VEYRET Eliot

PONCET Paul

Projet CAPI

Année universitaire 2024 – 2025

Présentation du projet :

Pour ce projet, nous devions créer une application de planning poker. Le but est que des joueurs puissent ajouter des tâches en vue d'un projet, par exemple, et de pouvoir par la suite leur donner une estimation de la difficulté de ces mêmes tâches.

Il nous faudra faire attention à bien respecter les règles du planning poker, tout en ayant la possibilité d'ajouter de possibles améliorations au jeu.

Voici le fonctionnement du jeu :

Chaque joueur possède un jeu de cartes avec une valeur différente pour chacune des cartes.

- Une tâche est présentée aux joueurs, et ces derniers prennent le temps de la comprendre.
- Chaque personne choisit la carte avec la valeur de difficulté qui correspond au mieux à la tâche selon elle.
- Tout le monde dévoile en même temps la carte qu'il a choisie.
- Les personnes échangent entre elles par rapport à la difficulté qu'elles ont attribuée.
- En fonction de l'échange ainsi que du mode de jeu sélectionné, les participants donnent une nouvelle fois une estimation de la difficulté de la tâche.
- Un niveau de difficulté est acté, et une nouvelle tâche est sélectionnée afin de reproduire le même schéma de jeu.

Nous avons donc dû réaliser ce projet en prenant en compte le schéma de jeu. Il fallait également réaliser tout cela avec une démarche reprenant les principaux points vus en cours concernant les méthodes agiles, telles que le pair programming, par exemple.

Présentation de la web app et choix techniques

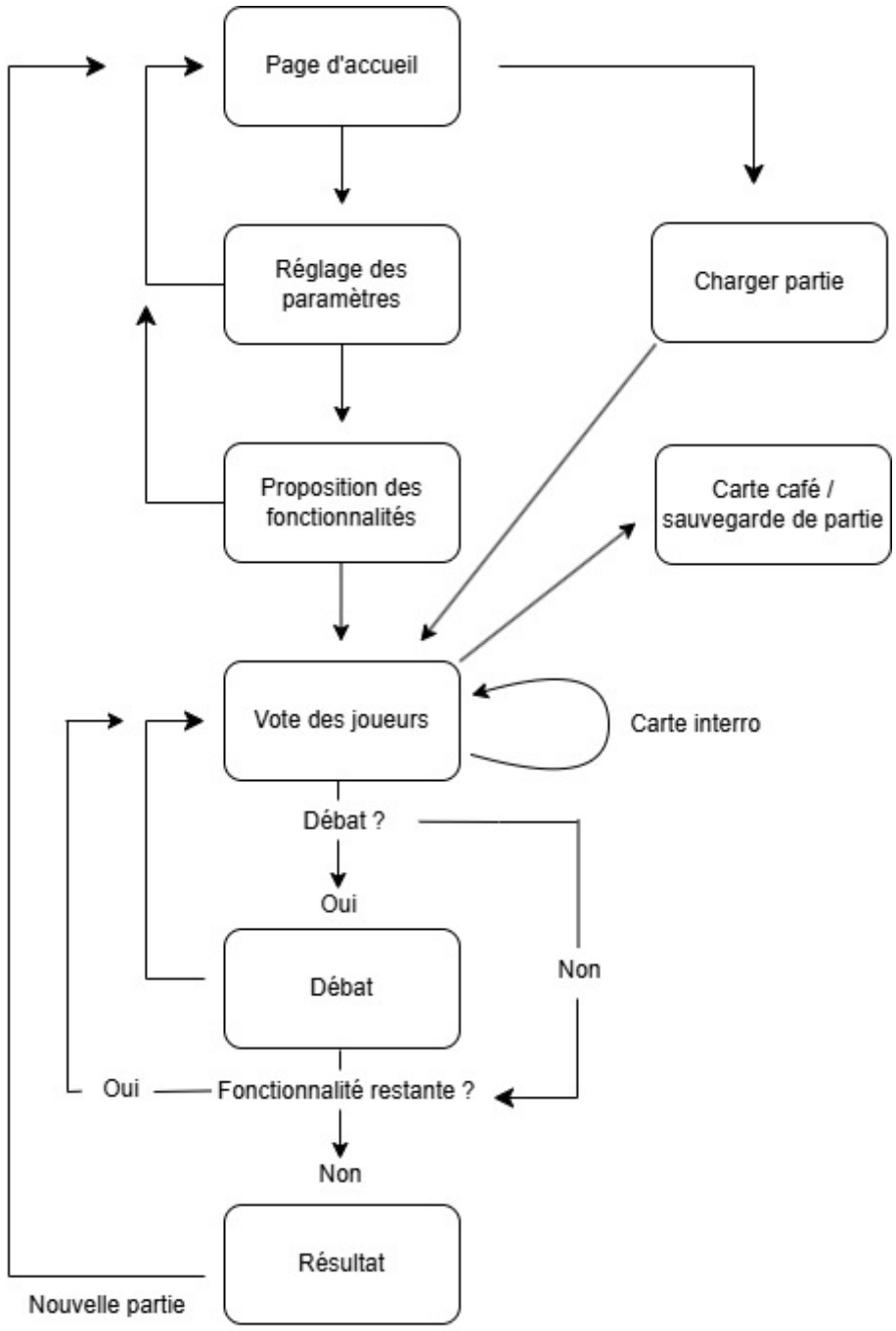
Dans le cadre de ce projet, nous avions la liberté de choisir le langage de programmation. Nous avons opté pour Python, un langage que nous avons utilisé dans de nombreuses matières. De ce fait, nous sommes beaucoup plus à l'aise avec ce langage plutôt qu'un autre.

