Rapport de conception
AP4B

DIEBOLT Matthieu, ATECHIAN Theo, BIN ROSLI Amir Zikry

Sommaire :

Les modifications apportées à la conception après la phase d'implémentation sont notées en vert dans la partie CONCEPTION, de plus une partie IMPLEMENTATION a été rajoutée au rapport

Dans le cadre de l'UE **Ap4B**, nous avons pour objectif de réaliser en groupe de trois un projet regroupant les différentes notions de Java et d'UML vues en cours et en TD. Pour ce faire, nous avions le choix entre deux sujets. Parmi eux, nous avons choisi le jeu de société ***Turing Machine.*** Notre but sera de recréer une version numérique adaptée à l'UTBM de ce jeu.

Introduction : Turing Machine

Turing Machine est un jeu de déduction très modulaire qui peut être joué en compétition, en coopération ou même en solo. Les joueurs doivent trouver un mot de passe en interagissant le moins possible avec la machine. À chaque tour, le joueur interagit au maximum 3 fois pour en déduire le mot de passe.

Pour s'adapter dans le monde de l'UTBM, on a décidé de changer le concept du jeu afin qu'ils s'agissent d'un groupe d'étudiants à la recherche d'une salle mystérieuse de l'établissement. Cette salle mystérieuse, ne reste pas à sa place, elle change à chaque partie.

Afin d'aider les étudiants à retrouver la salle mystérieuse, ils peuvent interroger une machine en proposant un numéro de salle. Les salles de l'UTBM sont composées de 3 chiffres allant de 111 à 555, de plus le 1er chiffre de la salle indique le nombre d'heure de CM dispenser dans la salle, le 2eme, le nombre d'heure de TD et le 3eme, le nombre d'heure de TP. Ainsi les étudiants à chaque journée peuvent interroger 3 fois la machine qui leurs proposent 4 à 6 choix de questions sur lesquelles tester leurs codes. Les questions permettent aux étudiants d'identifier le nombre d'heure des différents types de cours. Pour qu'au final, l'un d'entre eux retrouve la fameuse salle mystérieuse.

CONCEPTION :

La première partie de ce rapport porte sur la phase de conception de notre projet. Elle nous sera utile pour nous mettre d'accord sur la vision d'ensemble du projet, ainsi que pour nous aider dans l'implémentation de celui-ci.

I. Description générale

Dans cette partie, nous allons exposer une première vision générale de notre projet dans le cadre de la phase de conception. Cependant, certains points peuvent être amenés à évoluer au cours de la phase d'implémentation, en fonction des difficultés rencontrées ou de nouvelles idées.

1. Structure générale

Notre version du jeu ***Turing Machine*** sera implémentée uniquement en Java, un langage orienté objet. Les avantages de l'utilisation d'un langage orienté objet résident dans son extensibilité et sa réutilisabilité. Cela nous permettra, si nous le souhaitons, d'approfondir le projet à l'avenir, de faciliter la phase de maintenance et de résolution de problèmes, et même de réutiliser certaines classes pour de futurs projets. De plus, l'utilisation de Java nous permet d'implémenter assez facilement des interfaces graphiques. Pour les interfaces graphiques, nous allons principalement utiliser Swing.

Notre projet se divisera en quatre grandes parties : la gestion et la création des problèmes, la gestion et la création des joueurs, la vérification de la véracité des cartes critères selon un code donné, et enfin tout ce qui concerne les menus, avec la page d'accueil et la création de partie regroupant les joueurs et le problème.

Pour la gestion et la création des problèmes, nous allons créer une classe mère avec trois classes filles qui seront appelées en fonction du nombre de cartes critère souhaité dans le problème (voir diagramme de classe pour plus de détails). Les problèmes, quant à eux, seront chargés et choisis aléatoirement à partir de fichiers texte. Dans chaque fichier texte, on retrouvera le code réponse ainsi que les identifiants (id) des cartes associées à ce problème. Ensuite, la classe Problème à partir des identifiants obtenus, devra chercher les cartes correspondantes dans une base de données de cartes pour construire le problème. Enfin, les problèmes devront s'adapter graphiquement au nombre de cartes sélectionnées. Par exemple, si le joueur souhaite jouer avec cinq cartes, il faudra en afficher cinq…

2. Fonctionnalités

En termes de fonctionnalités graphique pour l'utilisateur de notre version du jeu ***Turing Machine***, nous avons prévu une page d'accueils ou l'utilisateur pourra consulter les règles, initialiser les paramètres de sa partie et la lancer. Les paramètres comprennent le choix du nombre de cartes critères souhaité, allant de 4 à 6, et le choix du nombre de joueurs, allant de 1 à 4. Ainsi, l'utilisateur de notre application pourra incarner plusieurs joueurs s'il le souhaite, jouer à plusieurs avec ses amis sur un même PC, ou bien jouer en solo contre la machine.

