PLANNING POKER

Valentin Lachand-Pascal

20 DECEMBRE 2024

BESSAD TAREK
LUCAS JOURNOUD

Table des matières

1. Introduction

1.1 Objectifs du projet

Le projet Planning Poker vise à offrir une plateforme interactive où les utilisateurs peuvent estimer collectivement la complexité des tâches d'un backlog.

Le flow d'estimation repose sur les règles du planning poker, un système de gamification des pratiques agiles, où les joueurs discutent et vote pour la complexité de chacune des tâches lors de tour de jeu afin d'aligner la vision des membres d'une équipe sur sa complexité. Lorsqu'une tâche fédère l'équipe, elle est automatiquement estimée sur le consensus des participants. Un vote unanime clôture le flow d'estimation de l'ensemble des tâches. Différent mode de jeu existe pouvant modifier légèrement les règles, comme le mode moyenne qui comme son nom l'indique, fait la moyenne de l'estimation de chaque joueur.

1.2 Présentation générale de l'application

L'application Planning Poker, développée en C++ avec Qt, offre une interface graphique intuitive et conviviale pour faciliter l'interaction entre les utilisateurs. Elle s'articule autour des fonctionnalités suivantes :

- ✓ Charger et enregistrer des backlogs au format JSON : Les tâches à estimer sont organisées dans un fichier JSON qui peut être chargé ou sauvegardé directement via l'application, garantissant une gestion simple et structurée des données.

- ✓ Créer une nouvelle session ou reprendre une session existante : Les utilisateurs peuvent démarrer une nouvelle session ou reprendre une session interrompue grâce à la sauvegarde de l'état dans le fichier JSON.

- ✓ Utiliser des modes de jeu différents : Deux modes principaux sont proposés, le mode strict (basé sur l'unanimité) et le mode moyenne (basé sur une estimation moyenne après plusieurs tours).

- ✓ Gérer les pauses et sauvegarder les sessions grâce à la carte "Café" : Une fonctionnalité unique qui permet de mettre en pause la session lorsqu'un joueur sélectionne la carte "Café", avec sauvegarde automatique de l'état de progression.

Cette conception vise à offrir une expérience utilisateur optimale tout en répondant aux besoins des méthodologies agiles.

1.3 Fonctionnalités principales

L'application propose plusieurs fonctionnalités clés, notamment :

- **Gestion complète des sessions :** Permet la création de nouvelles sessions, la reprise des sessions en pause et la sauvegarde automatique des données de progression.
- **Modes de jeu personnalisés** : Inclut un mode strict (obligation d'unanimité pour valider une estimation) et un mode moyenne (convergence des estimations via une moyenne après discussion).
- **Interface graphique intuitive** : Conçue avec Qt, l'interface offre une navigation fluide et une visualisation claire des informations, incluant des cartes personnalisées pour les estimations.
- **Gestion avancée des états de session** : Inclut des fonctionnalités comme la pause avec la carte "Café", permettant une flexibilité accrue lors des sessions collaboratives.
- **Sauvegarde et reprise des sessions en fichier JSON** : Facilite le stockage et le partage des données entre plusieurs utilisateurs ou sessions.

Ces fonctionnalités combinées assurent une expérience utilisateur robuste et alignée sur les besoins des équipes de développement agile.

2. Choix techniques

2.1 Langage de programmation : C++

Le choix du C++ comme langage de programmation utilisé s'explique par plusieurs raisons. Tout d'abord, ses performances élevées en font un langage idéal pour des applications interactives telles que le Planning Poker. La gestion fine de la mémoire est un atout majeur qui garantit la stabilité de l'application, en particulier lorsque des structures de données complexes comme des fichiers JSON doivent être manipulées. Le C++ est particulièrement performant dans des environnements où les ressources doivent être optimisées, comme les calculs fréquents et l'utilisation de fichiers de grande taille.

Le second aspect à prendre en compte (et le plus important) est que nous devions trouver un langage que nous maitrisions tout 2. Après plusieurs discussions, il est vite apparu que c'est le C++ qui remplirait ce rôle. En effet, nous avions tout 2 déjà réaliser plusieurs projets dans ce langage, et nos formations respectives incluaient du C++. En tant qu'équipe de deux développeurs ayant une expérience préalable significative dans ce langage, nous avons pu rapidement réaliser les bases du projet et prototyper efficacement les principales fonctionnalités. Cela nous a permis de nous concentrer sur les aspects plus complexes de l'application, tels que la gestion des interactions utilisateur et l'intégration de fichiers JSON. Ce choix a aussi favorisé une meilleure collaboration, car chaque membre du binôme maîtrisait les bases du langage, facilitant ainsi les discussions et les résolutions de problèmes.

2.2 Framework : Qt

Qt est un choix évident pour ce projet, grâce à sa capacité à fournir des outils puissants pour créer des interfaces graphiques multiplateformes. Ses avantages incluent :

- **Signaux et slots** : Ce système évènementiel simplifie la communication entre les différentes parties de l'application (par exemple, entre les boutons de l'interface et les fonctions associées). Ce modèle réduit les dépendances directes et améliore la maintenabilité.

- **Gestion native des fichiers JSON** : Les classes comme QJsonDocument ou QJsonObject permettent une lecture et une écriture fluide des fichiers JSON sans dépendances supplémentaires. Ces outils intégrés ont accéléré le développement des fonctionnalités liées à la gestion des données.

- **Style et personnalisation** : Les outils CSS de Qt offrent une flexibilité pour concevoir une interface moderne et intuitive. La possibilité de définir des styles précis a permis de rendre l'interface visuellement attrayante et professionnelle.

2.3 Difficulté

- **Réseaux** : Certains inconvénients sont apparus quant aux choix de ces solutions. Pour commencer, faire une application en réseau est beaucoup plus compliqué sur une application C++ que sur un projet en PHP par exemple. En effet, bien qu'il existe des librairies pour le réseau en C++ ainsi que des classes Qt permettant de faire ça, il reste très laborieux et complexe d'y mettre en place. Pour cette raison nous avons pris la décision de faire l'application qu'en local.

