RAPPORT

CONCEPTION AGILE DE PROJETS INFORMATIQUES

PLANNING POKER

ELIAS BAROUDI & NASSIM GHLAMI

TABLE DES MATIERES

1. INTRODUCTION & OBJECTIFS

Nous avions pour projet de développer une application de *Planning Poker* permettant à des joueurs de participer à une partie en respectant les règles vues en cours. Nous avons choisi de concevoir l'application pour un fonctionnement à distance. L'interface propose un menu permettant de configurer le nombre de joueurs, d'enregistrer leurs pseudos et de choisir les différentes règles du *Planning Poker* (règles strictes, moyenne, médiane, etc.). L'application respecte également les exigences techniques imposées, notamment le format de fichier JSON, et intègre une fonctionnalité de chronomètre conforme à la note 2 des consignes.

2. PRESENTATION GLOBALE DU PROJET

Dans cette section, nous présenterons les fonctionnalités principales du projet.

2.1 MODE HOTE

L'hôte joue le rôle de serveur et est responsable de la gestion de la session. Les fonctionnalités associées sont les suivantes :

- **Création de la partie :** L'hôte configure les paramètres, notamment la méthode de calcul des votes (moyenne, médiane, etc.) et le délai de vote.
- **Chargement du backlog :** L'hôte charge les éléments du backlog à traiter durant la session.
- **Gestion des clients :** L'hôte accepte les connexions des participants et diffuse les questions à l'ensemble des joueurs.
- **Collecte et calcul des votes :** Les votes des participants sont collectés et les résultats sont calculés en fonction du mode de calcul sélectionné.
- **Sauvegarde des résultats :** Les résultats finaux ou l'état actuel de la partie sont enregistrés dans un fichier JSON.

2.2 MODE CLIENT

Les joueurs rejoignent une session en saisissant l'adresse IP de l'hôte. Les fonctionnalités associées sont les suivantes :

- **Connexion à la session :** Les joueurs se connectent en renseignant leur pseudonyme et l'adresse IP de l'hôte.
- **Participation au vote :** Les joueurs votent en sélectionnant les cartes disponibles sur l'interface utilisateur.
- **Gestion des désaccords :** En cas de non-accord, conformément aux règles, un retour d'information est transmis à tous les participants via un écran affichant le détail des votes, indiquant qui a voté quoi.

2.3 GESTION DU BACKLOG

Les tâches à estimer sont gérées sous forme de backlog au format JSON. L'hôte a la possibilité d'importer un fichier JSON contenant une liste de fonctionnalités à évaluer. Les résultats de la session sont enregistrés dans un fichier JSON. Si un utilisateur choisit la carte « café », la partie est arrêtée prématurément, ce qui déclenche la sauvegarde d'un fichier

backlog incluant les tâches déjà traitées, ainsi qu'une modification du fichier original pour en retirer les tâches ayant été évaluées.

2.4 METHODE DE CALCUL DES VOTES

Voici la liste des modes de jeu implémentés dans l'application. Veuillez noter que ces modes de jeu entrent en vigueur à partir du 2e tour uniquement. Le premier tour se basera sur une unanimité afin de laisser aux utilisateurs le temps de discuter de la tâche.

- **Moyenne** : Calcul de la moyenne
- **Médiane** : Calcul de la médiane
- **Majorité relative** : Calcul de la majorité la plus grande
- **Majorité absolue** : Validé uniquement si tous les joueurs votent la même carte.

3. JUSTIFICATION DES CHOIX TECHNIQUES

3.1 POURQUOI PYTHON ?

Nous avons choisi Python comme langage de programmation en raison de notre maîtrise avancée de celui-ci, ainsi que de sa popularité en 2024 sur GitHub. En développant cette application avec Python, nous renforçons nos compétences dans un langage à la syntaxe claire et concise, largement reconnu pour sa simplicité et son efficacité.

De plus, Python dispose de bibliothèques particulièrement bien adaptées à notre projet. Dès la présentation du projet, il est apparu évident que Python était le langage le plus approprié, notamment en raison de son écosystème. Pour la gestion de l'interface utilisateur graphique, nous avons opté pour Tkinter, une bibliothèque puissante et facile à implémenter, idéale pour créer des interfaces intuitives. Par ailleurs, l'utilisation de Socket, permettant la communication réseau entre l'hôte et les clients, répond parfaitement à nos besoins techniques.

