Ensta Bretagne
2, rue François Verny
29806 Brest cedex
France

Projet d'informatique

FISE 2024

21 avril 2022

Jeu du tarot Africain

Nicolas Dias
Thomas Prévost

Table des matières

Le code source complet du projet est disponible sur GitHub :

https://github.com/thomas40510/ProjetInfoS2

I. Présentation du projet

Nous avons choisi, pour ce projet d'informatique, d'implémenter en Python le jeu du Tarot Africain, qui est un jeu de cartes opposant de deux à quatre joueurs, et demandant stratégie, réflexion mais aussi une part de chance.

1. Règles du jeu et objectifs du projet

Une partie de Tarot Africain se déroule assez simplement. Pour commencer, il faut se munir d'un jeu de tarot, que l'on trie. On place alors d'un côté les atouts (du 1 au 21 ainsi que l'excuse) et d'un autre le reste des cartes séparé et trié par couleur [1].

Début de partie

Donner, à chaque joueur, un paquet trié de cartes de couleur : elles représentent les vies du joueur et sont placées devant lui, visibles de tous les autres joueurs. Les cartes de tarot seront mélangées entre chaque partie.

Déroulement d'une partie

Au début de chaque manche, un joueur désigné distribue 5 cartes d'atout à chaque joueur. Chaque joueur, en commençant par le celui ayant distribué, placera alors un pari sur le nombre de tours qu'il va gagner après avoir consulté ses cartes. Chaque joueur est libre de proposer un nombre de son choix, à l'exception du dernier à parler devant proposer un nombre tel que l'ensemble des paris soit différent du nombre de cartes que les joueurs ont en main (5 au premier tour).

Les joueurs jouent alors à tour de rôle une carte de leur main, en commençant toujours par celui ayant distribué. Gagne le tour le joueur ayant placé la carte la plus forte (à noter que le joueur ayant placé l'excuse choisit sa valeur en fonction de sa stratégie : soit elle est la plus forte, soit elle est la plus faible). Ce joueur débutera le tour suivant, et ainsi de suite jusqu'à ce qu'il ne reste aucune carte en main.

En fin de manche, les joueurs perdent autant de vies qu'il leur manque de plis pour remplir leur pari (par exemple, un joueur ayant placé un pari de 3 et n'ayant gagné qu'un pli perd $3 - 1 = 2$ vies).

Commence alors la manche suivante, au cours de laquelle seront distribuées 4 cartes (puis 3 à la suivante, puis 2, puis une seule).

1. pique, carreau...

Tour à une seule carte

Le tour à une seule carte est particulier : chaque joueur ne regarde pas sa carte, mais la pose sur son front. Il est donc capable de voir uniquement les cartes des autres joueurs et estime s'il est capable de remporter ce pli.

Là encore, s'applique la même condition sur le dernier joueur à parier (la somme totale ne peut pas être égale à 1).

Fin de partie

Une partie se finit à l'issue de la manche d'une seule carte. À ce moment là, le jeu peut continuer, chaque joueur gardant le nombre de vies qu'il avait à l'issue de la partie précédente. Le joueur ayant distribué transmettant ce rôle à celui à sa gauche.

Le jeu continue alors de partie en partie jusqu'à ce que tous les joueurs sauf un soient éliminés.

2. Pistes envisagées pour l'implémentation

Il a alors été initialement choisi, pour implémenter le jeu de Tarot Africain, le découpage en plusieurs classes : une classe `Joueur`, une classe `Carte`, une classe `Manche` et une classe `Partie`. Structure permettant de manipuler ces différents objets afin de dérouler une partie complète.

Une première structure du programme a alors été schématisée à partir de ces pistes techniques, soulevant alors plusieurs questions au sein du binôme, impliquant finalement de reconsidérer le découpage qui avait été réalisé initialement.

3. Choix techniques

Notre choix, après l'étude préalable ayant été menée, s'est donc porté sur un découpage plus simple que celui initialement prévu : la classe `Carte`, bien trop simple, n'ayant pas de raison d'exister.

