Application de Planning Poker

Martin robin
Mesbahi Lysa

Le planning poker est une technique d'estimation utilisée dans le domaine de la gestion de projet, particulièrement dans le développement logiciel agile. Elle vise à estimer la complexité relative des tâches à accomplir. Cette méthode implique généralement l'équipe de développement, le product owner et d'autres parties prenantes dans un processus de discussion structuré pour parvenir à un consensus sur la complexité relative à des éléments du Backlog (liste de fonctionnalités ou de tâches à réaliser).

2. Objectifs du Projet

L'objectif de notre application est de fournir une plateforme efficace pour les joueurs participant à des sessions de planning poker, tout en respectant les règles établies au cours de notre formation. Cette application est utilisée localement, permettant aux joueurs de sélectionner leurs cartes tour à tour. Parmi les objectifs de l'application :

1. **Configuration des Paramètres :** Les utilisateurs peuvent personnaliser leur expérience en décidant du nombre de joueurs et en attribuant un pseudonyme à chacun d'eux via un menu convivial.
2. **Choix des Règles :** Le menu propose également une sélection de règles de planning poker, parmi lesquelles les joueurs peuvent choisir, telles que des règles strictes, moyennes, médianes, etc.
3. **Gestion du Backlog :** Les utilisateurs peuvent entrer une liste de fonctionnalités (backlog) au format JSON, offrant une structure souple pour représenter ces informations.
4. **Vote et Validation :** Chaque joueur vote pour la validité d'une fonctionnalité, conformément aux règles sélectionnées. En cas de désaccord, un nouveau vote peut être initié.
5. **Enregistrement du Backlog Validé :** Une fois que toutes les fonctionnalités du backlog sont validées, l'application enregistre un fichier JSON comprenant, pour chaque fonctionnalité, la difficulté estimée par l'équipe.

3. Choix du Langage

Notre choix de langage de programmation pour le développement de l'application de planning poker s'est orienté vers Python, accompagné des bibliothèques Tkinter et customTkinter. Cette décision stratégique repose sur plusieurs critères clés qui garantissent la robustesse,

l'efficacité et la convivialité de notre application, tout en facilitant le processus de développement.

1. **Rapidité de Développement :** Python est réputé pour sa productivité élevée. Les développeurs peuvent créer rapidement des prototypes et itérer plus efficacement, ce qui est un avantage dans un environnement agile où les changements peuvent être fréquents.
2. **Large Écosystème :** Python dispose d'un écosystème riche en bibliothèques et modules, ce qui facilite l'intégration de fonctionnalités supplémentaires. Tkinter est une bibliothèque standard de Python pour l'interface graphique, et son utilisation est intuitive.
3. **Tkinter pour l'Interface Graphique :** Tkinter est intégré à Python, offrant une solution native pour la création d'interfaces graphiques. C'est un choix judicieux pour des applications simples à intermédiaires, comme dans le cas du planning poker, où une interface utilisateur propre et fonctionnelle est cruciale.
4. **customTkinter pour Personnalisation :** L'utilisation de customTkinter suggère que vous avez recherché des moyens d'améliorer et de personnaliser l'aspect visuel de votre application au-delà des fonctionnalités standard de Tkinter. Cela montre une attention particulière à l'expérience utilisateur et à la personnalisation de l'interface graphique.
5. **Intégration Continue avec Python :** L'intégration continue est généralement bien supportée pour les projets Python. les intégrations directes avec des plateformes comme GitHub Actions peuvent être facilement mis en place pour automatiser les tests et la génération de documentation.

En résumé, le choix de Python avec Tkinter et customTkinter semble être une décision judicieuse en tenant compte de la simplicité, de la productivité, de la portabilité, et de la personnalisation nécessaires pour le développement de notre application.

4. Architecture Générale

1. **Fonctions principales :** Trois fenêtres principales ont été mise en place dans l’application : **Menu**, **Creation_Partie**, et **Partie**. Les fonctions **Changement_Fenetre** permettent de passer d'une fenêtre à une autre en détruisant les widgets de la fenêtre actuelle.
2. **Chargement de tâches :** Vous permettez le chargement d'un fichier JSON pour définir la liste des tâches (backlog). La fonction **Verification_fichier** vérifie la validité du fichier JSON.
3. **Création de partie :** La fenêtre **Creation_Partie** permet à l'utilisateur de définir le nom de la partie, charger un fichier de tâches, choisir le mode de jeu (strict ou moyenne), ajouter des joueurs, etc.Vous avez utilisé des Entry, Buttons, Labels, et une ScrollableFrame pour la gestion des joueurs.

