

RAPPORT DE PROJET – CAPI

Université Lumière Lyon 2
2024-2025

Planning Poker

Réalisé par

Elyas SAIDOUNE
Melissa BOULOUFA

Encadré par

Valentin LACHAND PASCAL

SOMMAIRE

I.	Présentation de l'application.....	3
II.	Justification des choix techniques.....	7
III.	Explication de la mise en place de l'intégration continue.....	8

I. Présentation de l'application

Ce jeu est une simulation du célèbre Planning Poker, une méthode collaborative utilisée pour estimer la difficulté des tâches à réaliser dans le cadre d'un projet informatique, en impliquant l'ensemble des membres de l'équipe.

Le projet a été implémenté en suivant la méthode de Pair Programming :

- L'un se concentrat sur l'architecture globale et l'autre sur la méthode courante (rédition du code). Nous échangions les rôles régulièrement.

Les fonctionnalités implémentées sont :

- Un menu permettant de :



Charger un fichier JSON (dans l'idéal bien structuré), choisir un mode de jeu parmi 5 (expliqués plus bas), ajouter des joueurs (en mettant le nom de chacun). Le nom fait maximum 20 caractères (vides en début et fin de nom exclus) et provoque une erreur (la case devient rouge et le joueur ne peut être ajouté) si :

- Il est vide.
- Un autre joueur utilise déjà ce pseudo.

Voir la liste des joueurs ajoutés et (éventuellement) les supprimer avec la croix, débuter la partie en cliquant sur Start (possible uniquement si tous les champs sont bien remplis et que le nombre de joueurs est supérieur ou égale à 2).



Exemple de fichier JSON bien structuré (inclus déjà rempli de notes dans le dépôt Git) :

```
{
  "fonctionnalites": [
    {
      "nom": "Connexion utilisateur",
      "description": "Permet aux utilisateurs de se connecter avec un email et un mot de passe.",
      "note": ""
    },
    {
      "nom": "Création de projet",
      "description": "L'utilisateur peut créer un projet et y associer des équipes.",
      "note": ""
    },
    {
      "nom": "Ajout de tâches",
      "description": "Les membres de l'équipe peuvent ajouter des tâches à un projet.",
      "note": ""
    },
    {
      "nom": "Estimation des tâches",
      "description": "Permet de voter sur l'effort nécessaire pour chaque tâche en utilisant le planning poker.",
      "note": ""
    },
    {
      "nom": "Tableau des tâches",
      "description": "Affiche un tableau Kanban des tâches avec leur statut (à faire, en cours, terminé).",
      "note": ""
    },
    {
      "nom": "Notifications",
      "description": "Envoie des notifications pour les événements importants (comme la fin d'un vote).",
      "note": ""
    }
  ]
}
```

Il y a un répertoire « fichiersJSON » avec 3 fichiers. Celui-ci, backlog_2.JSON, a déjà des notes préremplies.

- Une fois la partie commencée, le joueur voit affiché :

- Son nom

- Le nom de la fonctionnalité à évaluer ainsi que sa description (tirées du fichier JSON, on n'évalue que les fonctionnalités qui ont le champ « note » vide)
- Le numéro du tour en cours
- Un bouton pour quitter la partie (les données des tours précédents sont sauvegardées dans le fichier JSON)
- Une main de 12 cartes
- Un emplacement (cadre) pour placer la carte choisie
- Un bouton OK pour valider son choix et passer à la suite

Le jeu se déroule en local et comme suit en fonction du mode choisi :

1. Strict : les joueurs votent jusqu'à ce que l'unanimité soit acquise. A la fin de chaque tour, si l'unanimité n'est pas atteinte, une page de débat entre les deux extrémités s'affiche. Les joueurs ont 1 minute pour débattre et se mettre d'accord pour le prochain tour. Tant que l'unanimité n'est pas acquise, on recommence le processus autant de fois que nécessaire.