3.2 POURQUOI TKINTER ?

Tkinter est intégré nativement à Python, ce qui évite toute installation supplémentaire. Cette caractéristique facilite également la compatibilité de l'application sur différents systèmes d'exploitation. De plus, Tkinter se distingue par sa simplicité d'apprentissage et d'utilisation, ce qui en fait un choix idéal pour la création de menus, de boutons, de labels et l'affichage d'informations. Ces fonctionnalités répondent parfaitement aux besoins de notre projet.

3.3 POURQUOI SOCKET ?

Le module socket de Python est particulièrement simple à configurer et à utiliser pour gérer les connexions réseau. Il permet d'établir une communication efficace entre l'hôte (serveur) et les clients, ce qui est parfaitement adapté pour l'échange rapide des votes et des résultats.

De plus, il permet à l'hôte de gérer simultanément plusieurs clients sans que l'un d'eux bloque le serveur, garantissant ainsi une interaction fluide entre les utilisateurs, qui peuvent se connecter et voter indépendamment les uns des autres.

3.4 JUSTIFICATION ET ROLES DES CLASSES

Nous avons utilisé trois classes principales (PlanningPokerApp, HostGame, et ClientGame) pour structurer à la fois l'interface du projet et le déroulement du jeu. Voici un détail concernant chacune d'elles :

3.4.1 CLASSE PLANNINGPOKERAPP

La classe PlanningPokerApp constitue le point d'entrée principal de l'application et joue le rôle de coordonnateur entre les différentes fonctionnalités. Elle est responsable de la configuration et de la gestion de l'interface utilisateur, permettant à l'utilisateur de choisir son rôle, soit celui d'hôte, soit celui de client.

Cette classe possède trois attributs principaux :

- **main** : une instance de Tkinter, qui gère la fenêtre principale de l'application.
- **host_game** : une instance de la classe HostGame, créée lorsque l'utilisateur choisit le rôle d'hôte.
- **client_game** : une instance de la classe ClientGame, créée lorsque l'utilisateur choisit le rôle de client.

Une architecture bien conçue sépare les rôles pour améliorer la lisibilité et la maintenabilité du code. Cela permet d'assurer la flexibilité et l'extensibilité de l'application : la classe PlanningPokerApp permet d'intégrer de nouvelles fonctionnalités sans impacter les autres classes.

3.4.2 CLASSE HOSTGAME

Cette classe gère le fonctionnement de l'application côté Hôte, en prenant en charge la configuration des interfaces ainsi que le déroulement de la partie.

- **Configuration de la partie** : Une interface permet à l'Hôte de définir les paramètres de la partie qu'il s'apprête à lancer. Il peut y définir le temps de discussion, le temps de vote, et choisir également le backlog contenant les tâches à estimer. (setup_host_interface(), parcourir())
- **Hébergement du serveur** : Le script démarre un serveur et ouvre un port sur la machine(*start_server_thread()*), permettant aux autres utilisateurs de se connecter à la partie(*listen_for_clients()*). Chaque nouveau joueur est accueilli avec un tableau affichant la liste des participants(*handle_client(), update_table(), broadcast_pseudos())*.
- **Déroulement de la partie** : Le serveur orchestre le déroulement de la partie jusqu'à sa fin, marquée par l'envoi de différents tags (*start_game(), start_game_loop()*) :

- **@@START@@** : lancement de la partie
- **@@NEW@@** : nouvelle question
- **@@FEEDBACK@@** : feedback des réponses
- **@@END@@** : fin de la partie

- **Traitement des votes** : Lors de la récolte des votes, l'interface Hôte s'occupe de traiter les votes reçus et de les faire correspondre à la règle choisie pour la partie (*collect_votes()*).
- **Processus de sauvegarde** : L'Hôte est responsable de la génération du fichier sauvegardant les tâches traitées. Ces dernières sont enregistrées dans un fichier JSON, comme expliqué précédemment.

3.4.3 CLASSE CLIENTGAME

Cette classe gère, contrairement à la classe HostGame, l'interface côté Client. Elle met en place principalement des processus d'écoute du serveur ainsi que l'interfaçage permettant à l'utilisateur de communiquer avec l'Hôte.

