M4104C-PEL : Agilité et bonnes pratiques de programmation

*Compte-rendu de programmation du jeu du Mus :*



URL Github : [TodoniK/Mus (github.com)](https://github.com/TodoniK/Mus)

Groupe n°8



MARIS Xan     DUT INFORMATIQUE

DE LAS HERAS Mathis Semestre 4 | 2021 -2022

ROYET Jules                                                         TD2– TP3

Table des matières

[1.Choix et décisions en gestion de projet 4](#_Toc98157714)

[A. Comment le travail se répartit dans l’équipe ? 4](#_Toc98157715)

[B. Comment faites-vous le suivi des tâches à réaliser ? 5](#_Toc98157716)

[C. Quels outils utilisés ? 5](#_Toc98157717)

[2.Choix et décisions techniques 6](#_Toc98157718)

[A. Implémentation d’une classe équipe dans le programme 6](#_Toc98157719)

[- Diagramme de classe de cette équipe 6](#_Toc98157720)

[- Modification de la classe Opposants et de la classe JeuDuMus 9](#_Toc98157721)

[- Modification de la classe Manche 10](#_Toc98157722)

[- Modification de la classe Partie 11](#_Toc98157723)

[- Modification Partie Test 11](#_Toc98157724)

[B. Difficultés rencontrés au cours projet 12](#_Toc98157725)

[- Gestion de l’algo de tournante 12](#_Toc98157726)

[- Scénario double gehiago 12](#_Toc98157727)

[- Difficultés pour les tests 13](#_Toc98157728)

[C. Parties du code que l’on trouve clean 14](#_Toc98157729)

[- Algo joueurs parlant au prochain tour 14](#_Toc98157730)

[- Code de la classe Equipe 15](#_Toc98157731)

[3. Journal de bord 16](#_Toc98157732)

[A. Compte-rendu de première séance (25/01/2022) : 16](#_Toc98157733)

[B. Compte-rendu de deuxième séance (26/01/2022) : 16](#_Toc98157734)

[C. Compte-rendu de troisième séance (02/02/2022) : 16](#_Toc98157735)

[D. Compte-rendu du travail en autonomie (08/02/2022) : 17](#_Toc98157736)

[E. Compte-rendu de quatrième séance (09/02/2022) : 17](#_Toc98157737)

[F. Compte-rendu de cinquième séance (16/02/2022) : 18](#_Toc98157738)

[G. Compte-rendu de sixième séance (02/03/2022) : 18](#_Toc98157739)

[H. Compte-rendu de septième séance (09/03/2022) : 18](#_Toc98157740)

[4. Conclusion : 19](#_Toc98157741)

Table des illustrations

[Figure 1 : Diagramme de classe conceptuel du 1 vs 1 : 6](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157309)

[Figure 2 : Diagramme de classe conceptuel de notre projet avec le 2 vs 2 7](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157310)

[Figure 3 : Diagramme de classe logique de la classe Equipe 8](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157311)

[Figure 4 : Extrait du code de la classe Equipe 8](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157312)

[Figure 5 : Extrait du code de la classe Equipe 8](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157313)

[Figure 6 : Extrait du code du constructeur d'Opposants 9](#_Toc98157314)

[Figure 7 : Extrait du code de la classe JeuDeMus (attributs) 10](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157315)

[Figure 8 : Extrait du code de la classe Manche (changement attribut Map) 10](#_Toc98157316)

[Figure 9 : Extrait du code de Partie (modification de l'objet vainqueur et classe Resultat) 11](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157317)

[Figure 10 : Extrait du code de Partie (modif constructeur Score) 11](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157318)

[Figure 11 : Code de TestPartie (Ezku doit être gagnant au bout de 3 manches) 11](#_Toc98157319)

[Figure 12 : Extrait de la méthode jouer de la classe Mus 12](#_Toc98157320)

[Figure 13 : Code du test de gestion choix tournante 13](#_Toc98157321)

[Figure 14 : Code du test d'interrogation allié 14](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157322)

[Figure 15 : Constructeur de Deroulement (permettant d'obtenir une liste de joueur devant jouer au prochain tour) 14](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157323)

[Figure 16 : Extrait code classe Equipe 15](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157324)

[Figure 17 : Extrait code classe Equipe 15](file:///\\haya\dossiersetud\jroyet\Downloads\Compte-rendu_de_programmation.docx#_Toc98157325)

# 1.Choix et décisions en gestion de projet

## Comment le travail se répartit dans l’équipe ?

Au début du projet, nous avions décidé de travailler de manière solitaire, car cette première phase du projet visait à comprendre le code du 1 contre 1 et nous avions donc jugé qu’il était préférable que tout le monde ait une machine pour lire ce qui se passait dans le code.

