . x_i$$

Où x_i correspond à la sortie du neurone *i* et :

$$\sigma p_k^{(j)} = \frac{1}{1+exp(-(p_k^{(j)}))}$$

Et σ' sa dérivée.

10.1.2. Initialisation des poids :

L'initialisation des poids avant l'application de l'algorithme d'apprentissage par rétro-propagation du gradient est importante. Cette initialisation influe sur la vitesse de convergence du réseau mais aussi sur la qualité du réseau obtenu**. [38]**

10.1.3. Temps d'apprentissage :

Le temps de convergence d'un réseau dépend de l'espace initial de représentation des poids et de l'espace final après convergence. Effectivement, plus la valeur finale des poids initiaux est proche, plus la convergence est rapide. On peut distinguer deux types de méthodes dans la littérature:

- Les méthodes d'initialisation aléatoire dans un intervalle choisi de manière adéquate.
- Les méthodes basées sur des techniques non aléatoires. **[39]**

Plusieurs recherches ont été effectuées dans le même domaine, on distingue quelques uns dans ce qui suit :

Selon Burel G et Falhman S.E, il est suggéré de sélectionner des poids de manière homogène dans un intervalle qui varie en fonction des données à étudier. Fahlman propose des intervalles variant de [-4.0, 4.0] à [-0.5, 0.5] selon les données d'apprentissage. [40]

Bottou L-Y, propose d'initialiser les poids dans un intervalle [–a√ din,a√ din] , où "a" est calculé de sorte que la variance des poids corresponde au point où la pente de la tangente de la fonction d'activation est maximum, et "din" le nombre d'unités de la couche précédente. [41]

Lee et al, ont montré théoriquement que la saturation prématurée des neurones (variations des poids trop petites pour avoir un effet sur la sortie des neurones) augmentait avec les valeurs maximales des poids. Ils en concluent que des valeurs initiales petites accroissent la vitesse de convergence mais aussi que des valeurs trop petites dégradent cette vitesse. [42]

D'autres travaux sont basés sur des schémas d'initialisation et de méthode pseudo-inverse, où seule la première couche est initialisée aléatoirement, les autres couches étant initialisées en fonction des sorties produites par la couche qui les précède. Denoeux et Lengellé, proposent une méthode basée sur des données prototypes permettant d'accélérer la vitesse de convergence des réseaux. [4]

10.1.4. Qualité du réseau résultant :

L'apprentissage par rétro-propagation du gradient peut être vu comme l'optimisation d'une fonction ayant les poids du réseau pour paramètres. Ceci mène à une convergence de la fonction vers un minimum local qui peut être aussi global. Lorsque c'est le cas, on peut considérer que l'apprentissage a été fait correctement. Lorsque ce n'est pas le cas, le réseau obtenu n'est pas optimal. Si dans la plupart des cas, la solution trouvée est très proche de la solution optimale et donc acceptable, il reste des cas où la solution est médiocre.

Des études ont été faites pour initialiser les poids de manière adéquate aux données à apprendre **[43]**. Denoeux et Lengellé, montrent des résultats plus robustes lorsque l'apprentissage est effectué avec des prototypes **[44]**.

10.1.5. Limitations de la méthode de rétro-propagation :

Malgré l'utilisation la plus répandue de l'algorithme de rétro-propagation pour l'apprentissage supervisé des perceptrons multicouches, son implantation rencontre plusieurs obstacles techniques. **[45]**. Il n'existe pas de méthodes permettant de :

- Trouver une architecture appropriée (nombres de couches, nombre de neurones).
- Choisir une taille et une qualité adéquate d'exemples d'entraînement.
- Choisir des valeurs initiales satisfaisantes pour les poids, et des valeurs convenables pour les paramètres d'apprentissage permettant d'accélérer la vitesse de convergence.
- La convergence vers un minimum local pose un problème, ce qui entrave la convergence et entraîne une oscillation de l'erreur.
- La performance de cette approche diminue rapidement en fonction de la taille (complexité).
- Basée sur une recherche de gradient, cette approche est inapplicable si des connexions sont manquantes (fonctions discontinues).

11. Conclusion

La simulation des capacités des organismes vivants à s'adapter à leur environnement par apprentissage est simplifiée par les réseaux de neurones (RNA). Plusieurs chercheurs se sont penchés sur l'association des réseaux de neurones et des algorithmes évolutionnaires. De nombreuses recherches ont été menées sur cette évolution. Il semble que l'association de ces deux métaphores biologiques donne des résultats bien supérieurs à ceux obtenus en les utilisant séparément.

Chapitre 03 :
Les Algorithmes Evolutionnaire

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. **[15]**

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*. **[16]**

« 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. **[17]**

Les principes de la ***sélection naturelle*** 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. **[11]**

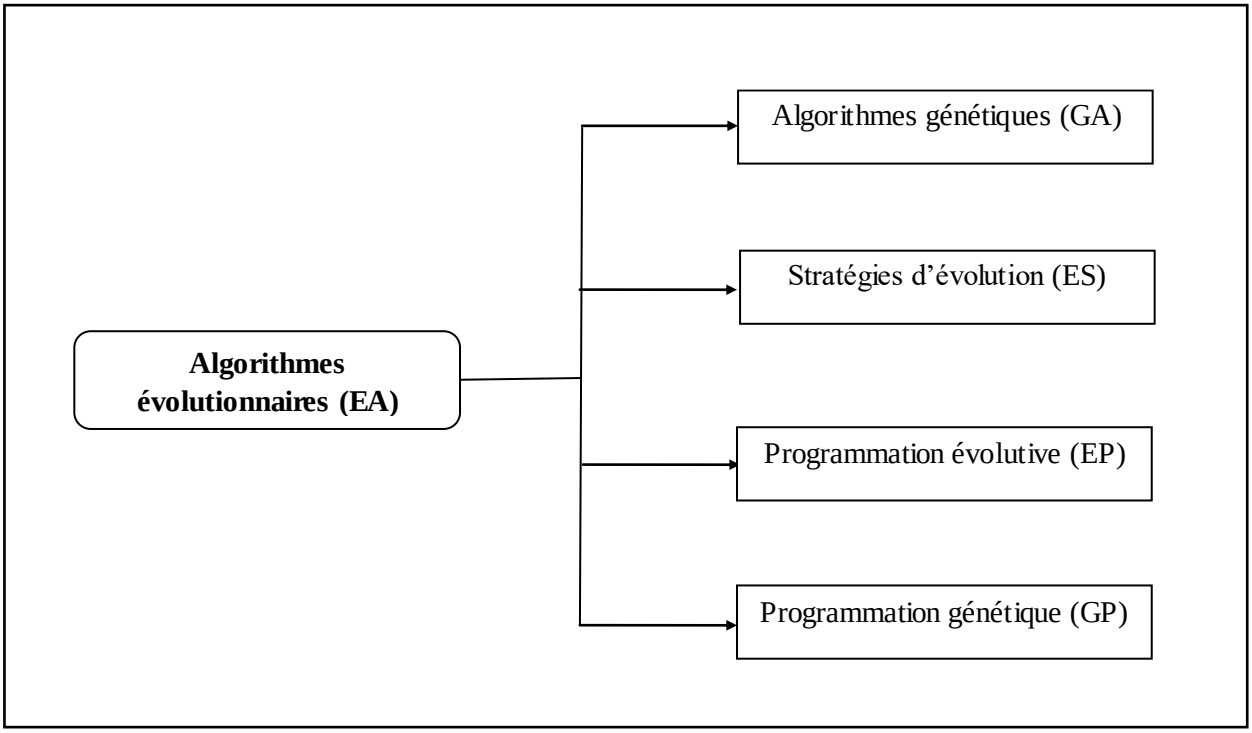


Figure 3.1 Différentes branches des algorithmes évolutionnaires.

2. 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. **[22]**

2.1. 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 :

2.1.1. 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]**

2.1.2. 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):

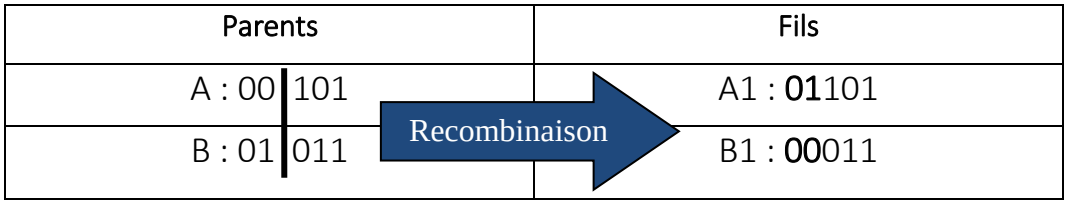


Figure 3.2 Application de l'opérateur de Croissement.

2.1.3. 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 :

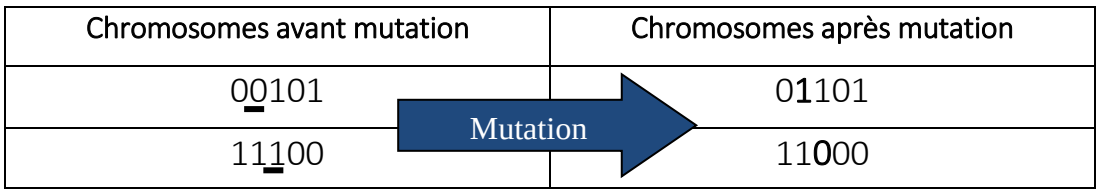


Figure 3.3 Application de l'opérateur de Mutation.

2.1.4. 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 3.1 : 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.

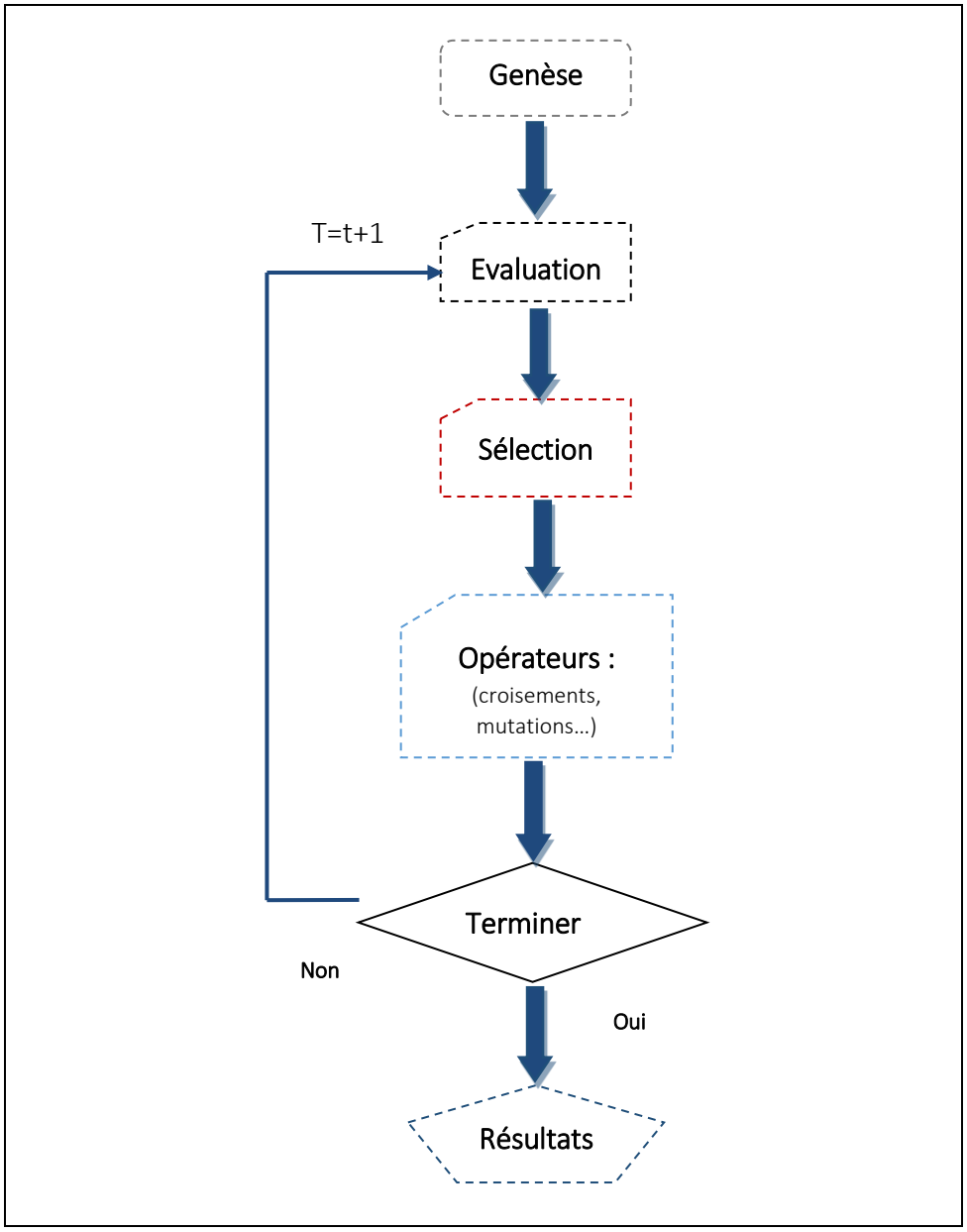


Figure 3.4 Organigramme d'un algorithme génétique.

3. 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 **[15]**. 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. **[20]**

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. **[21]**

Un EA est défini par :[1]

- **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. **[22]**

4. 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

[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.

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*.

5. 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. **[21]**

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. **[24]**

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

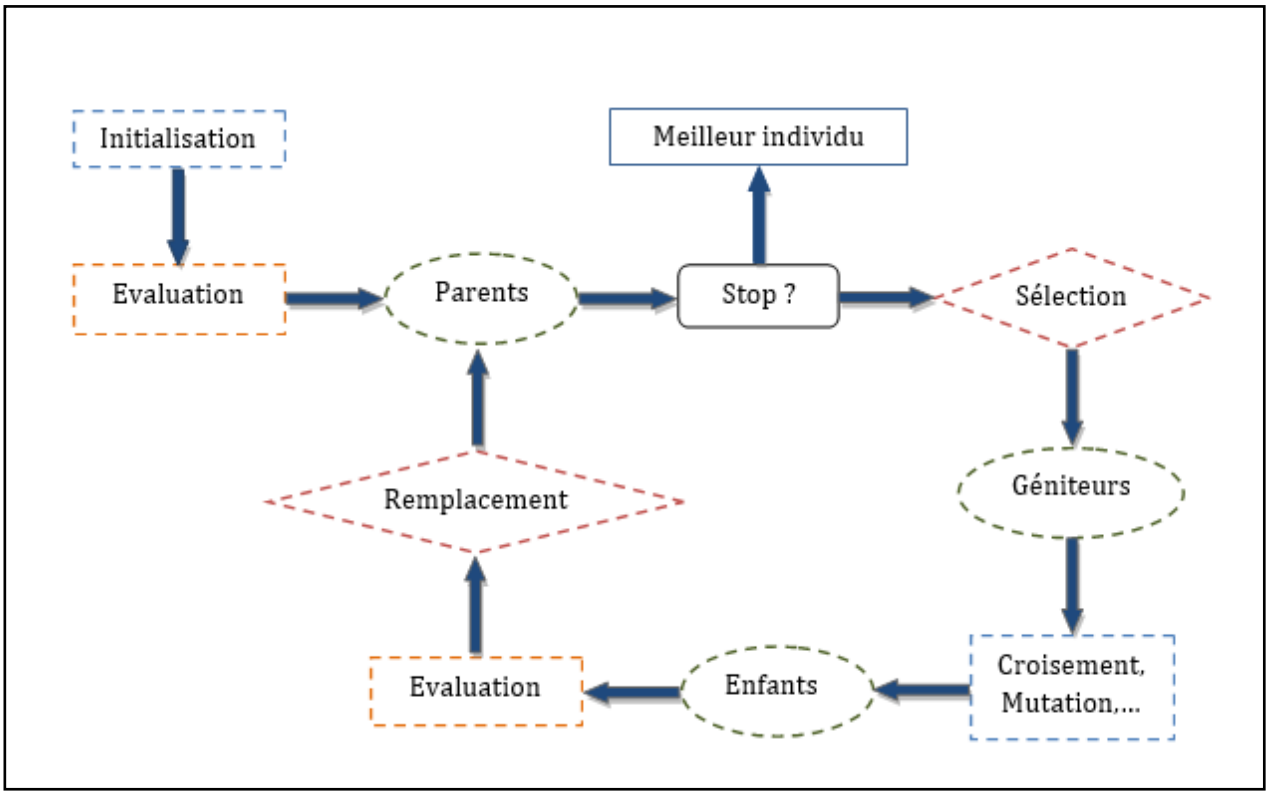


Figure 3.5 Organigramme d'un algorithme évolutionnaire.

