Planning Poker

ℹ️ Veuillez lire le fichier *README.md* (présent à la racine du projet) afin de faire fonctionner correctement notre application sur votre machine et de pouvoir accéder à la documentation de celle-ci.

I. Présentation du projet

Le Planning Poker représente une méthode cruciale d'estimation des tâches dans le domaine du développement logiciel. Cet outil permet à une équipe de développeurs de parvenir à un consensus quant à l'évaluation de la complexité de chaque tâche à réaliser.

L'application que nous avons développée offre une interface intuitive pour mettre en pratique cette méthode. Dès son lancement, elle propose cinq modes de jeu distincts : "strict", "moyenne", "médiane", "majorité absolue", "majorité relative". De plus, elle offre la possibilité de reprendre une partie précédemment sauvegardée, permettant ainsi une continuité sans faille, sauf dans certains scénarios :

- Lorsqu'aucune partie n'a été sauvegardée sur l'appareil, le bouton correspondant est désactivé, affichant un aspect gris.

- Une partie sauvegardée et terminée affichera un message pour informer l'utilisateur.
- En cas de corruption de la sauvegarde, qu'il s'agisse de clés manquantes ou de modifications extérieures à l'application, un message d'avertissement sera affiché.

Lorsqu'un utilisateur décide de commencer une nouvelle partie, il est redirigé vers la page de configuration des joueurs. Cette interface dynamique permet d'entrer les noms des joueurs et de configurer jusqu'à cinq participants. Les boutons d'ajout et de suppression de joueurs s'activent ou se désactivent automatiquement en fonction du nombre de joueurs, garantissant ainsi un ajustement dynamique. L'application vérifie également que chaque joueur configuré possède un nom avant de passer à la configuration des tâches.

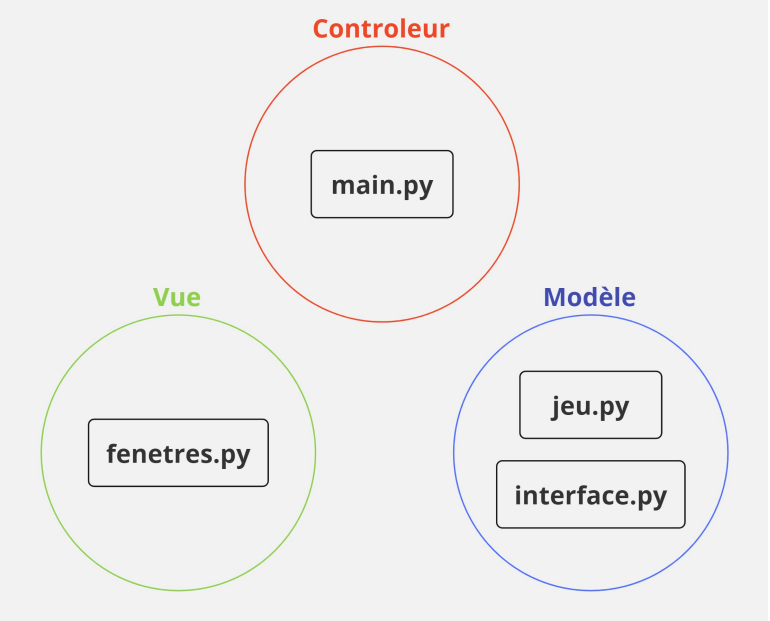
La page de configuration des tâches offre la possibilité de saisir le titre et la description de chaque tâche. Une tâche doit au minimum comporter un titre pour être enregistrée. La validation des tâches permet ensuite de démarrer la partie, à condition qu'au moins une tâche ait été créée et enregistrée. À tout moment lors de ces configurations, l'utilisateur peut revenir à l'étape précédente en cliquant sur le bouton en haut à gauche.

Lorsque la partie démarre, l'interface de jeu s'affiche en plein écran pour optimiser le confort de l'utilisateur. Cette page présente le mode de jeu sélectionné précédemment, la tâche en cours (avec son titre et sa description), ainsi que le joueur qui doit jouer. Un bouton dédié permet de quitter la partie à tout moment sans enregistrer les données, mais un avertissement est généré pour éviter toute manipulation involontaire. En accord avec les règles établies, la partie est enregistrée dès que toutes les tâches ont été évaluées ou dès que tous les joueurs sélectionnent la carte café.

