Institut national des sciences appliquées de Rouen

Rapport Projet C++ GM4

# Jeux de déduction


Auteurs :
Thibaut André-Gallis
thibaut.andregallis@insa-rouen.fr
Ghali Bennani
ghali.bennani@insa-rouen.fr
Malou Berthe
malou.berthe@insa-rouen.fr
Thomas Boeno
thomas.boeno@insa-rouen.fr
Redouane Bouazza
rbouazza@insa-rouen.fr

Auteurs :
Hiba El Mouhib
hiba.el_mouhib@insa-rouen.fr
Kévin Gatel
kevin.gatel@insa-rouen.fr
Bahaeddine Hilal
bahaeddine.hilal@insa-rouen.fr
Enseignants :
Jean-Philippe Kotowicz
jean-philippe.kotowicz@insa-rouen.fr

25 Mai 2022

# Table des matières

# Table des figures

# 1. Introduction

Un jeu de déduction est un jeu de société dans lequel un joueur doit découvrir un ou plusieurs éléments cachés par un adversaire, généralement en posant des questions selon des règles établies.

Il peut s'agir d'un jeu avec une grille ou un parcours ou d'un jeu de cartes. Parfois même, le jeu ne nécessite aucun matériel particulier.

Dans ce projet on se propose de programmer un Mastermind et un Wordle. Notre groupe étant assez grand, la conception et l'implémentation n'est pas un problème.

Le Mastermind ou Master Mind est un jeu de société pour deux joueurs dont le but est de trouver un code plusieurs coups. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz, dans les années 19701. Le jeu se base sur un jeu plus ancien : bulls and cows (taureaux et vaches) qui se jouait avec papier et crayon et des nombres au lieu de couleurs.

Wordle est un jeu de lettres en ligne gratuit développé en 2021 par Josh Wardle. Ce jeu est une adaptation directe du jeu télévisé américain Lingo (Motus en France) qui propose de faire deviner un mot par plusieurs tentatives, en indiquant pour chacune d'entre-elles la position des lettres bien placées et mal placées. L'originalité du jeu tient dans le fait qu'une seule énigme commune à tous les joueurs est proposée chaque journée.

Les deux jeux sont assez similaires niveaux fonctionnement, il est donc intéressant de créer une base de classe réunissant l'ensemble des points en commun puis de définir ou redéfinir certaines méthodes dans les classes spécifiques au jeu auquel nous voulons jouer.

Ce projet sera réalisé en C++, langage évalué ce semestre à la suite de l'apprentissage du langage C fait l'année dernière.

Dans une première partie nous verrons l'analyse des besoins résumant le projet en terme de fonctionnalité, manuel d'utilisation, contrainte et deadlines.

Dans une deuxième partie on s'intéressera au cahier de spécification avec la vison du projet, les fonctionnalités une nouvelle fois et l'ensemble des diagrammes UML composant la conception du projet.

Enfin on verra les spécificités techniques avec la documentation et les test unitaires.

# 2. Analyse des besoins

Notre projet consiste à implémenter un modèle permettant de gérer tous les jeux de type déduction. C'est-à-dire tous les jeux où un joueur choisit une combinaison d'éléments (que l'on appelle code) et un ou plusieurs autres joueurs ont pour but de trouver cette combinaison à partir d'essais et d'informations résultantes de ces essais. Les éléments peuvent prendre plusieurs formes comme par exemple des chiffres, des lettres, des couleurs ou encore des formes. Les jeux les plus connus dans ce type sont le Mastermind et le Wordle. Nous allons réaliser ce projet dans un but de divertissement de l'utilisateur potentiellement pour un site de jeu en ligne.

## 2.1 Ce que doit faire le système

- Enregistrer une combinaison donnée par un joueur ou fournir une combinaison aléatoire si il y a un unique joueur.
- Donner des informations aux joueurs suite à la saisie d'une combinaison (élément bien placé ou non, élément appartenant au code ou non etc...).
- Projet de trouver un code en un minimum de tentatives avec une intelligence artificielle

## 2.2 Les grandes fonctionnalités

- Choisir le mode de jeu :
  - 2 joueurs : l'un choisit un code et l'autre essaye de le déterminer.
  - 1 joueur : Le système choisit une combinaison aléatoire et le joueur doit la déterminer.
  - 1 joueur : Le joueur choisit une combinaison et assiste à la performance de l'IA qui essaye de trouver le code. L'idée est de voir chaque coup de l'IA et la réponse de la machine, en voyant clairement un compteur de coups ou le temps de résolution de l'IA.
- Entrer une combinaison de départ.
- Entrer une combinaison pour essayer de trouver celle de l'autre joueur (ou du système).
- Choisir les paramètres du jeu, comme par exemple :
  - Fixer le type d'élément.
  - Nombre d'éléments dans le code.
  - Nombre maximum de tentative pour trouver le code
  - Difficulté (qui correspond ici aux informations données à la suite d'une proposition de combinaison, qui peuvent être plus ou moins étoffées, celles-ci pouvant être le fait de donner les éléments qui sont bons, ou bien placés).

## 2.3 Manuel d'utilisation préliminaire :

Voici les étapes à suivre lors de l'utilisation de notre programme.
Pour commencer il faudra lancer le programme depuis le terminal, ensuite le premier choix à faire sera à quel jeu voulez vous jouer ou bien de commencer à jouer directement.
Si l'on décide de jouer directement sans prédéfinir le mode de jeu, ou les paramètres de jeu, ce sera un jeu 1 vs machine.
Une fois cela effectué, en fonction du mode choisi le programme vous demandera ou non d'entrer une combinaison de départ.
Le jeu démarrera ensuite vous aurez un certain nombre d'essai pour trouver le code, vous ne pouvez faire qu'une seule proposition à chaque fois et à chaque proposition vous obtiendrez le résultat donné par l'ordinateur concernant la justesse de votre proposition.
Le jeu s'arrête si vous trouvez la bonne combinaison ou si le nombre d'essai maximal a été atteint.
L'ordinateur vous demandera si vous souhaitez rejouer ou arrêter. Si vous rejouez vous reviendrez au menu de départ sinon le programme se fermera.

## 2.4 Contraintes :

La principale contrainte de notre projet sera qu'il doit tourner sur linux, et être codé en C++.

## 2.5 Calendrier

- 25 Mars 2022 : Rendu du cahier de spécification
- 8 Avril 2022 :Rendu des spécifications technique
- 25 Mai 2022 : Rendu du projet final

# 3. Cahier de spécifications

## 3.1 Vision du projet

Pour commencer, nous avons pour objectif de créer un programme permettant à un joueur de jouer à des jeux de déduction tel que par exemple Mastermind ou Wordle. Le programme offrira au joueur la possibilité de jouer du côté de celui qui devine la combinaison proposée par l'adversaire qui, dans ce cas, sera l'ordinateur. Il sera aussi possible d'inverser les rôles et le joueur pourra proposer une combinaison que l'ordinateur devra résoudre avec le moins de coup possible.
Afin d'avoir une large gamme de type de jeux nous avons prévu de proposer au joueur de jouer au Mastermind classique ou bien à une variante avec des lettres pour deviner des mots provenant d'un dictionnaire que nous fournirons. L'utilisateur pourra donc proposer une combinaison à l'ordinateur pour que celui-ci la devine ou alors pour deviner celle de l'ordinateur. Dans ce cas à chaque proposition du joueur, l'ordinateur va en retour donner les informations de correction correspondantes. C'est à dire, dans le cas du Mastermind, le nombre de couleurs bien placées et le nombre de bonnes couleurs mal placées. Ce qui va permettre au joueur d'en déduire une nouvelle proposition astucieuse pour arriver au final à la bonne composition.
Dans la version finale, un mode de jeu humain vs ordi sera implémenté, où l'ordi devra alors deviner la combinaison choisie au départ par le joueur humain. Ce sera en quelque sorte une IA, on essaiera alors de la rendre la plus performante possible.

