Université de Namur

Intelligence artificielle et programmation symbolique

Rapport du projet Tour de France

2e quadrimestre 2023-2024

Groupe 17

Youlan Collard
Margaux Leleu
Louca Mathieu
Yannis Van Achter

Table des matières

1 Introduction

Ce document représente le rapport du projet *Tour de France* réalisé dans le cadre du cours INFOB317 Intelligence artificielle et programmation symbolique.

Le projet se compose d'une application web et d'un serveur réalisé dans le langage Prolog, qui effectue les calculs de l'intelligence artificielle et du bot du jeu.

Le but de cette application est de proposer une alternative numérique au jeu de plateau du même nom. Le jeu peut alors se jouer à plusieurs en ligne ou en affrontant une ou plusieurs intelligences artificielles.

Ce rapport présente les différents éléments constituant le jeu et se découpe en quatre parties. Dans la première partie, le plateau de jeu et plus globalement l'application en elle-même seront présentés. La seconde partie concerne l'intelligence artificielle réalisée en Prolog. Ensuite, la troisième partie reprend les éléments concernant le bot. Enfin, la dernière partie concerne le lien entre l'application web et le serveur Prolog.

1.1 Règles du jeu

1.1.1 Introduction au jeu

Le jeu Tour de France est une simulation de la célèbre course cycliste du même nom. Il met en compétition quatre équipes de trois coureurs chacune. L'objectif est de remporter le classement général en obtenant le temps total le plus faible pour son équipe.

1.1.2 Déroulement d'une partie

Une partie se déroule sur une étape de plaine, représentée sur le plateau de jeu. Chaque joueur contrôle une équipe, soit humainement soit via un robot doté d'intelligence artificielle. Les coureurs se déplacent à l'aide de cartes secondes dont la valeur va de 1 à 12, pour un paquet de 96 cartes au total.

1.1.3 Règles générales

Les règles générales du jeu ont été détaillées dans un document annexe donné avec les consignes du projet. Les joueurs doivent les comprendre et les respecter pour participer efficacement au jeu. Nous présentons ici un court résumé et les adaptations mises en place dans les consignes du projet.

Pour se déplacer d'une case, un joueur doit dépenser une carte seconde. Un total de 96 secondes est donc nécessaire à un coureur pour arriver au bout du plateau.

Il peut y avoir un maximum d'un coureur par case. Certaines cases sont des cases chance : un coureur qui s'y trouve reçoit alors un bonus ou un malus allant de -3 à +3 secondes et lui permettant de continuer de se déplacer.

Une fois la ligne d'arrivée passée, les coureurs ne peuvent plus se déplacer et reçoivent un bonus en fonction de la case sur laquelle ils se trouvent.

1.1.4 Particularités du projet

Dans le cadre de ce projet, plusieurs adaptations sont apportées aux règles du jeu :

- Le jeu utilise 96 cartes secondes, numérotées de 1 à 12, représentées 8 fois chacune.
- Les joueurs tirent initialement 5 cartes, puis un nouveau lot après épuisement.
- Les joueurs peuvent poser des questions au bot explicateur du jeu, qui fournit des réponses en fonction des règles établies.

1.1.5 Résumé du jeu

Le jeu Tour de France offre une expérience de simulation réaliste de la célèbre course cycliste. Grâce à ses règles claires et à ses adaptations pour le projet, il constitue un défi passionnant pour les joueurs.

1.2 Client et interface utilisateur

Nous sommes partis sur un client web basé sur le moteur de jeu "Boardgame.io". En suivant les instructions de la documentation, nous avons adopté React.JS pour gérer l'affichage du jeu.

Toutes ces considérations sont détaillées dans la section "Plateau".

1.3 Serveur Prolog

Le serveur Prolog s'occupe exclusivement des calculs les plus lourds tels que le prochain coup de l'IA ou le traitement des questions par le bot.

Toutes ces considérations sont détaillées dans les sections "Intelligence artificielle", "Bot explicateur" et "Lien entre les éléments".

