Construction d'un joueur au jeu de Hex

Paul VILARS - 35719

1 Le jeu de Hex

Le jeu de Hex a une paternité controversée. Il est attribué d'abord à Piet Hein, mathématicien danois qui l'aurait découvert en 1942, alors qu'il cherchait à trouver un contre-exemple au théorème encore irrésolu des quatres couleurs. John Nash l'a ensuite popularisé (et redécouvert indépendamment en 1948), alors qu'il préparait sa thèse à l'université de Princeton, et cherchait un jeu dans lequel on pourrait savoir avec certitude quel joueur aurait une stratégie gagnante.

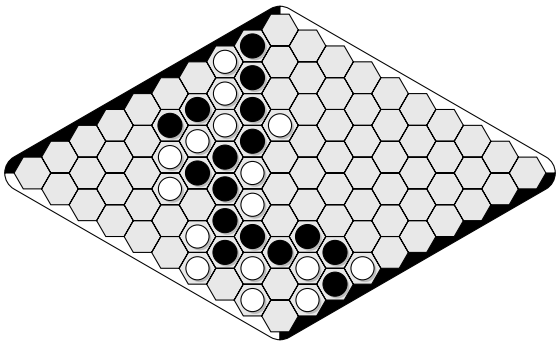

Figure 1: Un plateau de Hex de taille 10×10, dans lequel noir est vainqueur.

Les règles du jeu sont élémentaires : sur un plateau en forme de losange constitué d'hexagones, deux joueurs s'affrontent pour être le premier à relier les deux côtés opposés du plateau qui lui appartiennent. Chacun joue à tour de rôle en plaçant une pierre sur le plateau. Il est impossible de passer son tour ou de retirer une pierre. Tout placement est définitif.

Malgré cette apparente simplicité, le jeu de Hex dispose de propriétés qui le distinguent de la plupart des autres jeux à deux joueurs.

Propriété 1.1: Unicité du gagnant

Au jeu de Hex, au plus un joueur peut être gagnant.

En effet, si un joueur a relié ses deux bords, ce chemin constitue de fait une barrière infranchissable entre les bords du joueur adverse.

Propriété 1.2: Existence du gagnant

Au jeu de Hex, au moins un joueur est gagnant.

Celle-ci est nettement plus intéressante, car autrement dit, un match nul est impossible. La preuve de cette propriété, il semblerait connue de Nash, n'apparaît la première fois qu'en 1986 par Gale [3].

Une considération importante se détache des deux propriétés précédentes : *au jeu de Hex, jouer défensivement ou offensivement permet d'atteindre le même objectif.* Empêcher son adversaire de relier ses bords, c'est construire un chemin pour relier les siens, même si ce processus peut s'effectuer à l'insu du joueur non averti. Cette considération essentielle se prouvera utile pour construire notre adversaire.

Propriété 1.3: Existence d'une stratégie gagnante

Au jeu de Hex, il existe une stratégie gagnante pour le premier joueur.

Ce fut montré pour la première fois par Nash en 1952, et fut central dans la conception même du jeu. La preuve utilise un un argument dit "du vol de stratégie". Mais, bien que l'on connaisse l'existence d'une telle stratégie, elle est en pratique inconnue pour les grands plateaux (supérieurs à 7×7), le jeu se révélant rapidement d'une grande complexité.

2 Résoudre Hex

2.1 Pas facile...

Disons le tout de suite : Hex n'est pas un jeu facile pour un ordinateur. La première limitation est combinatoire. Une partie classique de Hex se joue sur un plateau d'au moins 10×10 (et plus généralement 11×11 ou 13×13, jusqu'à 17×17 au plus haut niveau). Cela signifie environ 10^{46} positions possibles, à titre de comparaisons, ce nombre s'élève à 10^{43} aux échecs et 10^{170} au go (il serait le même si le plateau de go était de taille 10×10 aussi). Ces chiffres rendent obsolète toute méthode telle que l'exploration exhaustive de graphe à la recherche d'une position gagnante ou la recherche d'attracteurs à l'intérieur de ce même graphe.

En 1976, le caractère PSPACE-Complet d'une version généralisée du jeu de Hex, dit jeu de Shannon (Shannon Switch Game Vertex), a été démontrée[2], la preuve spécifique à Hex datant de 1981[7]. Ces preuves apportent la certitude que le développement d'un algorithme déterministe pour *résoudre* le jeu de Hex est illusoire, et ouvrent ainsi la porte à la recherche de stratégies nouvelles et d'heuristiques plus poussées afin de tenter seulement de rivaliser avec les meilleurs joueurs humains.

