Rapport final du projet d'informatique

Création d'une API REST

Groupe 23 :
Hugo BERNARD
Zacharie BOUHIN
Clément GABAS
Mael GUIVARCH
Romane PARÈS

Encadrant :
Thierry MATHE

Année universitaire 2020 - 2021

Sommaire

Introduction

Dans le cadre de nos études en deuxième année à l'ENSAI, nous avons été amenés à réaliser un projet informatique. Le thème qui nous a été imposé est la création d'une API REST.

Une API, pour *"Application Programming Interface"*, ou *"Interface de Programmation"* en français, est un objet informatique dont le rôle est d'interagir avec un système tiers. Provenant de *"REpresentational State Transfer"*, l'architecture REST est un type d'architecture informatique sans état *(stateless)* pour laquelle les requêtes utilisent les méthodes HTTP traditionnelles (GET, POST, PUT, ...) tandis que les réponses à ces requêtes sont formatées (XML, JSON, ...).
Les API REST sont au cœur de l'architecture client-serveur, système dominant sur le Web. Elles permettent notamment d'établir et de normaliser les dialogues entre clients et bases de données.

Pour ce projet nous nous proposons de réaliser une application permettant à des clients de jouer à des jeux au travers d'une API REST et d'une base de données. Nous avons retenu le jeu du Puissance 4 et le Jeu de l'Oie. Ces deux jeux possèdent des caractéristiques suffisamment différentes pour mettre en évidence la flexibilité de notre application. La polyvalence de notre application est un objectif essentiel de ce projet, nous devons nous assurer que la structure de notre application soit capable d'accueillir tout type de nouveaux jeux et que leur implémentation soit simple.

Afin d'enrichir notre application et d'offrir plus de confort aux utilisateurs, nous avons choisi d'ajouter des fonctionnalités supplémentaires. Ainsi en plus de pouvoir jouer aux jeux proposés, les utilisateurs peuvent également accéder à leur profil afin de modifier leurs informations, ajouter des utilisateurs en tant qu'amis, afficher le classement général, le classement d'un jeu en particulier ou encore afficher leurs statistiques personnelles. Ils peuvent ainsi décider de jouer une partie entre amis ou d'affronter des inconnus afin d'augmenter leur score dans un jeu donné.

Pour des informations complémentaires et plus synthétiques, veuillez télécharger le projet en suivant la méthode ci-dessous et lire le fichier ***README.md*** associé.

Pour télécharger le projet, veuillez effectuer un git clone à l'adresse ci-dessous :

```
$ git clone https://github.com/clementgabas/Projet-Info
```

Pour plus d'informations sur git et pour savoir comment le télécharger, veuillez vous référer au site officiel.

Pour comprendre comment jouer à SteamEnsai, veuillez lire le guide d'utilisation disponible en annexe (cf **[B]**).

1 Présentation du projet

1.1 Cahier des charges

L'objectif de ce projet est de développer une API qui respecte l'architecture REST, permettant aux utilisateurs de jouer au jeu de l'oie ou au puissance 4. De plus nous avons choisi d'ajouter des fonctionnalités supplémentaires, rendant notre application plus complète. Pour atteindre ces objectifs nous avons établi un cahier des charges. La liste ci-dessous répertorie l'ensemble des fonctionnalités à implémenter afin de répondre au sujet qui nous a été donné :

— Développer une API respectant l'architecture REST.

— Créer une base de données et coder les méthodes permettant d'opérer sur cette base de données. La base de données de notre projet à été conçue de manière à être suffisamment flexible pour pouvoir accueillir de nouveaux jeux sans modifications majeures de celle-ci.

— Implémenter le jeu du puissance 4 et le jeu de l'oie en Python en respectant une architecture logique qui permet son implémentation à travers l'API pour le que le client puisse jouer.

La liste ci-dessous, répertorie quant à elle l'ensemble des fonctionnalités supplémentaires que nous nous sommes proposés d'ajouter :

— Créer et gérer un système de comptes utilisateurs. Ainsi, nous avons développé un système de création de compte, de protection et de stockage des informations relatives à ces comptes en base de données (notamment via le hashage de mots de passe), et un système d'authentification.

— Implémenter un système d'amis et de classement pour les utilisateurs disposant d'un compte.

— Permettre à un utilisateur de jouer avec ses amis ou bien d'affronter des inconnus.

— Permettre la modification des paramètres de jeu. Les utilisateurs qui le souhaitent peuvent modifier les règles du puissance 4 ou du jeu de l'oie pour jouer à des parties personnalisées lorsqu'ils jouent avec leurs amis. [1]

1.2 Organisation du travail

Il est essentiel de répartir le travail dans le temps, mais également entre les différents membres du projet afin de veiller au bon fonctionnement de ce dernier.

Pour ce faire, nous avons travaillé selon deux méthodes. Au cours de la phase de conception, tout le monde travaillait sur le même sujet afin que chacun puisse y apporter son point de vue et que le résultat final soit le fruit de la collaboration de chacun.

Puis, au cours de la phase de développement, le chef de projet répartissait les rôles à chacun en fonction de leurs préférences. En effet, le but était d'avancer le mieux possible et de garder une certaines continuité logique dans le code. Ainsi, si quelqu'un codait les menus pendant une semaine sans avoir fini, il était reconduit, car il savait déjà sur quoi il travaillait et connaissait donc

1. ⚠ Nous n'avons malheuresement pas encore terminé d'implémenter cette fonctionnalité...

l'architecture logique de son code. Nous expliquons plus en détail notre démarche dans la suite du rapport (cf [1.2.2]).

1.2.1 Répartition du travail dans le temps

Afin d'être le plus efficace dans la réalisation de ce projet, nous nous sommes organisés en deux temps :

— **La phase de conception :** Il s'agit d'une phase qui a duré un mois, de septembre à octobre. Au cours de cette première étape, nous avons réfléchi à l'architecture générale de l'application. Il s'agissait donc de cerner les attendus de ce projet et de proposer une modélisation qui y réponde le mieux possible. Cela s'est traduit en pratique par la réalisation de plusieurs diagrammes détaillant le fonctionnement de notre application et sa structure. Nous avons également au cours de cette phase défini le temps alloué aux différentes tâches nécessaires à la réalisation de ce projet, ainsi que l'organisation interne de notre groupe. Cette première étape s'est soldée par la remise du rapport d'analyse le 11 octobre 2020.

— **La phase de développement** : Cette phase a débuté à la mi-octobre et s'est poursuivie jusqu'à la fin novembre. Au cours de cette période, nous avons codé et implémenté tous les éléments nécessaires au fonctionnement de notre application. C'est dans ce laps de temps, que s'est déroulée la période d'immersion, du 4 au 6 novembre. Cette période nous a offert le temps nécessaire afin de pouvoir travailler tous ensembles sur le projet et donc d'avancer significativement dans ce dernier.

1.2.2 Répartition des tâches

Bien qu'il y ait un chef de projet, la hiérarchie adoptée par notre groupe n'est pas réellement verticale. Tous les membres sont, plus ou moins, au même niveau, même si le chef de projet est celui qui tranchera en cas de désaccord.

De plus, bien que nous ayons réalisé un diagramme de Gantt théorique, nous ne nous y sommes pas toujours tenu. En effet, chacun ayant son domaine de prédilection vis à vis du sujet, le chef de projet a réparti les tâches pour que chacun puisse coder dans le domaine où il est le plus à l'aise, tout en prenant toujours soin d'aider et d'expliquer lorsqu'un membre du groupe rencontrait une difficulté. Cette répartition des tâches se faisait de manière très régulière avec des réunions quasi-hebdomadaires où l'avancée de chacun était observée et les nouvelles tâches délivrées tout en prenant bien soin de définir les priorités parmi ces nouvelles tâches.

La réalisation de ces tâches a pu être faite de manière efficace grâce à l'utilisation de git, et, dans notre cas, d'un dépot github. En effet, bien que nous ayons eu quelques difficultés à le prendre en main au départ, notamment car nous exécutions la commande *git add .* sur la branche *master* et donc avions des conflits au moment de *push* nos *commit*, ce dernier s'est avéré très pratique afin de travailler à plusieurs sur le projet simultanément, en particulier lors de la période d'immersion. Grâce à ce dépôt, nous avons pu chacun travailler sur la partie sur laquelle nous étions le plus à l'aise sans gêner les autres membres du groupe. De plus, en exécutant la commande *git push* régulièrement, le chef du groupe pouvait prendre conscience de l'avancée de chacun et

donc observer de potentielles lacunes ou, au contraire, de bonnes idées à réappliquer dans le reste du code. Git nous a permis de ne pas démultiplier les différentes versions de notre code, et donc de toujours travailler sur la version la plus récente de notre travail. De plus, nous avons pu récupérer d'anciennes versions du code, ce qui nous a été très utile lors de la période d'immersion, ainsi qu'un jour lorsqu'une erreur avec un *git rm* a supprimé l'ensemble du code sur le dépôt ...

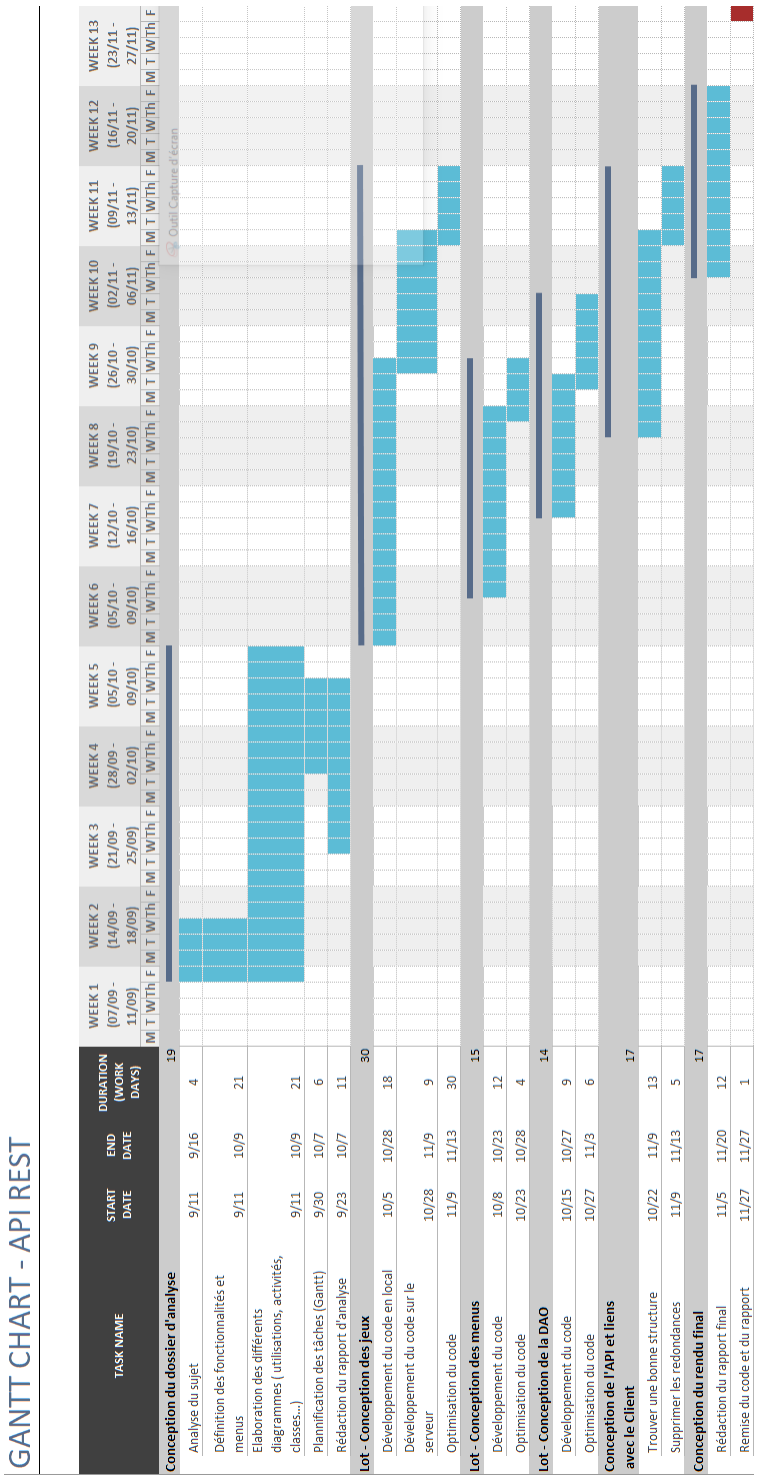

FIGURE 1 – Diagramme de Gantt

2 Modélisation

2.1 Diagramme de cas d'utilisation

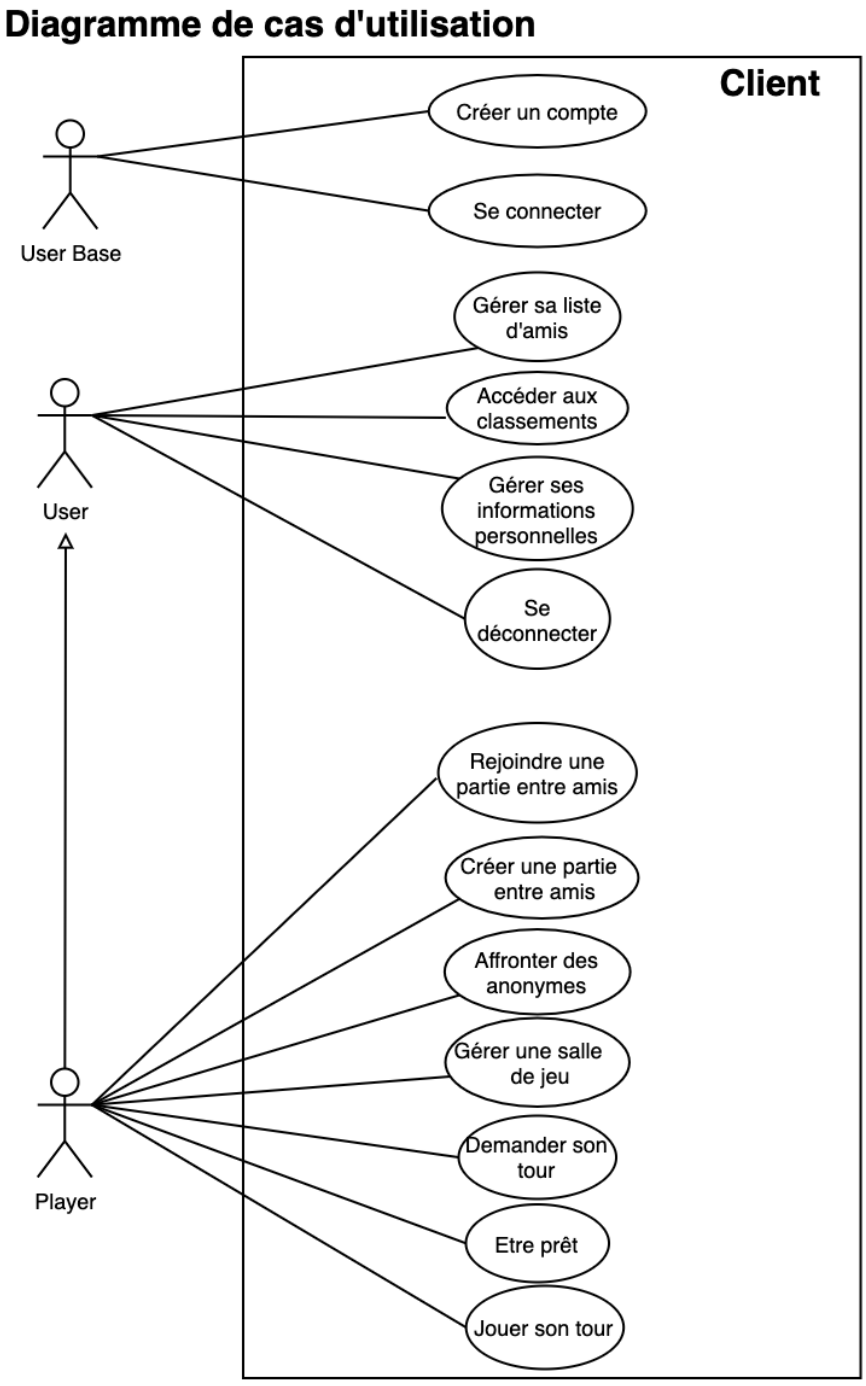

FIGURE 2 – Diagramme de cas d'utilisation de notre application

2.1.1 Point de vue de l'utilisateur connecté

Si le client se connecte, ou s'inscrit puis se connecte, il est désormais considéré comme un utilisateur connecté représenté par le User sur la figure 2.1. Il a donc accès au menu des utilisateurs connectés à partir duquel plusieurs options sont disponibles :

— `Se déconnecter` : ce qui le fait revenir au menu initial.