## 3.2 Fonctionnalités

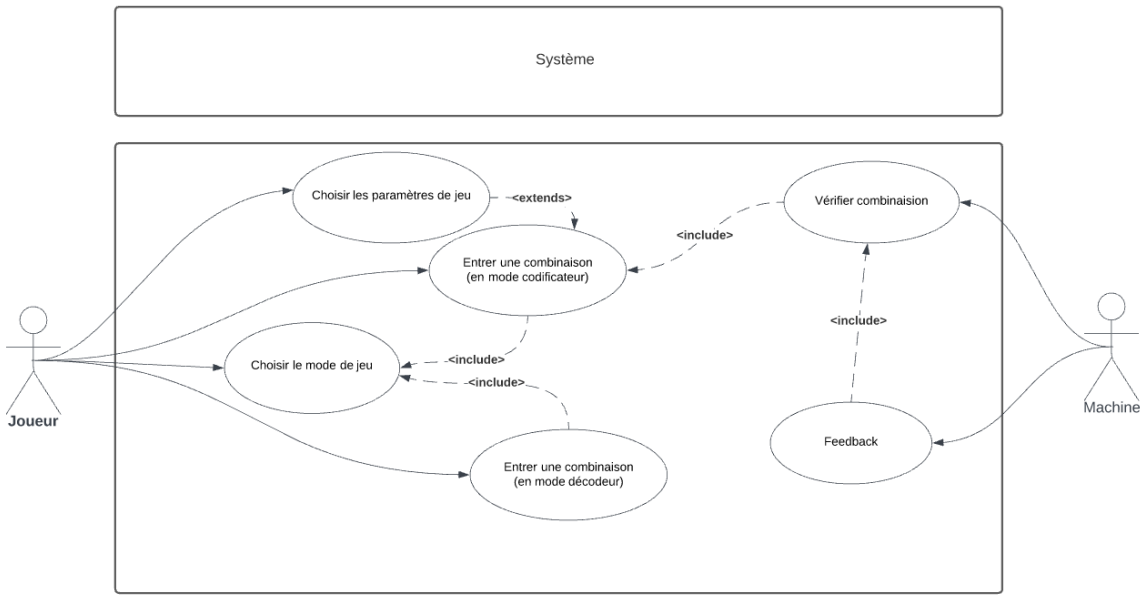


Figure 3.1 – Diagramme Use-case

On remarque que le joueur a assez peu d'action différente lors du jeu. En effet, au départ il choisit son mode et ses paramètres de jeu, puis il ne fait qu'entrer des combinaisons de couleur ou de lettre. De son coté, la machine vérifie la combinaison, dans un premier temps si elle est correcte syntaxiquement, puis renvoie un feedback sur la disposition des couleurs/lettres par rapport à la combinaison cherchée. Il est important de remarquer que le joueur dans notre jeu ne vérifiera jamais la combinaison, elle sera faite automatiquement par la machine. Dans la suite du projet, on parlera de machine ce qui est fait "automatiquement" par le système. L'IA ou l'ordi pourra être considéré comme un joueur dans notre cas d'utilisation.

## 3.3 Spécifications détaillées des fonctionnalités par cas d'utilisation

Les différents cas d'utilisations du côté de l'utilisateur sont :

- ✦ Le choix des paramètres de jeu (ceci étant le nombre de couleurs, le nombres de coups d'essais)
- ✦ Le choix du mode de jeu (1 joueur vs 1 joueur, un joueur vs une IA où le joueur devine la combinaison choisie par l'ordi, et ordi vs joueur où l'ordi devra deviner la combinaison choisie par le joueur au début de la partie)
- ✦ L'entrée d'une combinaison, si le joueur est codeur (dans le cas du mode de jeu 1v1 ; d'où la liaison "include" avec le cas "choisir le mode de jeu". Dans ce cas, le joueur propose une combinaison à faire deviner au second joueur.
- ✦ L'entrée d'une combinaison, si à l'issu du mode de jeu on a choisi 1 joueur vs 1 joueur ou 1 vs machine, dans le cas où le joueur est décodeur.

Du côté de la machine nous avons les cas d'utilisation suivants :

- ✦ Vérifier la combinaison : la machine va dans un premier temps faire une analyse syntaxique de la combinaison. En effet si l'utilisateur rentre une couleur ou une lettre qui n'est pas dans l'ensemble des éléments pouvant constituer une combinaison, alors la machine renverra un message d'erreur à l'utilisateur lui disant de rentrer une nouvelle combinaison valide. Puis, si la combinaison est correcte syntaxiquement, alors la machine va comparer la combinaison rentrée et la combinaison à deviner pour ensuite libérer un feedback.
- ✦ Feedback : La machine va renvoyer un feedback, un message qui comprend le nombre de couleurs/lettres bien placées et le nombre de couleurs/lettres appartenant à la combinaison souhaitée mais mal placées.

## 3.4 Diagrammes de séquences

On résumera l'ensemble des cas dans ce diagramme de séquence système :

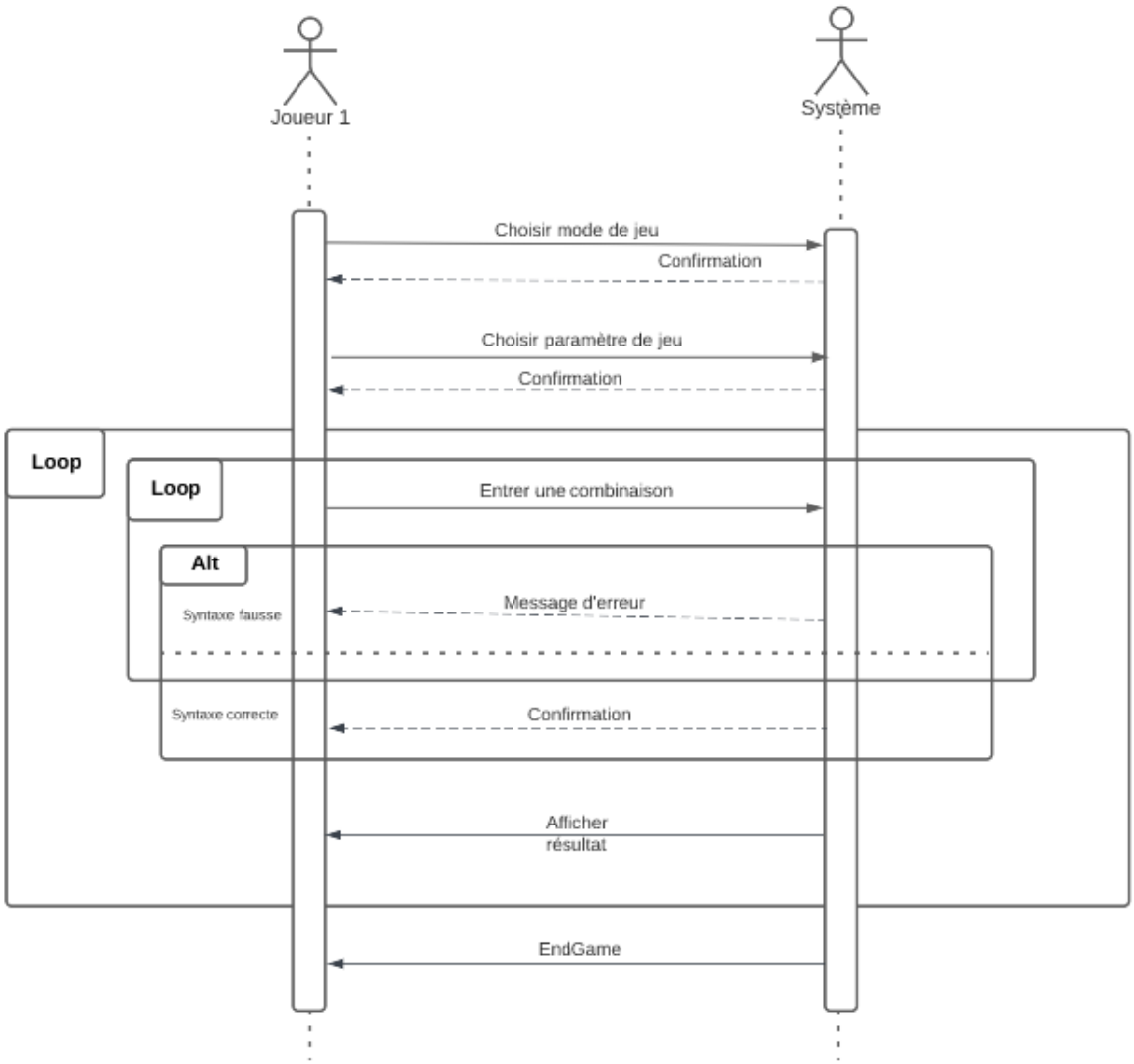


FIGURE 3.2 – Diagramme de séquence système général

Ce diagramme résume le déroulement d'une partie. Au départ le joueur peut choisir son mode de jeu, il le sélectionne et nous avons une confirmation. De même le joueur peut choisir ses paramètres, on aura une confirmation en retour également. Puis lorsque le jeu commence le joueur entre des combinaisons de couleur/lettre jusqu'à ce que le jeu se termine. Une combinaison entrée sera au préalable vérifiée syntaxiquement par la machine, si elle s'avère incorrecte un message d'erreur sera affiché et une nouvelle combinaison sera demandée au joueur. Dans le cas où la combinaison est bonne syntaxiquement, cette dernière est affichée au-dessus des précédentes. La boucle de jeu prend fin lorsque le joueur remporte la partie en devinant la combinaison recherchée ou bien si le nombre de tour max est dépassé.

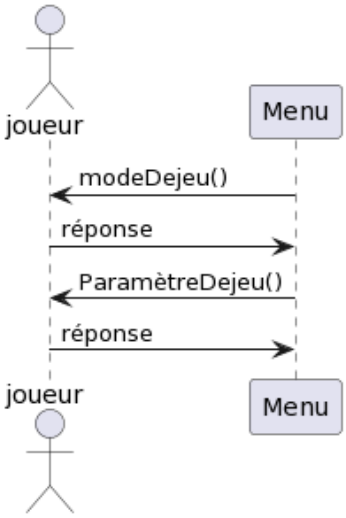


FIGURE 3.3 – Diagramme séquence : Menu

Le menu se lance, plusieurs options sont disponibles. Si le joueur veut directement jouer, il pourra, et s'il veut choisir un mode de jeu en particulier il pourra également. Le menu se mettra alors en attente de l'utilisateur pour qu'il choisisse le mode de jeu auquel il souhaite jouer. Le Joueur donne sa réponse et le menu la récupère. Ensuite le joueur pourra modifier les paramètres de jeu par défaut. Le menu demandera alors au joueur les paramètres et donc les règles de la prochaine partie. Une fois encore le joueur donne sa réponse et le joueur peut lancer une partie.