Une fois la partie lancée, l'utilisateur pourra voir à l'écran la partie avec le nombre de cartes critères qu'il a choisi. Il pourra ensuite sélectionner le joueur qui souhaite jouer et choisir une carte à interroger avant de proposer un code à vérifier sur cette carte. Si le code est juste, le joueur a gagné sinon la partie se poursuit. Dans notre version, nous avons donc supprimé la phase où tous les joueurs proposent leurs codes en même temps pour déterminer s'il y a un vainqueur. Nous pensons que cela rendra le jeu un peu plus amusant en introduisant le facteur temps : le joueur le plus rapide prend plus de risques (moins de temps de réflexion), mais il peut gagner avant les autres. De plus, la partie gérera le nombre d'essais des différents joueurs, car ils ne peuvent interroger que trois cartes par tour. Une fois que chaque joueur a interrogé la machine trois fois, on passe au tour suivant. L'utilisateur a également à tout moment la possibilité de quitter la partie ou relancer une partie.

Nous avons également réfléchi à d'autres fonctionnalités, mais nous ne les avons pas faites apparaître dans les différents diagrammes UML, car il s'agit de fonctionnalités secondaires que nous introduirons ultérieurement si notre jeu de base est fonctionnel et si nous avons le temps de les intégrer.

Parmi ces fonctionnalités, on retrouve la possibilité de consulter l'historique des propositions, une gestion du temps avec la proposition d'indices si la partie dure trop longtemps, ainsi que la possibilité de renommer un joueur. Et bien plus encore si de nouvelles idées émerge durant la phase d'implémentation.

Finalement, en plus de l'implémentation des fonctionnalités de base, nous avons introduit une horloge *timer* dans le jeu, ainsi qu'une section paramètres qui propose la gestion des changements de pseudo, ainsi que celle de l'horloge (play/pause).

3. Organisation des tâches

En ce qui concerne l'organisation des tâches, étant un groupe de trois personnes, nous nous sommes réparti les tâches en trois grandes parties qui nous semblent équilibrées. Une personne se chargera de la partie menu/création de partie ainsi que de la gestion des joueurs. Une deuxième prendra en charge tout ce qui concerne la création et la gestion des problèmes. Enfin, une troisième personne s'occupera de la vérification et de la création des cartes critères.

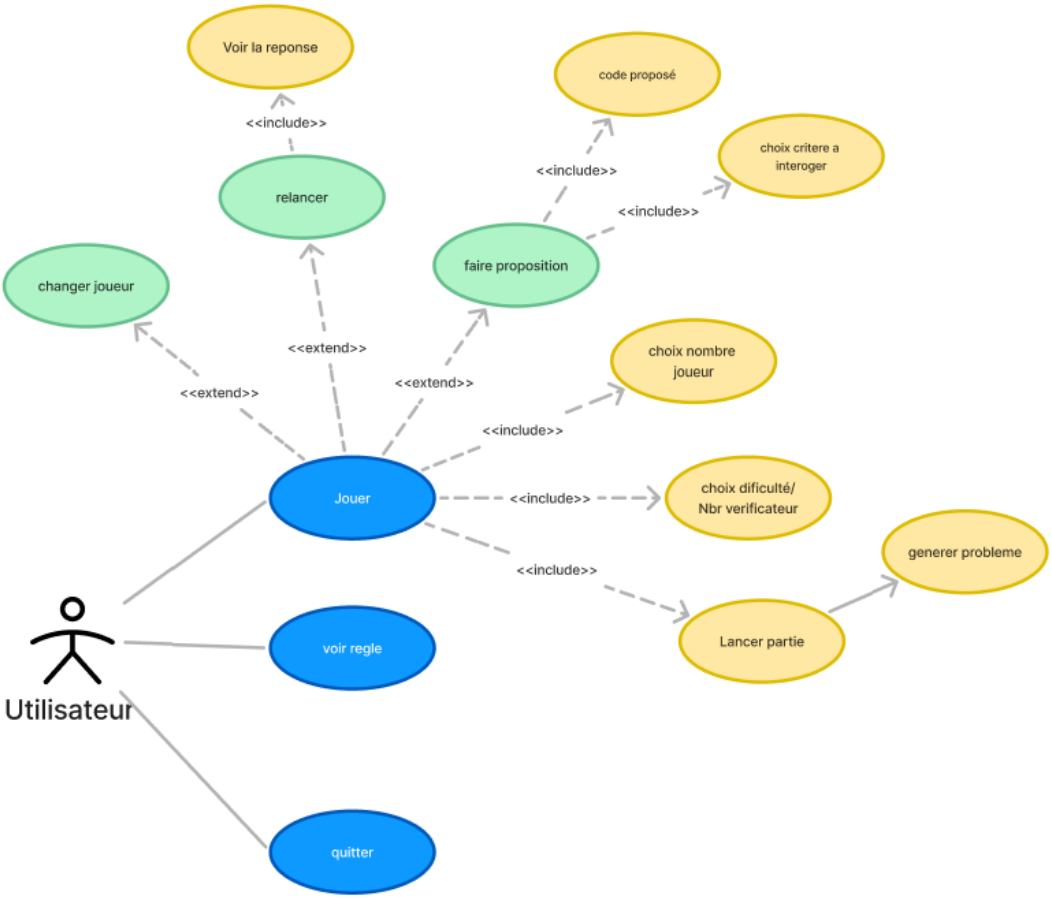
Il s'agit là d'une estimation de la répartition des tâches, qui peut évoluer si nous nous rendons compte que la répartition est inégale ou si une personne rencontre des difficultés sur une méthode ou autres.

