MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

UNIVERSITÉ PARIS CITÉ
UFR MATHÉMATIQUES ET INFORMATIQUE

Rapport de projet

Option Informatique 2 (L1)

Roland CHEN 1
Nicolas COQUIL-AMMELOOT 2
Jordan CRISOSTO 3
Jennifer ZAHORA 4

Encadré par Jérôme DELOBELLE

Année universitaire 2022 – 2023

Table Des Matières

1. [Projet 1 - Jeu des échelles et des serpents] 3
1.1 Description du projet 3
1.1.1 Description 3
1.1.2 Objectifs 3
1.2 Implémentation 4
1.2.1 Choix de conception 4
1.2.2 Fonctions 4
1.3 Remarques (optionnelle) 8
2. [Projet 2 - Qwixx] 9
2.1 Description du projet 9
2.1.1 Description 9
2.1.2 Objectifs 10
2.2 Implémentation 10
2.2.1 Choix de conception 10
2.2.2 Fonctions 11
2.3 Remarques (optionnelle) 14
3. [Déroulement du projet] 15
3.1 Travail de groupe en général 15
3.2 Répartition le travail 15
3.2.1 Jeu des échelles et serpents 15
3.2.2 Qwixx 15
3.3 Coding 16
3.3.1 Jeu des échelles et serpents 16
3.3.2 Qwixx 16
3.4 Difficultés 16
4. [Conclusion] 17
5. [Annexe] 18
Annexe 1 18
Annexe 2 18

1. [Projet 1 - Jeu des échelles et des serpents]

1.1 Description du projet

1.1.1 Description

Le jeu des serpents et des échelles est un jeu de plateau qui se joue entre 2 à 4 joueurs.

BUT

Être le premier joueur à arriver à la case 100.

POUR COMMENCER

Les joueurs lancent un dé, celui qui obtient le plus grand nombre commence.

Le plateau est constitué de 100 cases, la case 1 est la case de départ et la case 100 est la case d'arrivée. Le joueur avance sur la plateau en lançant le dé. Atterrissez sur une case échelle, il vous fait avancer plus loin, mais évitez les cases serpents qui vous font reculer.

RÈGLE+

Si le joueur obtient un 6, il avance de 6 cases, si la case est vide il pourra relancer (au maximum 3 fois 6 de suite) et si la case est à effet alors il reçoit l'effet de la case avant de relancer.

1.1.2 Objectifs

L'objectif principal est de modéliser le jeu au format textuel, d'implémenter des IA de sorte qu'on puisse jouer contre eux, et générer des statistiques en faisant jouer uniquement des IA. On doit pouvoir simuler des milliers de parties pour calculer: (1) Les coups moyens dans un jeu et (2) Le pourcentage de gain selon chaque position de joueur sur 4 joueurs.

Enfin, on doit implémenter une fonction de sauvegarde qui permet à l'utilisateur de sauvegarder la progression du jeu et de pouvoir la reprendre plus tard.

1.2 Implémentation

1.2.1 Choix de conception

Notre premier objectif est de modéliser les principaux éléments qui constituent le jeu.

ÉLÉMENTS PRIMAIRES

Dans ce jeu, il n'y a pas d'interaction entre joueurs.

Les éléments principaux sont : le dé, les cases d'effets, les joueurs et leur position.

Le dé est codé par la fonction «dice_base()» qui retourne aléatoirement un entier entre 1 et 6.

Les cases spéciales sont stockées dans des dictionnaires, car son format facilite le stockage des données avec un accès rapide à celle-ci, en entrant comme clé est la case affectée et sa valeur est la case d'arrivée.

Les joueurs sont stockés dans une liste (ordonnée) «LISTE_JOUEUR», et leur position est enregistrée dans une liste (ordonnée) «score_joueur», où chaque joueur est associé à un numéro du plateau. Sachant qu'une liste peut être triée, contrairement au dictionnaire, alors on peut l'utiliser pour ranger l'ordre des joueurs.

Tous les joueurs commencent par le numéro 0, ils lancent un dé qui donne un chiffre qui s'ajoute à leur numéro. Par exemple, si le joueur commence à la case 2 et lance un 5, il sera sur la case 2 + 5 = 7.