Il est important de garder à l'esprit qu'un joueur aura à chaque position à peu près une centaine de coups possible. L'arbre des possibilités qu'y en résulte est immense, notamment à cause de ce fait, le jeu a été comparé au Go en termes de complexité de résolution.

2.2 Joueurs actuels

Le premier joueur réellement performant fut présenté par Anshelevich en 1998. Baptisé Hexy, c'est un joueur informatique qui se distingue par l'exploitation de caractéristiques propres au jeu[1]. Celui-ci s'imposa comme le premier programme informatique capable de rivaliser avec les meilleurs joueurs, avant d'être surpassé par la version finale de Queenbee[8], proposée par van Rijswijck en 2000, qui utilise des versions poussées d'élagage alpha-beta et du machine learning. Ces approches basées sur le parcours optimisé d'arbre atteignirent leur aboutissement en 2008 avec Wolve, puis furent dépassées par l'usage de la recherche Monte-Carlo avec MoHex, tous les deux de Henderson[5]. MoHex est le meilleur joueur informatique à ce jour sur Hex, et est très largement inspiré des meilleurs algorithmes de Go, développés en parallèle, comme MoGo, puis AlphaGo.

2.3 Notre joueur

Conscient de cette difficulté, nous tenterons alors de proposer un joueur qui, s'il ne sera évidemment parfait, sera doté d'une capacité de jeu raisonnable face à un joueur humain amateur d'un niveau moyen.

Tout au long de notre cheminement, l'objectif suivant fut le fil rouge pour jauger des capacités du joueur en developpement : on cherchera à developper un joueur capable de répondre de manière pertinente en 30s sur un plateau de taille 10×10.

Le critère d'efficacité d'un coup étant particulièrement difficile à jauger de manière réellement pertinente sans faire affronter notre joueur à un grand nombre de joueurs humain (afin de lui donner un elo), ces appréciations seront purement basés sur notre expérience personnelle du jeu et de la théorie autour de celui-ci, qui, heureusement, est abondante. Comme nous le verrons, cela suffira pour en tout cas distinguer la qualité des coups entre nos différentes itérations.

En commençant naturellement par explorer la recherche Alpha-Beta, notre approche nous conduira rapidement à nous intéresser aux propriétés propres à Hex. On en déduira alors un algorithme crucial ainsi qu'une heuristique pour notre joueur. Enfin, nous concentrerons nos derniers efforts sur les moyens de rendre notre joueur plus efficace et réellement utilisable en conditions de jeu.

3 Une première approche : Alpha-Beta

On notera $G_{pos} = (V, E)$ le graphe des positions du jeu. On dit qu'il existe une arête d'une position A à une position B s'il existe un coup qui permet de passer de A à B. C'est dans ce graphe que l'algorithme minmax, et son optimisation Alpha-Beta, opère. Algorithme incontournable, son implémentation est donc une première approche naturelle. Dans toute la suite, on ne prendra pas en compte le nœud de départ dans la profondeur de recherche. $d = 0$ correspond donc à évaluer tous les positions voisines depuis une position (visite de tous les voisins directs dans G_{pos}).

3.1 Une première heuristique simple

Dans l'optique d'appréhender les capacités de notre recherche Alpha-Beta, il convient de l'essayer d'abord avec une heuristique simple et peu coûteuse. Une option classique est l'évaluation du score d'un plateau donnée en attribuant en amont un nombre de points fixé à chaque position du plateau. Les coups au centre du plateau sont par exemple très puissants, notamment en début de jeu. A l'inverse, les coups collés à la bordure de l'adversaire sont en général assez mauvais. On souhaite donc que notre plateau de point reflète ceci.

Nous avons alors repris une idée de Ryan Hayward[4] pour générer ces plateaux. Nous avons généré un grand nombre de parties entre deux joueurs jouant aléatoirement, en ajoutant 1 point sur la position du coup final de chaque partie. De ce fait, nous mettons en valeur les positions qui interviennent dans des stratégies gagnantes. Nos plateaux ainsi génerés étaient bien cohérents avec ceux présentés par Hayward.