Afin de mieux comprendre le fonctionnement de notre programme nous avons fait le diagramme de séquence suivant :

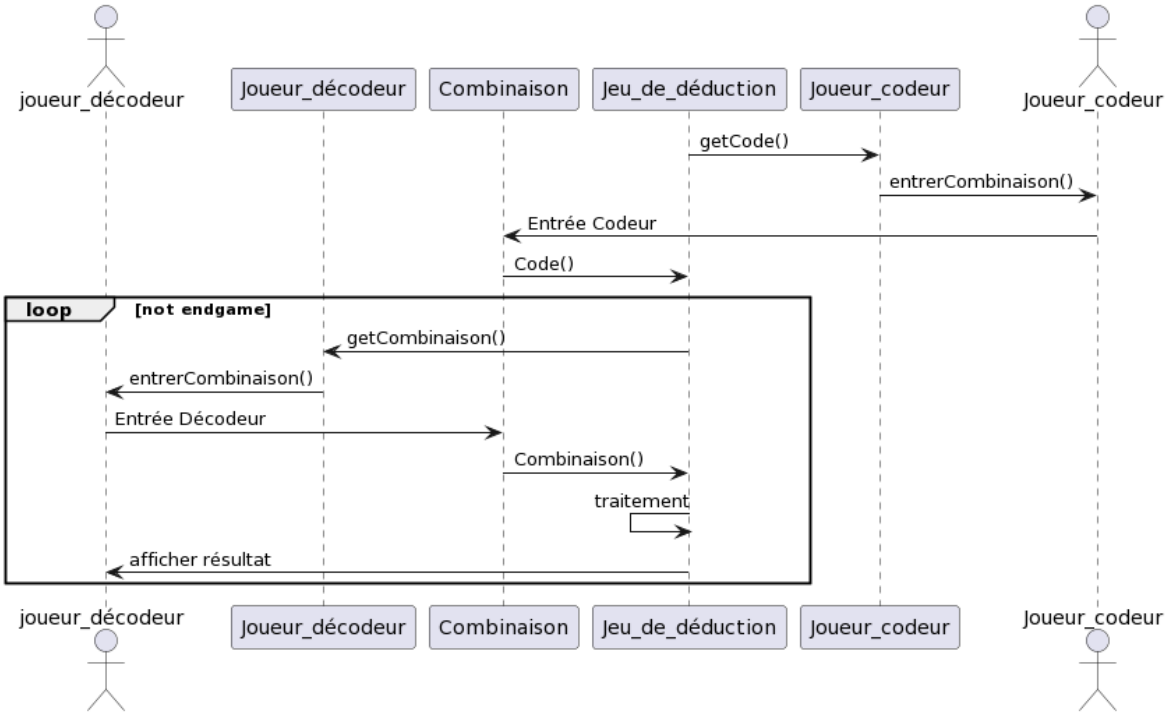


FIGURE 3.4 – Diagramme séquence : phase de jeu

Tout d'abord nous allons définir les entités du diagramme. Le "joueur décodeur" correspond au décodeur, c'est à dire l'utilisateur qui va chercher à deviner la composition secrète. Il pourra être un utilisateur humain ou une intelligence artificielle. De même le "joueur codeur" correspondra au codeur, l'individu qui va donner la composition secrète à deviner pour le décodeur. Il pourra également être un être un humain ou bien un ordinateur. "Jeu de déduction" correspond au système en lui-même, la machine et Combinaison indique la classe combinaison.
Au démarrage après avoir choisi le mode de jeu et les règles si le joueur le souhaite, dans le cas où un des joueurs est dans le rôle codeur, le système se met en attente du " joueur codeur" en lui demandant de rentrer la combinaison à deviner. Une fois que cela est fait le système la récupère en transformant l'entrée de l'utilisateur en un objet combinaison par le biais de la classe Combinaison. Dans le cas où un Joueur a pour mission de décoder, le programme lance ensuite une boucle dans laquelle le système demande au "joueur décodeur" une proposition. Le système la récupère, la traduit en objet combinaison puis par la suite la machine va traiter la proposition du "joueur décodeur", pour ensuite renvoyer sur l'écran le résultat du traitement correspondant au nombre de bonnes couleurs et au nombres de bonnes couleurs bien placées. Ensuite la boucle se termine et le système redemande au "joueur décodeur" une proposition. La boucle s'arrête si le nombre de tentative maximal a été atteint ou alors si pendant la phase de traitement le "joueur décodeur" a trouvé la bonne combinaison. Dans ce cas le système informe que le "joueur décodeur" a gagné, le "joueur codeur" dans l'autre cas.

## 3.5 Description du domaine

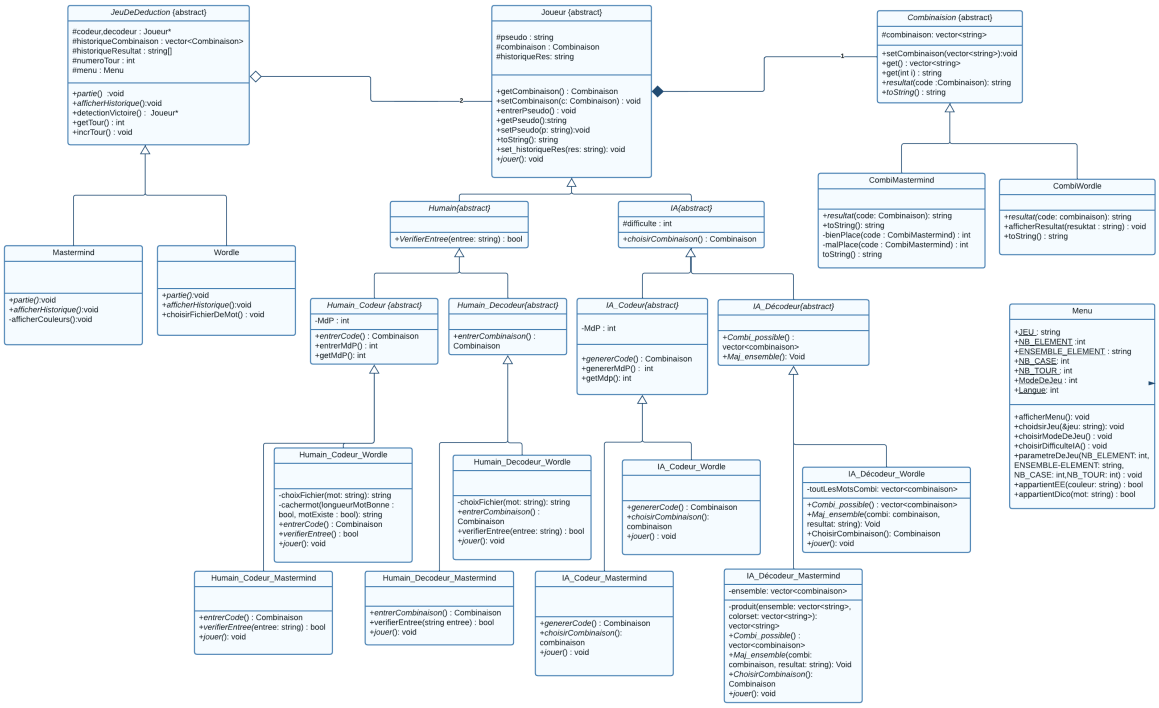


FIGURE 3.5 – Diagramme de classe

Nous avons réalisé 23 classes différentes (dont une qui sera décrite plus loin). Faisons une description de chacune des classes. Pour y voir plus clair, séparons ce diagramme en plusieurs familles de classe.

On commence par décrire les classes liées à *JeuDeDeduction* :

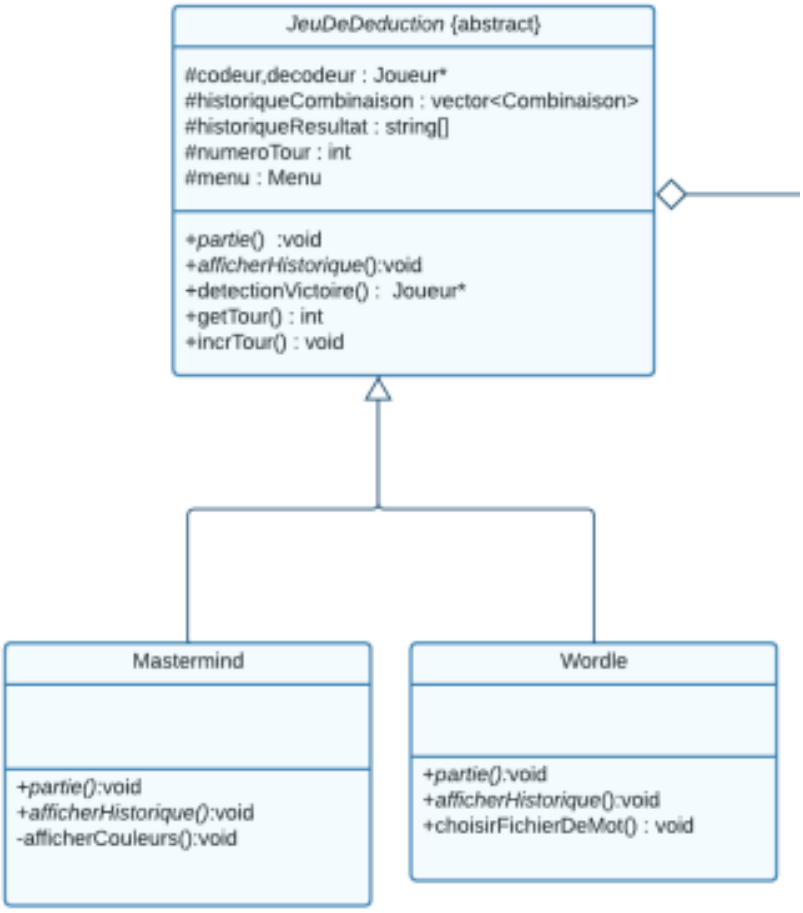


FIGURE 3.6 – Zoom n°1 du diagramme de classe

**Jeu de déduction**

Cette classe abstraite est la classe principale du projet. Elle comprend les méthodes permettant de lancer et de jouer une partie. Nous avons donc les méthodes suivantes :