ÉLÉMENTS SECONDAIRES

D'autres éléments importants, comme le mouvement et la condition de victoire/ d'arrêt ont été codés dans des fonctions.

1.2.2 Fonctions

Dans cette section, nous avons inclus les fonctions les plus importantes du jeu: celles des différents modes de jeu, de l'initialisation des joueurs, de fin de partie, celles qui font tourner le jeu, et les fonctions de sauvegarde.

MODE DE JEU ET CODES PRINCIPALS

«choice_gamemode()» : Sélection du mode de jeu - jouer avec des humains, contre un IA, ou faire jouer les IA.

«game_no_bot()» : Jeu local entre joueurs.

«game_with_bot()» : Jeu solo contre un bot.

«game_full_bot()» : Jeu automatique, 4 bots qui s'affrontent.

L'utilisateur peut saisir le nombre de parties qu'il souhaite exécuter, et les statistiques lui seront présentées.

SÉLECTION LES JOUEURS ET INITIALISATION

«add_player()»: L'utilisateur saisit le nombre de joueurs, entre son nom et via «get_noms_joueurs()».

«decide_order()»: Les joueurs lancent chacun leur tour pour déterminer l'ordre des joueurs, et la fonction retourne cet ordre.

«who_starts()» : Vérifie l'ordre des joueurs - il est possible qu'il y ait des doublons ou des triplés (voire des quadruplets). Dans ce cas, les joueurs qui ont obtenu le même chiffres relancent un dé pour se départager (voir Annexe 1 pour un exemple).

TOUR PAR TOUR

Une fois l'ordre défini, on utilisait une boucle «for» pour modéliser le tour par tour (*boucle)(fig.1). Lorsque la partie se termine, «running» devient False.

```
running = True
while running:
    for player_number, player_name in enumerate(ordre_joueur):
```

Figure 1. Boucle «while» pour faire le jeu tourner, et «for» pour le tour par tour

CONDITION VICTOIRE / D'ARRÊT

«check_win()» : Cette fonction vérifie si le joueur est sur la case 100 et annonce le gagnant.

«need_exact_win()» : Cette fonction recule le joueur s'il dépasse l'arrivée.

LES CASES À EFFET ET DÉPLACEMENT

«square_effect()» : active l'effet de la case sur laquelle un joueur tombe. Si le joueur tombe sur une case à effet, la fonction correspondante («got_echelle()» s'il tombe sur une échelle et «got_serpent()» si sur un serpent) mettra à jour sa position.

«play()» : permet au joueur de lancer le dé, mettre à jour sa position, vérifier si la partie est terminée, et répéter l'action si le joueur obtient un 6.

Procédure pas à pas:

```
###
#1 Move
dice = dice_base()
score_joueur += dice
```

Figure 2. Lancer le dé

(1) On demande au joueur de lancer son dé (fig.2).

```
#2 Check win condition exacte 100
if check_win(score_joueur, player_name):
    return score_joueur, False
```

Figure 3. Vérifier la victoire

(2) Ensuite, on vérifie si le joueur a atterrit sur la case 100, dans ce cas, la fonction retourne le score du joueur et «running» devient False (fig.3).

```
#3 If score_joueur > 100
if score_joueur + dice > 100:
    score_joueur = need_exact_win(score_joueur)
```

Figure 4. Faire le joueur reculer

(3) Dans le cas où le joueur a dépassé la case 100, alors il est ramené à 100 - dice_restant (fig.4).

```
#4 Effects
score_joueur = square_effect(score_joueur, bots)
###
```

Figure 5. Effectuer l'effect

(4) Si le joueur n'est pas prêt de gagner, alors on vérifie sa case d'arrivée et affecte l'effet de la case (fig.5).

```
if dice != 6:
    return score_joueur, running
```

Figure 6. Le dé n'était un 6

Après avoir reçu l'effet de la case, la fonction retourne la position actuelle du joueur et l'état de la partie (fig.6).

Nous avons utilisé une méthode d'inversion pour nous retrouver avec une succession de validation de conditions, ce qui évite d'avoir des if-else dans des if-else.