6. Éléments de base d'un Algorithme évolutionnaire

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

6.1. 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.[52] 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. **[25]**

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. **[20]**

6.2. 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.

6.3. 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 **[20]**. 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. **[25]**

6.4. Des opérateurs

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

6.4.1. 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 $fi/\sum fj$ (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 $n\ fi/\sum fj$. Les individus sélectionnés constituent une nouvelle population. **[26]**

Les différentes méthodes de sélection :

6.4.1.1. 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*.

6.4.1.2. 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).

6.4.1.3. 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.

6.4.1.4. 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, Afin d'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.

6.4.2. 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 agit de la manière suivante :

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 3.2 : Algorithme de mutation.

6.4.3 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 :

6.4.3.1. Croisement en deux points :

On choisit au hasard deux point de croisement et on échange les parties de chaine situées entre ces deux points. **[26]**

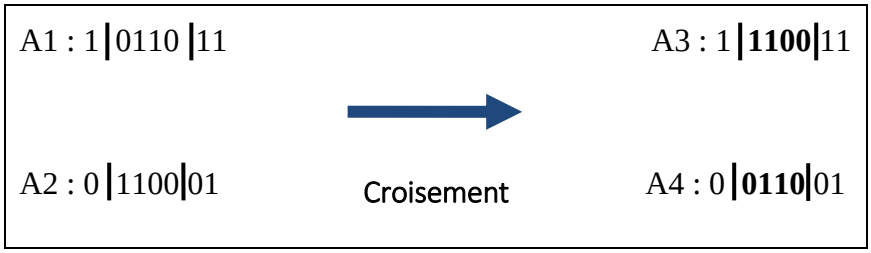


Figure 3.6 Croisement en deux points.

6.4.3.2. Croisement uniforme (multipoints) :

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.[53] Ce croisement peut être considéré comme une génération du croisement *multipoints* sans connaissance préalable du point de croisement. **[26]**

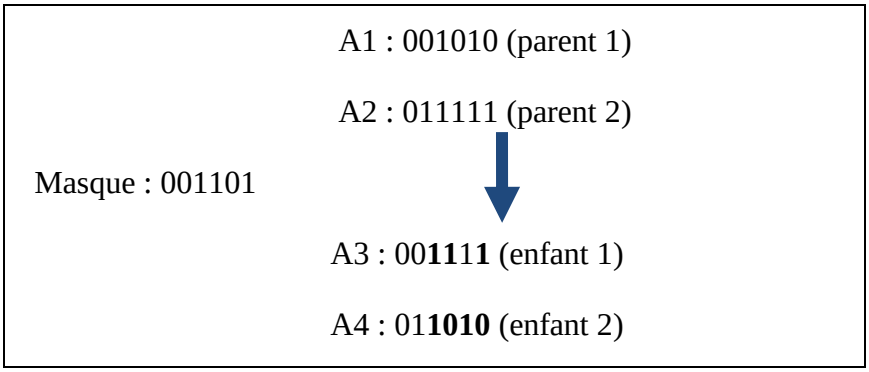


Figure 3.6 Croisement uniforme.

6.5. 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.

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. **[27]**

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 *(μ+λ)*.

- La première *(μ, λ)* : consiste à sélectionner les μ meilleurs parmi les λ enfants.
- La seconde *(μ+λ)* : 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 :

Il est rare que la recombinaison fonctionne sur le génotype contenant les variables du problème ici. Toutefois, elle paraît extrêmement bénéfique pour l'évolution des paramètres de mutation.

7.1.3. Mutation :

En raison de la réalité du codage, il y a un problème concernant la réalisation de la mutation. Les stratégies d'évolution suggèrent d'adopter un modèle fondé sur des distributions normales, avec des écarts-types qui peuvent être directement insérés dans le génotype.

En prenant en compte la représentation simplifiée (sans évolution directionnelle) mentionnée précédemment, la mutation est donc :

$$\forall i \in \{1, \ldots, k\} \quad \text{indice des gènes de } \vec{x} \text{ et } \vec{\sigma}:$$

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

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 :

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

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. **[50]**

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 exposé une vue d'ensemble de deux approches traditionnelles des EA. Nous avons exposé les principes généraux pour en examiner les particularités. Dans le prochain chapitre, nous allons examiner les bases des réseaux de neurones. Notre objectif est d'entraîner un réseau de neurones en utilisant un algorithme évolutif. Nous exposons leur approche d'apprentissage et comment l'initialisation de leurs poids influence leur temps de convergence et leurs résultats

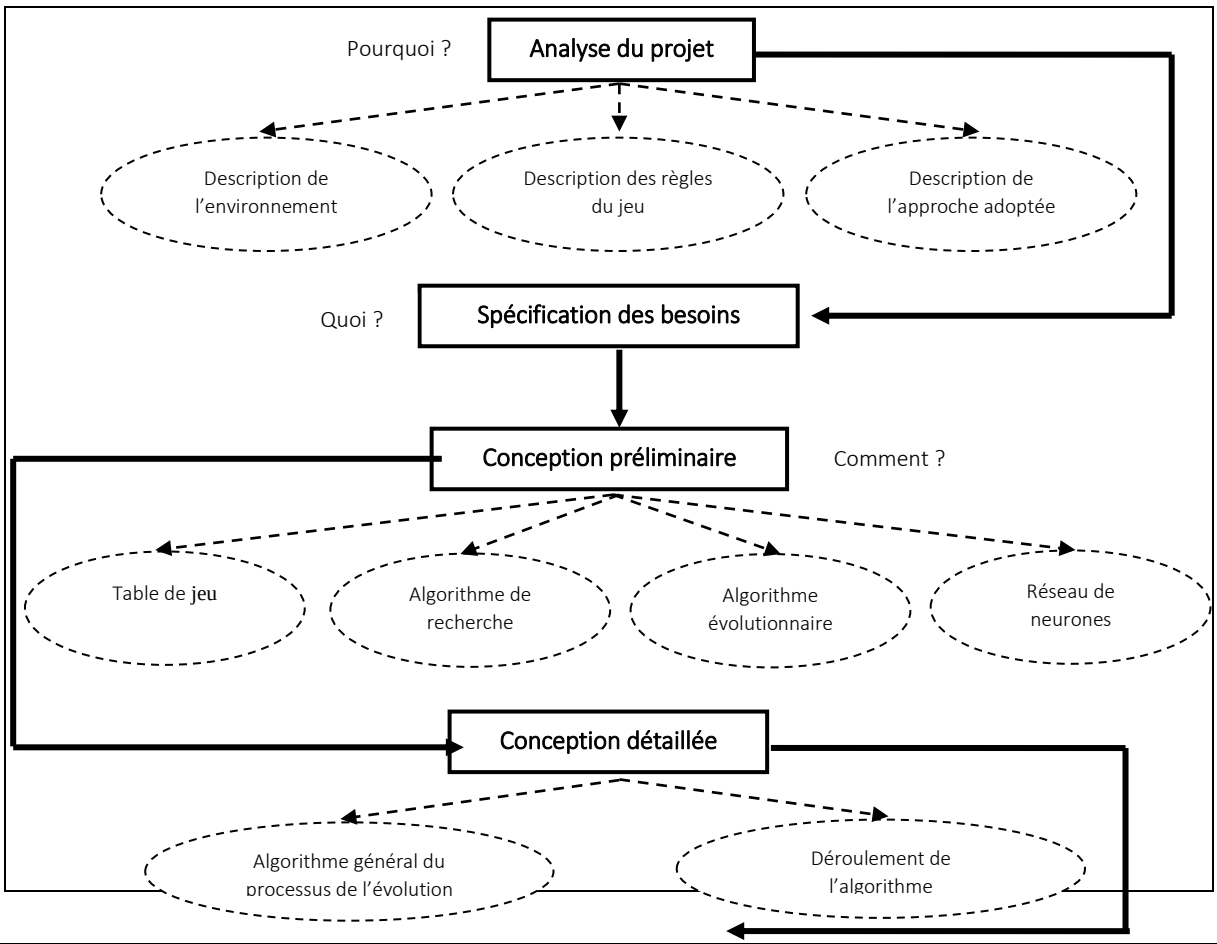
Chapitre 04 :
Analyse et Conception

1. Introduction

À partir des notions exposées dans les chapitres précédents, il est maintenant possible de faire une hybridation entre le calcul évolutionnaire et le calcul neuronal afin de concevoir une méthode efficace et optimale pour développer et améliorer les stratégies des jeux combinatoires. On a opté pour le jeu de dames comme terrain d'application de cette approche.

Cette initiative tire son inspiration des recherches antérieures de deux chercheurs américains, David B. Fogel et Kumar Chellapilla. Nous avons notamment utilisé leur article [5], qui offre une nouvelle approche pour élaborer des stratégies de jeu sans avoir besoin de connaissances préalables. Cette méthode illustre aussi les principes de la théorie évolutionnaire, en utilisant le concept de sélection naturelle où seules les stratégies les plus performantes sont sélectionnées.

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



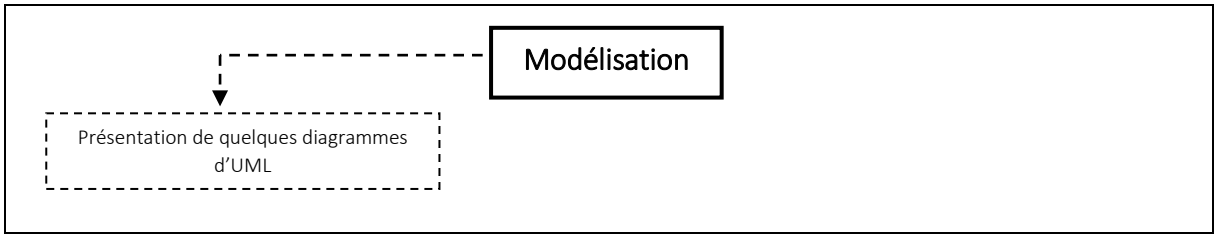


Figure 4.1 : Plan générale du chapitre de conception.

2. Analyse du projet

2.1. Description de l'environnement :

Le dispositif que nous allons élaborer constitue un cadre propice à la création et à l'amélioration des stratégies du jeu de dames. L'environnement lié au système lui permettra ainsi d'interagir avec un administrateur afin de configurer et de créer les stratégies utilisées par les divers adversaires machines, ou tout simplement avec un joueur pour jouer une partie de dames contre la machine.

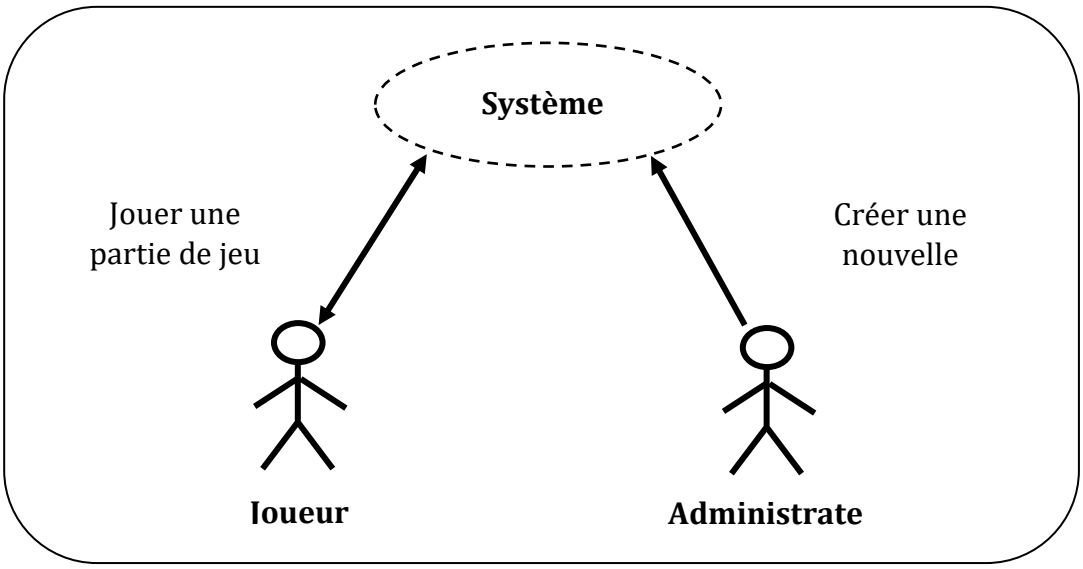


Figure 4.2 Schéma de l'environnement.

2.2. Description des règles du jeu :

2.2.1. But du jeu :

Éliminer ou immobiliser les pièces de son adversaire.

2.2.2. Matériel

On joue généralement le jeu de dame sur une table huit par huit (8x8) avec des cases de couleurs différentes.

Deux joueurs sont présents, symbolisés respectivement par les pièces noires et blanches. La figure suivante montre que chaque côté (joueur) dispose de 12 pions placés dans les 12 premières cases alternatives de la même couleur, avec la case extrême droite sur la première rangée de chaque joueur (coté) laissée ouverte, c'est-à-dire que cette case fait partie de l'ensemble de cases où on ne peut pas y placer les pièces. Comme le montre la figure suivante :

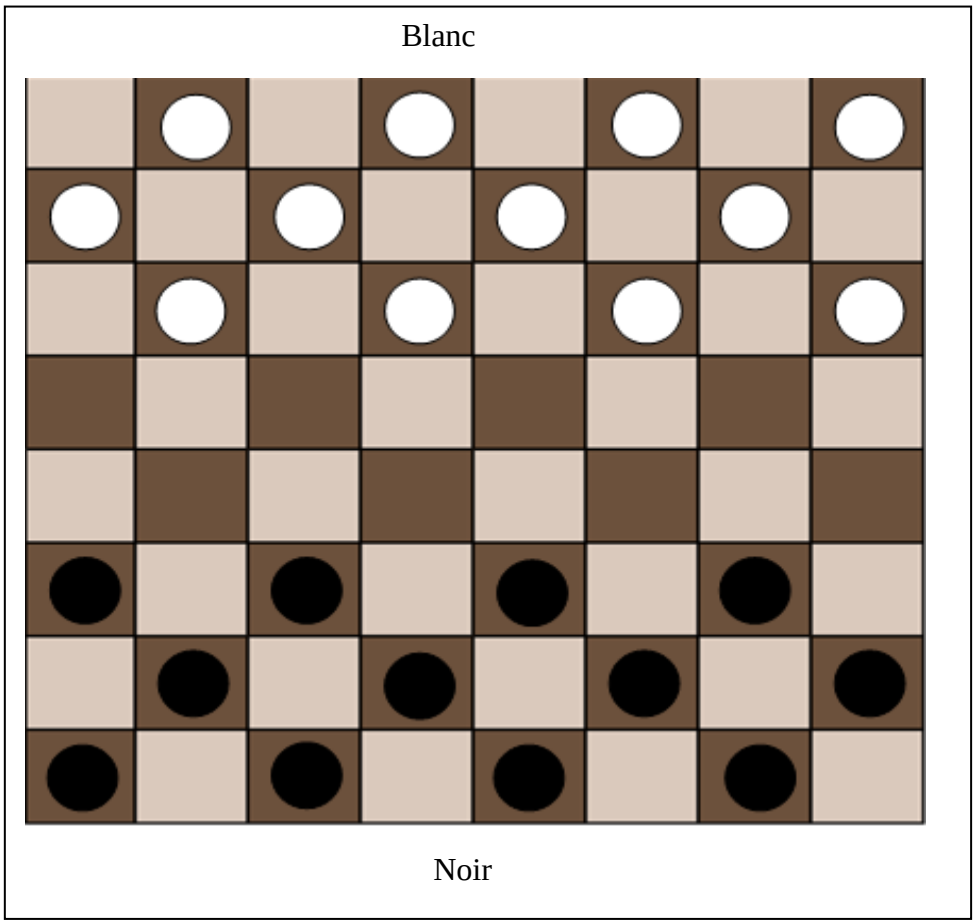


Figure 4.3 Position initiale du jeu de dames.

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 cotés. On peut faire avancer diagonalement les pièces une case à la fois.

2.2.3.2 L'enlèvement

Quand une pièce se trouve à proximité d'une pièce adverse avec une case vide derrière elle, elle doit sursauter et prendre la pièce adverse. On peut faire ce saut vers l'avant ou vers l'arrière. Après un premier saut, une pièce doit continuer à sauter jusqu'à ce qu'elle ait épuisé ses capacités. La priorité est accordée aux sauts et, en cas de sélection, le joueur préfère celui qui capture le plus de pièces adverses. La dernière rangée permet à une pièce de devenir un Roi et de pouvoir se déplacer librement dans toutes les directions.

2.2.4 Résultat du jeu

Le jeu prend fin lorsque l'un des joueurs n'a plus de mouvements, ce qui se produit généralement en retirant leur dernière pièce de la table. Cependant, cela peut aussi arriver lorsque toutes les pièces existantes sont stockées. Cela signifie que ce joueur perd sans mouvement et que l'adversaire gagne (l'objectif du jeu). Le jeu peut aussi s'arrêter à la fin de la partie.

