Projet - Agent intelligent pour le jeu Divercité

Agent pour le concours à remettre le 19 novembre 2024 sur Moodle (avant minuit) pour tous les groupes.

Code et Agent à remettre le 7 décembre 2024 sur Moodle (avant minuit) pour tous les groupes.

Rapport à remettre le 7 décembre 2024 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 se trouver à la **racine du fichier de rendu** compressé 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 de plateau *Divercité*. *Divercité* est un jeu stratégique où deux joueurs s'affrontent pour accumuler le plus de points en plaçant leurs pièces (appelées *cités*) sur un plateau, entourées de ressources de différentes couleurs. Le but du jeu est de créer des configurations maximisant les points en fonction de plusieurs règles.

Préparation du jeu Les joueurs se placent face à face, l'un jouant avec les pièces noires et l'autre avec les pièces blanches. Chaque joueur reçoit ses cités ainsi que trois jetons de ressources de chaque couleur. Une représentation du tableau est présentée à la Figure 1.

Déroulement d'une partie À chaque tour, un joueur doit placer une de ses pièces sur une case inoccupée du plateau. Il a deux options :

1. Poser une cité sur une case de pavés.
2. Poser une ressource sur une case en pointillés.

La pièce une fois posée ne peut plus être déplacée. Les joueurs alternent ainsi jusqu'à ce que toutes les pièces soient posées sur le plateau.

Lorsque toutes les pièces ont été jouées, il reste une dernière case de ressource inoccupée et la partie est terminée.

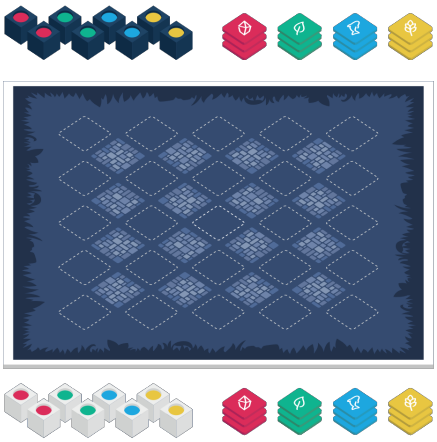

FIGURE 1 – Plateau de départ

Comptage des points Une fois toutes les pièces posées, les joueurs regardent chacune de leurs cités en comptabilisant les points en fonction des ressources qui les entourent. Il existe deux cas de figure pour le décompte des points :

1. **Divercité** : Si une cité est entourée de quatre ressources de couleurs différentes, elle rapporte 5 points à son propriétaire.
2. **Ressources identiques** : Si la cité n'est pas entourée de quatre ressources différentes, elle rapporte 1 point par ressource adjacente de la même couleur que la cité, pouvant rapporter entre 1 et 4 points.

Les ressources placées sur le plateau sont neutres et profitent à toutes les cités environnantes, quelle que soit leur couleur. Par exemple, un joueur pourrait être tenté de placer une ressource rouge près de ses cités rouges pour marquer des points, mais doit aussi considérer que cette action pourrait avantager l'adversaire.

En cas d'égalité au score, les deux joueurs seront départagés selon leur nombre de Divercité, si ce dernier est aussi égal, nous décomptons le nombre de cités entourées par quatre ressources de couleur similaire puis celles avec trois ressources et ainsi de suite jusqu'à désigner un gagnant. Si une égalité parfaite survient, le premier joueur remporte la partie car le deuxième est considéré comme ayant un avantage.

⚠**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 Discord ou lors des séances de laboratoire.

2 Règles du projet

Vous êtes libres 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 3.11.x ou supérieur**) 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 à fait permis à un agent de consacrer 10 minutes pour effectuer son premier coup, et d'utiliser les 5 minutes restantes pour la réalisation de tous les autres coups. Afin de garantir une certaine équité entre les équipes, l'utilisation de **GPU** et du **multithreading** est interdite et entrainera une disqualification. Dû aux limites physiques du serveur, votre agent ne doit pas allouer plus de **4 Gb de RAM** (largement suffisant) et **100 Mb de mémoire disque** après la compression. En cas d'action invalide, d'utilisation de mémoire trop importante ou de ressources non autorisées, **votre agent perdra inévitablement la partie**. Veuillez donc vous assurer du comportement de votre agent avant toute soumission sur Abyss ou pour le rendu du projet.

