Le Fantôme de l'Opéra

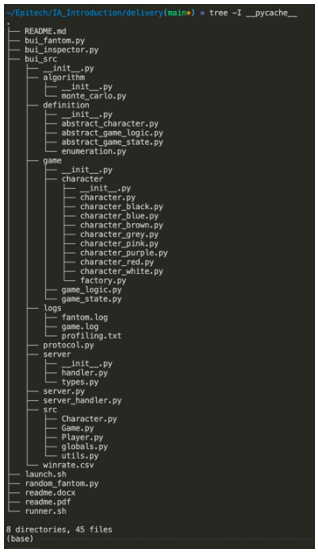
Introduction à l'Intelligence artificielle, par Louis-Albert Bui, Thibaud Bonnefoy et Bastien Roumanteau

Préambule

Nous avons réalisé un programme d'intelligence artificielle pour le jeu de plateau '*Le fantôme de l'opéra'*. Ce document décrit la réalisation et le fonctionnement des différentes parties qui composent ce programme.

Présentation du programme

Les différentes parties du programme seront présentées dans l'ordre chronologique de leur réalisation. Il a fallu d'abord étudier la logique de jeu pour la réaliser afin qu'elle puisse simuler des parties à partir d'une configuration C du jeu, puis nous nous sommes intéressés à la méthode de Monte Carlo. Afin de compléter et d'optimiser ce dernier, nous avons décidé d'implémenter un apprentissage au cours du temps à notre programme. Pour finir, nous avons réalisé un script permettant de relancer un nombre X de parties, et de déterminer notre nombre de victoire.

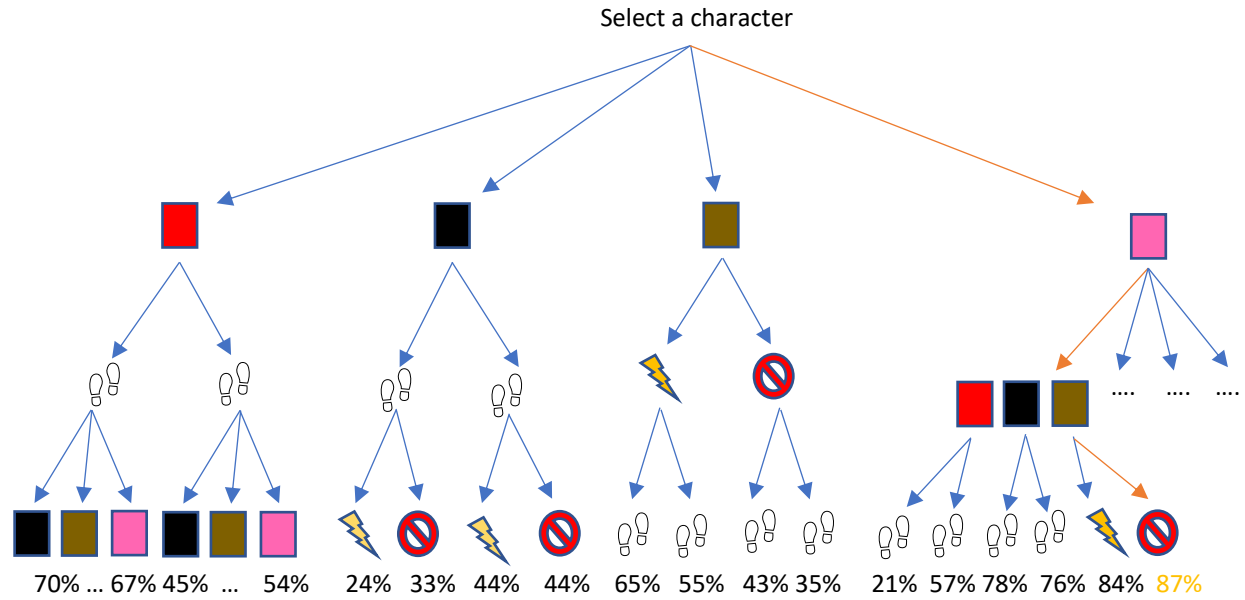


Logique de jeu

Nous avons d'abord étudié le jeu et toutes ses spécificités. Le but était ici de comprendre toutes les subtilités de ce dernier, afin de pouvoir le refaire pour pouvoir simuler des parties à partir d'une configuration donnée. La première implémentation du jeu de l'intervenant a été d'une grande utilité puisque nous avons pu nous en inspirer afin de pouvoir créer notre propre jeu, plus modulable. Il est composé de trois parties, premièrement la logique de jeu (game_logic.py) qui ne contient que les règles et les différents effets possibles, qui sert qu'à régir le jeu à partir d'une configuration. Vient ensuite l'état de jeu (game_state.py) qui contient lui la configuration actuelle du jeu, c'est-à-dire tous les paramètres relatifs au jeu, comme la position de la carlotta, le passage bloqué, la liste des différents personnages, ect.. La dernière partie concerne les différents personnages (character) qui contiennent les informations propres à ces derniers comme leur position ou s'ils sont suspects, ainsi que des méthodes qui implémentent leurs différents pouvoirs. La réalisation de cette partie était primordiale pour pouvoir commencer à travailler sur la méthode de Monte Carlo.

