Projet - Agent intelligent pour le jeu Abalone

Agent pour le concours à remettre le 19 novembre 2023 sur Moodle (avant minuit) pour tous les groupes.

Code et Agent à remettre le 7 décembre 2023 sur Moodle (avant minuit) pour tous les groupes.

Rapport à remettre le 7 décembre 2023 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 Abalone. Abalone est un jeu de stratégie au tour par tour, en un contre un, dont le principe est simple : expulser un maximum de billes de l'adversaire hors du plateau.

Ce plateau sur lequel le jeu se déroule est percé de 61 cercles supportant les billes. Chaque joueur possède 14 billes disposées d'une manière bien définie au départ. Le joueur blanc commence. À chaque tour, chaque joueur doit effectuer un et un seul déplacement dans n'importe quelle direction (horizontale, verticale, diagonale) sur une case adjacente. Ce mouvement revient à déplacer une ou plusieurs de ses billes (au maximum 3) dans le même sens.

Ces mouvements doivent cependant respecter quelques règles :

- On doit toujours déplacer des billes adjacentes.
- Si l'on déplace plusieurs billes, cela doit uniquement se faire dans l'une des deux directions parallèles à la ligne formée par ces dernières. Les mouvements en flèche (Figure 1) ne sont donc **pas autorisés** dans le cadre du projet!
- On peut pousser les billes adverses seulement si l'on est en supériorité numérique stricte et qu'il y a un espace vide (ou le bord du plateau) derrière la ligne formée par ces dernières.

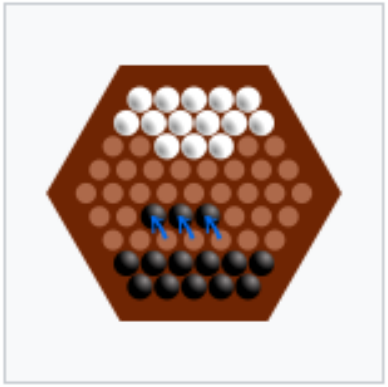

FIGURE 1 – Mouvements en flèche **interdits**

Plusieurs dispositions sont possibles pour commencer une partie. Lors de l'évaluation et du concours, trois de ces différentes dispositions seront testées.

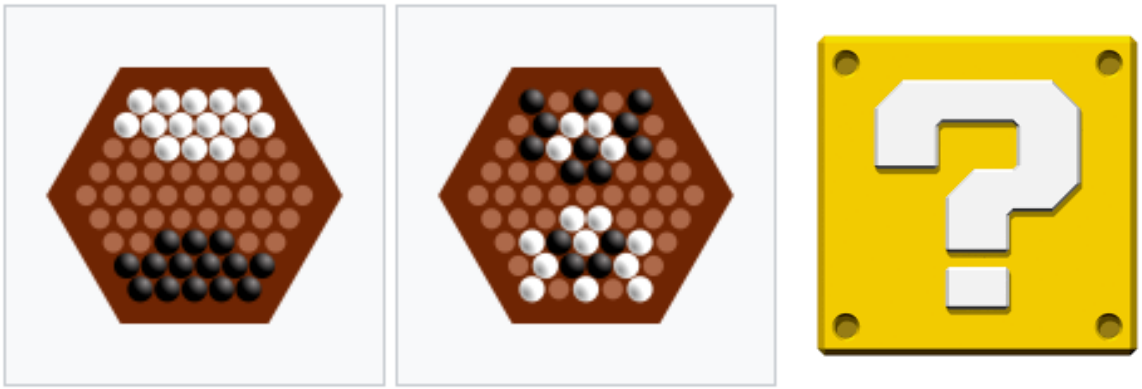

FIGURE 2 – Positions de départs possibles

La fin de partie est déclarée lorsque six billes d'une même équipe sont expulsées hors du plateau ou lorsque les 2 joueurs ont effectué 50 tours en tout (25 coups chacun). Le gagnant est décidé en comptant le nombre de billes perdues par chaque joueur. La majorité l'emporte. En cas d'égalité, c'est l'équipe ayant la plus petite somme des distances entre ses billes et le centre du plateau qui gagne. 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 Abalone. 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**), la possibilité de faire jouer votre agent (votre code compile et re-

tourne 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 tous les agents, nous les ferons jouer les uns 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. Les deux premiers matchs se feront avec la première configuration de départ. Les deux suivants avec la deuxième configuration. Le cinquième match se fera avec une autre configuration de départ qui vous est inconnue. 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/noire 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. 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.

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 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 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 version Python du jeu Abalone 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 :

— `board_abalone.py` : contient le code pour représenter le plateau de jeu. **Vous n'avez pas à comprendre ce fichier, ni à le modifier.**
— `main_abalone.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_abalone.py` : contient le code implémentant la mécanique du jeu et détermine le gagnant. **Vous n'avez pas à modifier ce fichier.**
— `player_abalone.py` : contient le code implémentant la classe PlayerAbalone dont votre agent va hériter. **Vous n'avez pas à modifier ce fichier.**
— `game_sate_abalone.py` : implémente toute la logique du jeu Abalone 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 GameStateAbalone` : contient toute l'information relative à un état du jeu. Cette classe contient :

1. Un constructeur :

```python
def __init__(self, elf, scores: Dict, next_player: Player, players: List[
Player], rep: BoardAbalone, 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 :

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

Le projet se base sur le package open source `seahorse`. Il est d'ailleurs fortement recommandé d'aller déposer 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==1.0.0
```

Ensuite, plusieurs modes d'exécution sont disponibles via la présence de différents arguments. Par exemple, `-c` permet de changer la configuration initiale du jeu et `-r` permet d'enregistrer une partie dans un fichier json. Pour obtenir la decriptions de tous les arguments, exécutez la commande suivante :

```
$ python main_abalone.py -h
```

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

```
$ python main_abalone.py -t local random_player_abalone.py random_player_abalone.py
```

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_abalone.py -t host_game -a <ip_address> random_player_abalone.py
```

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

```
$ python main_abalone.py -t connect -a <ip_address> random_player_abalone.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_abalone.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_abalone.py -t human_vs_computer random_player_abalone.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.**

5 Environnement virtuel

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 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** `abalone`**.**

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==1.0.0
```

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_abalone.py -t local random_player_abalone.py random_player_abalone.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.*

6 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 5.

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

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

Toute l'information de la partie (placement des billes, 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_abalone.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 utillisé dans `seahorse` et lève une exception de type ActionNotPermittedError.**

⚠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 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 3).
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.

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 `game_state_abalone.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. 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 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 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.
7. *Dès le 12 novembre* : Finalisez votre agent et assurez vous qu'il ne retourne pas d'erreurs.
8. *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

— *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 2021et 2022. 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 en automne 2021 et 2022. :-) 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!