Rapport de projet :

Jeu de Planning Poker

Balkis Ferjani

Yasmine Maddouri

Conception Agile des Projets Informatiques

Valentin Lachand-Pascal

20 décembre 2024

Université Lumière Lyon 2

Résumé

Ce rapport présente le développement d'un jeu de **Planning Poker** réalisé pour faciliter l'évaluation collaborative des fonctionnalités du **backlog** d'un produit. L'application permet aux joueurs de voter selon différentes règles de jeu. Un système de sauvegarde et de chargement des parties a été intégré à l'aide de fichiers **JSON**. Le développement a suivi une approche modulaire en utilisant **Python** et **Tkinter** pour l'interface utilisateur. L'intégration continue a été assurée via **GitHub Actions**, avec des tests unitaires et une documentation générée automatiquement par **Doxygen**. Ce projet illustre l'application des bonnes pratiques en développement logiciel, incluant la programmation en binôme, la gestion de projet et l'automatisation des processus.

Mot-clé : Poker Planning, Developpement, Jeu, Python, Intégration Continue.

Table des matières

Table des Figures

1. Contexte General

1.1 Introduction

Le Planning Poker est une méthode d'estimation utilisée en gestion de projet Agile. Ce projet vise à développer une application interactive de Planning Poker, respectant les règles standard du jeu et offrant divers modes de calcul pour l'évaluation des tâches.

1.2 Objectif

- **Faciliter l'estimation collaborative** en respectant les règles du Planning Poker.
- **Proposer plusieurs modes de calcul** tels que l'unanimité, la moyenne et la médiane.
- **Permettre la sauvegarde et la reprise des parties** grâce à des fichiers JSON.
- **Assurer l'intégration continue** avec des tests automatisés et une documentation générée.

1.3 Solution proposé

Le jeu Planning Poker a été développé en Python en respectant une architecture modulaire. Les principales caractéristiques de la solution incluent :

- **Interface utilisateur avec Tkinter** : Une application interactive qui permet de gérer les joueurs, les votes et le backlog.
- **Persistance des données avec JSON** : Sauvegarde et chargement de l'état du jeu pour permettre la reprise des parties.
- **Différents modes de jeu** : Les joueurs peuvent choisir entre les règles "strict", "moyenne", "médiane", "majorité absolue" et "relative".
- **Tests automatisés** : Les tests unitaires ont été réalisés avec `pytest` pour assurer la fiabilité des principales fonctionnalités (gestion des votes, sauvegarde du backlog, etc.).
- **Documentation** : Générée automatiquement avec Doxygen pour assurer une traçabilité et compréhension du code.

2. Modélisation et architecture :

2.1 Choix Techniques

- **Langage :** Python pour sa simplicité et lisibilité, il possède une syntaxe claire et concise, ce qui facilite l'écriture et la maintenance du code.
- **Framework UI :** Tkinter pour l'interface graphique, pour sa simplicité et son intégration native avec Python.
- **Gestion des données :** Fichiers JSON pour la persistance des données ainsi que pour leur format lisible et compatible avec d'autres systèmes.
- **Intégration Continue :** Github, qui permet de travailler simultanément sur le projet, même quand il s'agit de plusieurs développeurs, et chaque modification est tracée grâce à un historique clair des commits.
- **Conception** : UML, permet de visualiser la structure du projet grâce aux diagrammes de classes et d'améliorer la compréhension du projet.
- **Documentation :** Doxygen extrait les commentaires du code pour produire une documentation technique en HTML, PDF ou LaTeX.
- **Tests unitaires :** Pytest permet de faciliter l'écriture, l'exécution et l'organisation des tests unitaires.