Par la suite, il reste à vérifier lorsque le joueur obtient un 6 (rappel: règle des 3 relances).

La fonction «play()» est en partie une fonction récursive, lorsque le joueur obtient un 6, «play()» est rappelée avec relances += 1.

```
if relances == 3:
    print("Vous avez atteint la limite de relancés")
    return score_joueur, running
```

Figure 7. Condition d'arrêt (1) lorsque relances = 3

```
    score_joueur, running = play(score_joueur, player_name, relances)

return score_joueur, running
```

Figure 8. Récursion + condition d'arrêt (2)

INTELLIGENCE ARTIFICIELLE

L'IA effectue uniquement l'action d'appuyer sur ENTER pour lancer son dé. Le reste du programme calcule automatiquement son score et sa position dans le jeu.

SAUVEGARDE

«ask_to_load()» : elle demande à l'utilisateur, en tout début, de CONTINUER une partie existante avec «choose_load_file()» et «load_game()» ou de COMMENCER une nouvelle partie.

«order_saved_game()» : elle permet de sauvegarder l'ordre de la partie précédente et du joueur qui devait jouer.

La sauvegarde de la partie est possible au tour de n'importe quel joueur, en tapant "save" ou "sauvegarder". La fonctionnalité utilise le répertoire courant du fichier et nécessite l'existence d'un dossier prédéfini, "*saved_jeu_d_echelles*".

«choose_save_file()» : elle demande à l'utilisateur de choisir un emplacement de sauvegarde : (save1, save2, save3), il peut renommer sa sauvegarde s'il le souhaite.

«save_game()» : elle sauvegarde le jeu en fichier json. Les données enregistrées sont une liste contenant une liste par joueurs, contenant leur nom, leur position, et le nombre de tours qu'ils ont terminés.

1.3 Remarques (optionnelle)

AFFICHAGE

«update()» : elle met à jour la position du joueur sur le plateau en utilisant des fonctions «replace()» (qui efface la position précédente du joueur) et «update_plateau()» (qui place le pion dans sa nouvelle position)

«affiche_plateau()» : elle affiche la position des joueurs sur le plateau.