Puis, nous avons décidé de mettre en commun ce que nous avions compris afin de savoir si tout le monde avait plus ou moins saisi comment le code fonctionnait, mais aussi pour confronter nos idées sur la problématique de la réalisation du 2 contre 2.

Dès lors, nous avons entamé, assez naturellement, une phase de mob programming dans laquelle nous codions sur le pc de Mathis, en se donnant des idées au fur et à mesure.

Malheureusement, cette phase de travail a bien marché pour la première partie du code qui consistait à créer une classe Equipe et à remplacer les objets Joueurs par des objets Equipes dans les classes du projet. Une fois que le problème s’est complexifié, notamment pour le codage de la méthode permettant de faire une tournante entre les joueurs, nous n’étions plus trop efficaces en mob programming.

De ce fait, Jules a décidé de s’attarder un peu plus sur la réalisation des documents à rendre pour le projet, notamment ce compte-rendu, et à réaliser le diagramme de classe simplifié, via un plugin UML Maker proposé par le marketplace d’IntelliJ, afin d’y voir plus claire.

Néanmoins, nous avons remarqué qu’ils nous auraient été préférable de concevoir des algorithmes avec Tabularasa avant de se lancer dans des phases de codage comme nous l’avons fait. Nous aurions certainement perdu moins de temps sur le codage de fonctions comme celle de la tournante.

D’autre part, à chaque fin de séance, nous alimentions un petit compte-rendu qui recensait ce que l’on avait fait durant la séance : les problèmes rencontrés, les solutions apportées, le ressenti de l’équipe sur l’avancement du projet…

Avec du recul, cette organisation aurait pu être améliorée en donnant une partie du projet à coder à chacun. Cette phase d’organisation nous aurait certes pris du temps en début de projet, mais ce temps aurait assurément été rattrapé en phase de codage.

## Comment faites-vous le suivi des tâches à réaliser ?

En ce qui concerne le suivi des tâches et l’organisation du travail pour les séances suivantes, cela se faisait assez naturellement et intuitivement. En effet, lors des séances d’agilité, nous nous posions 5 minutes avant la fin afin d’une part, de taper le compte-rendu et d’autre part, savoir ce qui nous restait à faire pour la séance prochaine.

Pour communiquer avec les membres de l’équipe, nous avons créé un groupe Discord (« ZeMus ») sur lequel on postait les différentes versions du compte-rendu au fur et à mesure, mais aussi sur lequel on communiquait par rapport au projet.

## Quels outils utilisés ?

Nous avons utilisé différents outils au cours de ce projet, pour effectuer différentes tâches.

Pour la communication, nous avons essentiellement utilisé Discord.

Pour la rédaction du compte-rendu du projet, nous avons utilisé Word.

Pour le versionning de notre code, nous avons utilisé Git et Github.

Pour la rédaction de notre code, nous avons utilisé l’IDE IntelliJ.

Enfin, pour la réalisation d’éléments de conception tels que le diagramme de classe, nous avons utilisé les plugins natifs, mais aussi un plugin du marketplace d’IntelliJ (« UML Maker »).

# 2.Choix et décisions techniques

## A. Implémentation d’une classe équipe dans le programme

### - Diagramme de classe de cette équipe

Figure 1 : Diagramme de classe conceptuel du 1 vs 1 :

L’implémentation d’une équipe au sein de notre programme de Mus a évidemment été l’une des décisions les plus stratégiques de notre projet. En effet, nous avons décidé de créer cette classe, car elle allait nous apporter du confort et de la logique pour le reste du refactoring du code. Elle nous permet d’autre part de hiérarchiser proprement nos deux joueurs supplémentaires afin de pouvoir mettre en place le 2 vs 2. Nous pouvons donc dire que c’est du clean code.

