Blanchon Marine, Debret Léane

Rapport de projet

Planning Poker

Professeur référent : Mr Valentin LACHAND-PASCAL

Table des matières

Description du projet :

Nous avons développé une application collaborative de travail de Planning Poker, pour les équipes de développement agile. Elle est surtout utilisée pour les méthodologies Scrum. Elle permet aux équipes de voter sur la complexité de tâches jusqu'à arriver à un accord de façon rapide et efficace.

Fonctionnalités principales :

- Gestion du backlog :

  Importation de fichiers backlog au format JSON.
  Les tâches à estimer sont ensuite affichées, pour permettre une organisation claire des éléments à évaluer.

- Gestion collaborative :

  Chaque joueur peut estimer le coût d'une tâche, en fonction de ses propres critères. Les estimations sont ensuite discutées entre les participants pour atteindre un consensus collectif, renforçant la collaboration et l'implication de chacun. En général, ce sont les personnes qui ont mis les estimations aux extrêmes qui justifient leur choix.

- Plusieurs modes de jeu :

  Unanimité : La tâche en cours reste en jeu tant que tous les joueurs ne se sont pas mis d'accord sur la même estimation.
  Médiane : L'estimation finale est calculée comme la médiane des propositions.
  Moyenne : L'estimation est déterminée par la moyenne des votes.

- Une interface minimaliste et interactive :

  Elle a été conçue pour une prise en main facile, intuitive et rapide.
  Elle permet de naviguer entre les différentes étapes du jeu : choix des tâches ou des modes de jeu.

- Expérience ludique et collaborative :

  Facilite la participation active des membres de l'équipe.
  Rend le processus d'estimation des coûts amusant, tout en restant efficace.

Pour lancer l'application, vous devez vous assurer d'avoir Pygame installé sur votre machine. Toutes les étapes liées aux installations et au lancement de l'application sont détaillées dans le fichier *README.md.*

Choix techniques :

- Langage :

Pour l'application, nous avons choisi d'utiliser le langage Python. En effet, c'est un langage que nous avons beaucoup utilisé pendant notre formation à Lyon 2. De plus, nous avons toutes les deux pris Python en programmation de spécialité et comme nous avions également un projet à faire dans cette matière (et donc dans ce langage), nous avons pensé que ce serait plus simple de nous focaliser sur ce langage pour la majorité du semestre.

- Bibliothèque :

Pour la bibliothèque, nous avons choisi Pygame. Lors de la première séance de TD dédiée à ce projet, nous avons fait des recherches sur Internet (et demandé à ChatGPT…) pour trouver une bibliothèque adaptée au projet. La première réponse de ChatGPT a été Pygame. Nous avons donc appuyé nos recherches sur celle-ci et nous sommes rendues compte qu'en effet, elle était souvent conseillée, elle semblait permettre un développement simple et intuitif d'interface graphique.

Architecture du projet :

Notre application repose sur une architecture simple, mais bien structurée. En voici une description détaillée :

1) Les éléments principaux :

main.py (fichier principal) :
Le fichier *main.py* est le centre de notre projet. C'est le fichier qu'il faudra exécuter pour lancer l'application. Le fichier a plusieurs missions :

- Initialiser Pygame et configure la fenêtre principale.
- Appelle la fonction *run_menu* pour afficher le menu principal
- Contrôler le flux général de l'application.

app/ (dossier principal de l'application) :
Ce dossier contient le code source de l'application. Il regroupe toutes les fonctionnalités principales, classées dans plusieurs fichiers. Nous avons choisi de faire cette séparation afin de garder le fichier *main.py* propre, simple, court et facile à comprendre. Cela permet également de se familiariser plus facilement avec notre code. Voici un bref résumé du rôle de chaque fichier contenu dans ce dossier : *game.py* gère le déroulement du jeu, *menu.py* gère le menu principal de l'application, *settings.py* configure les paramètres avant le début de la partie et utils.py contient des fonctions utiles à l'application (gestion des fichiers et sauvegardes des données JSON, calcul des votes, …).

2) Gestion des données :

backlog_json :

Le dossier regroupe des fichiers, format JSON qui servent de base pour les données à traiter par l'application. Les fichiers contiennent les tâches à estimer. Chaque tâche est représentée par un objet JSON, avec deux attributs principaux : *task* (description de la tâche) et *estimated_cost* (coût estimé de la tâche. Nous mettons -1 par défaut, indiquant qu'elle n'a pas été estimée).
L'utilisateur charge un fichier JSON qui contient le backlog, les tâches sont affichées dans la commande.

requirements.txt :

