CAHIER DES CHARGES

Morpion Solitaire

Groupe L2K 2

MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

Cahier des charges

Morpion solitaire

Les informations d'identification du document

Référence du document :	
Version du document :	1.03
Date du document :	12/02/23
Auteur(s) :	OULD MOHAND Salah RATSIMIHAH Pierre, SY Thillo Aissata, ZAHEM Yasmine

Les éléments de vérification du document

Validé par :	Bruno Bouzy
Validé le :	13/02/2023
Soumis le :	13/02/2023
Type de diffusion :	Document électronique (.DOCX)
Confidentialité :	Réservé aux étudiants UFR Maths-Info de l'université Paris Cité

MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

Sommaire

Glossaire

Heuristique : c’est « l'art d'inventer, de faire des découvertes » en résolvant des problèmes à partir de connaissances incomplètes.
En d’autres mots c’est une méthode ou stratégie simplifiée utilisée pour trouver une solution rapide à un problème ou une réponse à une question. Cela peut être considéré comme un raccourci mental qui aide à prendre une décision ou à trouver une solution de manière plus efficace en utilisant des connaissances et des expériences précédentes.
Les heuristiques sont souvent utilisées lorsque la solution complète à un problème est trop complexe ou prend trop de temps à trouver.

S’il y’a donc une chose à en retenir cela serait qu’en algorithmique, une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.

UCB-T (Upper Confidence Bound applied to Trees) : L'UCB (Upper Confidence Bound) est une technique d'exploration-exploitation qui permet de choisir le meilleur parmi plusieurs options incertaines en prenant en compte la quantité d'informations disponible sur chacune d'entre La UCB-T (Upper est appliquée au MCTS pour sélectionner le meilleur nœud enfant à explorer.

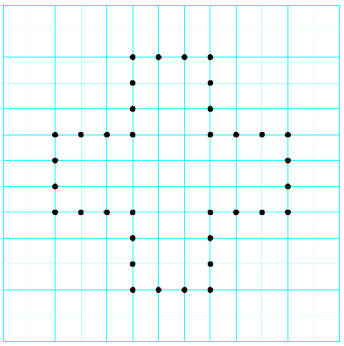
L'UCB-T combine l'estimation de la valeur d'un nœud enfant avec une incertitude mesurée par le nombre d'itérations de simulation effectuées pour ce nœud. Plus le nombre de simulations est élevé, plus l'incertitude sur la valeur est faible, et plus le nœud enfant est considéré comme fiable. En conséquence, le score UCT d'un nœud enfant est défini comme étant la somme de sa valeur estimée et d'un terme de pénalité qui dépend de l'incertitude sur la valeur. Le nœud enfant avec le plus haut score UCT est sélectionné pour être exploré en priorité.

NB : Les expressions définies ici sont repérables grâce au (⌘) qui les suit.

1. Introduction

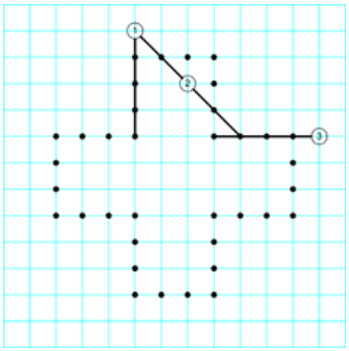
1.1. Contexte : l'environnement dans lequel s'inscrit le projet (stratégie, enjeux, domaine, etc.)

Le Morpion Solitaire est un jeu de réflexion de type casse-tête datant d'au moins 50 ans. En effet, comme son nom l'indique on y retrouve l'esprit du Morpion mais aussi le Solitaire étant donné qu'il se joue seule. Son origine exacte est inconnue, le plus ancien joueur est Daniel Goffinet un professeur de Maths Sup qui y jouait déjà en 1962 avec ses camarades. Cependant on peut retenir que le Morpion Solitaire a été vulgarisé en 1974 par Pierre Berloquin. Ce dernier était effectivement l'auteur d'une rubrique régulière intitulée "Jeux et Paradoxes" dans le magazine *Science & Vie,* ou il publiait chaque semaine des grilles avec de nouveaux records.



Il existe bien évidemment plusieurs variantes de ce casse-tête mais nous nous concentrerons sur la version dite 5D c'est-à-dire « 5 disjoints ». Les règles du jeu sont assez simples. Vous n'aurez besoin que d'un crayon et un papier ! Au départ, le joueur trace une croix grecque à l'aide de 36 symboles, chaque côté étant formé de quatre symboles tel qu'illustré.

