Conception Agile de Projets Informatiques

Laure GENTILI & Elyes KHALFALLAH

Avant de commencer, nous aimerions préciser que nous n'allons pas expliquer le code rendu dans ce rapport, mais plutôt nos choix d'implémentation et raisonnements derrière nos décisions: les commentaires présents suffisent pour bien comprendre leur fonctionnement.

Introduction :

Notre projet de "Planning Poker" représente une application logicielle destinée à faciliter la planification et l'estimation des tâches au sein d'une équipe de développement pratiquant une des nombreuses méthodes Agiles. L'objectif principal est d'optimiser le processus de planification en permettant aux membres de l'équipe de voter de manière anonyme sur la complexité des différentes tâches à accomplir.

Dans le cadre de ce projet, nous avons adopté une approche centrée sur l'expérience utilisateur, visant à fournir une interface intuitive tout en garantissant une fonctionnalité robuste. Pour atteindre ces objectifs, nous avons utilisé plusieurs patrons de conception, tels que le Singleton, Command, ainsi que des principes de développement orienté objet en Python.

Nous avons eu quelques difficultés le long du développement, notamment des contraintes qui ont ralenti la finalisation du projet selon notre plan initial. Les défis rencontrés, tels que la gestion d'imports récursifs et la coordination des différentes frames au sein de l'interface utilisateur, ainsi que la découverte de nombreux outils dont nous nous sommes jamais servis avant (GitHub, les Tests Unitaires, éventuellement Doxygène), ont constitué des opportunités d'apprentissage. Néanmoins, ce projet Planning Poker reflète notre engagement à relever des défis techniques et notre volonté de fournir un travail de qualité, malgré les imprévus. Ce rapport détaillera notre approche et nos choix de conception.

Justifications des Patterns Utilisés :

Nous avons choisi d'appliquer plusieurs patterns de conception afin d'optimiser la structure, la maintenabilité et la cohérence de notre code.

Singleton :

Le patron de conception Singleton a été adopté pour fair ene sorte qu'une unique fenêtre de notre programme s'affiche à la fois. Cette spécification évite d'avoir plusieurs jeux démarrés en même temps sur un ordinateur. En utilisant le pattern Singleton, nous favorisons une structure robuste et unique, permettant d'éviter des bugs concernant des modifications multiples du fichier JSON, par exemple.

Command :

Nous nous sommes servi du patron de conception Command pour encapsuler les actions et les opérations de l'interface utilisateur. En structurant les interactions utilisateur en boutons, nous permettons une extensibilité du système. A tout moment, nous pouvons créer, ou retirer l'accès à une fonctionnalité. Cela facilite également la gestion des actions utilisateur, comme la sélection de cartes, et permet d'ajouter de nouvelles fonctionnalités sans modifier directement les objets existants.

Observer :

Le patron de conception Observer est implicitement présent grâce à l'utilisation de la bibliothèque Tkinter en Python. Tkinter suit une architecture Model View Controller (MVC), où les widgets et leurs contrôleurs associés (gestionnaires d'événements) agissent comme des observateurs des changements du modèle. Par exemple, dans la classe JouerMenu, lorsque l'utilisateur sélectionne le nombre de joueurs, le contrôleur associé déclenche un événement, et les vues pertinentes (libellés) sont automatiquement mis à jour pour refléter le nombre de joueurs choisi. Cette séparation entre l'interface utilisateur (vues) et la logique sous-jacente (modèle) illustre le patron Observer, améliorant la flexibilité et la maintenabilité dans le code.

Justification des choix techniques :

Nos choix techniques ont malheureusement été faits davantage par obligation que par choix. Ne nous sentant pas à l'aise avec le WebDev ou les Frameworks, nous avons décidé d'utiliser entièrement et exclusivement Python. Cela s'est avéré être une arme à double tranchant, car notre choix a énormément facilité la création de menus, de boutons, etc en utilisant des classes d'objets et de la POO, mais ça nous a crée d'énormes obstacles lorsque l'algorithme du jeu réel a dû être écrit. Avec les fichiers SVG qui ne sont pas les plus faciles à manipuler, la culmination de tout cela a conduit à notre plus grand regret pour ce projet : ne pas avoir un jeu entièrement fonctionnel.

Néanmoins, nous développerons plus loin les raisons de nos choix :

Tkinter :

Tkinter a été choisi pour le développement de l'interface utilisateur en raison de sa simplicité d'utilisation et de son intégration native avec Python. Cela a permis un développement rapide de l'interface sans introduire de dépendances externes.