Le projet s'articule donc, à ce stade, autour de 3 classes principales :

— La classe `Joueur`, de laquelle sont héritées les classes `JoueurHumain` et `JoueurBot` permettant de faire jouer un humain ou la machine ;
— La classe `Manche` permettant de dérouler une manche de la partie ;
— La classe `Tarot` permettant de dérouler la partie.

En particulier, la classe `JoueurHumain`, permet de faire jouer l'utilisateur en lui demandant de rentrer chaque coup, tandis que la classe `JoueurBot` permet de faire jouer la machine en calculant chacun de ses coups, généralement de manière récursive, grâce au module `TA_Bots.py`. Le fonctionnement de ce dernier est explicité en II.2.

II. Le programme sur papier

Le programme construit autour d'un fichier principal contenant le jeu en lui-même, auquel viennent se greffer deux modules, respectivement pour les joueurs

et le calcul des coups de la machine.

1. Présentation générale

Le fonctionnement du programme, dans son ensemble, est plutôt simple. On pourra se référer entre autres au diagramme de classes et au diagramme d'actions présentés en annexe.

— L'utilisateur lance une partie par l'exécution de la classe `Tarot`. Il spécifie son nom.
— Est alors initialisée la partie, avec le nom des joueurs ; une manche est créée ainsi que les joueurs (humain ou machine) qui sont en lice.
— À chaque tour, un leader est désigné (soit le premier joueur en début de manche, soit le vainqueur du tour précédent), et chaque joueur place ses paris et joue. Si le joueur est la machine, son coup est calculé en fonction des coups des autres joueurs et leurs paris ; si le joueur est humain, le programme lui demande son pari et la carte qu'il choisit de poser, connaissant les paris déjà placés et les cartes posées.
— À l'issue de chaque tour est identifié le vainqueur du pli, et les joueurs perdent le nombre de point correspondant à l'écart entre leur pari et leur nombre de plis.

2. Fonctionnement en détails — jeu contre la machine

III. Le programme en fonctionnement

1. Figures imposées

Les figures imposées suivantes sont vérifiées par le programme :

1. **Factorisation du code** : le programme est composé de trois modules différents (`Tarot_Africain.py`, `TA_Bots.py` et `Joueurs.py`) responsables de diverses fonctionnalités bien spécifiques, et d'un module de tests, et est donc factorisé ;
2. **Documentation et commentaire du code** : tout naturellement, le programme est intégralement documenté et commenté ;
3. **Tests unitaires** : des tests unitaires ont été écrits pour chaque partie du programme ;
4. **Création d'un type d'objet (classe)** : cinq objets ont été créés dans le cadre de ce programme, à savoir `Joueur, Manche` et `Tarot`, mais aussi `JoueurHumain` et `JoueurBot` ;
5. **Récursivité** : la récursivité est exploitée dans le jeu de la machine, permettant d'explorer les différentes situations possibles afin d'identifier le coup optimal ;
6. **Héritage entre deux types créés** : les classes `JoueurHumain` et `JoueurBot` héritent toutes deux de la classe `Joueur` ;

7. **Héritage depuis un type intégré** : nous avons implémenté une « mémoire », héritée du type `list` de Python, prenant en charge les logs de toutes des différentes manches jouées au cours d'une partie, permettant non seulement d'effectuer des traitements statistiques sur les parties afin de permettre l'amélioration du fonctionnement des bots, mais aussi de conduire des tests unitaires. Une voie envisageable serait également de s'en servir pour proposer au joueur des statistiques sur sa partie.

2. Tests effectués

De nombreux tests ont été effectués afin de vérifier le bon fonctionnement et la cohérence du programme, ces derniers étant réunis dans le fichier `test_Tarot_Africain.py`. Nous détaillons donc ici les quatre jeux de tests unitaires ayant été écrits et exécutés.

Cohérence de la distribution

Un élément essentiel au bon déroulement de la partie est le fait qu'une même carte ne puisse être distribuée deux fois dans une même manche.