— `Accéder au profil` : il peut choisir d'accéder au profil, ce qui lui permettra par la suite de :

  — `Gérer ses informations personnelles` : il peut ainsi modifier ses informations telles que le mot de passe ou le pseudo, et même réinitialiser ses statistiques personnelles, s'il le souhaite.

  — `Gérer sa liste d'amis` : il peut ainsi afficher sa liste d'ami, ajouter des amis ou en supprimer.

  — `Accéder aux classement` : le client peut voir le classement général mais aussi le classement dans chaque jeu et également le classement de ses amis. Il peut également consulter ses statistiques personnelles.

  — `Jouer :` L'utilisateur connecté peut décider de jouer à l'un des jeux. Il devient alors un joueur, comme représenté sur la figure 2.1 par player.

Une fois que l'utilisateur a choisi de jouer, il doit choisir le jeu auquel il souhaite jouer puis décider s'il choisit de jouer contre des anonymes ou contre des amis. S'il décide d'affronter des joueurs anonymes, il rejoindra automatiquement une salle de jeu ou en créera une si aucune n'est disponible. S'il choisit de jouer contre des amis, il pourra choisir de créer ou de rejoindre une salle comme décrit ci-dessous

— `Créer une salle` : le joueur peut créer sa propre salle en choisissant préalablement le jeu auquel il souhaite jouer. Il peut alors inviter ses amis à l'aide d'un numéro de salle.

— `Rejoindre une salle :` le joueur peut également rejoindre la salle de l'un de ses amis à l'aide d'un numéro de salle.

Dans tous les cas une fois dans le salon de jeu, les utilisateurs, qu'ils disposent d'un compte ou non, pourront :

— `Afficher les membres de la salle :` Permet d'afficher le pseudo des autres joueurs présents dans la salle de jeu.

— `Être prêt :` Permet d'abord de choisir la couleur de son/ses pion(s) puis d'obtenir le statut "être prêt". Une fois que tous les joueurs sont prêts la partie se lance.

— `Quitter la salle :` Permet de quitter la salle. Le chef de la salle ne peut la quitter que lorsque tous les autres joueurs l'ont quittée.

Une fois que tous les joueurs sont prêts la partie se lance, les joueurs sont alors en partie et peuvent :

— `Jouer leur tour :` Selon le jeu auquel le joueur joue, il peut s'agir de lancer les dés, ou de poser son pion.

— `Demander leur tour` : Cette action n'est pas réalisée par l'utilisateur, elle se fait automatiquement. Chaque utilisateur requête sans le savoir régulièrement l'API pour savoir si c'est à son tour de jouer.

2.1.2 Point de vue de l'utilisateur anonyme

Le client qui ne se connecte pas, ni ne s'inscrit, ne possède que deux options :

— `Jouer en tant que anonyme` : Il peut alors jouer au jeu de son choix, à l'aide d'un compte temporaire qui lui est fourni automatiquement. Une fois la partie terminée, il est renvoyé sur le menu des utilisateurs anonymes.

— `Quitter` : Il peut choisir de quitter l'application

2.2 Architecture générale de l'application

2.2.1 Architecture d'une partie

Nous allons ici comprendre comment se déroule une partie au niveau architectural.

A) Lorsque les joueurs rejoignent une partie, afin que le chef de salle puisse lancer la partie, tous les joueurs doivent se définir comme prêts via le bouton ***Être prêt.*** Une fois que tous les joueurs de la salle sont prêts, la partie se lance.

B) Lorsqu'une partie est lancée, les utilisateurs requêtent toutes les demi-secondes l'API pour savoir si c'est leur tour de jouer ou non. Tant que ce n'est pas leur tour, ils continuent de requêter.

C) Si c'est le tour du joueur, il récupère d'abord la grille (ou le plateau) de jeu pour l'afficher. Ainsi, l'utilisateur peut observer l'état de la partie. Ensuite, il joue son tour en précisant son action. Cette action est transmise au serveur qui juge si elle est valide ou non.

   1. Si le coup n'est pas valide, le joueur recommence son tour au début.
   2. Si le coup est valide, il est enregistré en base de données.

   Ensuite, il récupère la grille une nouvelle fois pour l'afficher. Son tour est fini, il commence alors à requêter le serveur jusqu'à obtenir la permission de jouer.

D) Notons qu'à chaque fois que la grille est récupérée, il est d'abord vérifié si oui ou non c'est une grille gagnante. Si la grille récupérée au début du tour est gagnante, un autre joueur a remporté la partie. Si en revanche une grille gagnante est récupérée en fin de tour, cela signifie que vous avez gagné !

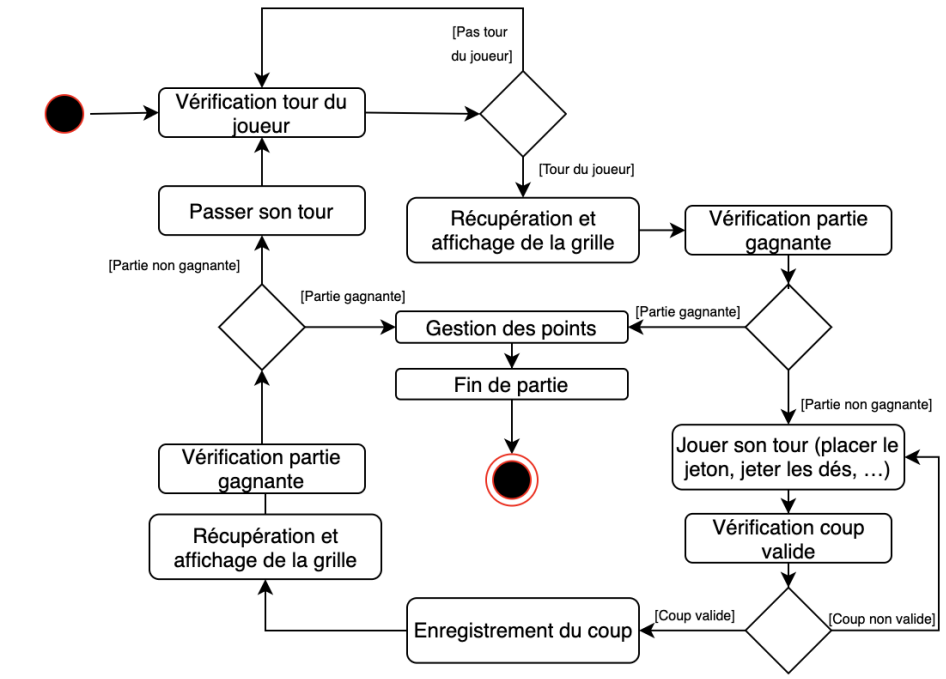

FIGURE 3 – Diagramme d'activité d'une partie de jeu

Lorsqu'une partie se termine, les scores de chaque joueur sont mis à jour grâce à la règle suivante :

$$\text{newScore} = \text{oldScore} + 20 \times \mathbb{1}_{vousavezgagne} - 10 \times \mathbb{1}_{vousavezperdu}$$

Nous avons arbitrairement choisi d'appliquer cette formule pour les parties entre amis comme pour les parties contre des inconnus mais cela pourrait être amené à changer si nécessaire. La phase de jeu est détaillée dans le diagramme de séquence, figure 4.

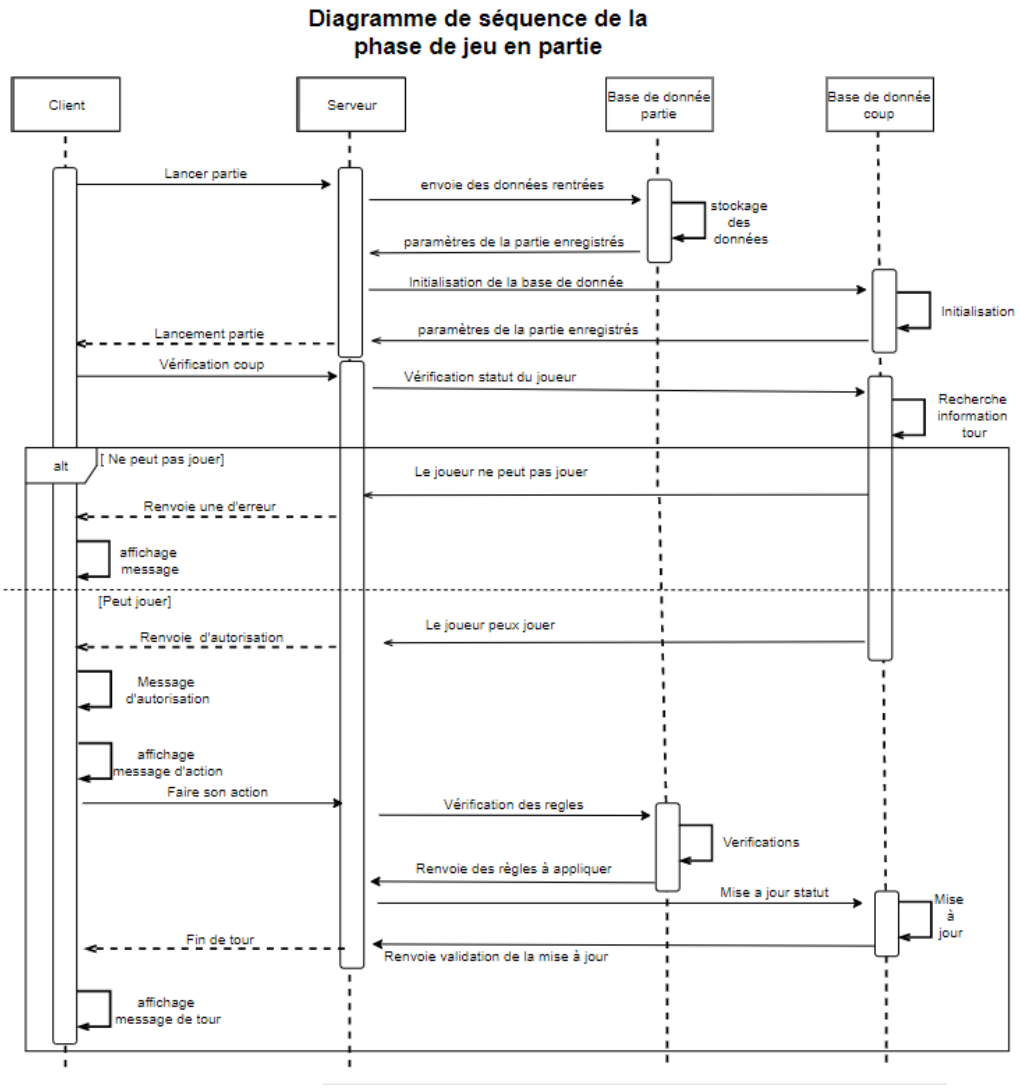

FIGURE 4 – Diagramme de séquence d'une partie de jeu

2.2.2 Architecture des fonctionnalités supplémentaires

Afin d'enrichir les fonctionnalités disponibles pour les utilisateurs, nous avons choisi d'ajouter des fonctionnalités supplémentaires, telles que la mise en place de classement entre les joueurs, la création d'un système d'amis entre joueurs et la possibilité pour les utilisateurs de modifier leurs informations personnelles. Nous allons ici détailler l'architecture logique de notre programme par quelques diagrammes UML.

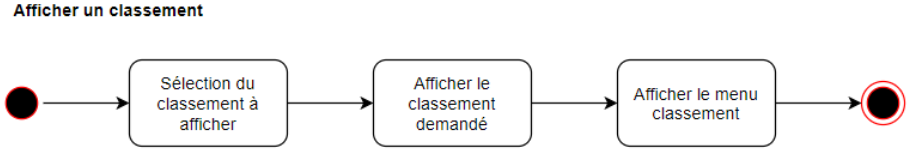

FIGURE 5 – Diagramme d'activité de l'affichage d'un classement.

Le diagramme d'activité ci-dessus représente les étapes successives faites lorsqu'un utilisateur souhaite afficher un classement. Ces étapes sont les mêmes lorsqu'il s'agit d'une liste d'amis, de paramètres de parties ou de statistiques personnelles.

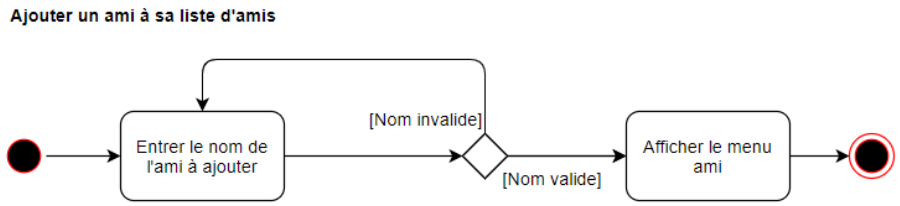

FIGURE 6 – Diagramme d'activité de l'ajout d'un ami.

De même, le diagramme d'activité ci-dessus représente les étapes successives nécessaires à l'ajout d'un ami. La suppression d'un ami repose sur le même principe.

Le diagramme de séquence ci-dessous représente les interactions entre clients, serveur et base de données lors de l'ajout d'un utilisateur en tant que ami.

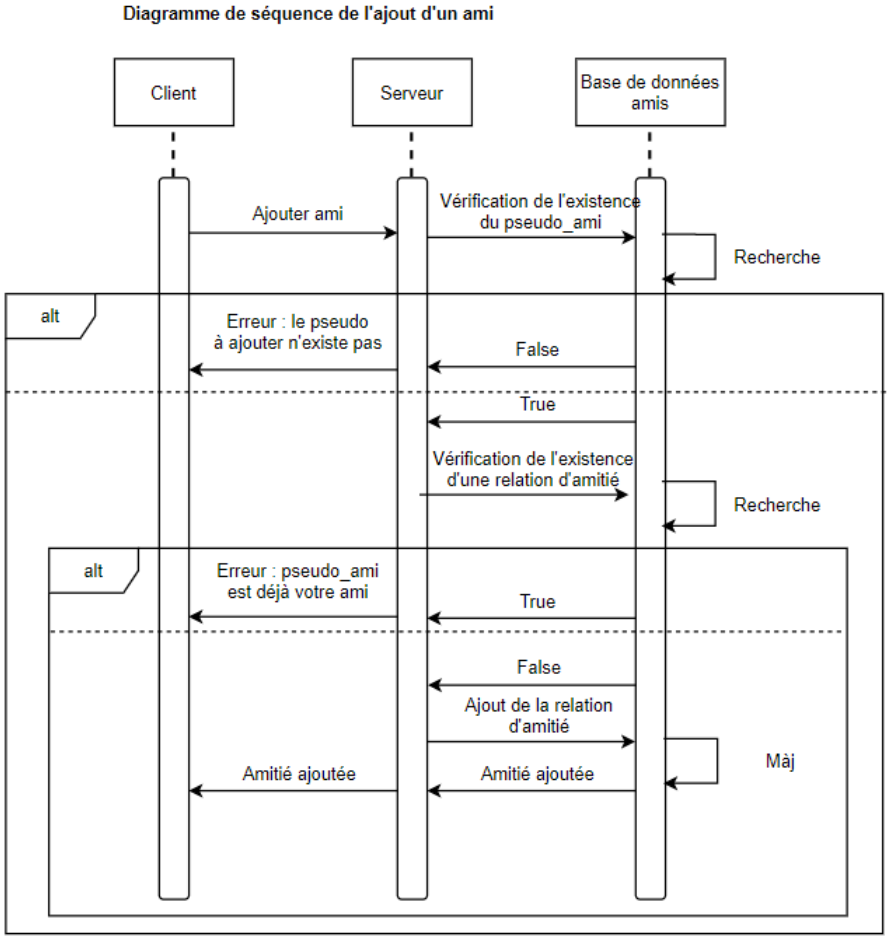

FIGURE 7 – Diagramme de séquence de l'ajout d'un ami.