Figure 2: A gauche, l'exemple fourni par Ryan Hayward. A droite, le résultat de notre implémentation. On remarque que les positions centrales sont bien avantagées, au contraire de celles sur les angles aigus.

3.2 Résultats

Les résultats sont en revanche sans appel : Alpha-Beta se retrouve effectivement démuni devant un jeu aussi complexe que celui de Hex. Nos tests montrent ainsi que même avec une heuristique aussi peu coûteuse (l'évaluation d'une position prends environ 2 µs), il est impossible sur un plateau de taille 8×8 de dépasser $d = 6$ sans que le temps de calcul devienne déraisonnable.

Il est intéressant de remarquer que les meilleurs joueurs humains ont généralement 8 à 9 coups d'avance. Notre joueur est completement dépassé.

Poursuivre dans cette voie est donc inenvisageable, à moins d'appliquer à la recherche des optimisations poussées (c'est la stratégie employée par Queensbee), ou de substituer à la profondeur de recherche une heuristique très solide. Fort heureusement, le jeu de Hex cache des propriétés d'une élégance rare, que nous chercherons à utiliser à notre avantage, et ainsi se diriger vers la 2e option.

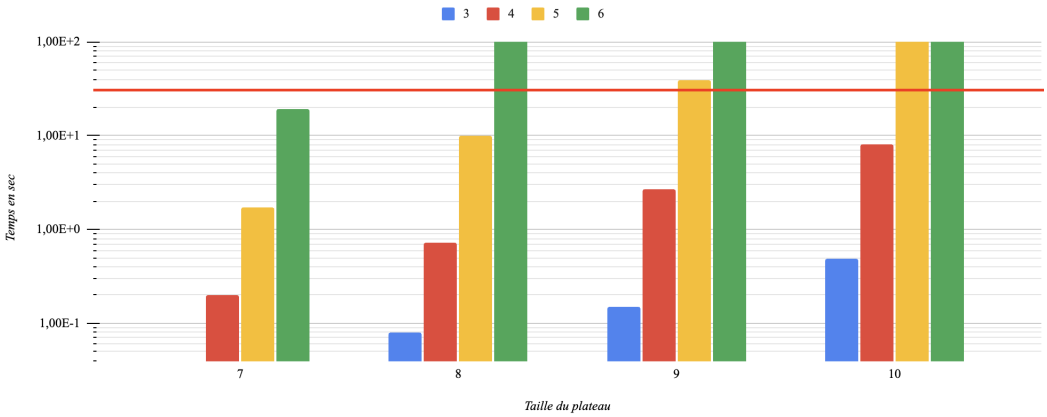

Figure 3: Durée d'exécution de la recherche Alpha-Beta pour différentes valeurs de d (couleurs) et n. En rouge, l'objectif fixé de 30s.

4 La théorie autour du jeu de Hex au service de notre joueur

L'approche envisagée dans les deux sections suivantes est inspirée de l'approche utilisée chez Hexy. Vadim V. Anshelevich la décrit dans les grandes lignes dans le papier suivant : [1], qui fut notre guide, tout en laissant une belle marge d'initiatives personnelles. On remarquera que même ses successeurs MoHex ou Wolve, reprennent les principes développés ci-après afin de guider leur choix de coup : ce sont des concepts très puissants pour l'étude du jeu de Hex.

4.1 Connections virtuelles, semi-virtuelles, et règles de déduction

En jouant au jeu de Hex, on s'aperçoit rapidement que deux positions peuvent être *de facto* reliés même si un chemin de couleur ne les relie pas explicitement. Autrement dit, il est possible de construire ce chemin quelques soient les coups adverses.

Figure 4: Deux situations classiques de ce cas de figure

Ce type de connection est formalisé à travers deux catégories, les *virtual connections* et les *semi-virtual connections*, que l'on définit de la manière suivante (ici du point de vue du joueur 1):

Definition 4.1: Sub-game

Soit (x, A, y) où x, y sont deux positions du plateau, et A un ensemble de position du position tel que x ou y n'y soient pas. On appelle un tel triplet un *sub-game*.

Definition 4.2: Virtual connection

(x, A, y) est une *virtual connection* pour le joueur 1 ssi 1 peut relier x et y par des positions de A même si 2 joue en premier. Elle est dite *minimale* s'il n'existe pas de (x, A', y) connection virtuelle où $A' \subsetneq A$

Definition 4.3: Virtual semi-connection

(x, A, y) est une *virtual semi-connection* pour le joueur 1 ssi 1 peut relier x et y par des position de A *seulement* s'il joue en premier. Le critère de minimalité est identique.