II. Les diagrammes UML

Pour simplifier la phase de développement de ce projet, nous avons d'abord réalisé un travail de conception à l'aide des diagrammes UML. Nous avons élaboré 5 types de diagrammes différents. En plus des 2 diagrammes demandés, nous avons ajouté le diagramme de séquence et le diagramme d'états-transition. Nous avons choisi de concevoir ces deux diagrammes supplémentaires car nous les trouvions pertinents pour une meilleure compréhension du déroulement d'une partie. De plus, nous avons également réalisé des diagrammes d'objets pour mieux comprendre les différentes instances de classes qui existeront au cours d'une partie.

1. Diagramme de cas d'utilisation (Use Case)

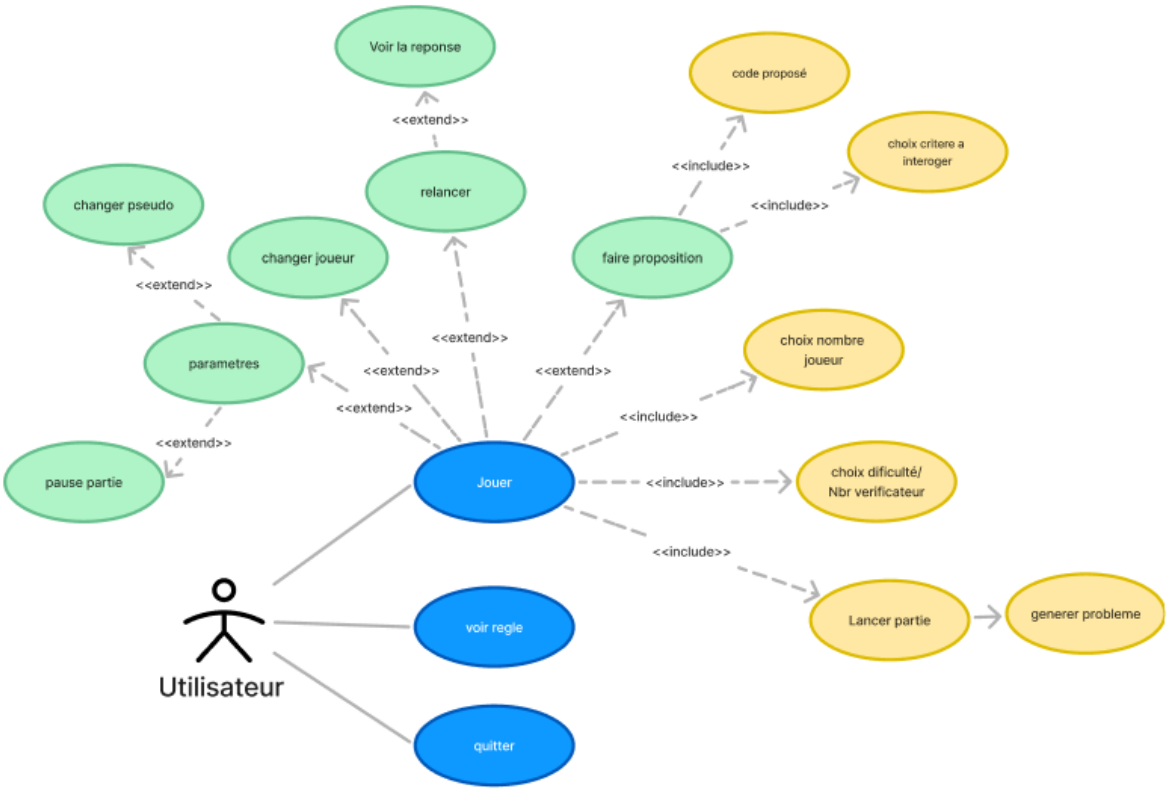
Le diagramme de cas d'utilisation est un système conçu pour les utilisateurs. Il fait la description des comportements du système vis à vis de l'utilisateur et nous permet de faciliter notre compréhension d'un projet grâce à sa représentation simple et expressive. Ce diagramme simplifie les échanges entre un client et le développeur du cahier des charges



L'utilisateur a d'abord le choix entre voir les règles, jouer et quitter le jeu. Pour pouvoir jouer, le joueur doit obligatoirement avoir choisis le nombre de joueur, le nombre de carte critère souhaité et lancer la partie ce qui va ensuite nous générer un problème avec les options choisis.

Une fois qu'on est dans la partie (le jeu), on a le choix de relancer le jeu qui avant de relancer une partie nous montre la solution du problème quitté. De plus le joueur a la possibilité dans le jeu de faire une proposition de code (qui est composé de 3 chiffres) pour cela il doit obligatoirement avoir proposer un code et avoir sélectionner un critère. L'utilisateur a également la possibilité dans la partie de changer de joueur.