- **Déploiement** : La notation faisant état du fait que l'enseignant devait pouvoir compiler le projet sur sa machine, le choix d'une configuration si complexe n'était peut-être pas très adapté. En effet nous utilisons Qt 6.7.3 sous Visual Studio 2022 avec l'extension Qt VS Tools ainsi que le compilateur MSVC 2022. Durant toutes ces étapes, un problème peut survenir empêchant l'enseignant de pouvoir compiler sur sa machine.

2.4 Structure orientée objet

L'application suit une architecture orientée objet pour garantir clarté et modularité. Les modules principaux sont :

- **PlanningPoker** : Classe centrale qui orchestre les interactions entre les différents composants.
- **EstimationDialog** : Fenêtre de dialogue permettant aux joueurs de saisir leurs estimations.
- **JsonViewerDialog** : Interface pour visualiser et manipuler les fichiers JSON chargés.

Cette structure permet de diviser efficacement les responsabilités et de faciliter les tests et modifications en structurant efficacement le code. Par exemple, l'ajout d'un nouveau mode de jeu ou d'un format de fichier serait simple à implémenter dans cette architecture. Chaque module est indépendant et communique avec les autres via des interfaces bien définies, réduisant ainsi les dépendances croisées. Un axe d'amélioration pourrait-être la mise en place de l'architecture MVC (Modèle Vue Controlleur) afin de séparer encore plus efficacement notre code.

2.5 Gestion des données avec JSON

Les fichiers JSON ont été choisis pour leur simplicité, leur lisibilité et surtout le fait que c'était une des principales consignes de ce projet. Ils permettent de stocker efficacement les données importantes, comme :

- La liste des tâches à estimer (backlog) ainsi que leur estimation.
- Les joueurs.
- L'état actuel d'une session (terminée ou non).
- Le mode de jeu (actuellement « Strict » ou « Average »)

L'utilisation des classes natives de Qt (e.g., QJsonDocument, QJsonObject) simplifie leur manipulation tout en garantissant la validité des données. Une validation systématique est effectuée lors du chargement pour éviter les erreurs d'exécution. Nous avons également choisi JSON pour sa portabilité, permettant une compatibilité avec d'autres applications ou langages si nécessaire.

2.6 Modes de jeu et règles

Deux modes de jeu ont été implémentés :

- **Mode strict** : Requiert une unanimité des joueurs pour valider une estimation. Ce mode favorise la discussion et permet une meilleure compréhension des divergences, encourageant une prise de décision collaborative.

- **Mode moyenne** : Calcul d'une estimation moyenne après le premier tour, utile lorsque l'unanimité n'est pas atteinte. Ce mode offre une alternative rapide tout en respectant les méthodologies agiles.

Ces modes s'alignent sur les besoins des méthodologies agiles tout en étant facilement extensibles. Par exemple, un mode basé sur la médiane ou une majorité relative pourrait être ajouté ultérieurement.

2.7 Gestion des sessions

La possibilité de sauvegarder et reprendre des sessions a été une priorité. Cette fonctionnalité repose sur :

Sauvegarde automatique : L'état des sessions est écrit dans un fichier JSON à chaque modification importante notamment au lancement d'une session, a chaque fin d'estimation de tâches et a l'appuie sur la carte café.

Reprise intuitive : Lorsqu'un fichier JSON est rechargé, l'application restaure tous les éléments essentiels (joueurs, backlog, mode de jeu). Cela garantit une continuité fluide pour les utilisateurs, même en cas d'interruption imprévue. Si le JSON est détecter comme vide, alors les joueurs pourront commencer une partie. A l'inverse s'il est déjà complet, ils ne pourront pas lancer de session, et ne pourrons que visualiser ses résultats. Si le JSON est incomplet (appuie sur la carte café, ou fermeture de l'application), alors l'application proposera au joueur de reprendre la partie ou elle s'était arrêtée.

2.8 Tests unitaires et intégration continue

Bien que prévus, les tests unitaires n'ont pas pu être pleinement réalisés en raison de plusieurs contraintes :

- **Difficulté de test avec Qt** : Qt propose une classe QTest permettant d'effectuer des tests unitaires. Malheureusement, ces derniers se sont avérés difficiles à mettre en place. De plus, l'utilisation de QTest nécessite l'installation du module QTest. Nous n'utilisons ni CMake ni qmake mais QtMsBuild. L'ajout de module supplémentaire bien que réalisable, aurait été assez complexe pour l'enseignant, en comptant le fait que l'enseignant doit pouvoir réussir à compiler l'application de son côté. Sans QTest nous n'utilisons que des modules de base de Qt (inclus dans l'installation) ce qui facilitera la configuration du projet nous l'espérons par l'enseignant.
- Ces problèmes restent toutefois solvables et représentent une opportunité d'amélioration pour de futures versions du projet.

2.9 Interface utilisateur

L'interface graphique, conçue avec Qt, est moderne et facile à utiliser. Les principaux points forts sont:

- **Personnalisation visuelle** : Les cartes et boutons utilisent des styles CSS pour garantir une esthétique attrayante.

- **Navigation intuitive** : Les éléments importants (sélection de fichier, estimation, etc.) sont clairement organisés.

Cependant, quelques points peuvent être améliorés, comme :

- **Gestion des messages d'erreur** : Les notifications peuvent être enrichies pour guider davantage les utilisateurs en cas de problème (par exemple, fichiers JSON invalides).

- **Accessibilité** : Certaines options pourraient être repensées pour répondre aux besoins d'un public moins technique, comme des infobulles ou des tutoriels intégrés.

Voulons êtres les maîtres totales de notre application, nous avons décidé de ne pas utiliser le designer de Qt mais de faire toute l'interface graphique de l'application en code afin de ne pas passer pas un service tiers et de pouvoir contrôler entièrement ce que nous voulions afficher.