Les deux règles suivantes permettent de construire de telles connections de manière inductive :

Propriété 4.1: règle AND

Soit (x, A, u) et (u, B, y) deux *virtual connections*, telles que $x \neq y$, $x \notin B$, $y \notin A$ et $A \cap B = \emptyset$ Alors :

- Si u est occupé par 1 : $(x, A \cup B, y)$ est une *virtual connection*.
- Si u est vide : $(x, A \cup B \cup \{u\}, y)$ est une *virtual semi-connection*.

Propriété 4.2: règle OR

Soit la famille de *virtual semi-connection* $(x, A_k, y)_{1 \leq k \leq n}$ entre les positions x et y. Alors, si $\bigcap_{k=1}^{n} A_k = \emptyset$ on a que $(x, \bigcup_{k=1}^{n} A_k, y)$ est une *virtual connection*.

4.2 Le HSEARCH

Le HSEARCH est un algorithme qui construit les connections virtuelles *minimales* d'un plateau donné par itération des règles AND et OR depuis un ensemble de connexions virtuelles donné (par exemple, celles triviales, c'est-à-dire les positions adjacentes).

Son principe général est le suivant : Pour toutes positions $g, g_1, g_2 \in [\![1, n]\!]^2$ l'algorithme regarde si il peut appliquer la règle AND à un des couples de *sub-game* (g_1, C_1, g) et (g, C_2, g_2) construit précedemment dans l'algorithme. Si c'est le cas, il ajoute alors la nouvelle connexion virtuelle ou semi-virtuelle à celles connues. De plus, si c'était une connection semi-virtuelle, il va alors chercher si il ne peut pas appliquer la règle OR à un sous-ensemble des connections semi-virtuelles (g_1, C_k, g_2) connues. Ce processus est répété jusqu'à ce qu'il n'y ait plus de nouvelle connexion.

Certaines conditions sont aussi ajoutées pour éviter des calculs redondants, et pour ne générer que les connexions virtuelles *minimales*, qui sont les seules réellement pertinentes.

4.3 Resultats

Ces règles, et par extension le HSEARCH permettent-elles de générer toutes les connexions virtuelles ? Il se trouve que ce n'est pas le cas (un contre-exemple est disponible en annexe). Ceci dit, la plupart des connexions virtuelles restent obtenables. Ce site[1] recense la plupart des connections virtuelles classiques : en tout, 70.7% d'entre elle ont pu être reconstruites grâce à notre implémentation du HSEARCH. Celui-ci fournit donc malgré tout des informations très avantageuses sur le plateau.

Il faut quand même remarquer que ce calcul est coûteux. Pour un plateau 10×10, le calcul des connexions virtuelles par notre implémentation est de l'ordre de quelques dixièmes de secondes. Seule la qualité de jeu de notre joueur pourra nous dire si le gain en information vaut un tel coût temporel.

5 Utiliser ces calculs pour notre joueur

5.1 Une fonction d'évaluation

Les connexions obtenues nécessitent un traitement assez subtil. Une bonne manière de voir cela est de considérer un chemin que le joueur 1 essaie de construire. Sans utiliser de pièces extérieures, il

[1] `http://www.mseymour.ca/hex_book/hexstrat.html`, regarder *interior template* et *edge template*

est probable que le joueur 2 parvienne à le bloquer. En revanche, si le joueur a deux (ou plusieurs) chemins alternatifs en réserve, ou si une de ses pièces placée plus tôt dans la partie se révèle être dans position judicieuse, alors il est probable que le joueur 1 parvienne à relier. Autrement dit, les connexions à l'intérieur du plateau doivent être évaluées comme un tout. Shannon a ainsi construit en 1953 une machine analogique qui utilisait la résistance (électrique) du plateau, et qui jouait selon ses dires de manière assez remarquable contre des joueurs humains[9]. On peut alors envisager une approche similaire. On définit la résistance d'une position (relativement au joueur 1 ici):

$$r_{a,1} = \begin{cases} 0 & \text{si la case est occupée par le joueur 1} \\ 1 & \text{si la case est vide} \\ \infty & \text{sinon} \end{cases}$$