2 Plateau

Comme cité plus haut (cf. Introduction), le client web de l'application Tour de France est basé sur les modules Boardgame.io et React.JS. De plus, afin de faciliter le développement, nous avons utilisé le module TypeScript.

2.1 Moteur de jeu

Le choix du moteur de jeu a été fait sur la base de sa popularité afin d'avoir de la documentation et des exemples à disposition pour faciliter le développement. De plus, sa simplicité et robustesse nous ont très vite séduits face à de grands noms comme Unity pour les applications .NET.

Boardgame.io a aussi l'avantage de gérer lui-même, via une configuration préalable, un système multi-joueur. Cela nous a permis de développer une seule partie sans avoir besoin de deux serveurs distincts pour gérer le jeu en réseau et les besoins de calcul de l'intelligences artificielles et du bot.

Actuellement, le jeu n'est pas disponible en réseau. Pour ce faire, il suffit de suivre la documentation.

2.2 Présentation et considérations du cours d'IHM

Nous avons cherché à avoir l'interface la plus claire possible pour les joueurs. Ainsi, les petites cartes en bas de l'interface ont été créées pour afficher la main de chaque joueur. Une barre latérale présente le classement temporaire et un bouton mis en évidence permet de poser des questions au bot. Nous n'avons pas eu le temps, mais un deuxième bouton peut facilement être implémenté pour demander de l'aide à d'intelligences artificielles en lui demandant quel coup permettrait d'être le plus efficace afin d'améliorer l'expérience utilisateur.

Sur la page d'accueil, nous retrouvons un bref résumé des règles ainsi qu'un bouton pour configurer une partie. Par défaut, il y a un joueur humain contre trois intelligences artificielles.

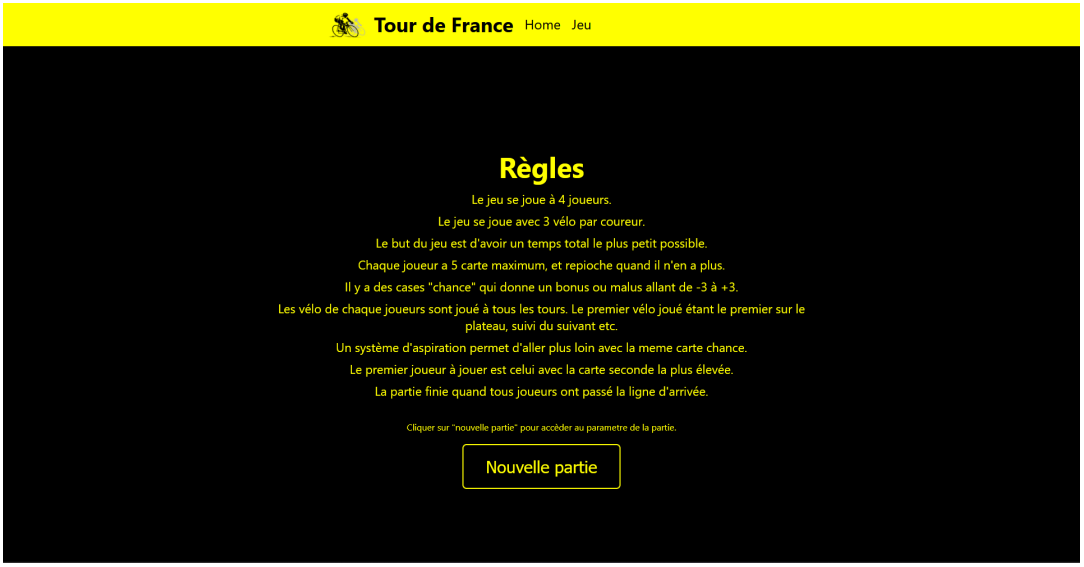

FIGURE 1 – Page d'accueil du site

Sur la page du jeu, on retrouve les différents drapeaux des équipes avec un numéro pour indiquer de quel coureur il s'agit. De plus, les cases qui tintées de jaune indiquent une case qui peut être visitée par le joueur actuel en fonction de la position de ses coureurs et des cartes de sa main.

FIGURE 2 – Page de jeu. On peut y voir les petits drapeaux des équipes qui se déplaceront.