II. Architecture du projet

1. Structure du projet

Le projet est organisé selon le modèle MVC (Modèle-Vue-Contrôleur) pour assurer une architecture logicielle claire et modulaire :

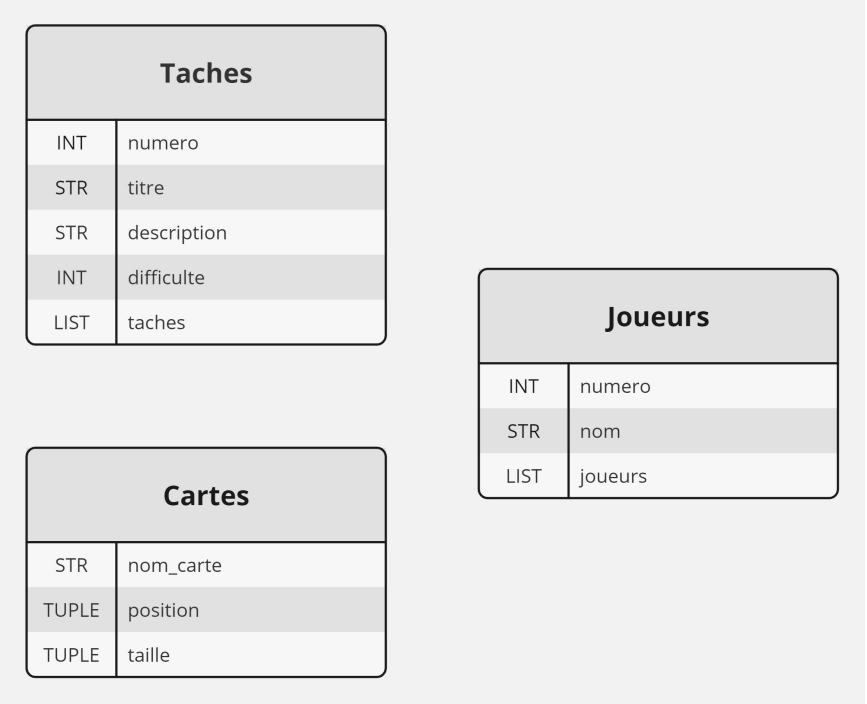


Schématisation du modèle MVC de l’application

Cette structure MVC offre une séparation claire des responsabilités, favorisant ainsi la maintenabilité, l'évolutivité et la lisibilité du code. Le modèle encapsule la logique métier, la vue se concentre sur la présentation visuelle, tandis que le contrôleur assure la synchronisation entre les deux, offrant ainsi une expérience utilisateur cohérente.

Modèle (fichiers *jeu.py* et *interface.py*)

- ***jeu.py*** : Ce fichier incarne l'essence même du jeu. Il consolide la logique métier du Planning Poker en prenant en charge la gestion des joueurs, des tâches, des cartes, ainsi que le déroulement complet d'une partie. C'est ici que sont implémentées les diverses mécaniques de jeu et les différents modes de jeu. La classe `Joueurs` est utilisée pour créer et gérer les joueurs du jeu, tandis que la classe `Taches` permet la création et la gestion des différentes tâches. La classe `Cartes` quant à elle est dédiée à la manipulation et à l'affichage des cartes utilisées dans le jeu.