Puis celle d'une connexion :

$$R_{ab,1} = \begin{cases} r_{a,1} + r_{b,1} & \text{si a b sont connectés (virtuellement ou non)} \\ \infty & \text{sinon} \end{cases}$$

On a alors un (gros) circuit de résistances, et en appliquant un voltage entre les deux bords du joueurs 1, on obtient R_1 la résistance équivalente du jeu pour ce joueur. En faisant de même pour le joueur 2, on en déduit finalement la fonction d'évaluation suivante :

$$E = \log\left(\frac{R_2}{R_1}\right)$$

5.2 Résistance effective d'un graphe

La littérature est abondante sur ce problème, dont la résolution peut en effet se montrer assez coûteuse pour un graphe de taille conséquente, et de nombreux algorithmes très poussés existent. Cependant, au vu de la taille relativement limitée de notre graphe (env. 100), notre approche sera bien plus modeste (quoiqu'inédite dans la littérature consultée). On considère le circuit suivant pour simplifier le calcul de r_{eq} :

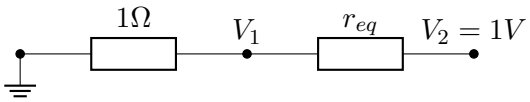

Figure 5: Circuit virtuel utilisé afin de calculer r_{eq}. Il est important de se souvenir que r_{eq} représente un gros circuit de résistors.

Dans ce circuit, on détermine alors les potentiels en tout point du circuit en appliquant autant de Loi des Noeuds en terme de Potentiel (Théorème de Millman) qu'il y a de position dans le graphe. Cette loi permet de donner le potentiel d'une position en fonction des potentielles de ses voisines. Son expression usuelle est donnée en annexe. On obtient alors un système linéaire $Ax = b$ à $n \times n$ inconnues (où n la taille du plateau), que l'on résout en inversant la matrice. Puis, on en déduit le courant $i = V_1/1$ et $r_{eq} = (1 - V_1)/i$.

Encore une fois, de nombreux algorithmes très performant existent pour résoudre un système linéaire (d'autant que A est éparse dans notre cas). Cependant, on remarque que le choix de la LNTP engendre un système de n^2 inconnues, et non n^4 qu'une application naïve des lois de Kirchoff donnerait (les inconnues étant alors les *différences* de potentiels). De ce fait, notre matrice reste de taille raisonnable (env. 100) et malgré cette implémentation sous-optimale, le calcul de la résistance d'un circuit reste de l'ordre de la milliseconde, c'est à dire négligeable devant le HSEARCH. Nous ne concentrerons donc pas nos efforts sur ce calcul.

5.3 Résultat comme heuristique brute

Les résultats de notre implémentation sont limités. D'un côté, les coups proposés par le joueur sont effectivement d'une pertinence remarquable. Sur des petits plateaux, le joueur parvient à identifier

correctement les zones risquées du plateau et à bloquer efficacement son adversaire.

D'un autre côté, cette considération à été faite sur des petits plateaux (comme 5×5) pour la simple raison que le temps de calcul devient rapidement trop coûteux pour être exploitable, à la fois sur les grands plateaux, mais aussi en augmentant la profondeur du parcours Alpha-Beta (celle-ci ne peut dépasser 0 à partir de 7×7), comme on peut le voir figure 6. Bien que l'on puisse affirmer que le HSEARCH est aussi un moyen pour anticiper plusieurs coups à l'avance (jusqu'à 8 expérimentalement), une profondeur d suffisante reste essentielle pour offrir la meilleure qualité de jeu.

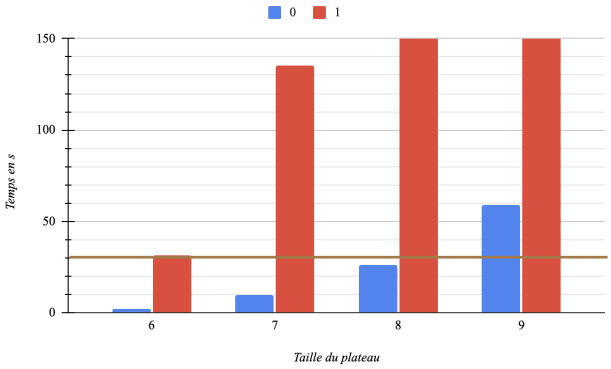

Figure 6: Durée d'exécution pour différentes valeurs de d (couleurs) et n. En marron, l'objectif fixé de 30s.