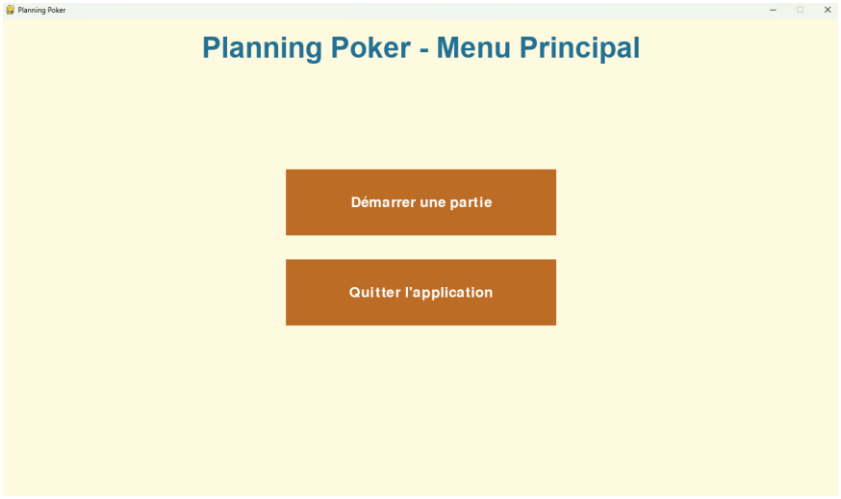
Ce fichier n'est actuellement plus dans notre projet mais nous souhaitions préciser que nous avions pensé à le faire et l'avions généré, mais comme notre projet ne contient que la bibliothèque *Pygame* à installer, l'utilité même de ce fichier ne nous semblait pas importante, donc nous avons décidé de le retirer.

3) Interface utilisateur :

Composants graphiques :

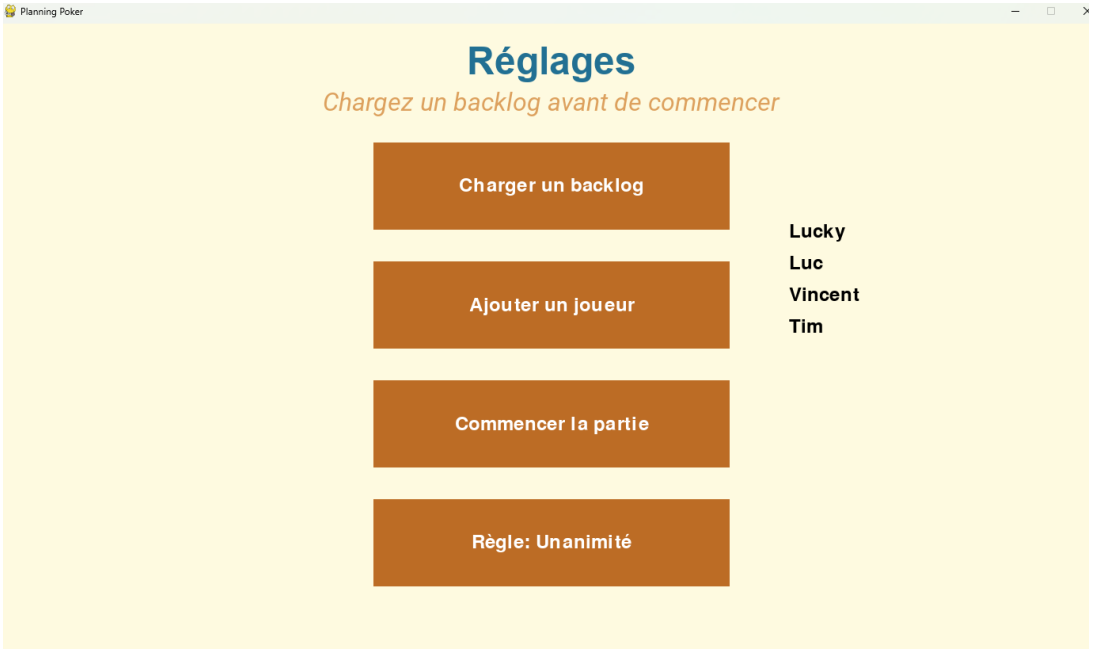
L'interface utilisateur est faite grâce à *Pygame*, une bibliothèque Python. Elle nous a permis de créer une expérience visuelle interactive et engageante. Lors du premier TD consacré à ce projet, nous avons essayé de rechercher une bibliothèque adaptée et celle de Pygame revenait très souvent, c'est pourquoi nous avons décidé de la découvrir et de l'utiliser.

Ecrans principaux :



Référence 1 – Menu principal

- **Le menu principal** : Cette page permet à l'utilisateur de commencer une partie, ou de quitter l'application.



Référence 2 – Ecran de démarrage

- **L'écran de démarrage** : cette page permet à l'utilisateur de charger un backlog (étape obligatoire avant de débuter la partie), de choisir le mode de jeu, d'ajouter les noms des joueurs, puis de commencer la partie.