Comme indiqué sur le diagramme, lorsque le client souhaite ajouter un ami dans sa liste d'amis, il transmet au serveur le pseudo de l'ami qu'il souhaite ajouter. Via la DAO, le serveur regarde si ce pseudo existe. S'il n'existe pas, une erreur est renvoyée. S'il existe, il est ensuite vérifié si un lien d'amitié n'existe pas déjà entre les deux utilisateurs. Si un lien d'amitié existe déjà, une erreur est renvoyée. Enfin, s'ils ne sont pas déjà amis, leur amitié est créée. La suppression d'un ami fonctionne sur le même principe.

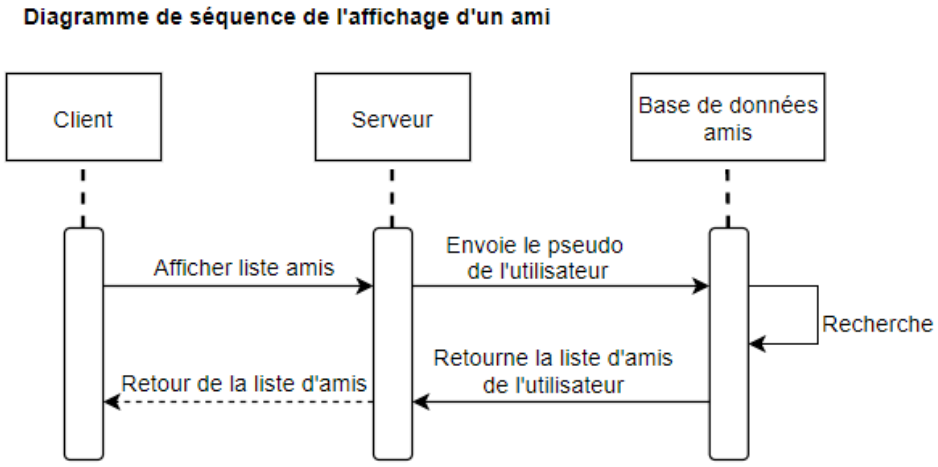

FIGURE 8 – Diagramme de séquence de l'affichage de la liste d'amis.

Le diagramme de séquence ci-dessus représente quant à lui les interactions entre client, serveur et base de données, lors de l'affichage de la liste d'amis d'un utilisateur. Ces interactions sont similaires lorsque l'on affiche le classement d'un utilisateur bien que le client envoie également le nom du jeu qu'il souhaite afficher. Le client communique les informations nécessaires à la recherche du contenu à afficher et les envoie au serveur, puis le serveur se sert de la DAO pour rechercher ce contenu dans la base de données. Le contenu est alors retourné au serveur et affiché au client.

2.3 Diagrammes relatifs à la base de données.

Au cours de ce projet, nous allons devoir conserver durablement des informations relatives aux utilisateurs de notre application, ou conserver temporairement des informations sur les parties en cours. Pour cela, nous allons utiliser une base de données SQL qui contiendra des tables comportant elles-mêmes toutes les informations nécessaires au bon fonctionnement de notre application. Afin de pouvoir communiquer avec cette base de données, nous serons amenés à mettre en place des modules dont la fonction première sera de faire l'interface entre les données et notre application. Nous avons cherché à rendre la base de données souple afin de faciliter l'ajout de nouveaux jeux par la suite. L'architecture et les relations de notre base de données seront explicitées dans les prochaines sections.

2.3.1 Diagramme entité relation

Le diagramme ci-dessous représente les tables de notre application, les champs qui les constituent ainsi que les liaisons entre tables. Ce diagramme est similaire aux modèle physique de données. Nous tâcherons néanmoins de nous focaliser davantage sur les relations et cardinalités entre les tables de notre base de données.

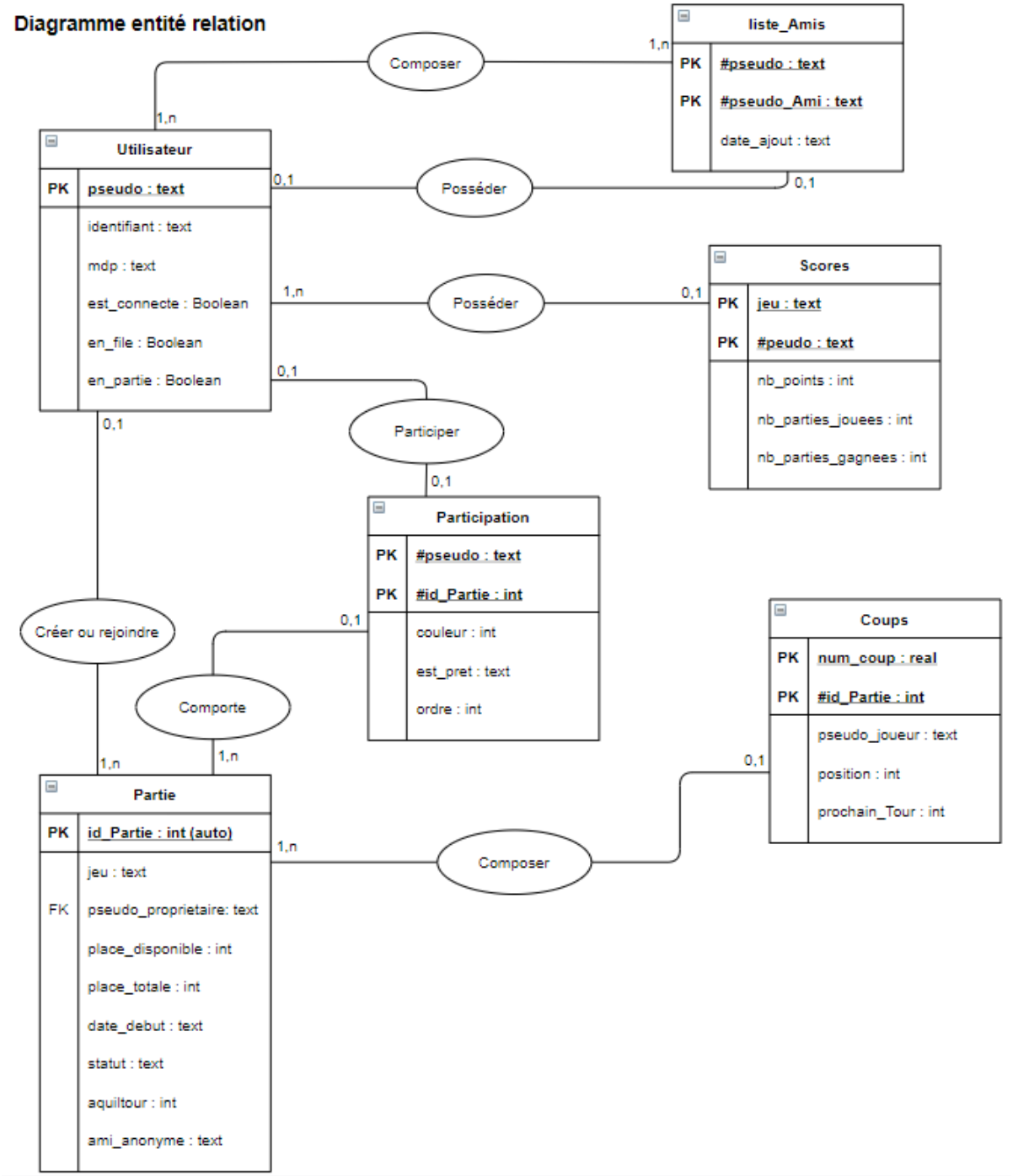

FIGURE 9 – Diagramme entité-relation de notre application.

Chaque utilisateur dispose d'une unique liste d'amis mais peut être présent dans les listes d'amis de plusieurs autres joueurs. La table `Utilisateur` est donc en relation avec la table `liste_Amis` Elle est également en relation avec la table `Partie` car les utilisateurs peuvent créer ou rejoindre une seule partie à la fois, mais une partie comporte plusieurs utilisateurs. La table `Score` est associée à la table `Utilisateur`. En effet, chaque utilisateur possède autant de scores qu'il y a de jeux.

La table `Participation` est associée à la table `Utilisateur`, car un utilisateur ne peut participer qu'à une seule partie à la fois. Cependant il y a plusieurs participations dans une partie, ce qui explique la relation entre la table `Participation` et la table `Partie`.

La table `Partie` est en relation avec la table `Coup`. En effet, une partie est composée de plusieurs coups, mais chaque coup est propre à une partie.

2.3.2 Modèle physique de données

Dans cette sous-section, le contenu de chacune de ces tables est décrit, et les différentes relations entre celles-ci sont symbolisées par les clés primaires et étrangères.

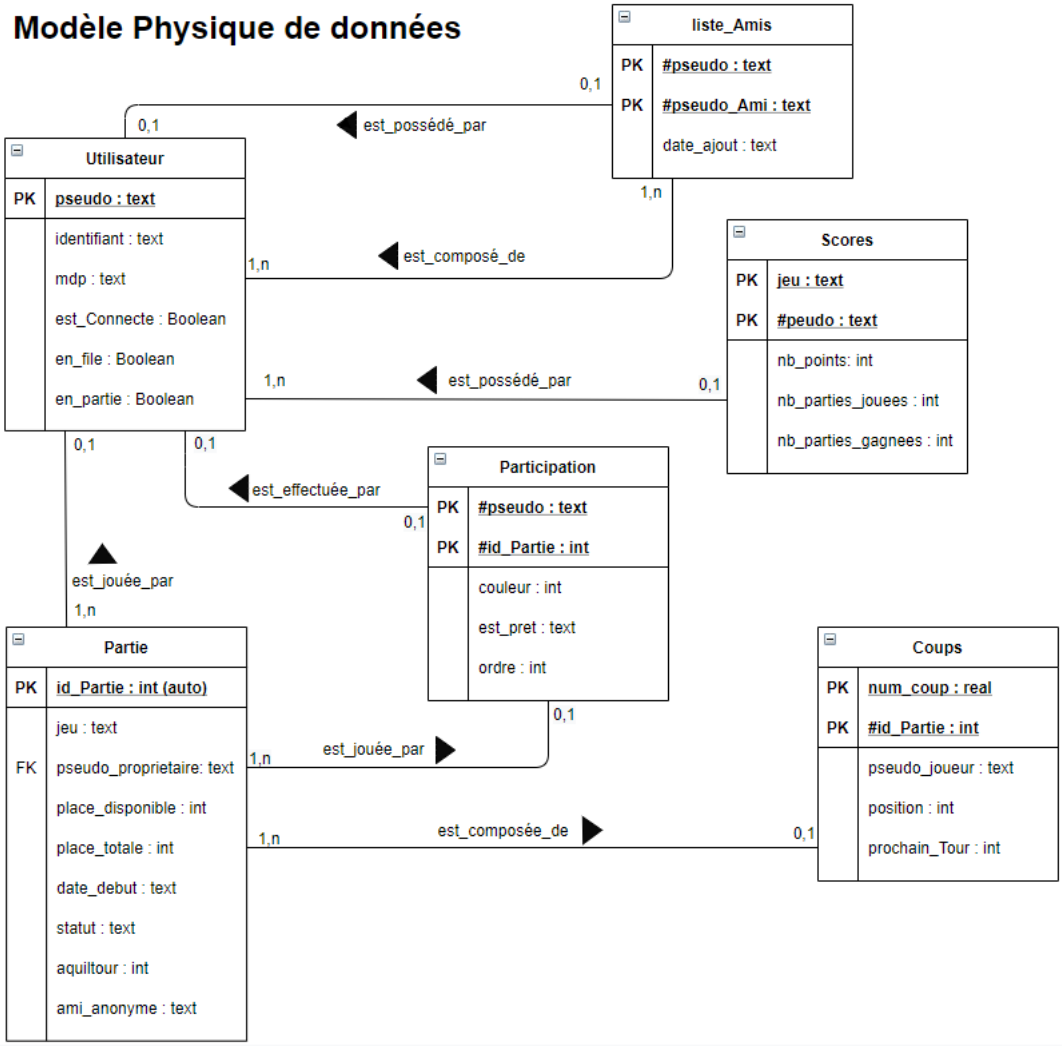

FIGURE 10 – Diagramme entité-relation de notre application.

Table Utilisateur

— `Pseudo` : Ce champ est la clé primaire de la table `Utilisateur`. Il assure donc l'unicité des utilisateurs dans la base et sera utilisé pour récupérer leurs informations.

— `identfiant` : Ce champ correspond à l'identifiant utilisé par l'utilisateur pour se connecter à son compte (c'est aussi un chaîne de caractères). Pour des raisons techniques évidentes telles que la connexion ou la modification de pseudonymes, lui aussi unique. Nous aurions pu choisir de l'utiliser comme clé primaire, mais nous avons privilégié l'association de pseudonymes libres et d'un identifiant à garder secret pour sécuriser la connexion.

— `mot_de_passe` : Il s'agit d'une chaîne de caractères correspondant au hash du mot de passe utilisé par les utilisateurs afin de se connecter à leur compte.

— `est_Connecte` : Ce champ est un Booléen indiquant si l'utilisateur est connecté ou non.

— `en_file` : Ce champ est un Booléen indiquant si l'utilisateur cherche à joindre une partie ou non.

— `en_partie` : Ce champ est un Booléen indiquant si l'utilisateur est en partie ou non.

Table liste_Amis

— `pseudo` et `pseudo_Ami` : Il s'agit du couple de champs qui constitue la clé primaire de la table `liste_Amis`. Ceci permet d'assurer l'unicité du couple pseudo et pseudo_ami, tout en permettant à un pseudo ou à un pseudo_Ami d'apparaître plusieurs fois dans la table. De plus le pseudo est une clé étrangère qui fait référence à la table utilisateur, afin de s'assurer qu'une liste d'amis soit toujours associée à un utilisateur.

— `pseudo_Ami` : Il s'agit d'une chaîne de caractères correspondant au pseudo d'un utilisateur figurant dans la liste d'amis d'un autre utilisateur.

— `date_Ajout` : Il s'agit d'une chaîne de caractères correspondant à la date d'ajout d'un ami dans la liste d'amis.

Table Scores

— `Jeu` et `pseudo` : Ce couple constitue la clé primaire de la table `Scores` et permet de s'assurer qu'un utilisateur possède toujours un unique score sur chaque jeu. De plus, le champ pseudo est également une clé étrangère vers la table Utilisateur permettant ainsi de s'assurer que les scores soient toujours définis pour un utilisateur précis sur chacun des jeux.

— `nbr_parties_jouees` : Il s'agit d'un entier correspondant au nombre de parties jouées par l'utilisateur.

— `nbr_parties_gagnees` : Il s'agit d'un entier correspondant au nombre de parties gagnées par l'utilisateur.

— `nb_points` : Il s'agit d'un entier correspondant au score d'un utilisateur sur un jeu. Le score total d'un utilisateur peut être calculé en sommant les nb_points d'un utilisateur sur chacun de ses jeux.

Table Participation

— `pseudo` et `id_Partie` : Il s'agit d'un couple qui est la clé primaire de la table `Participation` et s'assure donc de l'unicité de la participation d'un utilisateur à une partie.