2.3 Description de l'approche adoptée :

Ce projet s'inscrit dans le domaine de la théorie des jeux, des algorithmes évolutionnaires et des réseaux de neurones. Son objectif est de créer un environnement propice au développement et à l'amélioration des stratégies de jeu, en combinant de manière hybride les réseaux de neurones et les algorithmes évolutionnaires. L'accent est mis sur l'obtention de résultats optimaux tout en concevant un logiciel de haute qualité. L'hybridation des réseaux de neurones avec les algorithmes évolutionnaires est motivée par le constat que la recherche locale des réseaux de neurones est complémentaire à la recherche globale des algorithmes évolutionnaires. L'utilisation des réseaux de neurones dans le développement des stratégies de jeu présente un double avantage : chaque réseau de neurones représente une stratégie de jeu différente, tandis que le réseau agit comme une fonction d'évaluation pour le jeu, en analysant les positions des pièces sur la table de jeu.

3. Spécifications des besoins

Il s'agit d'une étape essentielle au commencement de chaque processus de développement. Elle a pour objectif de développer un logiciel approprié, elle vise à décrire les fonctionnalités générales 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

Pour la réalisation de ce travail, chaque table de jeu a été représentée par un vecteur de longueur 32 dont chaque composant correspond à une position disponible dans la table. Les composants dans le vecteur pourraient prendre des valeurs appartenant à l'ensemble **{-k, -1, 0, +1, +k}** Où :

- "k" est la valeur assignée pour le Roi.
- "1" est la valeur assignée pour la pièce.
- "0" est la valeur assignée pour une case vide.
- Le signe de la valeur (+/-) indique si la pièce en question appartient au joueur (positif) ou à l'adversaire (négatif).

Exemple : Au début du jeu, le vecteur initial sera comme suit :

(-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0 , 0, 0, 0, 0, 0, 0, 0, +1, +1, +1, +1, +1, +1, +1, +1, +1, +1, +1, +1)

Ainsi, chaque table a été découpée en sous sections (nxn) dans le but de fournir des informations spatiales d'adjacence ou de proximité, c'est-à-dire si deux cases sont des voisines, prés l'une l'autre, ou distantes.

On peut donc construire :

- **36** sous sections de (3x3).
- **25** sous sections de (4x4).
- **16** sous sections de (5x5).
- **09** sous sections de (6x6).
- **04** sous sections de (7x7).
- **01** seule section de (8x8).

Donc il ya en totale 91 sous sections possibles qu'on peut construire à partir de la table. La figure suivante montre des échantillons de (3x3), (4x4) et (5x5) sous sections carrées :

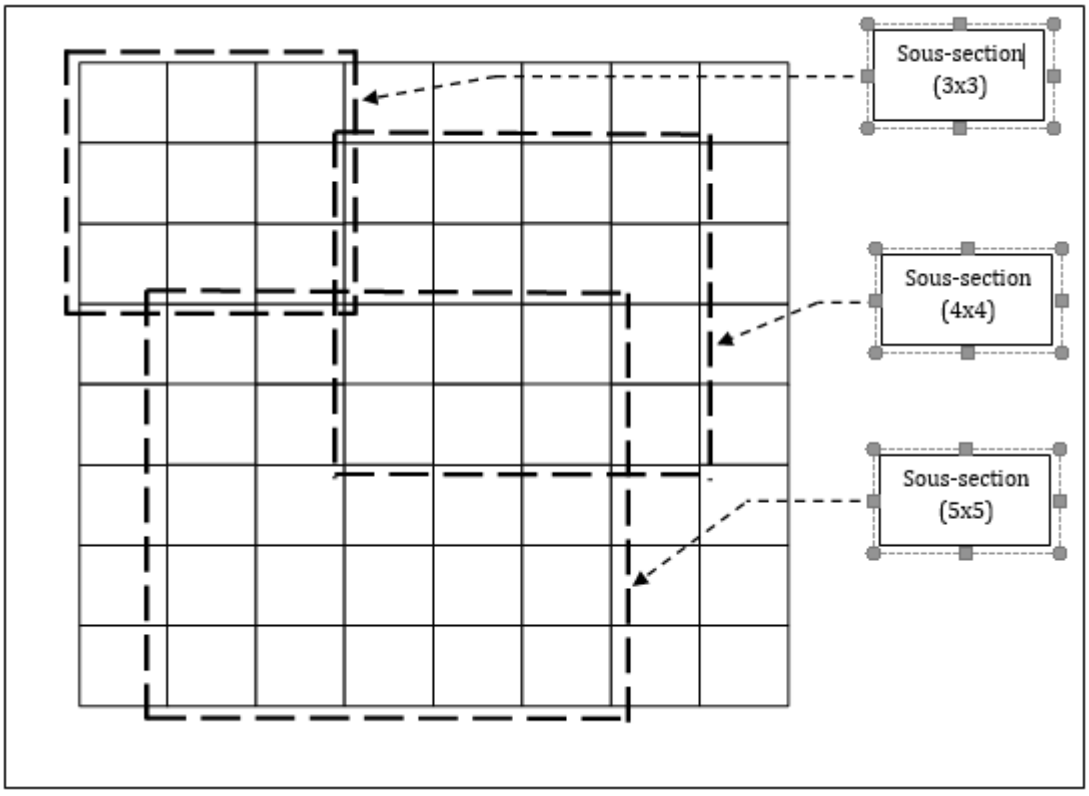


Figure 4.4 Découpage de la table d'un jeu de dame en sous sections.

4.2 Réseau de neurones

Trouver l'architecture adéquate d'un réseau de neurones à un problème donné n'est pas une chose facile. Trouver le nombre optimal de couches cachées ainsi que le nombre de neurones dans chaque couche et leurs connexions se fait plus de manière empirique que par une méthode basée sur un fondement théorique. **[4]**

Les réseaux de neurones auxquels nous nous intéressons sont des *perceptrons multicouches avec apprentissage par rétro-propagation du gradient.* En l'occurrence, nous nous sommes attachés à étudier des réseaux de neurones composés des couches suivantes :

- Une couche d'entrée.
- Trois couches cachées.
- Une couche de sortie contenant un seul neurone.

La deuxième, la troisième couche cachée et la couche de sortie ont une structure *entièrement reliée,* c'est-à-dire tous les neurones d'une couche sont reliés à tous les neurones de la couche suivante, tandis que les connexions de la première couche cachée étaient spécialement conçues pour capturer l'information spatiale de la table du jeu. La *fonction de transfert* appliquée sur chaque nœud cachée et de sortie était la tangente hyperbolique, (*tanh,* limitée entre [-1, +1]).

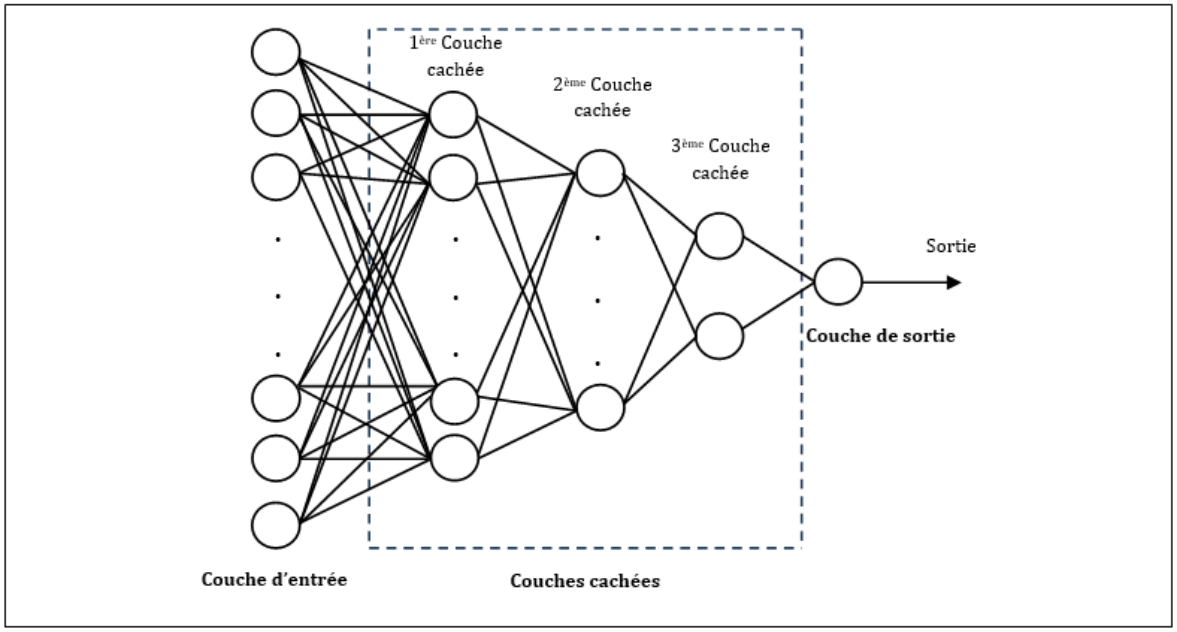


Figure 4.5 Architecture du Réseau de neurones.

En fait, les efforts précédents ont utilisé pour traiter les entrées de la table un réseau de neurones avec deux couches cachées comportant 40 et 10 nœud respectivement. Ainsi, la table (8x8) a été interprétée simplement en tant que vecteur (1x32), et le réseau de neurones a été forcé pour apprendre toutes les caractéristiques spatiales de la table afin d'évaluer le jeu. **[5]**

Pour ne pas handicaper la procédure d'apprentissage de cette manière, le réseau de neurones utilisé ici a mis en application une série de 91 nœuds de prétraitement qui recouvrent les (nxn) sous sections de la table.

Ainsi, pour permettre au réseau de neurones de générer les caractéristiques spatiales de la table entière du jeu depuis les sous sections qui pourrait par conséquent être traité dans les couches cachées suivante, les 36 sous sections possibles de **(3x3)** ont été fournies comme entrée aux 36 premiers nœuds (neurones) dans la première couche cachée.

Les 25 sous sections possibles de **(4x4)** ont été assignées aux 25 prochains nœuds dans cette couche, et ainsi de suite. En fin de compte, on obtiendra 91 sous sections qui ont été données comme entrée aux 91 nœuds de la première couche cachée.

Le schéma suivant montre la structure du réseau de neurones :

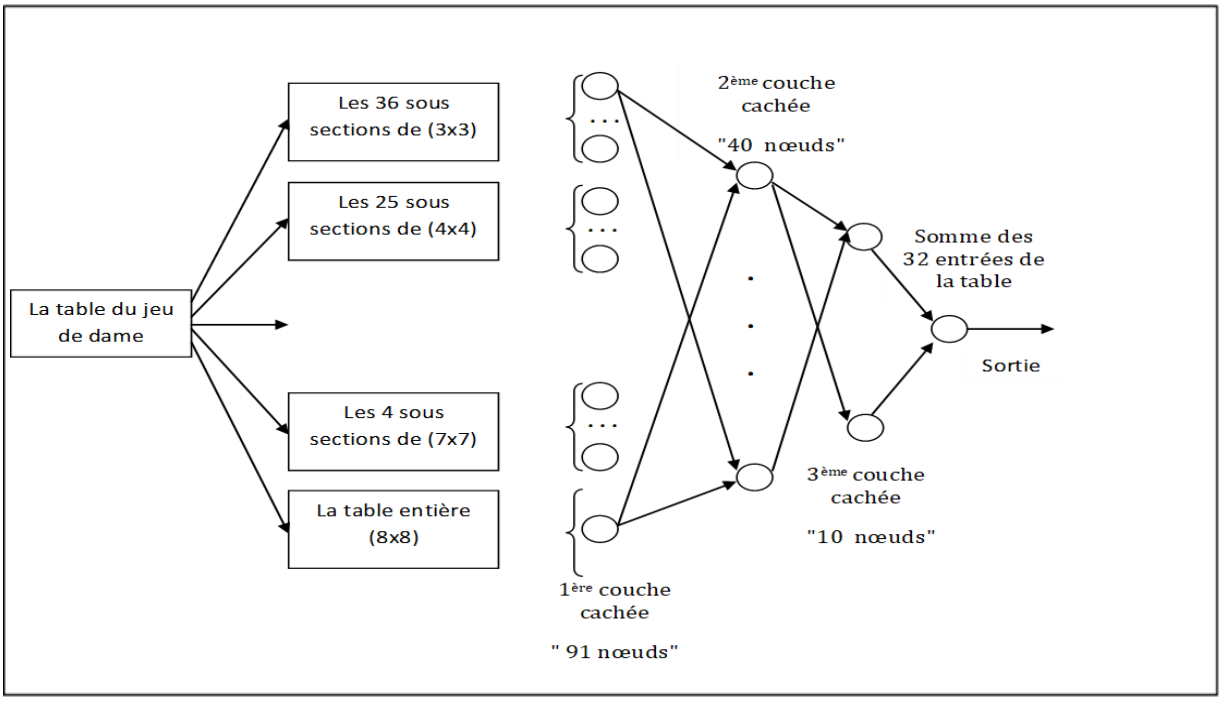


Figure 4.6 Structure du réseau de neurones

Remarque **:** Il est important de noter immédiatement que nous n'avons pas tenté d'offrir au réseau de neurones des caractéristiques utiles comme entrées.

4.3 Algorithme évolutionnaire

Dans le troisième chapitre intitulé "Les algorithmes évolutionnaires : principes et méthodes" nous avons exposé deux méthodes traditionnelles des EA. Nous avons examiné leurs caractéristiques pour déterminer une méthode appropriée pour la réalisation de notre projet. Nous avons opté pour les stratégies d'évolution.

Les travaux sur la représentation d'un réseau connexionniste dans un génotype ont été nombreux. Au point culminant, le réseau est entièrement codé dans le génotype, avec chaque poids codé de manière précise. Dans cette situation, les stratégies d'évolution se limitent à un problème classique d'optimisation multicritères.

4.4 Algorithme de recherche

Les algorithmes de recherche dans les jeux ont pour objectif d'évaluer les différentes actions à réaliser pour un joueur spécifique, puis de donner le meilleur retour. Cela signifie que le joueur doit connaître tous les coups jouables en créant ce qu'on appelle un arbre de jeu. Dans cette perspective, et afin de mener à bien notre projet, nous avons employé l'algorithme de recherche qui est exposé dans le premier chapitre intitulé « Introduction à la théorie des jeux », à savoir le Fail-Soft Alpha-Bêta.

5. Conception détaillée

Cette partie traite de l'application des algorithmes évolutionnaires sur les réseaux de neurones pour la création de nouvelles stratégies de jeu.

5.1 Algorithme général du processus d'évolution

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

Initialisation :

- Choisir 15 Stratégies (réseaux de neurones) en utilisant la distribution uniforme dans l'intervalle [-0.2, 0.2] pour déterminer les poids de chaque stratégie (5046 poids), les stratégies sont notées pi, i=1..15 et la valeur du roi est initialement fixée à 2.0

- Chaque stratégie à un vecteur adaptatif σi, i=1..15 chaque élément de ce vecteur correspond à un poids ou seuil et serve à contrôler le pas pour une nouvelle mutation (chaque vecteur se compose de 5046 éléments).
- Initialiser les éléments de ces 15 vecteurs à 0.05
- Application de l'opérateur de mutation sur chaque réseau de neurones pour générer une nouvelle population de 30 individus (stratégies).

Déroulement :

Répéter

Pour chaque stratégie initialiser leurs points totaux de jeu à 0 ;

Pour chaque stratégie choisir aléatoirement 5 adversaires ;

Pour chaque adversaire commencer le jeu avec lui (le joueur).

Comment jouer :

Le jeu se déroule en utilisant l'algorithme de recherche Fail-Soft Alpha-Beta.

Répéter

```
Mettre profondeur à 4 ;
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; // C'est pour libérer l'espace
      mémoire
      Si (score >= current) {
         current = score;
         MeilleurMouvement = m;
         Si (score >= alpha){
```

```
                alpha = score;
            MeilleurMouvement = m ;
            Si (score >= bêta)
               break;
         }
      }
   }
   return current;
}  // Fin de l'alpha-bêta fail-soft
```

Remarque :

« Mouvement » signifie d'établir s'il y a de mouvement obligatoire, si c'est le cas déterminer le maximum de mouvements successifs obligatoires et les organiser dans une liste et augmenter la profondeur par le plus petit chiffre paire supérieur ou égale le nombre de mouvements successifs, sinon (c'est-à-dire il n'y a pas de mouvement obligatoire) alors établir les mouvements normaux.