Pour les prochains modes, le premier tour de chaque tâche se joue quand même sur l'unanimité.

2. Moyenne : les joueurs votent et la carte gagnante est celle se rapprochant le plus de la moyenne des notes (Cf. fonction numberToCard() de la classe Card).

3. Médiane : similaire au mode moyenne, mais calcule la médiane à la place.

4. Majorité relative : les joueurs votent et la carte la plus utilisée est la gagnante. Si aucune carte n'est « plus » utilisée que les autres, la page de débat s'affiche puis le tour est rejoué.

5. Majorité absolue : similaire un mode majorité relative, mais ici avec la majorité absolue (plus de la moitié des cartes sont similaires).

Les cartes Joker « ? » et Café sont légèrement différentes des autres cartes :

- **Joker** : le joueur ne parvient pas à évaluer la tâche. Cette carte n'est pas prise en compte dans le calcul de la moyenne ni et la médiane. Pour les modes majorité et strict, si le Joker obtient la majorité ou l'unanimité, le tour recommence. De plus, il empêche l'unanimité même si c'est la seule carte différente des autres.
- **Café** : le joueur veut une pause. Si cette carte obtient l'unanimité ou la majorité, elle provoque un affichage « pause-café » et on peut retourner au jeu en cliquant sur la barre d'espace du clavier. Le bouton « quitter » ici sert à quitter la partie.

Quand une note est attribuée à une tâche, on affiche le nom de cette fonctionnalité ainsi que la note obtenue. Un bouton « Suivant » sert à passer à la fonctionnalité suivante. Le fichier JSON est sauvegardé à chaque ajout d'une note.

Une fois qu'on a fini de parcourir le fichier JSON (toutes les fonctionnalités ont une note), c'est la fin de la partie. Un petit résumé des notes attribuées est affiché sous forme de liste, et le joueur peut revenir au menu ou quitter la partie. Le fichier donné au début (au niveau du menu) est modifié et sauvegardé avec les notes.

Quelques images du jeu :





II. Justification des choix techniques

Langage :

Au départ, nous avions commencé à programmer cette application en Java. Une fois le backend terminé, nous nous sommes rendu compte que le frontend en Java nous prendrait trop de temps, comme cela avait été le cas dans des projets précédents. Nous avons donc décidé de convertir notre code en C# (facilement, car très semblable à Java) pour pouvoir modéliser le frontend avec Unity : c'est beaucoup plus simple ! Il faut bien évidemment de solides connaissances en C#, mais la création des objets, leur positionnement et leur intégration avec le backend dans Unity sont beaucoup plus pratiques et intuitifs que dans d'autres langages. Elyas avait déjà l'habitude d'utiliser Unity, tandis que moi, Melissa, j'ai appris à l'utiliser en programmant le Planning Poker. Je n'ai pas rencontré de difficultés et j'ai rapidement appris à manipuler les différents outils.

Architecture :

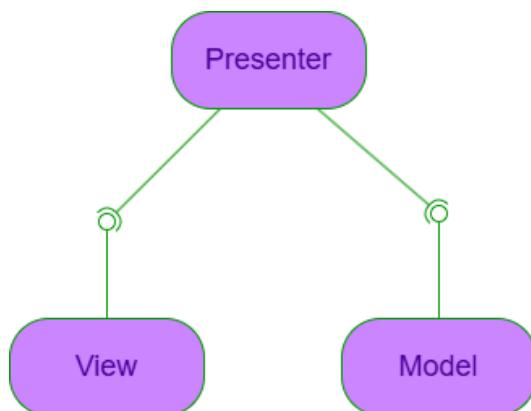


Diagramme de composants

Le projet est structuré selon le patron d'architecture MVP (Model View Presenter).

Les classes de la vue communiquent avec les classes du contrôleur et ce dernier communique avec les classes du modèle. Aucune communication n'est effectuée entre la vue et le modèle.