3 Abyss

Abyss est une plateforme conçue par l'équipe encadrante pour vous offrir un terrain de test où vous pouvez soumettre et évaluer en continu la qualité de vos agents. Chaque étudiant se voit attribuer un compte personnel avec un identifiant et un mot de passe, ce qui lui permet d'accéder à son profil pour y téléverser des agents. Le système fonctionne en continu, **permettant aux agents de s'affronter entre eux de manière automatique**, et ainsi, de consulter leurs performances. Bien que soumettre vos agents sur la plateforme soit facultatif, nous vous conseillons fortement de le faire. Cela vous permettra de vous assurer que votre agent respecte toutes les consignes de soumission et d'utiliser les résultats de vos parties dans votre rapport de projet. Le système qui mettra en place vos affrontements roule sur un processeur **AMD Ryzen 7 3800X**, les restrictions d'équité décrites dans la section précédente s'appliquent également. Le site est disponible ici et les fonctionnalités principales sont reprises ci-dessous. ⚠**Abyss est encore à un stade de développement récent, si un problème survient et qu'une maintenance est nécessaire, les agents téléversés ne seront plus en train de jouer**. Il est toujours possible, et fortement encouragé, de continuer de votre côté, de faire jouer vos agents, voire d'organiser d'autres affrontements avec les autres groupes.

Mise en ligne d'un agent : Une fois connecté, vous aurez la possibilité de mettre en ligne votre agent depuis la page principale d'Abyss. Le format requis de mise en ligne est celui décrit à la section 8 du sujet. Il est primordial de suivre ce format, sans quoi votre agent ne pourra pas être lancé par le système.

Profil d'utilisateur : Votre profil utilisateur donne accès aux principales fonctionnalités d'interaction avec le système Abyss. Il est partagé avec votre binôme pour mettre en commun vos agents mis en ligne et les parties que ces derniers ont jouées. Après la mise en ligne d'un agent, ce dernier est mis en attente d'une validation avant de pouvoir jouer. Une fois la validation effectuée, si tout est en ordre, votre agent passera en état inactif, vous pourrez ainsi le changer afin de le faire entrer dans la compétition contre ses adversaires. Pour maintenir un équilibre et une compétition saine, chaque groupe d'étudiants est limité à **deux agents actifs en même temps**. Vous aurez également accès à une liste des parties effectuées par vos agents. Chaque étudiant a la possibilité de télécharger les données de ses anciennes parties au format **JSON**. Cette fonctionnalité est particulièrement utile pour les groupes qui souhaitent réanalyser leurs parties ou récolter des données pour la conception de leurs agents.

Classement Elo et le fonctionnement des ligues : Le classement Elo de chaque agent est un élément important du site Abyss. Ce classement détermine la ligue à laquelle un groupe appartient. Le classement Elo du groupe est défini par l'Elo maximum parmi les agents valides du groupe.

Visualisation des parties : Le site propose également une vue détaillée des parties du système, vous permettant de consulter les résultats de toutes les rencontres précédentes, mais aussi de voir quelles sont les parties en cours, ainsi que celles programmées et annulées. Les parties sont réparties dans les quatre sections selon leur statut. De plus, grâce à une interface graphique, vous pouvez revoir toutes les parties terminées afin de faciliter l'analyse des performances.

Page palmarès et statistiques : Une page dédiée au palmarès permet de visualiser les meilleures statistiques des groupes sur la plateforme. Vous pouvez consulter les groupes ayant obtenu les meilleures performances sur de nombreux critères, dans le but de favoriser une compétition saine et stimulante entre les élèves.

⚠**Si vous avez des questions concernant Abyss, n'hésitez pas à demander à vos chargés de laboratoire. De plus, il s'agit de la première année où Abyss est déployé, si vous rencontrez un bogue ou une anomalie, utilisez le canal approprié du serveur Discord pour nous la signaler.** Nous vous invitons à partager avec nous vos commentaires et suggestions pour améliorer ce système sur notre serveur Discord ou via les évaluations sur Moodle.