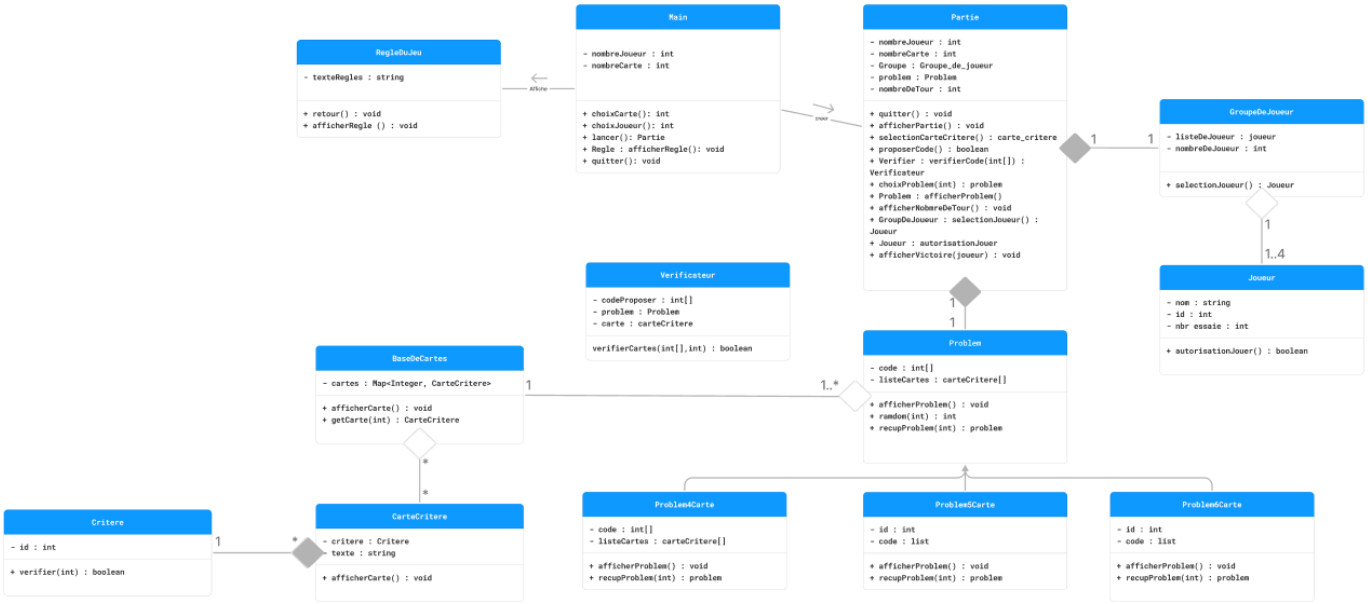
Voici la version après implémentation :



Le diagramme reste très similaire, nous avons juste ajouté la possibilité à l'utilisateur d'accéder à des paramètres où il pourra changer son pseudo ou encore mettre en pause la partie. Nous avons également changé le lien de 'voir la réponse' de <<*include*>> à <<*extends*>> afin de laisser le choix à l'utilisateur de voir ou non la réponse quand il quitte.

2. Diagramme de classe

Un diagramme de classe est une représentation statique de l'architecture que prendra un projet. Il montre les différentes classes avec leurs méthodes et attributs, ainsi que les relations qu'elles ont entre elles. Le diagramme de classe sert donc à modéliser la structure et l'organisation que prendra le projet, ainsi que les méthodes et attributs qu'il comportera et où elles se situeront. Il peut également servir de documentation au projet pour rapidement comprendre la structure générale.



Sur le diagramme, on peut voir un certain nombre de classes. La classe principale, où l'utilisateur se trouve lorsqu'il entre dans l'application, est la classe Main.

Dans la classe Main, on retrouve les méthodes pour initialiser et lancer la partie, mais également pour quitter le jeu et afficher ses règles.

Lorsque la méthode qui affiche les règles dans la classe Main est utilisée, elle fait appel à la méthode d'affichage des règles présente dans la classe RegleDuJeu.

Lorsque l'utilisateur lance une partie via la méthode Lance() dans la classe Main, cela crée une instance de la classe Partie avec le nombre d'utilisateurs choisis ainsi que le nombre de cartes sélectionnées dans Main. La classe Partie possède un grand nombre de méthodes puisqu'il s'agit de notre classe principale, qui gère le jeu dans son ensemble. Elle regroupe notamment le système de joueurs par composition, en créant un groupe de joueurs à l'aide de la classe GroupeDeJoueur, ainsi que la création du problème (par composition également) en utilisant la classe Probleme. Une partie est composé d'un groupe de joueur et d'un problème. Sans partie il ne pourrait pas exister de problème ou de groupe de joueur. Enfin, c'est à partir de cette classe que l'on récupère le code entré par le joueur ainsi que la carte sélectionnée, avant de lancer sa vérification via la classe Verification. Elle possède également d'autres méthodes destinées à la gestion de la partie.