3. Réalisation et exécution

3.1 Structure du code

L'application est conçue de manière modulaire, ce qui favorise une meilleure lisibilité, extensibilité et maintenance. Voici un aperçu des fichiers principaux et de leurs rôles :

- PlanningPoker.cpp` et `PlanningPoker.h :
- Ces fichiers constituent le cœur de l'application. Ils gèrent les composants principaux, tels que l'interface utilisateur principale et les interactions utilisateur.
- Ils implémentent la logique pour naviguer entre les écrans et orchestrer les fonctionnalités comme le chargement et la sauvegarde des fichiers JSON.
- EstimationDialog.cpp` et `EstimationDialog.h:
- Responsable de la gestion des estimations des joueurs. Ce module affiche les cartes disponibles, capture les sélections des joueurs et retourne les résultats à l'application principale.
- Les cartes sont générées dynamiquement à partir des ressources inclues dans le projet.
- JsonViewerDialog.cpp` et `JsonViewerDialog.h:
- Permet aux utilisateurs de visualiser, éditer et valider les fichiers JSON chargés.
- Une validation visuelle permet de détecter les erreurs dans les fichiers JSON avant de commencer une session.
- main.cpp : Point d'entrée de l'application. Ce fichier initialise l'environnement Qt, configure les ressources nécessaires et lance la fenêtre principale.

3.2 Fonctionnalités implémentées

Les principales fonctionnalités de l'application incluent :

- Modes de jeu (strict et moyenne) :
- Le mode strict requiert une unanimité des joueurs pour valider une estimation.
- Le mode moyenne calcule automatiquement une estimation basée sur les choix des joueurs après deux tours sans consensus.
- Gestion complète des fichiers JSON :
- Les fichiers JSON stockent les tâches, les états des sessions et les préférences des joueurs. Ces fichiers sont manipulés efficacement via les outils Qt pour garantir leur validité.
- Sauvegarde et reprise des sessions : Chaque session sauvegarde automatiquement l'état actuel. L'utilisateur peut reprendre une session en chargeant le fichier JSON correspondant.

3.3 Manipulation des fichiers JSON

La gestion des fichiers JSON est une composante essentielle de l'application. Voici un détail des opérations :

- **Lecture et écriture :**

  L'application utilise les classes `QJsonDocument` et `QJsonObject` pour lire et écrire les données JSON. Cela garantit une compatibilité et une robustesse dans le traitement des fichiers.

- **Validation des fichiers :**

  - Une validation automatique est effectuée au chargement pour s'assurer que les fichiers JSON contiennent toutes les informations nécessaires (tâches, joueurs, modes de jeu).
  - Les erreurs détectées sont signalées immédiatement à l'utilisateur à l'aide de boîtes de dialogue conviviales.

3.4 Modes de jeu implémentés

Deux modes principaux sont disponibles :

- **Mode strict** : Ce mode impose une unanimité parmi les joueurs pour valider une tâche. Il est particulièrement adapté aux équipes souhaitant approfondir les discussions avant de converger sur une estimation.

- **Mode moyenne** : Si l'unanimité n'est pas atteinte après deux tours, une estimation moyenne est calculée automatiquement. Ce mode offre une alternative plus rapide tout en permettant une discussion initiale.

Ces modes ont été implémentés de manière modulaire, ce qui facilite l'ajout de nouveaux modes à l'avenir. En effet, nous pourrions rajouter un mode de jeu en moins de 5minute si nécessaire.

3.5 Résultats obtenus

L'application atteint ses objectifs initiaux en offrant :

- U**ne interface utilisateur fluide et fonctionnelle :**

Grâce aux outils Qt, l'interface graphique est intuitive, avec des composants interactifs bien organisés (boutons, cartes, dialogues).

- **Une gestion fiable des données :**

Les fichiers JSON sont gérés avec une grande robustesse, garantissant la sauvegarde et la reprise des sessions sans perte de données.

- **Une expérience utilisateur optimisée :**

Les joueurs bénéficient d'une navigation simple entre les écrans, avec une assistance visuelle pour les erreurs de fichiers JSON ou les étapes du jeu.

Les résultats démontrent la qualité de la conception initiale et la robustesse des choix techniques. Ces bases solides permettent d'envisager des évolutions futures, telles que l'ajout de nouveaux modes de jeu ou une extension multijoueur en réseau.

4. Modélisation

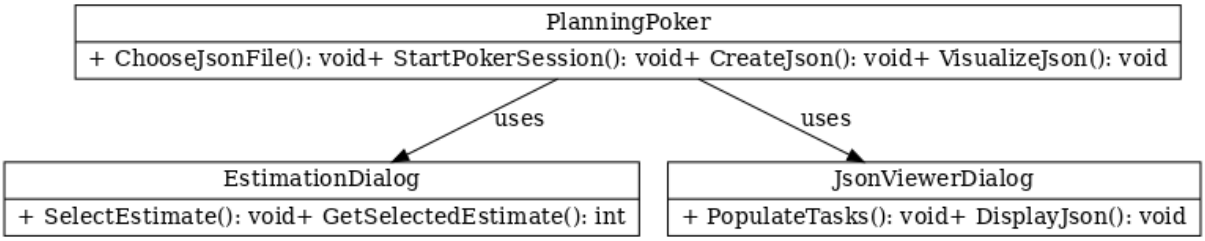
4.1 Diagramme de classes

Le diagramme de classes ci-dessous illustre les relations entre les trois principales classes de l'application :

PlanningPoker : Classe principale qui orchestre le fonctionnement de l'application.

EstimationDialog : Classe dédiée à la gestion des estimations des joueurs.

JsonViewerDialog : Classe responsable de l'affichage et de la manipulation des fichiers JSON.



Relations principales :

Utilisation : PlanningPoker utilise les classes EstimationDialog et JsonViewerDialog pour initier des dialogues interactifs.