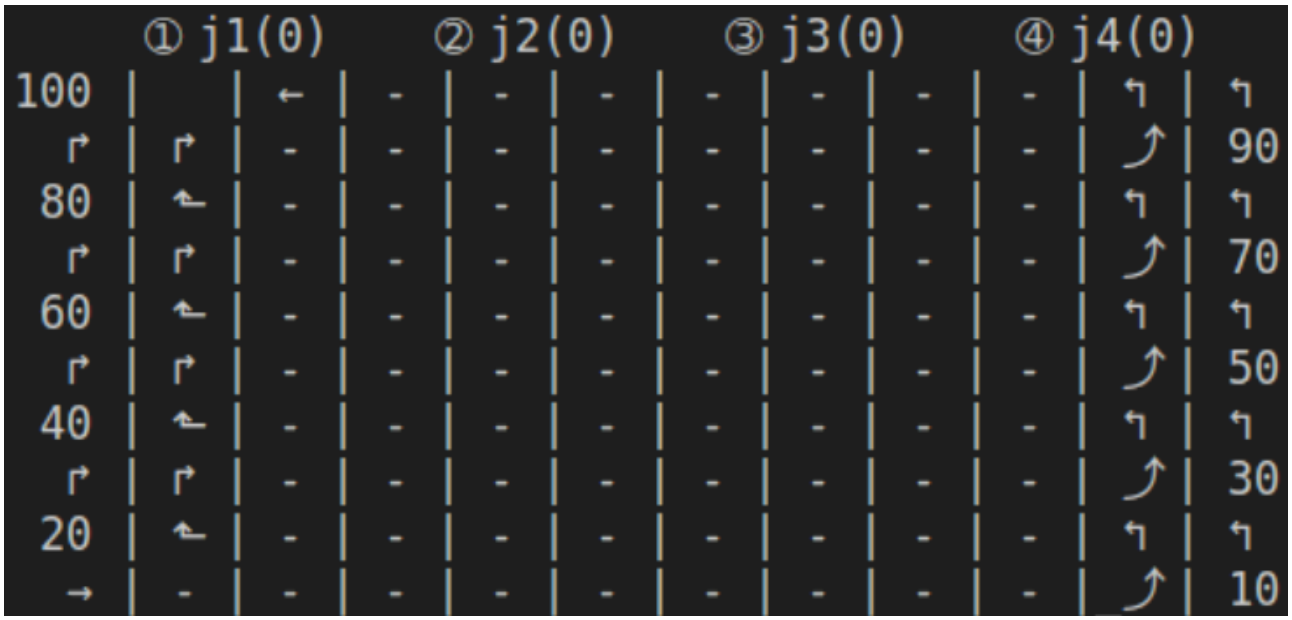

Figure 9. Déplacements

FONCTION TIME() ET AFFICHE()

La fonction «time()» désactive la fonction «sleep()» et la fonction «affiche()» désactive la fonction «print()» si ce n'est pas un humain qui joue.

2. [Projet 2 - Qwixx]

2.1 Description du projet

2.1.1 Description

Le jeu du Qwixx se joue sur une fiche de Qwixx (voir Annexe .2) qui se joue de 2 à 5 joueurs.

BUT

Avoir le plus de points en ayant le maximum de cases cochées et en essayant d'avoir le moins de pénalités.

POUR COMMENCER

L'ordre de lancer suit la liste de joueurs, le joueur actif joue en tout dernier. A chaque tour le joueur actif lance les six dés, celui-ci pourra cocher la somme des dés blancs et cocher la somme d'un dé blanc avec un dé coloré. Le reste des joueurs (les joueurs passifs) ne peut que cocher la somme des dés blancs ou passer le tour, sans conséquence.

Pour fluidifier le jeu, nous avons choisi de faire jouer le joueur actif après les joueur passifs.

RÈGLE+

Le joueur a la possibilité de cocher le numéro le plus à droite d'une ligne, à condition d'avoir au moins 5 croix sur cette ligne, il verrouille la ligne et retire le dé de cette couleur.

Le joueur actif peut passer ses deux phases de jeu mais il reçoit une pénalité !

Le jeu se termine lorsque 2 dés sont retirés du jeu ou si un joueur a accumulé 4 pénalités.

2.1.2 Objectifs

L'objectif principal est de modéliser le jeu au format textuel, d'implémenter des IA de sorte qu'on puisse jouer contre eux, et générer des statistiques en faisant jouer uniquement des IA. On doit pouvoir simuler des milliers de parties pour calculer:

(1) Le nombre de tours en moyenne dans un jeu.

(2) Le pourcentage de gain selon chaque position de joueur sur 5 joueurs.

(3) La ligne la plus fermée.

(4) La condition d'arrêt du jeu le plus fréquent.

Enfin, on doit implémenter une fonction de sauvegarde qui permet à l'utilisateur de sauvegarder la progression du jeu et de pouvoir la reprendre plus tard.

2.2 Implémentation

2.2.1 Choix de conception

Dans ce projet, la discussion sur la façon d'implémenter les éléments était difficile.

ÉLÉMENTS PRIMAIRES

Pour stocker les dés, nous avons hésité entre un dictionnaire (couleur: valeur) et une liste, le premier format était facile à stocker, le deuxième était rapide à créer grâce à une compréhension. Finalement, nous avons décidé de choisir la liste car les couleurs seront rangées (fig.10), et on pourra récupérer l'indice d'une couleur qui servira plus tard.

La fiche du joueur est représentée comme une liste de listes (fig.10), où chaque liste imbriquée représente une ligne, cela facilitera son affichage, par ailleurs, le nombre de pénalités est un entier qui est placé à la fin de sa liste.

```
                    index:0     index:1     index:2     index:3     index:4     index:5
liste_couleur = ["rouge", "jaune", "bleu", "vert", "blanc1", "blanc2"]
fiche_joueurs = [[[], [], [], [], 0] for joueur in liste_joueurs]
```

Figure 10. Listes de couleurs et de fiches

Lorsque le joueur coche une case le numéro coché de la ligne «i», il est ajouté à la liste de liste d'indice «i».