Architecture Orientée Objet :

L'approche orientée objet a été adoptée pour structurer le code de manière modulaire et réutilisable. Les différentes parties de l'application, telles que le menu principal, le menu de jeu, etc, sont encapsulées dans des classes, facilitant la maintenance et l'extension du code. Celà nous permet aussi une plus facile implémentation des Patrons de Conception.

Utilisation de bibliothèques tierces (svglib, reportlab, ...):

Ces bibliothèques ont été intégrées pour traiter les fichiers SVG, générer des images à partir de ceux-ci, et les afficher dans l'interface utilisateur au dessus de leurs boutons respectifs. Cela a permis d'améliorer la qualité visuelle de l'application au niveau du jeu. Aussi, pygame nous a permi l’implémentation d’une musique de fond douce accessible dans le menu Options.

Documentation :

Dans le cadre du projet "Planning Poker", une attention particulière a été accordée à la documentation pour garantir la compréhension du code source et faciliter la maintenance future. Toutefois, en raison de contraintes temporelles et techniques, nous avons principalement opté pour l'utilisation de commentaires intégrés directement dans le code source, plutôt que de recourir à la découverte d'un outil tiers dédié à la documentation, tel que Doxygen.

L'utilisation de commentaires intégrés a permis de documenter le code au fur et à mesure de son développement, facilitant la compréhension de la logique et des fonctionnalités spécifiques à chaque partie du programme. Bien que l'implémentation d'un outil de documentation automatisé n'ait pas été réalisée dans le cadre de ce projet, nous envisageons d'approfondir la documentation une fois que le jeu sera sans bugs. Cette approche nous a permis de maintenir une documentation continue tout au long du processus de développement, évitant ainsi de différer la documentation à la fin du projet et garantissant une compréhension immédiate du code.

Tests :

Dans un projet comme Planning Poker, où plusieurs fonctions interagissent, il est important de mettre en place des tests unitaires dans les parties cruciales du code. Cependant, étant donné nos lacunes... il est assez difficile pour nous d'en implémenter aisément.

De ce fait, nous avons préparé une liste de tests relativement simples que nous avons réussi à implémenter dans test.py (qu'il faut lancer à part entière) :

- Verifier qu'aucun nom est vide
- Verifier que le mode est bien mis à 0 dans le mode Strict
- Verifier que le mode est bien mis à 1 dans le mode Moyenne

Nous avons aussi créé une liste de tests que nous aurions aimé implémenter mais n'ont pas su comment faire:

- Tests pour les fonctionnalités de vote :
  Étant donné que le cœur du Planning Poker réside dans le vote des utilisateurs sur la complexité des tâches, il serait important de tester cette fonctionnalité. Un test pour verifier que chaque bouton renvoie correctement la valeur attribuée aurait été pertinent

- Tests pour la logique de jeu :
  Une vérification de la logique du jeu (Strict: les cartes sont les mêmes, ou pas) (Moyenne: pour les cartes a et b, la moyenne est c, donc le résultat doit être d) serait le test le plus important de tout notre code. Le jeu entier compte sur le bon fonctionnement de ces calculs et vérifications, donc l'implémentation de ce test serait primordial

- Tests pour les interactions avec l'interface utilisateur :
  Ces tests sont parmi les plus simples à faire, et pour certaines fonctionnalités, ils existent déjà. Un simple print() dans le terminal pour verifier qu'un bouton a bel et bien fait ce qu'il fallait est suffisant pour tester les interactions UI de Tkinter.

Bugs :

A la fin du développement de notre projet, nous avons découvert un bug assez particulier... après avoir lancé une partie avec x nombre de joueurs, si on relance une autre partie avec y nombre de joueurs, le programme essayera de se limiter (ou forcer) y joueurs, alors qu'il ne s'attend dans le code qu'à x joueurs.
Par exemple, lancer un jeu à 2, le finir, puis lancer un jeu à 3 sans redémarrer le programme fait que le 3e joueur ne joue juste jamais.
Lancer un jeu à 3, le finir, puis lancer un jeu à 2, le programme crash car il s'attend à un troisieme joueur qui, pour le coup, n'existe pas.

Conclusion :

En conclusion, le projet Planning Poker représente une exploration dans le développement d'une application visant à améliorer la planification au sein des équipes agiles. Bien que des défis aient émaillé notre parcours, notamment par la découverte de nouvelles technologies, ces obstacles ont été l'occasion d'acquérir de nouvelles compétences et d'approfondir notre compréhension du développement logiciel. Malgré les imprévus, notre engagement envers la qualité persiste, et ce projet témoigne de cela.