Monte Carlo

Nous avons choisi d'implémenter un Monte Carlo puisque cette méthode a piqué notre curiosité lors du cours, et qu'aucun membre du groupe en avait déjà réalisé un. Le fonctionnement de notre Monte Carlo est plutôt simple : avec un état de jeu (game_state), cette classe va réaliser un arbre de profondeur P (depth). Une fois toutes les possibilités trouvées, nous simulons un nombre X (number_simulated_games) de parties à choix afin d'avoir un pourcentage de victoires pour chaque branche. Afin d'optimiser la réalisation de ces parties, nous avons décidé de faire appel à des threads afin de paralléliser les parties. Mais face à la possibilité d'un grand nombre de branches, nous avons opté pour un apprentissage afin de ne pas rejouer les parties déjà simulées.



Carte
Déplacement
Activation du pouvoir
Non-activation du pouvoir
Chemin choisi

Apprentissage

Comme expliqué précédemment, cette partie a pour but d'alléger le programme en gardant le pourcentage de victoire d'une configuration donnée grâce à Monte Carlo. Le fonctionnement est très simple, lorsque nous arrivons au bout d'une branche, nous avons un état de jeu noté Sy et nous devons simuler X parties pour avoir un pourcentage de victoires. Mais avant de les simuler, notre programme va d'abord regarder dans le fichier winrate.csv à la recherche de cette même configuration Sy. S'il trouve la configuration Sy, alors il garde le pourcentage de victoire qui lui est associé dans ce fichier, et ne simule pas les X parties puisque ça a déjà été fait dans une partie antérieure. S'il ne trouve pas la configuration Sy, il simule alors les X parties, puis écrit dans le fichier la configuration Sy et son pourcentage de victoire afin qu'elle puisse être possiblement utilisée lors d'une partie future.

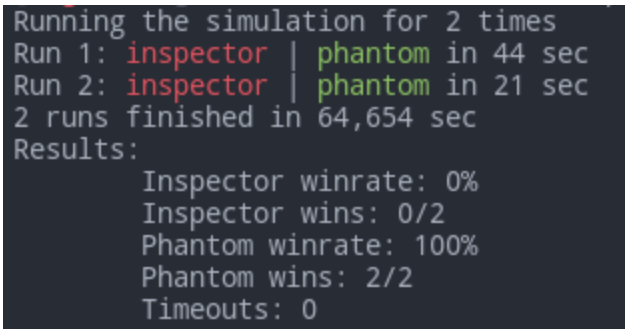
Choix

Après avoir obtenu les différents pourcentages de victoires pour chaque branche, le programme va alors choisir assez logiquement les chemins où le pourcentage de victoire est le plus haut.

Script de test / lancement

Notre script est composé d'un wrapper, launch.sh, permettant de lancer le serveur avec tous les composants. Il permet ainsi de parser le retour du serveur et d'en extraire les informations utiles après la fin de la partie sous la forme d'une ligne CSV contenant gagnant, fantôme, score-finale ou timeout en cas de timeout du serveur

Puis, il contient un wrapper de launch.sh, runner.sh, permettant d'exécuter de lancer un nombre, donné en paramètre, de parties et de récupérer des informations utiles. Nous avons ainsi, pour chaque partie, le si elle est gagnée, perdue, ou timeout ainsi que le temps d'exécution de celle-ci. Après que chaque partie ait été jouée, nous avons le temps total de l'exécution des parties, ainsi que le winrate, le nombre de victoires et le nombre de timeout.



Lancement du programme

Pour lancer un nombre N de parties, il suffit d'utiliser la commande :
'./runner.sh N [bui_inspector.py | random_inspector.py] [bui_fantom.py|random_fantom.py]'.
Pour lancer une partie, il suffit :
'launcher.sh [bui_inspector.py | random_inspector.py] [bui_fantom.py|random_fantom.py]'.
Pour lancer une partie et avoir le détail de l'output du server, il suffit d'ouvrir 3 terminaux, et d'utiliser respectivement :
'python bui_src/server.py'
'python [bui_inspector.py | random_inspector.py]
'python [bui_fantom.py|random_fantom.py]
Le détail de l'output sera sur le terminal du server, ainsi que dans les logs présents dans le dossier bui_src.

Observations

Après avoir réalisé et testé ce projet, nous avons pu faire quelques observations.

Winrate

Le winrate obtenu converge autour de 90% de victoires contre un joueur random.

Limites

Nous rencontrons malheureusement encore trop de time-out, dû au fait que nous n'avons pas encore rencontré toutes les configurations de jeu possible. Après enquête, nous avons remarqué que les time-out n'étaient pas dû au nombre de parties simulés, mais au nombre de branches finales de l'arbre. Assez logiquement, plus il y a de possibilités, plus l'exploration des branches est longue et plus il y a un risque de time-out.

Améliorations & hypothèses

Une fois que le fichier winrate.csv sera rempli de la quasi-totalité des configurations de jeu possible, nous pourrons augmenter la profondeur sans risque de time-out puisque plus aucune partie ne sera simulée, et nous pensons cette augmentation aura pour conséquence une augmentation de notre winrate.