L'implémentation modulaire permet une séparation claire des responsabilités, facilitant l'extensibilité et la maintenance du projet. Chaque classe peut être testée et améliorée indépendamment sans affecter les autres composants.

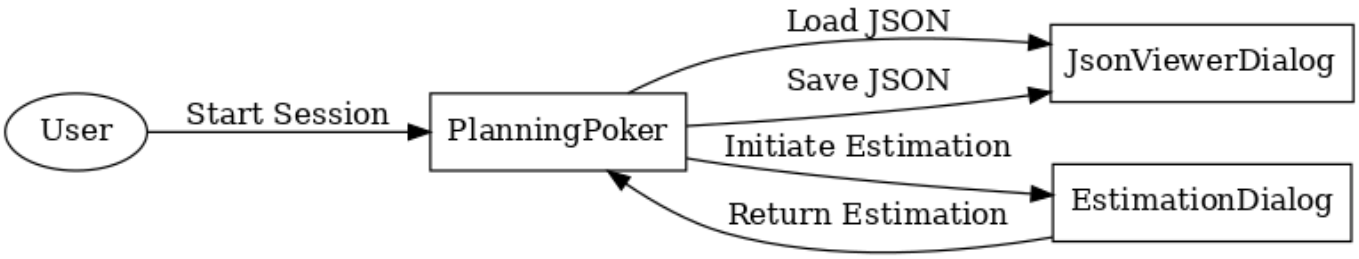
(Voir diagramme attaché : Diagramme de classes)

4.2 Diagramme de séquence

Le diagramme de séquence décrit le déroulement d'une session de jeu, depuis le chargement du fichier JSON jusqu'à la sauvegarde finale :

- **Initiation** : L'utilisateur démarre une session via l'interface principale (PlanningPoker).
- **Chargement** : PlanningPoker demande à JsonViewerDialog de charger un fichier JSON contenant le backlog des tâches et les paramètres de la session.
- **Évaluation** : PlanningPoker initie un dialogue d'estimation via EstimationDialog, où chaque joueur sélectionne une carte.
- **Résultats** : EstimationDialog retourne l'estimation sélectionnée à PlanningPoker.
- **Sauvegarde** : Une fois toutes les tâches évaluées, PlanningPoker utilise JsonViewerDialog pour sauvegarder l'état final dans un fichier JSON.

Ce flux garantit une gestion fluide et intuitive des interactions utilisateur tout en maintenant une structure logique robuste pour l'application.



5.1 Génération de la documentation

Pour garantir une documentation claire et détaillée du projet, Doxygen a été utilisé pour documenter toutes les classes, fonctions, et méthodes publiques du code.

Configuration :

Le fichier Doxyfile, inclus dans le projet, contient les paramètres nécessaires pour générer automatiquement la documentation. Cela comprend :

La description des modules principaux (PlanningPoker, EstimationDialog, JsonViewerDialog).

La documentation des paramètres et valeurs de retour des fonctions.

L'inclusion des diagrammes UML générés automatiquement pour visualiser les relations entre les classes.

Processus de génération :

La documentation est générée en exécutant le fichier batch UpdateDoxygenDocumentation.bat, qui compile les commentaires du code source et crée des pages HTML.

Les fichiers HTML générés fournissent une navigation facile pour consulter les fonctionnalités et modules du projet.

Utilité :

Cette documentation est essentielle pour les développeurs qui souhaitent comprendre rapidement l'architecture du projet et faciliter son extensibilité.

5.2 Configuration de l'intégration continue

Une intégration continue a été mise en place à l'aide de GitHub Actions pour automatiser les tâches critiques telles que la compilation, les tests, et la génération de documentation.

Workflow défini :

1. **Compilation** : Chaque push sur le dépôt déclenche une action pour compiler automatiquement le projet et s'assurer qu'il ne contient pas d'erreurs.

2. **Tests automatisés** : Bien que les tests unitaires n'aient pas été complètement intégrés (voir section 5.3), le workflow vérifie les bases du projet, comme la validation de la structure des fichiers JSON.
3. **Génération de documentation** : Les commentaires dans le code sont convertis en documentation grâce à Doxygen, et les résultats sont automatiquement déployés sur une branche dédiée.

4. **Avantages** :
   - Assure une rétroaction immédiate pour les développeurs en cas de problème dans le code.
   - Automatisation des tâches répétitives, réduisant les risques d'erreur humaine.

Comme nos différents commits laissent suggérer, nous avons aussi essayé d'automatiser la création de releases sur GitHub. Néanmoins, après plusieurs heures d'essai et d'échec infructueux pour la création d'un environnement MSVC compatible Qt nous avons malheureusement dû abandonner cette idée, même si nous n'étions pas loin de réussir.

5.3 Automatisation des tests unitaires

L'automatisation des tests unitaires a présenté des défis importants, principalement en raison de la complexité de tester une application graphique basée sur Qt.

- ✓ **Difficultés rencontrées :**

Interactions utilisateur : Tester les interactions, comme les clics sur les boutons ou la sélection des cartes, nécessite des outils spécifiques (e.g., Qt Test) qui demandent une configuration supplémentaire non encore finalisée.

Compatibilité CI : Les environnements CI/CD comme GitHub Actions ne sont pas optimisés pour exécuter des tests d'interface graphique sans environnement graphique dédié.

- ✓ **Solutions envisagées :**

Mise en place future de tests unitaires pour les composants non graphiques, tels que la gestion des fichiers JSON ou les calculs liés aux modes de jeu (moyenne, strict).

Utilisation d'outils comme Catch2 ou GoogleTest pour renforcer la couverture des tests sur les parties non-UI.

- ✓ **Limites actuelles :**

Malgré ces efforts, les tests unitaires complets pour les éléments graphiques n'ont pas été intégrés. Cependant, cette tâche reste une priorité pour les itérations futures du projet.

En conclusion, la documentation et l'intégration continue offrent une base solide pour maintenir et développer le projet. Bien que les tests unitaires ne soient pas entièrement aboutis, les fondations posées permettent de les intégrer plus facilement à l'avenir.