4. **Partie en cours :** La fenêtre **Partie** gère le déroulement du planning poker avec l'affichage de la tâche en cours, le choix des cartes par les joueurs, et la gestion du consensus. Les cartes sont représentées par des boutons.
5. **Gestion des règles de consensus :** il y’a deux modes de consensus, le mode strict et le mode moyenne.Le consensus est atteint selon les règles du mode choisi.
6. **Mise à jour du fichier JSON :** Le fichier JSON est mis à jour au fur et à mesure que les joueurs font leurs choix.
7. **Interface utilisateur :** L'interface utilisateur est interactive avec des boutons, des labels, des entrées, etc. Nous avons utilisé **customtkinter** pour personnaliser l'apparence de certains éléments.

5. Design Patterns :

Le choix des design patterns dépend largement de la structure de l’application, de ses fonctionnalités et des problèmes spécifiques que nous cherchons à résoudre. Cependant, voici trois design patterns couramment utilisés :

1. **Modèle Vue Contrôleur (MVC)** :
    - **Description :** Le modèle MVC sépare les composants d'une application en trois parties principales : le Modèle (qui représente les données et la logique métier), la Vue (qui gère l'interface utilisateur) et le Contrôleur (qui gère les interactions entre la Vue et le Modèle).
    - **Intégration :** l’application est structurée de manière à ce que les différentes parties suivent ce modèle.
2. **Observateur (Observer)** :
    - **Description :** Le modèle Observateur permet à un objet (appelé le sujet) de maintenir une liste de ses dépendants (les observateurs) qui sont notifiés automatiquement de tout changement d'état, généralement en appelant l'une de leurs méthodes.
    - **Intégration :** Il est utilisé pour notifier automatiquement les parties de l’interface utilisateur lorsque l'état de l’application change. Par exemple, si une tâche est marquée comme terminée, les observateurs (comme les composants d'interface utilisateur) sont notifiés pour mettre à jour leur affichage.
3. **Stratégie (Strategy)** :
    - **Description :** Le modèle de stratégie définit une famille d'algorithmes, les encapsule et les rend interchangeables. Il permet au client de choisir l'algorithme à utiliser à partir d'une famille et de le paramétrer indépendamment des clients qui les utilisent.
    - **Intégration :** le modèle de stratégie est utilisé pour gérer différentes manières d'effectuer certaines actions dans votre application. Par exemple, pour calculer une note, on utilise une famille d'algorithmes de calcul de note interchangeable.

6. Modes de Jeu

Pour garantir une expérience utilisateur diversifiée et adaptable à différentes préférences, deux modes de jeu distincts ont été intégrés dans l'application.

- **Mode Strict** : Dans ce mode, les membres de l'équipe discutent de la tâche à estimer. Ensuite chacun donne sa propre estimation pour représenter l'effort requis. Si les estimations diffèrent, les membres discutent des raisons de leurs choix. L'objectif est d'atteindre un consensus où tous les membres de l'équipe sont d'accord. Cela encourage la communication et la compréhension collective de la complexité de la tâche.

- **Mode Moyenne** : Dans ce mode, les membres de l'équipe donnent leur estimation individuelle de l'effort requis pour la tâche. Ensuite, les estimations sont moyennées pour obtenir une estimation finale. Cette méthode est utile lorsque les estimations varient et que la discussion ne parvient pas à un consensus unanime. Elle permet de prendre en compte les différents points de vue de l'équipe sans forcément parvenir à un accord.

7. Fonctionnalités Additionnelles :

En vue d'améliorer l'expérience utilisateur et d'offrir des fonctionnalités enrichies, notre application comporte plusieurs options :

- **Charger une ancienne partie** : les utilisateurs peuvent reprendre une partie qu'ils avaient déjà commencée.
- **Fichier JSON :** les utilisateurs doivent obligatoirement exporter un fichier JSON au début d'une partie sinon une erreur s'affichera.

Ces fonctionnalités additionnelles visent à rendre notre application non seulement fonctionnelle mais également conviviale, offrant aux utilisateurs un environnement de planification intuitif et adapté à leurs préférences individuelles.

8. Intégration Continue :

Pour mettre en place des mécanismes d'intégration continue dans notre projet, nous avons utilisé plusieurs outils et services afin d'automatiser les tests et de générer la documentation, contribuant ainsi à la qualité globale du projet. Voici les étapes détaillées de notre approche :

1. **Gestion du Code Source avec Git :**
    - Tout le code source du projet est hébergé sur une plateforme de gestion de code source, comme GitHub. Cela nous permet de suivre les modifications, de collaborer efficacement et d'avoir une vision claire de l'évolution du projet.
2. **Automatisation des Tests avec Pytest :**

- Nous avons écrit des tests unitaires exhaustifs à l'aide du framework Pytest. Ces tests couvrent différentes parties de l'application, y compris les fonctionnalités clés et les cas limites. Les tests sont conçus pour s'exécuter automatiquement à chaque modification du code source.