2.2 Conception

2.2.1 Diagrammes de cas d'utilisation

a. Graphique

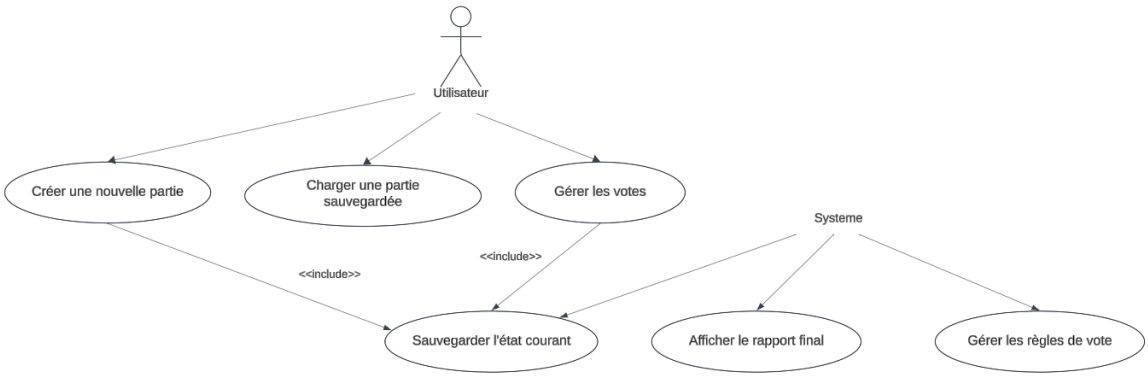

Figure 1: Diagramme de cas d'utilisation

b. Description Textuelle

I. Créer une nouvelle partie

A. **Acteurs** : Utilisateur, Système.

B. **Description** : L'utilisateur initialise une partie en entrant le nombre de joueurs, leurs pseudonymes et les règles.

C. **Scénario principal** :

L'utilisateur sélectionne "Créer une nouvelle partie".

Le système demande les paramètres nécessaires (nombre de joueurs, règles).

Le système crée la partie et sauvegarde automatiquement l'état initial.

II. Charger une partie sauvegardée

A. **Acteurs** : Utilisateur.

B. **Description** : L'utilisateur charge une partie précédemment sauvegardée.

C. **Scénario principal** :

L'utilisateur sélectionne "Charger une partie".

Le système récupère les données sauvegardées et initialise la partie.

L'utilisateur reprend la partie là où elle s'est arrêtée.

III. Gérer les votes

A. **Acteurs** : Utilisateur.

B. **Description** : Les joueurs soumettent leurs votes pour une fonctionnalité.

C. **Scénario principal** :

L'utilisateur entre les votes pour chaque joueur.

Le système valide les votes selon les règles définies.

Si la fonctionnalité est validée, elle est retirée du backlog.

Sinon, un nouveau tour de vote est lancé.

L'état est sauvegardé automatiquement après chaque tour.

IV. Afficher le rapport final

A. **Acteurs** : Système.

B. **Description** : Le système génère et affiche un rapport final après la validation ou le traitement de toutes les fonctionnalités.

C. **Scénario principal** :

Une fois toutes les fonctionnalités votées, le système compile les résultats.

Le rapport est affiché automatiquement sans intervention de l'utilisateur.

V. 5. Sauvegarder l’état

A. **Acteurs** : Système.

B. **Description** : Le système sauvegarde automatiquement l’état après chaque action critique (vote, création de partie, etc.).

C. **Scénario principal** :

Après chaque modification (nouvelle partie, vote), le système sauvegarde l'état dans un fichier JSON.

VI. 6. Gérer les règles de vote

A. **Acteurs** : Système.

B. **Description** : Le système applique les règles de vote sélectionnées pour valider ou non les fonctionnalités.

C. **Scénario principal** :

Le système applique les règles choisies (unanimité, moyenne, majorité, etc.) pendant le traitement des votes.

2.2.2 Diagrammes de Classes

Le modèle est composé des classes principales suivantes :

- **Application** : Coordonne les interactions entre les joueurs et le système.
- **Player** : Représente chaque participant.
- **Game** : Gère le déroulement d'une session de Planning Poker.
- **Gestionnaire Backlog** : Gère les fonctionnalités et les votes.