6. Résultats et analyses

6.1 Démonstration de l'application

L'application a été conçue pour offrir une expérience utilisateur fluide et intuitive. Voici les étapes principales de son utilisation :

- **Chargement d'un fichier JSON** : L'utilisateur peut sélectionner un fichier JSON contenant le backlog des tâches à estimer. Une validation est effectuée automatiquement pour détecter d'éventuelles erreurs dans le fichier.

- **Sélection des modes de jeu** : Avant de démarrer une session, l'utilisateur choisit entre le mode strict ou le mode moyenne.

- **Processus d'estimation** : Chaque joueur sélectionne une carte représentant une estimation. Ces choix sont ensuite traités pour produire des résultats conformes aux règles définies.

- **Sauvegarde de la session** : Une fois toutes les tâches évaluées, l'application enregistre automatiquement les résultats dans un fichier JSON, permettant ainsi une reprise ultérieure.

Des captures d'écran de l'application sont disponibles pour illustrer ces étapes clés, notamment l'interface principale, les dialogues d'estimation, et les résultats finaux.

6.2 Analyse des performances

L'optimisation en C++ combinée à l'utilisation de Qt garantit un temps de réponse rapide et une gestion efficace des ressources. Les performances clés incluent :

- **Chargement rapide des fichiers JSON** : Même pour des backlogs volumineux, l'application traite les données rapidement grâce à l'utilisation des classes natives de Qt.

- **Réactivité de l'interface utilisateur** : Les interactions sont fluides, et les transitions entre les écrans se font sans latence notable.

Ces performances assurent une expérience agréable pour les utilisateurs tout en minimisant les contraintes matérielles.

6.3 Limitations identifiées

Malgré les résultats satisfaisants, certaines limitations subsistent :

- **Tests unitaires incomplets** : L'absence de tests robustes pour les interactions graphiques limite la couverture globale des tests.

- **Gestion des erreurs JSON** : Bien que la validation soit implémentée, certains scénarios atypiques, comme des fichiers JSON très mal formatés, nécessitent une gestion d'erreurs plus avancée.

- **Interface utilisateur statique** : L'application ne permet pas encore une personnalisation approfondie de l'interface ou une configuration dynamique des options.

6.4 Améliorations futures

Pour adresser ces limitations et étendre les fonctionnalités de l'application, les améliorations suivantes sont prévues :

- **Intégration d'un mode multijoueur à distance** : Permettre à plusieurs utilisateurs de participer à une session via un réseau local ou une connexion internet.

- **Ajout de statistiques avancées** : Inclure des analyses détaillées pour chaque session, comme la répartition des estimations ou le temps moyen pour valider une tâche.

- **Extension des tests unitaires** : Développer une suite de tests couvrant les interactions graphiques, en exploitant des outils adaptés comme Qt Test.

- **Personnalisation de l'interface** : Offrir la possibilité aux utilisateurs de modifier les thèmes, les couleurs, ou les paramètres de l'application selon leurs préférences.

Ces évolutions garantiront une expérience encore plus riche et adaptée aux besoins des utilisateurs.

7. Conclusion

7.1 Synthèse des travaux réalisés

L'application Planning Poker atteint ses objectifs principaux en fournissant une plateforme efficace pour la collaboration et l'évaluation des tâches dans un environnement agile. Grâce à une interface intuitive, une gestion robuste des données JSON et des fonctionnalités adaptées aux besoins des utilisateurs, le projet répond aux attentes initiales. Les modes de jeu implémentés permettent une flexibilité dans les estimations, et les sessions peuvent être sauvegardées ou reprises à tout moment, offrant ainsi une grande praticité pour les utilisateurs.

En termes techniques, l'utilisation de C++ et Qt s'est révélée être un choix pertinent, garantissant à la fois performance et réactivité. Les outils comme Doxygen et GitHub Actions ont permis d'améliorer la qualité globale du projet, bien qu'il reste des points perfectibles, notamment dans la gestion des tests unitaires et la personnalisation avancée de l'interface. Cependant, les bases solides posées par ce projet offrent une plateforme idéale pour des évolutions futures ambitieuses.

7.2 Apprentissage et enseignements tirés

Le projet a permis de se confronter à des problématiques concrètes et d'acquérir une expertise approfondie dans plusieurs domaines :

- Concepts avancés de programmation C++ et Qt :
- Gestion de la mémoire et optimisation des ressources.
- Manipulation des signaux/slots pour synchroniser les interactions utilisateur.
- Intégration de bibliothèques pour la gestion des fichiers JSON.
- Travail collaboratif et pair programming :
- La collaboration a été essentielle pour avancer efficacement. Cela a permis de partager les compétences et de mieux répartir les tâches en fonction des points forts de chacun.
- Gestion des projets techniques complexes :
    - Mise en place d'une documentation technique structurée avec Doxygen.
    - Implémentation partielle d'un workflow CI/CD avec GitHub Actions, permettant de découvrir les défis et les opportunités liés à l'intégration continue.

➢ Ressenti de Tarek Bessad :

Je me suis particulièrement investi dans l'intégration graphique avec Qt, une tâche qui m'a permis d'explorer les nuances des interfaces utilisateur et d'améliorer mes compétences en design. Un moment marquant a été l'implémentation d'un système robuste de gestion des erreurs JSON. Ce défi m'a conduit à réfléchir à des scénarios atypiques, ce qui m'a aidé à renforcer ma capacité à résoudre des problèmes complexes tout en restant créatif dans les solutions proposées. J'ai également apprécié collaborer avec Lucas, en partageant des idées et en apprenant de son expertise technique, ce qui a renforcé ma propre compréhension des workflows et de l'organisation de projet.