- *«abstract» partie() : void* : Il s'agit de la méthode principale du main() décrivant le déroulé d'une partie selon le mode de jeu et ses paramètres. Celle-ci sera redéfinie de manière différente selon le jeu choisi, Mastermind ou Wordle.

- *«abstract» afficherHistorique() : void* : Cette méthode abstraite permet d'afficher le jeu entier comprenant les historiques des combinaisons rentrées avec le résultat de comparaison avec le code associé (nombre de couleur bien placé et mal placé). Cette méthode ne permet pas d'afficher le code car nous sommes du point de vue décodeur. Elle sera redéfinie de manière différente selon le jeu choisi.

- *detectionVictoire() : Joueur* : Cette méthode renverra null si aucun des joueurs ne gagne, sinon elle renvoie le joueur gagnant, codeur ou décodeur.

- *getTour() : int* : Cette méthode permet de renvoyer le nombre de tours.
- *incrTour() : void* : Cette méthode va incrémenté le nombre de tours.

**Mastermind**

Classe héritant de *JeuDeDeduction*,on y retrouve les méthodes suivantes :

- *partie() : void* :Il s'agit de la méthode permettant de lancer une partie.
- *afficherHistorique() : void* : Cette méthode permet d'afficher le jeu entier comprenant les historiques des combinaisons rentrées avec le résultat de comparaison avec le code associé (nombre de couleur bien placé et mal placé). Cette méthode ne permet pas d'afficher le code car nous sommes du point de vue décodeur, cette méthode ne s'applique qu'au mastermind.
- *afficherCouleurs() : void* :Il s'agit de la méthode qui affiche la liste des couleurs que le joueur peut choisir.

**Wordle**

Classe héritant de *JeuDeDeduction*, on y retrouve les méthodes suivantes :

- *partie() : void* :Il s'agit de la méthode principale permettant de lancer une partie wordle.
- *afficherHistorique() : void* : Cette méthode permet d'afficher le jeu entier comprenant les historiques des combinaisons rentrées avec le résultat de comparaison avec le code associé (nombre de couleur bien placé et mal placé). Cette méthode ne permet pas d'afficher le code car nous sommes du point de vue décodeur, de plus elle ne s'applique qu'au jeu Wordle.

La deuxième famille de classe est la suivante :

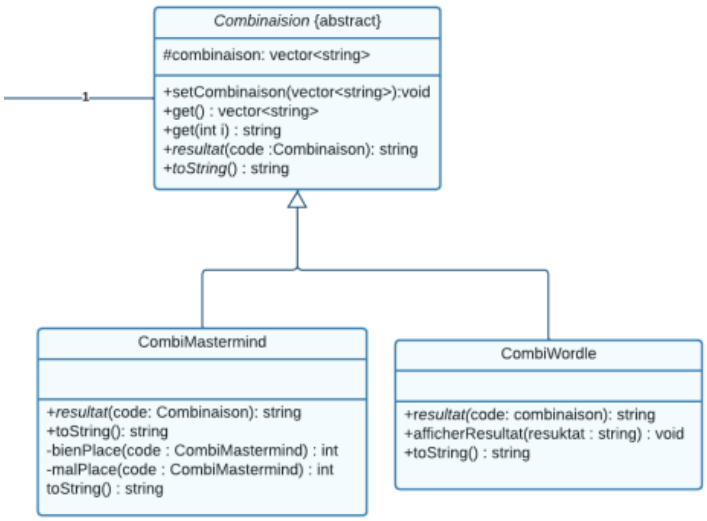


FIGURE 3.7 – Zoom n°2 du diagramme de classe

**Combinaison**

Cette classe est une classe abstraite qui permet de définir ce qu'est l'objet Combinaison. En effet l'utilisateur va devoir entrer une suite de mot (couleur ou lettre) séparée d'espace pour ensuite être transformée en objet Combinaison. Elle permettra ensuite pendant le jeu d'évaluer les propositions de combinaisons faites par le joueur. Cette classe a pour attribut un tableau de *string* correspondant à notre combinaison. Chaque *string* étant un élément de l'ensemble des éléments définis au début de la partie. Elle a pour méthodes :

- *get() : string[]* : Cette méthode permet de renvoyer le tableau de *string* contenu dans l'attribut *combinaison* de la classe.
- *get(int i) : string* : Elle permet de renvoyer le $i^{me}$ élément du tableau de string *combinaison* en attribut.
- *«abstract» toString() : string* : Cette méthode abstraite permet de transformer l'objet *Combinaison* en une chaîne de caractère.
- *«abstract»resultat(Combinaison code) : string* : Cette méthode sera redéfinie par ses classes filles, elles permettra de retourner le résultat de la différence entre la combinaison et le code entré en paramètre.

**CombiMastermind**

Classe héritant de *Combinaison*, elle permet de décrire les combinaisons d'un Mastermind.

- *bienPlace(code : Combinaison) : int* : C'est une méthode qui, à partir du mot cherché, le code, pourra retourner la liste des indices des lettres bien placées.
- *malPlace(Combinaison) : int* : C'est une méthode qui, à partir du mot cherché, le code, pourra retourner la liste des indices des lettres appartenant au mot cherché mais étant mis à la mauvaise place.
- *toString() : string* : Cette méthode permet le passage d'une combinaison de couleur à une chaîne de caractères.
- *resultat(Combinaison) : string* : Cette méthode permet de ressortir quels sont les couleurs bien ou mal placées.

**CombiWordle**

Classe héritant de *Combinaison*, elle permet de décrire les combinaisons du Wordle.

- *toString() : string* :Cette méthode permet le passage d'une combinaison de lettre à un mot.
- *resultat(Combinaison) : string* : Cette méthode permet de ressortir quels sont les lettres bien ou mal placées en fonction d'un code couleur.

La troisième famille de classe, la plus grande est celle qui regroupent l'ensemble des classes *Joueur* :