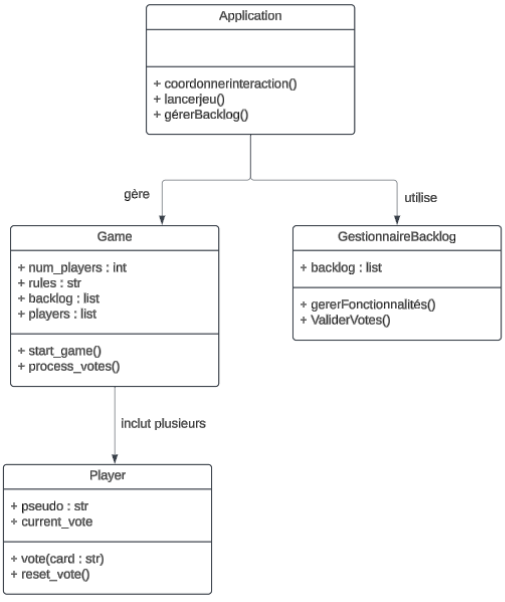

Figure 2: Diagramme de classes

3. Spécification des besoins

3.1 Besoins fonctionnels

1) **Gestion des joueurs :** Ajout de pseudos et saisie du nombre de joueurs.
2) **Modes de jeu :** Unanimité, moyenne, médiane, majorité absolue, majorité relative.
3) **Système de vote :**
    a) Chaque joueur sélectionne sa carte de vote.
    b) Les votes sont analysés en fonction des règles sélectionnées.
    c) Un nouveau tour est lancé si une fonctionnalité n'est pas validée.
4) **Validation des tâches :** Selon les règles sélectionnées.
5) **Sauvegarde et reprise :**
    a) Sauvegarde automatique après chaque action (vote, création de partie).
    b) Chargement d'une partie depuis un fichier JSON existant.
6) **Rapport Final :**
    a) Génération automatique du rapport à la fin de la partie.
    b) Affichage des fonctionnalités validées et des estimations collectées.
7) **Gestion des erreurs** :
    a) Validation des entrées utilisateur (pseudonymes, nombre de joueurs, règles de vote).
8) **Interface utilisateur :**
    a) Navigation intuitive dans le menu principal.
    b) Retour visuel après chaque action (vote soumis, sauvegarde réussie, etc.).

3.2 Besoins non fonctionnels

Les besoins non fonctionnels décrivent les contraintes et qualités du système qui ne sont pas directement liées aux fonctionnalités principales.

1) Performances : notre système répond rapidement aux actions de l'utilisateur, et le rapport final est généré et affiché immédiatement après la validation des dernières fonctionnalités.
2) Fiabilité : Notre système traite correctement les erreurs de saisie (par exemple, règles ou pseudonymes invalides).

3) Accessibilité : L'application est facile à utiliser, avec une interface utilisateur intuitive et des instructions claires. Les boutons et champs de saisie sont bien espacés et facilement accessibles.
4) Maintenabilité : Le code est organisé de manière modulaire, en suivant des pratiques de programmation claires pour faciliter les mises à jour et corrections futures. La documentation générée automatiquement avec Doxygen inclut des détails sur chaque classe et méthode.
5) Intégration Continue : Les pipelines d'intégration continue automatisent les tests et la génération de documentation à chaque mise à jour du code.

4. Implémentation

4.1. Menu Principal

Figure 3: Menu Principale de notre Application

1) Saisie du nombre de joueurs
    a) L'utilisateur est invité à entrer le nombre total de joueurs participant à la partie.
    b) Le système vérifie que la valeur saisie est valide (un entier supérieur à 1).
    c) En cas de saisie invalide, un message d'erreur est affiché, et l'utilisateur est invité à réessayer.

Figure 4: Fonctionnalité Saisit du nombre des joueurs et des règles du jeu

2) Attribution de pseudos
    a) Une fois le nombre de joueurs validé, l'application demande les pseudonymes pour chaque joueur.
    b) Les pseudonymes sont enregistrés dans une liste pour être associés aux joueurs lors des votes.

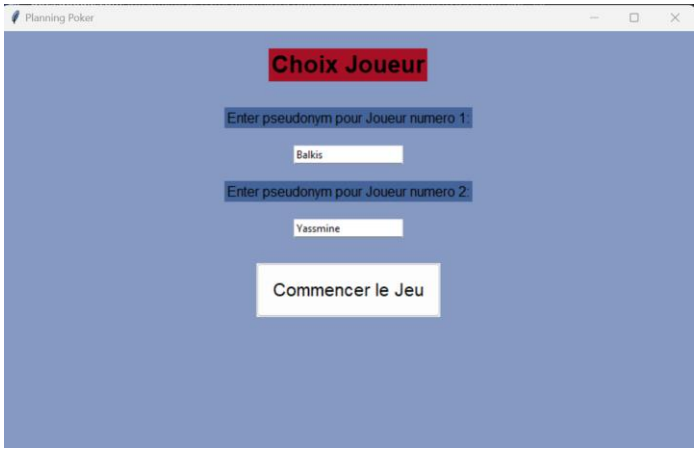

Figure 5: Fonctionnalité Définition des Pseudonyme des joueurs

3) Choix du mode de jeu
    a) L'utilisateur sélectionne l'un des modes de jeu disponibles : strict (unanimité), moyenne, médiane, majorité absolue, ou majorité relative.

b) Le système vérifie que le mode choisi est valide avant de poursuivre.