- Ressenti de Lucas Journoud :

J'ai trouvé un grand intérêt à structurer le projet et à travailler sur les fonctionnalités principales comme les modes de jeu et la sauvegarde des sessions. La mise en place de la documentation technique avec Doxygen m'a offert une meilleure compréhension de l'importance de la documentation dans les projets collaboratifs. Bien que la finalisation des tests unitaires graphiques ait été frustrante, cette expérience m'a permis d'approfondir mes connaissances sur les workflows CI/CD et d'en mesurer la valeur pour garantir la stabilité des projets. Travailler avec Tarek a été enrichissant, car cela m'a permis de mieux appréhender les défis liés à la conception d'interfaces utilisateur et de trouver des solutions à des problèmes complexes grâce à une collaboration active.

7.3 Perspectives et évolutions possibles

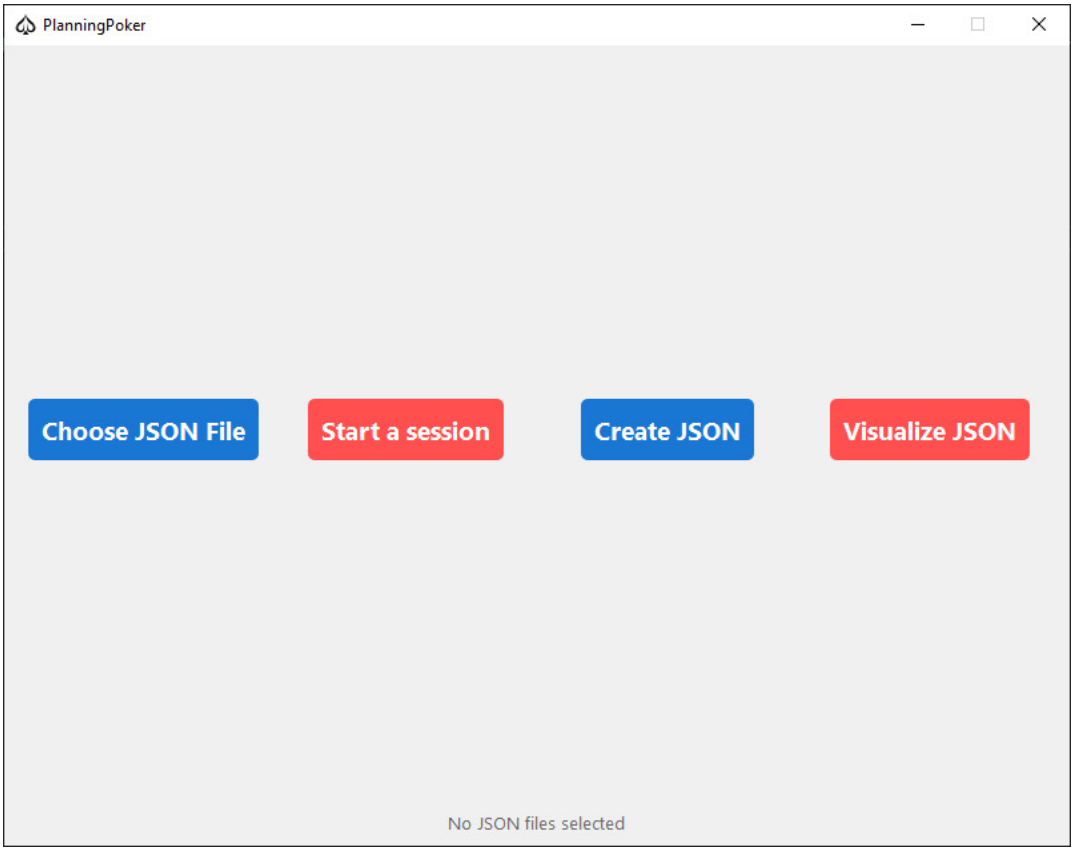
Le projet offre une base solide pour des évolutions futures qui pourraient rendre l'application encore plus performante et adaptée aux besoins des utilisateurs. Les pistes d'amélioration incluent :

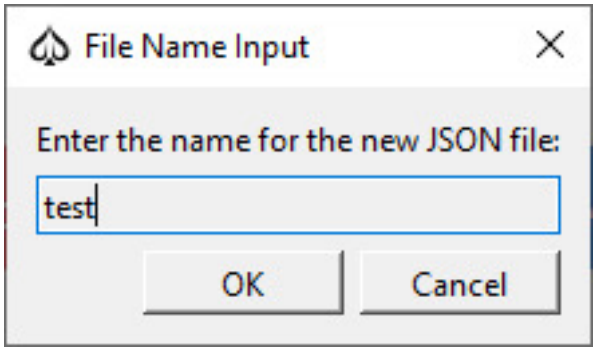
- **Extension en ligne** : Ajouter un mode multijoueur permettant à des utilisateurs distants de participer à une session via une connexion réseau.
- **Implémenter un système de chat intégré pour faciliter la communication entre les participants.**
- **Statistiques et visualisation des données** : Intégrer des graphiques et des analyses avancées pour chaque session, offrant des insights sur les estimations réalisées.
- **Ajouter des tableaux récapitulatifs pour évaluer les écarts entre les estimations des différents joueurs.**
- **Amélioration des tests unitaires** : Finaliser et étendre les tests unitaires pour inclure l'ensemble des fonctionnalités graphiques et non graphiques.
- **Automatiser les tests des fichiers JSON pour garantir leur robustesse face à des scénarios inattendus.**
- **Personnalisation et thèmes** : Permettre aux utilisateurs de configurer l'interface (thèmes, polices, couleurs) pour une expérience plus personnalisée.
- **Proposer un mode sombre pour améliorer le confort visuel lors de longues sessions.**
- **Accessibilité** : Rendre l'application plus accessible pour les utilisateurs ayant des besoins spécifiques, comme l'ajout de fonctionnalités compatibles avec des technologies d'assistance (lecteurs d'écran, commandes vocales).