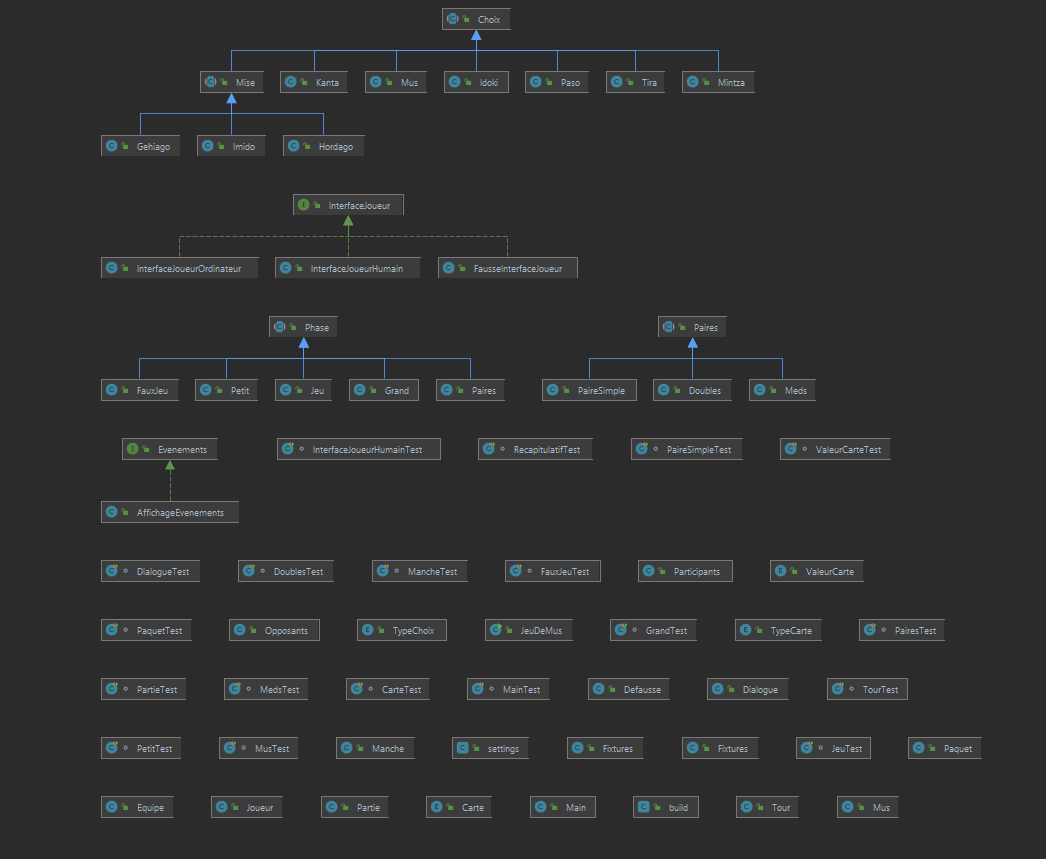


Figure 2 : Diagramme de classe conceptuel de notre projet avec le 2 vs 2

Comme nous pouvons le remarquer, nous n’avons modifié uniquement une seule chose dans la hiérarchie du projet, qui correspond à l’ajout de notre classe Equipe. En effet, tout comme Joueur, elle n’hérite de rien et permet simplement d’obtenir une liste de deux joueurs.

Voici le schéma conceptuel de notre classe :

