

IA : Méthodes et Algorithmes

Projet – Agent intelligent pour le jeu Divercité

Adam Nhaila (2413574) et Adrien Telitchko (2413607)

13 novembre 2024

Table des matières

1	Méthodologie	2
2	Résultats et évolution de l'agent	3
2.1	Heuristiques	3
2.2	Métrique de performance 1 : Ratio V/D des matchs en local en fonction de la version de notre agent	4
2.3	Métrique de performance 2 : ELO sur Abyss	5
3	Discussion	6
3.1	Avantages	6
3.2	Limites	6
3.3	Pistes d'amélioration de l'agent	6
4	Références	6

1 Méthodologie

Commençons par la méthodologie et les choix de conception faits pour notre agent. Nous avons fait le choix, comme recommandé, d'utiliser une stratégie basée sur l'algorithme Minimax. Nous avons fait l'hypothèse que nous faisons face à un agent logique et que, par conséquent : le premier joueur veut maximiser son score (joueur MAX) et le deuxième joueur veut minimiser le score du premier joueur (joueur MIN). L'idée est donc de considérer que l'adversaire (et nous-même) allons jouer au mieux pour réaliser notre objectif.

Comme le temps pour jouer est limité (15 minutes) et que le nombre d'états total possible est très grand à chaque tour du jeu Divercité, nous avons décidé de limiter la profondeur de recherche de notre agent à chaque tour grâce à un paramètre de profondeur. Ainsi, nous avons implémenté une heuristique afin d'orienter la recherche de l'agent vers les états les plus avantageux.

Choix d'initialisation du score : Score de mon agent – Score de l'adversaire

En ce qui concerne la stratégie elle-même, notre objectif a été d'obtenir un maximum de divercités (+5 points supplémentaires). De même, nous avons décidé d'empêcher au maximum l'adversaire d'avoir des divercités en mettant un malus lorsqu'il en fait une (-2 points). Un autre choix a été de récompenser notre agent lorsqu'il pose ses cités. Cela permet à l'agent de poser ses cités en priorité en début de partie plutôt que des ressources qui seraient ainsi posées sans nous faire gagner de point. La récompense est définie par :

$$\text{Récompense} = 2 \times (8 - (\text{nombre de cités restantes de la couleur concernée}))$$

Au niveau des ressources, nous avons décidé de pénaliser le fait d'avoir plus d'une ressource de la même couleur autour d'une cité de cette même couleur, et ce, pendant la moitié de la partie environ. Les pénalisations sont définies comme suit :

- Première pénalisation : -1 point pour chaque stack de 2 couleurs identiques, et ce, jusqu'au 11^e coup de notre agent.
- Deuxième pénalisation : -1 point pour chaque stack de 3 couleurs identiques, et ce, jusqu'au 12^e coup de notre agent.

Finalement, toujours pour la gestion des ressources, on effectue une pénalisation si 3 ressources disponibles de la même couleur et 0 pour une autre.

Concernant la gestion du temps, nous avons décidé de gérer de manière dynamique la profondeur de recherche dans la fonction `compute_action`. La profondeur de recherche est donc petite au début (trop d'états à analyser et faibles répercussions de nos choix sur le reste de la partie), puis elle augmente au fur et à mesure que le nombre de coups joués augmente (jouer de bons coups devient très important car chaque coup a une grande répercussion sur le reste de la partie).

Nous avons ajusté empiriquement cette profondeur au fur et à mesure de l'avancement dans la partie, en nous assurant qu'il restait toujours assez de temps à la fin de la partie pour notre agent (temps total pour jouer < 15 minutes). Pour cela, nous avons testé notre agent contre un joueur aléatoire, le joueur *greedy* et nos anciennes versions de l'agent, et vérifié que le temps restant à la fin de la partie était supérieur à 0.

Dans un souci d'efficacité, nous avons implémenté la méthode *alpha-beta pruning* afin de diminuer la complexité temporelle de calcul et donc de pouvoir augmenter par la suite la profondeur de recherche grâce au temps ainsi gagné.

Tous nos choix de conception seront évidemment discutés dans la partie dédiée à la discussion.

2 Résultats et évolution de l'agent

Ces choix de conception énoncés précédemment sont le fruit de beaucoup de réflexion et d'analyse de nos parties jouées sur Abyss. Nombre de parties jouées au 06/11/2024 = 1155.

Détaillons version par version les modifications de nos heuristiques. Précision importante : de la V1 à la V5, nous n'avions pas remarqué un bug dans notre fonction `minValue` qui empêchait nos agents d'aller à une profondeur supérieure à 1 !

Nous avons compris l'importance de ne pas négliger le temps que peut prendre le debugging du code.

2.1 Heuristiques

Pour toutes les versions, l'initialisation du score est la même : Score de mon agent – Score de l'adversaire.