Effectuer le meilleur mouvement qui entraîne le changement de la table, donc reste la réponse de l'adversaire

```
// Réponse de l'adversaire
Mettre profondeur à 4 ;
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; // pour libérer l'espace mémoire
      Si (score >= current) {
```

```
            current = score;
            MeilleurMouvement = m;
            Si (score >= alpha){
               alpha = score;
               MeilleurMouvement = m ;
               Si (score >= bêta)
                  break;
              }
           }
        }
        return current;
     }
     Until jeu est terminé
   Fin pour
 Fin pour
```

- Classer les différentes stratégies selon leur gain dans les différentes parties jouées.
- Sélectionner les 15 premières stratégies.
- **Pour chaque** stratégie créée une progéniture en appliquant l'opérateur de mutation selon les règles décrites précédemment :

  σ'i(j) = σi(j) . Exp (τ . Nj(0, 1)), j=1,…, Nw.

  W' i(j) = W i(j) + σ'i(j)Nj(0,1), j=1,…, Nw.

Jusqu'à obtenir une meilleure stratégie.

Algorithme 4.1 Algorithme général de l'application.

- **Description de la fonction eval ( ) :**

Quand la table du jeu est exposée au réseau de neurones pour l'évaluation, sa sortie scalaire a été interprétée comme une valeur (représentante) de cette table en fonction de la position du joueur dont les pièces ont été marquées par des valeurs positives. Plus l'évaluation de l'entrée (la table du jeu) est proche de +1, plus elle est élevée. De la même manière, lorsque la sortie est proche de -1, la table devient moins bonne. La valeur exacte de +1 a été attribuée à toutes les

positions qui étaient des victoires pour le joueur (par exemple, aucune pièce d'oppositions restantes), et la valeur de -1 a été attribuée à toutes les positions qui étaient des pertes.

5.2 Déroulement de l'algorithme :

Chaque nouvelle génération de joueurs sera définie par leur réseau de neurones qui lui est associé, ce qui constitue une stratégie de jeu. La structure générale de tous les réseaux de neurones utilisés est la même que celle décrite dans la figure 4.6. L'algorithme se déroule en quatre étapes qui seront présentées ci-après :

5.2.1 Choisir une population initiale :

Une population de 15 stratégies (réseaux de neurones), **Pi ; i=1…15**, définie par les **poids** et les **seuils** pour chaque réseau de neurones et la valeur associée de la stratégie de **k**, est créée au hasard (figure 4.7).

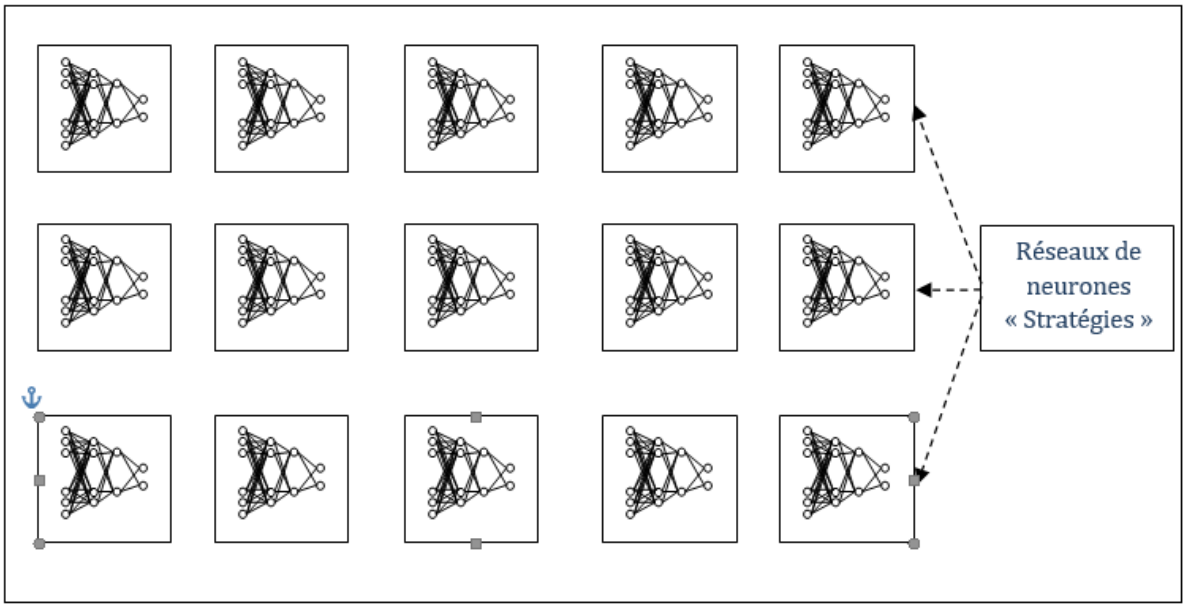


Figure 4.7 Population initiale contient 15 Réseaux de neurones.

Les poids et les seuils sont générés en utilisant la distribution uniforme entre **[-0.2, 2.0]**, avec la valeur initiale de l'ensemble **k** est **2**.

Chaque stratégie a eu un vecteur de paramètre auto-adaptatif associé **σi, i=1,…,15** où chaque composant a correspondu à un poids ou à un seuil et a servi à contrôler la taille de l'étape de la recherche de nouveaux paramètres mutés du réseau de neurones.

Pour être conformé à l'intervalle de l'initialisation, les paramètres auto-adaptatifs pour les poids et les seuils ont été initialisés à **0.05.**

5.2.2 Génération des progénitures « Mutation »:

Chaque parent a généré une stratégie progéniture (fils) en changeant tous les poids et les seuils associés (figure 4.8), et probablement la valeur de **k** aussi bien.

$$\sigma'_i(j) = \sigma_i(j) \cdot Exp(\tau \cdot Nj(0, 1)), \quad j=1,\ldots, Nw.$$

$$W'_i(j) = W_i(j) + \sigma'_i(j) Nj(0,1), \quad j=1,\ldots, Nw.$$

Où : Ajustement du vecteur poids et vecteur seuil

- **Nw:** est le nombre de poids et de seuils dans le réseau de neurones (ici c'est **5046**).
- $\tau = 1/sqrt(2sqrt(Nw)) = 0.0839.$
- **Nj(0,1) :** est une variable aléatoire gaussienne standard recalculé pour chaque **j.**
- La valeur **k'** du Roi fils a été obtenue par : $K'_i = K_i + d$, Où **"d"** a été choisi uniformément au hasard de **{-0.1, 0, 0.1}.**
- Pour la convenance, la valeur de **k'j** se situe dans l'intervalle **[1.0, 3.0]**.

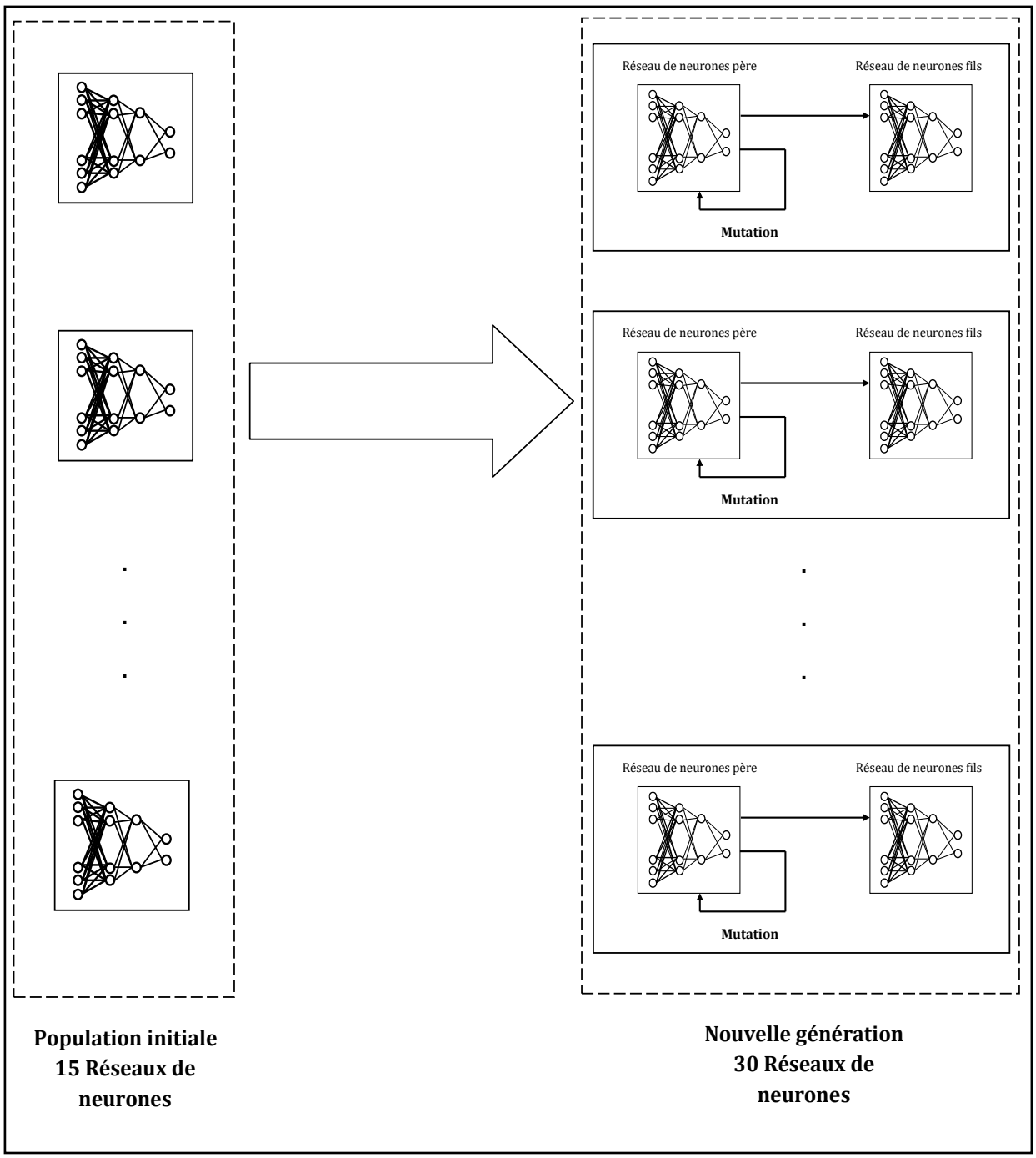


Figure 4.8 Application de l'opérateur de Mutation sur une population de 15 réseaux de neurones.

Tous les parents et leurs fils ont concurrencés pour la survie en jouant des parties de jeu de dames et en recevant des points pour leurs résultats du jeu. Pour chaque joueur on a associé cinq adversaires, qui sont choisis aléatoirement de la même population (figure 4.9).

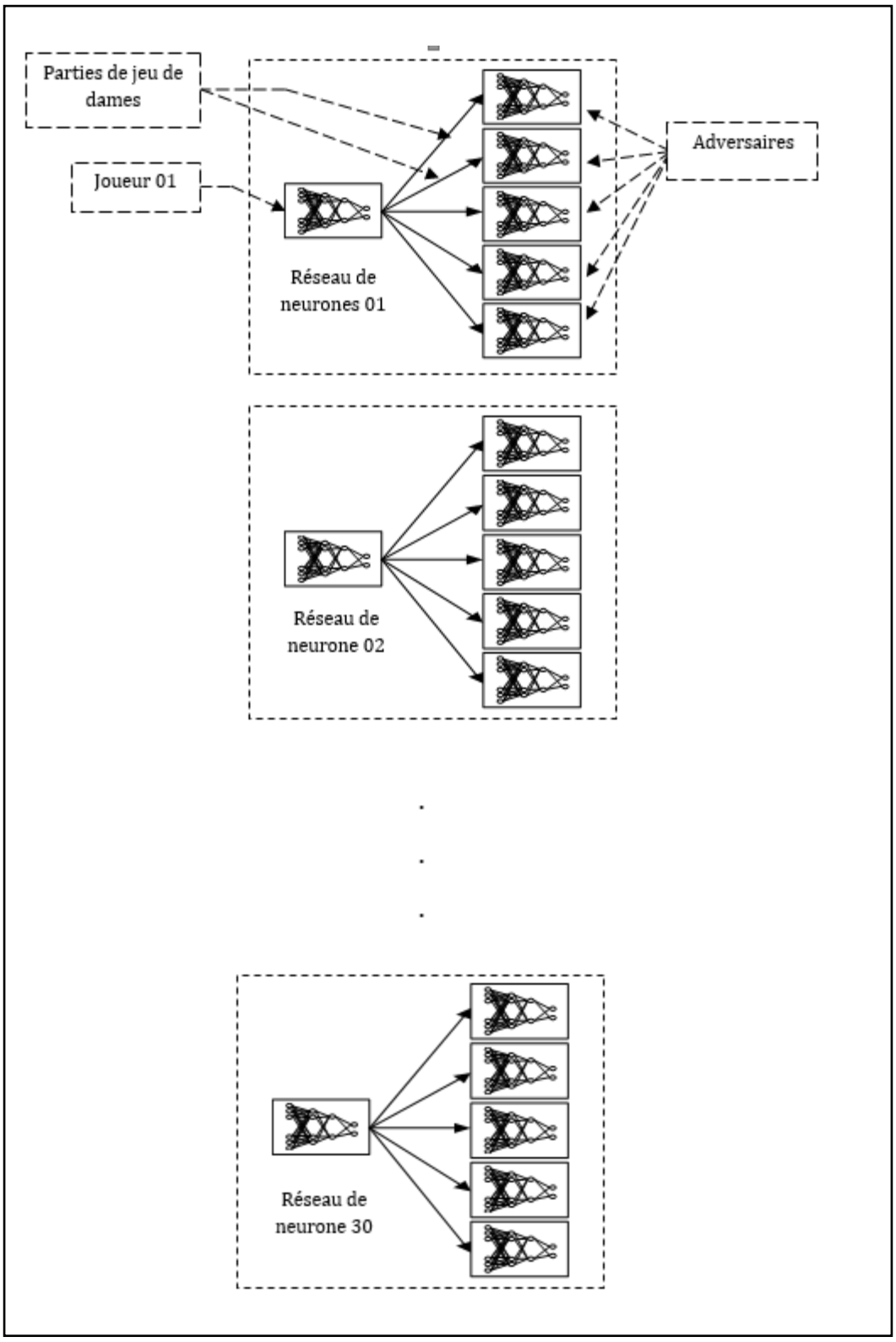


Figure 4.9 Chaque réseau de neurones joue une partie de jeu de dame contre 5 autres réseaux de neurones de la même génération.

Le joueur a toujours utilisé les pièces noires dans chacun de ces cinq jeux, tandis que l'adversaire qui a été choisi au hasard a toujours utilisé les pièces blanches. Au cours de chaque match, le joueur a obtenu -2, 0, ou +1 Points en fonction de sa **défaite**, de sa partie **nulle** ou de sa v**ictoire** dans le jeu. De manière similaire, chaque adversaire a également inscrit -2, 0, ou -1 point en fonction des résultats.
Ces valeurs étaient légèrement fluctuantes, mais elles ont été représentatives d'un protocole habituellement raisonnable pour que la perte soit deux fois plus coûteuse qu'une gaine. Il y avait un total de 150 jeux par génération, chaque stratégie jouant en moyenne 10 jeux.

5.2.3 Déroulement des jeux :

Chaque jeu est joué en appliquant l'algorithme de recherche **Fail-Soft Alpha-Bêta** sur l'arbre du jeu construit au début de la partie, à partir des positions initiales des pièces et les positions futures qu'on peut atteindre pour chaque joueur. L'intérêt de l'utilisation de cet algorithme est de choisir la stratégie de jeu appropriée à suivre pour chaque joueur en évaluant les différents coups possible pour chacun d'eux (réseau de neurones).

La profondeur de la recherche ***d*** est fixée par 4 niveaux, pour tenir compte des temps d'exécution raisonnables. En outre, quand les mouvements obligatoires sont appliqués, la profondeur de recherche est prolongée (on met ***f*** le nombre de mouvements obligatoires), parce que dans ces situations le joueur n'a aucune vraie décision à faire. Cette prolongation est faite par deux niveaux, jusqu'au plus petit chiffre pair qui est supérieur ou égale au nombre de mouvements obligatoires ***f*** qui se sont produits dans cette branche.

Exemple **:** si on se tombe sur 2 mouvements obligatoires, la prolongation sera faite en 4 niveaux.

Le meilleur mouvement à jouer est choisi itérativement en minimisant ou bien en maximisant depuis les feuilles de l'arbre de jeu à chaque profondeur selon que ce dernier correspond au mouvement de l'adversaire ou au mouvement du joueur, respectivement.

5.2.4 La sélection des meilleurs individus :

Après la fin de tous les jeux, les 15 stratégies qui ont obtenu le plus grand nombre de points ont été **sélectionnées** et seront les parents de la génération suivante. Par la suite, cette nouvelle génération rappellera le processus d'évolution.

