GREFFARD Brice

MARANDON Lorenzo

Projet IHM

Sommaire

Introduction :

Ce projet a été fait avec :

Python version: 3.9.20

Tkinter version: 8.6

Codé sur un environnement Windows (et testé sur Linux)

/ !\ : il nécessite les modules : **tkinter, PIL, random et idle3** de python**.**

Ce projet a pour but de manipuler et comprendre l'interfaçage homme machine en utilisant des fonctionnalités de la bibliothèque graphique Tkinter et de la programmation orientée objet, le tout par la création d'un jeu vidéo tour par tour. Les qualités générales à respecter sur ce projet sont l'ergonomie, du feedback à la confortabilité visuelle, la cohérence de la structuration du projet et du déroulement des tours.

Attention :

/!\ : due à une gestion différente des fenêtres entre chaque window manager , nous n'avons pas pu implémenter le blocage de la croix qui permet de fermer une fenêtre. Mais l'utilisation de la croix par l'utilisateur n'a pas été prévu et peux causer des erreurs (non critique). Il est attendue que l'utilisateur n'utilise que les boutons qui ont été mis à sa disposition par les développeurs.

Les images utilisées pour ce projet ont été générée par IA soit trouvée sur google, nous n'avons fait que les modifier pour qu'elle soit ajustée à notre jeu.

Tout les codes qui ont été générer par IA ou trouvé sur internet sont indiquer dans de leurs docstring. La majorité des codes qui n'était que de la mise en forme de texte ont été fait sur chatGPT

/ !\ : la police personnalisée ne se charge pas forcément, à ce moment-là une autre police que celle prévu est chargée (Arial).

Description du graphe des classes :

Relations entre les classes

Héritage :

Roturier hérite de Personne :

> Roturier ajoute des fonctionnalités spécifiques comme la gestion de l'humeur, de la capacité de production et du vieillissement.

Paysan et Artisan héritent de Roturier :

> Ces classes spécialisent les Roturiers pour définir des types de production et des comportements distincts.

Noble hérite de Personne :

> Noble ajoute des capacités de gestion des Villages et des relations avec d'autres Nobles.

Association :

Village contient des Paysan et des Artisan :

> Un village gère une liste de villageois (instances de Paysan et Artisan) et organise leurs interactions, comme la collecte d'impôts, les révoltes, ou les festivités.

Noble gère des Villages et des relations féodales :

> Un noble peut posséder plusieurs villages, représentés par une liste ou un dictionnaire, et entretenir des relations de vassalité avec d'autres nobles.

Dépendance :

Village utilise les Roturiers pour des calculs comme la production ou la gestion des événements internes.

Noble interagit avec les Roturiers via les villages pour percevoir des ressources ou imposer des changements dans la structure sociale.

Pour l'utilisateur :

Un effort a été fait du côté des visuels pour donner envie à l'utilisateur, des images, des Widget personnalisée et une police d'écriture personnalisée ont été choisi pour éviter le coté monotone de tkinter

Le programme a été pensée pour empêcher le joueur de commettre des erreurs , les seuls boutons cliquables sont des boutons qui son sure de ne pas créer d'erreur.

Un didacticiel aurait dû être intégré dans le jeu mais nous avons manqué de temps donc ce sont des images dans le dossier de jeu. Mais pour guider le joueur dans le jeu, des tooltips ont été mis en place.

Nous avons fait en sorte que chaque action du joueur ou des IA provoque un changement sur la carte ou dans l'affichage tête haute (HUD). Pour que le joueur ais toujours l'impression que ses actions on un impact.

Feedback :

Dans un jeu vidéo, le retour d'information doit être conçu de manière à optimiser l'expérience du joueur. Ici, nous avons imaginé une solution simple et efficace :

-Un carré entourant la case ou a cliqué l'utilisateur est un bon feedback pour savoir il se trouve sur le plateau.

-Le menu de gauche qui change en fonction de la case choisie et qui permet à l'utilisateur de toujours savoir ce qu'il peut faire

-Les cases de la carte changent en fonction des évènements et des actions des joueurs ce qui permet d'avoir un vu direct sur l'état de la partie.

Choix de développement:

Du côté du visuelle :

Nous avons fait une carte qui ressemble à un plateau de jeu classique pour ne pas perdre le joueur.

Cette carte fait partie des 3 éléments principaux de notre jeu avec la barre d'information en haut de l'écran et le menu a gauche.

Nous avons fait la barre d'information en haut car c'est un élément récurant dans les jeux de gestions classique et nous voulions que la prise en main soit facile et rapide, elle permet de voire rapidement les informations importantes que sont l'argent/ressource , les unités et l'évènement en cours. Nous y avons également ajouté le bouton « fin de tour » pour qu'il soit facilement accessible et un menu déroulant « Menu » .