4 Tournoi final

Vous aurez également l'opportunité de participer à un tournoi Challonge. Le tournoi aura lieu vers la fin de la session avec la version finale de votre agent, et est facultatif. 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 match est composé de 3 parties. À l'issue, les joueurs sont classés sur leur nombre de victoires (et le cumul de leur score en cas d'égalité). Seulement 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 trois manches d'une partie, la couleur blanche/noire sera attribuée une fois par joueur. La répartition de la dernière manche est aléatoire. Vous pourrez suivre les résultats de vos agents en direct sur le site du tournoi. Le tournoi aura lieu quelques jours après la remise de la version compétitive de votre agent. Le lien du tournoi ainsi que sa configuration finale vous seront fournis ultérieurement. Les matchs seront lancés sur la même machine que celle qui héberge Abyss, les spécifications sont donc les mêmes que celles décrites dans la section 3

5 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** et rendu au **format PDF**. Étant donné que vous êtes libres 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 du fonctionnement de votre agent. 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 conception faits (fonctionnement de l'heuristique utilisée, stratégie choisie, spécificités propres de votre agent, gestion des 15 minutes allouées, etc.).
4. **Résultats et évolution de l'agent** : Reportez les performances de votre agent en le testant contre les implémentations qui vous sont fournies et vos différentes versions. Spécifiez un ou plusieurs critères de mesure permettant de chiffrer vos résultats et mettez-les en avant grâce à des graphiques, des tableaux ou toutes autres techniques de visualisation pertinentes selon vous. On vous suggère également d'utiliser vos résultats sur Abyss pour appuyer vos analyses.
5. **Discussion** : Discutez des avantages et limites de votre agent final. Évoquez différentes pistes d'améliorations envisageables selon vous.
6. **Références** (si applicable).
7. **Annexes** (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 bibliographiques sont autorisées, ainsi que pour la page de garde). Si vos graphiques et tableaux vous font dépasser la limite, vous pouvez utiliser la section **Annexes**. Vous êtes libre de structurer le rapport comme vous le souhaitez du moment que tous les éléments mentionnés précédemment sont inclus.

6 Ressources fournies

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

⚠**Si vous n'êtes pas familier avec la programmation orientée objet en python, vous pouvez consulter cette fiche explicative.**

Les fichiers fournis sont :

— `board_divercite.py` : contient le code pour représenter le plateau de jeu. **Vous n'avez pas à comprendre ce fichier, ni à le modifier.**
— `main_divercite.py` : contient l'ensemble des fonctions permettant de lancer une partie, de l'enregistrer et de pouvoir la visioner via une interface graphique. Les détails concernant les commandes pour interagir avec l'environnement (notamment lancer une partie) vous sont fournis plus bas. **Vous n'avez pas à modifier ce fichier.**
— `master_divercite.py` : contient le code implémentant la mécanique du jeu et détermine le gagnant. **Vous n'avez pas à modifier ce fichier.**
— `player_divercite.py` : contient le code implémentant la classe `PlayerDivercite` dont votre agent va hériter. **Vous n'avez pas à modifier ce fichier.**

— `game_sate_divercite.py` : implémente toute la logique du jeu Divercité à l'aide de différentes 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 de ce fichier pour vous aider à mieux l'appréhender :

- `class GameStateDivercite` : contient toute l'information relative à un état du jeu. Cette classe contient :

1. Un constructeur :

```
def __init__(self, elf, scores: Dict, next_player: Player, players: List[
Player], rep: BoardDivercite, step: int):
    """Initialize a state of the game.
    Arguments:
    scores -- a dictionnary with the score of each player
    next_player -- the next player to play
    players -- a list with the players
    rep -- the current board of the game
    step -- the step to which the game state belongs
    """
    ...
```

2. Diverses fonctions utilitaires :

```
def get_step(self) -> int
def is_done(self) -> bool
def generate_possible_actions(self) -> Set[Action]
def compute_scores(self, id_add: int) -> Dict[int, float]
def get_neighbours(self, i: int, j: int) -> Dict[str,Tuple[int, int]]
```

— `my_player.py` : votre agent. **Vous devez modifier ce fichier.** Si vous souhaitez modifier des classes provenant des fichiers du jeu, il est recommandé d'utiliser l'héritage de Python dans ce fichier. N'oubliez pas de convertir les objets lorsqu'il vous sont fournis (e.g. `GameStateDivercite` lors d'un appel de `compute_action`)
— `random_player_divercite.py` : un agent aléatoire si vous voulez gagner une partie facilement. ;)
— `greedy_player_divercite.py` : un agent glouton qui devrait être facilement perdre face à vos agent ayant une plus grande profondeur de recherche.