Cette condition est donc vérifiée pour chaque nombre de joueurs et chaque nombre de cartes possibles, pour un total de $20 \times 4 \times 3 = 240$ tests. Ce nombre, correspondant à 20 tests par nombre de joueurs pour chaque nombre de cartes en main, permet en effet de vérifier statistiquement la cohérence de la distribution des cartes malgré que cette dernière soit aléatoire. Nous estimons que ce nombre de tests, s'ils reviennent tous positifs, est suffisant pour vérifier cette condition.

En l'occurrence, le résultat est bien revenu positif à chaque exécution.

Cohérence des paris

De la même manière, une partie ne peut se dérouler correctement s'il est possible de placer des paris faux ou incohérents. Il était donc essentiel d'écrire des tests unitaires vérifiant cette cohérence.

Ainsi, pour chaque nombre possible de joueurs et chaque nombre possible de cartes en main, les paris des joueurs, aussi bien humains que machines, sont recueillis et testés afin de vérifier leur cohérence. En particulier, il est vérifié le fait que la somme des paris ne soit pas égale au nombre de cartes en main (du fait de la condition sur le pari du dernier joueur à parier), mais aussi que chaque pari est positif et inférieur au nombre total de plis possibles.

Sont donc, de la même manière que pour la cohérence de la distribution, exécutés $20 \times 4 \times 3 = 240$ tests. Tous sont revenus positifs, la cohérence des paris est donc bonne.

Cohérence de la partie

Au-delà de la cohérence des paris, est également essentielle la cohérence du choix des cartes, que ce soit par un joueur humain ou par la machine. En particulier,

il est nécessaire de vérifier que ne sera jamais posée par un joueur une carte qu'il ne possède pas, et qu'il ne puisse pas poser deux fois la même carte.

Ce jeu de tests vérifie donc, pour chaque nombre de joueurs et chaque nombre de cartes en main possibles, si chaque carte posée était bien possédée par le joueur avant d'être jouée et qu'il ne l'a pas jouée deux fois.

Ce qui constitue donc un total de 1 800 tests.

Perte de points

La dernière chose à vérifier est le fait que les joueurs perdent bien des vies.

Pour cela, toujours pour chaque nombre de joueurs et chaque nombre de cartes en main possibles, il est vérifié pour vingt manches différentes qu'il n'en existe aucune au cours de laquelle aucun joueur n'a perdu de vie.

3. Limitations observées

Plusieurs limitations ont été identifiées, majoritairement en lien avec le fonctionnement du programme dans la console, sans interface utilisateur. Cette restriction étant à l'origine d'un affichage pas toujours intuitif et clair pour l'utilisateur, qui doit, pour jouer, se contenter d'un affichage textuel.

Malgré tout, en termes d'implémentation uniquement technique, aucune limitation majeure n'a pu être identifiée, mise à part une optimisation éventuelle des décisions du bot, qui dans le cadre de ce projet ne nous semble pas pertinente.

4. Perspectives et améliorations

Ces constatations orientent donc tout naturellement la suite du projet vers l'intégration d'une interface utilisateur permettant d'apporter une solution aux différentes limitations induites par le mode de jeu en textuel, que nous avons pu identifier lors des tests du jeu.

Aucun choix technique n'a encore été fait concernant l'implémentation de l'UI, même si PyQt5 semble être une solution envisageable.

Une autre amélioration envisageable serait également de permettre à l'utilisateur de se créer un profil stocké sur une base de données, lui permettant d'accéder à différentes statistiques concernant ses parties précédentes. Cette piste ne sera pas nécessairement traitée immédiatement, la priorité étant donnée à l'implémentation de l'UI, mais reste ouverte comme perspective intéressante du projet.

Une dernière perspective d'amélioration, tenant uniquement de l'expérience utilisateur, serait d'ajouter la possibilité pour l'utilisateur de choisir la difficulté de la machine avant de commencer la partie. Ceci lui permettrait de jouer contre une machine plus faible s'il ne parvient pas à gagner contre une machine plus forte. À noter cependant que cette amélioration nécessiterait une refonte conséquente des algorithmes de décision, et serait hors de propos dans le cadre de ce projet, bien qu'intéressante.

Annexe I. Diagramme de classes

Annexe II. Fonctionnement schématique d'une partie