- **Processus de connexion à l'Hôte** : L'interface utilisateur permet de connecter la machine à celle de l'Hôte afin de rejoindre la partie (*connect_to_server()*). Une fois connectée, la machine entre dans un état d'écoute (*listen_to_server()*) jusqu'à ce que le server envoie le signal indiquant que la partie a commencé. Pendant ce temps-là, le client peut être amené à actualiser sa table selon les retours du server(*update_table()*).
- **Déroulement de la partie** (*start_game_loop()*) : Comme mentionné précédemment, le déroulement de la partie est orchestré par le serveur. Cela signifie que l'interface Client réagit en fonction des tags reçus du serveur. On trouve donc dans le code principalement des conditions de test sur les données reçues (par exemple : if tag == '@@tag@@').
- **Envoi de vote :** L'utilisateur dispose d'une interface de vote avec plusieurs cartes. Lorsqu'il clique sur l'une de ces cartes, un vote est envoyé au serveur (*send_vote()*) et l'interface client est bloquée jusqu'à ce que le serveur passe à l'étape suivante.
- Il a été jugé pertinent de mettre l'accent sur le fonctionnement des tags @@NEW@@ et @@FEEDBACK@@, qui sont les deux tags principaux régissant la partie :
    - @@NEW@@ : Ce tag signifie que le Client doit s'attendre à recevoir une nouvelle condition. La réception de ce tag entraîne un changement dans l'interface. Il est important de préciser que ce tag intervient uniquement lorsque la question est nouvelle, ce qui n'implique pas les situations où les utilisateurs ne sont pas d'accord sur les votes.
    - @@FEEDBACK@@ : Contrairement à @@NEW@@, ce tag est reçu lorsque les utilisateurs ne valident pas la condition du mode de jeu. Chaque utilisateur

reçoit alors un tableau avec la liste des joueurs et leurs votes, ce qui leur permet de discuter des résultats.

- Il est également pertinent de parler des décomptes (*start_countdown()*) qui ont été un vrai défi, puisqu'il fallait gérer les erreurs générées par des décomptes dynamiques à partir des retours du serveur. Il a donc été décidé d'implémenter les timers uniquement chez les clients afin de faciliter l'implémentation du décompte.

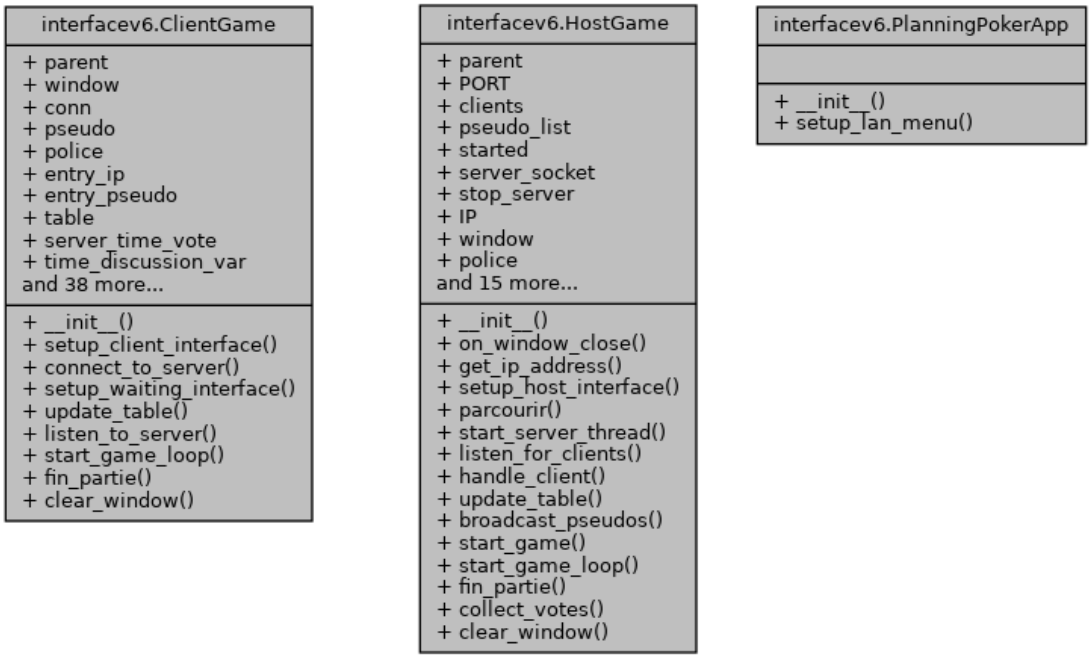
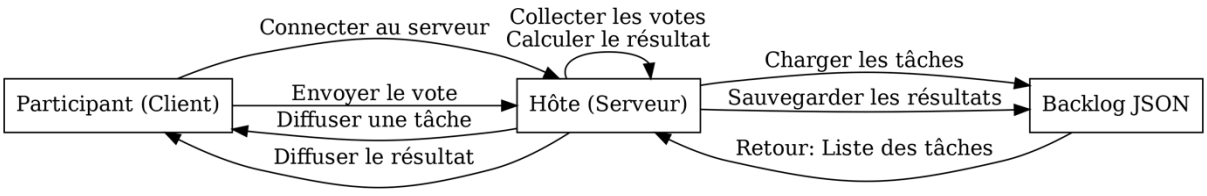