Modèle UML des classes importantes de l'application

- ***interface.py*** : Ce fichier orchestre l'aspect visuel et interactif de l'application. Dans l'optique d'utiliser la bibliothèque graphique `pygame`, nous avons dû concevoir manuellement les éléments graphiques requis. Voici les objets standardisés que nous avons créés :
  - `Fenetre` : Gère la création et la gestion d'une fenêtre graphique grâce à `pygame`. Cette classe permet de définir la taille de la fenêtre, son titre, sa couleur de fond et de gérer son affichage.
  - `Rectangle` : Utile pour créer et manipuler des rectangles à l'écran. Cette classe permet de définir les propriétés comme la position, la taille, la couleur de fond, et la couleur des bords du rectangle, tout en offrant des méthodes pour l'affichage sur une surface.
  - `Bouton` : Gère la création et la gestion de boutons interactifs. Elle permet de définir des attributs tels que la position, la taille, le texte, la taille de police, la couleur du texte et du bouton, facilitant ainsi la manipulation et l'affichage des boutons sur une surface.
  - `BoiteTexte` : Permet de créer et de gérer des boîtes de texte. Cette classe offre des fonctionnalités pour définir la position, le texte, la taille de police, la couleur et l'alignement du texte, ainsi que des paramètres pour contrôler la longueur maximale des lignes de texte affichées sur une surface donnée.

- `BoiteSaisie` : Dédiée à la création et à la gestion de zones de saisie interactives. Elle offre des options pour définir la position, la taille, la taille de police, la couleur, le nombre maximal de caractères autorisés, et contrôle la longueur maximale des lignes de texte saisies sur une surface spécifique.

Vue (fichier *fenetre.py*)

- ***fenetre.py*** : Ce fichier orchestre l'aspect visuel et interactif de l'application. Il représente la couche visible de l'application, assurant la présentation graphique des différentes interfaces utilisateurs. Sa responsabilité principale inclut la gestion de l'affichage, la disposition, l'apparence et les interactions visuelles. Il interagit étroitement avec le contrôleur pour refléter en temps réel les changements opérés par l'utilisateur. De plus, il se connecte au modèle pour créer des objets requis par l'interface graphique ou instancier les données nécessaires à leur affichage. Ce fichier sert d'interface directe entre l'utilisateur et les mécanismes logiques du jeu, assurant ainsi une expérience utilisateur cohérente et intuitive.

Contrôleur (fichier *main.py*)

- ***main.py*** : En qualité de contrôleur central, ce fichier assume la gestion des interactions entre la vue et le modèle. Il agit en réponse aux actions de l'utilisateur en sollicitant les fonctionnalités du modèle et en coordonnant les mises à jour nécessaires au niveau des vues. C'est ici que se déroule l'orchestration globale de l'application. En interprétant les actions de l'utilisateur, il appelle les méthodes appropriées du modèle pour traiter les données et maintenir la logique de jeu. Par la suite, il actualise les interfaces visuelles, garantissant ainsi la cohérence de l'expérience utilisateur.