Pour ce qui est de l'application web, nous avons choisi d'utiliser Flask, car nous l'avions déjà utilisé lors d'autres projets de création de pages web. Il est assez facile à prendre en main, et il existe un grand nombre de documentations le concernant. Il permet également d'avoir une assez grande liberté concernant la structure de notre application web. Cela semblait donc être un bon choix en vue du projet.

Concernant les codes nécessaires au bon fonctionnement de notre application web, nous avons tout d'abord notre fichier app.py, qui permet de lancer l'application et qui fera appel aux différents fichiers HTML. Nous avons, par conséquent, des fichiers HTML ainsi qu'un fichier CSS pour les différentes pages, telles que la page d'accueil ou encore celle des résultats et leur visuel.

Explication des programmes

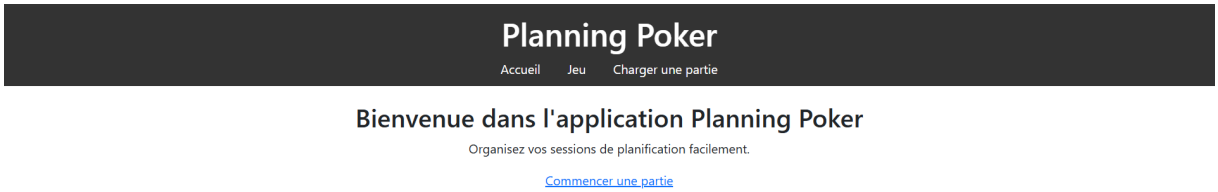
Nous avons décidé de nous focaliser sur le jeu en local afin de nous concentrer sur des fonctionnalités que nous jugions plus importantes. Avant de commencer à coder, nous avons réalisé un diagramme afin d'avoir une meilleure vision de ce que nous allions devoir créer, avec les différentes pages et options.



Pour chaque page web, nous avons une fonction associée, et nous allons détailler leur fonctionnement.