Référence 3 – Ecran de jeu principal

- **L'écran de jeu principal** : cette page affiche les tâches à estimer, les options de vote, et les résultats des estimations des joueurs dans un second temps.

4) Interaction utilisateur :

L'utilisateur interagit avec l'application grâce à des boutons ou menus graphiques. Les joueurs peuvent voter pour des tâches, sélectionner des modes de jeu, et naviguer entre les écrans.

5) Gestion des ressources :

Le dossier assets contient toutes les ressources nécessaires au bon fonctionnement de l'application : les images pour les cartes à joueur et les polices que nous avons utilisées.

6) Documentation technique :

La documentation générée avec Doxygen est stockée dans le dossier *documentation*. Vous pouvez retrouver le fichier *index.html* dans *documentation > html > index.html*. Le fichier *Doxyfile* contient les paramètres permettant de générer la documentation technique.

7) Tests unitaires :

Le dossier tests contient les scripts des tests unitaires, pour valider les fonctionnalités critiques de l'application : tests sur l'importation et l'affichage du backlog, sur le calcul des résultats pour chaque mode de jeu, sur l'interface utilisateur et vérifier que les interactions fonctionnent correctement.

8) Gestion des versions :

Le fichier *.gitignore* permet d'exclure les fichiers inutiles. Initialement, il a été créé pour supprimer les dossiers venv et s'assurer que ces dossiers ne soient pas créés à nouveau par la suite.

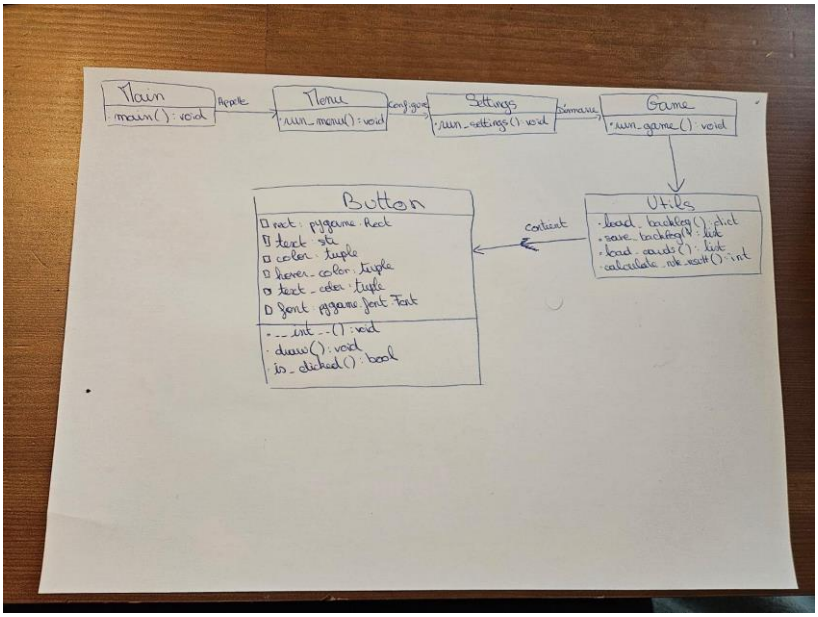
Intégration continue :

L'intégration continue du projet nous a permis d'harmoniser notre développement et de faciliter notre travail collaboratif. Cela nous a permis de maintenir une bonne qualité du code : automatiser l'exécution des tests unitaires et donc vérifier que chaque modification dans le code respecte les exigences fonctionnelles déjà existantes. Cela nous a fait gagner du temps, en détectant rapidement les erreurs après modification du code.

Pour ce projet nous avons mis e place des processus simples mais efficaces pour intégrer cette approche. Nous avons mis en place des tests unitaires avec *unittest*, une bibliothèque standard Python, pour écrire et exécuter des tests sur les fonctionnalités principales. Ces tests unitaires, définis dans le répertoire *tests/* sont exécutés par *unittest* pour valider la fonctionnalité. Si tout fonctionne correctement, les modifications sont validées pour m'intégration, sinon nous sommes alertées afin de corriger le problème rencontré.