La classe GroupeDeJoueur servira à la création d'un groupe de joueurs allant de 1 à 4, et pourra donc créer au maximum 4 instances de la classe Joueur par agregation.

La classe Probleme est, quant à elle, une classe mère avec trois classes filles, correspondant au nombre de révélateurs souhaité. Nous avons fait le choix de créer des sous-classes puisque chaque problème, bien qu'il s'agisse d'un problème unique, présente des particularités différentes (un nombre de cartes différent, un nombre de problèmes définis dans le fichier texte

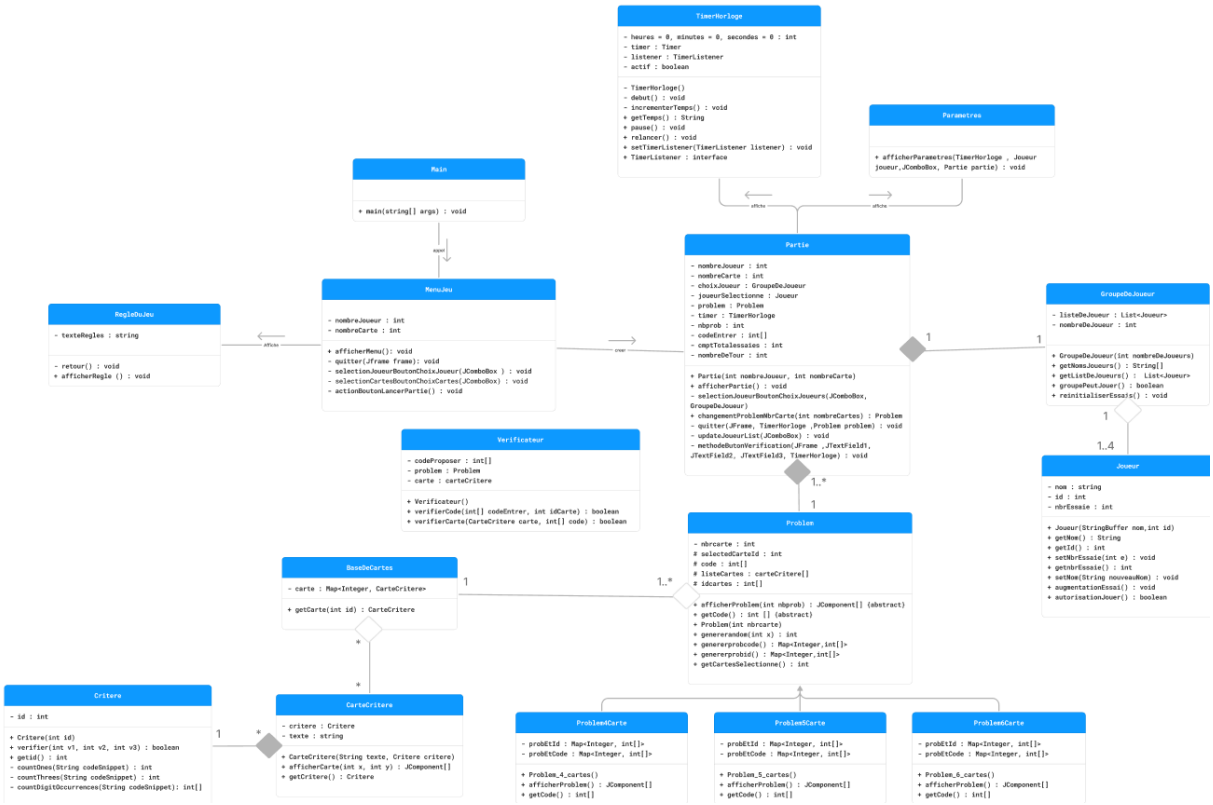
distinct, etc.). Cela nous laisse également la possibilité d'introduire de nouvelles fonctionnalités par la suite, comme par exemple donner un indice dans les problèmes plus difficiles avec 6 cartes. La classe Probleme possède des méthodes pour récupérer les problèmes depuis un fichier texte et pour générer un nombre aléatoire permettant de sélectionner un problème différent à chaque partie. À partir de ces deux méthodes, elle stocke dans ses attributs le code réponse et recherche les cartes correspondantes, selon leurs identifiants, dans la classe BaseDeCartes pour les ajouter à sa liste de cartes. Enfin, Probleme affiche les cartes récupérées.

Notre système de cartes critères est composé de deux classes : la classe CarteCritère, qui représente la carte elle-même avec le texte à afficher, et par composition un critère provenant de la classe Critère. Dans la classe Critère, nous allons initialiser un certain nombre de critères possibles pour le jeu et leur attribuons un id unique. Lors de la création d'une carte critère, nous pourrons alors fournir à la carte le texte à afficher ainsi que l'id du critère correspondant.

De plus nous allons créer par agrégation un certain nombre de cartes jouables dans la classe BaseDeCartes en utilisant le système de dictionnaire intégré à Java. Ainsi, chaque carte possédera un id propre à la carte creer et pourra être récupérée pour un problème donné.