Par souci de facilité pour le joueur et le développeur , nous avons centraliser les actions possibles dans le menu a gauche de l'écran. Ce menu change en fonction des actions du joueur et bloque automatiquement toute actions impossibles pour éviter les problèmes. Nous avons tout de même séparé le bouton « fin de tour » car il a une importance spéciale et que nous voulions éviter qu'il soit cliqué malencontreusement.

Du côté jouabilité :

Nous avons essayé au mieux de respecter le cahier des charges donné et les seuls changements sont des ajouts pour améliorer l'expérience utilisateur.

Nous nous sommes écartés de la consigne de base qui était que le joueur peut interagir avec toute la carte directement et nous avons fait un territoire que le joueur peut étendre et développer. Nous avons fait en sorte que le joueur puisse se développer et attaquer seulement des cases qui juxtapose son territoire. Cela donne une meilleure impression de progression dans la partie et ça donne aussi un coté stratégique car vous savez que pour attaquer ou être attaque , il faut être proche de l'ennemi.

Lors de la conquête de case neutre via le bouton « conquérir » , l'action de devoir cliquer sur une case valide et retourner sur le bouton « conquérir » peut-être redondant mais nous avons gardé ce fonctionnement car dans une partie classique il est rare d'avoir assez de ressource/argent pour conquérir un grand nombre de case d'un coup. Et ça permet de garder les boutons centralisés.

Pour la guerre et la gestion des unités, nous avons fait le choix d'avoir des unités blessées après une guerre et qui reviennent au tour suivant. Cela permet d'éviter de tout perdre en une seule guerre et ça évite aussi que l'utilisateur puisse détruire un autre joueur en ne gagnant qu'une seule guerre.

Nous avons décidé qu'un village ne serait pas détruit même lorsqu'il n'a plus d'habitant car construire un village coute chère et il n'y a pas de raison que le village disparaisse s'il n'y a personne.

Par soucis de facilité de programmation et car ça existe dans la vie réelle, nous avons permis au territoire d'être découpé : Si un ennemi décide de conquérir une case qui liait deux bouts de votre territoire, vous aurez deux territoires distincts que vous pouvez toujours relier en reprenant la case.

Par soucis de réalisme et pour éviter de pouvoir enchainer les guerres, il faut au moins 1 unité pour aller en guerre.

Les IA qui servent d'ennemi ne sont pas très développé ni très intelligente et ne peuvent faire que quelque action (voire dans la documentation de GestionIA.py).

Les boutons importants possèdent un tooltip(info-bulle) qui sert de mini tuto mais qui n'apparait que si le joueur reste x seconde sur le bouton. Cela permet au nouveau joueur d'être guidé dans leur prise de décision et ça ne dérange pas les joueurs expérimentés car ils n'ont pas le temps de voir s'afficher l'info-bulle.

Une option de sauvegarde / chargement de partie a été pensé mais trop tard et nous n'avions pas les connaissance/ compétence pour l'implémenter

Modularité du code :

Le code de notre jeu vidéo est structuré en plusieurs modules distincts afin d'en faciliter la compréhension, la maintenance et l'évolution. Chaque module est dédié à une fonctionnalité spécifique, ce qui permet de garder une organisation claire et efficace.

Dans main.py :

C'est le fichier à exécuter pour lancer le jeu, il contient une suite de classe qui permettent de changer de manière fluide le menu dans lequel on se trouve.

Dans la classe Option et ParametreJeux se trouve des paramètres choisis par le joueur qui vont changer la façon dont le plateau de jeux se créé.

C'est dans la classe PlateauJeux que se déroule réellement l'initialisation et le lancement du jeux.

Dans GestionMenu.py :

C'est ici qu'est le moteur du jeux, tout l'affichage des widgets et les actions de l'utilisateur passe par ici

Pour gérer les interactions, nous utilisons des structures de dictionnaires que nous avons complexifiées (Voire la documentation) et qui permettent une gestion optimale des boutons. Cela garantit une organisation claire et un accès rapide aux différentes fonctionnalités, tout en facilitant les ajustements et l'évolutivité du système.

Dans GestionEvenement.py :

C'est ici que se passe le choix et l'exécution des évènements.

Les évènements sont ceux demandés dans le sujet , rien de plus n'a été ajouté.

Dans GestionIA.py :

C'est ici que sont gérée les IA , leurs actions et leurs rapports de fin de tour. Les IA appellent les même fonction qu'un joueur humain et sont limitées par les mêmes choses.