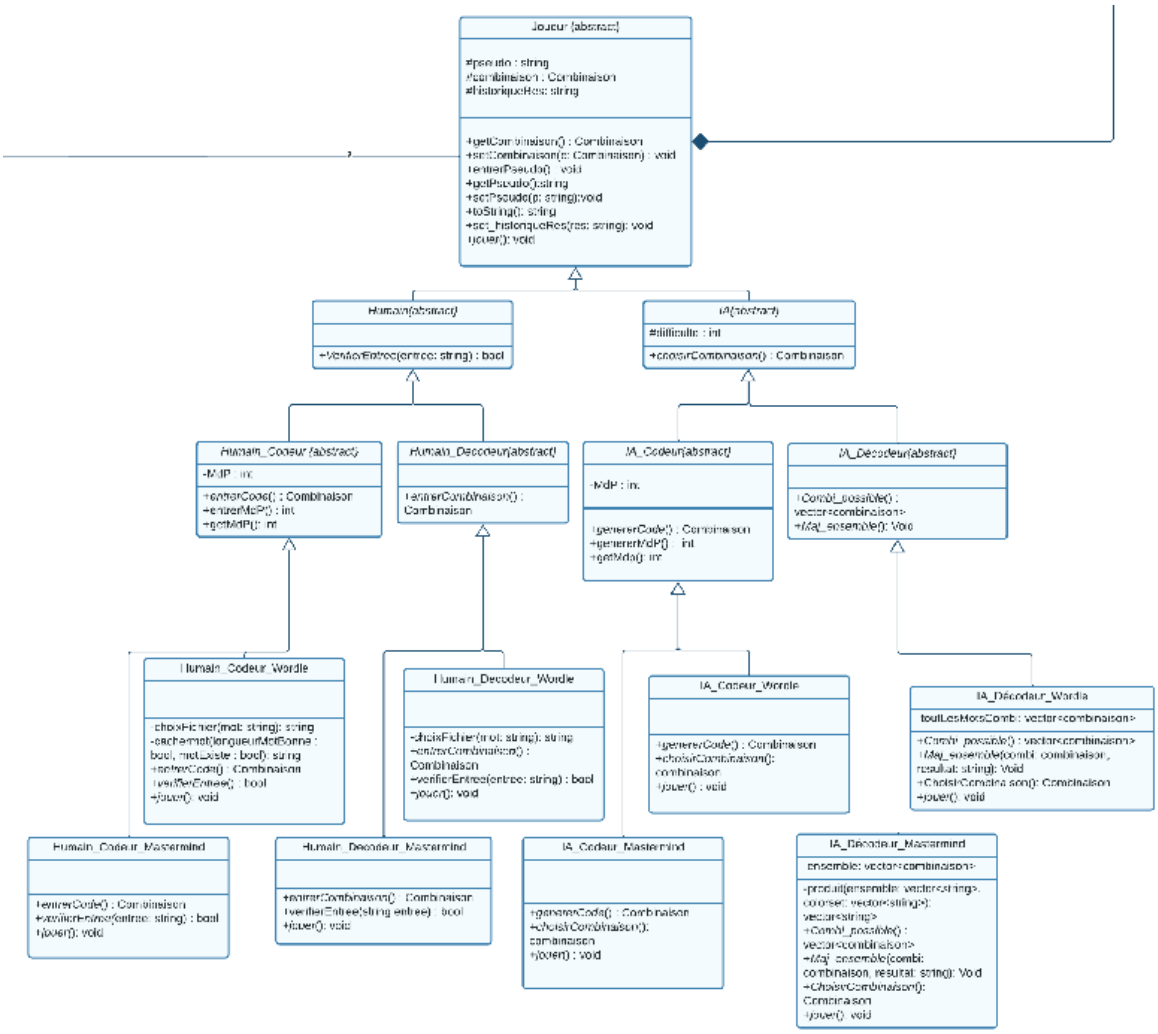


FIGURE 3.8 – Zoom n°3 du diagramme de classe

**Joueur**

Il s'agit d'une classe abstraite qui définit tout joueur, humain ou IA, du jeu de déduction. On y retrouve deux attributs, le premier est un élément de la classe *Combinaison* et le second est le pseudo que le joueur choisira au début de la partie.

On y retrouve les méthodes suivantes :

- *getCombinaison() : Combinaison* : Elle renvoie la combinaison stockée (en attribut) du joueur.
- *setCombinaison(Combinaison c) : Combinaison* : Elle affecte une combinaison à l'attribut *combinaison* du joueur
- *entrerPseudo() : void* : Cette méthode gère la saisie du pseudo par le joueur.
- *getPseudo() : string* : Cette méthode renvoie le pseudo stocké (en attribut) du joueur.

- *setPseudo(string p) : void* : Cette méthode affecte le pseudo passé en paramètre à l'attribut *pseudo* du joueur.

- $set_historiqueRes(string res) : void : Mthode permettant d'avoir en argument l'historique de ses rsultats.toString() : string : Permet de transformer l'objet Joueur en chaine de caractre, on renverra ici le pseudo du joueur.$

**Humain**

Classe qui hérite de *Joueur*, elle décrit les joueurs humains.

- *«abstract» VerifierEntree(string entree) : bool* : Cette méthode abstraite permettra de vérifier que la combinaison rentrée par le joueur décodeur est syntaxiquement correct selon le jeu de déduction choisi : Mastermind ou Wordle.

**IA**

Classe qui hérite de *Joueur*, elle décrit les joueurs non humains.

- *«abstract» choisirCombinaison() :combinaison* : Cette méthode permettra à l'IA de choisir une combinaison de manière intelligente selon le mode de jeu de déduction choisi : Mastermind ou Wordle.

**Humain_Codeur**

Cette classe abstraite permet de décrire les joueurs humains en mode codeur que ce soit pour le jeu Mastermind ou bien pour le jeu Wordle. Elle hérite de la classe *Humain*

- *«abstract» entrerCode() : void* : C'est une méthode abstraite qui devra être redéfini de manière différente selon le jeu Mastermind ou Wordle. Ces méthodes permettront au joueur codeur d'entrer la combinaison ou le mot que le joueur décodeur devra ensuite trouver.

**Humain_Decodeur**

Cette classe abstraite permet de décrire les joueurs humains en mode décodeur que ce soit pour le jeu Mastermind ou bien pour le jeu Wordle. Elle hérite de la classe *Humain*

- *«abstract» entrerCombinaison() : Combinaison* : Cette méthode permet de récupérer la proposition du joueur humain en mode décodeur. Elle va demander à l'utilisateur d'entrer une chaîne de caractère correspondant à la combinaison qu'il veut proposer. La méthode va ensuite récupérer la chaîne de caractère et la transformer en un objet de type combinaison en sortie.

**IA_Codeur**

- *«abstract» genererCode() :* C'est une méthode abstraite qui devra être redéfinie de manière différente selon le jeu Mastermind ou Wordle. Ces méthodes permettront à la machine de générer un code (combinaison ou mot) aléatoire selon 3 difficultés différentes que l'IA devra ensuite décoder automatiquement.

- *genererMdp() :* Cette méthode permet de créer un mot de passe permettant à un utilisateur de vérifier le code secret généré par la machine.

- *getMdp() :* Méthode retournant l'attribut MdP de l'IA.

**IA_Decodeur**

On propose d'utiliser l'algorithme de Donald Knuth pour résoudre le jeu de déduction.

- *Combi_possible() :* Cette méthode permet de créer l'ensemble $S$ de toutes les combinaisons possibles dans le cas du Mastermind ou prendre l'ensemble des mots présents sur le dictionnaire pour le Wordle.

- *Maj_ensemble() :* Si la proposition est incorrecte, cette méthode permet retirer de $S$ tout code qui ne donnerait pas le même résultat si le code était le mot à deviner.

**Humain_Codeur_Mastermind**

Classe héritant de *Humain_Codeur*, elle décrit les joueurs humains codeur dans le jeu Mastermind.

- *entrerCode() : Combinaison* : Cette méthode permet au joueur codeur d'entrer la combinaison que le joueur décodeur devra ensuite trouver. Cette méthode est seulement fonctionnelle pour le mode de jeu Mastermind.

- *VerifierEntree(string entree) : bool* : Permet de vérifier si l'entrée du joueur codeur peut être interpréter en une combinaison dans le cadre du jeu Mastermind. Cela implique une vérification du nombre d'élément entré et de leur existence (couleur reconnue par le programme). Ainsi, la méthode retourne vrai si l'entrée peut être interprétée comme une combinaison et faux sinon.

- *jouer() : void* : Permet au joueur de jouer un tour.

**Humain_Codeur_Wordle**

Classe héritant de *Humain_Codeur*, elle décrit les joueurs humains codeur dans le jeu Wordle.

- *entrerCode() : Combinaison* : Cette méthode permet au joueur codeur d'entrer le mot que le joueur décodeur devra ensuite trouver. Cette méthode est seulement fonctionnelle pour le mode de jeu Worlde.

- *verifierEntree() : bool* : Permet de vérifier si l'entrée du joueur codeur peut être interpréter en un mot dans le cadre du jeu Worlde. Cela implique une vérification du nombre d'élément entré et de leur existence (caractère de l'alphabet). Il faut de plus vérifier que le mot obtenu est bien dans le dictionnaire lié au jeu. Ainsi, la méthode retourne vrai si l'entrée peut être interprétée comme un mot du dictionnaire et faux sinon.

- *jouer() : void* : Permet au joueur de jouer un tour.

- *choixFichier(mot : string) : string* : Méthode privée permettant de choisir le bon chemin pour aller chercher dans le fichier correspondant au mot voulu.

- *cachermot(longueurMotBonne : bool, motExiste : bool) : string* : Méthode privée permettant de cacher le code lorsque le codeur veut le rentrer au début de la partie.

**Humain_Decodeur_Mastermind**

Classe héritant de *Humain_Decodeur*, elle décrit les joueurs humains codeur dans le jeu Mastermind.

- ✦ *entrerCombinaison() : void* : Méthode qui permet la saisie au clavier d'une chaîne de caractère par le joueur décodeur, cette chaîne sera ensuite traduite en une Combinaison si cela est possible.

- ✦ *verifierEntree() : bool* : Permet de vérifier si l'entrée du joueur décodeur peut être interpréter en une combinaison dans le cadre du jeu Mastermind. Cela implique une vérification du nombre d'élément entré et de leur existence (couleur reconnu par le programme). Ainsi, la méthode retourne vrai si l'entrée peut être interprétée comme une combinaison et faux sinon.

- ✦ *jouer() : void* : Permet au joueur de jouer un tour.

**Humain_Decodeur_Wordle**

Classe héritant de *Humain_Decodeur*, elle décrit les joueurs humains codeur dans le jeu Wordle.

- ✦ *entrerCombinaison() : Combinaison* : Méthode qui permet la saisie au clavier d'une chaîne de caractère par le joueur décodeur, cette chaîne sera ensuite traduite en un mot si c'est possible.

- ✦ *verifierEntree() : bool* : Permet de vérifier si l'entrée du joueur décodeur peut être interpréter en un mot dans le cadre du jeu Worlde. Cela implique une vérification du nombre d'élément entré et de leur existence (caractère de l'alphabet). Il faut de plus vérifier que le mot obtenu est bien dans le dictionnaire lié au jeu. Ainsi, la méthode retourne vrai si l'entrée peut être interprétée comme un mot du dictionnaire et faux sinon.