2.2.1 Facilité d'UX

Dans le but d'avoir la meilleure interface possible, nous avons cherché à avoir chaque case du plateau indépendante via un fichier SVG [1].

Grâce au fichier SVG, nous avons pu avoir accès aux cases de manière indépendante afin de :

1. Placer les cyclistes sur le plateau
2. Permettre de faire le choix de la case qui sera visée par le vélo joué.

Ces considérations sont arrivées très tôt afin d'améliorer l'UX des utilisateurs.

2.3 Architecture et choix d'implémentation

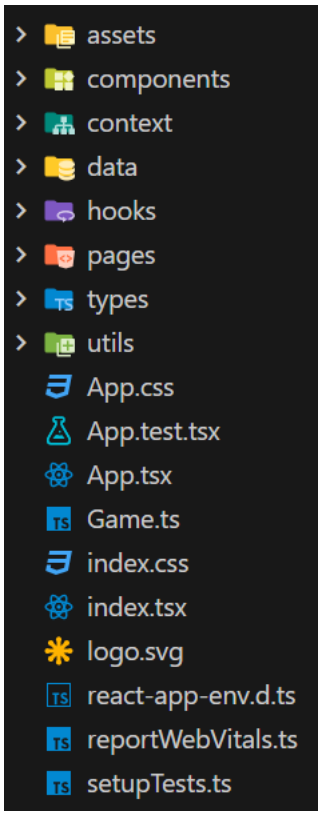

FIGURE 3 – Liste des dossiers et fichiers dans le dossier `src` de l'application React.JS

Nous avons suivi une structure classique pour un projet React.JS avec TypeScript. La seule différence avec un template pour ce genre d'application est le fichier `Game.ts` qui contient les fonctions et objets nécessaires au fonctionnement

1. Le fichier SVG a été fourni par Simon Loir, qui a fait le projet avec le groupe 6 lors de l'année académique 2021-2022

de notre moteur de jeu (cf. Boardgame.io). De plus, le dossier `data`, qui est généralement utilisé pour interagir avec une API, contient des constantes. Nous allons approfondir cela plus loin.

2.3.1 assets

Ce dossier contient les images et composants SVG utilisés dans le projet. On y retrouve la carte et les cartes.

2.3.2 components

Ici sont stockés tous les composants de l'application, de la carte avec la gestion de l'affichage des vélos sur celle-ci au bot, en passant par les mains des joueurs.

Ce dossier est dépendant du dossier `assets` pour les images qui y sont présentes ainsi que du dossier `hooks` pour la gestion de l'état interne du composant.

2.3.3 context

Ce dossier, comparable aux `hooks`, contient la gestion de l'état du jeu à un plus haut niveau. On y retrouve les paramètres de la partie qui sont partagés entre les pages et permettent de définir le client de Boardgame pour créer les appels automatiques à l'IA lorsque c'est nécessaire. On retrouve également le contexte pour le choix du vélo et de la carte à utiliser. Si, avec toutes ces informations, la case finale choisie est trop incertaine, certains mécanismes sont mis en place pour lever l'ambiguïté via des interactions avec l'utilisateur.

2.3.4 data

Traditionnellement utilisé pour interagir avec une ou plusieurs API, ce dossier a trouvé un autre but dans le cadre du projet.

On y retrouve les données de représentation du plateau utilisées par Boardgame et des joueurs utilisées par les `components` afin d'afficher les noms d'équipe et leurs drapeaux sur la carte.

2.3.5 hooks

Contient la logique et la gestion des données dans l'affichage des composants et l'interaction avec les données. Je pense notamment à l'interaction avec l'API Prolog afin de poser des questions au bot.

2.3.6 pages

Ici se créent les différentes pages de l'application. La page `Home` permet de définir les paramètres de la partie afin de savoir qui est humain et qui est une IA. On y retrouve aussi un bref rappel des règles. Ces données sont stockées via le contexte de l'application (cf. context). Ensuite, elles servent à créer le client Boardgame suivant les besoins, qui se trouve à la page `game`.