- **V1** : En plus du score, initialisation d'un **best score** = Score de mon agent – score de mon adversaire. On parcourt la liste des actions possibles depuis l'état actuel. Pour chaque nouvel état obtenu, on initialise de nouveau le score avec : score de notre agent à ce nouvel état – score de l'adversaire à ce nouvel état. Si la différence des 2 scores ≥ 5 , un bonus de 5 points est ajouté à la variable score. Calcul du nombre de divercités puis ajout de 5 points par divercité trouvée. Si le score est supérieur au **best score**, alors **best score** devient le score ainsi calculé.

Une première remarque est qu'il peut paraître surprenant de parcourir de nouveaux états à l'intérieur même de notre heuristique. Cependant, nous avons pensé que cela permettrait d'aller « tendanciellement » vers les meilleurs états et que cela pourrait être mieux que d'aller vers les meilleurs états directement.

- **V2** : Identique à V1, mais avec un repérage des divercités que possède notre joueur, avec une implémentation dans le code beaucoup plus claire.
- **V3** : En plus de repérer les divercités de notre agent, on pénalise de -1 le score si on repère une divercité pour l'adversaire.
- **V4** : On ajoute des points si une cité est posée en parcourant la liste de nos pièces. Si la pièce est une cité : **score** $+= 2 \times (2 - \text{nombre de cités restantes de cette couleur})$. Cela encourage l'agent à poser des cités afin que le score soit maximisé rapidement, tout en laissant une marge de manœuvre car poser une ressource peut rester plus avantageux parfois.
- **V5** : Identique à V4 sauf qu'on favorise encore plus le dépôt de cités le plus rapidement possible : **score** $+= 2 \times (8 - \text{nombre de cités restantes de cette couleur})$.

De la V1 à la V5, la gestion de la profondeur n'est pas dynamique et est fixée arbitrairement à 10, puisque de toute façon notre agent est encore incapable d'aller à une profondeur supérieure à 1 à cause du bug dans `minValue`.

- **V6** : Résolution du bug. Profondeur fixée à 2. Première pénalisation = -1 point pour chaque **stack** de 2 couleurs identiques. Deuxième pénalisation = -1 point pour chaque **stack** de 3 couleurs identiques. Ces pénalisations sont effectuées tout au long de la partie (jusqu'au 20^e coup de l'agent puisqu'on possède 20 pièces). Implémentation de

l'alpha-beta pruning pour réduire le temps de calcul.

- **V7** : -2 points si on repère une divercité réalisée par l'adversaire, au lieu de -1.
- **V8** : On arrête d'explorer de nouveaux états au sein de l'heuristique et se concentre uniquement sur l'état actuel pour aller directement vers les meilleurs états évalués. On reprend les pénalisations précédentes pour plusieurs ressources de la même couleur autour d'une cité, sauf que cette pénalisation ne s'applique plus que pour les 4 premiers coups de l'agent. Introduction d'une gestion dynamique de la profondeur en fonction du nombre de coups joués : 2 si nombre de coups ≤ 10 , 3 si $10 < \text{nombre de coups} < 14$, 4 pour le reste de la partie.
- **V9** : Observation que les adversaires sur Abyss ne jouent jamais de jaune en premier, donc ajout de +4 au score si on pose du jaune dans les deux premiers coups. Pénalisation de score si des cités restent non posées :

$$\text{score-} = 2 \times (8 + \text{nombre de cités restantes dans cette couleur})$$

- **V10** : Implémentation simplifiée, similaire à V8, mais avec une meilleure gestion des ressources en pénalisant -2 points si on a 3 ressources de la même couleur et 0 pour une autre. Pénalisation = -1 point pour chaque **stack** de 2 couleurs identiques autour d'une cité, jusqu'au 11^e coup de l'agent, et -1 pour chaque **stack** de 3 couleurs jusqu'au 12^e coup.

Comme vous pouvez le constater, certains choix de conception (surtout dans les premières heuristiques) étaient assez surprenants voire naïfs. Notre compréhension du projet et du fonctionnement de l'algorithme Minimax était encore limitée à ce moment-là.

Analysons maintenant l'évolution des performances des agents à la fois en les testant entre eux, mais aussi en regardant l'évolution de l'ELO sur Abyss ainsi que le ratio Victoire/Défaites.