— `couleur` : C'est un entier correspondant à la couleur des pions du joueur.

— `est_pret` : Il s'agit d'une chaîne de caractères correspondant au statut du joueur dans le salon et indiquant s'il est prêt à commencer la partie.

— `ordre` : Il s'agit d'un entier correspondant à l'ordre dans lequel les joueurs vont jouer leur tour une fois en partie.

Table Partie

— `id_Partie` : Il s'agit de la clé primaire de la table `Partie`. C'est un entier qui s'auto-incrémente, chaque id_Partie est associé à une unique partie.

— `jeu` : Il s'agit d'une chaîne de caractères correspondant à l'identifiant du jeu sur lequel se déroule la partie.

— `pseudo_proprietaire` : Il s'agit du pseudo du propriétaire de la partie.

— `place_disponible` : Il s'agit du nombre de places restantes dans une partie.

— `place_total` : Il s'agit du nombre de places total dans une partie.

— `date_début` : Il s'agit d'une chaîne de caractère indiquant la date de début de la partie.

— `statut` :

— `aquiltour` : Il s'agit d'un entier indiquant le joueur dont c'est le tour.

— `ami_anonyme` : Il s'agit d'une chaîne de caractères indiquant si la partie est entre amis ou avec un anonyme.

Table Coups

— `num_coup` : Il s'agit de la clé primaire de la table `Coups`. Ce champ est un réel correspondant au numéro du coup joué par un joueur dans sa partie.

— `id_Partie` : Il s'agit d'une clé étrangère vers la table `Partie` s'assurant qu'un coup est toujours associé à une partie.

— `pseudo_Joueur` : Ce champ correspond au pseudo du joueur qui a joué le coup.

— `position` : Il s'agit d'un entier correspondant à la position du joueur sur le plateau de jeu.

— `prochain_Tour` : Il s'agit d'un entier permettant de savoir si le joueur est autorisé à jouer son prochain tour ou non.

2.4 Diagrammes de packages

Lors de ce projet, nous avons cherché à diviser notre code en plusieurs couches ayant chacune un rôle spécifique. L'avantage est de pouvoir facilement apporter des modifications au code tout en limitant le risque d'erreur.

Sur le diagramme ci-dessous nous avons représenté le diagramme de packages de notre application.

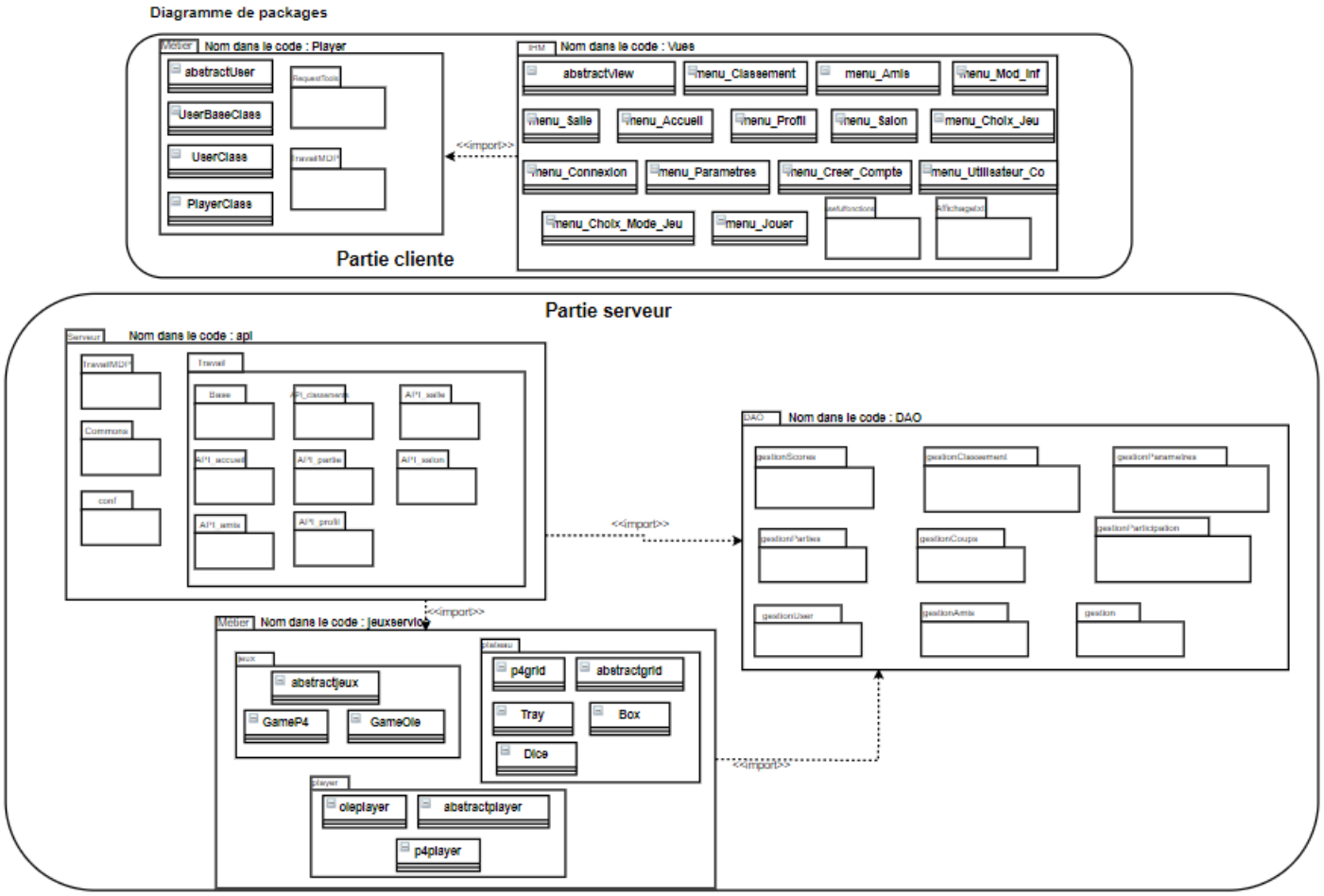

FIGURE 11 – Diagramme de packages de notre application

Ce diagramme correspond en fait à deux diagrammes de packages :

1. **Diagramme de l'application cliente**

   — `Package métier:` il regroupe l'ensemble des classes et fonctions qui serviront aux utilisateurs de notre application afin de jouer aux jeux qu'ils souhaitent et de d'accéder aux fonctionnalités supplémentaires de notre application en communiquant avec le serveur et en renvoyant les informations à l'IHM pour qu'elles soient affichées.

   — `Package IHM:` ce package regroupe l'ensemble des classes qui permettront l'affichage des menus et l'enregistrement des choix de l'utilisateur.

   Les tâches sont donc bien séparées : l'IHM se contente de faire l'affichage et le métier effectue les actions nécessaires au bon fonctionnement de l'application cliente.

2. **Diagramme de l'application serveur API**

— `Package API:` il s'agit de l'ensemble des fonctions nécessaires au fonctionnement de l'API. On y trouve des fonctions qui reçoivent des informations de l'utilisateur, qui communiquent avec la DAO et qui renvoient des informations à l'utilisateur. De plus, nous y avons rédigé les scripts permettant le hachage des mots de passe, les scripts anti-injection SQL, ainsi que la fonction qui permet de récupérer l'adresse de notre API

— `Package DAO:` Ce package est lui-même composé de packages, car nous avons choisi de ne pas faire de classe pour la DAO. En effet, nos DAO reçoivent et enregistrent dans la base de données des informations très variées, suivant des buts également variés. Il nous est donc apparu superficiel de créer une classe abstract DAO qui ne posséderait pas ou peu de méthodes abstraites.

— `Package Jeux:` il s'agit de l'ensemble des fonctions nécessaires au déroulement des jeux.

Là encore les tâches sont séparées : le serveur API récupère les requêtes des clients et appelle la DAO lorsqu'il doit travailler avec la base de données ou le métier lorsqu'il doit travailler avec les jeux.

2.5 Diagrammes de classe

Dans cette partie nous allons décrire l'ensemble des classes de la partie métier de notre application. Comme pour le diagramme de packages, on distingue deux types de classes métiers : les classes métiers de la partie serveur et les classes métiers de la partie cliente.

2.5.1 Diagramme de classe du client

On peut y lire quatre classes :

— `AbstractUser` : Il s'agit d'une classe qui correspond à un utilisateur abstrait. C'est de cette classe qu'hériteront les UserBases qui sont les utilisateurs anonymes et les Users qui sont des utilisateurs disposant d'un compte.

— `UserBase` : Il s'agit d'une classe qui correspond aux utilisateurs anonymes de notre application.

— `User` : Il s'agit de la classe qui correspond aux utilisateurs de notre application qui disposent d'un compte. Ils disposent de méthodes supplémentaires par rapport aux UserBases qui correspondent aux fonctionnalités supplémentaires que nous avons implémentées.

— `Player` : Il s'agit de la classe qui correspond aux utilisateurs en jeu. Cette classe en plus des méthodes de la classe User disposent de méthodes qui permettent à un utilisateur d'effectuer ses actions lorsqu'il est en partie.

Ci-dessous est représenté le diagramme de classe de la partie cliente :

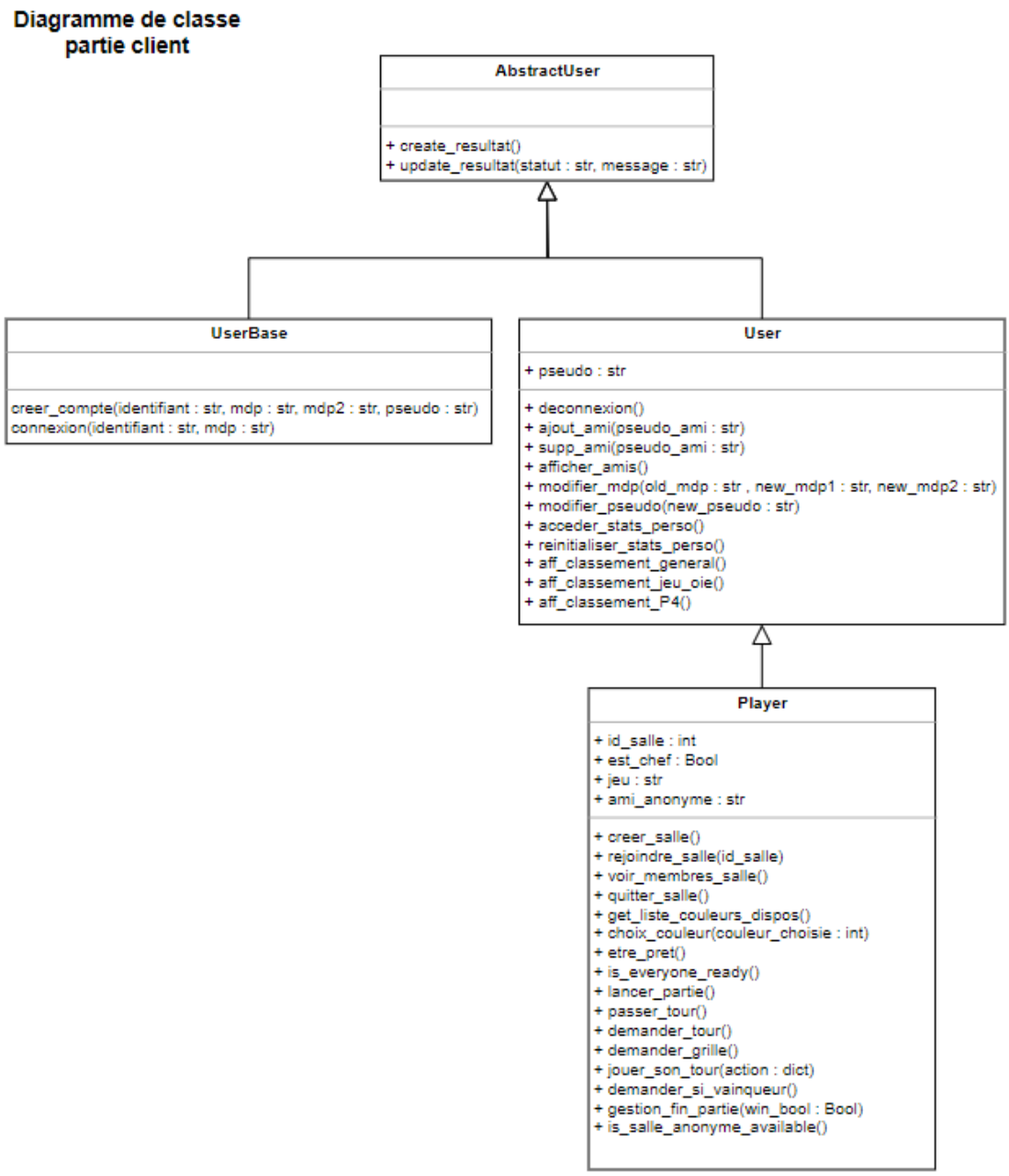

FIGURE 12 – Diagramme de classe de la partie cliente de notre application

2.5.2 Diagramme de classes du serveur

Nous avons également développé des classes sur la partie serveur de notre application, ces classes regroupent l'ensemble des attributs et méthodes nécessaires au fonctionnement de nos jeux avec une API.

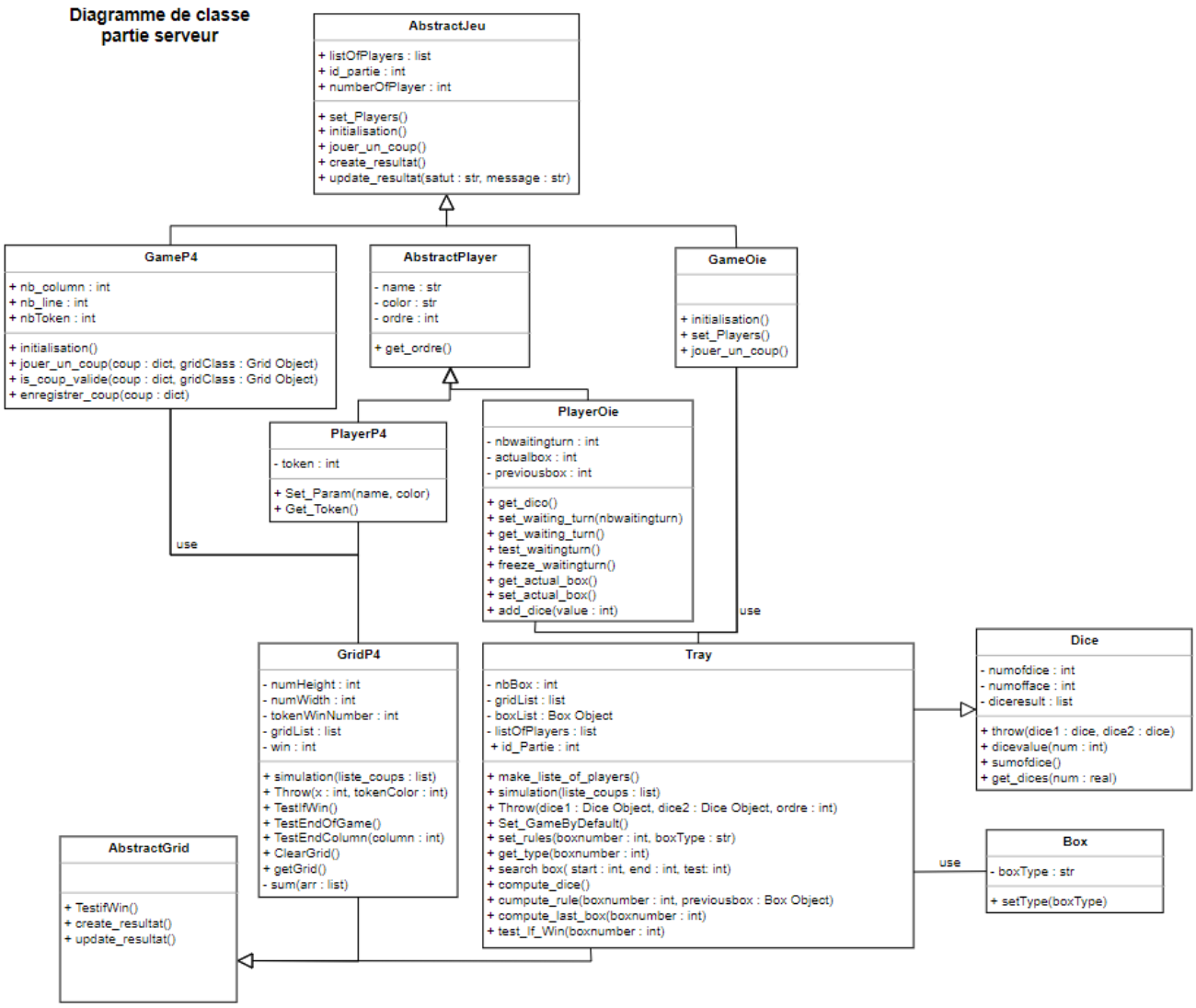

FIGURE 13 – Diagramme de classe de la partie serveur de notre application

— `AbstractJeu` : Il s'agit d'une classe abstraite dont héritent tous nos jeux.