2.3.7 types

Déclaration des types TypeScript utilisés dans l'application. Ces types ont aidé au développement pour savoir ce qui était attendu en paramètre des fonctions et composants et le retour.

2.3.8 utils

Quelques fonctions utiles. Je citerais `shuffle` et `deepCopy` qui sont très connues et sont des algorithmes trouvés en ligne. Il y a également deux fonctions de traduction afin de passer du plateau défini dans le dossier `data` aux clés du SVG qui affiche la carte contenue dans `assets`.

Ayant reçu le SVG après la création de notre plateau de jeu (sous la forme d'une structure de données, cf. data), nous n'avons pas voulu réimplémenter toutes les clés de la carte. Nous avons alors créé une fonction pour traduire dans un sens et dans l'autre.

2.3.9 Fichier de configuration

Le fichier `exemple.env` contient toutes les informations nécessaires pour les variables d'environnement. On y retrouve surtout le point API du serveur Prolog.

2.4 Intégration de l'intelligence artificielle

Boardgame.io propose un système de "bot" qui sont des intelligences artificielles faciles à intégrer aux jeux. On peut donc facilement les intégrer au projet. Cependant, ayant eu des problèmes de temps, l'intégration ne s'est pas passée comme prévu et nous avons dû ruser pour intégrer l'IA Prolog dans le jeu. Nous vérifions si le joueur courant est un joueur IA, et dans ce cas, nous appelons la fonction `bot` qui s'occupe de faire l'appel à l'API Prolog.

2.5 Améliorations futures

Parmi les contributions futures au projet, on peut citer la mise en réseau du jeu via un mode multi-joueur.

3 Intelligence artificielle

3.1 Réflexion sur l'intelligence artificielle

Dans le cadre du projet, il était également demandé qu'un ou plusieurs joueurs puissent être simulés par une intelligence artificielle.

Après réflexion, la meilleure stratégie dans le jeu du Tour de France semble être d'avancer le plus loin possible tout en s'assurant une certaine proximité des vélos du joueur. Les autres joueurs doivent être considérés afin d'assurer qu'aucune chute ne soit causée lors du tour.

Dans ce cadre, l'intelligence artificielle de ce projet a été réalisée avec cet objectif. Son but est donc de calculer le meilleur coup d'un joueur afin d'aller le plus loin possible tout en gardant ses vélos groupés et en s'assurant de ne pas causer de chute.

3.2 Implémentation

Afin de réaliser l'intelligence artificielle en Prolog, plusieurs éléments ont été nécessaires. Ces éléments sont présentés dans les sections suivantes.

3.2.1 Implémentation du plateau

Le but de l'intelligence artificielle est de simuler un joueur humain. Elle doit donc être capable de jouer un coup sur le plateau. À cette fin, il est nécessaire que l'intelligence artificielle possède une représentation du plateau afin de pouvoir le parcourir et calculer une nouvelle position en fonction du coup joué.

La représentation du plateau est similaire à celle réalisée dans l'application web. En effet, le plateau est représenté par une série de prédicats contenant deux données. La première donnée est une clé représentant la case du plateau. Ces clés sont les mêmes que celles utilisées dans l'application web afin d'assurer une coordination entre les deux. Le deuxième élément donné par le prédicat est une liste des cases accessibles en un coup depuis la case actuelle. Cette liste permettra par la suite de manière récursive de calculer une nouvelle position en fonction de la position actuelle et de la valeur d'une carte.

En plus de cette liste de prédicats représentant le lien entre les différentes cases du plateau, une liste de prédicats permet aussi de stocker le nombre de places disponibles sur une case. De ce fait, il est possible pour l'intelligence artificielle de savoir si son mouvement pourra être joué ou non.

Enfin, une dernière liste de prédicats reprend les clés des cases situées après la ligne d'arrivée. Ce prédicat permet à l'intelligence artificielle de savoir si un coup lui permet de passer la ligne d'arrivée, ou si un vélo est déjà dans une position située au-delà de la ligne, et donc de ne pas calculer de branche pour ce vélo.