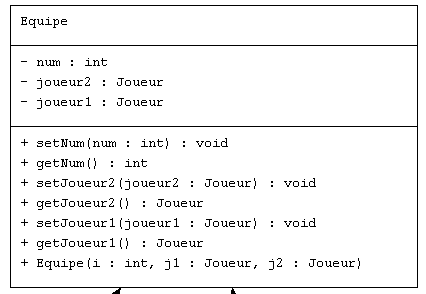


Figure 3 : Diagramme de classe logique de la classe Equipe

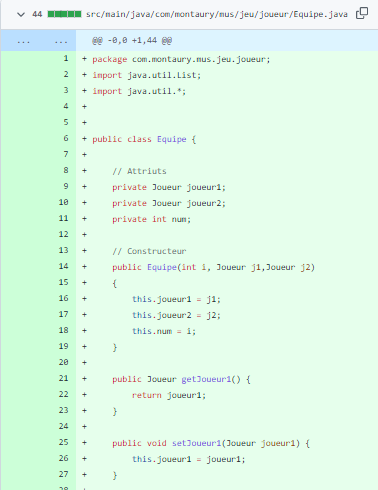
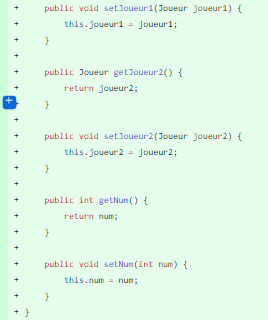


Figure 4 : Extrait du code de la classe Equipe

Figure 5 : Extrait du code de la classe Equipe

L’implémentation de cette nouvelle classe au sein de notre projet nous a demandé beaucoup de travail d’adaptation du code. Nous allons passer en revue quelques modifications que nous avons dû apporter au code pour qu’il soit fonctionnel avec l’équipe.

### Modification de la classe Opposants et de la classe JeuDuMus

Nous avons donc commencé par modifier le code d’Opposants en ajoutant deux joueurs dans ses attributs et modifiant son constructeur afin qu’il initialise à présent une équipe.



Figure 6 : Extrait du code du constructeur d'Opposants

Par conséquent, nous avons dû modifier la classe JeuDuMus afin que le jeu s’initialise avec deux équipes de deux joueurs. Nous avons donc instancié deux équipes composées pour l’une d’un joueur humain et d’un joueur ordinateur (Joueur.ordinateur()), et pour l’autre de deux ordinateurs.



Figure 7 : Extrait du code de la classe JeuDeMus (attributs)

Dès lors, nous avons cherché dans l’ensemble des classes, les parties du code qui utilisaient des objets de type Joueur afin de les remplacer par des équipes.

### Modification de la classe Manche

Nous avons donc modifié la map contenant le score de chaque joueur, qui contient maintenant le score par équipe.

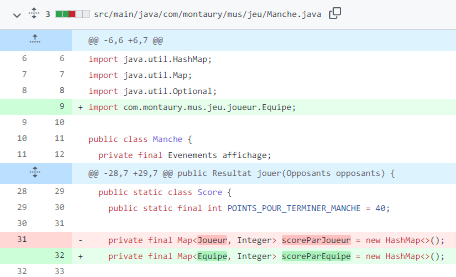


Figure 8 : Extrait du code de la classe Manche (changement attribut Map)

### Modification de la classe Partie

De la même manière, nous avons modifié la classe Partie afin que celle-ci nous retourne désormais les manches gagnées par les équipes.

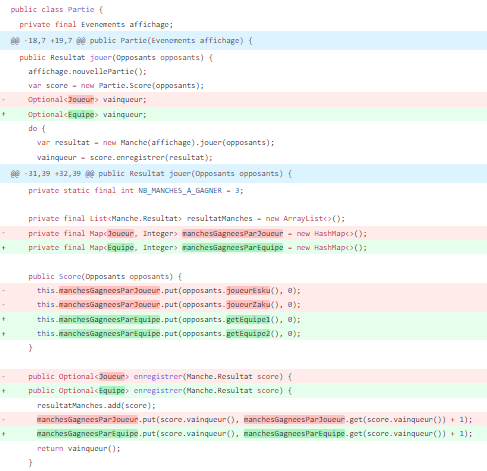
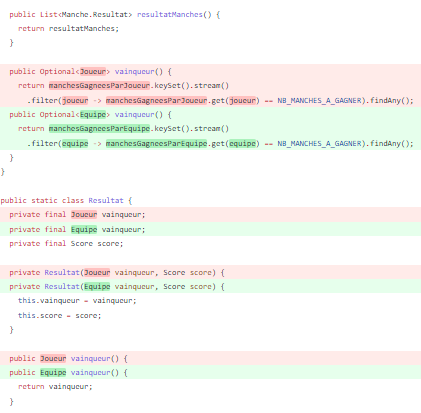