Les IA ont été pensée pour juste remplacer une action humaine, il ne serait donc pas difficile de faire du jeu un jeux multijoueur en local sur un même écran ou même un jeu solo ou l'utilisateur joue tous les Nobles.

Dans GestionGuerre.py :

C'est ici que les guerres sont gérées, de l'affichage jusqu'au calcule.

La grande difficulté des guerres a été de devoir gérer l'attente de la fonction principale le temps que l'utilisateur fasse ses choix. La méthode .wait_window() qui permet cela a été trouvée lors de recherche internet.

Et dans les guerres animer , un long moment a été nécessaire pour trouver un moyen de bloquer l'exécution de la fonction le temps d'afficher les gifs tout en continuant de faire marcher les gifs. Là encore plusieurs recherche internet sur le fonctionnement asynchrone de tkinter ont été nécessaire.

Dans GestionClass.py :

Toutes les classes utiles pour le jeu sont ici.

Une hiérarchie de classes a été mise en place pour éviter la redondance et les attributs et méthodes sont adaptés aux responsabilités de chaque classe.

Les méthodes de destruction d'instance comme tuer_villageois() ou detruire_village() sont gérer par des class superviser pour éviter les problèmes si ces objets sont référencés ailleurs.

L'utilisation de @property pour les assesseurs et les mutateurs aurait dû être faite mais elle a été oubliée durant un bon moment de la programmation et les assesseur/ mutateur était utilisée trop de fois pour pouvoir changer. Nous avons donc décidé de continuer de ne pas utiliser les @property pour garder une constance dans le code.

Dans CreationMap.py :

C'est la méthode qui nous permet de créer une carte de taille modulable, Nous avons fait le choix de faire un algorithme simple pour avoir une carte plus ou moins cohérente. Cette cohérence donne un côté « réaliste » ce qui permet une meilleure immersion plutôt que simplement des cases posées au hasard.

Dans WidgetCustom.py :

C'est ici que sont définit nos widgets customisés.

Nous avons eu la nécessité de faire des widgets customisés pour embellir le jeu final et pour simplifier la programmation.

La spinbox a dû être recrée car elle ne permettait pas de mettre la même valeur minimal et maximal et on en a profité pour ajouter deux niveaux d'incrémentation pour améliorer le gameplay

Le bouton a dû être recrée pour pouvoir choisir le style de chaque mot et pouvoir mettre des mots et des images dans un même bouton. On en profiter pour l'adapter au mieux a notre structure d'affichage de widget.

Cette architecture modulaire rend le développement plus intuitif et simplifie l'ajout ou la modification de fonctionnalités.

Robustesse du code :

Nous avons conçu le code de manière à éviter toute situation pouvant générer des erreurs. Aussi bien pour le joueur que pour les IA. Cela garantit une expérience fluide et sans imprévus. Même si cela limite les actions de l'utilisateur à cliquer sur des boutons.

Cependant, cette approche ne signifie qu'aucun ou peut de système de gestion d'erreurs ont été implémenté, car les situations problématiques sont simplement empêchées par la conception même. Mais a posteriori , plus de try, except aurai pu être mis en place surtout sur des fonctions qui sont fréquemment utilisées comme modifier_argent et modifier_ressource qui peuvent vite rendre le jeu illogique.

Fonctionnement général :

En lancent main.py, vous tomberez sur l'écran d'accueil. Sur cet écran vous pouvez soit :

-Lancer une partie

-Aller dans les options

-Quitter le jeu

Dans la fenêtre d'option, vous pouvez :

-Comme dans un jeu classique, choisir la taille de la fenêtre .

-Vous pouvez aussi choisir d'avoir ou non le didacticiel : Le didacticiel n'a pas été implémenté en jeu car nous manquions de temps, mais ça aurait été une ou des images explicatives de tous les menus avec des bouton « fermer » et « suivant » pour respectivement fermer le didacticiel ou passer à l'image suivante.

-Et enfin vous pouvez choisir d'activer la triche ou non : cette fonction avait été créée avant tout pour faire du débogage mais on a décider que ce serai une bonne idée d'en faire une option pour les utilisateurs qui souhaiterai voire les limites du jeu.

Lorsque vous lancez une partie vous devrez tout d'abord choisir les paramètres de la partie :

-Le nombre de joueur et la taille de la carte sont des paramètres assez explicites.

-Le choix de la seed aurait pu être fait ici mais nous avons décidé de nous concentrer sur d'autre chose.

Une fois vos choix fait vous pouvez :

-Soit vous dire que vous avez oublié quelque chose dans les options et cliquer sur « retour ».

-Soit tout va bien et vous pouvez lancer une partie avec vos paramètres personnalisées.