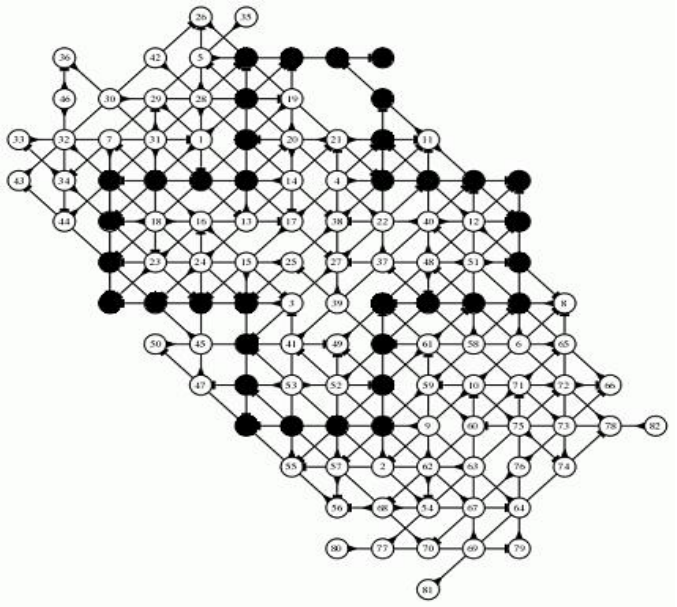
Par la suite, le joueur trace un point sur une intersection libre horizontalement, verticalement ou en diagonale, afin de faire des alignements de cinq points, sans la possibilité de placer un alignement dans le prolongement d'un autre d'où l'appellation « 5 disjoints ». Le but est de réussir à faire le plus d'alignements possible. Voici un exemple d'illustration de ce à quoi pourrait ressembler la grille après 3 coups.



1.2. Historique du contexte dans lequel s'inscrit le projet.

Dans le cadre de nos apprentissages nous sommes chargés de mener à bien un projet en groupe tout le long du second semestre en vue de la validation de notre deuxième année de licence d'informatique. En effet, quoi de mieux que l'élaboration d'un projet, de sa conception à sa réalisation, pour restituer les acquis et connaissances transmises depuis la première année de licence mais aussi développer de nouvelles connaissances ?!

Pour en revenir a notre thème, le Morpion Solitaire peut être intéressant à étudier du point de vue de son implémentation. En effet, nous avons pu voir que le record moyen pour les humains est d'environ 60. Mais en faisant appel à des ordinateurs on peut repousser cette limite. En février 2008, le français Tristan Cazenave (professeur à l'université Paris Dauphine) a battu son propre record (précédemment de 78) en proposant une grille de 80 obtenu à l'aide d'un ordinateur. C'est seulement plus de deux ans après, en aout 2010, que l'américain Christopher D. Rosin a établi le record et reste jusque-là indéfait. Il a utilisé un algorithme et logiciel très puissant afin de nous produire une grille optimisée de 82 coups. Enfin, il a été démontré qu'on ne peut dépasser les 84 coups pour cette version du Morpion