- ✦ *jouer() : void* : Permet au joueur de jouer un tour.

- ✦ *choixFichier(mot : string) : string* : Méthode privée permettant de choisir le bon chemin pour aller chercher dans le fichier correspondant au mot voulu.

**IA_Codeur_Mastermind**

Classe héritant de *IA_Codeur*, elle décrit les joueurs non humains codeur dans le jeu Mastermind.

- ✦ *genererCode() : Combinaison* : Cette méthode permet à la machine de générer une combinaison secrète que le joueur décodeur devra ensuite résoudre (qu'il soit humain ou IA). Cette méthode est seulement fonctionnelle pour le mode de jeu Mastermind.

- ✦ *jouer() : void* : Permet à l'IA de jouer un tour.

**IA_Codeur_Wordle**

Classe héritant de *IA_Codeur*, elle décrit les joueurs non humains codeur dans le jeu Wordle.

- ✦ *genererCode() : Combinaison* : Cette méthode permet à la machine de générer une combinaison secrète que le joueur décodeur devra ensuite résoudre (qu'il soit humain ou IA). Cette méthode est seulement fonctionnelle pour le mode de jeu Wordle.

- ✦ *jouer() : void* : Permet à l'IA de jouer un tour.

**IA_Decodeur_Mastermind**

Classe héritant de *IA_Decodeur*, elle décrit les joueurs non humains décodeur dans le jeu Mastermind.

- *Combipossible() : Combinaison[]* : Cette méthode crée l'ensemble de départ qui sera composé de toutes les possibilités c'est à dire toutes les suites de couleurs admissibles comme code.
- *jouer() : void* : Permet à l'IA de jouer un tour.
- *choisirCombinaison() : combinaison* : Méthode retournant la meilleur des combinaisons possibles à jouer.

**IA_Decodeur_Wordle**

Classe héritant de *IA_Decodeur*, elle décrit les joueurs non humains décodeur dans le jeu Wordle.

- *Combipossible() : Combinaison[]* : Cette méthode charge l'ensemble des mots du dictionnaire qui représente l'ensemble des mots possibles pour une langue et un nombre de lettre donné.
- *jouer() : void* : Permet à l'IA de jouer un tour.
- *choisirCombinaison() : combinaison* : Méthode retournant la meilleur des combinaisons possibles à jouer.

Enfin, la dernière famille de classe est celle contenant juste la classe *Menu* :

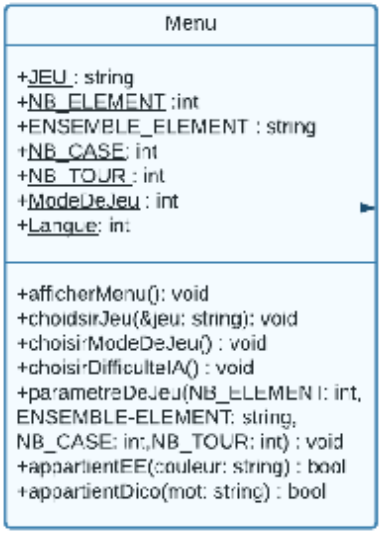


FIGURE 3.9 – Zoom n°4 du diagramme de classe

**Menu**

C'est la classe contenant tous les paramètres du jeu : le nombre d'élément, le nombre de case, le mode de jeu, la langue et le nombre de tour. Elle contient de plus un chemin d'accès vers le fichier contenant les mots du dictionnaire pour une langue et un nombre de lettre donné.

- *afficherMenu()* : Méthode qui affiche un menu explicite et simple qui propose à l'utilisateur de visionner les règles du jeu, de jouer (en réel ou par intelligence artificielle) ou de quitter.

- *choisrJeu() :void* : Méthode qui pemet à l'utilisateur de choisir entre 2 différents mode de jeu : Mastermind ou Wordle.
- *choisirModeDeJeu()* : Méthode permettant de choisir le mode de jeu qu'il souhaite. Humain vs Humain, Humain(décodeur) vs IA(codeur), Humain(codeur) vs IA(décodeur) ou IA vs IA.
- *parametreDeJeu()* : Méthode qui permet à l'utilisateur de choisir les paramètres de jeu avec lesquels il souhaite jouer (on peut choisir les paramètres classiques du jeu original ou modifier le nombre d'essais possibles et le nombre de cases à remplir).
- *appartientEE(string couleur)* : Vérifie si une couleur aux couleurs disponibles proposées par le mode de jeu Mastermind.
- *appartientDico(string mot)* : Vérifie si un mot est inclus dans le dictionnaire des mots reconnus par le mode de jeu Wordle.

---

Cela n'a pas été mentionné dans le diagramme de classe ou précédemment, mais nous utiliserons une classe appelée *FonctionsUtiles* en tant que bibliothèque regroupant l'ensemble des méthodes utiles que nous avons besoin dans le code. Nous n'avons pas trouvé utile de la mentionner dans le diagramme de classe mais il est important de mentionner ici son existence.

## 3.6 Diagramme de navigation

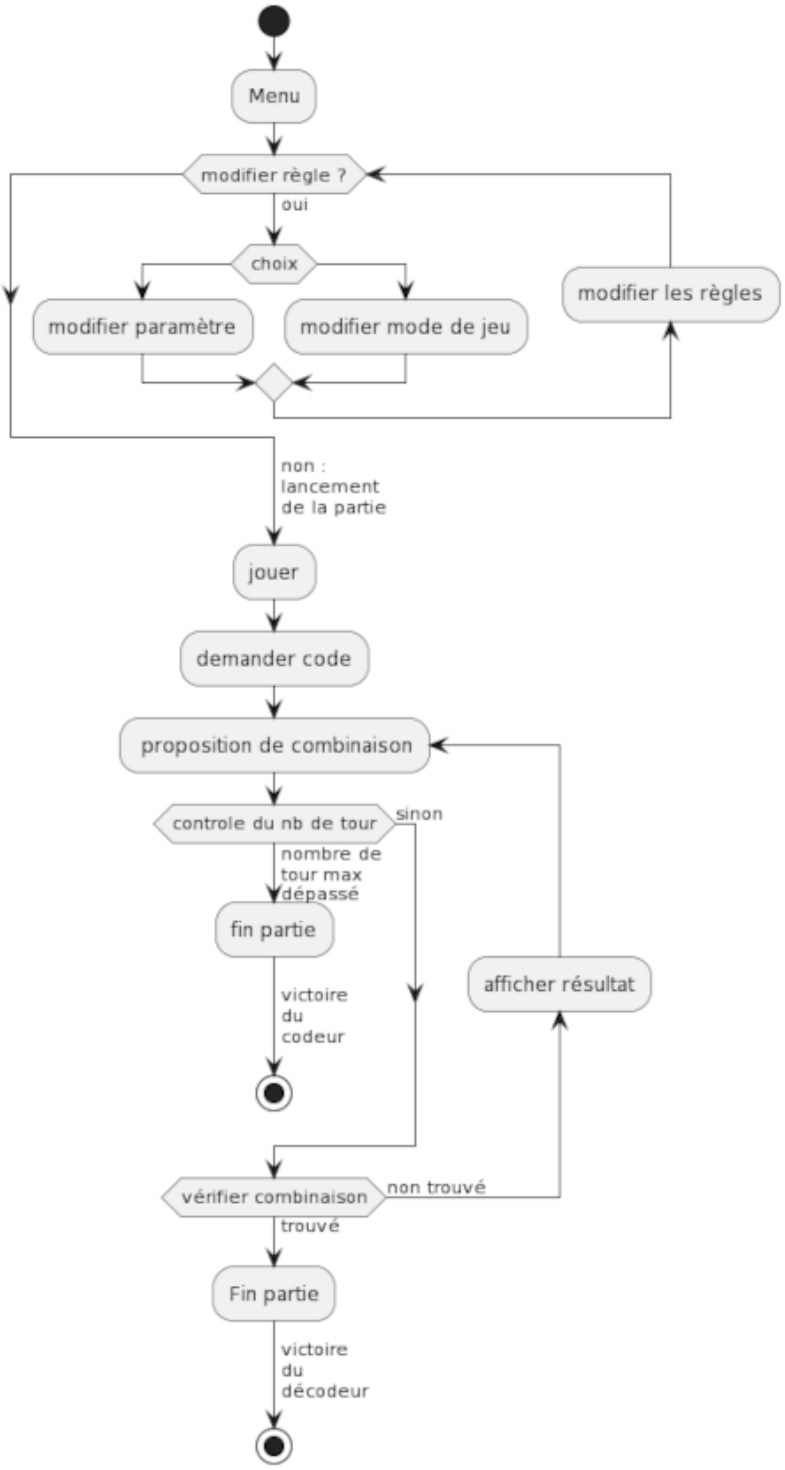


FIGURE 3.10 – Diagramme de navigation

Le diagramme ci-dessus décrit le cheminement d'une partie, en partant du choix des paramètres de cette partie jusqu'à la victoire de l'un des deux joueurs. La partie progresse en deux phases : dans la première, l'utilisateur choisit les règles puis on entre dans la deuxième phase dès qu'il décide de lancer la partie. Une première combinaison est alors demandée, il s'agit du code (entré par le codeur) que le joueur de décodeur devra trouver. Puis une succession de combinaison sera ensuite demandée au décodeur jusqu'à la victoire de l'un des deux joueurs.