— `GameP4` : Il s'agit de la classe qui permet le déroulement d'une partie de puissance 4. Elle hérite d'AbstractJeu et utilise les attributs et méthodes de playerP4 et GridP4 pour permettre aux utilisateurs qui jouent au puissance 4, de jouer leurs coups et de les enregister.

— `GameOie` : Il s'agit de la classe qui permet le déroulement d'une partie du jeu de l'oie. Elle hérite d'AbstractJeu et utilise les attributs et méthodes de PlayerOie et Tray pour permettre aux utilisateurs qui jouent au jeu de l'oie, de jouer leurs coups et de les enregister.

— `AbstractPlayer` : Il s'agit d'une classe abstraite dont héritent tous les joueurs, peu importe le jeu auquel ils jouent.

— `PlayerP4` : Il s'agit d'une classe qui hérite d'AbstractPlayer et permet d'identifier les différents joueurs qui jouent au jeu du puissance 4.

— `PlayerOie` : Il s'agit d'une classe qui hérite d'AbstractPlayer et permet d'identifier les différents joueurs qui jouent au jeu de l'oie.

— `AbstractGrid` : Il s'agit d'une classe abstraite qui permet de définir les plateaux des jeux présents dans notre application, et d'implémenter les actions et vérifications nécessaires au déroulement de la partie.

— `GridP4` : Il s'agit d'une classe qui hérite d'AbstractGrid et qui implémente les actions qui permettent aux joueurs de jouer au puissance 4.

— `Tray` : Il s'agit d'une classe qui utilise la classe Box et hérite d'AbstractGrid et de Dice. Elle implémente les actions qui permettent aux joueurs de jouer au jeu de l'oie.

— `Box` : Cette classe permet de décrire des cases selon la règle qui lui est affectée au jeu de l'oie.

— `Dice` : Il s'agit d'une classe qui permet la gestion des dés : le lancer des dés, la lecture et le renvoie du résultat de ceux-ci.

3 Implémentation

3.1 Structure de l'application

L'objectif de ce projet était de créer une API qui respecte l'architecture REST. Les API reposent sur le fonctionnement client-serveur. Un client envoie une requête à un serveur, et le serveur répond à la requête du client. Dans le cadre de l'architecture REST (Representational State Transfer), les requêtes du client sont des requêtes HTTP et les objets renvoyés par le serveur sont au format JSON. De plus les requêtes doivent être formulées avec le bon mot clé, par exemple si l'on souhaite modifier des données il faut utiliser la requête PUT. Enfin, point essentiel, l'API REST est *stateless*, c'est-à-dire que chaque requête contient l'ensemble des informations requises par le serveur pour être comprise

Ce fonctionnement client-serveur, se retrouve dans notre projet. En effet, en plus de la base de données, il est possible de relever deux parties distinctes : la partie serveur qui contient API et DAO et la partie cliente qui regroupe les classes clientes et les menus aussi appelés interface homme-machine. Tout au long de ce projet, nous avons cherché à distinguer au mieux chacune de ces classes afin de les séparer en différentes couches et donc de faciliter leurs modifications et de limiter le nombre d'erreurs. En somme, nous avons cherché autant que possible à limiter les dépendances entre les différentes couches.

3.2 Partie Serveur

3.2.1 API REST

L'API (Application Programming Interface) que nous avons développée dans ce projet, fonctionne grâce à un ensemble d'adresses appelées "endpoints" associées à des verbes HTTP. Par exemple l'endpoint et le verbe associé à la fonctionnalité permettant à un joueur de savoir si c'est le tour de jeu de l'utilisateur est :

```
1 @app.route('/home/game/room/turns', methods=['GET'])
```

Listing 1 – enpoint et méthode HTTP pour demander si c'est son tour de jouer

A l'intérieur de ces endpoints, nous définissons l'ensemble des fonctions nécessaires au traitement des informations envoyées ou requêtées par l'utilisateur.

Ces endpoints sont présents sur la partie cliente et sur la partie serveur de notre application, afin de savoir respectivement où se réalisent l'envoie et la réception de la réponse côté client et afin de savoir où se réalise le traitement de la requête du client côté serveur. Ainsi, notre API est un ensemble d'endpoints et de verbes qui permettent la communication entre client et serveur. Nous avons donc regroupé tous ces endpoints et les fonctions qui y sont associées dans plusieurs scripts :

— `Base`: Il s'agit d'un script où est rédigée la fonction *make_reponse* qui nous permet de standardiser toutes les réponses de l'API.

— `API Acceuil`: Il s'agit du script qui contient les endpoints relatifs à la connexion et à l'inscription d'un utilisateur sur notre application.

— `API Amis:` Ce script contient l'ensemble des endpoints nécessaires à la gestion de la liste d'amis d'un utilisateur.

— `API Classement:` Ce script contient tous les endpoints associés à l'affichage des classements.

— `API Partie:` Ce script contient les endpoints qui permettent la vérification et la réalisation des actions faites par les utilisateurs en partie.

— `API Profil:` Il s'agit du script qui définit les endpoints nécessaires à la modification des informations personnelles des utilisateurs, et l'accès aux statistiques personnelles.

— `API Salle:` Il s'agit du script qui contient les endpoints relatifs à la gestion des salles de jeux, par exemple la création d'une partie de puissance 4.

— `API Salon:` Il s'agit du script qui définit les endpoints permettant aux utilisateurs d'être prêt, de choisir leur couleur et de lancer la partie.

Notre API respecte également l'architecture REST, car le verbe associé à chaque endpoint est toujours celui qui correspond dans le protocole HTTP à l'action requêtée par le client. Ainsi c'est la méthode DELETE qui est utilisée pour supprimer un ami, c'est la méthode GET qui est utilisée pour récupérer les informations relatives au tour de jeu d'un joueur, c'est la méthode POST qui est utilisée pour ajouter un utilisateur à notre base de données etc... De plus le format des réponses de notre serveur est le format JSON, et la communication entre client et serveur est sans état, c'est-à-dire que l'interaction client serveur se fait toujours avec un couple de requêtes et de réponses, il n'y a pas de canal de communication qui s'ouvre entre un client et le serveur.

3.2.2 Communication entre l'API et la base de données.

Notre API utilise une base de données afin de conserver durablement les informations relatives aux utilisateurs, et aux parties de jeu. Afin de permettre à l'API d'accéder à ces informations, nous avons créé des fonctions qui permettent la communication entre la base de données et l'API. Il ne s'agit pas d'une DAO car nous trouvions qu'il était superficiel de créer des classes statiques pour chacune des fonctionnalités de notre application. Il s'agit donc d'un ensemble de fonctions qui permettent de lire et de modifier les informations disponibles dans la base de données. La connexion à la base de données se fait à l'aide du package `sqlite3`. Ce package permet de se connecter à la base de données et d'exécuter les requêtes SQL nécessaires à la modification ou la récupération d'informations dans cette base. Nous avons donc codé plusieurs scripts permettant la récupération et la transmission de ces informations, un pour chaque fonctionnalité de notre API, on peut ainsi lister :

— `gestion:` Ce script permet de récupérer le chemin d'accès à la base de données. Cela est utile si son positionnement relatif au fichier *api.py* est modifié.

— `gestionAmis:` Ce script regroupe l'ensemble des fonctions qui permettent de récupérer ou de modifier des informations liées à la gestion des amis comme l'affichage de la liste d'amis ou la suppression ou l'ajout d'un ami dans la base de données.

— `gestionClassement:` Ce script contient les fonctions qui permettent d'afficher les classement (général ou propre à un jeu). Les requêtes SQL associées sont plus complexes.

— `gestionCoups`: Ce script permet de définir les fonctions qui peuvent enregistrer les coups effectués par les joueurs en partie et récupérer les informations relatives aux anciens coups.

— `gestionParticipation`: Ce script permet d'accéder aux informations sur la participation des joueurs à une partie, la couleur de leur pion, leur statut (si ils sont prêts ou non) et l'ordre dans lequel les joueurs vont jouer.

— `gestionParties`: Ce script permet de modifier ou d'accéder aux informations de la base de données qui s'occupent de créer et de supprimer des parties de la base de données. Il permet également l'accès aux informations qui permettent d'ajouter ou d'annuler la participation d'un joueur à une partie et de lancer la partie.

— `gestionScores`: Les fonctions présentent dans ce script permettent de modifier et de récupérer dans la base de données, le score des joueurs de notre application.

— `gestionUser`: Ce script contient les fonctions qui vérifient les informations relatives à un utilisateur et qui permettent de l'ajouter ou de modifier certaines de ses informations à la base de données.

Ainsi notre API utilise un ensemble de fonctions afin de récupérer et de sauvegarder les informations requêtées ou envoyées par les utilisateurs.

3.2.3 Fonctionnement des jeux.

Les deux jeux que nous avons codés pour ce projet sont le puissance 4 et le jeu de l'oie. L'implémentation de ces derniers est très similaire car il s'agit de jeux au tour par tour qui disposent d'un plateau de jeu et de jetons. Les différences principales, hormis le but final du jeu bien sûr, sont que le puissance 4 se joue avec autant de jetons que nécessaires, tandis que chaque joueur possède, au jeu de l'oie, un unique jeton.
Pour un rappel sur le fonctionnement logique du tour par tour côté client, vous pouvez vous référer à la partie [**2.2.1**].
Ici, nous allons décrire l'architecture logique côté serveur. Lorsque c'est le tour d'un client, nous avons déjà vu qu'il effectue les actions suivantes dans l'ordre : récupération de la table de jeu, vérification de victoire, jouer, récupération de la table, vérification de victoire, passer son tour.

Côté serveur, la récupération de la grille se fait à l'aide la fonction get_grille à l'adresse "/home/game/room/grid" via la méthode **GET**. Le serveur requête la base de données pour récupérer l'ensemble des coups joués dans la partie depuis le début jusqu'à présent puis transmet ces informations aux classes jeux. Ces classes simulent une partie avec les coups joués depuis le début et renvoient l'état de la grille une fois tous ces coups joués. Le serveur renvoie alors au client la grille dans son état pour que l'IHM puisse l'afficher pour le joueur.
La vérification de victoire est assez similaire. Elle se réalise via la fonction demander_si_vainqueur et là encore, la grille est simulée. Cette fois en revanche, la grille pour l'afficher n'est pas renvoyée mais un test afin de savoir si c'est une grille gagnante est effectué (joueur sur la case 63 pour le jeu de l'oie, 4 jetons alignés pour le P4, ...).

Enfin, pour jouer son tour (via la fonction jouer_son_tour), il est seulement vérifié que le coup joué est bel et bien valide. Concernant le jeu de l'oie, c'est assez simple car le coup correspond au lancé des 2 dés. Pour le puissance 4 en revanche, il faut simuler une grille et vérifier que la colonne dans laquelle le jeton est joué n'est pas déjà pleine (voir même qu'elle existe bien si le joueur essaye de jouer en dehors de la grille).

Vous l'aurez compris, la fonction phare des jeux est la méthode simulation de la classe Grille qui permet de simuler une grille via l'ensemble des coups joués précédemment. Ainsi, pour implémenter un nouveau jeu, il faut que ce jeu dispose d'une méthode simulation pour simuler les grilles. Ensuite, d'autres méthodes sont utiles comme la méthode testIfWin qui permet de savoir si une grille est gagnante ou la méthode is_coup_valide mais dans l'ensemble, la méthode simulation est centrale.

De cette manière, si dans le futur nous souhaitons implémenter un nouveau jeu, il suffit de créer une classe Grille (ici, nous avons la classe Tray pour le jeu de l'oie et Grid pour la puissance 4) afin de gérer le plateau de jeu, ainsi que ses règles. De la même façon, une méthode Throw permettrait de lancer le tour du jeu (lancer de dés, pose de jetons...) et de vérifier les différentes règles à appliquer. Une méthode de testIfWin testerait, quant à elle la fin de partie, à savoir si un joueur a rempli les conditions de victoire en premier. Enfin, une méthode simulation permettrait de reconstituer tous les tours de jeu.
Ensuite, il faudrait créer une nouvelle classe Player afin de gérer les joueurs ainsi que leurs propriétés dans le jeu (oie = tour d'attente, case actuelle... / p4 = couleur de jeton du joueur). En fonction du jeu à implémenter, il faudra adapter la structure, et plus particulièrement le résultat du tour. En effet, dans le jeu de l'oie, ce dernier (case d'arrivée du joueur) est stocké dans la classe Player ; tandis que pour le P4, il (jeton joué dans une colonne) est stocké dans la classe Grid. Ceci explique le fait que la méthode simulation reconstitue le jeu par un biais différent selon le jeu : historique des colonne jouées dans Grid pour le P4, et dernière case pour chaque joueur dans Player pour le jeu de l'oie.

3.3 Partie cliente

La partie cliente de notre application regroupe l'ensemble des menus présents dans notre application ainsi que les "classes clientes" qui possèdent toutes les méthodes nécessaires à la réalisation des actions disponibles pour les utilisateurs.

