Projet - Agent intelligent pour le jeu Avalam

Agent pour le concours à remettre le 20 novembre 2022 sur Moodle (avant minuit) pour tous les groupes.

Code et Agent à remettre le 8 décembre 2022 sur Moodle (avant minuit) pour tous les groupes.

Rapport à remettre le 8 décembre 2022 sur Moodle (avant minuit) pour tous les groupes.

Consignes (en bref)

- Le projet doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de votre soumission, donnez votre rapport au format **PDF** (matricule1_matricule2_Projet.pdf)
- Lors de votre soumission, votre code `my_player.py` et ses dépendances (**requirements.txt**) doivent être au format **ZIP** (matricule1_matricule2_Projet.zip). Plus de détails sont fournis dans le sujet, notamment pour les dépendances et le fichier **requirements.txt**.
- Indiquez vos nom et matricules dans le fichier PDF et en commentaires en haut du fichier de code soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale.

1 Introduction

Munis de vos nouvelles expertises dans le domaine de l'intelligence artificielle, vous devez maintenant implémenter un agent qui puisse jouer efficacement au jeu Avalam. Avalam est un jeu de stratégie au tour par tour, en un contre un, dont le principe est simple : prendre le contrôle du plateau de jeu en déplaçant 48 pions de façon à former des tours.

Ce plateau sur lequel le jeu se déroule est percé de 48 cercles supportant les pions. Chaque joueur possède 24 pions disposés d'une manière bien définie au départ. À chaque tour, chaque joueur doit effectuer un et un seul déplacement dans n'importe quelle direction (horizontale, verticale, diagonale) et **quelque soit la couleur du pion ou de la tour**. Ce mouvement revient à empiler les pions les uns sur les autres, avec un maximum d'empilement de 5 pions sur une tour. Un joueur est déclaré comme propriétaire de la tour lorsqu'un pion de sa couleur en occupe le sommet.

Ces mouvement doivent cependant respecter quelques règles :

* On doit toujours déplacer une tour complète. Ainsi, chaque mouvement va engendrer une case vide.
* On ne peut jamais poser de pions sur un trou vide.

Les conséquences logiques sont les suivantes :

* Une tour de pions ne peut jamais diminuer ou rester de taille égale lors d'un mouvement.
* Une tour de pions de taille 5 ne pourra jamais se déplacer.
* Une tour de pions (de taille 1 à 5) dont les 8 trous adjacents sont vides ne pourra pas être déplacée.

Le plateau de départ est constitué d'une alternance entre les pions de chaque couleur. Le centre reste vide.

FIGURE 1 – Disposition de départ d'une partie d'Avalam.

La fin de partie est déclarée lorsque plus aucun mouvement ne peut-être effectué sur la plateau de jeu. Le gagnant est décidé en comptant le nombre de pions de chaque joueur situé **au sommet** des tours de pions. La majorité l'emporte. En cas d'égalité, c'est le nombre de tours de taille maximum qui compte. Dans le cas improbable ou il y a encore une égalité, alors il y a une égalité entre les deux agents.

⚠**Nous vous conseillons de lire la liste complète et succincte des règles (dont la majorité sont citées ci-dessus) disponibles sur le lien suivant et de regarder une courte vidéo explicative afin de vous familiariser avec les règles du jeu, disponible sur Youtube. Une version physique du jeu sera également disponible en laboratoire.**

En cas de problèmes, n'hésitez pas à communiquer avec votre chargé de laboratoire à l'aide de Slack.

2 Énoncé du projet

Pour ce projet vous participerez par **équipe de deux** à un tournoi Challonge dont le but est de développer un agent automatique qui puisse jouer le mieux possible à Avalam. Vous êtes libre d'utiliser les méthodes et les bibliothèques de votre choix. Les seules contraintes sont le langage de programmation (**Vous devez implémenter votre agent en python 3**), la possibilité de faire jouer votre agent (votre code compile et retourne une action valide à chaque fois que c'est son tour) et la limite de temps : chaque agent a un budget temps de **15 minutes** à répartir sur l'entièreté de ses actions. Ainsi, outre une stratégie standard d'allouer un même temps d'exécution pour chaque action, il est tout a fait permis à un agent d'utiliser 10 minutes pour effectuer son premier coup, et d'utiliser les 5 minutes restantes pour la réalisation de tous les autres coups.