Figure 9 : Extrait du code de Partie (modification de l'objet vainqueur et classe Resultat)

Figure 10 : Extrait du code de Partie (modif constructeur Score)

### Modification Partie Test

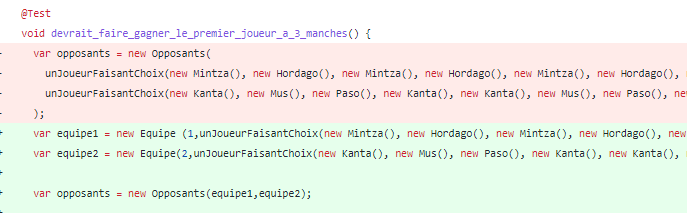


Figure 11 : Code de TestPartie (Ezku doit être gagnant au bout de 3 manches)

Nous avons dû apporter d’autres modifications au fur et à mesure sur d’autres classes mais toutes les présenter reviendrait à passer en revue 80% des commits du projet.

D’autre part, nous avons également modifié l’intégralité des tests déjà mis en place par M.URRUTY afin qu’ils traitent désormais des équipes.

## B. Difficultés rencontrés au cours projet

### - Gestion de l’algo de tournante

Par la suite, nous avons donc été confronté au problème de tournante dans le jeu, pour que nos joueurs Zaku et Ezku soient interrogés l’un après l’autre, puis les autres joueurs de l’équipe dans certains scénarios (par exemple si le Zaku se couche durant une manche, etc..).

Pour cela, nous avons donc dans un premier temps modifié la méthode « jouer » de la classe Mus, afin que celle-ci renvoi l’ordre des joueurs devant parler au prochain tour en fonction de leur position dans la file. En effet, nous avons :

Zaku Coéquipier Zaku Ezku Coéquipier Ezku

0 1 2 3

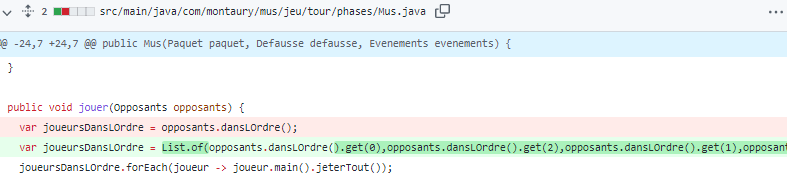


Figure 12 : Extrait de la méthode jouer de la classe Mus

On obtient donc une liste avec l’ordre suivant : Zaku puis Ezku puis Coeq Zaku et finalement Coeq Ezku. Le fait de retourner l’ordre grâce à la position s’apparente à du code smell et nous améliorerons ceci par la suite.

Néanmoins, bien que le jeu soit fonctionnel à 4 grâce à cette modification, nous avons éprouvons toujours des difficultés pour les mêmes scénarios (ceux où l’un des joueurs Ezku ou Zaku se couche) qui à l’heure actuelle ne sont toujours pas résolues.

### Scénario double gehiago

Nous avons quelques difficultés pour le scénario où l’on joue Gehiago, et que l’adversaire suit en faisant également Gehiago. En effet, le jeu ne nous réinterroge pas et le jeu passe à la phase suivante.

### Difficultés pour les tests

Dès lors, nous avons donc essayé de débugger notre code à la main mais aussi de manière automatique, notamment en mettant en place des tests.

Nous avons donc mis en place un test reprenant exactement ce scénario. Dans ce test, le joueur allié du Zaku doit aller Idoki si Zaku se tire (Tira).

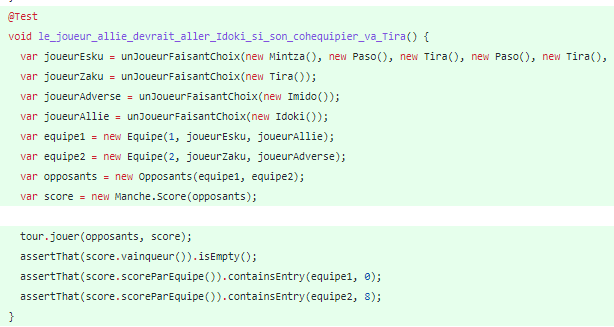


Figure : Code du test de gestion choix tournante

Néanmoins, ce test ne passe pas. Encore à l’heure actuelle, nous éprouvons quelques difficultés à valider ce test car notre liste d’opposants est vide à un moment du test pour une raison inconnue.