3.2.2 Récupération des données

Une fois que le plateau est implémenté et disponible pour l'intelligence artificielle, celle-ci doit avoir accès à l'état actuel du plateau afin de pouvoir réaliser ses calculs. La récupération des données se fait donc sur la base des données fournies par le websocket. Ces dernières sont passées sous forme d'une liste de listes. Il s'agit donc d'une liste constituée de sous-listes, et chacune de ces sous-listes reprend les données d'un joueur. Chaque sous-liste commence par un numéro allant de 0 à 3, 0 correspondant au joueur joué par l'intelligence artificielle.

Par la suite, les données passées par le websocket sont analysées par un prédicat permettant d'extraire les données d'un joueur en se basant sur le numéro de ce dernier.

3.2.3 Arbre minimax

Ces deux premières étapes correspondent à la mise en place nécessaire aux calculs du coup par l'intelligence artificielle.

Les calculs sont réalisés sur la base d'un arbre minimax, implémenté par le prédicat du même nom. Ce dernier a pour profondeur le nombre de cartes du joueur joué par l'intelligence artificielle, car cela représente le nombre de coups qu'il est possible de connaître pour ce joueur. L'idée de ce prédicat est de parcourir chaque mouvement possible de manière récursive et d'appliquer ce mouvement à l'état actuel. Ensuite, l'arbre est continué en calculant les coups possibles depuis ce nouvel état, et ce jusqu'à obtenir un des cas limites de la récursion.

Ces cas limites sont au nombre de trois et sont les suivants : le nouvel état ne présente plus de cartes, il est donc impossible de calculer de nouveaux coups ; le nouvel état est un état final ; ou la profondeur maximale est atteinte.

Un cas limite représente une feuille de l'arbre, et on y calcule donc le score du mouvement ayant permis d'atteindre cette feuille. Ce score est ensuite remonté dans l'arbre afin de définir quelle branche devrait être empruntée par l'intelligence artificielle.

3.2.4 Calcul du score

Lorsqu'une feuille de l'arbre est atteinte, il est donc nécessaire d'y calculer le score du mouvement. Pour ce faire, le prédicat *score* prend en argument le mouvement, la liste de tous les vélos du joueur et renvoie le score. Le score se base donc sur le mouvement permettant au joueur d'aller le plus loin possible tout en gardant ses vélos les plus groupés possible, et ce sans causer de chute.

Afin de calculer l'avancement permis par le mouvement, une soustraction est réalisée entre la nouvelle et l'ancienne position du vélo. Pour ce qui est de la distance entre les vélos, cette dernière est calculée en soustrayant la position des deux vélos non déplacés à celle du vélo déplacé. À ce résultat, on ajoute la valeur 5, valeur arbitraire permettant tout de même une certaine marge et s'assurant que

l'intelligence artificielle n'est pas trop restreinte par cette contrainte de proximité. Le résultat de ce calcul de proximité est ensuite soustrait à la distance permise par le coup, ce qui donne le score de la feuille.

3.2.5 Cas limite d'une chute

Afin d'assurer une continuité entre l'intelligence artificielle et l'application web, il a été décidé de faire en sorte que l'intelligence artificielle ne cause pas de chute lors des déplacements qu'elle crée. À chaque branche, on teste donc si le mouvement d'arrivée cause une chute. Si c'est le cas, alors la branche est coupée afin de ne pas calculer d'éléments qui ne seront pas gardés car ne remplissant pas cette contrainte.

4 Bot explicateur

Afin de réaliser le bot capable de répondre aux questions de l'utilisateur, un template de code nous a été fourni avec les consignes du travail. Ce template a été utilisé comme inspiration pour notre bot. Cependant, le bot de ce projet ne suit pas exactement la même structure que celui présenté dans le template.

Afin de réaliser notre bot, nous nous sommes donc inspirés du bot fourni avec les consignes, ainsi que des exercices réalisés en travaux pratiques. Notre bot est donc composé de deux grandes catégories : la partie analyse de la question et la partie production de la réponse. Chacune de ces parties sera détaillée dans une sous-section dédiée.

