année 2022-2023



**Projet Algo S4**

**Risk en python**

ASINSA gr.96

ROSART vittore

pascal antoine

olmeta vincent

chapard clement

# Détail de la structure et du fonctionnement du projet

## structure des fichiers et interactions

Le projet contient en tout 4 fichiers python : *Main*, *Carte*, *Rules* et *Widget*.

Comme le nom l’indique, le fichier principal est le fichier *Main*, c’est celui qui sert à lancer pour jouer au jeu. Cependant, il ne contient en soi que l’affichage du menu, des interfaces de login joueurs, du tableau des scores et des règles. C’est ici que la liste d’objet JOUEURS est créée à partir d’un fichier csv. Une fois les joueurs connectés, le contenu de ce fichier n’est plus utilisé. C’est le fichier *Carte* qui prend ensuite le relai avec les fichiers *Rules* et *Widget* avec pour seule input une liste d’objet de joueurs connectés. Le fichier *Carte* affiche le jeu et gère les interactions du joueur. Il est le lien avec la partie backend du jeu ; le fichier *Rules* et la classe *Game*. Cette dernière contient les fonctions qui font les calculs de probabilités, le contrôle des règles, le placement initial des joueurs, les missions, … (détails dans découpage fonctionnel). La classe Territoire est également défini dans ce fichier. *Widget.p*y contient quant à lui des widgets plus complexes utilisés dans notre fenêtre de jeu.

## Liste imports

Tkinter, csv tkinter.font tkinter.Scrolledtext subprocess sys pygame pygame.locals hashlib random json time pandas math

## Classes et fonctions utilisées

### dans main.py

Dans le fichier main, nous définissions 3 classes différentes.

La première et la plus longue, c’est MainMenu. Dans la fonction \_\_init\_\_ , on initialise la fenêtre tkinter et ses paramètres, on importe les images nécessaires, on lit le fichier Joueurs.csv pour créer la liste d’objet joueurs (classe définie plus tard) et on définit les polices d’écriture.

Ensuite les fonctions suivantes sont utilisées pour afficher les différents widget (Button, Scale, Label,…) nécessaires au parcours du menu, et n’ont pas d’intérêt particulier à part sur la maitrise du module Tkinter : boutons\_menu, new\_game, create\_widgets, backmenu, escape, escapeTOP, escapeTOP2, back, playerbutton\_enter, playerbutton\_leave, Rules\_info, STARTLOGIN, login\_win, classwin, NewAccount

Deux fonctions intéressantes sont les fonctions liées aux mots de passe. En effet, la fonction addplayer et checkmdp utilisent l’algorithme de hachage SHA256 pour ne jamais stocker directement le mot de passe d’un joueur. Lors de la création de compte, le mot de passe est encodé et stocké sous forme de hash. Lors de la vérification du mot de passe, les caractères entrés sont hashés et comparé à ce qui est stocké pour le joueur sélectionné. C’est aussi dans ces deux fonctions qu’on vérifie sur un joueur est sélectionné, si le mot de passe ou le pseudo sont conformes, si le joueur n’est pas déjà connecté.

Enfin la fonction pygame\_launcher lance la partie en initialisant la classe PygameWindow et lançant la boucle Pygame.

La classe Joueur est également définie dans ce fichier, et elle sera utile tout le long du jeu. Elle prend en paramètre le pseudo, le mot de passe (hashé), et le nombre de victoires.

La dernière classe CreateToolTip n’est pas de nous (source citée dans le code) et sert à créer un petit message d’information quand on survole un widget Tkinter. Cette dernière est utilisée sur le widget back\_button créé par la fonction STARTLOGIN.

Le détail de chaque fonction est indiqué en docstring, pour des raisons de lisibilité et de clarté ils ne seront pas détaillés ici.

### dans carte.py

Ce fichier ne contient qu’une seule classe, PygameWindow, qui initialise la fenêtre de jeu et contrôle toutes les interactions du jeu. Elle prend en paramètre la taille de la fenêtre (plein écran) et la liste d’objet des joueurs connectés.

La fonction \_\_init\_\_ est remplie de tous les paramètres, listes, état, constantes, polices, couleurs et widgets qui seront utilisés plus tard. C’est là aussi qu’on créer la fenêtre Pygame.

La fonction la plus longue de cette classe est celle qui gère la boucle principale, elle sera détaillée en dernier.