Par ailleurs, nous avions des difficultés à identifier des tests qui nous serait utile pour nous aider à comprendre quelles parties du code ne fonctionnait pas. Néanmoins, dans la continuité du précédent test, nous avons mis au point un test permettant de savoir si le joueur allié est interrogé dans le cas où le Zaku se tire.

Finalement, nous n’avons pas éprouvé de difficultés particulières pour le nommage de variable ou de classes.

Figure : Code du test d'interrogation allié

## C. Parties du code que l’on trouve clean

### Algo joueurs parlant au prochain tour

Anciennement, nous créions des instances pour chaque joueur dans nos méthodes et nous affections de manière manuelle les ordres de passage à chacun. Nous avons amélioré ce code d’une part en stockant nos joueurs dans des files (Dequeue) et donc en accédant à chaque joueur grâce à une boucle.

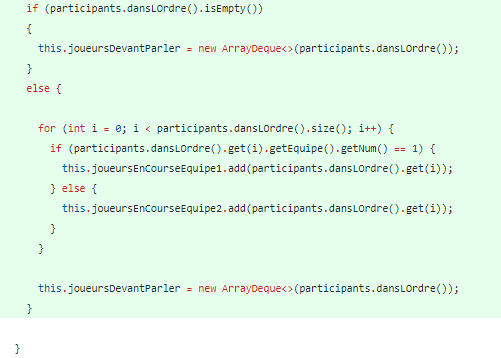


Figure : Constructeur de Deroulement (permettant d'obtenir une liste de joueur devant jouer au prochain tour)

Nous trouvons que cette partie du code est clean car nous avons automatisé l’affectation des joueurs à la Dequeue.

### Code de la classe Equipe

La classe Equipe est également plutôt bien réussi et optimisé du fait qu’elle soit assez simpliste. Nous pouvons donc dire qu’elle s’apparente également à du clean code.

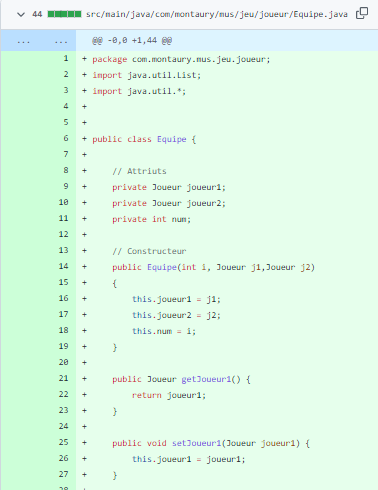
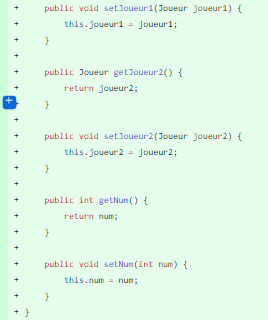


Figure : Extrait code classe Equipe

Figure : Extrait code classe Equipe

Le reste de notre code pourrait éventuellement être amélioré.

# 3. Journal de bord

## A. Compte-rendu de première séance (25/01/2022) :

Durant cette première séance, nous avons eu une première approche du jeu du Mus. C’est un jeu de carte qui se pratique à 1 contre 1 ou le plus souvent à 2 contre 2. Pour espérer gagner une partie, il faut d’abord remporter 3 manches. Pour cela, il faut avoir de la chance (beaucoup), bluffer (à nos risques et périls) et prendre les bonnes décisions. Ce jeu vient avec un ensemble de terme qui nous permettent de jouer comme : «Mintza », « Mus », « Idoki », « Tira ».

M. URRUTY nous a donc dans un premier temps, expliqué les règles du jeu. Puis dans un second temps, nous avons réalisé un certain nombre de parties afin de nous acclimater. Nous avons mis beaucoup de temps à nous familiariser avec les termes à utilisés pour jouer à l’instar des règles qui étaient compliquées à prendre en main notamment pour le jeu en équipe.