4.1 Analyse de la question

En ce qui concerne l'analyse de la question, celle-ci est réalisée à partir du prédicat `lire_question/2`. Pour ce faire, le prédicat divise la chaîne de caractères reçue sur le caractère représentant l'espace, qui est le caractère `' '`, en ignorant les caractères spéciaux. Cette division génère une série de mots qui est ensuite stockée dans une liste qui servira dans la suite du bot.

4.2 Production de la réponse

Afin de produire une réponse en lien avec la question posée, nous avons décidé de nous inspirer du template donné en utilisant l'idée d'avoir un mot clé lié à une réponse. Pour ce faire, notre bot présente des prédicats `mot_cle/1` listant tous les mots-clés correspondant aux questions auxquelles notre bot est capable de répondre.

Pour assurer qu'une réponse soit produite même en cas de faute de frappe, la différence entre les mots de la question et le mot-clé est calculée. Si cette dernière est raisonnablement acceptable, alors les mots sont considérés comme étant identiques. Le calcul de la différence entre les deux mots est réalisé par le prédicat *isub*. Ce dernier renvoie donc une valeur de différence, et si cette dernière est inférieur à 0.83, alors les deux mots seront acceptés.

Dans le but de lier ces mots-clés aux réponses correspondantes, nous avons réalisé les prédicats `reponse/2`. Ces prédicats prennent un premier élément comme entrée. Ce premier élément correspond au mot-clé précédemment présenté, et chacun des mots des prédicats `mot_cle` est lié à un prédicat `reponse`. Le deuxième argument du prédicat `reponse` est un élément de sortie. Il correspond à la réponse à donner à l'utilisateur suite à sa demande.

5 Lien entre les éléments

5.1 Client

Du coté client, nous avons fait le choix d'utiliser un module de gestion de requête HTTP très connu qu'est `Axios`. Il nous permet s'effectue des requêtes `get` à l'API prolog où s'exécute le bot. En revanche, passé une structure de donnée `JSON` au point API de l'IA nous a posé plus de problème

5.2 Serveur Prolog

Du côté du serveur Prolog, les requêtes sont gérées par des sockets liant les différents fichiers en Prolog correspondant à l'intelligence artificielle et au bot. Pour ce faire, le module HTTP est utilisé.

5.2.1 Set up Serveur

D'abord, on initie le serveur en local sur le port 8080. Différentes routes sont ensuite utilisées par les prédicat `http_handler/3`. Le premier permet de rediriger les requêtes à la racine du serveur vers une autre page, à savoir la page d'accueil qui elle-même est géré par le second handler. Le second handler ainsi que les deux derniers permettent de gérer les 3 routes que possèdent notre serveur : ./home, ./bot/Question et ./ia, la première s'occupe juste de centraliser les requêtes, la seconde est la route pour accéder au bot à question et la dernière permet d'accéder à l'IA.

5.2.2 Différentes Routes

Ensuite, chaque route est associé par le handler à un prédicat :

Pour /home, le prédicat `home_page` permet d'afficher une page html afin de voir la documentation de l'API.

Pour /bot, le prédicat `answer` récupère avec `produire_reponse` la réponse du bot puis le renvoie en JSON.

Pour /ia, le prédicat `answer_ia` extrait d'abord les données de la requête avec `http_read_data` puis transforme à l'aide du prédicat `spliter/2` les données en une liste de liste afin que l'IA puisse prend en paramètre le board. Avec le prédicat `get_move_IA`, on récupère le move donné par l'IA puis change le format de Move avec les prédicats `format_move` et `move_to_char` avant de le renvoyer dans un JSON au client.