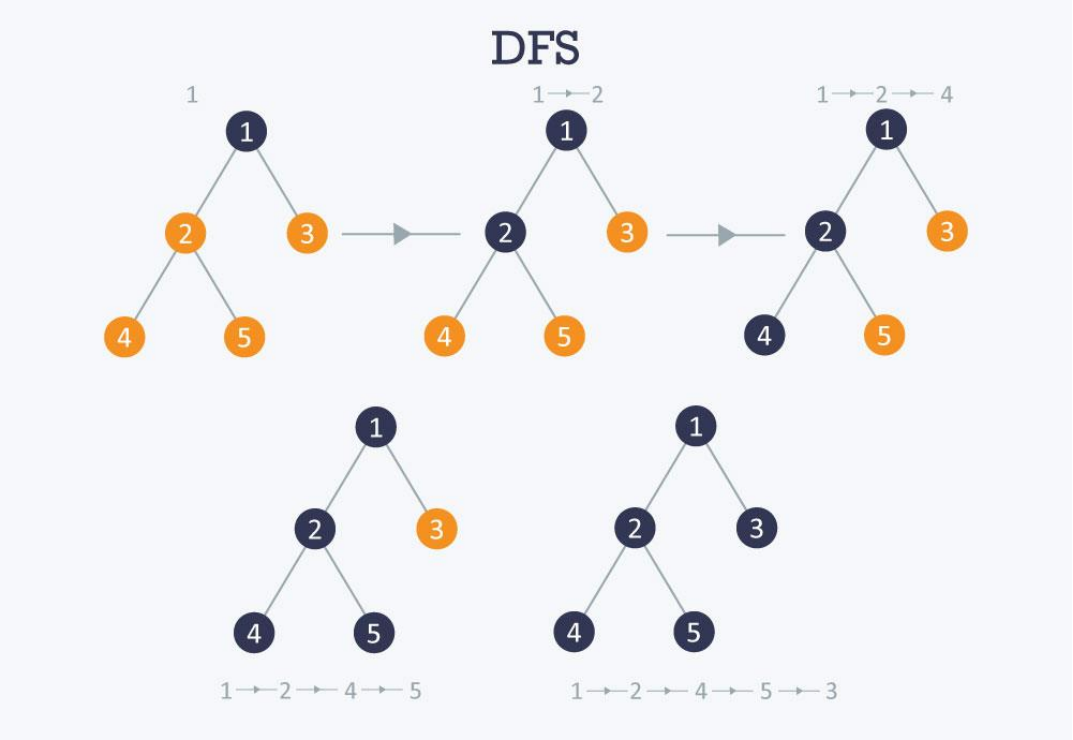


Grille de 82 coups proposé par C. Rosin

Solitaire.
En bref, ce projet nous permettra d'avoir une approche beaucoup plus concrète et pratique, qui se rapprocherait au plus de ce que l'on pourrait rencontrer dans le milieu professionnel.

1.3. Outils à exploiter

Nous allons donc utiliser notre cours d'algorithmique et structures de données afin de se pencher sur l'implémentation d'algorithme de recherche arborescente afin de parcourir la grille. Nous avons d'abord le *Depth First Search,* un algorithme de parcours en profondeur qui se définit de manière récursive. Son application la plus simple consiste à déterminer s'il existe un chemin d'un sommet à un autre. L'exploration d'un parcours en profondeur depuis un sommet S fonctionne comme suit. Il poursuit alors un chemin dans le graphe jusqu'à un cul-de-sac ou alors jusqu'à atteindre un sommet déjà visité. Il revient alors sur le dernier sommet où on pouvait suivre un autre chemin puis explore un autre chemin. L'exploration s'arrête quand tous les sommets depuis S ont été visités. Bref, l'exploration progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S. Il est long mais très efficace ; se présente comme suit.



Et enfin la recherche arborescente *Monte Carlo Tree Search* qui est un algorithme de recherche *heuristique* (⌘) utilisé dans le cadre de la prise de décision, mais aussi beaucoup utilisée en Intelligence Artificielle tel que dans les jeux par exemple. Il s'agit d'une approche probabiliste qui consiste à effectuer de nombreuses itérations, simulations en utilisant des données aléatoires, des échantillons pour estimer la probabilité d'obtenir un résultat particulier à travers différents scenarios possibles.

L'algorithme MCTS est un algorithme qui explore l'arbre des possibles. La racine est la configuration initiale du jeu. Chaque nœud est une configuration et ses enfants sont les configurations suivantes. MCTS conserve en mémoire un arbre qui correspond aux nœuds déjà explorés de l'arbre des possibles. Une feuille de cet arbre (un nœud n'ayant pas d'enfants) est soit une configuration finale (soit un des joueurs a gagné, ou s'il y a match nul), soit un nœud dont aucun enfant n'a encore été exploré. Dans chaque nœud, on stocke deux nombres : le nombre de simulations gagnantes, et le nombre total de simulations. Chaque étape est composée de quatre phases :