4) Chargement d'un fichier JSON existant

a) Une option permet de charger un fichier JSON contenant un backlog de fonctionnalités sauvegardées précédemment.

b) Le système vérifie la validité du fichier (format, données correctes) avant de charger les informations.

c) En cas d'erreur (par exemple, fichier non trouvé ou corrompu), un message est affiché, et l'utilisateur peut réessayer.

4.2. Système de Vote

1) **Mode Strict (Unanimité)** : Les joueurs votent jusqu'à ce qu'un consensus soit atteint pour valider une fonctionnalité.
2) **Mode Moyenne** : Après un premier tour de vote, le système calcule la moyenne des votes pour valider la fonctionnalité.
3) **Mode Médiane** : Le système utilise la valeur médiane des votes pour valider une fonctionnalité.
4) **Mode Majorité Absolue** : La fonctionnalité est validée si plus de la moitié des joueurs votent pour la même valeur.
5) **Mode Majorité Relative** : La fonctionnalité est validée si une valeur obtient plus de votes que les autres.

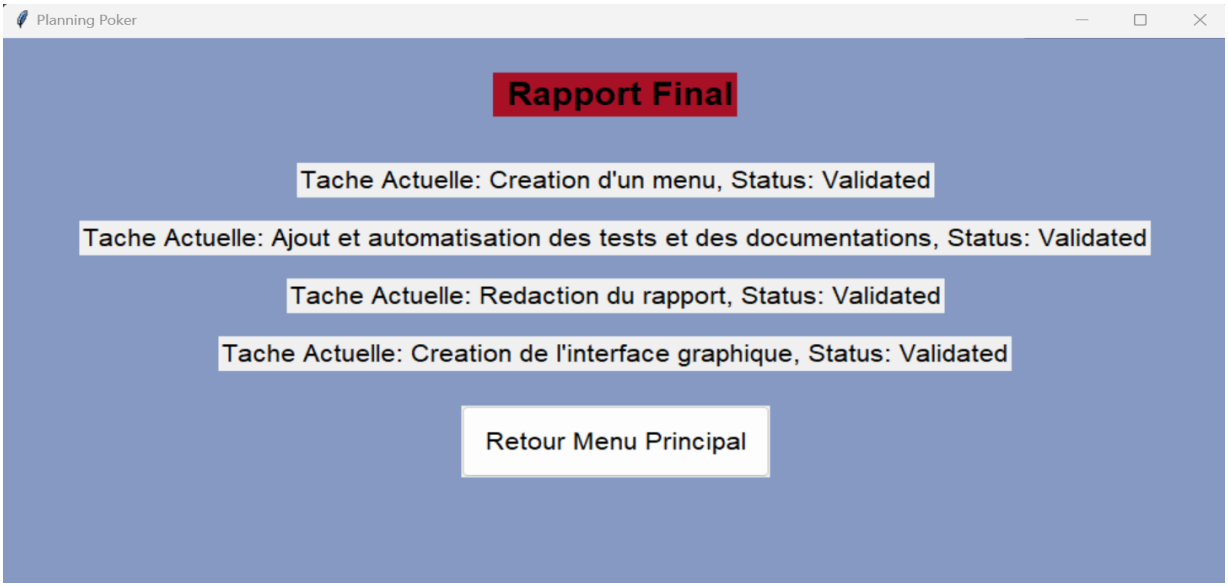

Figure 6: Fonctionnalité Sauvegarde et Affichage du rapport finale

4.3. Sauvegarde et Chargement

1) Sauvegarde automatique après chaque vote
2) Chargement à partir d'un fichier JSON existant

5. Intégration Continue

5.1. Automatisation des Tests :

Nous avons mis en place des tests unitaires avec **Pytest** pour garantir la fiabilité du code. Chaque module dispose de tests associés, couvrant les fonctionnalités principales (gestion du backlog, système de vote, etc.).

5.2. Génération de Documentation :

Nous avons utilisé **Doxygen** pour générer automatiquement une documentation complète du projet. Cette documentation inclut les descriptions des classes, des méthodes et des interactions principales.