5.2.3 Answer_ia en Détail

Pour mieux comprendre comment fonctionne le prédicat `answer_ia`, prenons un exemple. Imaginons que nous souhaitions renvoyer l'état du Board actuel. Le prédicat va alors extraire les données de la requête pour retrouver une chaîne de caractères du style :

Exemple de chaîne de caractères

```
"0.1;2;3.37_A_left;0_B_left;0_B_left*
1.5;2.24_A_left;0_B_left;0_B_left*
2.3.37_A_left;0_B_left;0_B_left*
3.8;5;7;9;8.27_A_left;0_B_left;0_B_left"
```

Alors, le prédicat `spliter/2` permet de formater la chaîne de caractères en une liste de listes de ce style :

Liste de listes

```
[[0,[1,2,3],[37_A_left,0_B_left,0_B_left]],
 [1,[5,2],[24_A_left,0_B_left,0_B_left]],
 [2,[3],[37_A_left,0_B_left,0_B_left]],
 [3,[8,5,7,9,8],[27_A_left,0_B_left,0_B_left]]]
```

Pour ce faire, la chaîne de caractères est d'abord split en fonction des `*` avec le prédicat `split_string` pour donner une liste de ce style :

Liste scindée par '*'

```
["0.1;2;3.37_A_left;0_B_left;0_B_left",
 "1.5;2.24_A_left;0_B_left;0_B_left",
 "2.3.37_A_left;0_B_left;0_B_left",
 "3.8;5;7;9;8.27_A_left;0_B_left;0_B_left"]
```

Puis le prédicat `sub_split` est appliqué sur chaque élément de la liste. Ce prédicat split la chaîne de caractères en fonction des `.` pour donner une liste de ce style :

Liste scindée par '.'

```
[["0", "1;2;3", "37_A_left;0_B_left;0_B_left"],
 ["1", "5;2", "24_A_left;0_B_left;0_B_left"],
 ["2", "3", "37_A_left;0_B_left;0_B_left"],
 ["3", "8;5;7;9;8", "27_A_left;0_B_left;0_B_left"]]
```

Ensuite, le prédicat utilise le prédicat `split_again`. Celui-ci permet d'abord pour chaque premier élément des sous-listes de tranformer la valeur en une valeur numérique puis utilise le prédicat `subsub_split` pour encore split les deux derniers éléments de chaque sous-liste. Il prend donc en paramètre les deux sous-listes et applique le prédicat `split_int` pour la première sous-liste (correspondant à la main du joueur) et applique le prédicat `split_atom` pour la seconde sous-liste (correspondant à la positions des vélos du joueur). Les deux prédicats vont d'abord split en fonction des `;` ce qui va donner ceci :

Liste scindée par ';'

```
[["0", ["1", "2", "3"], ["37_A_left", "0_B_left", "0
   _B_left"]],
 ["1", ["5", "2"], ["24_A_left", "0_B_left", "0_B_left
    "]],
 ["2", ["3"], ["37_A_left", "0_B_left", "0_B_left"]],
 ["3", ["8", "5", "7", "9", "8"], ["27_A_left", "0
    _B_left", "0_B_left"]]]
```

Puis pour le premier prédicat, il va tranformer en valeur numérique tout les éléments de la liste qui sera générer par le split. Par exemple : `[1, 2, 3]`. Le second prédicat de son coté transforme en atom tout les éléments de la liste qui sera générer par le split. Par exemple : `'37_A_left';'0_B_left';'0_B_left'`.

Une fois que cela est appliqué pour chaque joueur, alors on arrive à la liste finale qui peut directement être donnée au prédicat de l'IA.

On récupère ensuite le résultat dans le format-ci : `carte utilisée,nouvelle position du vélo bougé,ancienne position du vélo bougé`.

Le format ne pouvant pas directement être renvoyer dans un JSON, nous utlisons le prédicat `format_move/2` pour récupérer les données sous la forme-ci : `[carte utilisée,(nouvelle position du vélo bougé,ancienne position du vélo bougé)]`.

Le prédicat `move_to_char` permet ensuite de retransformer la valeure de la carte utilisée en un string puis de tranformer la liste en une chaine de caractère afin de formater ensuite le move dans un JSON qu'on renvoie au client.

6 Déploiement

6.1 Lancer le client

Pour lancer le jeu, vérifier que vous avez installé Node.JS afin d'installer les dépendances du client web.

Pour celà utilisé la commande suivante dans le dossier `client` ;

```
npm install
```

Vous pouvez ensuite lancer la commande suivante qui démare un serveur pour le client web :

```
npm start
```

6.2 Lancer le serveur

Ici, vérifier que vous avez installé prolog afin de lancer le serveur.

