DUBOIS Julien
Avec
GUIRAUD Philippe

Réussite des alliances

Rapport de projet

Mai 2020, Polytech Clermont-Ferrand

Introduction

Ce rapport résume les choix qui ont été pris concernant l'organisation générale du code de la réussite des alliances, certaines fonctions obligatoires et les extensions ajoutées. Philippe et moi utilisons Github afin de garder une trace de chaque modification et simplifier le partage du projet à distance. Je me suis principalement occupé des extensions (notamment l'interface graphique) mais j'ai également aidé Philippe pour certaines fonctions un peu plus compliquées (comme reussite_mode_manuel). En cas de problème nous utilisons Discord qui permet d'intégrer du code facilement. Nous nous sommes servi de la documentation officielle du langage Python (https://docs.python.org) et de documentations non-officielles pour Tkinter (impossible de trouver une documentation officielle). Notre programme peut fonctionner en mode graphique comme en mode console (voir Extensions > Des graphismes).

Organisation du code

Le code du jeu est réparti en 7 grandes catégories :

- **Constantes** : contient toutes les constantes du jeu, directement accessible au début du fichier.
- **Mode console** : contient des fonctions liées à l'affichage en mode console (exemple : *afficher_reussite()*). Permet de créer facilement des menus et de vérifier que les entrées utilisateur correspondent à ce qui est attendu.
- **Mode graphique** : propose des fonctions liées à l'interface graphique (exemple : *dessiner_reussite()*). Simplifie la création de widgets tkinter et des menus du jeu.
- **Utile jeu** : contient toutes les fonctions essentielles au fonctionnement du jeu (exemple : *init_pioche_alea()*). Elles sont utilisées un peu partout dans le programme.
- **Statistiques** : propose des fonctions statistiques tel que *res_multi_simulation()* et liées à la création de graphiques.
- **Fichier** : C'est dans cette zone que sont rassemblées toutes les fonctions liées aux opérations sur des fichiers (exemple : *init_pioche_fichier()*).
- **Principal** : Les fonctions principales sont regroupées ici (exemple : *reussite_mode_auto()*). Ces fonctions se chargent d'appeler toutes les autres.

Fonctions obligatoires

Les fonctions principales (*reussite_mode_auto()*, *reussite_mode_manuel()*, *lance_reussite()*) étant très ouvertes, voici les choix que nous avons pris les concernant :

- *lance_reussite()* demande à l'utilisateur s'il souhaite charger la pioche depuis un fichier ou non puis lance la partie. Si la pioche est chargée depuis un fichier, une vérification est faite sur celui-ci afin de s'assurer qu'il comporte bien une pioche valide. Une fois la partie terminée, le joueur a la possibilité de sauvegarder la pioche utilisée (peu importe le mode choisi).
- *reussite_mode_manuel()* affiche après chaque action les cartes visibles ainsi qu'un menu proposant de piocher une carte, d'effectuer un sauter ou encore de quitter.
- *preparer_reussite()* propose au joueur de régler certaines variables initiales comme le mode de jeu (manuel ou automatique), le nombre de cartes du jeu, activer ou non l'affichage ou encore régler le nombre de tas maximal pour gagner.

Chacunes de ces fonctions existent également en version graphique (même nom mais finissant par *_gui()*).

Extensions

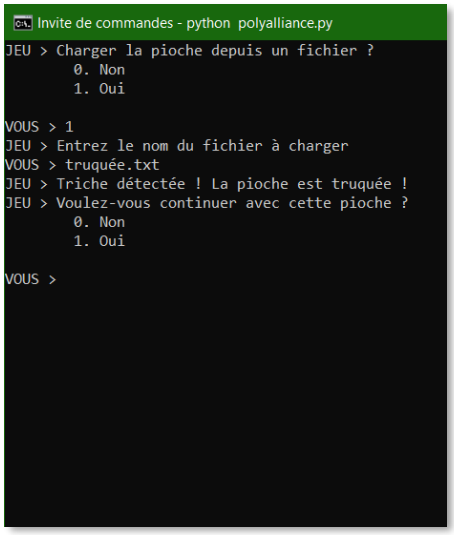
Nous avons choisi d'ajouter au jeu un maximum d'extensions. Toutes sont intégrées au programme principal et accessibles depuis l'interface (à la fois en console et graphiquement).

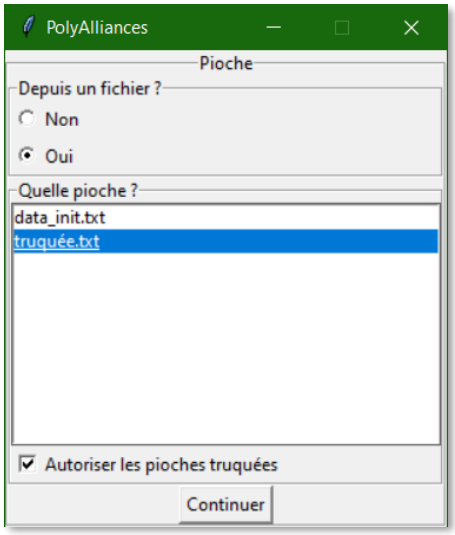
Des graphismes

Afin de rendre le jeu plus convivial, une interface graphique basée sur le module Tkinter a été ajoutée au jeu. Toutes les fonctionnalités accessibles en console le sont aussi en mode graphique. Par défaut, le jeu est en mode graphique mais il est possible de forcer le mode console en changeant la variable *FORCE_CONSOLE* à *True*. Les images sont chargées au début et référencées dans un dictionnaire. Le joueur navigue dans différents menus avant de pouvoir véritablement jouer. Voici un exemple de partie :

Gestion des pioches truquées

A l'aide de la fonction *verifier_pioche()* le jeu peut détecter les pioches « truquées » (nombre de cartes incorrect ou cartes en doubles) et avertir le joueur. En mode console, la confirmation du joueur est nécessaire avant de charger une telle pioche. En mode graphique, les pioches truquées ne peuvent être chargées que si la case « Autoriser les pioches truquées » est cochée (voir ci-dessous).





Statistiques

Le jeu intègre 4 extensions liées aux statistiques :

- `res_multi_simulation()` permet d'effectuer rapidement et sans affichage un grand nombre de parties pour ensuite retourner le nombre de tas restants. Afin d'étoffer les tests statistiques, nous avons décidé de lui ajouter un paramètre optionnel supplémentaire `niveau_triche`. Cet argument peut valoir `0`, `1` ou `2` (`0` par défaut). S'il vaut `0`, les simulations sont réalisées sans aucune modification des pioches générées. S'il vaut `1`, les pioches sont modifiées à l'aide de `meilleur_echange_consecutif()`. Enfin un niveau de `2` lance l'utilisation de `meilleur_echange()` (cette fonction cherche le meilleur échange possible afin d'améliorer la pioche).
- `statistiques_nb_tas()` affiche certaines données statistiques suite à une simulation. Cette fonction possède également un paramètre optionnel `niveau_triche`. Elle calcule la moyenne, la valeur minimale et maximale du nombre de tas et le nombre de parties renvoyant respectivement moins et plus de tas que la moyenne.
- `probabilite_victoire()` effectue autant de simulation qu'il n'y a de cartes et renvoie une liste contenant une estimation de la probabilité de victoire en fonction du nombre de tas maximal pour gagner.
- Ces fonctions sont utilisées pour afficher des graphiques (à l'aide de `creer_graphique()`). Voici un exemple de graphique :