Une fois dans le jeu, vous n'avez que 2 façons de gagner / perdre :

-Sois vous vassalisez tout le monde

-Sois vous détruisez tout le monde

Les IA peuvent aussi gagner de la même manière

Vous pouvez faire des actions indiquées par les boutons activés dans le menu à gauche.

Pour savoir pourquoi un bouton est désactivé il y'aura sois un des prix en rouge pour indiquer un manque de ressource. Soit l'info-bulle donnera les conditions pour que le bouton soit actif.

Durant une partie vous devez faire attention à plusieurs choses :

-Le bonheur de vos villageois : qui offre des bonus de production lorsqu'il est haut et a l'inverse des malus lorsqu'il est bas. Il peut même y avoir une révolte.

-Les ressource/argent et le nombre d'unité de vos ennemis : vous pouvez toujours vérifier si vous pouvez vassaliser un ennemi . Et vous ne voulez pas avoir moins d'unités que vos ennemis proches car vous courrez le risque de perdre une guerre et de vous faire capturer.

-L'expansion de votre territoire : vous voulez avoir la place de faire un maximum de villages mais attention a ne pas trop s'approcher des frontières ennemis, vous risquez de vous faire attaquer.

-Vos propre ressource/argent : Vous voulez pourvoir recruter des unités en cas de problème mais vous voulez aussi construire/développer vos villages pour générer plus. Mais vous ne voulez pas être trop bas en ressource et vous faire vassaliser par un ennemi (Les IA dans le jeu n'ont pas la capacité de vassaliser un joueur).

Si vous décidez d'attaquer un ennemi, une nouvelle fenêtre va apparaitre pour vous permettre de choisir les unités que vous envoyez au combat :

-Vous devez envoyer au moins 1 unités et vous devez avoir les ressources pour faire la guerre.

-Si toutes ces conditions sont vérifiées, Les boutons « guerre rapide » et « guerre animé » serons disponible. Il n'y a aucune différence en termes de résultat entre les deux.

Après la guerre, la fenêtre vous montrera le rapport avec qui a gagné / perdu et le rapport des unités.

Une fois votre tour terminé (plus d'argent/ressource ou plus d'action à faire) , vous pouvez cliquer sur « fin de tour » en haut a gauche. Ce bouton va mettre fin a votre tour et lancer le tour des autres joueurs.

/ !\ : Une fenêtre d'attente de tour a été faite mais du au tour quasi instantané des IA , elle ne s'affiche pas

Une fois que c'est de nouveau votre tour , une fenêtre de rapport des tours de IA vas s'afficher.

Conclusion :

Si nous avions à refaire le projet aujourd'hui , il y'aurai des changements majeurs dans la méthode de travail car nous manquions de réflexion sur le déroulé entier du programme :

Pour faciliter la programmation , nous avions séparé le projet en 3 grand axe :

-La création et gestion de la carte et des villages

-La gestion des actions de l'utilisateur

-Et la gestion de la guerre et des unités

Même si cette méthode c'est prouvé efficace pour le début du développement , nous nous somme heurter à des problèmes de lors de l'implémentation de la dernière phase :

-Les unités et surtout les chevaliers auraient dû être des classes à part entière et pas juste des nombres. Cette découverte tardive nous a contraint à devoir faire des pirouettes de programmation pour implémenter les chevaliers comme demandé.

-Et d'autre élément comme les vassaux ,la capture des chevaliers ou la capaciter de sauvegarder/charger une partie aurai dû être plus réfléchie au début du projet lors de la phase de brainstorming.

Nous manquions également de connaissance et de pratique sur tkinter et sur la poo(surtout de pratique dans le cas de la poo). Beaucoup de choix graphique et logique aurait été différent si nous nous étions plus documentés avant de commencer à programmer.

Pour conclure , ce projet a été une bonne introduction a l'interfaçage homme machine, nous avons constaté que le développement d'une IHM efficace nécessite une réflexion approfondie en amont, aussi bien sur le plan fonctionnel que technique. Il ne s'agit pas seulement de créer une interface visuellement attrayante, mais de concevoir un système ergonomique et intuitif pour les utilisateurs finaux. Cela implique de faire des choix stratégiques, notamment :

-Les fonctionnalités à inclure (et leur priorité).

-Les éléments d'interface (boutons, menus déroulants, cases à cocher, etc.) et leur disposition.

-Les interactions possibles entre les différentes parties de l'IHM.

Nous avons également appris qu'une interface mal planifiée peut rapidement devenir difficile à modifier ou à étendre si les bases ne sont pas solides.

Ce projet nous a permis de mieux appréhender les enjeux et les exigences liés à la création d'interfaces homme-machine tout en nous initiant à des outils pratiques.