3.3.1 Interface Homme Machine

L'implémentation d'une Interface Homme Machine (IHM) qui permet aux utilisateurs de naviguer sur notre application, s'est faite au travers de plusieurs menus. Pour cela nous avons utilisé le package PyInquirer qui nous permet de créer des menus et d'enregistrer les choix des utilisateurs dans des dictionnaires. Ces dictionnaires sont ensuite utilisés pour savoir quelle action réaliser,

appeler les méthodes nécessaires et transmettre les éventuelles informations de l'utilisateur aux fonctions qui réaliseront l'action choisie. Ils héritent tous d'une classe abstraite AbstractMenu, et surchargent tous les méthodes display_info() et make_choice() qui permettent respectivement d'afficher un message lors de l'arrivée sur un nouveau menu et de proposer l'ensemble des actions présentent dans un menu. Nos menus sont imbriqués les uns dans les autres, à différents niveaux, ci-dessous un schéma représentant l'ensemble des menus présents dans notre application et les possibilités qu'ils offrent :

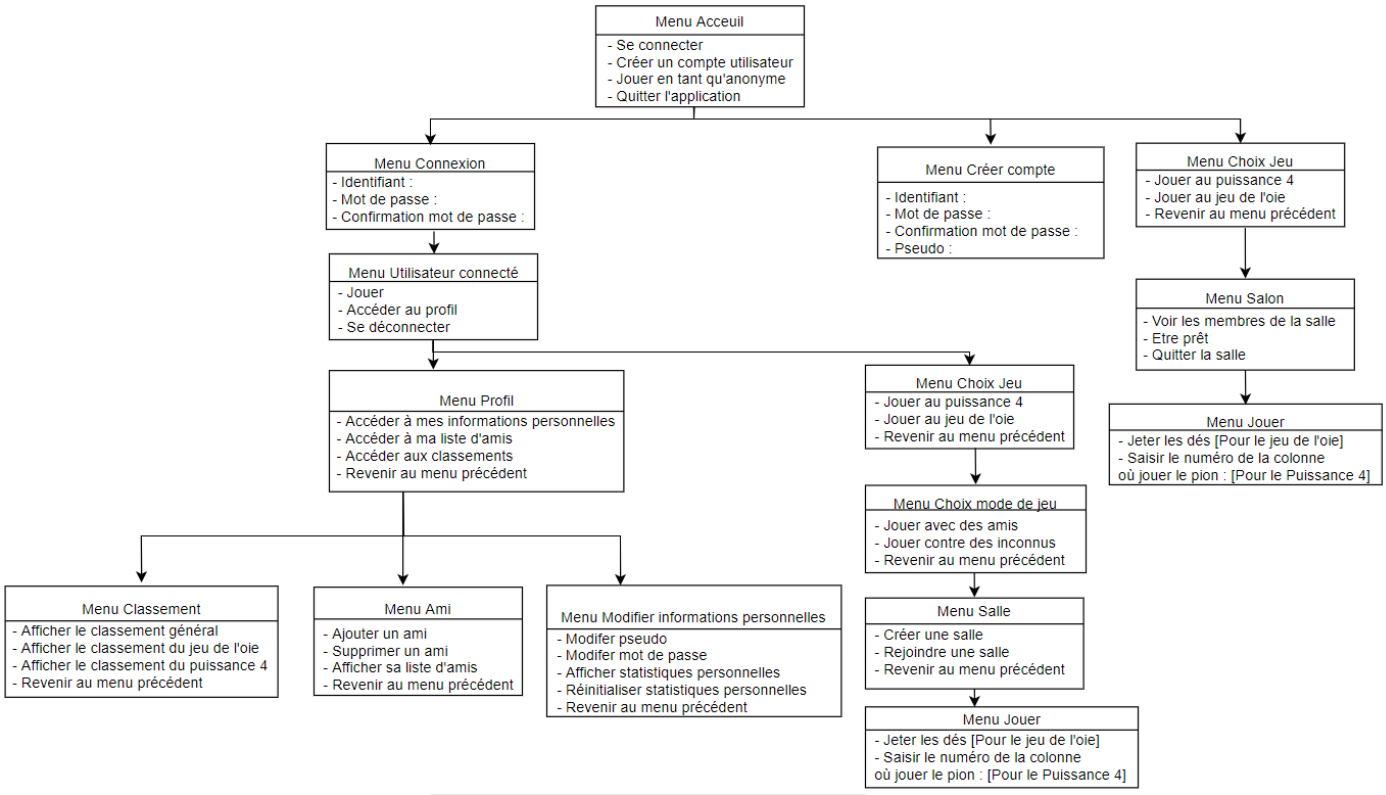

FIGURE 14 – Schéma modélisant le cheminement au sein de nos menus.

3.3.2 Classes clientes

Les actions disponibles pour les utilisateurs de notre application sont implémentées dans les classes clientes. Il s'agit des classes représentées sur le diagramme de classe de la partie cliente de notre application. Ces méthodes sont appelées lorsque l'utilisateur choisi une action dans le menu. Ces méthodes envoient une requête à l'aide du package `requests` à l'endpoint associé dans notre API avec le verbe HTTP correspondant. Une fois la requête réceptionnée par l'API, elle est traitée par la fonction contenue dans l'endpoint requêté. Puis l'API envoie une réponse au client, au format JSON et selon le statut de la requête, l'action est réalisée ou non. Dans tous les cas l'utilisateur est redirigé vers un menu pour retenter son action si elle a échoué ou poursuivre son cheminement dans notre application.

3.4 Tests de notre code

3.4.1 le module *test*

Le principal intérêt d'effectuer des tests est de s'assurer du bon fonctionnement d'un programme. Ils s'avèrent très utiles dès lors que l'on cherche à réaliser un projet comprenant de nombreux packages. Le développement des tests s'est axé autour d'*Unittest*, l'un des modules de prédilection pour réaliser des test unitaires avec Python. Nous avons notamment utilisé *TestCase*, dont le principe d'utilisation est de créer une sous-classe qui comprendra l'ensemble des tests à réaliser. L'avantage de cette fonctionnalité est de grandement simplifier la syntaxe et la cohésion des tests et de participer activement au principe de "forte cohésion, faible dépendance".

3.4.2 Tests de la DAO

Pour des raisons de limites de temps pour le rendu du projet, nous avons préféré nous atteler à 100% à coder notre API et notre application client et avons donc manqué de temps pour rédiger des tests unitaires pour chaque classe et chaque méthode de chaque classe. Nous avons cependant eu le temps de coder des tests pour les fonctions de la DAO. Nous avons décidé de réaliser ces tests en créant pour chacun d'entre eux un utilisateur temporaire dans la base de données qui sera supprimé dès la fin du test. Nous avons également fait appel au module *secrets* afin de générer des pseudonymes, identifiants et mots de passe aléatoires et sécurisés pour ces utilisateurs temporaires. Le principal intérêt de l'appel à ce module reste toutefois d'assurer l'unicité des identifiants. Bien qu'imparfaite, cette méthode reste optimale au vu des impératifs de temps et des outils à disposition. Nous avons en effet songé à utiliser le module mock pour créer un objet imitant le comportement de notre base de données et pouvoir effectuer des tests de manière plus sécurisée. La première option a néanmoins été privilégiée pour cause de contraintes temporelles.

4 Remerciements & Bilans personnels

4.1 Remerciements et conclusion

Avant de conclure réellement ce rapport avec nos notes personnelles, nous aimerions conclure la partie "rapport" a proprement parler avec quelques remerciements.
Nous remercions notre tuteur, Monsieur Thierry MATHE pour l'aide, conseils, remarques et réponses qu'il nous aura apporté au cours du projet.
Nous remercions également notre camarade Elouen Ginat qui nous à aider au début du projet pour bien comprendre le fonctionnement d'une API et pour bien cadrer les attendus du projet.
Nous remercions notre enseignant Monsieur Rémi PEPIN pour les rapports et messages qu'il a transmit au cours du projet et pour son enseignement, et plus particulièrement pour ses corrections de TP dont nous avons pu, parfois, nous inspirer et qui ont su, souvent, nous aider à comprendre.
Nous remercions enfin tous nos amis et notre famille qui ont su, à n'en pas douter, faire preuve du soutien moral nécessaire à la réalisation de ce projet qui a été, comme vous pourrez le lire dans les bilans personnels, extrêmement éprouvant et chronophage.

4.2 Hugo BERNARD

Apport au groupe et tâches réalisées.

Ce projet avait pour objectif de construire une API REST. Nous nous sommes organisés au travers de réunions régulières, au cours desquelles nous nous répartissions le travail. J'ai beaucoup apprécié que tout le monde se soit investi dans le projet car lors de mon projet de 1ère année cet investissement était moins présent. Par conséquent j'ai le sentiment d'avoir été moins utile que lors du projet de 1ère année. En effet, même si j'ai participé à l'implémentation de plusieurs fonctionnalités de notre projet, je pense que cette participation est plus faible que celle d'autres membres du groupes. Par conséquent j'ai cherché à compenser cette faible participation en m'investissant plus dans la réalisation des différents diagrammes et dans la rédaction du rapport final.

J'ai commencé à concevoir des menus basiques à l'aide de PyInquirer. C'est sur la base de ces menus que nous avons conçu une partie de l'Interface Homme Machine. Cependant ces menus ont été grandement améliorés et étoffés au fur et à mesure que le projet a avancé. Puis lors de la phase d'immersion, j'ai travaillé sur une première version de l'affichage des classements. J'ai donc mis en place les endpoints de l'API nécessaires à l'affichage des classements , les méthodes côté client et côté serveur qui permettent aux utilisateurs de requêter l'API pour obtenir les classements, ainsi que les fonctions associées permettant à l'API de communiquer avec la base de données. Par la suite ces fonctions ont été reprises par Zacharie qui les a grandement amélioré, en permettant non seulement d'afficher le classement des meilleurs joueurs mais aussi en permettant d'afficher le classement des amis d'un utilisateur.

Après la période d'immersion j'ai réfléchi à la conception d'un système de modification de paramètres. J'ai alors développé les menus, adapté la base de données et mis en place les endpoints nécessaires au bon fonctionnement de cette gestion des paramètres. Malheureusement, par manque de temps nous n'avons pas pu mettre en place cette fonctionnalité. Enfin lors des dernières semaines du projet, je me suis occupé de la séparation du code en couche. En effet, il était important de bien distinguer chaque couche de notre application et certains scripts étaient un peu confus. Cette séparation du code a été poursuivie par Clément qui l'a encore perfectionné. Finalement je me suis occupé de la rédaction du rapport final, soit la mis à jour de l'ensemble des diagrammes concernant notre application et la description de l'implémentation et du fonctionnement de notre application. Bon nombre de diagrammes ont évolué au fur et à mesure que notre application a pris forme, il a donc fallu prendre ces modifications en considération dans notre rapport.

Enseignements tirés du projet.

Tout d'abord lors de ce projet j'ai appris ce qu'était une API et comment est-ce que celle-ci fonctionnait. Il est assez facile de s'imaginer ce qu'est un fonctionnement client-serveur, cependant il est plus difficile de réaliser jusqu'où ce fonctionnement est utilisé. C'est pourquoi j'ai été surpris d'apprendre que l'allocution des tours de jeu entre les joueurs fonctionnait sur un système de requêtes régulières de chaque joueur au serveur. J'ai aussi appris à mettre en place des fonctionnalités dans une API, et donc toutes les implications qui vont avec : création de la base de données, le système de requêtes et de réponses à l'aide de endpoints et de verbes HTTP et la création de fonctions qui communiquent avec la base de données.

J'ai également appris à créer une base de données. Lors du projet d'informatique de première année nous avions utilisé un système de sérialisation des données à l'aide du package *pickle* dans des fichiers qui contenaient des dictionnaires. Cette année j'ai donc participé à la conception d'une base de donnée avec SQLite et à l'implémentation des fonctionnalités qui permettent d'interagir avec celle-ci.

Ce projet a aussi été l'occasion de découvrir GitHub. Il s'agit de la plateforme sur laquelle était stockée les fichiers qui contiennent notre API. Bien que difficile à prendre en main celle-ci s'est avérée très pratique au cours de ce projet afin de travailler à plusieurs simultanément sur le même code.

Difficultés rencontrées, pistes d'amélioration et bilan.

La première difficulté rencontrée au cours de ce projet a été l'assimilation du concept d'API. Je n'en avais jamais entendu parlé auparavant et son fonctionnement nous a été présenté très succinctement. Il a donc fallu faire de nombreuses recherches, et plusieurs tentatives d'implémentation, avant de finalement parvenir à faire communiquer un client et un serveur.

Une autre difficulté a été la manipulation de Git. En effet, c'était la première fois que j'utilisais un tel logiciel. Il a donc fallu un certain temps avant de comprendre le fonctionnement de ce logiciel. De plus j'avais quelques appréhensions vis-à-vis de fausses manipulations qui supprimeraient l'ensemble de notre travail, bien qu'un retour en arrière est toujours possible.

La dernière difficulté que j'ai rencontré lors de ce projet a été la gestion du temps. En effet, il me faut un certain temps pour assimiler de nouvelles notions ce qui a pu ralentir la progression de mon travail, notamment lors de l'implémentation de la première version des fonctionnalités liées au classement.

Je trouve l'informatique très intéressant, on peut réaliser tout type d'applications et des applications de qualités à l'aide de Python, comme l'illustre respectivement la liste des sujets de projet et notre application. Pour conclure, je souhaite remercier tous les membres du groupe. Ils ont su de par leur investissement et leur état d'esprit, rendre le travail sur ce projet agréable et ont ainsi développé une superbe application.

4.3 Zacharie BOUHIN

Ce projet informatique était un vrai défi comparé à celui de l'année dernière, où, l'on avait la possibilité de former nos propres groupes. En effet, le système de composition des groupes, la charge de travaille à première vue et le fait que le sujet soit imposé pouvait paraitre inconfortable. Ce qui paraissait l'être dans un premier temps.

Organisation au sein du groupe et répartition des tâches.

Au lancement du projet, la vision de l'organisation c'est assez facilement réparti. La première chose de faite a été de désigner Clément Gabas chef du groupe, cela s'est fait d'un commun accord. Je tiens à noter que cet accord était un excellent choix. Tout au long du projet, Clément a su maîtriser parfaitement se rôle au sein du groupe, que ce soit via les différentes réunions pour faire un bilan de nos avancées et des nos tâches à venir, ou bien via lien qu'il faisait entre les membres du groupe et notre tuteur, monsieur Mathe, lors de nos différents rendez-vous projets.

La première partie du travail était concentré sur les différents diagrammes UML, et le premier bilan qu'on doit en faire est qu'on avait très peu de bases. En plus d'un sujet que l'on maitrisé peu voire pas du tout : création d'une API REST, nous avons dû revoir tout l'architecture des diagrammes à plusieurs reprises. Au niveau de la réparation des taches, on s'était mis d'accords pour que chacun fasse des diagrammes, quitte à ce que nous soyons plusieurs à travailler sur un même diagramme. De souvenir, j'étais chargé de travailler sur les diagrammes de séquences et d'activités. Cette période de travail étant en présentiel, il était assez simple pour nous de nous réunir pour discuter de nos avancé respective et comparé ce que nous avions produit. Cette première partie du projet s'est conclue par la rédaction du rapport d'analyse. Encore une fois nous y avons tous apporté notre pierre a l'édifice ce qui m'a donné le premier sentiment d'être dans un groupe ou tout le monde est investi, ce qui n'est pas souvent le cas.

La deuxième partie du projet a été la partie code. C'est à partir de là que l'on a pu noter les premiers problèmes du sujet. Déjà pressentit lors de la partie UML, nous avons constaté que notre sujet était assez fourni en travail, et, de mon point de vu, bien plus compliqué que d'autre sujet. Ainsi, avec Mael, nous avons dans un premier temps chercher à en apprendre un maximum sur `FLASK` et comment l'employer pour créer une API Rest. Après avoir correctement compris `FLASK`, et avec un peu d'aide, Clément a pu faire la plupart des relations métiers- API – DAO - Base de données. En se basant sur son travail, j'ai travaillé sur certaine fonctionnalité telles que la gestion des classements. Après avoir fini la partie « application », Romane et Clément ont implémenter les jeux dans le programme, pendant que Hugo mettait, entre autres, les diagrammes et Mael s'occupait des tests. De mon côté, je m'attaquais à la documentation de notre programme. Une fois tout cela fait, il nous resta plus que la rédaction de rapport bien entamée par Hugo et la correction de quelques coquilles.

Ce projet n'a pas été de tout repos, il nous est arrivé à plusieurs reprise de rester bloquer sur

certaines erreurs pendant des heurs, mais par la bonne communication au seins du groupe, nous avons su, pour certaines d'entre elles, corrigé ces erreurs ensemble.

Retour personnel

Dans cette partie j'aimerai parler de mes différents ressentis pendant l'élaboration de ce projet. Tout d'abord, au vu du projet informatique précédant, j'étais assez effrayé de ce projet. En effet l'année dernière, j'ai vraiment eu le sentiment d'être lâché dans le vide sans vrai encadrement et vraies consignes. Les erreurs plus que fréquentes dans les données ont fait que nous avions du, à trop de reprise, corriger des coquilles. Par chance, notre sujet n'avez aucune donnée à étudier donc aucune crainte là-dessus. Je tiens donc à remercier monsieur Pépin, qui a quant à lui toujours été clair et compréhensif dans les différents mails envoyés.