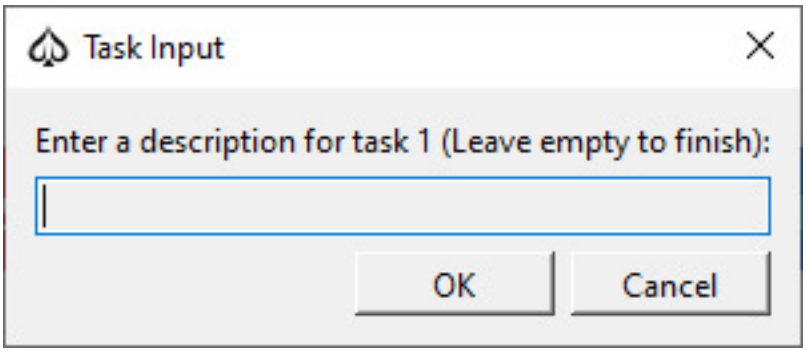
En conclusion, ce projet a été une excellente opportunité d'appliquer des connaissances théoriques à un projet concret tout en explorant des domaines techniques variés. Les défis rencontrés et les solutions apportées témoignent de l'engagement et de l'apprentissage constant, posant ainsi des bases solides pour des projets futurs. La complémentarité au sein de l'équipe a permis de tirer le meilleur parti de nos compétences respectives, renforçant la qualité globale du projet et ouvrant la voie à de nouvelles perspectives.