2.2 Métrique de performance 1 : Ratio V/D des matchs en local en fonction de la version de notre agent



FIGURE 1 – Ratio V/D des matchs en local en fonction de la version de notre agent

Ici, il semblerait que la version 7 soit la plus performante (matchs mutuels entre nos agents). Nous regardons essentiellement la performance de nos agents face aux versions précédentes les plus récentes pour savoir s'il était pertinent ou non de la mettre sur Abyss. (Voir annexe 1)

2.3 Métrique de performance 2 : ELO sur Abyss

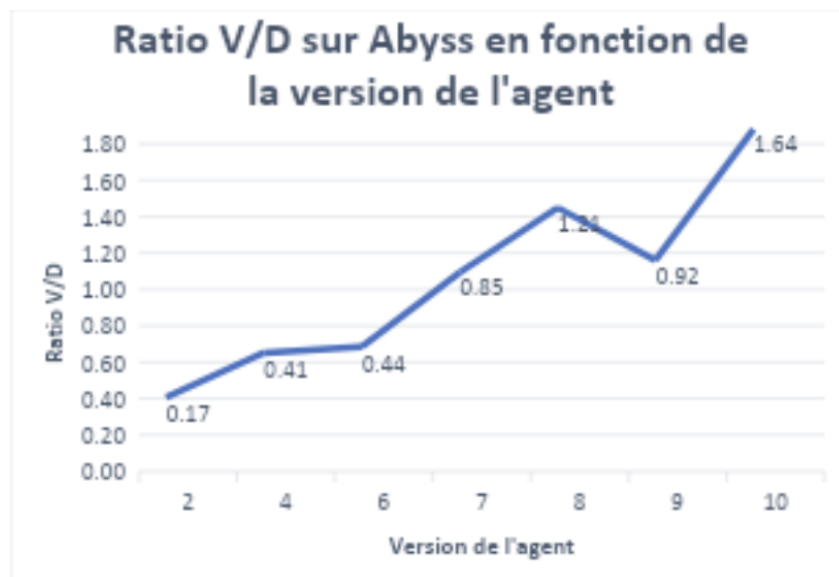



FIGURE 2 – Ratio V/D sur Abyss en fonction de la version de l'agent

Nombre total de matchs joués à ce jour sur Abyss : 1265.

Même si la V10 ne semble pas être la plus performante de nos agents en local, elle est cependant largement en tête devant les autres versions sur Abyss.

NOM DE L'AGENT	PERFORMANCE	ELO	ACTIF
V10=V8+RM+clearcod	141V-86D	1114	<input checked="" type="checkbox"/>
V9=V8+opti+yell	102V-116D	877	<input type="checkbox"/>
V8abCorrectedPenalDivOppoBeginFixForDeleteDepthDyna	134V-111D	951	<input type="checkbox"/>
V7abCorrectedPenalDivOppo	115V-135D	817	<input type="checkbox"/>
V6abCorrected	23V-52D	556	<input type="checkbox"/>
V4citydiv	66V-161D	490	<input type="checkbox"/>
Dauphin des MaraisV2	4V-24D	493	<input type="checkbox"/>



Ligue Requin

Peak ELO
1321

Série de victoire
6

Victoires
451

Défaites
574

FIGURE 3 – Résultats des agents sur Abyss

N.B : V8 exclue pour dépassement de la limite de temps

De façon similaire, même si la variation dans les ligues est assez grande, la V10 passe l'essentiel de son temps dans la ligue Narval, et c'est cette même version qui nous a permis d'atteindre une fois le pic à 1321 (1^{er} dans la ligue Baleine avec 44 joueurs à ce moment-là sur Abyss).

Nous pouvons en tirer un enseignement : il faut aussi se tester par rapport aux autres

pour pouvoir évaluer l'efficacité de notre agent. En effet, il suffit que la stratégie adoptée soit légèrement différente de ce que les adversaires anticipent pour bien performer. La V7 et la V9 sont les plus performantes en local mais sont loin d'être les meilleures sur Abyss.

3 Discussion

3.1 Avantages

- Une stratégie pour obtenir des divercités payante. En moyenne, sur 20 parties prises au hasard sur Abyss, notre agent réalise 2,4 divercités par partie, soit 13 points apportés en moyenne par partie grâce aux divercités.
- Une limite de temps toujours respectée avec une profondeur allant jusqu'à 4.

3.2 Limites

- Un temps restant à la fin de la partie très dépendant de l'adversaire et souvent très largement supérieur à 5 minutes lors des tests en local (voir annexe 2).
- Une gestion des cités non optimale, souvent placées par paires de 2 de la même couleur.

3.3 Pistes d'amélioration de l'agent

- Améliorer la gestion du nombre de cités restantes afin d'éviter qu'il en reste 2 d'une couleur et 0 d'une autre couleur (sauf dans les cas très bénéfiques).
- Adapter la profondeur de recherche en fonction du temps restant dans la partie et l'étape (le nombre de coups joués), et plus seulement en fonction de l'étape de la partie.

Par ailleurs, en ce qui concerne les métriques de performance, le ratio Victoire/Défaites nous est apparu comme étant une mesure plus juste pour la comparaison des agents, plutôt que l'ELO, qui est une mesure très variable d'une partie à une autre. À titre d'exemple, notre agent (V10) est capable de battre certains adversaires à 1200 d'ELO mais aussi de perdre certaines parties contre des joueurs à 950 d'ELO.

