Odin Gabin
Lefaure Evan

Projet CAPI : Rapport

Université Lyon 2

I. PRÉSENTATION

A) Le Projet CAPI

Le projet CAPI est un projet réalisé dans le cadre de la matière “conceptions agiles de projets informatiques” du Master Informatique de l’université Lyon 2. Le projet consiste en la réalisation d’une application de planning poker, une pratique permettant de produire des estimations sur les efforts à fournir pour développer une fonctionnalité dans un projet informatique, en mettant l’accent sur l’utilisation des méthodes agiles étudiées le long du semestre. Le choix du langage de programmation est libre mais un certain nombre fonctionnalités est attendu.

B) Les Objectifs

L'objectif principal est de produire un code fonctionnel respectant la pratique de certaines méthodes agiles telles que l’utilisation de design patterns et l’implémentation de l’intégration continue. Ce projet apparaît comme “professionnalisant”, en effet, avec l’utilisation des méthodes agiles et le fait de travailler en binôme via un outil de gestion de versions le tout avec un délai à respecter nous met dans un cadre d’entreprise et nous permet donc de nous former sur les différents aspects du projet (langage Python, design pattern, intégration continue) tout en nous préparant au monde du travail.

II. LES CHOIX TECHNIQUES

A) Langage et outils

Pour la réalisation de ce projet nous avons décidé d'utiliser le langage de programmation Python car ce dernier possède une syntaxe claire, est facile de compréhension et possède une large documentation sur internet. De plus, nous utilisons Python dans un grand nombre de projets dans notre formation, cela fait donc sens d'utiliser ce langage plutôt que de se pencher sur des langages web que nous n'avons pas travaillé depuis un certain temps.

Pour l'outil de gestion de versions nous avons décidé d'utiliser la plateforme Github car nous connaissons le fonctionnement du logiciel git et que Github facilite l'implémentation de l'intégration continue, nous reviendrons sur ce point plus tard dans le rapport.

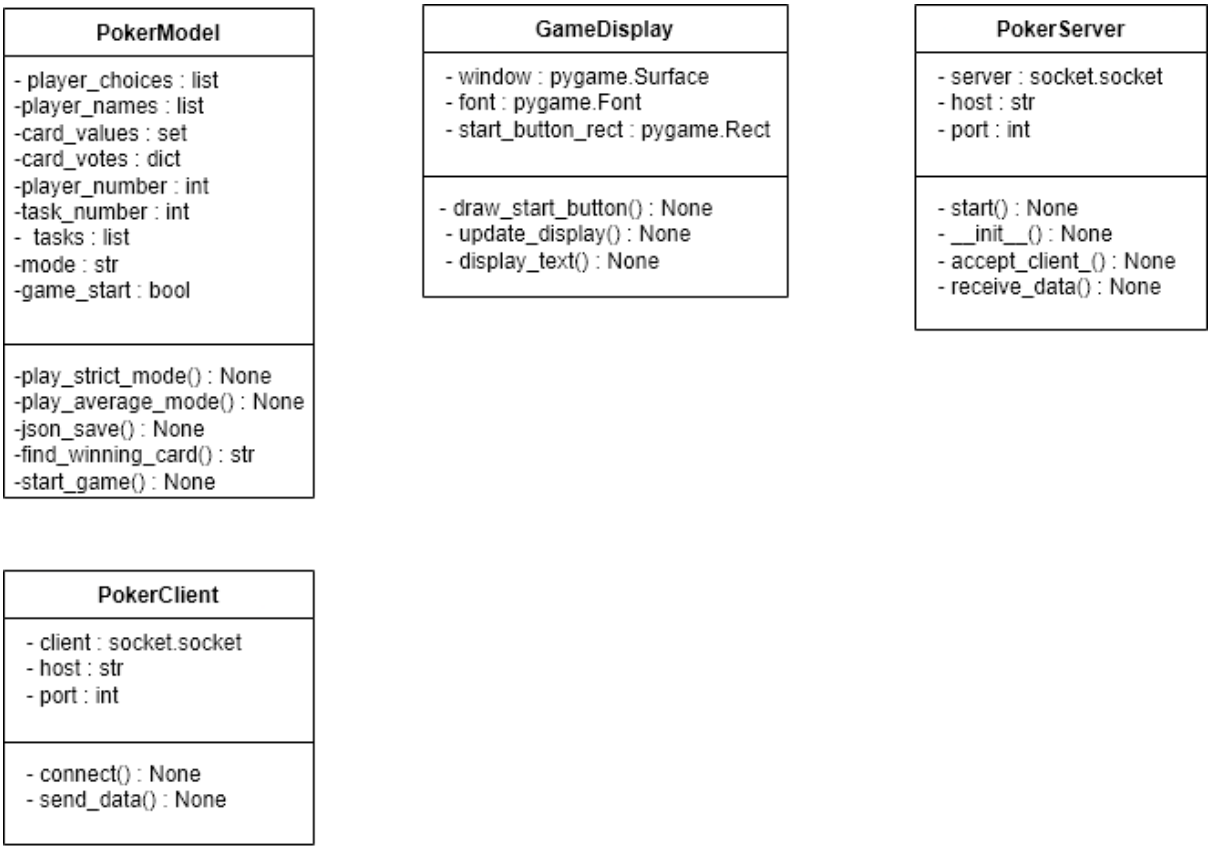
Au niveau de l'environnement de travail nous travaillons principalement sur Visual Studio Code qui permet une très bonne lisibilité du code et intègre les commandes git. L'un de nos PC ayant cessé de fonctionner lors du développement du projet l'un de nous à également travailler sur l'éditeur PyCharm.