Ensuite, rendez vous depuis le dossier courant du projet dans dossier `server` qui contient les fichiers prolog nécéssaire

Vous pouvez utiliser la commande suivante :

```
swipl socket.pl
```

6.3 Vérifier que les deux serveur sont lancer

Vous pouvez dès lors vous rendre à l'adresse : http://localhost:8080/home de votre navigateur internet afin d'obtenir une courte documentation de l'API prolog.

Pour jouer au jeu, allez à l'adresse : http://localhost:3000 de votre navigateur internet et lancer une partie.

7 Organisation interne

Afin de mener le projet à bien, nous avons décider de nous répartir les taches sur base de nos compétences.

7.1 Répartition du travail

C'est ainsi que Louca et Margaux ont tout deux travaillés sur le serveur Prolog et surtout la création de l'intelligence artificielle, du bot et des websockets. Pendant que Yannis et Youlan s'occupaient de créer un client web pour interagir avec le jeu.

Louca a pris en charge la création des websockets et l'intercation entre les requêtes envoyées depuis le plateau et les modules du bot et de l'intelligence artificielles. Pendant ce temps, Margaux a réalisé les modules du bot et de l'intelligence artificielle.

Yannis et Youlan se sont réparti les taches sur base de leur familiarité avec le sujet. Yannis s'est occupé de l'affichage du jeu avec React et CSS. Pendant que Youlan développait le jeu avec toutes ses subtilité en matière de règles.

7.2 Problème rencontré

Chaque sous-groupe a eu ses problèmes, cela nous rappelant les erreurs du passé en matière de communications. Nous avons donc pu rapidement identifier quand et comment ses problèmes pouvaient survenir afin de les résoudre au plus vite ou dans le meilleur des cas, s'en prémunir.

Nous avons également du faire attention à la différence de niveau dans le groupe, certain ayant du mal à comprendre le code des autres. Il y a donc eu un effort en matière de lisibilité du code via des noms de variables très long mais très explicites, de même pour les fonction et prédicats parfois accompagné de documentation.

8 Amélioration globale du jeu

Etant donné les limitations imposées par les consignes du travail, notre version du jeu du tour de France n'est pas optimale par rapport à ce qu'elle pourrait être. En plus des différentes améliorations discutées dans la partie client de ce rapport, d'autres potentielles améliorations peuvent être notées.

Tout d'abord, l'implémentation du moteur de jeu en Prolog permettrait d'implémenter une seule fois toute les règles du jeu. Cependant, comme Prolog ne permet pas le stockage de données modifiées aussi simplement que d'autres langages, cette tâche pourrait s'avérer compliquée.

D'autre part, Une des plus grandes difficultés, plus par manque d'expérience avec ce langage, sera la gestion d'utilisateurs distant pour un mode multi-joueurs et des parties en simultanée. Une approche par micro-service pourrait alors être envisagée afin de laisser le serveur prolog se charger du jeu et d'avoir un autre serveur en JavaScript, PHP, Python ou TypeScript pour gérer les utilisateurs et le coté multi-joueur.

Enfin, l'intelligence artificielle réalisée se base sur une stratégie déduite, et pourrait probablement être améliorée en mettant au point une meilleure stratégie après avoir passé un certain temps à joueur au jeu. De plus, la version actuelle de notre intelligence artificielle choisi de ne pas profiter des aspirations. Cette légère amélioration pourrait permettre à notre intelligence artificielle d'obtenir de meilleures performances.

9 Conclusion

Ce projet nous a permis d'approfondir nos connaissances dans le domaine du web par la création d'une application utilisant une librairie capable de gérer un plateau, et dans le langage Prolog par la création de websockets, d'un bot et d'une intelligence artificielle.

De plus, ce projet nous a permis de mettre en pratique les concepts théoriques présentés lors du cours, comme la réalisation d'un arbre minimax afin de déterminer le coup optimal d'un joueur.

Ce projet a donc représenté une bonne opportunité de mettre en place des concepts théoriques en pratique et d'approfondir nos connaissances sur différents points.