2.2.2 Fonctions

Dans cette section, nous avons inclus les fonctions les plus importantes du jeu qui étaient largement différentes de celles du 1er projet: celles de base, de fin de partie, et celles qui font tourner le jeu.

SÉLECTION DES JOUEURS ET INITIALISATION + TOUR PAR TOUR + SAUVEGARDE ET MODE DE JEU ET CODES PRINCIPALS

Identique au jeu des échelles et des serpents, à part que «add_player()» est la seule fonction utilisée pour ajouter des joueurs et que le nom du dossier de sauvegarde est "saved_quixx".

FONCTIONS DE BASE

«get_dices()»: Elle renvoie un entier de 1 à 6 qui est stocké dans la liste de dé (chaque couleur).

«playing_order()»: Elle réitère l'ordre des joueurs et de leurs fiches, de manière à ce que le joueur actif soit le dernier à jouer.

«lock_line()»: Elle permet de verrouiller une ligne et «remove_dice()» retire le dé de la même couleur que la ligne. Lorsque qu'un dé est retiré du jeu, la couleur est changée en None dans la liste de couleur et dans la liste des dés.

CONDITION VICTOIRE / D'ARRÊT

«check_end()»: Elle vérifie si un joueur possède 4 pénalités ou si 2 dés ont été retirés. Dans les deux cas, le jeu s'arrête.

PROPOSITIONS ET GAMEPLAY

Le joueur n'écrira pas le coup qu'il veut jouer, au lieu de cela, on a choisi de donner l'ensemble des coups possibles grâce à «propose_white()» et «propose_combo()», de cette façon il pourra choisir dans un panel de possibilité, par ailleurs l'IA n'aura qu'à piocher parmi un ensemble de propositions.

«is_playable()»: Elle vérifie si un numéro est jouable sur une ligne. La ligne fonctionne comme un file, la dernière valeur ajoutée est contrôlée et la nouvelle valeur est ajoutée à la fin.

```
if player_fiche[couleur_ind] == []:
```

```
        return True


if 0 <= couleur_ind <= 1 # Lignes jaune et rouge.:
    return player_fiche[couleur_ind][-1] < value
return player_fiche[couleur_ind][-1] > value # Lignes bleue et verte.
```

Figure 11. Un snippet de la fonction «is_playable()»

«place_x()»: Elle permet de cocher un numéro dans une ligne.

Lorsque le joueur actif lance ses 6 dès…

[Phase 1]

«play_white()»: S'il est possible, elle renvoie les possibilités de coups, sinon il passe automatiquement le tour du joueur.

«propose_white()»: Elle retourne une liste de couleurs de ligne jouables. Elle vérifie si la somme des dés peut être cochée sur une ligne avec «is_playable()». Dans le cas de 2 et 12, «add_special()» vérifie la ligne et si il est jouable sur cette ligne. (voir fiche Qwixx)

[Phase 2]

«play_color()»: S'il est possible, le joueur pourra choisir une ligne et un numéro à cocher dans la dictionnaire des possibilités (dé blanc + dé coloré), sinon il passe automatiquement son tour, au bout de 2 skips, le joueur obtient une pénalité.

«propose_combo()»: Elle retourne un dictionnaire où les clés sont les couleurs et leur valeur est une liste contenant des nombres à cocher. Elle parcourt la liste de dés blancs et dés colorés, elle fait la somme et vérifie si elle est jouable grâce à «is_playable()» et «add_special()» (fig.12).

```
if combo in [12, 2]:
    add_special(player_fiche, line_color, combo, type=dict)
# else if it's any other number [3, 4, 5, 6, 7, 8, 9, 10, 11]
elif is playable(player fiche, indice_couleur(line color), combo):
    add_to_combo_dict(line_color, combo)
```

Figure 12. Un snippet de la fonction «propose_combo()»

Lorsque un dé est retiré du jeu, la fonction le prend en compte et passe à la couleur suivante (fig.13).

```
if color_dice is None:
    continue
```

Figure 13. La condition si un dé est retiré

Grâce au système de proposition, l'implémentation de l'IA est simple car elle ne peut que choisir dans une liste de coups déjà créée.

INTELLIGENCES ARTIFICIELLES

Dans le cadre de notre projet, il nous est demandé d'implémenter une IA, nous en avons implémenté deux IAs.