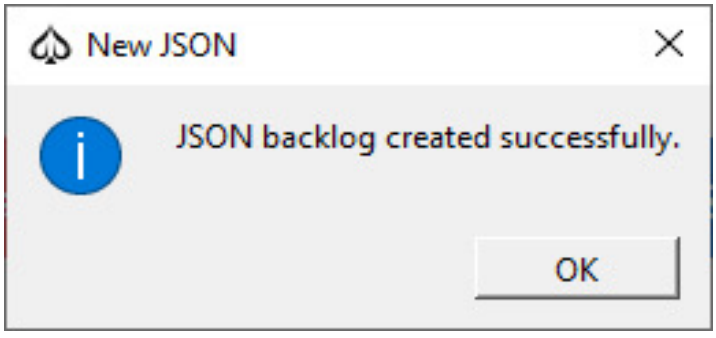
8. ANNEXES

8.1 Création json

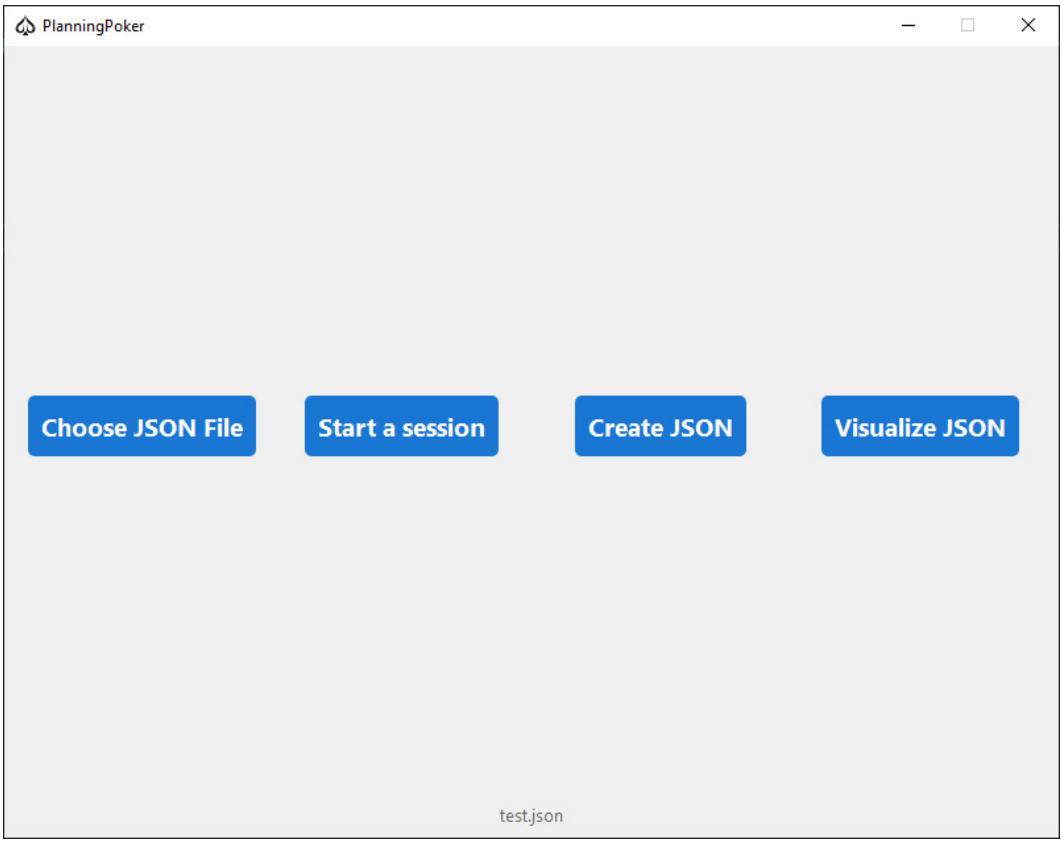


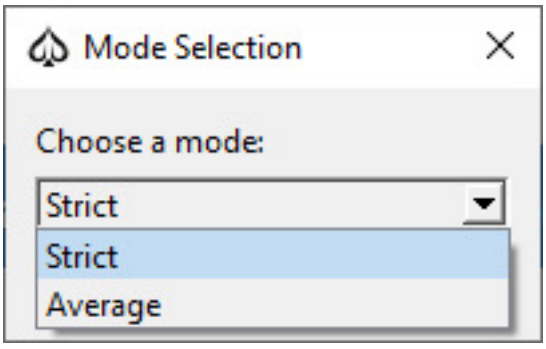


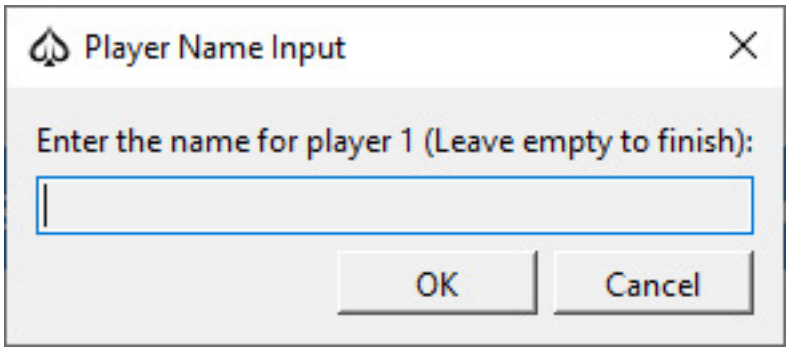


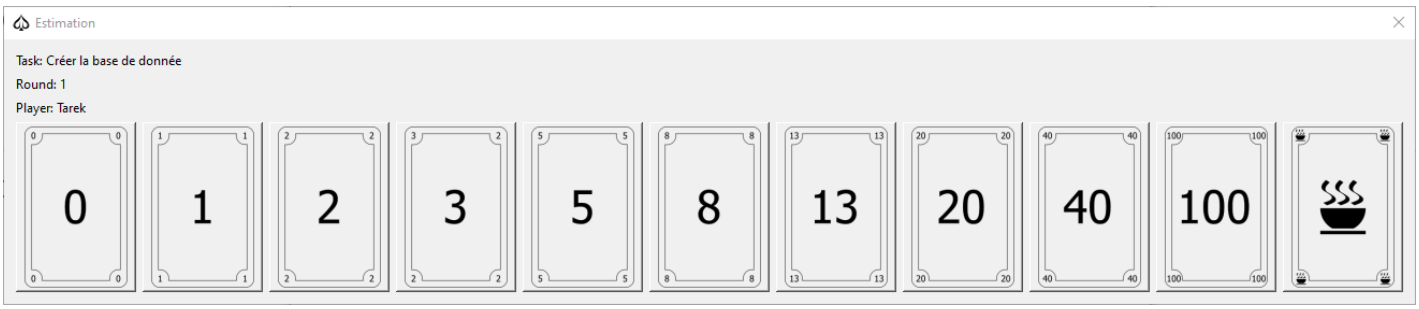


8.2 Start Session

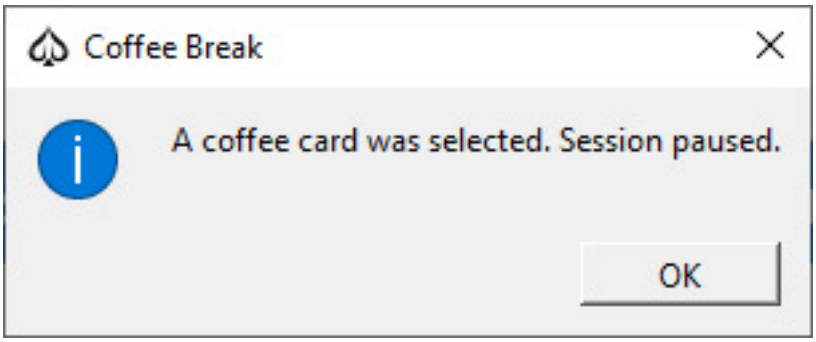


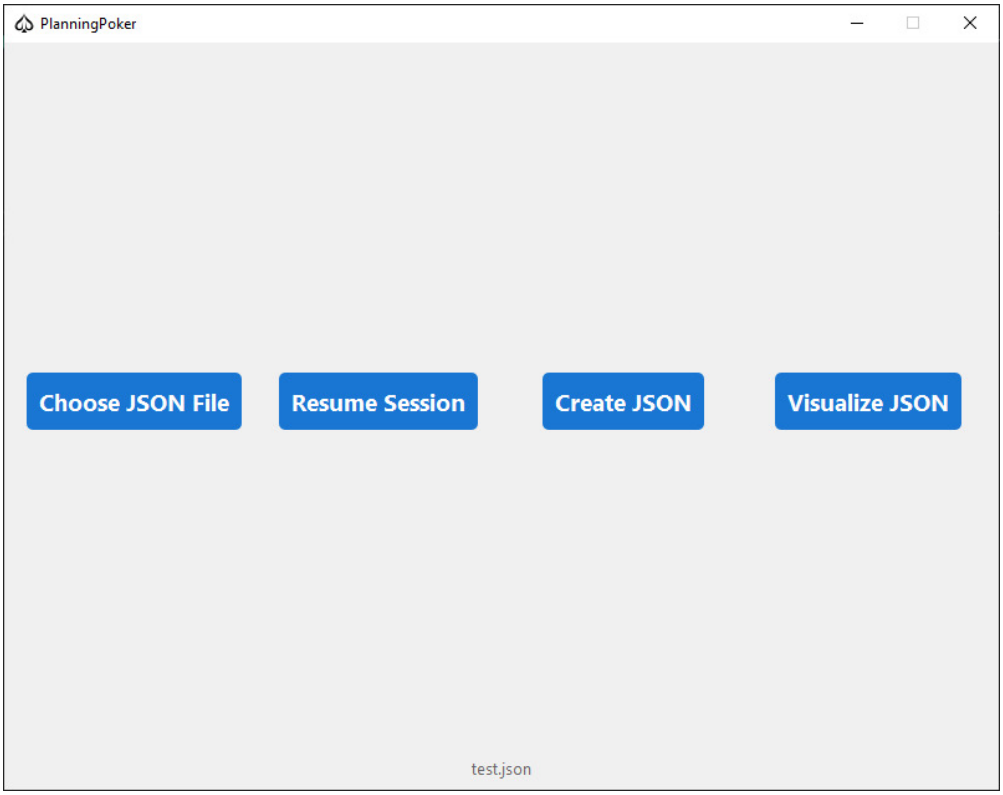






8.3 Carte café





8.4 Session finie