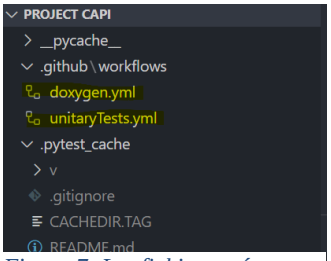

Figure 7: Les fichiers créent pour la documentation et testes unitaires

5.3. Pipeline CI :

Nous avons configuré **GitHub Actions** pour exécuter automatiquement :

- Les tests unitaires à chaque commit.
- La génération et le déploiement de la documentation.

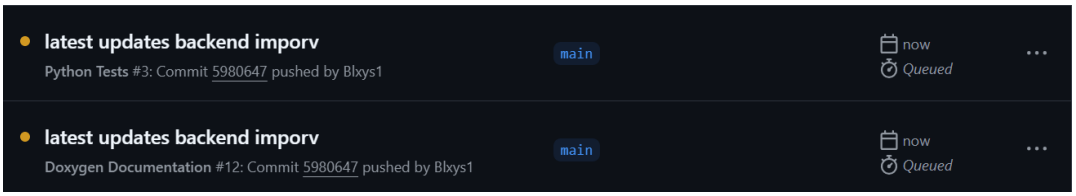

Figure 8: Le process de l'exécution du Github Actions

Après chaque opération “push” dans notre branche principale on a :

Figure 9: Exécution validé des actions après un PUSH

5.4. Suivi du Projet

Le suivi des tâches a été effectué à l'aide de **Github Project**, assurant la traçabilité des issues, des commits et des progrès réalisés.

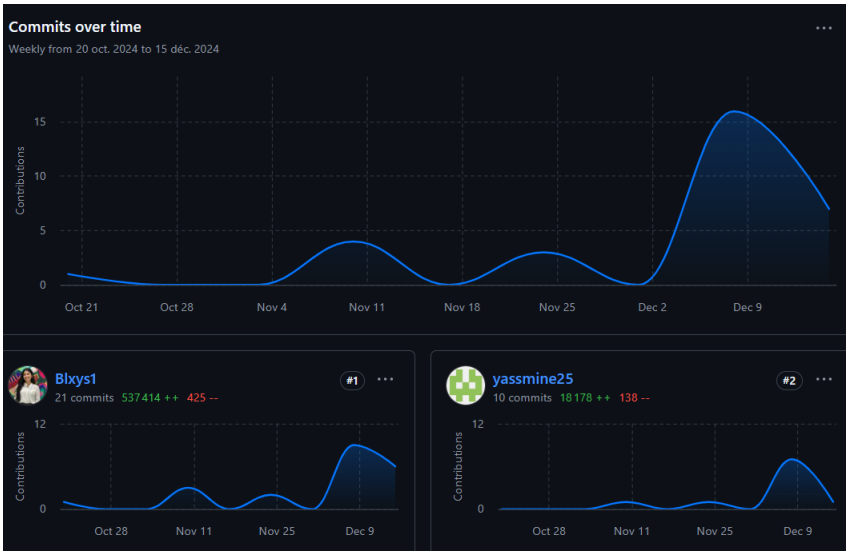

Figure 10: Tableau de bord extrait de Github, qui représente le suivi du projet

Conclusion

Conclusion générale :

Le développement de cette application de Planning Poker a permis d'explorer des concepts avancés de développement logiciel, notamment la programmation en binôme, la gestion de projet Agile et l'intégration continue. L'application offre une expérience interactive et conforme aux règles de jeu tout en restant extensible pour de futures améliorations.

Améliorations Potentielles :

Cependant, certaines fonctionnalités prévues dans le backend n'ont pas encore été intégrées à l'interface utilisateur. Cela ouvre des perspectives intéressantes pour de futures améliorations, notamment :

- **Gestion de la Carte Café** : Enregistrer automatiquement l'état du backlog lorsque tous les joueurs utilisent cette carte.
- **Premier Tour Basé sur l'unanimité** : Appliquer la règle de l'unanimité au premier tour avant de basculer vers d'autres méthodes de calcul.
- **Interface Utilisateur Modernisée** : Utiliser des Framework plus avancés comme PyQT ou Kivy pour améliorer l'expérience utilisateur.

Ces améliorations futures permettront de rendre l'application encore plus robuste, ergonomique et adaptée aux besoins des équipes de développement travaillant en méthode Agile.