B) Architecture et classes

Le code est structuré en 3 classes principales composées de plusieurs variables et méthodes: PokerModel (Model), GameDisplay (Vue) et GameController (Contrôleur). Nous avons donc mis en place une architecture MVC car c'est une méthode de travail que nous avions déjà pu étudier et cette architecture permet une organisation claire du code et permet également une grande flexibilité, il est facile d'ajouter ou modifier des fonctionnalités car chaque classe gère une partie différente de l'application. De plus, le modèle MVC est un patron de conception souvent utilisé lors de la pratique de méthodes agiles.

Le code du projet n'étant pas particulièrement long, nous avons décidé de placer les classes dans un fichier commun et non pas dans des fichiers séparés afin de faciliter la réalisation du projet.

Une dernière classe PokerServer permettant la gestion d'un serveur afin de permettre aux joueurs de réaliser une partie de planning poker en ligne est également implémentée, cette dernière suit le même format que les autres classes du projet.



C) L'interface

Il est important de faire un point sur l'interface de notre projet, en effet nous avons mal abordé cette partie du développement et nous avons donc fait le choix de mettre de côté cette partie pour nous concentrer sur l'essentiel : la production d'un code fonctionnel en utilisant les méthodes agiles. Nous avons décidé d'utiliser la bibliothèque PyGame permettant la création de petit jeu via Python mais la prise en main était assez complexe et nous n'avions jamais utilisé cette librairie. Nous aurions dû mieux aborder cette partie en effectuant plus de recherche, l'utilisation de la librairie TKinter aurait sûrement été une meilleure solution pour nous.
Le jeu se déroule donc essentiellement via la console de l'environnement de travail.

III. Les Patterns

A) Le Modèle MVC

Comme expliqué au sein de la partie “Architecture” du rapport, notre code repose essentiellement sur l’utilisation du pattern **Modèle-Vue-Contrôleur**, ce dernier a été adopté pour organiser et structurer l’application. Cette séparation claire des responsabilités permet une gestion flexible du code : le modèle gère la logique métier et contient les méthodes principales, la vue gère l’interface utilisateur et le contrôleur coordonne les interactions entre les deux. L’utilisation de ce pattern nous paraissait évidente pour ce genre de projet sachant qu’il permet également de faciliter la mise en place des tests unitaires.

B) Observer

Le design pattern **Observer** a été étudié lors des cours de conceptions agiles et permet de gérer les événements de l’interface utilisateur. La méthode “handle_events()” de la classe “GameController” observe les événements PyGame et déclenche des actions en réponse aux événements (saisie, entrée, etc.). L’utilisation de ce pattern nous a permis de créer un système réactif où les actions des utilisateurs sont traitées efficacement afin de leur offrir la meilleure expérience possible.

C) Strategy

Une partie de notre code repose sur le pattern **Strategy** : le choix du mode de jeu. En effet, la méthode “start_game()” de la classe “PokerModel” permet de choisir entre deux modes de jeu distincts (mode stricte et mode moyenne) en fonction du choix de l’utilisateur. Ce n’est pas une implémentation totale du pattern Strategy mais on retrouve la base du pattern : la sélection d’une stratégie basée sur une condition externe, ici le choix de l’utilisateur. Le choix de ce pattern pour cette partie du projet fait sens car il permet de modifier le fonctionnement d’un projet via les choix utilisateurs.

IV. INTEGRATION CONTINUE

A) Documentation et Intégration Continue :

Concernant l'intégration continue, nous avons mis en place le processus en définissant un flux de travail (workflow) dans GitHub Actions. Ce workflow inclut des étapes pour exécuter les tests unitaires à chaque modification du code, ainsi que pour générer automatiquement la documentation à l'aide de Doxygen. Cela garantit la vérification constante du code et la génération régulière de la documentation du projet.
Pour documenter notre projet avec Doxygen, nous initions une configuration via un fichier Doxyfile dédié. Cela nous permet de préciser les paramètres essentiels à la génération de la documentation, en prenant soin d'inclure les détails spécifiques à notre code Python.

Une grande attention a été portée à documenter le code pour qu'il soit compréhensible et facile à entretenir à l'avenir. Les commentaires ont été rédigés avec soin pour expliquer les fonctionnalités, les structures de données et les algorithmes utilisés.
Dans notre flux de travail (workflow) GitHub Actions, on installe Doxygen et on configure une étape dédiée à la génération automatique de la documentation à chaque modification du code.


Code Blame 30 lines (22 loc) · 588 Bytes

```yaml
name: Generate Doxygen Documentation

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Install Doxygen
      run: sudo apt-get install doxygen -y

    - name : Install Dot
      run : sudo apt-get install graphviz -y

    - name: Generate Documentation
      run: doxygen conf/Doxyfile

    - name: Deploy Documentation

      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./documentation
```

B) Tests unitaires

Nous avons ajouté une autre étape au flux de travail GitHub Actions pour exécuter les tests unitaires. Cette étape peut installer les dépendances de test et exécuter les commandes pour lancer les tests.

En intégrant ces étapes dans notre workflow GitHub, on assure que la documentation est régulièrement mise à jour et que les tests unitaires sont exécutés automatiquement à chaque modification du code, garantissant ainsi une meilleure qualité et fiabilité du projet.