Il est donc à présent crucial de trouver un moyen d'améliorer le temps de calcul, sans perdre en qualité de prédiction.

6 Rendre notre joueur utilisable

6.1 Propager les calculs

Une première considération est assez évidente : jusqu'à présent, notre programme recommence de zéro (des connexions triviales) le HSEARCH à chaque position à évaluer. Au vu du coût exorbitant de celui-ci, un premier moyen serait de chercher à limiter les calculs redondants, et en cherchant plutôt à propager les connexions virtuelles connues à travers plusieurs coups consécutifs. Ainsi, notre première optimisation intègre une mise à jour dynamique des structures (dans notre implémentation : des tables de hachages) contenant les connexions virtuelles, afin de retirer uniquement celles impactées par l'ajout d'une nouvelle pièce. De ce fait, lors de la recherche Alpha-Beta, le HSEARCH ne s'effectue *in fine* que sur une portion bien plus limitée du plateau. Cette optimisation a donné les meilleurs résultats lorsque le HSEARCH était recalculé juste après la mise à jour du plateau (donc plusieurs fois par branche, en ne réparant qu'une petite portion des connexions à chaque fois).

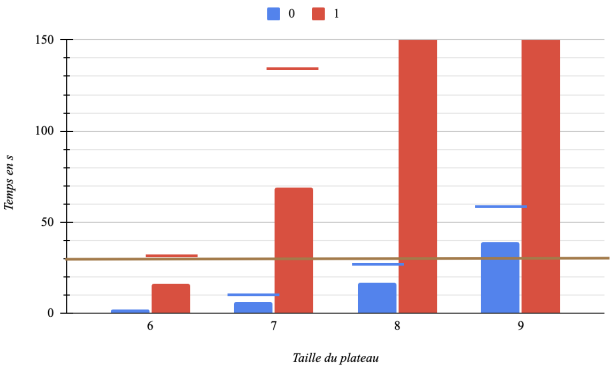

Figure 7: Durée d'exécution pour différentes valeurs de d (couleurs) et n. En marron, l'objectif fixé de 30s. Les barres de couleur donnent les précédentes valeurs

Les résultats de notre implémentation sont visibles figure 7. Cette optimisation a pu diviser les temps de calculs jusqu'à 2.

6.2 Ordonner les coups

L'ordonnation de coups (*move ordering* en anglais) est une composante cruciale de tout joueur informatique sur des jeux complexes. C'est lui qui permet de s'assurer que le joueur étudie en priorité les positions réellement pertinentes, sans perdre de temps de calcul sur des positions peu avantageuses. Notre approche fut inspirée par celle exposée ici[6], mais notre implémentation est complètement différente, puisque on a fait le choix de réutiliser encore une fois les informations issues du HSEARCH. Ainsi, chaque position s'est vu attribuer un compteur, qui, selon le joueur, indique le nombre de connexions semi-virtuelles dans laquelle cette position est impliquée. Plus ce nombre est élevé, plus est raisonnable de penser que cette position est cruciale pour la suite du jeu de l'adversaire.

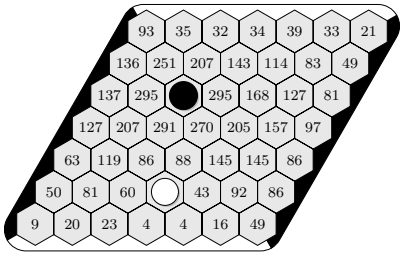

Figure 8: Pour chaque position, le nombre de connection semi-virtuelle noire auquelle elle appartient selon le HSEARCH. Le joueur (blanc) se concentre alors vers les zones sensibles (autour du pion noir) et sera détourné des zones à faible interêt (entre son pion et son bord).

A l'inverse, les positions qui n'interviennent que très peu dans ces connexions sont écartées. Enfin, si l'adversaire est sur le point de relier ses bords (Il existe une connexion semi-virtuelle entre ses bords du plateau), alors notre joueur ne se concentrera uniquement sur les positions concernés (ce qui retire un nombre conséquent de positions à évaluer).