Votre code devra être soumis sur Moodle sous format **ZIP**. Après avoir reçu tout les agents, nous les feront jouer les un contre les autres. Le tournoi sera divisé en deux phases :

1. **Phase de qualification :** Les différents agents seront séparés dans plusieurs poules. Tous les joueurs dans la poule se rencontrent et chaque rencontre est composée de 5 parties. A l'issue, les joueurs sont classés sur leur nombre de victoires (et le cumul de leur score si égalité). Seul une partie des meilleurs joueurs par poule seront qualifiés pour la phase suivante (un ratio d'environ 50% de la poule).
2. **Phase éliminatoire :** Les joueurs s'affrontent les uns contre les autres en élimination directe jusqu'à la finale et la petite finale.

Sur les cinq manches d'une partie, la couleur blanche/jaune sera attribuée deux fois par joueur. La répartition de la dernière manche est aléatoire. Vous pourrez suivre les résultats de vos agents en live sur le site du tournoi. Celui-ci aura lieu quelques jours après la remise concours. Le lien du tournoi ainsi que la configuration finale du tournoi vous seront fournis ultérieurement.

3 Rapport

En plus d'implémenter votre agent, vous devez rédiger un rapport qui détaille votre stratégie de recherche et vos choix de conception. Celui-ci peut être rédigé au choix **en français ou en anglais**.

Étant donné que vous êtes libre d'implémenter la solution de votre choix, qu'elle soit inspirée de concepts vus en cours, ou de vos connaissances personnelles, nous attendons une explication claire et détaillée de votre solution. Précisément, votre rapport doit contenir au minimum les informations suivantes :

1. **Titre du projet**.
2. **Nom d'équipe** sur Challonge, ainsi que la liste des membres de l'équipe (nom complet et matricule).
3. **Méthodologie :** Explication du fonctionnement de votre agent, des choix de conceptions faits (principes de l'heuristique utilisée, la stratégie choisie, prise en compte du time-out, spécificités propres de votre agent, gestion des 15 minutes allouées, etc.).
4. **Résultats et évolution de l'agent :** Reporter l'amélioration de votre agent en le testant contre vos versions précédentes et les implémentations qui vous sont fournies.
5. **Discussion :** Discutez des avantages et limites de votre agent final.
6. **Références** (si applicable).

Le rapport ne doit pas dépasser **5 pages** et doit être rédigé sur une ou deux colonnes à simple interligne, avec une police de caractère 10 ou plus (des pages supplémentaires pour les références et le contenu bibliographique sont autorisées, ainsi que pour la page de garde). Vous êtes libre de structurer le rapport comme vous le souhaitez tant que vous incluez les éléments mentionnés précédemment.

4 Ressources fournies

Une implémentation Python du jeu Avalam est fournie sur Moodle. Elle implémente l'ensemble des caractéristiques du jeu, et permet à un utilisateur de jouer contre un ordinateur via une interface graphique ou de faire s'affronter deux ordinateurs (deux agents intelligents...).

Avant toute chose, si vous n'êtes pas familier avec la programmation orientée objet en python, vous pouvez consulter cette fiche explicative.

Les fichiers fournis sont :

— `gui.py` : contient le code pour afficher l'interface graphique. **Vous n'avez pas comprendre ce fichier, ni à le modifier.**

— `game.py` : contient l'ensemble des fonctions permettant de lancer une partie, de l'enregistrer et de pouvoir la rejouer. Les détails concernant les commandes pour interagir avec l'environnement (notamment lancer une partie) vous sont fournies plus bas. **Vous n'avez pas à modifier ce fichier.**

— `SimpleHTTPServer.py` : contient le code de lancement d'un petit serveur web pour le gui (interface graphique). **Vous n'avez pas comprendre ce fichier, ni à le modifier.**

— `SimpleWebSocketServer.py` : contient l'implémentation de la socket web permettant d'interagir avec l'interface graphique du jeu. **Vous n'avez pas comprendre ce fichier, ni à le modifier.**

— `avalam.py` : implémente toute la logique du jeu Avalam à l'aide de différentes classes et fonctions. **Vous n'avez pas à modifier ce fichier.** Il est cependant nécessaire de comprendre globalement cette

classe afin de faire interagir vos agents avec l'état du jeu à un instant t. Voici pour l'occasion une brève description du contenu du fichier pour vous aider à mieux l'appréhender :

1. `class Board` : contient tout l'information relative à l'état d'une plateau de jeu. Cette classe contient :