2. Patterns

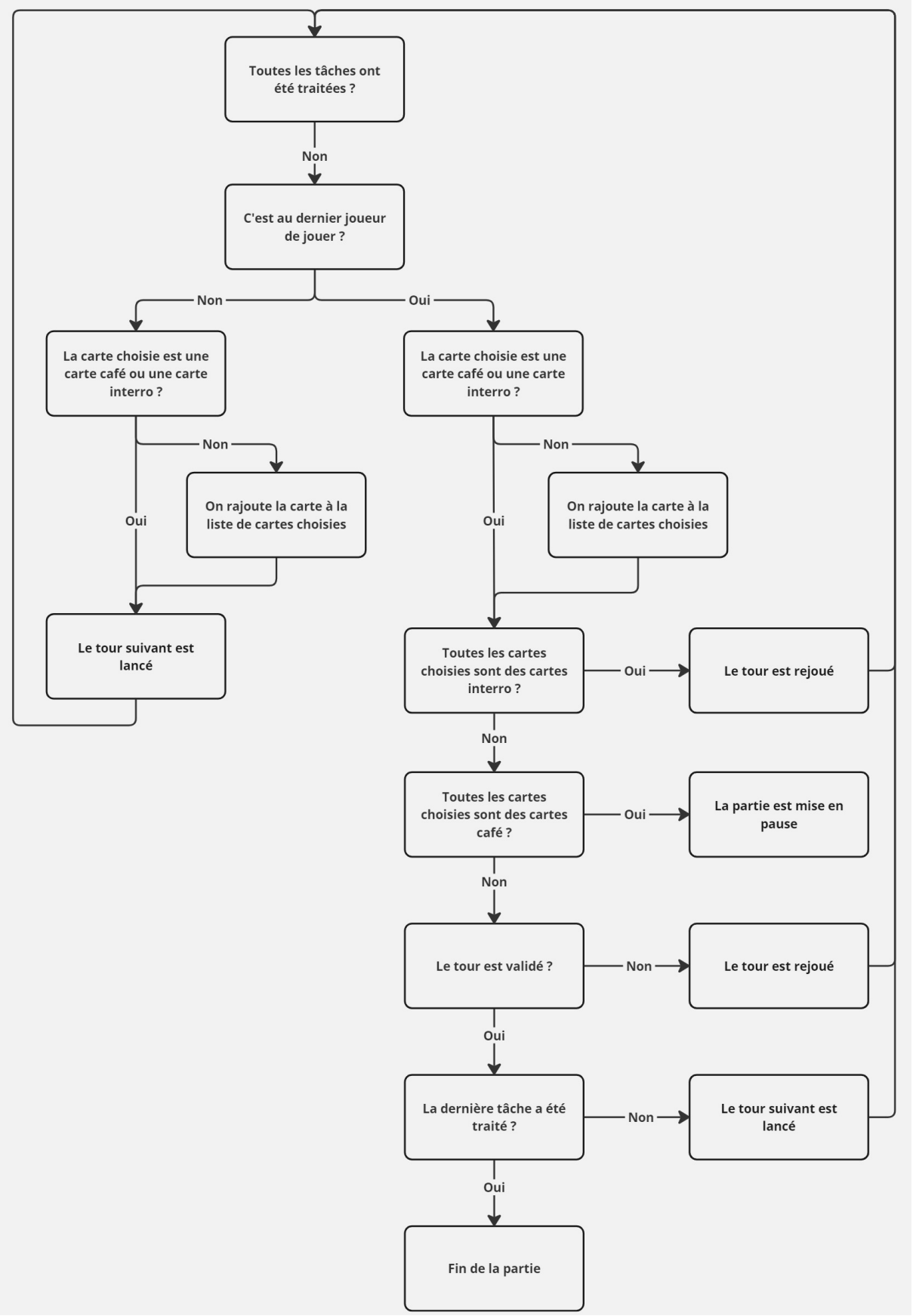
- Les classes `Joueurs` et `Taches` adoptent le pattern de conception Singleton. Ce pattern garantit qu'une classe possède une unique instance et fournit un point d'accès global à cette instance. Dans le contexte du Planning Poker, il est crucial de maintenir une seule et unique liste de joueurs et de tâches pour l'ensemble de l'application. Cela assure une gestion cohérente et synchronisée des données des joueurs et des tâches tout au long du déroulement de la partie. Dans le cas de la classe `Joueurs`, la liste `joueurs` est partagée entre toutes les instances de la classe, assurant qu'il n'existe qu'une unique liste `joueurs` pour l'ensemble des instances de `Joueurs`.