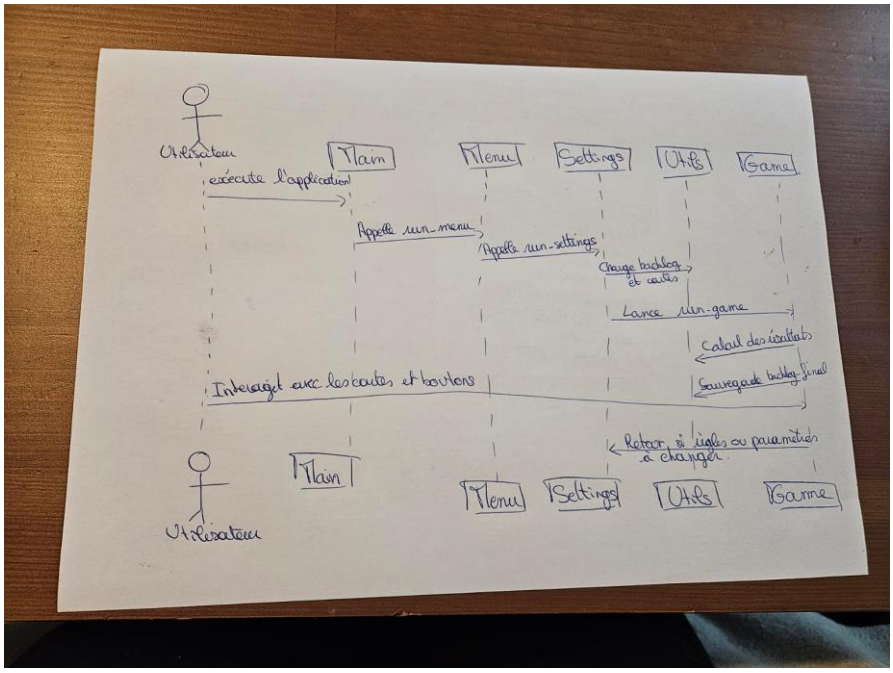
Compréhension globale :

Maintenant, voici quelques illustrations de notre projet. Le but est d'apporter une vision d'ensemble du projet et de se familiariser au mieux avec notre code. Tout d'abord, nous avons fait deux diagrammes. Le premier est un diagramme de classes, simple. Nous avons décidé de le simplifier et clarifier au maximum, le but étant de ne pas surcharger d'informations et d'aller à l'essentiel. C'est pourquoi nous avons décidé de ne pas faire apparaître les paramètres et leur typage. Selon nous, cette partie n'est pas nécessaire à la compréhension simple et globale de la structure de notre code dans le contexte de ce rapport. La classe *Button* est la seule à être explicitement définie en tant que classe dans notre code, mais les autres classes ajoutées au diagramme sont des pseudo-classes qui ont des fonctions, rôle ou relations essentielles dans le code.



Référence 4 - Diagramme de classes

Ensuite, nous avons fait un diagramme de séquence. Selon nous, cette représentation était nécessaire pour être sûr de bien avoir compris les principes et fonctionnements principaux de l'application. Ce diagramme illustre le déroulement global du programme en fonction des classes principales, permettant ainsi d'avoir une vision plus globale du projet :



Référence 5 - Diagramme de séquences

Pour finir, afin de mieux comprendre le fonctionnement de l'application, voici un cas d'utilisation :

CU1 Estimation d'une tâche

Portée : Application de Planning Poker

Niveau : Utilisateur

Acteur principal : Joueur

Déclencheur : Le joueur démarre une partie

Garanties minimales :

- Le backlog des tpaches est chargé
- Au moins deux joueurs sont inscrits

Garanties de succès :

- Toutes les tâches du backlog sont estimées

- Pour chaque tâche, un accord a été mis en place sur leur coût

Scénario nominal :

1. Le joueur charge le backlog
2. Les joueurs s'inscrivent
3. Les joueurs choisissent le mode de jeu
4. Chaque joueur vote en sélectionnant une carte représentant son estimation de coût appropriée à la tâche
5. L'application calcule le consensus, en fonction du mode de jeu qui a été sélectionné
6. Les résultats de vote des joueurs sont affichés
7. Si il n'y a pas d'accord entre les joueurs (mode unanimité), la tâche est remise en jeu pour un nouveau vote
8. Passage à la tâche suivante
9. Fin de la partie si toutes les tâches sont estimées

Extensions :
1.a. Le fichier JSON est mal formaté
2.a. Il y a moins de deux joueurs
6.a. Si au moins un joueur sélectionne la carte pause café, le jeu se termine et laisse donc les joueurs discuter entre eux

Conclusion :

Pour finir, nous avons beaucoup aimé travailler sur ce projet. Ce projet nous a permis de créer une application avec de réels enjeux. Nous avons l'habitude de travailler en binôme donc la répartition des tâches s'est faite rapidement et naturellement. Nous nous sommes prévues des temps de travail ensemble, en présentiel au maximum. Il arrivait que nous avancions chez nous séparément sur certaines fonctionnalités mais nous attendions de nous voir ensuite pour comparer et faire le point ensemble.

Pour ce projet, nous aurions aimé inclure plus de fonctionnalités, notamment l'ajout d'un chronomètre. Mais avec le temps imparti et les erreurs que nous avons dû résoudre (ainsi que la documentation Doxygen), nous avons préféré rester sur une version simple du projet pour nous assurer d'avoir une version fonctionnelle.