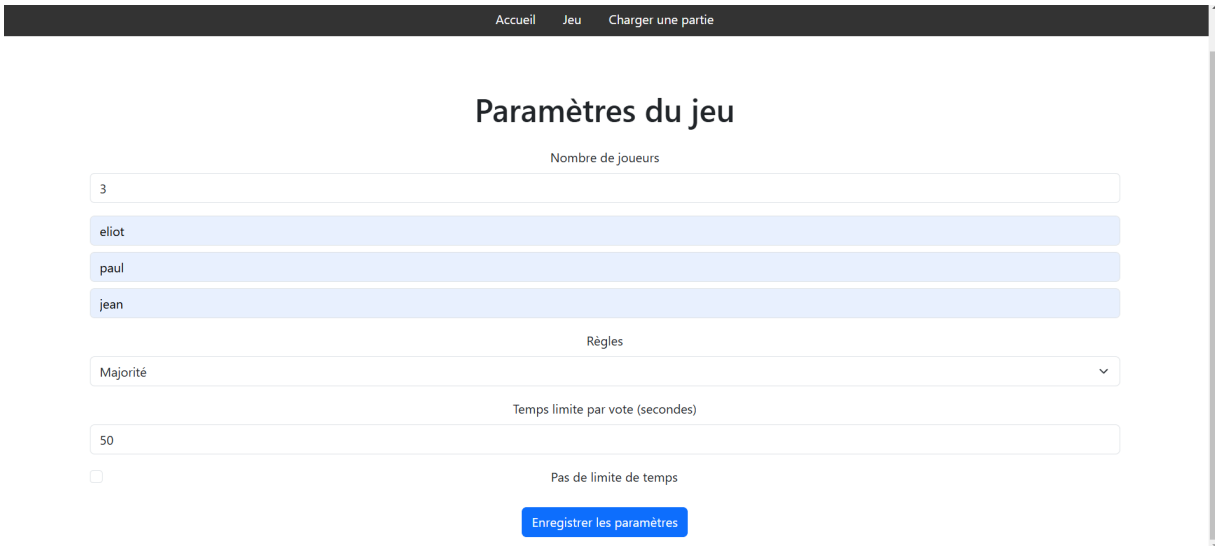
Résultat de la web app

Pour commencer, nous avons “home” et “settings”, qui vont retourner le template de la page d’accueil de notre site. Cela renvoie vers le fichier HTML du même nom correspondant.



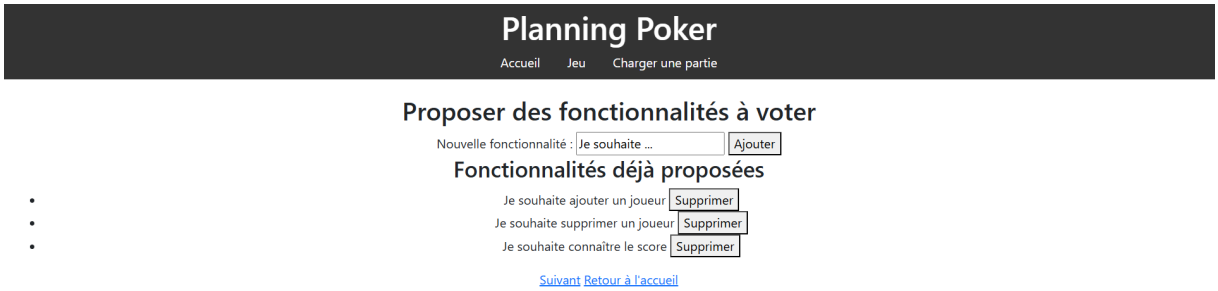
Nous avons une page d’accueil, avec un petit message de bienvenue, ainsi que la possibilité, en dessous, de lancer une nouvelle partie. On peut également charger une partie précédemment arrêtée (et sauvegarder localement).

Avec “settings”, nous allons pouvoir configurer les différents paramètres de la partie, tels que le nombre de joueurs (pouvant aller jusqu’à 10) avec leur prénom, la limite de temps par vote, ainsi que le choix des règles de la partie en termes de vote (unanimité ou majorité). Nous avons toujours la possibilité de retourner à la page d’accueil. Ici, l’utilisation de la méthode GET a pour objectif l’affichage des éléments et le chargement d’informations pour l’utilisateur.