Nous avons choisi cette architecture car nous avons l'habitude de l'implémenter et qu'elle est idéale pour ce type de projet.

Classes :

Toutes les classes sont documentées, et la documentation est plutôt exhaustive et elle détaille bien la structure du projet. Nous voulions faire un diagramme de classes mais la documentation nous paraît finalement amplement suffisante.

⚠ La classe OpenJSONFile vient d'un plugin :

<https://github.com/gkngkc/UnityStandaloneFileBrowser>

III. Explication de la mise en place de l'intégration continue

Mise en place de la génération de documentation :

Nous avons utilisé Doxygen afin de générer automatiquement la documentation à partir des commentaires du code. Elle est déployée sur une GitHub pages, voici le lien (tous les fichiers sont documentés) :

<https://egoia.github.io/planning-poker/html/annotated.html>

Pour l'intégration continue, nous avions initialement opté pour GitLab parce que nous avons l'habitude de travailler avec ce logiciel. Un fichier .gitlab-ci.yml était utilisé pour configurer les pipelines d'intégration continue (CI).

Il se composait de plusieurs sections, chacune définissant un aspect spécifique du processus comme les étapes de compilation, de tests et d'autres tâches automatisées nécessaires au bon déroulement du projet.

La génération de documentation fonctionnait bien, mais nous avons rencontré des complications concernant l'automatisation des tests.

En raison du manque de documentation en ligne aidant à la configuration d'un fichier Yaml en GitLab compatible avec Unity, nous avons basculé vers GitHub car nous avions perdu trop de temps avec GitLab.

Sur GitHub, nous avons créé un fichier similaire dans notre dépôt, planning-poker/.github/workflows/documentation.yml, adapté à cette plateforme.

Ce fichier est utilisé par GitHub Actions, un outil de CI/CD pour exécuter des tâches automatisées chaque fois qu'un événement spécifique survient (comme un push sur la branche ToUnity).

Là encore, nous n'avons rencontré aucune difficulté pour la génération de la documentation (nous nous sommes aidés de vos exemples).

Mise en place de l'automation des tests unitaires :

Pour les tests unitaires, nous nous sommes aidés d'internet, de vidéos YouTube ainsi que de ChatGPT pour comprendre comment configurer un workflow GitHub compatible avec Unity.

Malgré ces ressources, nous avons eu beaucoup de mal à rédiger un fichier Yaml fonctionnel. Les principales complications étaient dues à la compatibilité entre GitHub Actions et le gestionnaire de tests Unity Test Framework, que nous n'avions jamais utilisé auparavant.

Nous avons créé un fichier nommé first_workflow.yml, situé dans /planning-poker/.github/workflows/, pour automatiser l'exécution des tests unitaires à chaque modification du code. Ce fichier configure un pipeline CI capable d'installer Unity sur la VM utilisée par GitHub Actions, exécuter les tests unitaires définis dans le projet (Assets/Scripts/Tests/testScript.cs) et générer un rapport détaillant les résultats des tests.

Comme nous avons perdu énormément de temps là-dessus, la couverture des tests unitaires n'est malheureusement pas parfaite.

Nous avons tout de même veillé à tester beaucoup d'éléments du backend comme le calcul de la note en fonction du mode, la modification et la sauvegarde du fichier JSON et quelques fonctions utiles au calcul de la note (numberToCard).

Nous reconnaissons que des tests plus complets auraient amélioré la robustesse de l'application, si nous avions plus de temps, nous aurions testé plus en profondeur les fonctions de calcul ainsi que celles liées au fichier JSON, et quelques fonctionnalités du controller comme la récupération des extrêmes (pour le débat). Quant au frontend, nous estimons qu'il n'a pas tant besoin de tests unitaires car nous testons déjà l'application en lançant une partie et en utilisant l'interface.

En dehors de l'implémentation des tests dans l'intégration continue qui nous a posé beaucoup de soucis, nous nous sommes beaucoup amusés à réaliser ce projet.