Finalement, la dernière classe, la classe Verificateur, devra se charger du questionnement d'un joueur, en fonction d'un code donné ainsi que de la carte sélectionnée dans la liste des cartes du problème. Dans un premier temps, elle vérifiera si le code correspond au code correct, et si ce n'est pas le cas, elle interrogera le critère de la carte donnée et retournera sa véracité ou non.

Voici la version après implémentation :



Le diagramme a pas mal évolué tout en gardant la même logique de base. De nouvelles classes ont été créées telle qu'une classe 'main' (l'ancien 'main' est devenu 'MenuJeu'), 'Paramètres' et 'TimerHorloge'. Ces deux dernières classes ont été ajoutées pour associer des fonctionnalités au jeu en plus de celles prévues de base. Beaucoup de noms d'attributs et de méthodes ont changé pour laisser place à des noms plus explicites dans le code.

De plus, de nombreuses modifications résident dans l'ajout de *getters* et *setters* mais aussi dans l'adaptation Java avec une interface graphique. Par exemple, certaines de nos méthodes doivent prendre des labels en paramètre pour fonctionner ou encore retourner des composants graphiques, ce que nous ne savions pas avant l'implémentation.

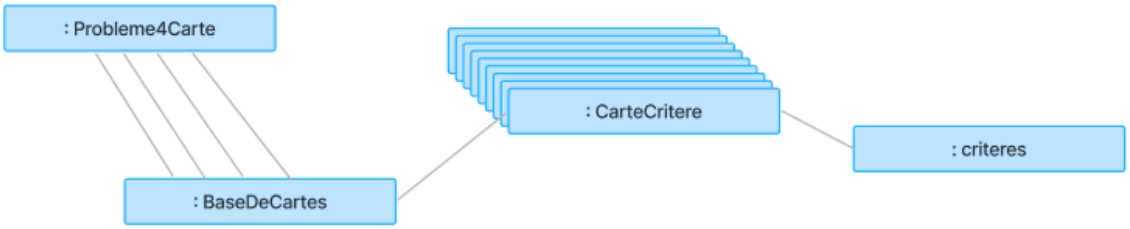
Nous avons également ajouté de nouvelles méthodes auxquelles nous n'avions pas pensées pendant la phase de conception, même si notre logique reste la même. Le plus grand changement de logique que nous avons effectué pendant l'implémentation est la gestion du nombre d'essais des joueurs. Pendant l'implémentation, nous nous sommes rendu compte que nous n'avions pas assez pris en compte cette partie. Nous pensions juste à faire des « tours » dans la partie, mais nous n'avions pas assez réfléchi à comment les gérer. Nous avons donc ajouté plusieurs méthodes dans la gestion des joueurs à cet effet.

3. Diagramme d'objets

Un diagramme d'objets est une représentation statique à un moment donné de l'exécution, qui montre les instances créées d'objets. Il permet également d'illustrer les relations entre les instances des objets. Ainsi, les diagrammes d'objets nous permettent de mieux visualiser l'état du système à un instant précis en nous montrant les instances créées et les relations entre elles.

Voici plusieurs diagrammes d'objets qui montre les instanciations de certaines classes.

Gestion des cartes :

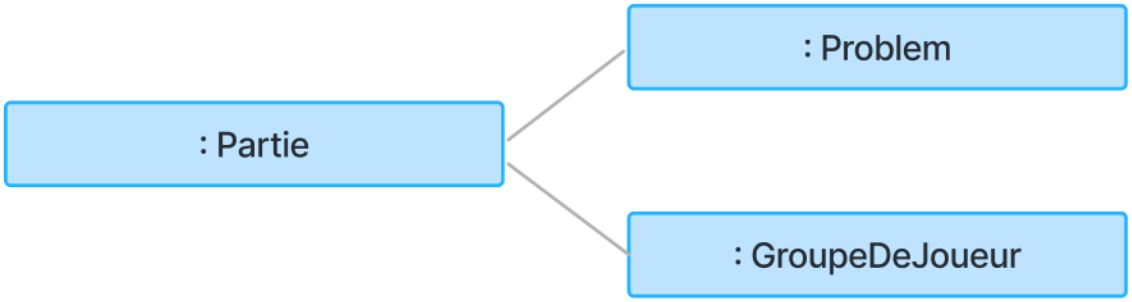


Lorsqu'un problème est créé, il va chercher les cartes indiquées dans le fichier texte dans la classe BaseDeCartes selon leurs ID. La classe BaseDeCartes instancie plusieurs fois la classe CarteCritere, créant ainsi toutes les cartes critère qui seront jouables. Les cartes critère sont chacune composées de critères provenant de la classe Critere.

Gestion des joueurs :

La classe GroupeDeJoueur instancie entre 1 et 4 fois la classe Joueur en fonction du choix de l'utilisateur pour créer le groupe de jeu.