   (a) Un constructeur :

```
def __init__(self, percepts=initial_board, max_height=max_height, invert=
False):
    """Initialize the board.
    Arguments:
    percepts -- matrix representing the board
    invert -- whether to invert the sign of all values, inverting players
    max_height -- maximum height of a tower
    """
    self.m = percepts
    self.rows = len(self.m)
    self.columns = len(self.m[0])
    self.max_height = max_height
    self.m = self.get_percepts(invert)  # make a copy of the percepts
    ...
```

   (b) Des fonctions utilitaires d'application des règles (très utiles !) :

```
def is_tower_movable(self, i, j)
def is_action_valid(self, action)
def get_actions(self)
def get_towers(self)
def get_tower_actions(self, i, j)
```

   (c) Une fonction d'action :

```
def play_action(self, action)
```

   (d) Diverses autres fonctions utilitaires :

```
def clone(self)
def get_percepts(self, invert=False)
def is_finished(self)
def get_score(self)
def dict_to_board(dictio)
def load_percepts(csvfile)
```

2. `class Agent` : Chaque agent fonctionne sur un serveur XML-RPC à part. Il ne vous est pas demandé de comprendre le fonctionnement de ce type de serveur mais sachez qu'il sert à exécuter un script python de manière totalement isolé et sur lequel il est possible d'effectuer des requêtes. Cela permet d'utiliser différents blocs de code (en l'occurrence ici vos agents) sans avoir à regrouper tout ces différents blocs de code ensemble. Cette classe est une classe abstraite. Lorsque vous allez implémenter votre agent, il devra respecter les exigences de la classe `Agent`, et **devra au minimum contenir la fonction :** `play()` dont les arguments sont explicités dans `avalam.py`. Cette fonction prend en entrée l'état actuel du jeu et doit renvoyer la prochaine action à effectuer.

— `my_player.py` : votre agent. **Vous devez modifier ce fichier.**

— `random_player.py` : un agent aléatoire si vous voulez gagner une partie facilement. ;)

— `greedy_player.py` : un agent glouton en guise de baseline.

Pour lancer une partie il faut au préalable lancer un serveur XML-RPC pour chaque agent que vous souhaitez utiliser à l'aide de la commande suivante :

```
python path/to/agent.py --bind (ou -b) ADDRESS --port (ou -p) PORT
```

L'idée est de remplacer l'adresse `ADDRESS` par votre adresse locale ('`localhost`') et le port local `PORT` sur lequel vous voulez que le serveur communique avec votre agent intelligent. Par exemple :

```
python path/to/random_player.py --bind localhost --port 8000
python path/to/greedy_player.py --bind localhost --port 8080
```

⚠**Chaque agent doit être sur un port différent (par exemple 8000 et 8080).**

Une fois le(s) serveur(s) démarré(s) sur leur thread respectif, pour lancer une partie, il vous suffit de lancer la commande suivante :

```
python game.py AGENT1 AGENT2 (--time SECONDS) (--verbose) (--no-gui)
```

Ici, il est nécessaire de remplacer `AGENT1` et `AGENT2` par une adresse locale (`http://localhost:PORT` en remplaçant `PORT` par le port du serveurs XML-RPC de l'agent que vous voulez confronter) ou par '`human`' pour jouer vous même. Par exemple :

```
python game.py http://localhost:8000 http://localhost:8080 --time 900
ou
python game.py http://localhost:8080 human --time 900
```

⚠**Un agent peut tout à fait jouer contre lui même si vous spécifié la même adresse pour les deux joueurs.** L'interface graphique est alors accessible sur n'importe quel navigateur à l'adresse `http://localhost:8030`. L'option *--no-gui* permet d'afficher le déroulé des parties sur le terminal courant.

⚠**Veillez à lancer votre agent sur un port différent de celui de l'interface Web (différent de 8030).**

⚠**Utilisateurs de Windows** : si un problème survient lors de la tentative d'accès à l'interface graphique (*404 Not Found* ou *Unreachable* notamment), nous vous recommandons d'installer l'application **Windows Subsystem for Linux (WSL)** pour lancer les agents et la partie sur un kernel Linux tout en travaillant sur votre machine (et vos fichiers) Windows.

Pour obtenir la documentation complète de `game.py`, exécutez la commande : '`game.py --help`'. On y retrouvera les commandes disponibles afin d'enregistrer et de rejouer une partie par exemple.