Le projet se base sur le package open source `seahorse`. Vous pouvez le soutenir en déposant une étoile sur sa page github. Pour lancer une partie il faut donc au préalable installer `seahorse` à l'aide de la commande suivante :

```
$ pip install seahorse colorama
```

Ensuite, plusieurs modes d'exécution sont disponibles via la présence d'arguments. Par exemple, `-r` permet d'enregistrer une partie dans un fichier json. Pour obtenir la description de tous les arguments, exécutez la commande suivante :

```
$ python main_divercite.py -h
```

Pour lancer une partie en local avec GUI, il faut par exemple lancer la commande suivante :

```
$ python main_divercite.py -t local random_player_divercite.py random_player_divercite.py
```

La commande suivante lance une partie en local entre l'agent aléatoire et l'agent glouton, les logs sont

enregistrés dans une json, la GUI n'est pas ouverte.

```
$ python main_divercite.py -t local random_player_divercite.py greedy_player_divercite.py -r -g
```

Si l'on souhaite organiser une partie contre un agent d'un autre groupe, il faut lancer la commande suivante pour héberger le match :

```
$ python main_divercite.py -t host_game -a <ip_address> random_player_divercite.py
```

L'équipe que l'on souhaite affronter devra quant à elle lancer :

```
$ python main_divercite.py -t connect -a <ip_address> random_player_divercite.py
```

Il faudra alors remplacer <ip_address> par l'adresse IP de l'ordinateur qui héberge la partie. Pour obtenir cette dernière, exécuter la commande `ipconfig` (Windows) ou `ifconfig` (Mac, Linux) dans un terminal.

Afin de se familiariser au jeu dans un premier temps, il est possible de jouer manuellement l'un contre l'autre avec la commande suivante :

```
$ python main_divercite.py -t human_vs_human
```

Finalement, pour pouvoir étudier le comportement de votre agent, il peut être intéressant de jouer manuellement contre ce dernier avec :

```
$ python main_divercite.py -t human_vs_computer random_player_divercite.py
```

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

⚠**Il est préférable de ne pas utiliser le navigateur `safari` pour afficher l'interface graphique.**

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

7 Environnement virtuel

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 la gestion des environnements virtuels est un atout important dans votre future carrière.

⚠**L'ensemble des commandes suivantes sont à réaliser avec la console dans le répertoire `Divercite`.**

Création d'un environnement virtuel

```
$ python -m venv venv
```

Un dossier `venv` est normalement créé.

Activation de l'environnement virtuel

Pour Windows :

```
$ . .\venv\Scripts\Activate.ps1
```

Pour Linux et MacOS :

```
$ source venv/bin/activate
```

*La console devrait afficher **(venv)** comme préfixe.*

Installation d'une dépendance

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

```
(venv) pip install seahorse colorama
```

*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.*

Lancement d'un agent

```
(venv) python main_divercite.py -t local random_player_divercite.py random_player_divercite.py
```

L'interface graphique devrait alors apparaitre.

Requirements

Le projet étant assez libre, il est possible (non essentiel) d'ajouter des librairies telles 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 virtuel et pip permettent d'exporter simplement ces dépendances via la commande suivante.

```
(venv) 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.

```
$ python -m venv tmp
$ . .\tmp\Scripts\Activate.ps1     #(Windows)
$ source tmp/bin/activate          #(MacOS et Linux)
(tmp) 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.*

8 Code à produire

Vous devez remettre un fichier `my_player.py` avec votre code. Pour ce faire, vous être entièrement libres d'utiliser la méthode que vous voulez. Si vous utilisez 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 7.

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

Input : `self:PlayerDivercite, current_state:GameState`.

Toute l'information de la partie (placement des cités et ressources, scores de chaque équipe, etc.) est contenue dans l'objet `current_state`. 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 dans l'état de son avancement.

Output : `Action`

Une action est composée de deux objets GameState, définis comme suit :

— `current_game_state` : état actuel du jeu.
— `next_game_state` : état du jeu après avoir effectué le mouvement proposé.

Pour obtenir un exemple minimal d'un joueur fonctionnel (mais mauvais), vous pouvez regarder l'agent `random_player_divercite.py` ainsi que `greedy_player_divercite.py`.

⚠**Dès lors que l'agent soumet une action, il doit s'assurer que celle-ci est bien valide (i.e., elle est présente dans l'ensemble retourné par** `current_state.get_possible_actions()`**). 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 utilisé dans** `seahorse` **et lève une exception de type** `ActionNotPermittedError`**.**

8.1 Pour Abyss et le concours

Si vous souhaitez participer et soumettre votre agent sur Abyss ou pour le concours, le rendu est légèrement différent. Il devra contenir à minima votre fichier `my_player.py` contenant votre implémentation de la classe `MyPlayer` héritant de `PlayerDivercite` et les librairies utilisées dans un fichier `requirements.txt`.