- Pour les classes `Joueurs`, `Taches`, `Cartes`, `Fenetre`, `Rectangle`, `Bouton`, `BoiteTexte`, `BoiteSaisie`, une encapsulation rigoureuse des propriétés est implémentée, offrant une multitude de méthodes pour l'affichage, la détection des survols, les interactions par clic, etc. L'encapsulation des propriétés et des fonctionnalités dans ces classes permet de maintenir un haut niveau d'abstraction et de modélisation des différents éléments de l'application. Chaque classe possède des méthodes spécifiques pour l'affichage, l'interaction et la gestion des données. Cela offre une séparation claire des responsabilités, facilitant ainsi la maintenance et l'évolutivité du code.
- La classe `Partie` adopte le patron de conception State qui est utilisé pour modéliser l'état changeant de la partie. Les attributs tels que `premier_tour` et `partie_finie` représentent l'état courant de la partie. Les méthodes telles que `jouer_tour`, `fin_tour` et `fin_partie` modifient ces états, et le comportement de la classe évolue en fonction de ces derniers. Avec des attributs comme `premier_tour` et `partie_finie`, cette classe adapte son comportement en fonction de ces états. Cela permet une gestion fluide du déroulement de la partie et la prise en compte des règles du jeu, tout en maintenant une logique claire et compréhensible.
- La classe `FntAccueil` utilise le pattern de conception Composite. Ce pattern permet de créer une structure arborescente d'objets individuels et de groupes. En l'occurrence, `FntAccueil` représente une `Fenetre` contenant plusieurs autres objets tels que des `Bouton` et des `BoiteTexte`.
  La classe `FntConfigJoueurs`, tout comme les précédentes, applique le pattern Composite. Elle représente une `Fenetre` regroupant plusieurs autres éléments tels que des `Bouton`, `BoiteTexte` et `BoiteSaisie`.
  De même, la classe `FntConfigTaches` suit le pattern Composite. Elle représente une `Fenetre` comprenant divers éléments comme des `Bouton`, `BoiteTexte` et `BoiteSaisie`.
  Enfin, la classe `FntJeu` utilise le pattern de conception "Composite". Elle représente une `Fenetre` regroupant un ensemble d'objets tels que des `Rectangle`, `Bouton`, `BoiteTexte`, `BoiteSaisie` et `Cartes`. Ce pattern permet une organisation hiérarchique des composants graphiques pour une meilleure gestion de l'interface utilisateur.
  Le pattern Composite est employé pour créer une structure arborescente d'objets. Dans le contexte de ces classes représentant des interfaces graphiques, cette approche permet d'organiser les éléments visuels de manière hiérarchique. Par exemple,

`FntAccueil` contient des `Bouton` et des `BoiteTexte`, facilitant ainsi la gestion des éléments composants une fenêtre. Cela rend également l'ajout, la suppression ou la modification des éléments graphiques plus modulaires et évolutifs.

3. Focus sur le traitement d’une partie

Nous avons choisi de détailler visuellement cette section du code car elle représente le cœur essentiel de l'application, et implique un processus de traitement relativement complexe. Voici la logique appliquée par l'application lors du déroulement d'une partie. Il est à noter que le schéma présenté ci-dessous ne couvre pas tous les détails et omet intentionnellement certaines informations. Ce traitement vise à garantir le bon déroulement de la partie en gérant les exceptions telles que les cartes "interro" ou "cafe", tout en appliquant de manière correcte les règles choisies par l'utilisateur. Ce schéma représente donc la méthode `jouer` de la classe `Partie`.



Schématisation de la méthode `jouer` de la classe `Partie`

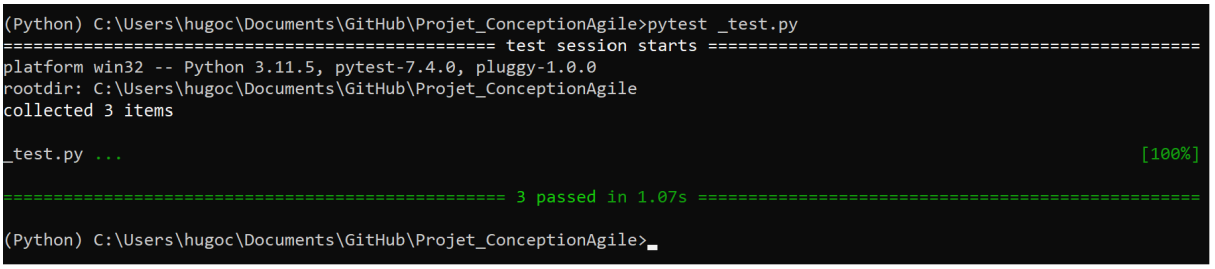
4. Intégration continue

Test unitaires

Les tests unitaires ont été un pilier fondamental dans l'évolution et l'amélioration de l'application. Ils ont permis d'assurer le bon fonctionnement des fonctionnalités cruciales, d'identifier et de résoudre des bugs potentiels, et d'améliorer la fiabilité globale de l'application, notamment dans la gestion du déroulement d'une partie.