“settings” nous retourne, quant à lui, le fichier HTML du même nom. Nous utilisons la méthode POST afin de sauvegarder nos données ou de soumettre un formulaire.

Une fois tous les paramètres configurés, un clic sur Suivant permet d'accéder à la page où les joueurs peuvent ajouter les tâches. Ces tâches serviront ensuite de base pour évaluer leur niveau de difficulté et engager une discussion entre les participants.



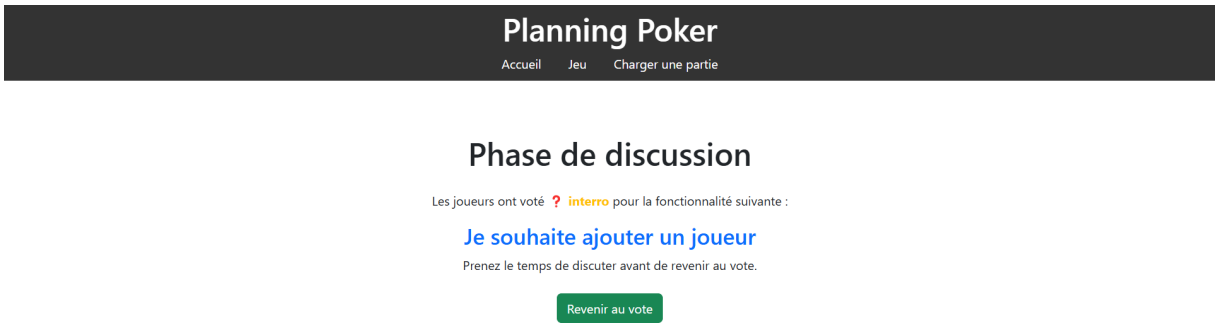
Ici, les utilisateurs vont pouvoir entrer les tâches de leur choix. Il y aura la possibilité de les supprimer avant de commencer la partie, dans le cas où deux tâches sont similaires, par exemple. Il est possible de revenir à la page d'accueil à tout moment ou de passer directement à l'étape suivante.

Pour cette partie, nous utilisons les fonctions "propose_features" et "delete_feature". La première concerne la proposition d'ajout de tâches par les utilisateurs. Quant à la fonction "delete_feature", elle permet de supprimer une tâche spécifique de la session actuelle.

Si on passe à la page suivante, nous allons pouvoir commencer à donner une difficulté aux tâches précédemment ajoutées.

Il y a toutes les valeurs ainsi que "café" et "interro" (pour interrogation). L'utilisateur va pouvoir sélectionner la valeur de son choix et, une fois cela fait, l'application passera à l'utilisateur suivant, et ce jusqu'au dernier. Une fois que tous ont choisi, en fonction du mode de jeu sélectionné, une page s'affichera avec le conflit ou la prochaine fonctionnalité.

Si tous les utilisateurs ne comprennent pas la fonctionnalité et souhaitent finalement avoir plus de temps pour la comprendre ou en discuter, ils peuvent choisir “interro”. Cela les mènera à cette page :



Is auront le temps d’en discuter ensemble et pourront revenir au vote par la suite.

À noter que cette carte est celle par défaut si un utilisateur n’a pas fait de choix durant le temps imparti.

S’ils cliquent sur café, cela enregistrera la partie, et les utilisateurs pourront la reprendre directement après une courte pause ou plus tard, en ayant sauvegardé la partie en local.

Sinon, comme dit précédemment, il pourra y avoir (ou non) une page de conflit en fonction du mode sélectionné et des votes:

Ici, c'est le mode "majorité" qui a été choisi. On décide donc de faire discuter les deux personnes avec les avis les plus opposés afin de réduire l'écart, jusqu'à ce que tout le monde soit d'accord sur une même difficulté ou qu'une majorité le soit.

Dans le code, nous aurons la fonction "game", qui, comme les autres, renvoie au fichier HTML du même nom.