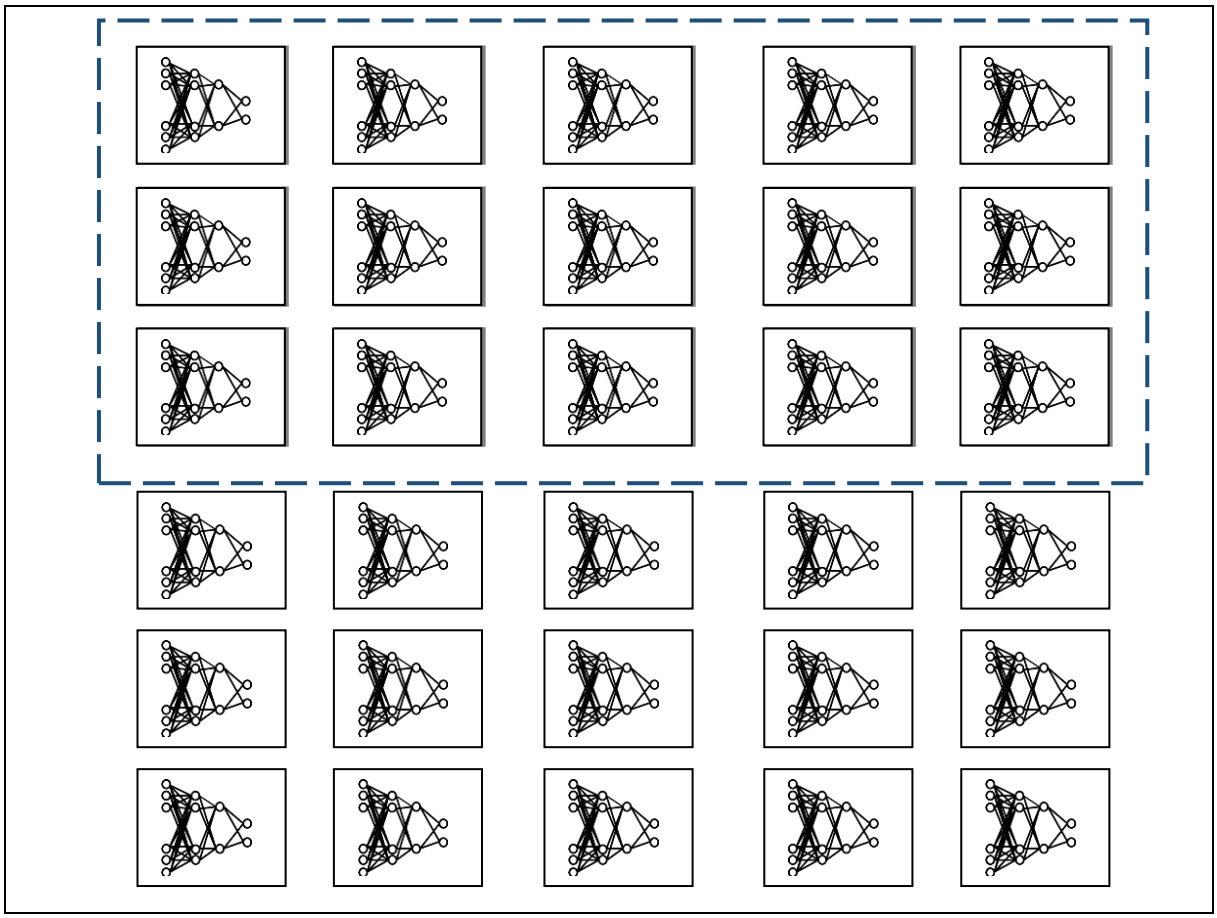


Figure 4.10 Application de l'opérateur de sélection sur les 15 premiers réseaux de neurones.

6. Analyse Comparative des Méthodologies Employées dans Notre Étude par Rapport aux Recherches Précédentes

Critère	Travaux de Chellapilla et Fogel	Notre Travail
Algorithmes Principaux	Algorithmes génétiques et réseaux neurones	Algorithmes évolutionnaires et réseaux neurones
Algorithme de Recherche	Algorithme de recherche alpha-beta et réseaux neuronaux pour évaluation	Algorithme de recherche fail-soft alpha-beta et hybridation neuronale
Objectif des Algorithmes	Optimisation inspirée de l'évolution biologique et amélioration des stratégies de jeu par apprentissage neuronal	Optimisation inspirée de l'évolution biologique et intégration du calcul neuronal pour une optimisation plus efficace

Spécificité des Algorithmes	Sélection, croisement, mutation ; réseaux neuronaux pour évaluation	Sélection, croisement, mutation ; réseaux neuronaux pour évaluation et optimisation des poids
Traitement des Évaluations	Limité par les bornes alpha et beta (coupure rigide) ; réseaux neuronaux pour des évaluations plus précises	Tolérance des valeurs légèrement hors des bornes (Fail-soft) ; évaluations précises grâce aux réseaux neuronaux
Efficacité du Temps de Calcul	Moins optimisé pour certaines configurations ; amélioration grâce à l'apprentissage neuronal	Optimisation significative du temps de traitement grâce au fail-soft et aux réseaux neuronaux
Performance dans la Recherche	Recherche efficace mais avec coupures strictes ; amélioration continue grâce à l'apprentissage neuronal	Recherche plus flexible et rapide grâce au fail-soft et aux évaluations neuronales
Adaptabilité et Flexibilité	Adaptabilité limitée par la rigidité de l'algorithme alpha-beta ; haute adaptabilité grâce à l'apprentissage neuronal	Meilleure adaptabilité grâce à la tolérance des valeurs fail-soft et l'apprentissage neuronal continu

En résumé, je distingue mon approche par une meilleure optimisation et flexibilité, particulièrement en termes de temps de traitement et d'efficacité de la recherche. En combinant les avantages des algorithmes évolutifs, du calcul neuronal et de l'algorithme fail-soft alpha-beta, j'ai ainsi amélioré les stratégies de jeux combinatoires de manière significative par rapport aux travaux de David Fogel et Chellapilla.

7. Modélisation

La modélisation a pour objectif principal de proposer une approche préalable pour diminuer la complexité du système étudié lors de la conception et organiser la réalisation du projet en définissant les différents modules et étapes. Différentes méthodes de modélisation sont employées. Dans notre travail, nous utilisons une approche objet qui repose 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 : [47]

En 1994, UML a été développé chez Rational Software Corporation sous l'impulsion de Grady Booch et James Rumbaugh. Il est né de l'effort visant à unifier et à standardiser trois méthodes de modélisation dominantes des années 90 : OMT, Booch et OOSE.

En 1997, UML 1.1 a été standardisé par l'OMG (Object Management Group) en réponse à une demande de collaboration entre plusieurs entreprises, dont 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, UML a continué d'évoluer, avec plusieurs versions successives. La version actuelle est UML 2.5.1, publiée en 2017. Au fil des ans, de nombreuses améliorations et extensions ont été apportées au langage de modélisation UML.

7.3. Objectifs de l'UML : [48]

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 : [49]

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 :

7.5.1. Diagrammes des cas d'utilisation :

- **Administrateur**

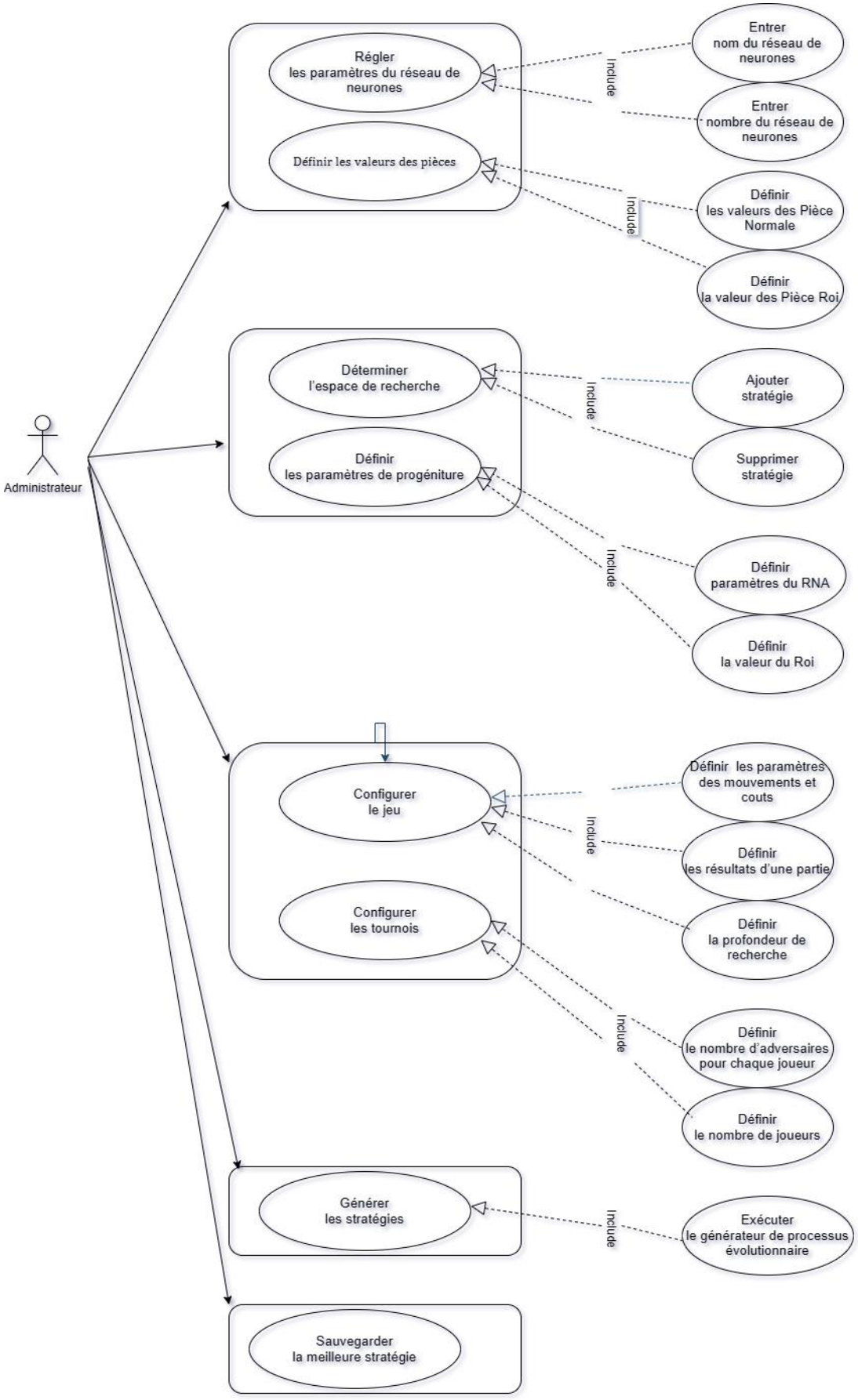


Figure 4.11 Diagramme de cas d'utilisation « Administrateur ».

- **Joueur**

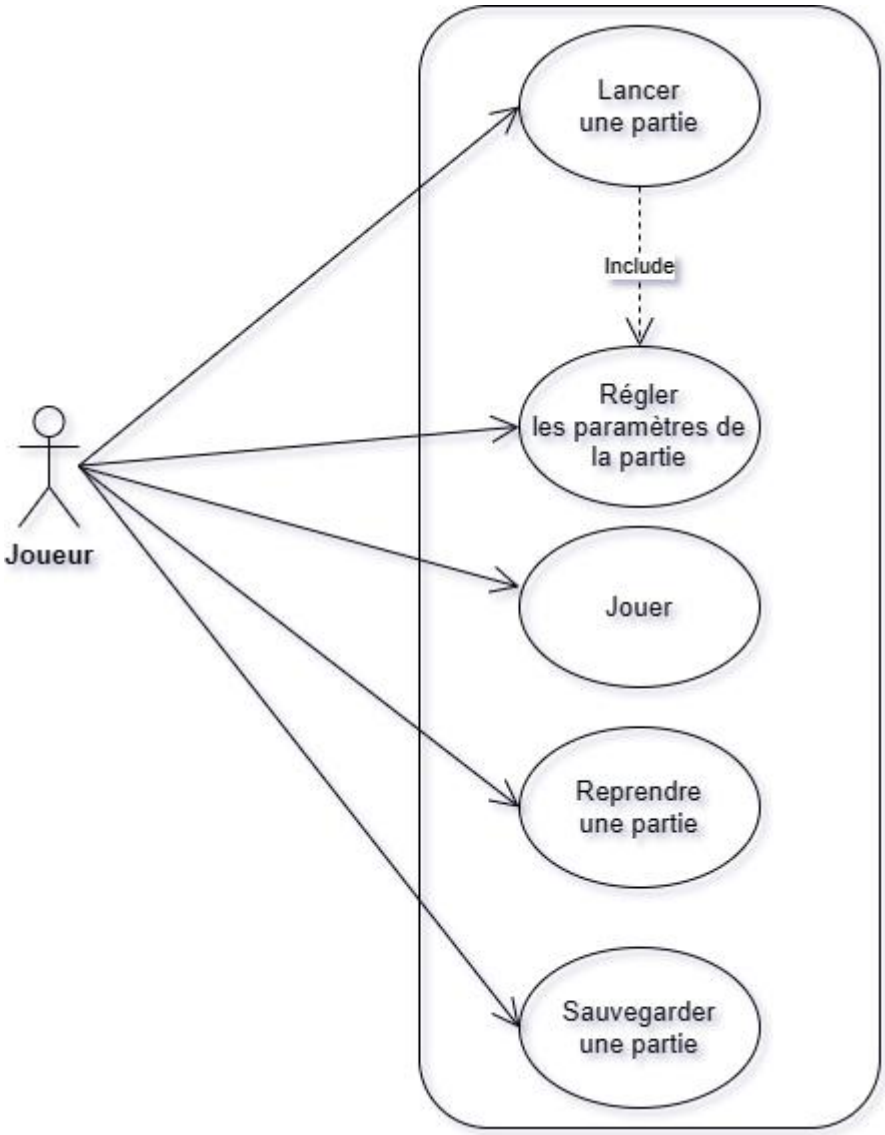


Figure 4.12 Diagramme de cas d'utilisation « Joueur».

7.5.2 Diagramme de classe

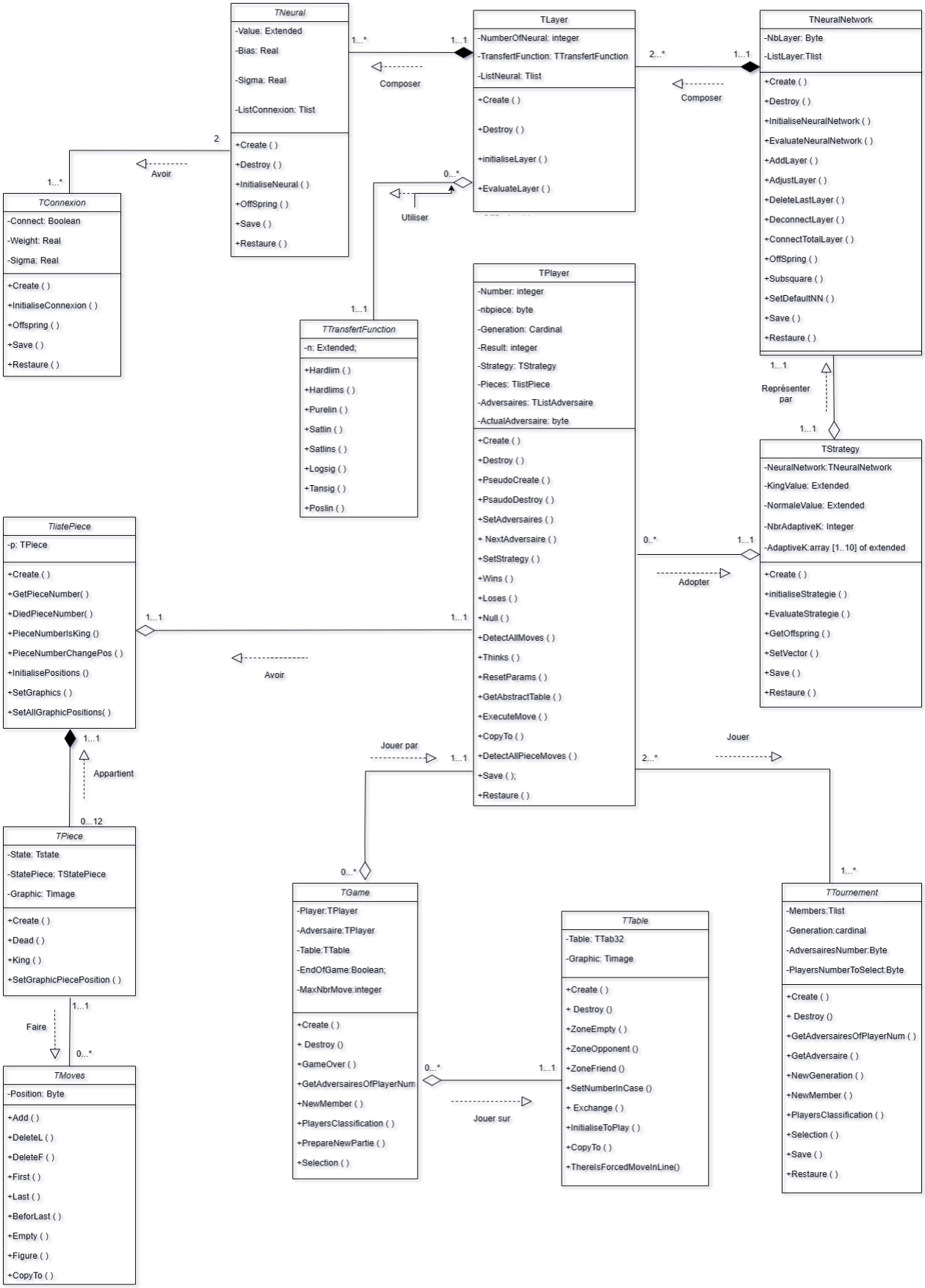


Figure 4.13 Diagramme de classes : UML.