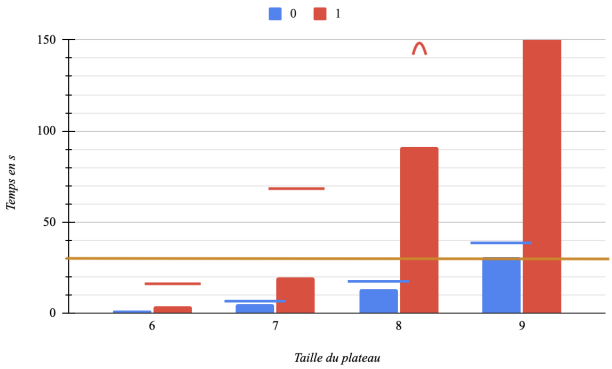

Figure 9: Durée d'exécution pour différentes valeurs de d (couleurs) et n. En marron, l'objectif fixé de 30s. Les barres de couleur donnent les précédentes valeurs

Les résultats de notre implémentation sont visibles figure 9. Cette optimisation fut cruciale, divisant jusqu'à un facteur 4 le temps de calcul pour $d = 1$ voire 10 pour $d = 2$ (non visible sur la figure).

6.3 Poser une limite de temps avec l'IDDFS

Malgré tous nos efforts, notre joueur reste particulièrement lent sur les grands plateaux. Nous sommes alors contraints de le limiter en temps. Une optimisation classique fut réalisée pour cela : notre joueur dans sa version finale effectue un Alpha-Beta itéré sur des profondeurs successives (sur le principe du parcours en profondeur itéré, ou IDDFS) jusqu'à être à court de temps.

Il va de soi que cette optimisation, quoique essentielle pour rendre le joueur utilisable, laisse un impact significatif sur la qualité de jeu de notre programme.

Enfin, on peut noter que l'implémentation originale d'Hexy fonctionne avec $d = 2$. Nous sommes encore loin de l'égaler.

7 Conclusion

L'implémentation d'un joueur informatique dans un jeu aussi complexe que celui de Hex ouvre la voie à un grand nombre de possibilités. Celle choisie et menée à terme ici, afin de d'obtenir un joueur utilisable, n'est pas totalement satisfaisante. L'objectif évoqué dans la partie 2.3 n'a été qu'artificiellement atteint, la limite de temps bridant excessivement notre joueur. Cela illustre bien la difficulté que représente le développement de tels joueurs, et les limites des algorithmes classiques autour des jeux à deux joueurs. Cependant, notre implémentation n'est pas dénuée de qualité puisqu'elle propose malgré tout des coups d'une très bonne qualité, qui en font un succès de ce point de vue. On a pu aussi souligner la possibilité de réaliser de tels joueurs sans reposer totalement sur des recherches dans G_{pos}, qu'elles soient Alpha-Beta ou Monte-Carlo, mais davantage en s'intéressant aux propriétés propres au jeu étudié ici.

References

[1] Vadim V. Anshelevich. "A hierarchical approach to computer Hex". In: *Artificial Intelligence* 134 (2002), pp. 101–120.

[2] S. Even and R. E. Tarjan. "A combinatorial problem which is complete in polynomial space". In: *Journal of the Association for Computing Machinery* 23 (1976), pp. 710–719.

[3] David Gale. "The game of Hex and the Brouwer fixed point theorem". In: *American Mathematical Monthly* 86.10 (1979), pp. 818–827.

[4] Ryan B. Hayward. *Hex. A playful introduction.* American Mathematical Society, 2020. ISBN: 9781470464929.

[5] Thomas Philip Henderson. "Playing and solving the game of Hex". PhD thesis. University of Alberta, 2010.

[6] Rune K. Rasmussen, Frederic D.Maire, and Ross F. Hayward. "A Move Generating Algorithm for Hex Solvers". In: *Artificial Intelligence* 4304 (2006), pp. 637–646.

[7] Stefan Reisch. "Hex ist pspace-vollständig". In: *Acta Informatica* 15 (1981), pp. 167–191.

[8] Jack Van Rijswijck. "Computer Hex: Are Bees Better Than Fruitflies ?" MA thesis. University of Alberta, 2000.

[9] C. E. Shannon. "Computers and Automata". In: *Proceedings of the Institute of Radio Engineers* 41 (1953), pp. 1234–1241.

8 Annexe

8.1 Une connection virtuelle que HSEARCH ne trouve pas

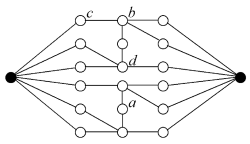

Figure 10: Si le joueur adverse joue en a, on joue en b. Il doit alors jouer en c, et on relie en jouant en d