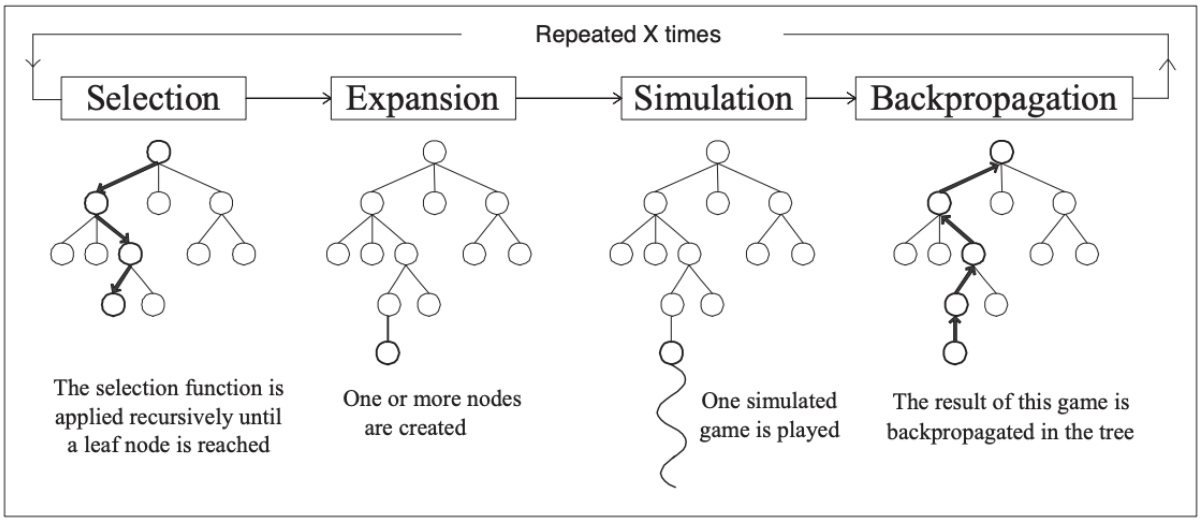
- **Sélection** : dans cette phase, on parcourt l'arbre des coups possibles à partir de la racine actuelle jusqu'à une feuille non explorée. Cela se fait en sélectionnant le nœud enfant avec le plus haut score d'évaluation UCB-T (Upper Confidence Bound applied to Trees) (⌘).

- **Expansion** : une fois qu'on a atteint une feuille non explorée, on l'expand en ajoutant de nouveaux enfants correspondant à des coups possibles non évalués.

- **Simulation** : à partir de la feuille récemment expandue, on simule aléatoirement une suite de coups jusqu'à la fin de la partie pour obtenir une estimation de la valeur de chaque coup.
- **Propagation** : les résultats de la simulation sont alors remontés à la racine en ajoutant à chaque nœud les informations nécessaires à la mise à jour de ses scores d'évaluation.

Le MCTS itère ces étapes plusieurs fois pour sélectionner finalement le meilleur coup à jouer en fonction des informations accumulées dans l'arbre.

Donc le MCTS est beaucoup plus rapide que le DPS mais reste néanmoins assez peu efficace. Il se présente comme suit :



2. Description de la demande

2.1 Les objectifs : Résultats que le projet doit atteindre

Ainsi, comme tout groupe, nous ne visons que la réussite de notre projet. Pour se faire nous devons matérialiser une plateforme de jeu permettant à un utilisateur de jouer au Morpion Solitaire.

En effet, à l'aide d'une interface textuelle on va permettre au joueur d'avoir une grille du jeu. De plus il va pouvoir positionner ses points comme il le souhaite et obtenir son score à la fin. La partie s'arrête lorsqu'il n'est plus possible de poser le moindre point sur la grille de jeu.

Par la suite, nous serons en mesure de programmer une intelligence artificielle capable de résoudre le jeu de la croix de la Malte de façon exacte. En s'aidant des outils à disposition tel que l'algorithme de recherche arborescente *Monte Carlo*, il aura pour but d'optimiser l'alignement des points afin d'obtenir le meilleur score.

2.2 Fonctionnalités du produit

Comme mentionnée plus haut, nous allons expliciter les fonctions de notre projet.

1) **Fonctionnalité 1** : L'interface textuelle

En effet, il nous faudra une interface par laquelle on peut dialoguer avec l'ordinateur par l'intermédiaire de commandes qu'il faut taper au clavier. Nous allons donc la coder avec la syntaxe du langage Python, d'abord nous aurons la grille du casse-tête sous forme d'une croix grecque de 36 points. Puis l'utilisateur sera en mesure de placer ses coups de façon aléatoire, sur l'ensemble des points disponibles sur la croix afin d'obtenir des alignements de 5 soit horizontalement, verticalement ou encore en diagonale.
L'utilisateur pourra par exemple voir le nombre de coups réalisés à la partie en cours et cela coup après coup, le meilleur score enregistrés mais il aura aussi la possibilité de revenir sur une ancienne partie non terminé. Le programme est aussi censé pouvoir dire au joueur s'il est possible ou non de jouer un autre coup, dans le cas contraire, le jeu s'arrête et le score s'affiche à l'écran. On parle ici d'une interface utilisateur, c'est-à-dire des caractères semi-graphiques pour afficher les menus, les boîtes de dialogue, les messages ou tout autre élément à destination de l'utilisateur.