Enfin, le serveur d'interface graphique (GUI) étant exécuté en parallèle, il est possible que celui plante, vous empêchant de relancer une partie normalement. Il faut alors le *kill* à la main à l'aide des commandes suivantes :

— Sur **Windows**, la commande suivante vous permet de récupérer l'id du processus lié au GUI :

```
netstat -ano | findstr :8030
>>> TCP    127.0.0.1:8030     0.0.0.0:0      LISTENING       65328
```

Il vous suffit alors d'arrêter le processus <PID> en question avec la commande :

```
taskkill /PID <PID> /F
>>> Operation reussie : le processus avec PID 65328 a ete termine.
```

— Sur **MacOS/Linux** :

```
kill -9 $(lsof -t -i:8030)
```

En cas de problèmes, n'hésitez pas à communiquer avec votre chargé de laboratoire à l'aide de Slack.

⚠**Nous vous conseillons vivement de vous assurer que vous êtes capables de faire tourner le code du projet au plus tôt.**

5 Environnement virtuel Conda

De la même façon qu'au Devoir1, il est important pour la correction de pouvoir aisément reproduire l'environnement sous lequel vous travaillez. Il en va de même pour le travail collaboratif et surtout pour l'organisation du tournoi. Réellement, maîtriser Conda et la gestion des environnements virtuels est un atout important dans votre future carrière.

⚠**L'ensemble des commandes suivantes sont à réaliser avec la console Conda. Se référer à la fiche pratique Conda pour l'installation.**

Création d'un environnement Conda

(base) conda create --name projet python=3.8

*Une validation est demandée par la saisie du caractère **y**.*

Activation de l'environnement virtuel

(base) conda activate projet

*La console Conda devrait afficher **(projet)** comme préfix.*

Lancement d'un agent

(projet) python path/to/random_player.py --bind localhost --port 8000

L'agent random devrait être en écoute sur le port 8000 en attente du début de partie face à un autre agent.

Installation d'une dépendance

Une fois l'environnement actif dans la console Anaconda, on peut simplement utiliser **pip** pour installer un package à l'environnement.

(projet) pip install numpy

*Néanmoins, il faudra donc bien penser à générer le fichier **requirements.txt** afin de nous permettre d'exécuter votre agent dans un environnement fonctionnel, sous peine de ne pas pouvoir concourir au tournoi.*

Requirements

Le projet étant assez libre, il est possible (non essentiel) d'ajouter des librairies telle que numpy par exemple. Il est donc nécessaire de fournir un fichier **requirements.txt** en respectant le formalisme de ce type de fichier, pour que l'on puisse installer votre agent. L'environnement Conda et pip permettent d'exporter simplement ces dépendances via la commande suivante.

```
(projet) pip freeze > requirements.txt
```

*Attention, il faut veiller à exécuter cette commande avec l'environnement activé (présence du préfix entre parenthèse). Il suffit maintenant de fournir le fichier **requirements.txt** dans le rendu ZIP.*

Vérification finale

Afin de s'assurer que l'export nous permettra de rouler votre agent lors du tournoi, il est intéressant de créer un nouvel environnement, d'y installer les dépendances exportées puis d'y faire jouer son agent.

```
(base) conda create --name tmp python=3.8
(base) conda activate tmp
(projet) pip install -r requirements.txt
```

*Si vous êtes capable de rouler une partie entre votre agent et un autre agent via cet environnement **tmp**, vous êtes assurés que nous pourrons installer et utiliser votre agent lors du tournoi et de la notation.*

6 Code à produire

Vous devez remettre un fichier `my_player.py` avec votre code. Pour ce faire, vous être entièrement libre d'utiliser la méthode que vous voulez. Si vous utiliser des librairies externes (`numpy`, `keras`, `pytorch`, etc.), veuillez les lister dans le fichier `requirements.txt` lors de votre soumission comme cela est décrit dans la partie 5.

Au minimum, votre solution devra hériter de la classe `Agent` et implémenter la fonction centrale `play()`. Cette fonction doit retourner une action selon l'état actuel du jeu :

Input : `self:Agent, percepts:dict, player:int, step:int, time_left:float`.