— Pour les librairies, assurez-vous de **ne fournir que les librairies nécessaires à l'exécution** de votre agent. Les librairies d'entrainement ou non utilisées dans l'agent final ne doivent pas être fournies.
— Si vous souhaitez ajouter des **fichiers supplémentaires** : mettez les dans un dossier nommé au format *src_matricule1_matricule2*. Assurez-vous que les liens (les `import`) entre les fichiers sont corrects et que votre agent fonctionne avec cette architecture. Ce format est privilégié afin de ne pas écraser les fichiers de deux agents différents lors d'un match.
— Pour tout fichiers supplémentaires, veuillez également utiliser des noms de fichiers sans espaces, par exemple, préférez utiliser `my_function.py` au lieu de `my function.py`. Les accents et les caractères spéciaux autres que le tiret bas sont également à éviter.

Lors du concours, il est nécessaire de nommer le zip de rendu du code au format suivant : *NomEquipe_matricule1_matricule2.zip*.

Un point important à prendre en compte lors de la soumission d'un agent sur le système Abyss est que les fichiers JSON peuvent contenir des informations dessus, il s'agit des attributs de votre classe héritant de `PlayerDivercite` et des valeurs qu'ils possèdent à la fin de votre tour de jeu. Si vous ne souhaitez pas que ces attributs apparaissent, il vous suffit de les **renommer en les préfixant d'un tiret bas**. Vous pourrez retrouver un exemple ci-dessous :

```python
class MyPlayer(PlayerDivercite):
    def __init__(self, piece_type: str, name: str = "bob",*args) -> None:
        super().__init__(piece_type,name,*args)
        self.count_pieces = 0 # Cet attribut apparaitra dans le JSON
        self._nodes = [] # Cet attribut n'apparaitra pas dans le JSON
    def compute_action(self, current_state: GameState, **kwargs) -> Action:
        possible_actions = current_state.get_possible_light_actions()
        # Variable non contenu comme attribut, elle n'apparaitra pas dans le JSON
        ...
```

Nous vous recommandons d'utiliser ce motif de noms afin de filtrer des enregistrements de vos parties les attributs que vous ne voudriez pas voir apparaitre, n'oubliez pas que vos adversaires ont aussi accès

à ce fichier ;) Mais il est également recommandé de filtrer des attributs pouvant référencer des structures de données pouvant grandir durant votre tour de jeu et/ou contenir beaucoup de données. Par exemple, une liste de nœuds d'un MCTS ou une table de transpositions devraient être filtrées s'ils prennent trop de valeurs. La raison de cette recommandation est la limitation en espace mémoire et capacité de calcul du serveur d'Abyss mais également le risque de crash lors de l'écriture qui entrainerait la perte du fichier.

9 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 our sur Abyss 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 note maximale, 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. Présence de toutes les informations demandées (voir Section 5).
3. Qualité de l'explication de la solution mise en œuvre.
4. Qualité du code (présence de commentaires, structure générale, élégance du code).
5. 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 effectuée 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.**

10 Quelques conseils pour vous aider

Conseils généraux

— Le projet est conçu pour que le travail puisse 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 votre avancée. 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 `game_state_divercite.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 laboratoire afin de demander des conseils.
— 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. Cette analyse va vous être très utile pour concevoir votre agent.
2. *Dès le 10 septembre* : Familiarisez vous avec le code du projet et comprenez les différents fichiers qui vous sont donnés. Assurez-vous que vous êtes capables d'exécuter le code fourni et l'agent aléatoire.
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 en le mettant en compétition sur la plateforme Abyss. Améliorez au fur et à mesure vos heuristiques d'évaluation. C'est souvent le point central de la qualité d'un agent.
5. *Dès le 29 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.
6. *Dès le 12 novembre* : Finalisez votre agent et assurez vous qu'il ne retourne pas d'erreurs.
7. *Dès le 26 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 En fonction de vos connaissances préalables et de vos intérêts respectifs, nous vous proposons plusieurs choix d'architecture pour la conception de vos 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, 2022, et 2023. 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 le top 10 des agents en automne 2021 et 2022.
— *Niveau expert* : Procédez avec un algorithme de type *apprentissage automatique*. Aucun algorithme de ce type n'a bien performé aux éditions précédentes et demande généralement beaucoup de travail. 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!