2) **Fonctionnalité 2** : Simulation aléatoire

Etant donné que le Morpion Solitaire est un jeu de type casse-tête, il est très rare d'obtenir la même grille de coups, en réalité chaque partie peut être différente de la précédente, ce qui ajoute une incertitude et une variété au jeu. Les jeux de simulation aléatoire incluent souvent (tel que dans notre cas) des éléments de stratégie, de chance et de prise de décision. Dans notre cas, chaque partie

constituerait donc une simulation aléatoire, dans laquelle on prévoit que l'utilisateur seul puisse réaliser en moyenne autour de 25 coups. En effet l'utilisateur va d'abord positionner les coups les plus évidents, ceux aux extrémités en l'occurrence. Il va juste cherche à aligner le maximum de 5 points possible coup après coup sans chercher à les optimiser, il va donc très vite se retrouver bloqué, sans plus aucun coup à poser. En débutant, si le joueur n'a pas une vue d'ensemble pour poser des coups stratégiques, il lui sera difficile de dépasser la barre des 25 coups.

Nous allons donc tenter de pousser au maximum ces simulations et de remplir le plus de grilles différentes possibles afin de s'assurer que la limite ne soit point la grille proposée par le programme mais en réalité une imposé par les règles du jeu.

3) **Fonctionnalité 3** : Monte Carlo Search

L'algorithme de recherche arborescente Monte Carlo Search est très utilisée dans l'intelligence artificielle notamment dans le développement de jeux. Nous sommes chargés de réaliser une IA qui sera à son tour capable de résoudre le casse-tête, pour se faire nous allons faire appel au MCST dont le principe est expliqué dans la partie 1.3. Nous allons donc procéder à une évaluation des coups par simulations de Monte-Carlo, une simulation consistant en une partie aléatoire jouée jusqu'à la fin. L'évaluation sera obtenue en calculant la moyenne des longueurs des parties obtenues. C'est ainsi que l'on parviendra à obtenir un résultat très rapide même s'il reste peu efficace. Le MCST a lui seul, pourrait établir en moyenne 60 coups. Un facteur important de la recherche Monte-Carlo est le choix de la stratégie de simulation, pour influer sur les choix des coups aléatoires pendant la simulation jusqu'il n'y en ait plus.

4) **Fonctionnalité 4** : Recherche Arborescente et DFS

En plus de faire appel a l'algorithme MCTS, nous pourra aussi expérimenter l'usage de recherche arborescente avec le Depth First Search. Comme nous l'avons vu encore une fois dans la partie 1.3, c'est un superbe algorithme en termes d'efficacité mais contrairement au MCST il présente une faille car il manque de rapidité. Comme c'est un algorithme de recherche arborescente en profondeur, il va parcourir l'arbre en essayant d'atteindre la dernière feuille avant de remonter pour explorer les autres nœuds de l'arbre. En d'autres termes si on devait le ramener au Morpion Solitaire, il va s'appliquer sur la grille en traversant chaque chemin qu'il emprunte aussi loin que possible avant de passer à un autre. On peut le coder de manière récursive, ou itérative à l'aide d'une pile qui stocke à chaque étape les prochains nœuds à visiter. Il permettrait donc de réaliser des choix de coups optimales qui influerait donc grandement sur le score final.

5) **Fonctionnalité 5** : Interface Graphique (optionnel)

Finalement, si le temps nous le permet, nous nous attèlerons à la mise en place d'une interface graphique pour avoir un meilleur rendu visuel et esthétique. En effet il s'agira ici de rendre l'interface ergonomique et intuitive afin que l'utilisateur la comprenne tout de suite. Cela permettra à l'utilisateur d'être plus à l'aise avec l'environnement de jeu afin d'avoir la meilleure expérience interactive possible.

Cette dernière répondra aux mêmes critères que l'interface textuelle dans le cadre de notre implémentation du Morpion Solitaire. Donc son objectif reste de rendre optimale l'interface utilisateur, donc il nous faudra concevoir une interface accessible et facile à prendre en main pour tout type de support.