- **La vérification du choix des cartes :** Le test unitaire `test_jouer()` a été conçu pour valider le processus de choix d'une carte par un joueur. Cela a permis de s'assurer que la méthode `jouer` de la classe `Partie` ajoute correctement la carte aux listes `cartes_choisies` et `log_cartes_choisies`, tout en incrémentant le joueur actuel. Ce test a été précieux pour valider ces aspects critiques du déroulement d'une partie.
- **La validation de la règle de fin de tour :** Le test unitaire `test_fin_tour()` a été déployé pour garantir la conformité de la règle de fin de tour. Ce test a permis de vérifier que la méthode `fin_tour` met bien à jour la difficulté de la tâche, incrémente correctement `tache_actuelle` et réinitialise `premier_tour` selon les règles spécifiées. Ainsi, cette validation a grandement contribué à assurer la logique du passage d'un tour à l'autre.
- **La réinitialisation des attributs à la fin de la partie :** Le test unitaire `test_fin_partie()` a été dédié à l'évaluation de la méthode `fin_partie` qui réinitialise les attributs de l'objet `Partie` lorsque la partie est terminée. Cela a permis de s'assurer que les attributs essentiels de la partie, tels que le joueur actuel, les cartes choisies et la tâche actuelle, sont correctement réinitialisés à la fin de chaque partie.

Ces tests unitaires ont joué un rôle crucial en garantissant la robustesse, la fiabilité et la cohérence de l'application. Ils ont permis de détecter et de corriger des erreurs potentielles, offrant ainsi une expérience utilisateur plus fluide et conforme aux attentes.