Lorsqu'il y aura un conflit, les utilisateurs seront dirigés vers la page de débat. Sur cette page, il sera possible de refaire un vote afin que tout le monde puisse se mettre d'accord. Le code retournera alors au fichier HTML "game" pour permettre aux joueurs de voter de nouveau.

Une fois que toutes les tâches auront été traitées, une page récapitulative affichera les résultats, où nous pourrons voir, pour chaque tâche, la difficulté qui lui a été attribuée.

Nous aurons également la possibilité d'exporter nos résultats.

À partir de cette page, nous pouvons retourner à la page d'accueil afin de recommencer avec d'autres tâches.

Nous avons donc pu détailler les différentes étapes du fonctionnement de notre application web, en expliquant à la fois son utilisation par l'utilisateur et les choix des méthodes utilisées lors de sa conception côté code.

Méthode de travail :

Comme méthode de travail, nous avons réalisé du pair-programming en travaillant sur un même poste de travail. Nous pouvions, chacun à notre tour, voir ce que l'autre était en train de coder, afin de comprendre, mais aussi d'aider en cas d'erreur. De ce fait, nous avions un meilleur recul sur ce que nous faisions. Cette méthode nous a également permis d'améliorer notre manière de programmer en collaborant.

Nous avons décidé de travailler en sprints. Nous avons eu 4 sprints avec différentes user stories, auxquelles nous avons, au préalable, attribué une difficulté de réalisation. Cela facilite le sprint planning. Nous avons commencé avec les user stories relativement faciles, mais néanmoins primordiales pour la web app. (cf. Annexe 1)

Concernant notre repository, nous avons créé une branche par sprint basée sur une branche "develop", elle-même basée sur la branche "main". Pour chaque sprint, une fois terminé, nous faisions une pull request vers "develop" afin d'y fusionner la version à jour. Lors du sprint suivant, nous partions de ce que nous avions dans "develop", puis nous refaisions la même démarche que celle expliquée à l'instant.

Pour un sprint, la pull request sert à obtenir une validation de l'équipe, ce qui, dans notre cas, n'était pas forcément nécessaire grâce au pair-programming, mais que nous faisions tout de même.

Une fois l'application suffisamment avancée sur "develop", nous faisions une pull request de "develop" vers la branche "main".

Intégration continu :

Concernant l'intégration continue, nous nous sommes beaucoup aidés de la documentation fournie avec les consignes du projet.

Nous n'avons pas mis en place l'intégration continue dès le début, afin de nous concentrer sur le squelette du projet et de nous laisser le temps de comprendre comment cela fonctionnait, ainsi que ce à quoi cela allait ressembler lors de la création de la documentation. Il en est de même pour les tests. Nous avons effectué des tests avec les exemples qui nous ont été fournis et avons essayé d'en approfondir certains.

Une fois cela fait, nous avons commencé à implémenter les tests unitaires pour le code Python, ainsi que sa documentation automatisée.

Pour ce faire, dans la branche "develop" de notre repo GitHub, nous avons ajouté les fichiers nécessaires à son bon fonctionnement. Nous avons un fichier Doxyfile, qui permet de sélectionner le document dont nous souhaitons générer une documentation ainsi que le format de celle-ci. Nous avons également la possibilité de modifier certaines fonctionnalités (les fichiers à inclure/exclure, la présentation, etc.).

Le fichier .yml, dans le dossier github/workflows, sert à déclencher le processus et à exécuter toutes les étapes nécessaires. À chaque push dans "main", ce fichier génère la documentation dans une autre branche du repo nommée "gh-pages". Il suffit alors de télécharger les fichiers, de les décompresser, et d'ouvrir le document contenant la documentation.

Un second fichier .yml est présent dans le même dossier, mais celui-ci n'a pas besoin d'un fichier Doxyfile et s'exécute à chaque push et pull request sur "main" et "develop". Ce dernier concernera les tests unitaires, et exécutera les différents tests mis en place autour de l'application. Nous avons choisi cette fréquence car il est crucial selon nous de vérifier le bon fonctionnent des tests avant de fusionner les branches, tandis que la documentation peut être générée une fois que le code est validé.