Je n'ai pas grand-chose à dire sur la première partie du projet car elle s'est faite dans des conditions plutôt normales au vu de la situation sanitaire actuelle. Cependant le deuxième temps du projet qui s'est fait en grande partie en distanciel a été beaucoup éprouvante. En effet dans un contexte où une partie des étudiants, dont moi, se sont sentis abandonnés par l'administration de l'école (et du groupe GENES) qui, de mon point de vue, ne se rend pas compte des difficultés que peut rencontrer un étudiant pendant ce second confinement. Ainsi, bien que fort chronophage, ce projet informatique a été pour moi un exutoire, un défouloir, un moyen de se changer les idées en ses temps difficile, et je ne pense pas être le seul dans ce cas. Pour preuve, comme dit précédemment, ce sujet me paraissait long et compliqué, et pourtant nous avons réussi a avancé efficacement et je suis, personnellement très satisfait du résultat.

Pour apporté une note positive, je tenais à dire que ce projet, a confirmé l'intérêt que j'avais pour l'informatique, bien que ce ne soit pas une matiere que je maîtrise parfaitement. Il m'arrivait de passer des journées (et des nuits) à coder, à essayer des choses et à chercher des erreurs dans le code sans voir le temps passer. De plus, j'ai beaucoup appris sur le fonctionnement de python, que ce soit les API, la relation entre Python et SQL via PyInquirer, oui bien l'architecture elle même du code. Et enfin, j'ai appris à utiliser Git, et à vraiment travailler à plusieurs sur un même code.

Message aux membres du groupe

Je voulais absolument vous remercier tous les quatre Clément, Hugo, Mael et Romane pour ce projet. Au début j'avais un peu peur car je n'avais pas eu la chance d'échanger avec certains d'entre vous. Mais dès les premières séances j'ai constaté qu'il y avait une excellente alchimie. Sur ce projet vous avez fait un super boulot, j'espère avoir été un bon crewmate. Encore merci et bravo à tous.

4.4 Clément GABAS

Ce projet a été pour moi autant un supplice extrêmement chronophage qu'un véritable bonheur du point de vue technique et intellectuel, et c'est ce que je vais essayer de retranscrire dans ma note personnelle.

Mon apport au groupe

Dès l'annonce du groupe, presque sans concertation, les autres membres m'ont proposés de tenir le rôle de "chef de groupe". J'ai évidemment accepté car je pense avoir les qualités requises pour encadrer un groupe, d'autant plus en informatique, une matière que j'apprécie tout particulièrement, et d'autant plus que j'ai déjà tenu ce rôle, officieusement bien sur, dans mes projets de l'année passée.
En tant que chef de groupe, j'ai assuré tout au long du projet les interactions entre les membres et j'ai réparti l'ensemble des tâches à chacun en se basant sur les qualités, défauts et motivations propres à chacun. Étant assez bon en algorithmique, j'ai pu aider chaque membre dans leurs tâches lorsqu'ils rencontraient des difficultés et j'ai su les aiguiller pour que tous les scripts codés par des membres différents suivent un pattern logique similaire et que la mise en relation des différents scripts se fasse sans encombre.
Utiliser git et un dépôt github nous a grandement aidé dans ce domaine puisqu'en ayant constamment accès aux travaux des autres et en expliquant les avancées de chacun sur notre conversation de groupe, nous avancions sans trop de difficultés pratiques. J'organisais une réunion par semaine environ afin d'observer l'avancée de chaque membre et pour pouvoir donner de nouvelles tâches, et bien sur, en parallèle, j'avais également des tâches à réaliser.
J'ai codé, à titre personnel, une grande partie des vues, quelques scripts dans les classes joueurs de la partie client, une grande partie de l'api et des DAO et j'ai implémenté au serveur les jeux codés par romane en local. En effet, j'ai beaucoup contribué à la rédactions du code mais cela reste un travail d'équipe.

Ce que j'ai appris du projet

Ce projet à grandement développé mes connaissances en python, que se soit par la découverte et la bonne compréhension des API, la découverte de multiples packages comme requests, FLASK, tabulate ou encore PyInquirer, mais également au niveau logique de code, j'ai appris à mieux découper mes codes en packages et sous packages indépendants et faiblement corrélés pour limiter les problèmes et j'ai surtout découvert l'importance et l'intérêt de standardiser les réponses, que ce soit par la méthode make_reponse de notre API ou la réponse update_resultat de nos classes clients. Je vais détailler.
Tout d'abord, via le cours et les TP de 2A, je savais comment requêter une API mais j'ignorais comment en créer une. Dorénavant, je sais comment faire. De plus, je maîtrise mieux les subtilités des packages FLASK et requests, surtout requests lorsque j'ai été bloqué pendant deux jours sur l'erreur suivante *TypeError : Object of type set is not JSON serializable jsonify.*

Ensuite, par le développement du client, j'ai découvert le package tabulate, utilse lorsqu'on doit se contenter d'affichage console, et surtout le package PyInquirer qui s'avère est très pratique pour faire un menu en console sans avoir à utiliser des "input".
De plus, j'ai maintenu mes connaissances en sql en développant une bonne partie des DAO.
Enfin, et c'est je pense ma meilleure découverte du projet, j'ai compris l'intérêt de standardiser ces réponses. En ayant des réponses standards qu'on adapte en fonction du contexte, le code gagne en lisibilité, en compréhension et il devient beaucoup plus simple de le mettre à jour et de rajouter des fonctionnalités. C'est pour cela que nous avons développé les fonctions make_reponse et update_resultat.

Ce que je changerai si c'était à refaire

Si nous avions aujourd'hui ce projet à faire, je ne pense pas que je changerai beaucoup de chose. En revanche, maintenant que j'ai compris l'intérêt de coder en package et de bien séparer les plus de scripts possibles pour que chaque chose soit à sa place, nous ferions ça dès le début alors qu'au cours de notre projet, nous avons plusieurs fois faire une refonte de notre code pour mieux séparer les différentes actions (exemple : pendant longtemps, les requêtes du client pour l'api étaient au sein même des menu). Aussi, j'essayerai, dans la mesure du possible, de coder les tests unitaires avant de coder le code, ce qui paraît plus logique.

Est-ce que l'informatique m'attire plus maintenant

Je le savais déjà et le projet n'a fait que renforcer ce que je pensais : j'ai un goût très prononcé pour l'algorithmique en général mais je déteste la planification. C'est-à-dire que j'ai pris beaucoup de plaisir à coder le projet, souvent en travaillant sans m'en rendre compte jusqu'à des heures très tardives, et en y prenant souvent du plaisir, mais j'ai détesté devoir réfléchir aux diagrammes de classes, ...Cependant, l'informatique n'est pas la matière qui me plaît le plus. Je considère plutôt l'informatique comme un outils pour les autres matières, notamment statistiques et économiques, qui le requièrent, et j'espère pouvoir continuer à utiliser l'informatique comme telle.

Remerciements

J'exprime un remerciement tout particulier à chacun des membres du groupe. Je suis, à titre personnel, très satisfait de l'entente que nous avons eu tout au long du projet et, même si de très nombreux aspects sont largement perfectibles dans notre code, j'ose dire que je suis également content de notre travail ! Bravo à vous !

4.5 Mael GUIVARCH

Organisation au sein du groupe

J'ai grandement apprécié tant le sérieux dont ont fait preuve tous mes partenaires de projet que l'ambiance qui régnait au sein du groupe. Nous partagions nos avancées au travers de réunions fréquentes et la répartition de nos tâches était toujours clairement définie.

La première partie du projet nous a permis de tous participer à la conception des diagrammes UML et à l'élaboration de l'application. Au niveau du code, mon rôle a été tout d'abord de rechercher les principes du fonctionnement du module *flask* afin de mieux cerner l'utilité des endpoints et la manière dont on implémente les interactions entre clients et serveurs avant que Clément n'implémente l'API. Mon autre rôle était de gérer en parallèle la mise en place des tests pour les modules permettant le dialogue avec la base de données.

Difficultés

Ma première remarque est que mon niveau en programmation ne m'a pas permis d'être aussi productif que mes camarades. Cet écart de niveau couplé à l'excellent travail réalisé par les autres membres a contribué à rendre mes apports au projet moins pertinents que lors de mes précédents projets.

D'un point de vu technique, j'ai également rencontré des difficultés au moment de la réalisation des tests. Une erreur liée au logiciel que j'utilisais pour lire les fichiers au format .db ainsi qu'à des problèmes de droits au niveau des dossiers m'a tout d'abord empêché de les vérifications de mon travail. Ce problème a été très chronophage et il m'a fallu un certain temps pour en déceler l'origine et poursuivre la conception des tests. Tout au long du projet, j'ai néanmoins bénéficié de l'aide de mes camarades qui m'ont permis de mieux percevoir les enjeux et de surmonter l'ensemble des difficultés rencontrées.

Aspects bénéfiques du projet

Je suis globalement très satisfait du projet, tant pour le résultat que pour ce qu'il m'a apporté. J'ai réellement le sentiment d'avoir progressé et de mieux saisir l'utilité des concepts vus en cours, notamment la séparation des responsabilités sans laquelle le projet aurait été un véritable calvaire à programmer.

La découverte de l'utilité pratique de git a également été une agréable surprise pour moi. Nous avions abordé son importance en cours mais je dois avouer que je restais sceptique quant à sa pertinence pour un simple projet académique. Après un rapide temps d'adaptation, il s'est néanmoins avéré que ça nous a considérablement facilité la tâche.

J'ai également bien plus apprécié ce sujet que celui des précédents projets. L'aspect ludique y a grandement contribué. J'ai eu l'impression d'avoir participé pour la première fois à mettre en place une application réellement aboutie, et la variété de fonctionnalités implémentées m'a permis d'avoir une vue d'ensemble et d'appréhender au mieux le rôle des différentes couches. La partie API et la programmation des interactions entre clients, serveurs et base de données restent pour moi les parties les plus intéressantes de ce projet. J'ai néanmoins moins apprécié la programmation des tests ainsi que l'analyse fonctionnelle, bien qu'essentielles au projet.

Conclusion

Finalement, j'ai rencontré un certain nombre de difficultés et le projet était parfois assez chronophage pour l'ensemble du groupe. La réalisation de cette API s'est néanmoins révélée très formatrice.

L'analyse fonctionnelle a été le berceau de nos idées et joue à ce titre un rôle fondamental au sein du projet. J'ai néanmoins trouvé cette partie bien plus rébarbative que la programmation à proprement parler. Il est à noter que l'aspect concret et divertissant du sujet a sans doute contribué au maintien de notre motivation. J'ai l'impression que ce projet m'a même apporté davantage que les cours puisque les impératifs de fonctionnement m'ont poussé à réellement approfondir par moi-même un certain nombre de sujets pour trouver des solutions viables à tous mes problèmes.

Enfin, je pense pouvoir dire que nous sommes tous satisfaits du "produit" fini et je tiens à remercier l'ensemble du groupe pour son implication et son sérieux.

4.6 Romane PARÈS

Ce nouveau projet informatique aura été indéniablement très enrichissant, et ce, à titre académique, mais aussi individuel. Ainsi, il m'a permis d'approfondir mes compétences en matière d'informatique, que ce soit en modélisation ou en implémentation, tout comme en termes d'organisation. De plus, les difficultés rencontrées auront été bénéfiques, même s'il m'aura fallu du recul pour en prendre conscience.

Organisation

Un des aspects les plus compliqués à appréhender (outre la tâche elle-même) lors de ce type de projet, demeure le travail en équipe. Forte de mes expériences précédentes (dont notamment les difficultés rencontrées lors du projet informatique réalisé en première année), je savais pertinemment qu'un des aspects essentiels pour mener à bien notre mission consistait en la réalisation d'un travail en bonne harmonie. De ce point de vue, mes attentes ont été entièrement comblées puisqu'une très bonne entente s'est immédiatement instaurée entre Clément, Hugo, Maël, Zacharie et moi, faisant que chaque membre a trouvé très rapidement sa place. Il est également à remarquer que nous n'avions pas tous les mêmes affinités par rapport aux tâches requises, mais qu'au final ces différences se sont avérées constituer un atout supplémentaire, justement de par cette complémentarité.

Par ailleurs, cette bonne communication au sein du groupe a été notamment facilitée grâce à Clément, qui en tant que chef de projet, s'est chargé de répartir de manière hebdomadaire les différentes missions. Chacun d'entre nous a ainsi eu ses propres tâches à réaliser et bien qu'une part importante du travail se soit alors réalisé individuellement, les nombreux échanges pour partager notre avancée, les modifications effectuées ... ou bien même se faire aider quand nous butions sur une difficulté particulière, ont été essentiels. Par ce bon état d'esprit, nous avons pu maintenir une très bonne dynamique et donc de terminer notre projet dans le temps imparti.

Les tâches réalisées

Lors de ce projet ma mission principale a été de développer des jeux (jeu de l'oie et puissance 4). Pour cela, j'ai appréhendé la gestion des règles, l'affichage graphique des grilles, la gestion multi-joueurs ainsi que la vérification « fin de partie ». Tout cela a, somme toute, représenté une part de travail importante. En effet, bien que gardant le souvenir d'un jeu de plateau enfantin, toute la complexité des règles du jeu de l'oie en a rendu l'implémentation difficile. Ainsi, par exemple, la gestion de la case 31 « puits » a été l'une des principales difficultés rencontrées étant donné qu'il fallait élaborer un code permettant un passage simple vers l'API. Pour cela, il m'a fallu d'une part réfléchir aux paramètres à passer dans la base de données, avant de déterminer la forme de ces derniers (integer, booléens...). (Il en a été de même pour le puissance 4.) Un des autres aspects intéressants était de trouver une architecture relativement similaire entre les deux jeux afin d'implémenter de nouveaux jeux dans le futur, sans avoir à réfléchir à chaque fois à une nouvelle structure. Enfin, les différentes parties de code, comme tout code digne de ce nom, ne

fonctionnant pas d'emblée correctement, il a été nécessaire de détecter et corriger les différentes erreurs. De même, il a fallu vérifier toutes les entrées possibles d'un utilisateur afin que celui-ci respecte les règles. D'autre part, la gestion graphique a également été une partie non négligeable puisque prenant beaucoup de temps et de réflexion.

Par la suite, j'ai également été en charge de l'ébauche de la gestion du mode joueur anonyme, joueur pour lequel deux options sont possibles : rejoindre la partie en cours si elle existe, ou bien en créer une nouvelle si aucune n'est en cours. Il est à noter que Clément s'est chargé de terminer cette tâche, tout comme il a contribué à l'implémentation des jeux sur le serveur.

Difficultés rencontrées

Une des principales difficultés lors de la réalisation de ce projet concerne la création d'une API, API que la plupart des autres groupes n'avaient pas à faire. Cette partie de notre travail n'a, par ailleurs, pas été facile dans la mesure où nous n'avons eu que très peu d'explications quant à son concept et un manque d'accompagnement en matière d'exemples qui auraient pu nous être utiles.

La conséquence directe de ce que je viens d'évoquer a été une charge de travail considérable, nous obligeant à passer des journées entières à travailler tous sur ce projet. Ceci a été d'autant plus compliqué par le contexte de ce deuxième confinement, faisant que nous devions tous gérer à distance les uns des autres. De plus, cela s'est fait en partie, au détriment d'autres matières, pour lesquelles nous avons eu moins de temps pour étudier.

Enfin, outre le manque d'accompagnement évoqué précédemment, je trouve que nous n'étions pas suffisamment préparés à de telles tâches et qu'il serait bénéfiques que le cursus de première année nous y prépare mieux.

Conclusion

Bien qu'avisée que le projet informatique représente une part importante du travail abordé lors du premier semestre de deuxième année à l'ENSAI, je n'aurais jamais imaginé qu'il puisse être à ce point chronophage... La satisfaction d'être parvenus au terme des tâches à réaliser et d'avoir réussi à concevoir des applications qui fonctionnent n'en aura alors été que meilleure.

Pour terminer, comme le stipule le proverbe : « l'union fait la force », donc un merci tout particulier à mes camarades pour leur gentillesse, leur disponibilité, leur confiance... et tout ce travail en « bonne intelligence » qui a été des plus productifs.