Gestion d'une partie :



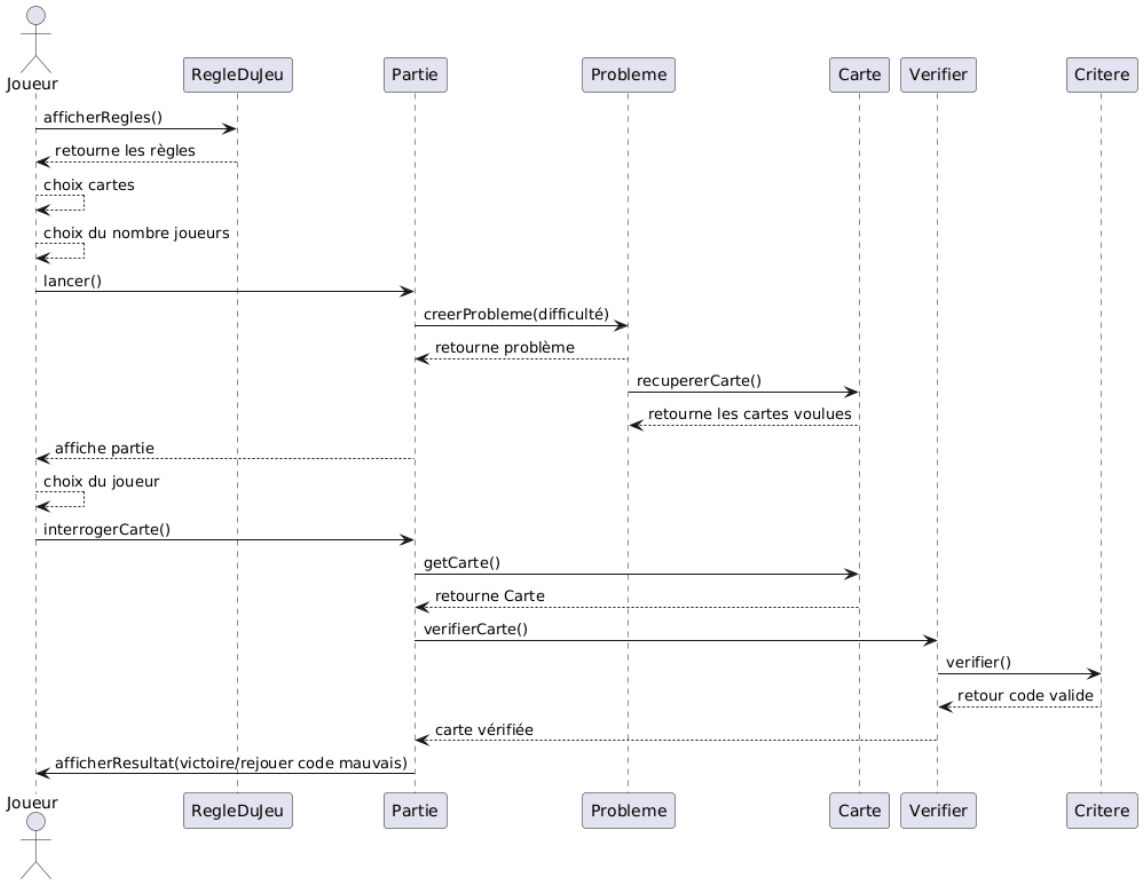
Une Partie est composée d'une instance de la classe Problem et d'une instance de la classe GroupeDeJoueur.

Les diagrammes d'objets restent les mêmes, notre logique d'instanciations n'a pas changé.

4. Le diagramme de séquence

Le diagramme de séquence permet de modéliser les interactions entre les différents objets à la suite d'un évènement externe au système.