3. **Intégration Continue avec GitHub Actions :**
    - GitHub Actions a été configuré pour automatiser le processus d'intégration continue. À chaque "push" ou "pull request" sur la branche principale, GitHub Actions déclenche une série de workflows, comprenant notamment l'exécution des tests unitaires. Cela garantit que toute modification du code est immédiatement testée, réduisant ainsi les risques d'introduction d'erreurs.
4. **Génération Automatique de Documentation avec Doxygen :**
    - Nous avons intégré Doxygen pour générer automatiquement la documentation à partir du code source. Un fichier de configuration Doxyfile spécifique au projet a été créé pour définir les paramètres de génération. Cette documentation est mise à jour à chaque modification du code source, assurant ainsi qu'elle reste toujours à jour.
5. **Publication de la Documentation :**
    - La documentation générée est publiée sur une plateforme dédiée, ce qui permet aux membres de l'équipe et aux utilisateurs d'accéder facilement à une documentation toujours à jour. Cela contribue à la compréhension du code et facilite la collaboration.
6. **Notifications Automatiques en Cas d'Échec de Tests :**
    - Des notifications automatiques sont configurées pour informer l'équipe en cas d'échec de tests. Cela permet une réaction rapide pour résoudre les problèmes potentiels dès qu'ils sont détectés.

L'intégration continue, avec son processus d'automatisation des tests et de génération de documentation, a grandement amélioré la qualité globale du projet. Les avantages incluent la détection précoce des erreurs, la garantie que la documentation est toujours à jour, et une augmentation de la confiance dans le code déployé. Ce processus renforce également la collaboration au sein de l'équipe en fournissant des retours rapides sur les modifications apportées au code.

9. Conclusion

En conclusion, ce projet représente une étape significative dans le développement d'une application robuste et fonctionnelle. À travers l'utilisation de technologies modernes, la mise en œuvre de bonnes pratiques de développement, et l'intégration de mécanismes d'intégration continue, nous avons réussi à créer une application fiable et de haute qualité.

L'adoption de méthodes telles que l'intégration continue a grandement contribué à la stabilité du projet en identifiant rapidement les erreurs potentielles et en assurant que la

documentation reste à jour. Les tests unitaires systématiques ont renforcé la confiance dans la robustesse de notre code.

Les fonctionnalités principales, notamment le mode de jeu strict et moyenne, ainsi que la gestion efficace des tâches, sont des jalons importants dans la réalisation des objectifs du projet. La documentation générée par Doxygen offre un guide complet pour les développeurs et les utilisateurs, facilitant l'extension future de l'application.

10. Instructions pour l'utilisation du Projet :

1. **Installation des Dépendances :** Assurez-vous d'avoir installé les dépendances nécessaires avant de lancer l'application.

   Vérifiez également que vous avez Python, Tkinter, Doxygen et Pytest installés sur votre machine.

2. **Exécution de l'Application :** Après avoir installé les dépendances, lancez l'application en exécutant le script Python principal. Vous pouvez le faire avec la commande suivante : *Python CAPI_MARTIN_MESBAHI.py*

   Cela ouvrira l'interface graphique de l'application.

3. **Configuration de Doxygen :** Si vous souhaitez mettre à jour la documentation à l'aide de Doxygen, assurez-vous d'avoir Doxygen installé sur votre système. Ensuite, suivez les étapes suivantes :

   - Ouvrez une fenêtre de terminal.
   - Accédez au répertoire conf.
   - Exécutez la commande suivante pour générer la documentation : *doxygen Doxyfile*

   La documentation générée sera disponible dans le dossier **docs**.

4. **Exécution des Tests Unitaires :** Pour exécuter les tests unitaires, utilisez la commande suivante : *pytest test_main.py*

Assurez-vous que toutes les dépendances pour les tests unitaires sont installées.

5. **Notes Importantes :**

   - Assurez-vous que le fichier de tâches est correctement chargé avant de commencer une partie.
   - Veillez à suivre les indications dans l'interface graphique pour naviguer entre les différentes fonctionnalités de l'application.
   - Pour quitter l'application, utilisez les options prévues dans l'interface graphique.

12. Références

[...]

13. Annexes

[...]

14. Conclusion Générale

[...]