L'IA facile est bête mais elle doit pouvoir jouer avec le joueur, sa stratégie: jouer des coups aléatoires.

[Procédure]

«propose_white()» et «propose_combo()» créent respectivement la liste et le dictionnaire des possibilités du bot, il nous suffit d'utiliser la fonction «random.choice()» sur la liste des propositions + "pass" (fig.14).

```
chosen_color = choice(prop_whites[:-1] + ["pass"]) # passif
chosen_color = choice(liste_couleur_dict + ["pass"]) # actif choisir ligne
choosen_number = choice(prop_color[chosen_color])# actif choisir numéro
```

Figure 14. Choisir un coup

L'IA moyen est plus intelligente, sa stratégie: jouer des coups avec un écart d'au plus 2.

[Procédure]

Pour la phase passive, l'IA passe en revu les possibilités dans prop_whites et elle évalue la somme des dés blancs avec le dernier numéro de chaque ligne (ouverte).

Lorsque l'écart est de 1 (meilleur coup), l'IA renvoie la couleur de la ligne pour la jouer directement. Lorsque l'écart est de 2, l'IA enregistre ce coup auquel cas il n'existe pas un meilleur coup. S'il n'existe ni de bon coup, ni de meilleur coup, l'IA passe son tour.

De la même manière pour la phase active, l'IA joue les coups avec un écart inférieur ou égal à 2, elle prend aussi le risque d'obtenir une pénalité si les conditions ne sont pas réunies.

2.3 Remarques (optionnelle)

AFFICHAGE

«colored()»: Elle colore le texte sur le terminal, grâce au module colorama.

«update_fiche()»: Elle ajoute des 'x' aux cases cochées dans la fiche à afficher.

«affiche_fiche()»: Affiche la fiche de qwixx du joueur (fig,15).

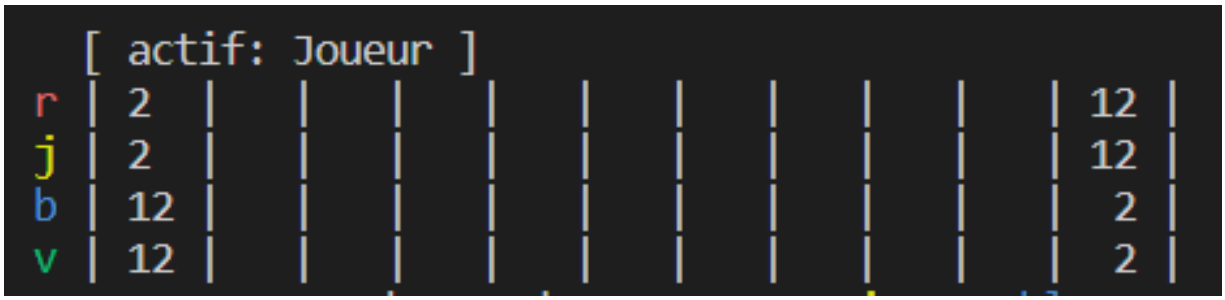

Figure 15. Fiche de joueur actif

FONCTION TIME() ET AFFICHE()

Identique au jeu des échelles et des serpents.

3. [Déroulement du projet]

3.1 Travail de groupe en général

Généralement, nous choisissons de nous réunir entre 12h et 14h le lundi, afin de discuter du déroulement du projet et des objectifs pour la prochaine réunion. Avant la réunion du mercredi, nous vérifions la fonctionnalité du code afin de pouvoir montrer notre avancement au professeur.

3.2 Répartition le travail

3.2.1 Jeu des échelles et serpents

Dans un premier temps, Jordan s'est occupé de créer un prototype du jeu: lancer le dé et ajuster le score du joueur en conséquence. Jennifer a ajouté les fonctions essentielles, puis Roland l'a implémenté sur plusieurs joueurs et a défini la structure principale du code.

Après cela, Roland a implémenté la fonction «time()» et l'affichage du plateau (optionnel), et Jennifer a ajouté la sauvegarde. Nicolas s'est occupé de coder l'IA et du mode solo avec une IA, et Jennifer a implémenté son code. Jordan a ajouté le mode de jeu entre IA et les statistiques.