## B. Compte-rendu de deuxième séance (26/01/2022) :

Durant cette séance, nous avons eu quelques difficultés pour utiliser l’IDE IntelliJi. En effet, le module grundle était inaccessible. Par conséquent, nous continué de nous informer sur les règles du Mus ainsi que le vocabulaire. Nous avons mis en pratique nos connaissances au cours de quelques parties afin de s’assurer que le but du jeu était bien compris par l’intégralité des membres du groupe.

Finalement, nous avons terminé la séance en regardant avec M. URRUTY, les classes et méthodes présentes dans le code du jeu afin de nous familiariser avec celui-ci. Nous avons donc de nouveau travaillé en groupe et n’avons éprouvé aucunes difficultés.

## C. Compte-rendu de troisième séance (02/02/2022) :

Durant cette séance, nous avons pu commencer à coder car le souci avec Graddle et IntelliJ a été résolu par le CTI.

Nous avons éprouvé quelques difficultés à savoir par où commencer. Nous avons donc commencé par éplucher, chacun de notre côté, le code notamment en utilisant le debugger pour voir les enchainements entre appels de méthodes. Puis, une fois bien dans le bain, nous avons décidé d’organiser notre travail en mob programming en utilisant le PC de Mathis.

Nous avons confronté nos idées et conclu qu’il était nécessaire de faire une classe Equipe. Celle-ci devait comporter des Joueurs que nous avons stockés au départ dans des variables puis par la suite dans une Dequeue. Nous avons également opéré des modifications sur la classe Opposants. La map est maintenant composée d’un objet Equipe et d’un entier. Nous avons adapté les méthodes de calcul du score par Equipe. Nous avons également modifié la définition et réalisation de l’algorithme de tournante avec les rôles des joueurs.

Finalement, nous avons regardé et changer quelques lignes de codes de la classe Partie, afin que celle-ci soit en adéquation avec le nouveau principe d’équipe. Au fil du temps, les idées sont venues d’elle-même et nous n’avons donc pas ressenti de difficultés, hormis pour le choix du type d’une équipe où nous avons débattu pour savoir si on utilisait une map, liste ou file.

## D. Compte-rendu du travail en autonomie (08/02/2022) :

Lors de notre séance de travail en autonomie toujours en mob programming, nous avons mis en place l’affectation de joueurs dans le constructeur d’équipe, ainsi qu’un getter et setter d’équipe. Nous avons donc utilisé ces nouveaux getters dans la classe Phase du jeu. D’autre part, nous avons également veillé à ce que les instances de joueurs possèdent des noms significatifs (classe jeu).

Finalement, nous avons fini de coder la méthode de tournante entre joueurs et vérifié que celle-ci fonctionnait bien. Nous devons donc désormais coder l’attribution des points par manche, puis par partie mais aussi le fait d’isoler les joueurs entre eux et comparer leurs cartes. Pas de difficultés spécialement rencontrées.

## E. Compte-rendu de quatrième séance (09/02/2022) :

Lors de cette séance nous avons poursuivi le travail en mob programming car celui-ci nous réussit du fait que la confrontation de nos idées nous fait avancer assez rapidement. Néanmoins, Jules se sent un peu lésé sur le code. Pour continuer à rester productif, il commencera le comte-rendu sur Word, tout en s’informant sur l’avancé du code et pourquoi pas en donnant quelques idées.

D’autre part, nous avons modifié la plupart des classes de Test notamment les classes TourTest et MancheTest en adaptant le code afin que les objets utilisés (anciennement joueurs) soient remplacés par des équipes. Nous nous sommes attardés également sur la classe MusTest en adaptant les méthodes de cette classe notamment en s'assurant que les joueurs se font distribués des cartes. Par ailleurs, nous avons modifié les méthodes retournant un certain scénario lorsqu'un joueur joue un coup. Nous sommes en train de finir de modifier les autres classes de test afin que ceux-ci passent sans encombre. Une fois que ceux-ci seront validés, nous régleront le problème de création de nouvelle liste contenant seulement les joueurs qui joueront à la manche suivante.