7.5.3. Diagrame de séquence

Administarateur

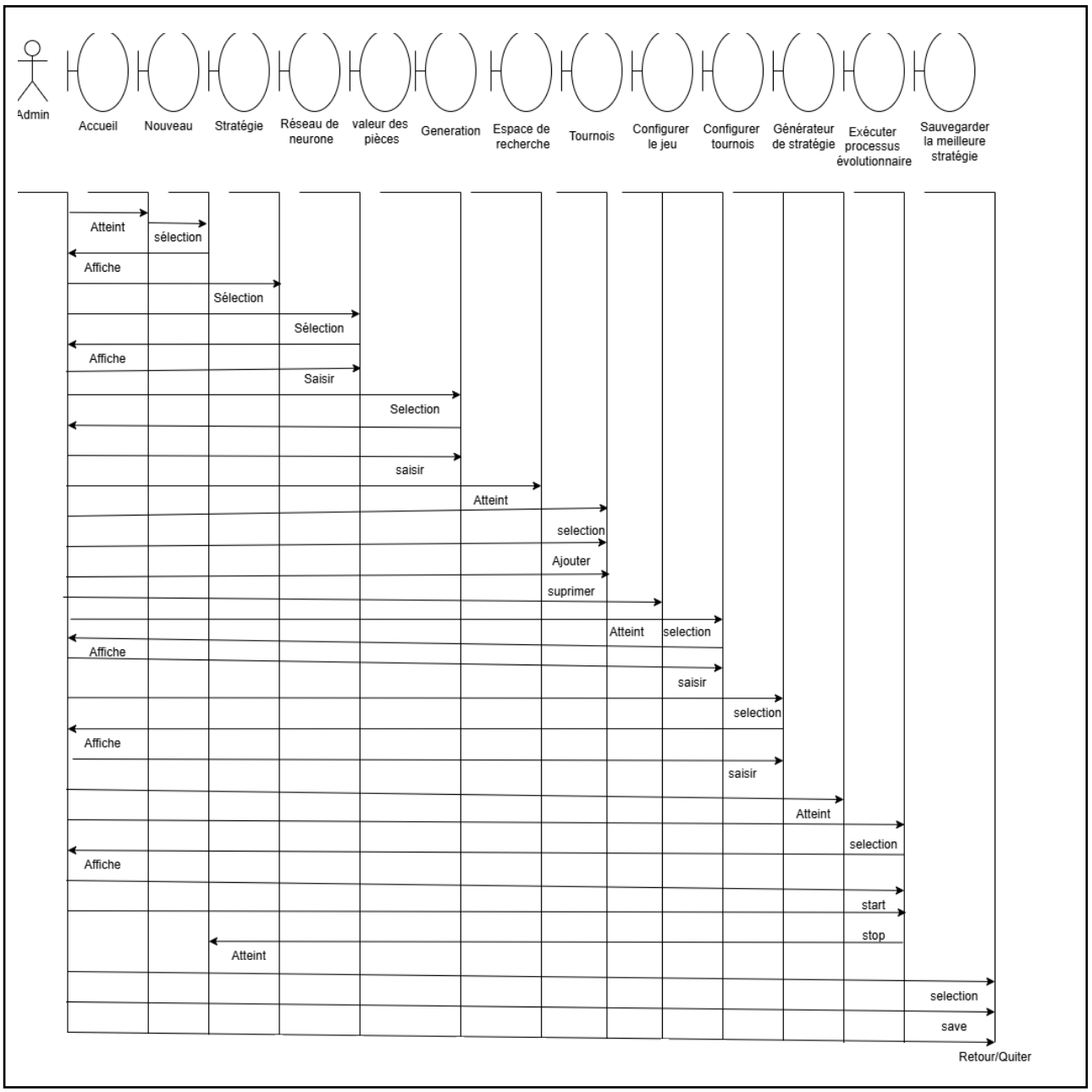


Figure 4.14 : Diagramme de séquence 'Créer une Stratégies'

Joueur

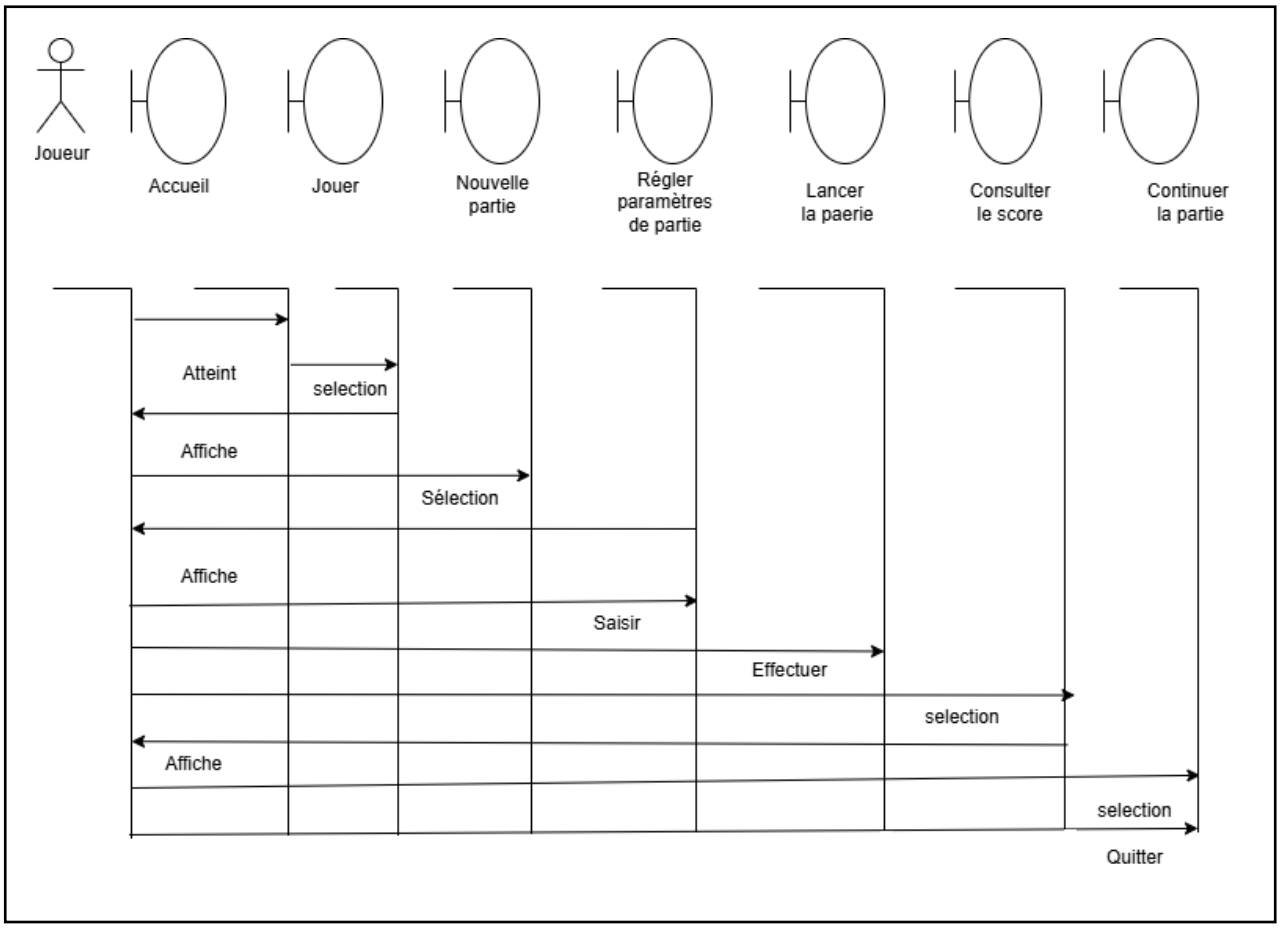


Figure 4.15 : Diagramme de séquence ‘ jouer’

8. Conclusion

L'association des diverses formes de diagrammes permet d'obtenir une vision globale des aspects statiques et dynamiques des systèmes. Comme on peut le voir, l'activité de conception a simplifié la compréhension de notre système, ce qui entraîne une transition vers l'activité de mise en œuvre.

Dans le prochain chapitre, nous allons exposer la mise en place et la mise en œuvre de notre application. Afin de démontrer l'efficacité de l'approche présentée dans les sections précédentes, il est nécessaire de concevoir un logiciel sous forme d'un prototype pédagogique qui permet d'améliorer un jeu combinatoire (jeu de dames) en fonction des éléments abordés dans ce chapitre

Chapitre 05 : Implémentation

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 du logiciel 'Game strategy programing'

Comme son nom l'indique, notre programme vise à évaluer des stratégies de jeu combinatoire (comme le jeu de dames) en utilisant l'idée d'hybridation des réseaux de neurones avec des algorithmes évolutionnaires. La programmation de ces stratégies est effectuée en suivant un processus évolutif qui comprend au départ un ensemble de stratégies élaborées de manière aléatoire. Ce programme facilite la création de nouvelles générations, leur importation et leur enregistrement, car le processus d'une seule génération devient très long, par exemple une configuration par défaut prend presque 5 heures.

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

Modèle : 81H7

Version : 10.0.19045 Build 19045

Processeur : Intel ® Core ™ i3-6006U CPU @ 2,00 GHz 2,00 GHz

Mémoire installée (RAM) : 12,00 Go

Type du système : Système d'exploitation 64 bits, Processeur x64, Microsoft Windows10 Professionnel N.

3.2. Environnement logiciel de développement 

Une fois l'analyse et la conception terminées, il nous reste à sélectionner une plateforme de développement adaptée afin de mettre en œuvre tous les efforts que nous avons déployés tout au long de ce sujet. Nous avons opté pour l'utilisation de l'environnement de développement Borland Delphi pour la réalisation de ce prototype.

Delphi est un logiciel de développement visuel rapide sous Windows (Rapid Application Developpement) qui utilise le langage Pascal Orienté Objet. Il permet de concevoir des applications fenêtrées qui peuvent être exécutées directement et redistribuées librement sous Windows ou DOS, avec un minimum de programmation.

L'apprentissage de ce langage est simple car les objets utilisés possèdent des caractéristiques et des méthodes. Les caractéristiques de l'objet (couleur, taille,...) sont les caractéristiques de l'objet, tandis que les méthodes sont les procédures (classiques ou événementielles) et les fonctions qui y sont associées.
La version 7 de Delphi a été sélectionnée car elle offre tous les outils indispensables pour concevoir, tester et déployer des applications, tels qu'une bibliothèque de composants réutilisables importante, une série d'outils de conception, de modèles d'applications, de fiches et d'experts de programmation que les versions précédentes du logiciel ne proposaient pas.

4. Aperçus sur les classes utilisées

Dans cette partie, nous allons exposer les principales méthodes de classe. La classe **TTournement** débute en gérant le tournoi. Elle propose des méthodes pour accueillir de nouveaux joueurs, sélectionner des adversaires, classer les joueurs en fonction de leurs résultats et choisir les meilleurs parmi eux.

En utilisant la classe **TPlayer**, on peut identifier les coups envisageables, effectuer un coup en modifiant la table du jeu et sélectionner le meilleur coup en utilisant l'algorithme de recherche Fail-soft Alpha-Beta.

La classe **TStrategy** met en place la configuration du réseau de neurones, elle autorise l'activation des stratégies, crée une stratégie fille en suivant les lois décrites dans le chapitre précédent, et réalise l'évaluation de la table du jeu.

Enfin, la classe ***TTable*** effectue la gestion de la table du jeu.

5. Présentation de quelques méthodes de classe

Nous avons utilisé différentes méthodes pour créer notre application, nous allons vous présenter les plus importantes:

Grâce à la méthode **PlayerClassification**

```
procedure TTournament.PlayersClassification;
var
 i, j, indmin: Byte; // Déclaration de variables pour les boucles et l'index minimum
 p, p1: TPlayer; // Déclaration des variables de joueur
begin
 // Tri des joueurs  par classification
 for i := 1 to members.Count - 1 do
 begin
```

```
p := members.items[i - 1]; // Sélection d'un joueur à comparer
indmin := i; // Initialisation de l'index minimum
// Recherche du joueur avec la plus petite classification
for j := i + 1 to members.Count do
begin
p1 := members.Items[j - 1]; // Sélection d'un autre joueur pour la comparaison
// Vérification si le joueur actuel a une classification plus basse que le joueur précédent
if p1.Result < p.Result then
begin
p := p1; // Mise à jour du joueur actuel avec le joueur ayant une meilleure classification
indmin := j; // Mise à jour de l'index minimum
end;
end;
// Échange des positions des joueurs dans la liste pour trier
p1 := members.items[i - 1]; // Sélection du joueur à sa position d'origine
members.items[i - 1] := p; // Déplacement du joueur avec la meilleure classification
members.items[indmin - 1] := p1; // Déplacement du joueur précédent à la position précédente du joueur actuel
end; // Attribution des numéros de classement aux joueurs
for i := 1 to members.Count do
begin
p := members.items[i - 1]; // Sélection d'un joueur
p.Number := i; // Attribution du numéro de classement
end;        end;
```

L'utilisation de la méthode **AlphaBetaFailSoft** de la classe **TPlayer** est réalisée pendant la compétition afin de trouver le meilleur coup. La description fictive de cette méthode est la suivante :

```
procedure TPlayer.AlphaBetaFailSoft(
 Table: TTable;
 Adversaire: TPlayer;
 BestMove: TMove;
 ply: Byte;
 Alpha: Extended;
 Beta: Extended;
 Var score1: Extended;
 Tour: Integer;
 Result: Integer);
```

```
var
    current, score: Extended; // Variables pour stocker les scores actuels
    pile: TPile; // Pile pour stocker les mouvements possibles
    table1: TTable; // Copie de la table actuelle
    move: TMove; // Mouvement en cours d'évaluation
    ad, pl: TPlayer; // Copie de l'adversaire et de soi-même
    k2, k: Integer; // Variables d'itération
    tab: array[1..14] of Byte; // Tableau pour stocker les mouvements
    tabs: TAbstractTable; // Tableau abstrait pour évaluation de la stratégie
    h: Integer; // Variable temporaire
   begin
```

```
k := 0;
pile := TPile.Create; // Initialisation de la pile de mouvements
DetectAllMoves(Table, pile); // Détection de tous les mouvements possibles
Inc(itera); // Incrément de l'itération
arrival := arrival + 1; // Incrément de l'arrivée
if pile.Count = 0 then
score1 := -1 // Si la pile est vide, attribuer un score négatif
else
begin
if ply <= 0 then
begin
tabs := GetAbstractTable(Table, Adversaire); // Obtention du tableau abstrait
strategy.SetVector(tabs); // Définition du vecteur stratégique
score1 := Strategy.EvaluateStrategy; // Évaluation de la stratégie
end
else
begin
current := -1; // Initialisation du score actuel
table1 := TTable.Create; // Création d'une copie de la table actuelle
pl := TPlayer.PseudoCreate; // Création d'une copie de soi-même
ad := TPlayer.PseudoCreate; // Création d'une copie de l'adversaire

while pile.Count > 0 do
begin
table.CopyTo(table1); // Copie de la table actuelle
```

```
Adversaire.CopyTo(ad); // Copie de l'adversaire

Self.CopyTo(pl); // Copie de soi-même

move := pile.Depiler; // Récupération d'un mouvement à évaluer

pl.ExecuteMove(table1, ad, move); // Exécution du mouvement sur une copie de la table

ad.AlphaBetaFailSoft(table1, pl, BestMove, ply - 1, -Beta, -Alpha, score1, Tour, Result);

score := -score1; // Inversion du score obtenu

if score >= current then

begin

current := score; // Mise à jour du score actuel

k := move.Count; // Mise à jour du nombre de mouvements

for k2 := 1 to move.Count do

begin

tab[k2] := move.First; // Stockage des mouvements dans un tableau

move.DeleteF; // Suppression du mouvement traité

end;

if score >= Alpha then

begin

Alpha := score; // Mise à jour d'Alpha

if score >= Beta then

break; // Arrêt de l'itération si Beta est dépassé

end;

end;

if Assigned(move) then

move.Free; // Libération de la mémoire du mouvement traité
```

```
end;
table1.Free; // Libération de la mémoire de la copie de la table
ad.PsaudoDestroy; // Destruction de la copie de l'adversaire
pl.PsaudoDestroy; // Destruction de la copie de soi-même
score1 := current; // Attribution du score actuel
end;
end;
pile.Free; // Libération de la mémoire de la pile de mouvements
Dec(itera); // Décrément de l'itération
BestMove.Clear; // Effacement du meilleur mouvement précédent
for k2 := 1 to k do
begin
BestMove.Add(tab[k2]); // Ajout des mouvements stockés dans le meilleur mouvement
end;
end;
```