Diagramme de séquence du fonctionnement de l'application



Montre comment l'hôte diffuse des tâches aux clients, collecte leurs votes, calcule les résultats, et sauvegarde les données.

Inclut les interactions avec le fichier JSON pour charger les tâches et sauvegarder les résultats.

4. EXPLICATION DE LA MISE EN PLACE DE L'INTEGRATION CONTINUE

La mise en place de l'intégration continue constitue un élément clé dans le développement d'un projet informatique, comme abordé en cours. Elle a permis de garantir la qualité du code tout au long du projet, ainsi que la génération automatique d'une documentation complète sur le code, selon des règles prédéfinies.

4.1 TESTS UNITAIRES

Les tests unitaires sont essentiels pour maintenir une version fonctionnelle et correcte du code. Ils permettent aux développeurs de détecter les erreurs le plus tôt possible lors du développement de nouvelles fonctionnalités.

Dans le cadre de notre projet, cela s'est révélé plus complexe que prévu, étant donné que nous avons travaillé sur une application impliquant une gestion importante de l'interface. En effet, le code est composé à 60 % de fonctions dédiées à la gestion de l'interface, et non uniquement de logique métier.

4.1.1 CHOIX DE L'OUTIL

L'outil choisi dépend du langage de programmation utilisé. Dans notre cas, avec Python, nous avons opté pour l'outil pytest. Il s'agit d'un outil qui permet d'exécuter des programmes écrits en .py et d'effectuer des assertions en utilisant les fonctions du script principal.

Le script de test charge d'abord les classes et les méthodes du script principal, puis il est de notre responsabilité d'écrire les tests manuellement.

4.1.2 STRATEGIE

Le fichier test.py a été développé en parallèle au développement du script principal. Il a été primordial d'appliquer cette méthode pour garantir dès le début une application fonctionnelle, mais surtout pour tester chaque nouvelle fonctionnalité et détecter les erreurs le plus tôt possible, sans attendre la fin du développement pour tout tester.

4.1.3 EXPLICATIONS TECHNIQUES

Concernant la mise en place du script de test : Il a été crucial pour nous de nous concentrer uniquement sur les aspects essentiels du déroulement de l'application, en raison de la complexité du code lié à la gestion des fenêtres. Voici un résumé du fonctionnement des fonctions écrites pour tester notre code (le fichier complet est disponible sur le GitHub du projet) :

- **test_get_ip_address** : Cette fonction teste simplement si la fonction permettant à l'Hôte d'obtenir son adresse IP fonctionne correctement.
    - Vérification que l'IP contient 4 parties.
    - Vérification que ces parties sont numériques.

    - Vérification que les parties sont comprises entre 0 et 255.

- **test_backlog_loading** : Cette fonction teste le bon fonctionnement du chargement du backlog.
    - Génération d'un faux fichier backlog.
    - Utilisation de mocks.
    - Vérification que les données du backlog correspondent aux données utilisées dans le code.
- **test_client_connection** : Cette fonction teste le bon fonctionnement de la connexion d'un client à un hôte.
    - Utilisation de mocks.
    - Vérification que les mocks sont bien connectés.
- **test_vote_processing** : Cette fonction teste le bon fonctionnement de la collecte des votes côté Hôte.
    - Utilisation de mocks.
    - Utilisation d'une fausse liste de joueurs.
    - Test selon tous les modes de jeu implémentés.
    - Vérification, à l'aide d'un bloc try, que la collecte se déroule correctement.
- **test_server_initialization** : Cette fonction teste le bon fonctionnement de l'initialisation du serveur.
    - Utilisation de mocks.
    - Vérification que le port est bien défini et que l'hôte possède une liste de clients initialisée comme vide.

Veuillez noter l'utilisation des mocks, qui s'est avérée très utile dans le contexte de notre application. Cette notion, découverte lors de la mise en place des tests unitaires, repose sur le principe d'imitation. Si une fonction attend un type d'objet précis, le mock va imiter cet objet. Cela nous permet d'isoler correctement les fonctionnalités pour les tester sans avoir à mettre en place la configuration complète de l'application.