4 Références

- Cours du module 2 : <https://moodle.polymtl.ca/pluginfile.php/829732/course/section/55889/M2-recherche-adversaire.pdf?time=1726671350067>
- Ludovox (Youtube) : <https://youtu.be/TrWlP9EzGm0?si=rhFIFd7WcnNeKK93>

Annexes

				(Joueur ligne - Joueur colonne)									
Résultats	Aléatoire	Greedy	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
Aléatoire			4 - 25	3 - 23	7 - 18	4 - 18	7 - 33	3 - 24	8 - 31	12 - 22	5 - 19	6 - 23	
Greedy			21 - 22	17 - 26	17 - 24	17 - 22	17 - 22	15 - 25	15 - 25	20 - 28	18 - 20	20 - 28	
V1	28 - 8	21 - 20		16 - 15	19 - 20	28 - 17	28 - 27	11 - 22	15 - 21	12 - 25	15 - 24	13 - 28	
V2	26 - 7	21 - 20	17 - 18		24 - 20	28 - 27	28 - 27	11 - 22	15 - 21	12 - 25	15 - 24	17 - 27	
V3	23 - 8	22 - 19	26 - 21	24 - 17		22 - 20	22 - 20	16 - 22	16 - 22	13 - 22	14 - 24	17 - 23	
V4	24 - 6	28 - 18	23 - 21	27 - 23	25 - 20		25 - 21	10 - 15	10 - 15	8 - 21	17 - 26	9 - 22	
V5	28 - 7	28 - 18	23 - 21	27 - 23	25 - 20	25 - 21		10 - 15	10 - 15	8 - 21	17 - 26	9 - 22	
V6	31 - 8	27 - 17	27 - 20	27 - 20	24 - 17	15 - 14	15 - 14		17 - 18	14 - 18	22 - 26	22 - 20	
V7	27 - 5	30 - 20	29 - 16	29 - 16	24 - 17	15 - 14	15 - 14	20 - 18		14 - 18	27 - 22	22 - 20	
V8	28 - 8	32 - 20	25 - 16	25 - 16	27 - 14	16 - 13	16 - 13	16 - 17	16 - 17		21 - 20	17 - 16	
V9	32 - 7	23 - 19	26 - 12	26 - 22	24 - 16	24 - 15	24 - 15	24 - 22	24 - 22	28 - 24		17 - 13	
V10	30 - 3	28 - 20	23 - 17	23 - 17	24 - 16	28 - 18	28 - 18	22 - 19	22 - 23	21 - 22	25 - 20		

FIGURE 4 – Résultats des parties en local entre toutes les versions de nos agents

				(Joueur ligne - Joueur colonne)							
Temps restant	Greedy	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
Greedy		899 - 890	899 - 885	899 - 876	899 - 874	899 - 887	899 - 797	899 - 735	899 - 789	899 - 735	899 - 733
V1	883 - 899		885 - 884	890 - 885	891 - 894	890 - 889	887 - 834	888 - 814	888 - 798	886 - 694	888 - 834
V2	894 - 899	885 - 886		889 - 884	891 - 894	889 - 889	885 - 831	887 - 815	887 - 803	888 - 693	888 - 818
V3	887 - 899	875 - 885	879 - 887		882 - 883	883 - 890	880 - 827	881 - 782	880 - 812	880 - 784	883 - 835
V4	882 - 899	872 - 884	873 - 879	881 - 880		877 - 886	880 - 784	876 - 682	881 - 756	876 - 526	878 - 679
V5	892 - 899	883 - 885	883 - 878	887 - 882	885 - 878		887 - 783	884 - 682	886 - 747	889 - 560	887 - 685
V6	815 - 899	742 - 883	736 - 878	779 - 880	630 - 858	752 - 885		765 - 651	882 - 619	727 - 463	820 - 607
V7	729 - 899	631 - 883	642 - 878	714 - 879	667 - 876	654 - 884	686 - 740		775 - 602	639 - 499	775 - 624
V8	586 - 899	719 - 884	702 - 880	716 - 880	287 - 878	257 - 887	477 - 790	465 - 715		458 - 477	205 - 701
V9	616 - 899	605 - 882	691 - 881	767 - 879	503 - 877	454 - 883	480 - 713	468 - 634	386 - 336		477 - 97
V10	661 - 899	673 - 882	736 - 885	769 - 880	546 - 878	555 - 884	613 - 764	436 - 643	459 - 528	328 - 451	

FIGURE 5 – Temps restant à la fin de la partie pour chacun de nos agents (avant le dernier coup). Moyenne = 600 secondes restantes pour V10 mais très dépendant de l’adversaire.