Toute l'information de la partie (placement et taille des tours, couleur des pions, etc.) est contenue dans l'objet `percepts`. Vous pouvez utiliser la fonction `dict_to_board()` dans `avalam.py` pour convertir le dictionnaire `percepts` en une instance de la classe `Board`. Les autres informations présentes (étape en cours, temps restant, etc.) peuvent également aider votre agent à prendre des décisions durant la partie en se repérant mieux dans la chronologie et l'état de son avancement.

Output : `action`

Une action est un tuple contenant 4 éléments (`i1, j1, i2, j2`), définis comme suit :

- `i1 (int)` : coordonnée en ligne (de gauche à droite) de la tour à déplacer.
- `j1 (int)` : coordonnée en colonne (de haut en bas) de la tour à déplacer.
- `i2 (int)` : coordonnée en ligne (de gauche à droite) de la tour à recouvrir.
- `j2 (int)` : coordonnée en colonne (de haut en bas) de la tour à recouvrir.

Il est bon de noter que l'indexage (i,j) se base sur un carré de 9x9 (i et j allant de 0 à 8) englobant le plateau de jeu complet. Comme nous avons pu le voir sur une illustration précédente et comme vous le verrez sur l'interface graphique, la totalité des 81 cases n'est pas jouable (uniquement 48 cercles). Il est donc très utile de s'aider des fonctions utilitaires d'application des règles citées plus tôt.

Pour obtenir un exemple minimal d'un joueur fonctionnel (mais mauvais), vous pouvez regarder les agents `random_player.py` et `greedy_player.py`.

⚠**Dès lors que l'agent soumet une action, il doit s'assurer que celle-ci est bien valide (i.e., elle est autorisée par les règles du jeu). Si une action non autorisée est soumise, le joueur perd automatiquement la partie au profit de son adversaire. Un vérificateur de validité d'actions est disponible pour vous aider et éviter de lever une exception de type InvalidAction.**

⚠Afin de garantir une certaine équité entre les équipes, l'utilisation de **GPU** et du **multi-threading** est interdite et entraînera une disqualification. Du aux limites physiques du serveur accueillant la compétition, votre agent ne doit pas allouer plus de **4Gb de RAM** (largement suffisant).

7 Critères d'évaluation

L'évaluation portera sur la qualité du rapport (50% de la note) et du code (50% de la note) fournis, ainsi que des résultats de votre agent face à plusieurs agents d'une difficulté croissante. **Le classement de votre agent durant le tournoi n'aura aucune influence sur l'évaluation mais sera l'objet de point bonus sur la note finale.** Dès lors, il est tout a fait possible d'être éliminé rapidement lors du concours et d'obtenir une bonne évaluation, ou même d'être premier au concours et obtenir une mauvaise évaluation (p.e., à cause d'un rapport négligé). Les principaux critères d'évaluation sont les suivants :

1. Qualité générale et soin apporté au rapport.
2. Qualité de l'explication de la solution mise en œuvre.
3. Qualité du code (présence de commentaire, structure générale, élégance du code).
4. Performance face à différents agents d'évaluation (agents aléatoires, agents gloutons, d'autres agents intelligents, etc.)

⚠**La note minimale sera attribuée à quiconque ayant copier/coller une solution ou proposant un code contenant une erreur à la compilation. Prenez bien soin de vérifier le contenu de votre rendu. Notez que vu la liberté laissée à l'implémentation, il est très aisé de détecter les cas de plagiats.**

⚠**L'évaluation sera effectué sur une machine Linux raw, veuillez donc à bien spécifier (si nécessaire) les bibliothèques externes dans un fichier** `requirements.txt` **afin de pouvoir les installer avec la commande** `pip install -r requirements.txt`**.**

⚠**Le jury préférera une solution simple accompagnée d'une riche explication logique à une solution trop complexe mettant en œuvre des concepts mal-compris de l'élève.**