## 3.7 Interactions entre les composants (associations)

La classe Mastermind doit avoir accès à la combinaison , transmise par la classe Combinaison, entrée par le Joueur décodeur (qu'il soit codeur ou bien décodeur) à chaque tour, et doit aussi toujours avoir accès à la combinaison initialement entrée par le joueur codeur. D'autre part cette même classe Mastermind aura besoin du nombre de tour "numeroTour" du joueur décodeur afin de détecter une possible victoire du joueur codeur.
Tous les éléments de la classe Menu sont eux aussi nécessaires au déroulement du jeu tout au long d'une partie. Ils doivent donc être accessibles à tout instant par chacune des classes et ne doivent pas être modifiés au cours d'une partie.

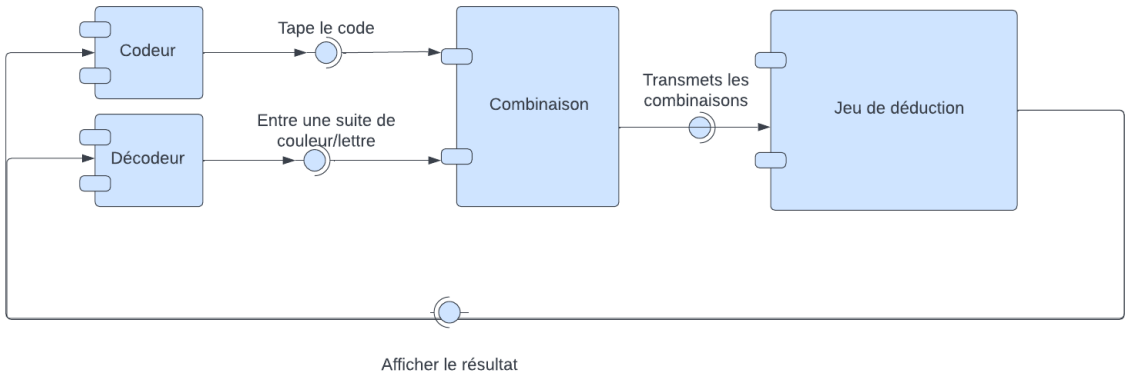


FIGURE 3.11 – Diagramme de composants

## 3.8 Tests d'intégration

Cette section vise à regrouper toutes les conditions nécessaires à vérifier de manière à ce que le code fonctionne correctement. Ainsi on mettra en gras les cas puis en dessous le résultat attendu.

**Cas : Choisir le mode de jeu**

Résultat : Proposer au joueur un nombre fini de modes différents.

**Cas : Choisir les paramètres**

Résultat : Proposer au joueur Le système doit proposer au joueur s'il souhaite modifier des paramètres du jeu dans la classe Menu et les modifier le cas échéant.

**Cas : Entrer une combinaison**

Résultat : Le joueur va pouvoir saisir une suite d'élément correspondant à une combinaison. Le système devra renvoyer un message d'erreur si la suite d'élément n'est pas bonne syntaxiquement ou si elle ne correspond pas au format d'une combinaison. Sinon la combinaison est affichée à la suite des autres avec le résultat correspondant au feedback de la machine (nombre de couleurs/lettres bien et mal placées).

**Cas : Le joueur décodeur a trouvé la combinaison du codeur**

Résultat : Le jeu s'arrête puis le système affiche l'historique des coups joués et précise la victoire du joueur décodeur.

**Cas : Le joueur n'a pas trouvé la bonne combinaison**

Résultat : Le numéro de tour est incrémenté de 1, la partie continue si le numéro de tour est inférieur au nombre de tour défini dans les paramètres, sinon elle s'arrête et le codeur gagne la partie.

**Cas : Le nombre de tour maximum pour deviner est dépassé**

Résultat : Le jeu s'arrête, le système affiche l'historique des coups joués et précise la victoire du joueur codeur.

# 4. Spécifications techniques

## 4.1 Rédaction et documentation des interfaces

La documentation été générée par l'outil doxygen. L'ensemble des documents générés se trouvent dans le répertoire "doc". Il comprend le dossier html et le pdf.

## 4.2 Tests unitaires

Dans cette partie nous allons évaluer et tester tous les cas possibles des méthodes et décrire leur fonctionnement suivant chaque cas. Les méthodes abstraites, n'étant jamais appelées, ne feront pas l'objet de test unitaire.

**Jeu de déduction**

- *detectionVictoire() : Joueur* : On vérifie dans le cas où le nombre de tours permis est dépassé, sans que le joueur codeur ait rentré la combinaison correcte , que la méthode renvoie le joueur codeur comme gagnant . On vérifie dans le cas où le joueur décodeur rentre la combinaison correcte avant le nombre de tours permis que la méthode renvoie le joueur décodeur comme gagnant. Sinon on vérifie que la méthode renvoie null.

- *getTour() : int* : On vérifie que cette méthode retourne l'entier contenu dans l'attribut numeroTour de la classe.

- *incrTour() : void* : On vérifie que cette méthode incrémente correctement de 1 le nombre de tour.

**Mastermind**

- *partie() : void* : On vérifie qu'une partie de Mastermind se lance lorsque le joueur choisit l'option Jouer avec le mode de jeu Mastermind dans le menu.

- *afficherPartie() : void* : On vérifie que cette méthode affiche correctement toutes les combinaisons rentrées depuis le début du jeu, avec le résultat de la comparaison avec la combinaison de couleurs associée.

- *afficherCode() : void* : Dans le cas où le mot de passe rentré est identique à l'attribut *MdP* du joueur *Codeur*, on vérifie que le code affiché correspond bien à la combinaison de couleur cherchée. Sinon on vérifie qu'on reste bien dans une boucle jusqu'à ce que le joueur codeur rentre correctement le code ou choisit d'abandonner l'action.

**Wordle**

- *partie() : void* : On vérifie qu'une partie de Wordle se lance lorsque le joueur choisit l'option Jouer avec le mode de jeu Wordle dans le menu.

- *afficherPartie() : void* : On vérifie que cette méthode affiche correctement toutes les combinaisons rentrées depuis le début du jeu, avec le résultat de la comparaison avec le mot recherché.

- *afficherCode() : void* : Dans le cas où le mot de passe rentré est identique à l'attribut *MdP* du joueur *Codeur*, on vérifie que le code affiché correspond bien au mot cherché. Sinon on vérifie qu'on reste bien dans une boucle jusqu'à ce que le joueur codeur rentre correctement le code ou choisit d'abandonner l'action.

- *choisirFichierDeMot() : void* : Cette méthode va choisir le fichier comprenant l'ensemble des mots, ces mots seront ensuite stockés dans l'attribut dictionnaire du menu.

**Classe Combinaison**

- *get() : int[]* : On vérifie que la méthode renvoie l'attribut *combinaison* de la classe.

- *get(int i) : int* : On vérifie que la méthode renvoie le $i^{me}$ élément de l'attribut *combinaison* de la classe.

- *toString() : String* : On vérifie que l'on renvoie bien la chaîne de caractère décrivant notre combinaison lors de l'affichage.

**Classe CombiMastermind**

- *bienPlace(code : Combinaison) : int* : On vérifie que l'entier renvoyé par la méthode correspond au nombre de couleurs bien placées dans le code.

- *malPlace(code : Combinaison) : int* : On vérifie que l'entier renvoyé par la méthode correspond au nombre de couleurs appartenant au code mais étant placées incorrectement.

**Classe CombiWordle**

- *bienPlace(code : Combinaison) : int[]* : On vérifie que le tableau renvoyé correspond aux indices des lettres bien placées.

- *malPlace(code : Combinaison) : int[]* : On vérifie que le tableau renvoyé correspond aux indices des lettres appartenant au mot recherché mais étant mal placées.

**Joueur**

- *getCombinaison() : Combinaison* : Nous devons nous assurer que la combinaison retournée par cette méthode est bien celle que nous souhaitons. C'est à dire que pour le joueur codeur, la *combinaison* renvoyée doit être celle entrée en début de partie. Pour le joueur décodeur, la combinaison en sortie sera la dernière qu'il ait proposé. Dans les deux cas, cette combinaison est la valeur de l'attribut *combinaison* du joueur.

- *setCombinaison(Combinaison c) : void* : On vérifie que l'attribut *combinaison* ait bien changé.

- *getPseudo() :string* : On vérifie que le string retourné est bien identique à l'attribut *Pseudo* du joueur.

- *setPseudo(string p) :void* : On vérifie que l'attribut *pseudo* ait bien changé.

- *entrerPseudo() : void* : On vérifie que le string rentré par le joueur est bien stocké dans l'attribut *Pseudo* du joueur .

**Humain Codeur**

- *entrerMdP() : int* : Nous devons vérifier que le programme demande bien une saisie clavier à l'utilisateur et que l'attribut *MdP* de la Classe est lui aussi bien mis à jour. Nous devons aussi lever une erreur dans le cas limite où le l'utilisateur saisie un mot de passe qui n'est pas conforme : nous acceptons seulement un mot de passe sous la forme d'un entier.