À chaque nouvelle partie, la méthode **InitialiseToPlay** de la classe **TTable** est responsable de mettre en marche la table du jeu de dames. On peut décrire son pseudo code comme suit :

```
procedure TTable.InitialiseToPlay;
var
 i: Byte; // Déclaration de la variable d'itération
begin
 // Initialisation des valeurs dans la table pour commencer une partie
 for i := 1 to 32 do
```

```
begin
 if i <= 12 then
 table[i] := i // Attribution des valeurs positives aux 12 premières cases
 else
 begin
 if i >= 21 then
 table[i] := -(33 - i) // Attribution des valeurs négatives aux cases 21 à 32
 else
 table[i] := 0; // Attribution de zéro aux autres cases
 end;
 end; end;
```

La méthode ***EvaluateStrategy*** qui appartient à la classe ***TStrategy***, effectue une évaluation de la table du jeu

```
zfunction TStrategy.EvaluateStrategy: Extended;
 var
 Layer: TLayer; // Déclaration d'une variable de type TLayer pour stocker la dernière couche
 Neural: TNeural; // Déclaration d'une variable de type TNeural pour stocker le neurone évalué
 begin
 NeuralNetwork.EvaluateNeuralNetwork; // Évaluation du réseau neuronal
 Layer := NeuralNetwork.ListLayer.Items[NeuralNetwork.ListLayer.Count - 1]; // last c
 Neural := Layer.ListNeural.Items[0]; // select first couche
 EvaluateStrategy := Neural.Value; // Récupération de la valeur calculée par le neurone
 end;
```

6. Création et intégration des interfaces

Voici une série d'interfaces accompagnées de leurs scénarios descriptifs :

6.1. Interface d'accueil

Il s'agit de la première fenêtre qui apparaît aux utilisateurs. Afin de générer une nouvelle génération, il suffit de cliquer sur le bouton **Nouveau** dans le menu **Ficher**, ou vous pouvez également utiliser le bouton **Nouveau** du bar d'outils. Un volet apparaîtra, contenant une liste hiérarchique d'éléments représentant tous les paramètres du processus évolutionnaire, comme illustré dans la figure ci-dessous.

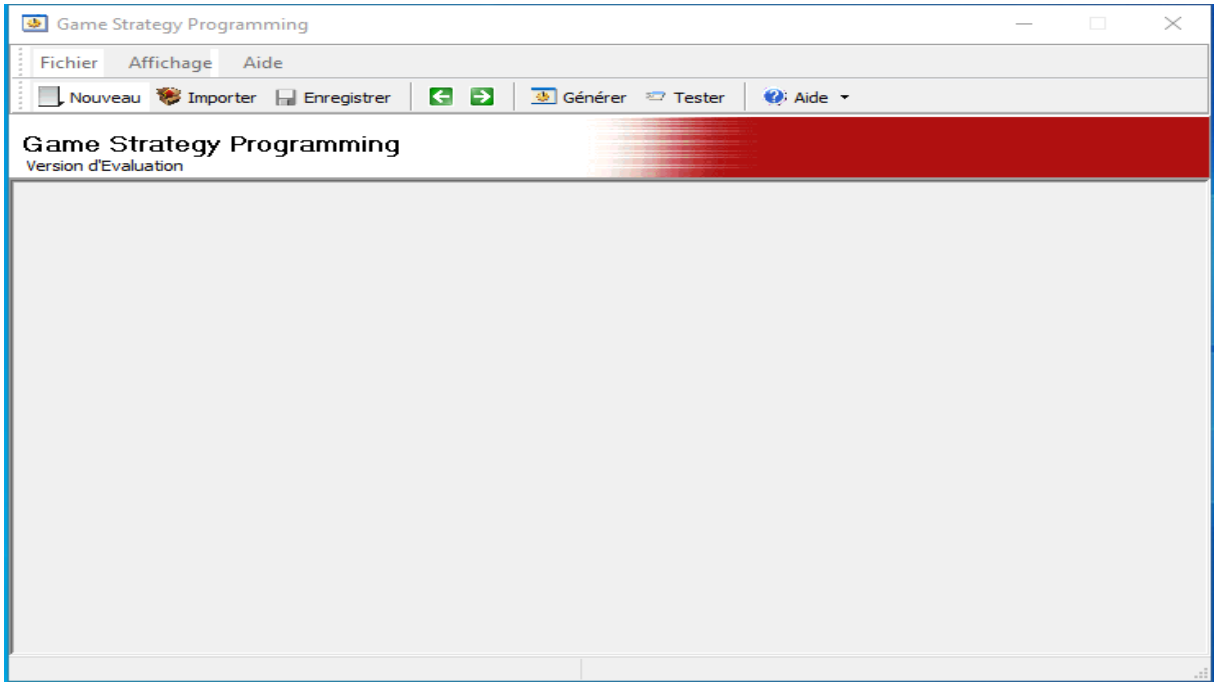


Figure 0.1 Interface d'accueil.

6.2. Interface de configuration des paramètres de la génération

Il est nécessaire de configurer les éléments qui représentent tous les paramètres d'évolution les uns après les autres, en respectant leur ordre de configuration, car certains paramètres devront être configurés avant d'autres. Ainsi, afin de configurer les paramètres de la génération,

nous suivons les étapes mentionnées ci-dessous.

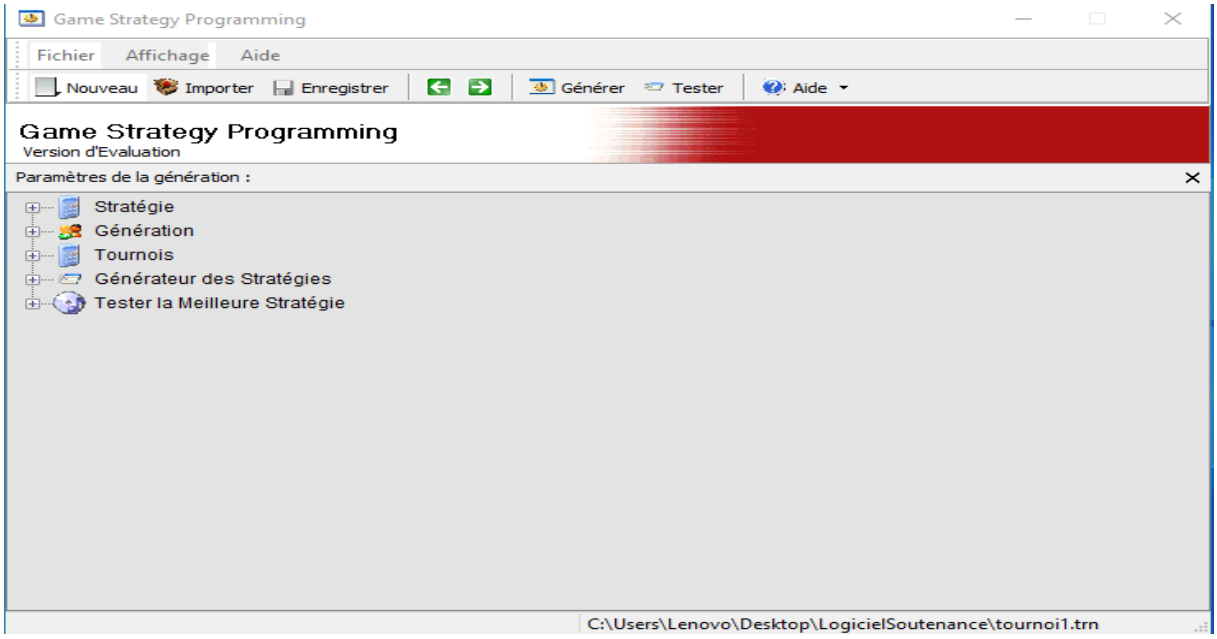


Figure 5.2 Interface de configuration des paramètres de la génération.

6.3. Interface nouvelle stratégie

La première étape consiste à configurer la stratégie, qui se compose de deux éléments distincts :

- Un réseau de neurones, dont la structure restera la plus cruciale, car elle joue un rôle essentiel dans le développement stratégique. Il dispose de poids et de polarisations (seuils) grâce auxquels les différentes évaluations sont maintenues. Nous avons donc opté pour une structure spatiale similaire à celle du réseau de neurones défini dans le chapitre précédent pour le jeu de dames.

Pour le jeu de dames, il y a deux types de pièces, la pièce normale et la pièce roi. La première ne change pas dans sa configuration initiale, tandis que la pièce d'un roi devrait être modifiée lors de la génération. D'ordinaire, la pièce normale est de 1 pendant que la pièce roi vaut 3.

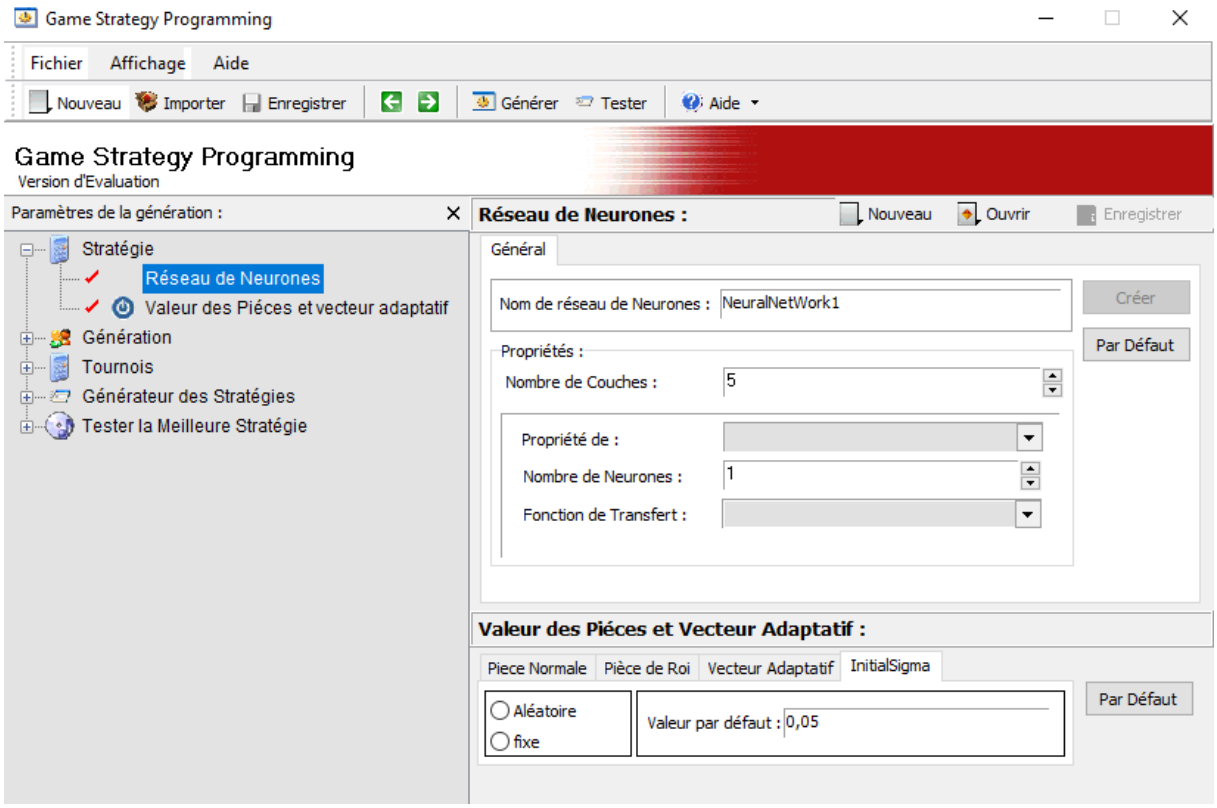


Figure 5.3 Interface de création de nouvelle stratégie.

6.4. Interface « Génération »

Cela correspond à la deuxième étape, qui se compose de : - Un espace de recherche qui regroupe toutes les stratégies, chacune ayant sa propre configuration (réseau de neurones et valeur des pièces).

- L'utilisation de paramètres de progéniture permet d'obtenir de nouvelles stratégies en appliquant l'opération de progéniture à chaque stratégie de l'espace de recherche.

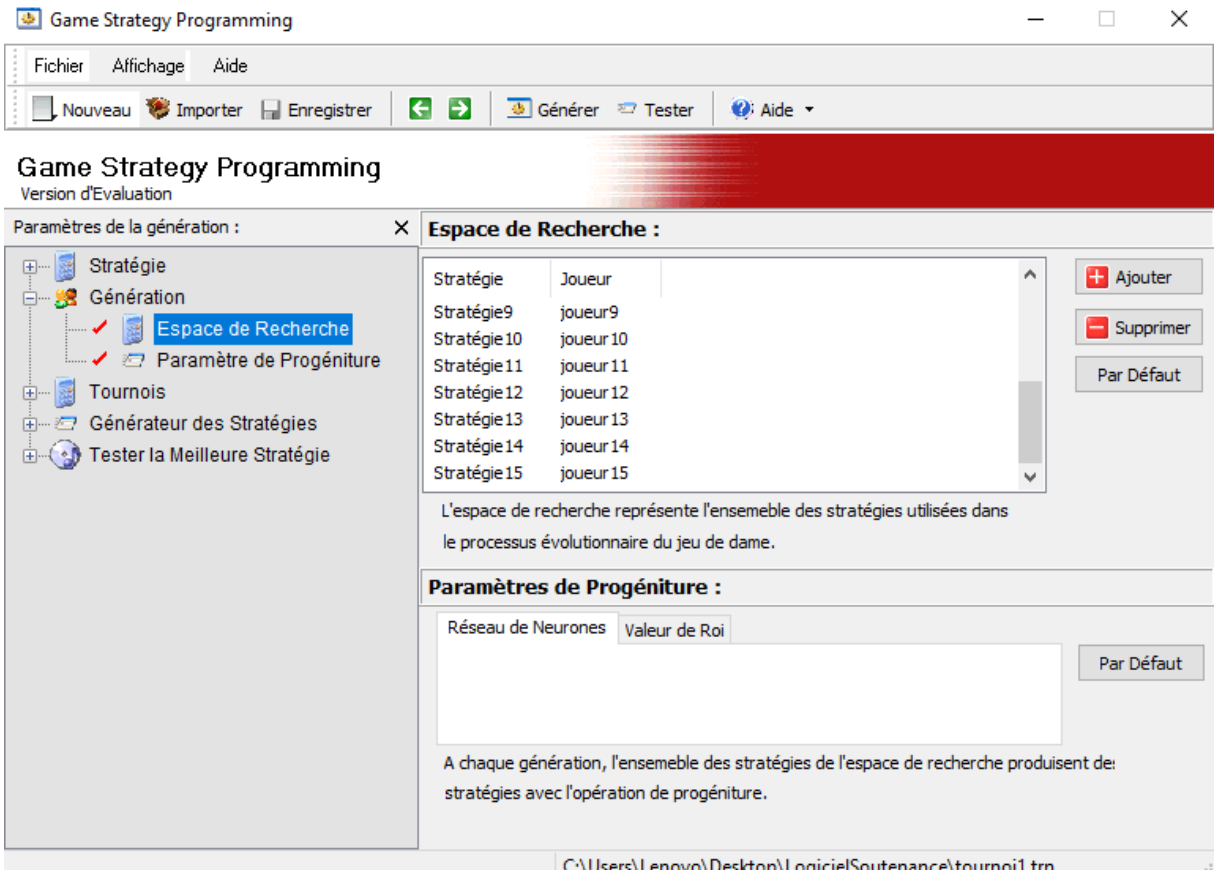


Figure 5.4 Interface de création de nouvelle génération

6.5. Interface « Tournois »

Dans la troisième étape, un tournoi est organisé avec un groupe de joueurs, chaque joueur appliquant sa stratégie correspondante dans l'espace de recherche. Nous sommes autorisés à calculer la fonction d'évaluation de chaque stratégie selon cette correspondance. La fonction d'évaluation d'une stratégie correspond à l'ensemble des résultats obtenus.