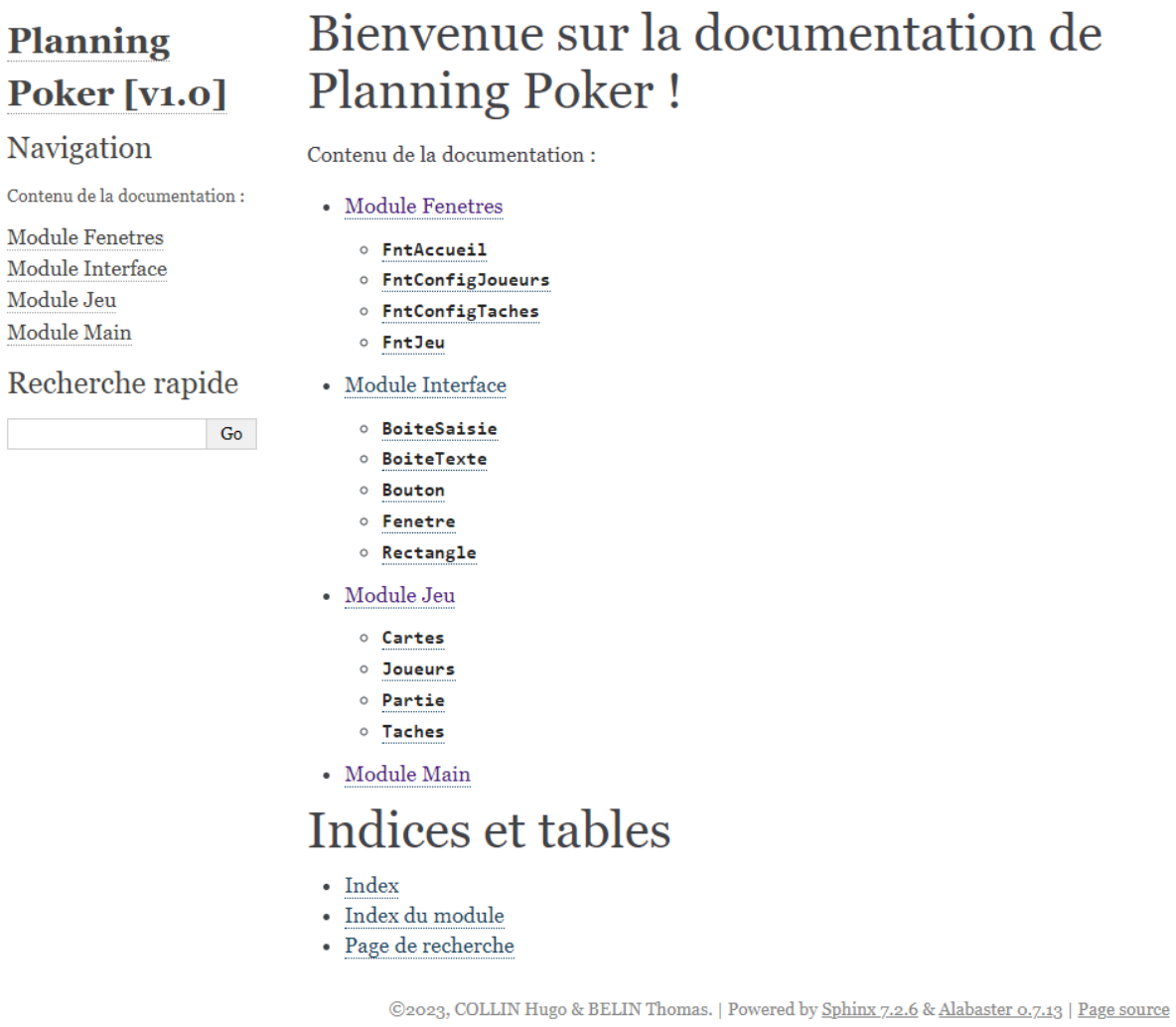


Aperçu de l'exécution de `pytest`

Documentation

Pour la documentation, nous avons opté pour l'utilisation de Sphinx, un outil de génération de documentation pour les projets Python. Sphinx offre plusieurs avantages notables :

- **La génération automatique de la documentation :** Sphinx simplifie la création de documentation en permettant d'extraire automatiquement des informations à partir du code source, ce qui réduit la charge de travail nécessaire pour documenter le projet. Chaque `docstring` présente sous une classe ou une méthode est extraite afin de créer la documentation. Pour cela nous avons donc renseigné une description détaillé, les attributs, les arguments et ce qui est renvoyé par la méthode.
- **La facilité d'utilisation :** Grâce à sa structure intuitive et à sa syntaxe simple, Sphinx permet de rédiger la documentation de manière claire et accessible pour les développeurs et les utilisateurs finaux.
- **La barre de recherche et la navigation :** Il propose une barre de recherche et une navigation efficace, offrant aux utilisateurs une expérience de lecture fluide et facilitant l'accès aux différentes sections de la documentation.
- **La création d'un site web local :** Sphinx génère automatiquement un site web local à partir de la documentation, permettant ainsi aux utilisateurs d'accéder facilement à toute la documentation du projet à partir d'un navigateur.



Aperçu de la page d'accueil de la documentation de Planning Poker

Il est essentiel de souligner que nous avons accordé une attention particulière à la précision de la documentation du code pour chacune des classes et méthodes que nous avons créées. Des commentaires détaillés ont été ajoutés aux parties complexes et essentielles du code, améliorant ainsi sa compréhension pour les personnes consultantes. Cette approche vise à garantir une facilité de compréhension pour l'ensemble du code source.

III. Conclusion

Le développement du Planning Poker nous a permit d'explorer et d'appliquer une variété de compétences allant de la conception logicielle à l'implémentation détaillée, tout cela en utilisant des méthodes de conception agile. Notre application offre une solution efficace pour estimer la complexité des tâches de développement grâce à une interface intuitive et complète.

Dans cette version actuelle, nous avons pu établir des axes d'amélioration clairs. L'ajout d'un chronomètre apporterait une dimension temporelle à la partie, permettant une gestion plus précise du temps passé par les joueurs. En parallèle, l'intégration de logs d'événements de partie fournirait une traçabilité détaillée des actions et décisions prises durant le jeu, facilitant ainsi l'analyse post-partie. De plus, l'implémentation d'un chat entre les joueurs directement dans la fenêtre de jeu offrirait un moyen de communication instantanée, favorisant la collaboration et les échanges pendant la session de jeu.