## F. Compte-rendu de cinquième séance (16/02/2022) :

Cette séance n’a pas été très productive du fait que nous soyons un peu perdus dans la masse de code. En effet, les idées ne venaient plus aussi facilement qu’avant du fait que le code se complexifie. Nous avons donc essayé de ré-éplucher le code. Puis, toujours en appliquant le principe de mob programming, nous avons rencontré une difficulté pour un scénario bien particulier. En effet, lorsque l’on est esku et que le se couche en utilisant « Tira », le jeu ne lance pas de dialogue avec notre allié afin que celui-ci fasse un choix en conséquence. Nous sommes donc en train d’adapter le code de la classe Dialogue afin que la méthode « prochainJoueur » nous retourne notre allié et que celui-ci entre dans le dialogue.

## G. Compte-rendu de sixième séance (02/03/2022) :

Cette séance a été plus productive que la précédente, mais nous éprouvons toujours beaucoup de difficultés avec notre problème de dialogue. Nous avons aujourd’hui travaillé en pair programming avec Xan (groupe Jules et Xan). Mathis a travaillé de son côté mais nous avons tout de même communiqué. Toujours dans la résolution de notre problème de dialogue, nous avons produit un test qui met en œuvre le scénario que nous identifié précédemment. Nous nous sommes rendus compte, en débuggant le code de notre test, que : une fois que nous avons joué Tira, les autres joueurs ont des pointeurs null. Avec Mathis, nous essayons donc de remédier à ce problème en identifiant les méthodes étant à l’origine de l’affectation des pointeurs en fonction des joueurs. Nous étudions comment modifier la méthode « adversaireDe » de la classe Participant notamment en utilisant une Deque plutôt qu’une liste. Celle-ci nous permet de remplir deux files avec les joueurs devant parler au prochain tour, en fonction des actions effectuées.

## H. Compte-rendu de septième séance (09/03/2022) :

Durant cette séance, nous n’étions que deux (Xan et Jules) car Mathis était malade. Nous avons été plutôt productifs, malgré le fait que le jeu ne fonctionne toujours pas en deux contre deux.

Xan de son côté a tenté en vain de résoudre le problème de tournante qui ne fonctionne pas pour certains scénarios particuliers. Xan est resté bloqué longtemps sur ce problème, en essayant de nombreuses choses dans le code. Il a fini par trouver une partie de la solution en bouclant sur la file de joueurs actuelle. En appliquant une condition melant le rang du joueur actuel dans la file modulo 2, il a réussi à plus ou moins replacer ce joueur dans une file représentant l’ordre de parole pour le prochain tour.

De son côté, Jules a bien avancé le compte-rendu, celui-ci est quasiment terminé. Il a généré le diagramme de classe du projet via un plug-in d’IntelliJ, qu’il a intégré au compte-rendu. Il a également continué de rédiger et agencer les différentes parties du compte-rendu.

# 4. Conclusion :

En conclusion, nous dirons que nous avons beaucoup tiré parti de ce projet. En effet, celui-ci nous a formé à plusieurs aspects du refactoring mais aussi à certains aspects du travail et codage en équipe. Nous avons appris qu’il est préférable de d’abord se poser sur le fonctionnement du code avant de vouloir se précipiter à coder. De plus, il est souvent très intéressant de refaire quelques éléments de conception du projet comme le diagramme de classe, car ceux-ci nous offrent une vision autre du projet qui est utile pour trouver de nouvelles idées et avancer. En outre, l’organisation du travail et de l’équipe est primordiale, il aurait été intéressant dans notre cas de faire des « to do list » avec des tâches assignées à chaque membre de l’équipe.

Finalement, si nous avions eu plus de temps, il est évident que nous aurions fait marcher notre jeu en deux contre deux, notamment en réglant nos soucis de code au niveau de l’interrogation des joueurs dans certains scénarios particuliers. Par ailleurs, nous aurions également produit plus de test en fin de programmation d’une part, pour que notre projet soit durable et puisse être modifié en s’assurant que la base du jeu est bonne et d’autre part, pour nous assurer que l’ensemble des règles du Mus sont bien respectées par notre programme, notamment en programmant différents scénarios. Enfin, nous aurions pu peaufiner notre code en réduisant les codes smells et en les remplaçant par du code plus propre.