2.3 Critères d'acceptabilité et de réception

On souhaite pouvoir jouer au morpion solitaire. Le jeu doit permettre de :

- ✓ Proposer de démarrer avec une grille pré-enregistrée dans un fichier ;
- ✓ Jouer un coup (poser un point et définir l'alignement associé) ;
- ✓ Indiquer le score à chaque étape et le score final ;
- ✓ De plus l'IA doit être en mesure de résoudre le casse-tête ;

Notre programme ne pourra donc être validé en l'absence des critères énoncés plus haut. Mais si le temps nous le permet nous aimerions que le jeu puisse permettre de :

- ✓ Annuler le coup précédemment joué
- ✓ S'approcher au plus possible du record mondial
- ✓ D'avoir une interface graphique attrayante pour l'utilisateur

3. Déroulement

3.1 Planification : Grandes phases du projet et étapes principales

Afin de mener à bien notre projet, nous allons le sectionner en différentes parties afin qu'il puisse le plus abouti possible. Nous aurons donc :

- *La phase de spécification* : Nous allons chercher le plus de ressources possibles concernant notre thème. Les recherches et études ayant été déjà faites sur le sujet vont nous permettre d'élargir nos horizons.

- *La phase de conception* : Nous allons commencer à réfléchir aux différentes étapes intermédiaires de programmation dont on aura besoin.

- *La phase de programmation* : Nous allons ainsi mettre en place l'ensemble de nos recherches effectués lors des 2 phases précédentes. En bref, nous allons programmer le code de notre projet en Python.

- *La phase de test et d'intégration* : Nous allons donc vérifier l'efficacité et l'efficience de nos programmes à l'aide de test unitaires avant de les assembler (enchainer) pour l'exécuter en une seule fois.

- *La phase de recettes* : Il s'agit ici de s'assurer que notre programme fonctionne et se déroule tel que planifié dans le cahier des recettes. Et nous allons donc finir par la rédaction du rapport final.

3.2 Ressources : humaines et matérielles

Concernant les ressources humaines, pour ce projet, nous sommes une équipe de quatre étudiants en deuxième année de licence d'informatique avec des connaissances équivalentes, de plus encadré par Mr Bouzy Bruno.

Pour le matériel, nous nous servirons de nos ordinateurs personnels ou encore ceux mis à disposition par notre UFR. Nous coderons avec les logiciels et IDE de notre choix sous Python 3 puis nous les mettrons régulièrement en commun via le logiciel client *Tortoise SVN*.

Nous disposons aussi des ressources bibliographiques à notre portée tel que le compte-rendu de Tristan Cazenave [1] ou encore le livre du détenteur du record C. Rosin [2] et enfin le troisième volet de thèse pour la soutenance de doctorat de M. Bernard Helmstetter [3] mais aussi plein d'autres annexes citées en références.

4.Contraintes

Comme dans tous domaines, nous pouvons rencontrer quelques obstacles concernant la mise en place de ce projet. En effet, la principale demeure celle du temps. Le délai accordé pour la phase de programmation peut s'avérer être un peu juste pour s'assurer de la qualité du programme (cohérence, optimisation de syntaxe, ajout de commentaires etc.). Notamment pour la prise en charge du suivi des améliorations à apporter lors des bugs ou erreurs indexés par les tests unitaires.

Ensuite, nous avons celle qui en découle directement. En effet, le temps disponible pourrait nettement influencer la qualité de l'interface graphique que l'on pourrait obtenir en dernière étape de conception.

5.Reférences

Voici la bibliographie, les pages et articles en bref les références qui vont nous accompagnés tout au long de notre projet.

[1] Tristan Cazenave, «Reflexive Monte Carlo Search, GPW, (2007).

[2] Christopher D. Rosin, « Nested Rollout Policy Adaptation for Monte Carlo Tree Search », IJCAI-11 : AAAI Press, pages 649–654, (2011).

[3] Thèse du Dr.Helmstetter, Analyses de dépendances et méthodes Monte-Carlo dans les jeux de réflexion : https://octaviana.fr/document/126279233#?c=&m=&s=&cv=

Powerpoint Recherche IA : https://helios2.mi.parisdescartes.fr/~bouzy/Doc/IAL3/02_IA_recherche_BB.pdf

DFS : https://www.researchgate.net/publication/2840090_Depthfirst_search_solves_Peg_Solitaire

Reflexive MCS : https://www.lamsade.dauphine.fr/~cazenave/papers/reflexmc.pdf

Morpion Solitaire : https://fr.wikipedia.org/wiki/Morpion_solitaire

Site du jeu : http://www.morpionsolitaire.com

Dictionnaire de maths récréatives : http://www.recreomath.qc.ca/dict_solitaire_morpion.htm

Maxi Sciences : https://www.maxisciences.com/jeu/voila-comment-gagner-au-jeu-du-morpion-a-coup sur_art36584.html#:~:text=Pour%20vous%20ouvrir%20le%20plus,adversaire%20%C3%A0%20bloquer%20votre%20alignement.