Les fonctions charger\_image, charger\_coord\_texte, init\_couleurs sont executées depuis \_\_init\_\_. charger\_image va créer les images pygame et les mask de toutes les images qui seront utilisées. charger\_coord\_texte va lire un fichier json et retourner un dictionnaire avec la position pour chaque territoire de là où sera affiché le nombre de troupes. init\_couleurs va mettre à jour tous les fichiers images des territoires pour qu’ils aient la couleur de leur joueur. Cette liste de couleur est définie dans \_\_init\_\_. Pour s’y faire, dans une boucle parcourant la liste objet des territoires et après avoir associé la bonne couleur du joueur, la surface va être transformée en un mask, composé de 0 s’il n’y a pas de pixel et 1 s’il y en a un. Les surfaces sont composées d’une grande partie transparente. Ensuite une surface est créée à partir de ce mask, qui sera noir et blanche en fonction des 0 et des 1 (noir si pas de pixel, blanc si pixel). Le noir est ensuite retiré et la couleur du joueur est appliqué à la surface. On met ensuite à jour la surface du territoire. C’est la façon la plus rapide et efficace de changer uniformément la couleur d’une surface sur fond transparent. La fonction changer\_couleur fonctionne de la même façon, avec un paramètre couleur en plus.

Les fonctions afficher\_fenetre, affiche\_des, add\_border, add\_texts, afficher\_mission servent à afficher les éléments graphiques sous certaines conditions (phase de jeu).

Lorsqu’un joueur sélectionne un pays, il faut que cela soit visible facilement. Le nom et le défenseur du territoire sélectionné est affiché sur la fenêtre de jeu mais le territoire est également assombri par la fonction changer\_lumi. Elle divise par 1,5 les valeurs (r,g,b) de la couleur du pays si il n’est pas sélectionné et les multiplie par 1,5 si le pays était déjà sélectionné ce qui a pour effet de changer la luminosité en gardant la même teinte.

La sélection des pays se fait grâce à la liste self.select . Elle est mise à jour par la fonction select\_deux\_surfaces qui prend en paramètre le territoire sélectionné. Si la liste est vide, on ajoute le territoire, si elle est à moitié remplie et qu’on reclique sur le même territoire, il est supprimé. Si le territoire est différent du premier, il est ajouté. Si on reclique sur le deuxième territoire sélectionné, il est retiré. On ne peut pas retirer la première sélection avant d’avoir enlevé la deuxième. Quand un territoire est ajouté ou enlevé de la liste, on met à jour sa luminosité.

Les fonctions end\_turn, next\_player, check\_loss entrent dans la gestion des tours et des règles. Elles changent la phase du jeu si nécessaires, passent au joueur suivant et vérifient si un joueur n’est pas éliminé (plus de territoire). La fonction end\_turn vérifie également si le joueur qui fini son tour a rempli sa mission. Si oui, la partie prend fin.

joueur\_win est appelée lors de l’affichage de l’écran de victoire pour mettre à jour dans le fichier csv le score du joueur gagnant (score += 1)

Pour finir : la boucle principale gérée par main\_loop. Chaque phase de jeu est associée à un chiffre dans la variable self.view ; 0 pour le renforcement, 1 pour l’attaque, 2 pour le déplacement, 3 pour l’écran de victoire et 4 pour la répartition des troupes après avoir conquis un territoire.

C’est dans ces différentes phases qu’on affiche certaines parties de la fenêtre et certains widgets. Il faut ensuite vérifier pour chaque clic si la souris est à une position d’un bouton/territoire/widget. Il faut penser à mettre à l’échelle les coordonnées de la souris à celles du widget s’il s’agit d’un mask car les coordonnées dans le mask et dans la fenêtre ne sont pas les mêmes.

Il faut vérifier une partie des règles aussi et afficher des messages d’erreur si besoin (répartition de toutes les troupes effectuées, déplacement uniquement entre territoires adjacents, territoire non conquis, nombre minimum de troupes atteint, …)

La phase d’attaque est la plus complexe. On y affiche les sélecteurs de troupes et de dés ainsi que le bouton attaque et aide.

Si un joueur veut attaquer un certain pays non adjacent au sien, il peut sélectionner ces 2 pays et cliquer sur le bouton d’aide qui affichera le chemin le plus efficace pour aller conquérir ce territoire (voir détail fonction complexe pour plus d’informations).

Sinon, il sélectionne deux territoire (le 1er étant le sien) pour attaquer le territoire. Si l’attaque réussie, le nombre de troupes est mis à jour. Si le territoire est conquis, le jeu passe temporairement en phase 5 pour répartir des troupes dans ce nouveau territoire.

Cette fonction main\_loop vérifie donc constamment la phase du jeu et les actions effectuées pour gérer toutes les interactions possibles. Les conflits ont également été réglé pour éviter tout problème dans l’enchainement des phases jusqu’à la victoire d’un joueur.

### dans rules.py

Ce fichier contient 3 classes : territoires, Game et Missions.

La première permet de définir toutes les caractéristiques d’un territoire : le joueur auquel il appartient, la couleur qu’il doit prendre, la zone(continent) dont il fait partie… Certains paramètres comme joueur ou couleur sont modifiés au cours de la partie.

La deuxième Game contient la plupart des fonctions utilisées pour jouer au cours de la partie. C’est ici que les règles du jeu sont codées