Table des figures

Annexes

Table des Annexes

A Information

Si vous souhaitez utiliser l'application sans avoir à vous créer de compte, trois utilisateurs sont à votre dispositions :

Identifiant	**Mot de passe**	**Pseudo**
admin1	1	pseudo1
admin2	2	pseudo2
admin3	3	pseudo3

Si vous souhaitez vous créer votre propre compte veillez à bien respecter les consignes données dans la suite.

B Guide d'utilisation

Bienvenue sur l'application *SteamEnsai*, une plateforme où vous pouvez jouer contre vos amis, mais aussi contre des inconnus.

Ce guide a été créé dans le but de faciliter votre utilisation de l'application. Ainsi, si vous n'avez jamais utilisé *SteamEnsai*, ce guide est fait pour vous.

Dans ce qui va suivre, nous vous présenterons les fonctionnalités de base, présentes dans le menu initial. Puis nous vous ferons un panel de toutes les rubriques disponibles une fois connecté. Enfin, nous vous montrerons comment lancer une partie et comment y jouer.

B.1 Lancement de l'application

Au lancement de l'application, vous êtes un utilisateur anonyme, il faut donc vous identifier, alors trois options s'offrent à vous.

Se connecter :
Sélectionnez *Me connecter* via les flèches directionnelles. Si vous avez déjà un compte, il suffira juste de rentrer votre identifiant et votre mot de passe. En cas d'erreur de saisie, il vous sera proposé soit de réessayer, soit de revenir au menu initial.
Attention ! Vous ne pourrez pas vous connecter, si vous êtes déjà connecté sur l'application.

S'inscrire : Sélectionnez *Créer un compte utilisateur*. Il vous sera demandé d'entrer quatre paramètres :

1. Votre identifiant.
2. Votre mot de passe.
3. Confirmation de votre mot de passe.
4. Votre pseudo.

Si l'*identifiant* ou le *pseudo* est déjà utilisé, il vous sera proposé à nouveau de vous inscrire. Pour le *mot de passe*, ce dernier doit contenir au moins huit caractères dont :

i Une lettre majuscule

ii Une lettre minuscule

iii Un chiffre

iv Un caractère spécial parmis les caractères suivants ! @ # $ % & _ = ?

Si la confirmation du mot de passe est différente de ce dernier, cela vous le sera signalé. Il vous sera alors proposé, à nouveau, de vous inscrire. Une fois inscrit, vous serez redirigé vers la partie *Me connecter*.

Attention ! Il sera possible de modifier votre *identifiant* et *votre mot de passe* par la suite.

Si vous le souhaitez vous pourrez aussi **quitter l'application.**

Une fois connecté, un message de confirmation vous sera envoyé. Vous serez redirigé vers le menu "connecté", où vous pourrez soit jouer, soit accéder à votre profil.

FIGURE 15 – Message d'accueil et menu initial

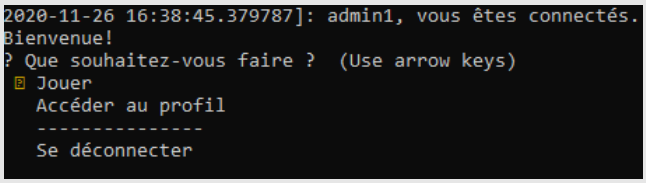

FIGURE 16 – Message de bienvenue et menu connecté

B.2 Accès au menu connecté, consultation du profil

Dans cette partie nous présenterons la partie consultation de votre profil. Quatre options s'offrent à vous, comme vous pouvez le voir sur la figure 17.

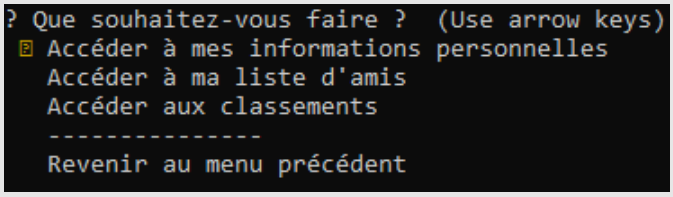

FIGURE 17 – Menu consultation du profil d'un joueur connecté

Accéder à vos information personnelles :
Ici en sélectionnant *Accéder à mes informations personnelles* vous pourrez effectuer les tâches suivantes :

I *Modifier mon pseudo* : cette modification ne nécessite pas de prérequis, mais attention, comme lors de l'inscription, vous ne pourrez pas sélectionner un pseudo déjà existant. Si c'est le cas, une alerte vous sera transmise.

II *Modifier mon mot de passe* : ici il vous sera d'abord demandé votre précédent mot de passe, ensuite votre mot de passe enfin la confirmation de ce mot de passe. Attention le nouveau mot de passe doit respecter les mêmes conditions que le mot de passe initial.

III *Accéder à ses statistiques personnelles* : en choisissant cette possibilité, il vous sera affiché dans un tableau le nombre de parties jouées, le nombre de partie gagnées ainsi que le pourcentage

de victoire.

IV *Réinitialiser ses statistiques personnelles* : cette possibilité vous permet de réinitialiser vos scores ; **attention** cette action est irrévocable, une fois vos statistiques effacées, elles ne pourront être récupérées.

V *Revenir au menu précédent.*

Accéder à votre liste d'amis

En sélectionnant *Accéder à votre liste d'amis* vous aurez la possibilité de voir, ajouter, et supprimer un ami. Plus précisément vous pourrez :

I *Ajouter un ami* : cet ajout ne nécessite pas de prérequis, il ne faut pas qu'il y ait réciprocité de la relation. En effet, il suffit juste de rentrer le pseudo de la personne que vous voulez ajouter. Une fois cette tâche effectuée, le pseudo sera ajouté à la liste d'amis. Si le pseudo entré n'existe pas ou si le pseudo est déjà dans votre liste d'amis, un message vous le signalera.

II *Supprimer un ami* : de même que l'ajout, la suppression d'ami ne nécessite pas de réciprocité. Il suffit juste de rentrer le pseudo de l'ami que vous voulez supprimer. Si le pseudo entré n'existe pas ou si le pseudo n'est pas dans votre liste d'amis, un message vous le signalera.

III *Afficher ma liste d'amis* : en choisissant cette possibilité, il vous sera affiché dans un tableau la liste de vos amis, la date d'ajout, s'il est connecté et s'il est en partie. Si vous n'avez pas d'ami, ce tableau sera vide.

IV *Revenir au menu précédent.*

Accéder aux différents classements

En sélectionnant *Accéder aux classements* vous aurez la possibilité de voir les classements suivants :

I *Afficher le classement général* : il vous sera renvoyé le top 10 mondial tous jeux confondus et votre position par rapport à ce classement. Il sera également transmis le même classement mais entre vous et vos amis.

II *Afficher le classement du jeu de l'oie* : il vous sera renvoyé le top 10 mondial du *jeu de l'Oie* et votre position par rapport à ce classement. Il sera également transmis le même classement mais entre vous et vos amis.

III *Afficher le classement du puissance 4* : même procédé que le classement précédant mais pour le *Puissance 4*.

Vous pourrez aussi revenir au menu précédent.

Revenir au menu précédent : Si vous revenez au menu précédant, vous aurez la possibilité lancer une partie, ce qui sera détaillée dans la section suivante.

B.3 Accès au menu connecté, jouer contre des adversaire

Si vous choisissez de jouer, vous aurez ensuite à choisir le jeu auquel vous voulez jouer, entre le *Puissance 4* et le *Jeu de l'Oie*.

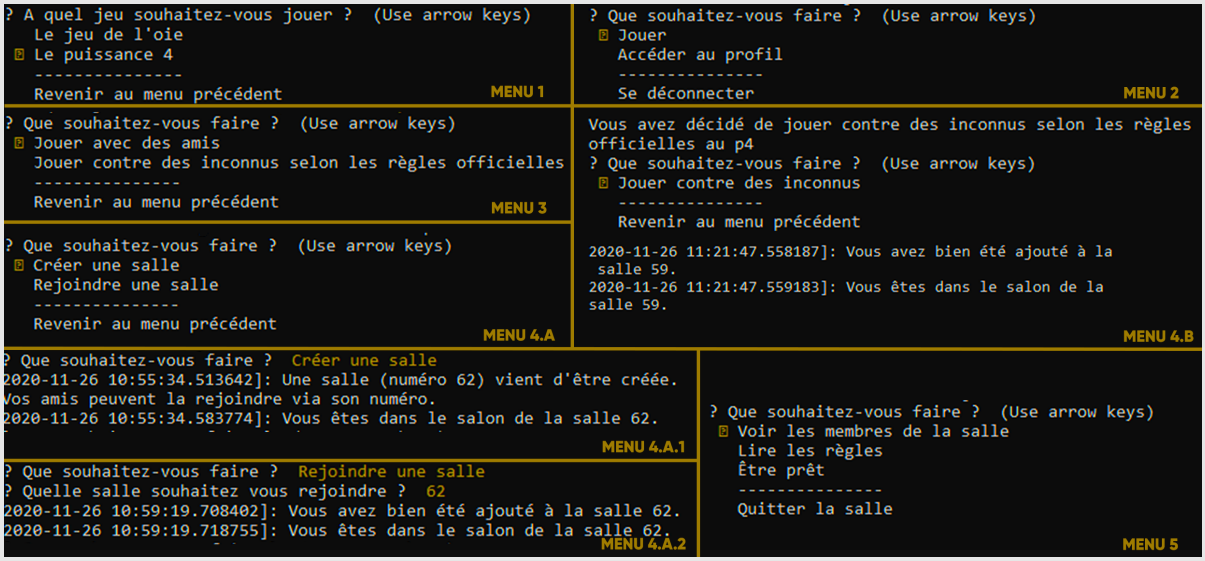

FIGURE 18 – Chemins possibles pour débuter une partie, cas du puissance 4

Comme précisé sur la figure 18, différents chemins s'offrent à vous pour jouer aux jeux disponibles. En effet, après avoir choisi le jeu auquel vous voulez jouer, vous aurez à disposition plusieurs possibilités.

Si vous voulez jouer contre vos amis (Menu 4.A) : vous pouvez soit rejoindre une salle gérée par un ami, soit créer votre propre salle.
Si vous choisissez *Créer une salle* (Menu 4.A.1) Une salle sera créée avec un numéro particulier, et vous y serez le chef.
Attention ! étant chef de la salle, vous ne pourrez pas la quitter tant qu'un autre joueur sera aussi dans cette salle.

Vous pouvez donc aussi décider de *Rejoindre une salle* d'un ami (Menu 4.A.2), pour cela, il suffit juste de rentrer le numéro de cette salle.
Attention ! Si la salle de cet ami est faite pour un jeu différent que celui que vous avez choisi, vous ne pourrez la rejoindre

Si vous voulez jouer contre des anonyme (Menu 4.B) : Il suffit de sélectionner *Jouer contre des inconnus selon les règles officielles* puis sur *Jouer contre des inconnus* ainsi vous rejoindrez une salle pour jouer contre des anonymes.
Attention ! Il se peut que vous soyez chef de cette salle, dans ce cas, de nouveau, vous ne pourrez

la quitter tant qu'un autre joueur sera aussi présent.

Comme pour la partie consultation du profil, à tout moment vous pouvez *Revenir au menu précédent.*

Si vous ne faites pas ce dernier choix, vous arriverez dans une salle où, quelque soit le chemin parcouru, sera affiché le Menu 5.
Une fois dans ce menu, vous pourrez accéder à ces différentes options :

I *Voir les membres de la salle* : il vous sera alors affiché un tableau contenant tous les membres de la partie, ainsi, si vous êtes seul, il n'y aura que votre pseudo.

II *Lire les règles* : ici, vous sera affiché un texte résumant les règles du jeu que vous avez choisi.

III *Être prêt* : cette option vous permettra de vous mettre prêt à démarré la partie. Par la suite vous pourrez choisir la couleur qui vous représentera dans la partie.
Attention ! Si vous êtes seul dans la partie, vous ne pourrez vous mettre prêt. De plus si la couleur choisie est déjà prise, il vous sera demandé d'en choisir une nouvelle.

Vous avez également la possibilité de *quitter la salle* si vous le souhaitez.

Une fois tous les joueurs prêts, la partie se lance.

B.4 Lancement d'une partie, début et fin.

Au lancement de la partie, que ce soit au *Puissance 4* ou au *Jeu de l'Oie*, un message vous signalera si c'est votre tour ou non. Lorsque se sera le cas, des spécificités apparaîtront selon le jeu.

Au puissance 4 : La règle est simple, il vous faut aligner quatre de vos jetons quelque soit le sens (diagonal, horizontal ou vertical). Pour cela il vous suffira juste de choisir la colonne dans laquelle vous souhaitez mettre votre jeton.
Attention ! il peut vous être demandé de rejouer pour deux raisons : soit la colonne choisie est pleine, soit le numéro de colonne choisie est incorrect. Les règles sont plus précisément définies par la suites (sous section B.5.1).

Au jeu de l'Oie : Pour gagner la partie il faut arriver pile sur la case 63, sauf que certaines cases ont des règles particulières. Ainsi, lorsque ce sera votre tour, si vous pouvez jouer, il vous sera demandé d'appuyer sur entrée, ainsi deux dés de six faces seront lancés et la positions de votre pion évoluera en fonction des règles. Les règles sont plus précisément définies par la suites (sous section B.5.1).
Attention ! Il se peut que ce soit votre tour, mais que vous ne puissiez pas jouer, en fonction des règles, ne soyez pas étonné.
A la fin de la partie, vous serez redirigé vers le menu connecté. Si vous avez gagnez la partie , vous

remporterez 20 points, à l'inverse si vous perdez, il vous sera retiré 10 points. Pour information à la création du compte il vous est attribué 1000 points pour chaque jeux

B.5 Règles des jeux

B.5.1 Puissance 4

Les joueurs choisissent chacun une couleur de pion. Le joueur qui commence met un premier pion dans l'une des colonnes de son choix. Le jeton tombe au bas de la colonne. Son adversaire insère à son tour un jeton, le but étant de contrer l'autre au fur et à mesure du jeu pour qu'il n'arrive pas à former une rangée de 4 jetons de sa couleur, dans un sens, comme dans l'autre et en diagonale.

Le joueur qui arrive à aligner 4 pions est le gagnant. La partie est nulle et recommencée si aucun des deux n'y est arrivé et que la grille est remplie.

Bon match !

B.5.2 Jeu de l'Oie

Une partie se joue entre deux et cinq joueurs. Ils jouent chacun leur tour. A chaque tour, ils lancent deux dés et avancent leurs pions du nombre de cases correspondant.

— S'ils tombent sur une case "Oie" (Goose), ils avancent à nouveau d'autant de case que leur lancé.

— S'ils tombent sur une case "Pont" (Bridge), ils rejoignent l'autre case pont, que cela les fassent reculer ou avancer.

— S'ils tombent sur une case "Hôtel", ils passent deux tours.

— S'ils tombent sur une case "Puits" (Well), ils sont bloqués jusqu'à qu'un joueur tombe sur cette case et prenne leur place. Cette case n'est disponible qu'en jouant à plus de deux joueurs.

— S'ils tombent sur une case "Prison" (Jail), ils sont bloqués jusqu'à qu'un joueur tombe sur cette case et prenne leur place.

— S'ils tombent sur une case "Labyrinthe" (Labyrinth), ils reculent de 12 cases.

— S'ils tombent sur une case "Tête de mort" (Skull), ils recommencent au début.

— Si leur deux dés forment un 6 et un 3, ils se rendent sur la case Dice63, que cela les fassent avancer ou reculer.

— Si leurs deux dés forment un 4 et un 5, ils se rendent sur la case Dice54, que cela les fassent avancer ou reculer.

— Pour gagner, il faut tomber exactement sur la case 63. Si vous la dépassez, vous devez reculer d'autant de cases que vous ne l'avez dépassée..

Bonne partie!