4.2 GENERATION DE LA DOCUMENTATION

La documentation est essentielle pour rendre le code plus clair. Pour quelqu'un qui n'a pas travaillé dessus, elle permet d'avoir une vision précise des classes, des méthodes, des attributs, ainsi que du fonctionnement général du code.

4.2.1 CHOIX DE L'OUTIL

Tout comme les tests unitaires, la génération de la documentation dépend du langage de programmation utilisé. Dans notre cas, nous avons utilisé l'outil Doxygen, qui permet de

générer automatiquement une documentation à partir des balises insérées dans le code lors du développement de l'application. Pour fonctionner correctement, Doxygen nécessite un fichier de configuration nommé Doxyfile. Ce fichier contient des informations telles que le nom du produit, le dossier de sortie où la documentation sera générée, le dossier contenant les fichiers à documenter, ainsi que d'autres paramètres pour configurer le traitement des fichiers de manière optimale.

4.2.2 STRATEGIE

Tout comme pour les tests unitaires, il est important de suivre l'avancement de la mise en place de la documentation automatique au fur et à mesure du développement du projet, afin de faciliter le travail et d'améliorer la qualité de la documentation.

4.3 UTILISATION DE GITHUB ACTIONS

Une fois les tests unitaires prêts en local, il a fallu automatiser cette tâche afin que, pour chaque commit sur le repository, les fonctions soient testées automatiquement. Cette automatisation a été rendue possible grâce à GitHub Actions, un outil de GitHub permettant d'automatiser des actions à l'aide de workflows configurés manuellement. Ce passage va donc expliquer la mise en place de ces workflows sur le repository.

4.3.1 CONFIGURATION GITHUB ACTIONS

Avant de mettre en place ces processus, il a été nécessaire de configurer GitHub Actions sur le repository. Cela passe par l'activation des permissions de read et write pour le workflow. Ensuite, il convient de créer un dossier .github/workflows dans lequel seront définies les procédures à suivre.

4.3.2 TESTS UNITAIRES

Pour les tests unitaires, il a fallu mettre en place un fichier au format .yml dans lequel nous spécifions la procédure à suivre lors du déclenchement de GitHub Actions :

- On précise la branche concernée (main).
- La version de l'OS (Ubuntu).
- La liste des actions à réaliser :
    - Effectuer un checkout du code.
    - Installer Python.
    - Installer pytest.
    - Exécuter les tests (pytest src/test.py).

Ce fichier, situé dans le dossier .github, sera exécuté à chaque déclenchement de GitHub Actions, ce qui permet d'exécuter les tests automatiquement à chaque commit.

4.3.3 DOXYGEN

La procédure pour Doxygen est bien différente. En effet, Doxygen a la particularité de générer une interface web permettant de parcourir la documentation. Tout comme pour les tests unitaires, Doxygen doit être déclenché via un fichier .yml dans le dossier .github/workflows. On y retrouve, tout comme pour les tests unitaires, la configuration suivante :

- Branche concernée (main)
- Version de l'OS (Ubuntu)
- Les étapes à réaliser :
    - Checkout du repository
    - Installation de Doxygen
    - Installation de Dot (pour le style de la page générée)
    - Localisation de la configuration Doxyfile (conf/Doxyfile)
    - Exécution de la génération de la documentation
    - Destination de la branche et du dossier de documentation (gh-pages)

Ensuite, Doxygen a besoin d'un fichier de configuration pour fonctionner. Le fichier .yml ne suffit pas, car il gère uniquement le lancement de la génération de la documentation. Il est donc nécessaire d'avoir un fichier qui indique clairement à Doxygen comment procéder. Dans notre cas, le fichier étant déjà configuré pour fonctionner localement, il suffira simplement de le déposer dans un répertoire appelé conf sur le repository.

Enfin, pour consulter la documentation générée par Doxygen, il est possible soit de télécharger les fichiers générés dans le dossier html dans la branche gh-pages, soit de configurer le repository dans les paramètres de GitHub Pages en précisant la branche gh-pages du repository et en indiquant la racine de la branche. Cela permet un accès direct via Internet grâce au lien généré par GitHub Pages (https://eliasbaroudi.github.io/projet-conception/html/index.html).

Tous les fichiers concernés sont évidemment présents sur le repository.