Après avoir corrigé tous les bugs, Roland a rédigé le readme, Jennifer a affiné le code et a aussi implémenté la fonction «affiche()» à partir du 2ème projet.

3.2.2 Qwixx

Par l'expérience du 1er projet, l'organisation est devenue plus simple. Nous avons gardé plus ou moins la même organisation que pour notre premier projet.

Jennifer et Roland se sont occupés de programmer «game_no_bot()». Nicolas s'est occupé de «game_with_bot()» avec l'IA facile et l'IA moyen, avec l'aide de Jennifer et Roland.

Comme la première fois, Roland a ajouté l'affichage et Jennifer a ajouté les fonctions de sauvegarde. Jordan a codé «game_full_bot()» pour les statistiques, avec l'aide de Jennifer.

Enfin, Jennifer a affiné le code et Roland a rédigé le readme pour le rendu.

3.3 Coding

3.3.1 Jeu des échelles et serpents

Nous utilisons essentiellement la messagerie de groupe de discord afin de nous partager le code. Nous nous envoyions celui-ci chaque fois que nous avions modifier, ajouter ou/et supprimer une fonction. À l'exception de certains cas, tout le monde a codé individuellement. Dans ces cas, nous utilisions codeshare afin de modifier de code “direct”, quand nous travaillions côté-à-côté.

3.3.2 Qwixx

Pour le deuxième projet, Jennifer a proposé de changer d’organisation concernant la communication du code source, car au cours du premier projet, il est arrivé que des personnes travaillent sur une version plus ancienne du code. Pour la plupart, nous avons utilisé Github pour partager le code et les fusionner, ce qui a rendu le flux de travail plus pratique.

Comme dans le cas du 1er projet, une partie importante de ce projet a également été codée individuellement. Lorsqu'il était difficile de répartir les tâches de manière égale ou que la tâche s'avérait plus compliquée, Roland et Jennifer ont codé ensemble.

3.4 Difficultés

La principale difficulté de ce projet était de se partager équitablement les tâches. La distribution de celles-ci, mise de manière abstraite, a eu pour conséquence d’une différence de charge de travail entre membres de notre équipe. Malgré l’expérience de cette difficulté dans le premier projet, par dessus cela, nous avons eu du mal à évaluer la difficulté des tâches distribuées, ce qui a conservé le déséquilibre, encore une fois, de la répartition des tâches.

Au niveau du code, le deuxième projet était plus rude que le premier de par l’augmentation du nombre de fonctions à coder. Nous avons également eu quelques difficultés à maîtriser Github.

4. [Conclusion]

En définitive, malgré les nombreuses difficultés du travail en groupe que nous avons dû surmonter, nous avons réussi à coder les deux projets et nous avons même ajouté des fonctionnalités optionnelles, comme par exemple l'affichage textuel. Ainsi, ce projet nous a aidé pour améliorer notre travail d'équipe et nous a entraîné pour les futurs travaux en groupe que nous devrons faire si nous travaillons dans l'informatique.

5. [Annexe]

Annexe 1

[Procédure pas à pas de who_starts()]

Chaque joueur lance le dé, on obtient: 5(j1) 5(j2) 2(j3) 4(j4).

La fonction «decide_order()» trie une première fois…

```
On a: 5(j1) 5(j2) 4(j4) 2(j3)
[Départagez-vous j1 et j2!]
```

Or j1 et j2 ont tiré le même numéro, ils doivent relancer le dé pour se départager !

```
On a: 5(j1) 4(j2)
['j1', 'j2', 'j4', 'j3']
```

Ainsi la fonction «who_starts()» prend en compte le changement et permute l'ordre des joueurs qui est j1 → j2 → j4 → j3 qui est traduit en [« j1 », « j2 », « j4 », « j3 »].

Cas exceptionnel :

- S'il reste des doublons, alors «who_starts()» réarrange jusqu'à obtenir un ordre définitif.- S'il y a 2 doublons, c'est-à-dire : 2(j1) 2(j2) 4(j3) 4(j4) alors une fonction spéciale «doublon_exception()» va réarranger l'ordre.

Annexe 2