Il est évident que le nombre total de joueurs est égal au nombre de stratégies, ainsi qu'aux stratégies de filiation. La structure de cet élément est la suivante :

- Dans le jeu, on définit le nombre de mouvements et de coûts envisageables dans une partie, ainsi que le résultat d'une partie et la profondeur de recherche.

- Lors du tournoi, on calcule le nombre d'adversaires et le nombre de joueurs à choisir pour les générations futures.

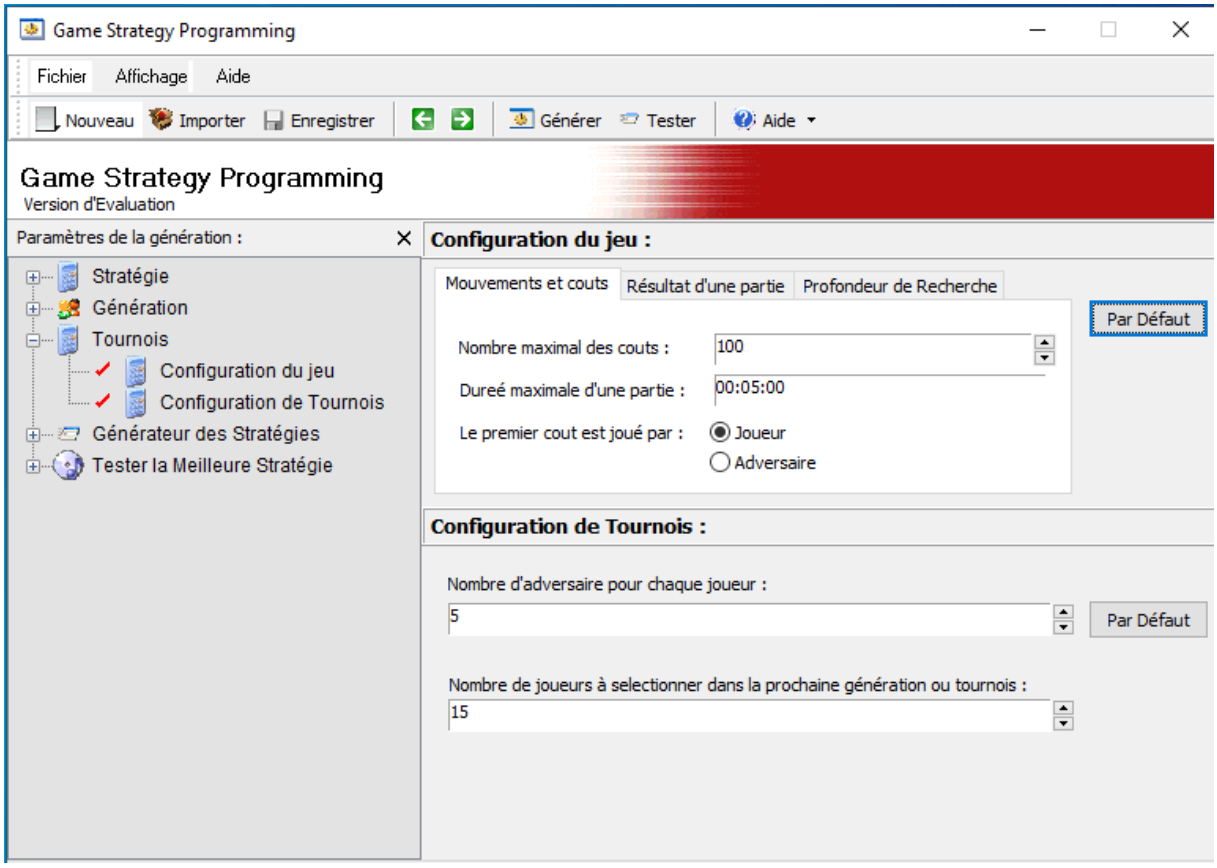


Figure 5.5 Interface de tournois.

6.6 Interface « Exécuter le générateur du processus évolutionnaire »

Après avoir configuré les paramètres précédents, la quatrième étape consiste à générer notre algorithme évolutionnaire. Pour ce faire, il suffit de cliquer sur le bouton "**Générer**" dans la barre d'outils. Cela ouvrira un volet contenant la plupart des informations nécessaires pour visualiser le processus de génération. Une fois prêt, vous pouvez démarrer le processus en cliquant sur le bouton "**Démarrer**".

Dans l'onglet "**Général**", vous avez le contrôle sur l'exécution des différentes étapes du processus évolutionnaire telles que l'Initialisation, la Progéniture, l'Évaluation, la Sélection et la Mise à jour. Notez que l'étape d'initialisation est uniquement réalisée lors de la première génération.

L'onglet "**Détail**" vous permet de suivre l'évaluation de chaque génération, où les joueurs compétitionnent entre eux. À tout moment, vous pouvez interrompre le processus en cliquant sur "**Stop**", ce qui vous permet de tester la meilleure stratégie obtenue.

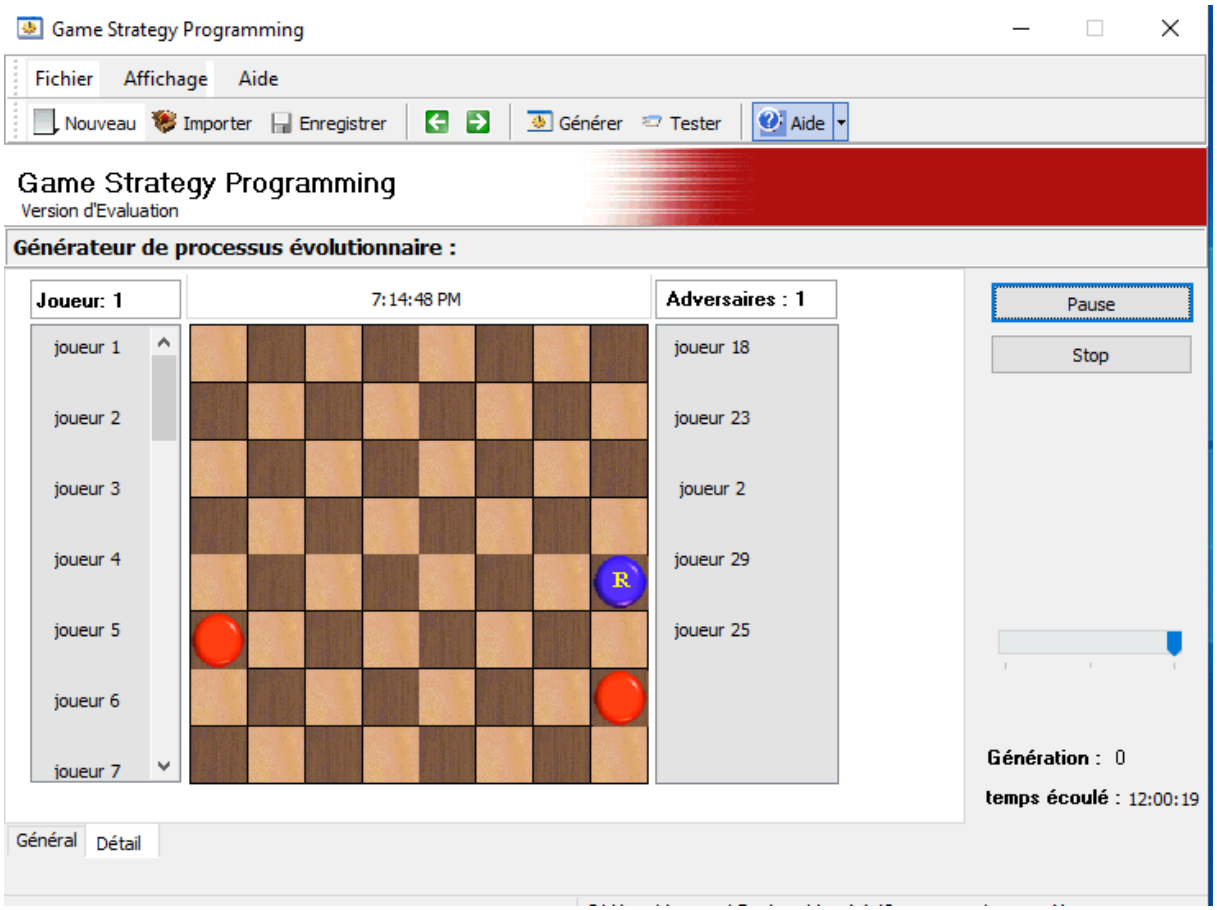


Figure 5.6 : Interface d'exécution du générateur du processus évolutionnaire.

6.7. Interface « Exécuter le test »

Enfin, la dernière tâche à accomplir consiste à tester la meilleure stratégie obtenue à partir de plusieurs générations successives, en jouant un jeu de dames contre l'ordinateur qui utilise cette stratégie. Afin d'y parvenir, sélectionnez le bouton « Tester » dans la barre des options. Une fenêtre apparaît, puis sélectionnez « Nouvelle partie » afin de débuter un nouveau jeu.

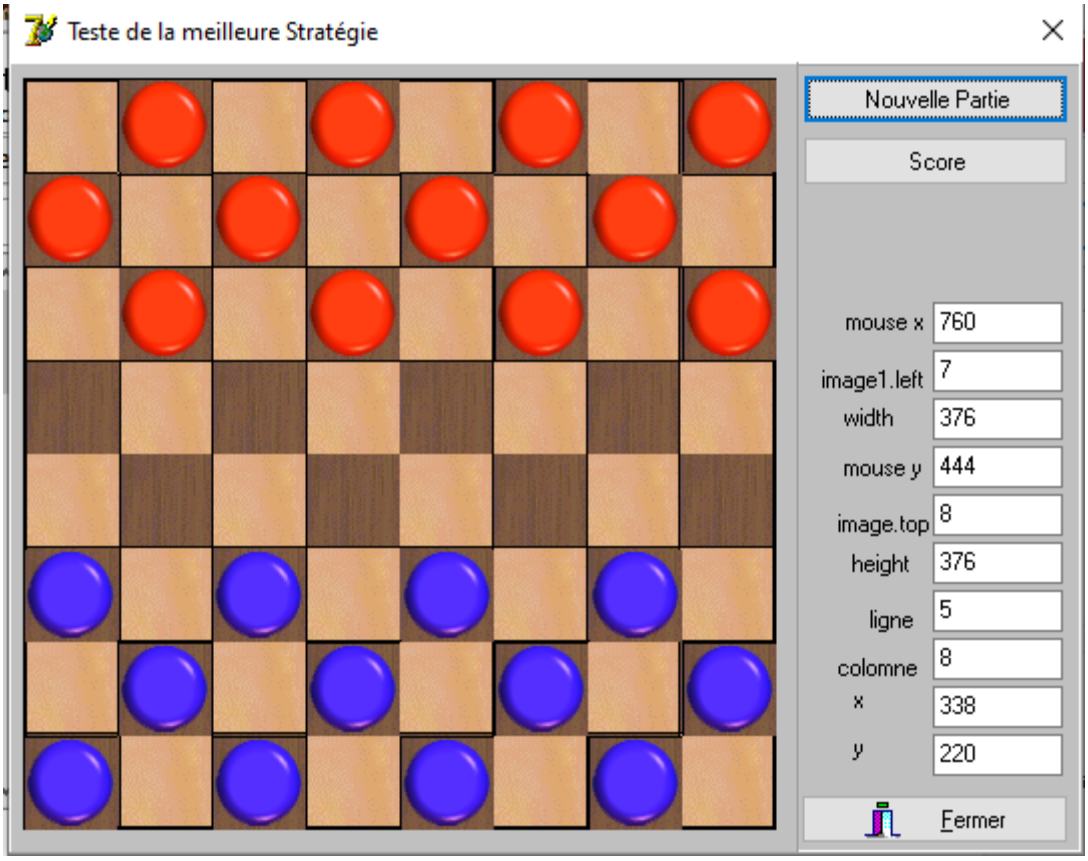


Figure 5.7 Interface d'exécution du test.

7. Conclusion

Nous avons vu dans le chapitre précédent que la création d'un logiciel qui répond aux besoins et aux objectifs énoncés dans cette étude nécessite tout d'abord une compréhension approfondie du problème à résoudre. Par la suite, il est primordial de mettre en place un plan de travail précis qui détermine les étapes essentielles pour atteindre une solution améliorée. Finalement, une représentation précise du problème simplifie la mise en place, ce qui conduit à un logiciel fiable.

Conclusion générale

Dans cette étude, nous avons exploré les intersections entre trois domaines distincts : la théorie des jeux, les algorithmes évolutionnaires et les réseaux de neurones. Notre approche s'inspire des capacités d'adaptation des organismes vivants, en suivant les recherches de D.B. Fogel et Chelapilla, qui visent à optimiser les performances individuelles pour la survie.

Dans le domaine de la théorie des jeux, les stratégies de jeu ont été définies comme des entités qui peuvent s'adapter à des environnements en constante évolution. En utilisant des méthodes évolutives pour analyser les réseaux de neurones, notre objectif était d'améliorer les poids initiaux afin de favoriser la convergence vers des solutions optimales plutôt que des minimes locaux.

Les résultats de nos expériences ont conduit à la découverte de techniques évolutionnaires performantes pour ajuster les stratégies de jeu basées sur les réseaux de neurones, notamment les stratégies dérivées. Même si la complexité du problème et les capacités informatiques actuelles peuvent restreindre les résultats obtenus, nous avons observé des performances stables et répétables dans des situations similaires.

L'hybridation des approches évolutionnaires et neuronales présente de nombreuses opportunités dans divers domaines tels que la robotique, la vision par ordinateur, la reconnaissance de formes et la mécanique. L'objectif de notre contribution est d'ouvrir de nouvelles perspectives de recherche dans le domaine de la théorie des jeux, notamment en examinant les interactions entre les algorithmes évolutionnaires et les réseaux de neurones afin de résoudre des problèmes de prise de décision essentiels, tout en améliorant l'efficacité des algorithmes.

Références

[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.

[4] : Anne Spallanzani, " Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole".

[5]: Kumar Chellapilla, David B. Fogel "Evolving an Expert Checkers Playing Program without Using Human Expertise"

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

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

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

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

[16]: Hichem talbi, " Algorithmes évolutionnaires quantiques pour le recalage et la segmentation multi-objectif d'images", 2009.

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

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

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

[23]: ED Hermes *" algorithmes génétiques et réseaux de neurones application des commandes de processus* ".Bruxelles 1995.

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

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

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

[35]: Khadir Mohamed Tarek, "Principes de base des réseaux de neurones artificiels et apprentissage". Support de cours. Université Badji Mokhtar, Annaba.

[38]: Kolen J.F. et Pollack J.B., "*Back-propagation is sensitive to initial conditions*". Technical Report 90-JK-BPSIC, CIS Dept., Ohio St Univ., Columbus, Ohio, 1990.

[39]: Thimm G. et Fiesler E., "*High Order and Multilayer Perceptron Initialization*". IDIAP technical report 94-07, 1994.

[40]: Falhman S.E., "*An Empirical Study of Learning Speed in Backpropagation Network*s". Technical report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1988.

[41]: Bottou L-Y., "*Reconnaissance de la parole par réseaux multi-couches*". Proceedings of the International Workshop on Neural Networks and Their Applications, pp 197-217, 1988.

[47]: Olivier Guibert, " *Analyse et Conception des Systèmes d'Information – Méthodes Objet Le langage de modélisation objet UML*"

[49]: Olivier Sigaud, " *Introduction à la modélisation orientée objets avec UML*.

[50]: 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.

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

[8]: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

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

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

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

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

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

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

[29]:http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm#.UXQklaJA17E

[30]:http://www.igm.univmlv.fr/~dr/XPOSE2002/Neurones/index.php?rubrique=Architecturedesreseauxdeneurones

[31]:https://www.google.dz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CCwQFjAA&url=http%3A%2F%2Fwww.info.fundp.ac.be%2F~jpl%2FCours%2FOptim%2520Combi%2FPartie%25205%2520neurones%2520.doc&ei=NiZ0UbO8LoHw4QSftYDYBQ&usg=AFQjCNEukETP_C7l106DJW4mswO6WsMIZA&bvm=bv.45512109,d.ZWU

[34]:http://www.sylbarth.com/nn.php

[36]:http://www.memoireonline.com/04/12/5750/m_Identification-et-commande-des-systemes-non-lineaires15.html

[48]: http://creativecommons.org/licences/by-nc-sa/2.0/fr

[52] :http://souqueta.free.fr/Project/files/TE_AG.pdf

[53] :https://fr.scribd.com/document/546797039/Chapitre2