Annexe

Annexe 1 - Résumé Sprint / US

Nous avons opté pour des premiers sprints relativement légers afin de mieux gérer la charge de travail liée à nos autres projets. En revanche, les derniers sprints étaient plus denses, profitant du fait que nous avions moins de cours en fin de semestre.

- Sprint 1 -

1. Avant la partie / réglages

1. **US1** : En tant qu'utilisateur, je peux définir le nombre de joueurs pour une partie. (Estimation : 0.5)

2. **US2** : En tant qu'utilisateur, je peux saisir un pseudo pour chaque joueur. (Estimation : 0.5)

3. **US3** : En tant qu'utilisateur, je peux choisir parmi plusieurs règles de planning poker (strictes, moyenne, médiane). (Estimation : 0.5)

- Sprint 2 -

2. Proposition des fonctionnalités

4. **US4** : En tant qu'utilisateur, je peux proposer des fonctionnalités à voter. (Estimation : 1)

5. **US5** : En tant qu'utilisateur, je peux voir la liste des fonctionnalités déjà proposées. (Estimation : 1)

6. **US6** : En tant qu'utilisateur, je peux supprimer des fonctionnalités proposées. (Estimation : 2)

7. **US20** : En tant qu'application, je permets aux joueurs de proposer différentes fonctionnalités à tour de rôle, puis laisse le joueur principal décider desquelles conserver ou supprimer. (Estimation : 4)

- Sprint 3 -

3. Système de vote

8. **US8** : En tant que joueur, je peux voter pour une fonctionnalité à l'aide d'un système de cartes. (Estimation : 10)

9. **US9** : En tant qu'application, je valide ou non une fonctionnalité en fonction des règles choisies. (Estimation : 4)

10. **US10** : En tant que joueur, je peux revoter si une fonctionnalité n'est pas validée. (Estimation : 2)

11. **US21** : En tant qu'application, j'affiche les résultats finaux une fois que toutes les fonctionnalités ont été votées. (Estimation : 4)

12. **US22** : En tant qu'application, je fais débattre les deux joueurs ayant les votes les plus extrêmes si une fonctionnalité n'est pas validée. (Estimation : 5)

- Sprint 4 -

4. Interface et expérience utilisateur

13. **US11** : En tant qu'utilisateur, je peux utiliser l'application en mode local (tour par tour sur un seul dispositif). (Estimation : 1)

14. **US12** : En tant qu'utilisateur, je peux utiliser l'application à distance (chaque joueur sur son propre dispositif). (Estimation : 10/15)

15. **US13** : En tant qu'utilisateur, je peux naviguer dans un menu ergonomique regroupant toutes les options. (Estimation : 2)

5. Fonctionnalités supplémentaires

16. **US14** : En tant qu'utilisateur, je peux utiliser un chronomètre pour limiter le temps de vote. (Estimation : 5)

17. **US15** : En tant qu'utilisateur, je peux discuter avec les autres joueurs dans un espace de chat intégré. (Estimation : 10)

6. Chargement, sauvegarde et export

18. **US16** : En tant qu'utilisateur, je peux charger une liste de fonctionnalités (backlog) au format JSON. (Estimation : 5)

19. **US17** : En tant qu'utilisateur, je peux exporter le backlog complété avec les estimations dans un fichier JSON. (Estimation : 5)

20. **US18** : En tant qu'utilisateur, je peux sauvegarder l'état d'avancement si tous les joueurs choisissent la carte café. (Estimation : 5)

21. **US19** : En tant qu'utilisateur, je peux reprendre une partie sauvegardée après avoir choisi la carte café. (Estimation : 0.5)

7. Discussion et réflexion

22. **US20** : En tant qu'utilisateur, je peux déclencher une phase d'interrogation sur une fonctionnalité. (Estimation : 2)

23. **US21** : En tant qu'application, je laisse les joueurs discuter d'une fonctionnalité si la carte interrogation remporte le tour. (Estimation : 2)