8 Quelques conseils pour vous aider

Conseils généraux

— Le projet a été conçu pour que le travail puisse bien être réparti tout au long de la session. Profitez-en et commencez dès maintenant! Essayez d'enrichir votre rapport au fur et à mesure de l'avancée de votre projet. Cela vous fera gagner du temps lors du rendu et vous permettra de garder le fil de vos recherches.
— Assurez-vous de bien comprendre le fonctionnement théorique des méthodes que vous souhaitez utiliser avant de les implémenter. Également, il est plus que conseillé de réaliser d'abord les agents du Morpion et de Puissance 4 du deuxième laboratoire.
— Prenez le temps de comprendre la structure du code de `avalam.py` en y mettant des "*print statements*" et des commentaires. Si cela peut vous sembler laborieux, vous gagnerez beaucoup de temps par la suite lors de vos éventuels *debugging* (qui arriveront quasi certainement :-)).
— Commencez par réaliser des agents simples, puis rajoutez des fonctionnalités petit à petit.
— Trouvez le bon compromis entre complexité et facilité d'implémentation lors de la conception de votre agent. Le temps de calcul pour chaque action mis à disposition de votre agent étant limité, il est également à considérer.
— Travaillez efficacement en groupe, et répartissez vous bien les tâches.
— Tirez le meilleur parti des séances de TP afin de demander des conseils aux chargés de laboratoire.
— Profitez de ce travail de groupe pour vous initier à l'outil de développement collaboratif Git ainsi qu'aux bonnes pratiques de l'outil.
— Un agent efficace n'est pas forcément celui utilisant les méthodes les plus compliquées. Un algorithme simple mais bien implémenté, robuste et efficace a toute ses chances.

⚠Bien que ludique et motivant, le concours n'est pas utilisé pour l'évaluation de votre projet. De plus, il est possible d'y passer énormément de temps. Dès lors, faites attention à ne pas négliger les autres parties du projet ainsi que les échéances de vos autres cours.

Suggestion de planning pour votre travail (purement indicatif)

1. *Dès maintenant* : Familiarisez vous avec les règles du jeu, jouez avec, et imaginez vos propres stratégies. Pour cela, essayez de reconnaître quels sont les états du plateau qui vous sont favorables, et ceux qui vous sont défavorables.
2. *Dès le 11 septembre* : Familiarisez vous avec le code du projet et comprenez les différents fichiers qui vous sont donnés. Assurez-vous que vous êtes capable de lancer les agents aléatoires.
3. *Dès que le module 2 a été vu* : Commencez à concevoir (sur papier) l'architecture de votre agent ainsi qu'une heuristique. Afin de récolter plusieurs idées (idéalement complémentaires), il est intéressant de faire cette étape individuellement puis de mettre en commun avec votre partenaire de groupe.
4. *Dès que le laboratoire 2 a été vu* : Commencez l'implémentation de votre agent. Testez le souvent afin de détecter les bugs au plus vite.
5. *Après la relâche* : Améliorez vos heuristiques d'évaluation. C'est souvent le point central de la qualité d'un agent.
6. *Dès le 23 octobre* : Convergez vers l'architecture finale de votre agent et implémentez de façon efficace et robuste vos différentes idées. N'hésitez pas à le faire jouer contre vous et contre des agents d'autres groupes. Prenez note des statistiques de performance afin d'alimenter votre rapport.
7. *Dès le 6 novembre* : Finalisez votre agent et assurez vous qu'il ne retourne pas d'erreurs.
8. *Dès le 20 novembre* : Nettoyez votre code et, surtout finalisez votre rapport. N'oubliez pas que le rapport compte pour 50% de la note du projet. Il est donc important d'y apporter beaucoup de soin.

⚠Surtout ne sous-estimez pas le temps nécessaire pour débugger votre agent (une partie peut durer jusqu'à 30 minutes!) ainsi que le temps de rédaction du rapport.

Suggestion d'architecture pour votre agent

— *Niveau standard* : Procédez avec un algorithme de type *minimax* avec une bonne heuristique et enrichi de mécanismes supplémentaires. Ce type d'algorithme a gagné le concours en automne 2021. C'est un algorithme de ce type que nous vous recommandons pour le projet.
— *Niveau avancé* : Procédez avec un algorithme de type *Monte-Carlo Tree Search*. Il peut-être intéressant de chercher des ressources complémentaires pour vous aider, par exemple des articles détaillant des architectures pour d'autres jeux. Plusieurs algorithmes de ce type étaient dans dans le top 10 des agents en automne 2021.
— *Niveau expert* : Procédez avec un algorithme de type *apprentissage automatique*. Aucun algorithme de ce type n'a bien performé en automne 2021. :-) Si vous décidez de partir dans cette direction, il est plus que conseillé de lire de la documentation supplémentaire à ce sujet.

⚠Partir sur une architecture plus compliquée n'a aucun impact sur la note que vous recevrez et va rendre votre projet plus compliqué. Les niveaux avancés sont plus à destination des étudiants voulant expérimenter des algorithmes de leur choix.

Bon travail, bonne chance, et surtout, amusez vous!