- *getMdP() : int* : Il suffit de vérifier que cette méthode retourne l'entier contenu dans l'attribut *MdP* de la classe.

**IA Codeur**

- *genererMdP() : int* : Il faut vérifier que l'attribut de la classe est bien mis à jour. Il faut également s'assurer de notifier erreur si le mot de passe généré n'est pas conforme aux normes établis (normalement impossible car le mot de passe est généré automatiquement). Nous allons accepter seulement un mot de passe de catégorie int.

- *getMdp() : int* : Il faut simplement s'assurer que cette méthode retourne l'entier contenu dans l'attribut Mdp de la classe.

**Humain Codeur Mastermind**

- *entrerCode() : Combinaison* : Il faut vérifier que le programme demande à l'utilisateur d'entrer une combinaison jusqu'à ce qu'il entre une combinaison dont la syntaxe est correcte. Ensuite il faudra vérifier que l'attribut combinaison du joueur codeur est bien mis à jour.

- *VerifierEntree(string entree) : bool* : Nous devons vérifier que cette méthode renvoie *true* seulement si la chaîne de caractère donnée en argument peut être transformée en une combinaison dont les couleurs sont reconnues par le Mastermind (et que la combinaison est de bonne taille).

**Humain Codeur Wordle**

- *entrerCode() : Combinaison* : Il faut vérifier que le programme demande à l'utilisateur d'entrer un mot jusqu'à ce qu'il entre un mot dont la syntaxe est correcte et qui appartient au dictionnaire. Ensuite il faudra vérifier que l'attribut combinaison du joueur codeur est bien mis à jour.

- *VerifierEntree(string entree) : bool* : Nous devons vérifier que cette méthode renvoie *true* seulement si la chaîne de caractère donnée en argument peut être transformée en un mot du dictionnaire associé à la partie.

**Humain Décodeur**

- *entrerCombinaison()* : Nous devons vérifier que la chaîne de caractère rentrée par l'utilisateur est bien conforme à une combinaison du jeu.

**Humain Decodeur Mastermind**

- *entrerCombinaison() : void* : Nous devons vérifier que la chaîne de caractère rentrée par l'utilisateur est bien conforme à une combinaison dans le jeu de Mastermind, c'est à dire une suite de couleurs parmi les couleurs autorisées par le jeu.

- *verifierEntree() : bool* : Nous devons vérifier que lorsque l'utilisateur rentre une combinaison impossible alors la fonction retourne faux et inversement dans le cas du jeu Mastermind.

**Humain Decodeur Wordle**

- *entrerCombinaison() : Combinaison* : Nous devons vérifier que la chaîne de caractère rentrée par l'utilisateur est bien conforme à une combinaison dans le jeu de Wordle, c'est à dire une suite de lettres.

- *verifierEntree() : bool* : Nous devons vérifier que lorsque l'utilisateur rentre une combinaison impossible alors la fonction retourne faux et inversement dans le cas du jeu de Wordle.

**IA Codeur Mastermind**

- *genererCode() : Combinaison* : Il faut vérifier que la combinaison générée est correcte syntaxiquement (que les couleurs choisies sont reconnues). Il faut ensuite vérifier que l'attribut combinaison est bien mis à jour.

**IA Codeur Wordle**

- *genererCode() : Combinaison* : Il faut vérifier que la combinaison générée est correcte syntaxiquement (que le mot est reconnu par le dictionnaire). Il faut ensuite vérifier que l'attribut combinaison est bien mis à jour.

**IA Decodeur Mastermind**

- *Combi_possible() : Combinaison[]* : Il faut vérifier que l'on parvient bien a créer l'ensemble des possibilités d'une longueur donnée et que cela s'effectue sans erreur et sans oubli d'aucun cas.

- *Maj_ensemble() : Void* : On peut vérifier que la méthode enlève automatiquement tous les cas considérés comme étant impossible et que l'ensemble se met à jour sans causer des erreurs.

- *Choisir_combi() : Combinaison* : Il faut vérifier que la combinaison générée est correcte syntaxiquement et que le choix se fait de la manière que l'on choisit.

**IA Decodeur Wordle**

- *Combi_possible() : Combinaison[]* : Il faut vérifier que le chargement du dictionnaire se fait sans erreurs et que notre liste contient bien tous les mots possibles.

- *Maj_ensemble() : Void* : On peut vérifier que la méthode enlève automatiquement tous les cas considérés comme étant impossible et que l'ensemble se met à jour sans causer des erreurs.

- *Choisir_combi() : Combinaison* : Il faut vérifier que la combinaison générée est correcte syntaxiquement et que le choix se fait de la manière que l'on choisit.

**Menu**

- *afficherMenu() : void* : On vérifie que l'ensemble des possibilités auxquelles le joueur a accès est bien affiché, à savoir "Joueur", "Choisir Mode de jeu", "Modifier Paramètres".

- *choisirModeDeJeu(int ModeDeJeu) : void* : On vérifie que le programme attend bien une saisie clavier de la part de l'utilisateur, et que cette saisie est bien prise en compte. C'est à dire que la valeur contenue en attribut *ModeDeJeu* de la classe *Menu* doit bien être mis à jour après la saisie du joueur. Il faudra aussi s'assurer qu'une saisie est redemandée en cas d'erreur de saisie

ou de valeur non prise en compte par le programme. D'autre part le programme doit afficher les différents choix disponibles pour que l'utilisateur sache à quel entier correspond à quel mode de jeu.

- *choisirDifficulteIA() : void* : On vérifie que le programme attend bien que l'utilisateur saisisse au clavier la difficulté qu'il souhaite mettre en place et que cette saisie est bien prise en compte. C'est à dire que la valeur contenue en attribut *difficulteIA* de la classe *Menu* doit bien être mise à jour après la saisie du joueur. Il faut aussi vérifier que la machine redemande à l'utilisateur de saisir sa difficulté en cas d'erreur (mauvaise syntaxe ou autre). D'autre part le programme doit afficher les 3 choix possibles avec le chiffre correspondant.

- *parametreDeJeu(int NB_ELEMENT, int ENSEMBLE-ELEMENT, int NB_CASE,NB_TOUR) : void* : On vérifie que l'ensemble des paramètres est bien demandé au joueur. Dans le cas limite où le joueur rentrerait un entier trop grand, vérifier qu'un message d'erreur est affiché puis vérifier qu'on lui demande bien de rentrer un nouvel entier. Vérifier ensuite que les paramètres sont bien changés si on lance une partie juste après.

- *appartientEE(string couleur) : bool* : On vérifie bien que la méthode renvoie vrai lorsque la couleur appartient aux couleurs possibles du jeu Mastermind, et faux dans le cas contraire.

- *appartientDico(string mot) : bool* : On vérifie bien que la méthode renvoie vrai lorsque le mot appartient au dictionnaire du jeu Wordle, et faux dans le cas contraire.

---

L'ensemble des tests unitaires ont été réalisés dans la classe *UnitTest.cpp*. Faire make test dans le dossier permet d'ensuite visualiser les résultats dans le fichier texte "resultat.txt".

# 5. Conclusions et Perspectives

Pour commencer ce projet a été très formateur pour nous tous dans plusieurs domaines de la programmation.

Premièrement nous avons pu appliquer les nombreuses notions vues en cours sur notamment toute la partie conception avec laquelle nous n'étions pas si familier avant cela. Cela nous a permis de comprendre l'importance de la conception pré-implémentation surtout lorsque le projet se réalise en groupe. Nous avons pu nous familiariser d'avantages avec les nombreux diagrammes du modèle UML.

Grâce à une bonne conception pré-code, nous avons pu facilement travailler sur un projet où l'ensemble des personnes du groupe était d'accord, cela a permis de gagner du temps et d'avoir une bonne synergie au sein du groupe.

Le langage C++ était pour nous tous un nouveau langage que l'on a su maîtriser grâce aux différents TD mais surtout grâce à ce projet qui nous a demandé une rigueur et une propreté du code indiscutable.

Outre la partie pré-codage et implémentation, la partie gestion de projet était omni-présente dans ce projet. En effet cela fut la première fois que l'on faisait un projet avec un aussi grand groupe. Il a fallu s'adapter, maîtriser des outils comme GIT permettant ainsi de travailler sur la même version à chaque fois. La partie "management" également avec la bonne répartition des tâches pour chacun, les deadlines fixées pour que le projet avance en temps et en heure par exemple.

Pour conclure nous avons apprécié travailler sur ce projet qui comportait un sujet qui nous était familier, le mastermind et le wordle, mais avec des notions encore trop floues au départ, ce qui nous a permis de mieux comprendre et de savoir utiliser de nouveaux outils. La taille du groupe était plus importante qu'à notre habitude mais cela n'a pas posé de problèmes, nous avions plus d'idées, cela demandait juste une phase plus clair de conception et de spécifications pour être sûr que nous allions tous dans la même direction au moment de commencer à implémenter nos parties respectives.

Si nous avions eu plus de temps nous aurions pu réfléchir à d'autres fonctionnalités à rajouter telles que la possibilité d'augmenter ou diminuer la difficulté de l'IA, la possibilité d'arrêter une partie puis de la recommencer une fois le programme terminé, ou encore de faire une interface graphique permettant à l'utilisateur de jouer avec sa souris.