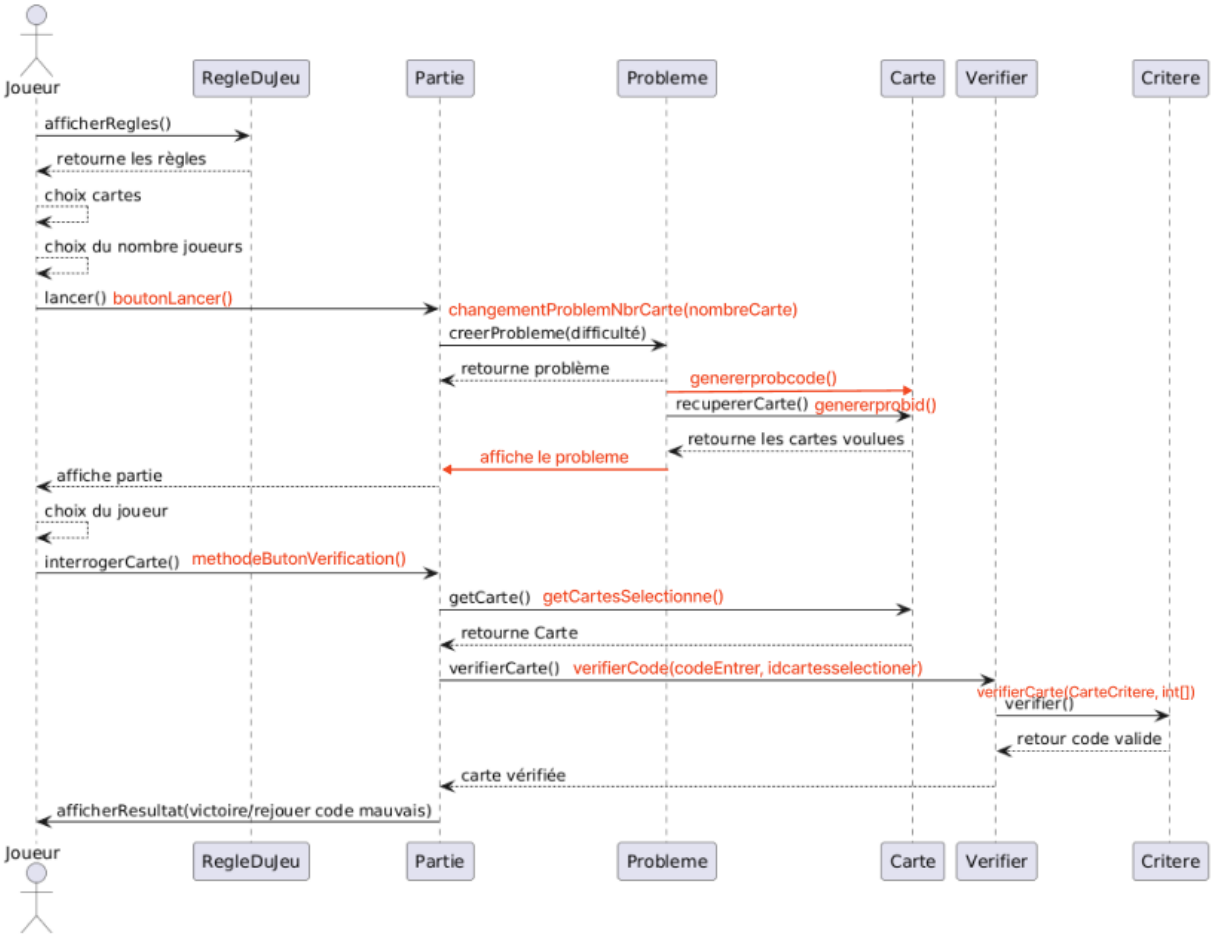
Ce diagramme explique le déroulement d'une partie du jeu Turing Machine :



Tout d'abord, le joueur demande à voir les règles du jeu. Ensuite il sélectionne les cartes critères ainsi que le nombre de joueur dans la partie. Par la suite, une partie se créer, i.e. un problème va être mis en place en fonction de la difficulté choisie, des cartes critères sélectionnées. Vient ensuite, la sélection du joueur qui va commencer la partie, le joueur va alors, demander une carte critère, il va par la suite vérifier son code avec cette carte. Enfin, après vérification, le joueur sait si son code est bon ou mauvais. Dans le cas d'un code correct, le joueur gagne. Dans l'autre cas, le joueur renouvelle son action en interrogeant et vérifiant avec une seconde carte, jusqu'à trois par tours.

À noter que dans ce diagramme, nous ne prenons pas en compte la gestion du changement de joueur en cours d'une partie, ni la gestion de la création des cartes critères afin de rendre le diagramme plus simple et compréhensible.

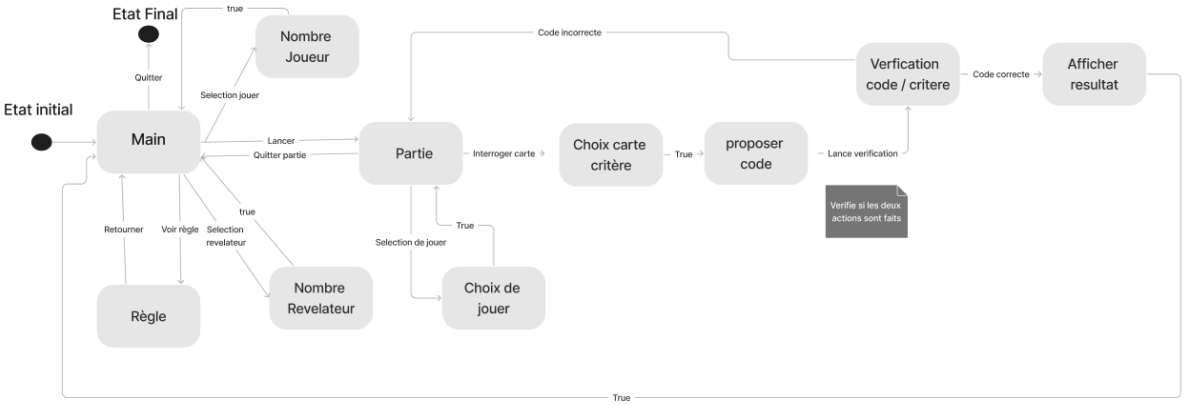
Voici la version modifiée du diagramme de séquence :



Le diagramme de séquence reste le même, seulement certaines méthodes ont changé de nom pour donner des noms plus implicites au code. Il s'agit d'une simplification du diagramme. Certaines fonctionnalités et méthodes manquent mais il nous montre bien que notre structure générale a peu évoluée, seulement certaines méthodes utiles à cette structure ont été ajoutées.

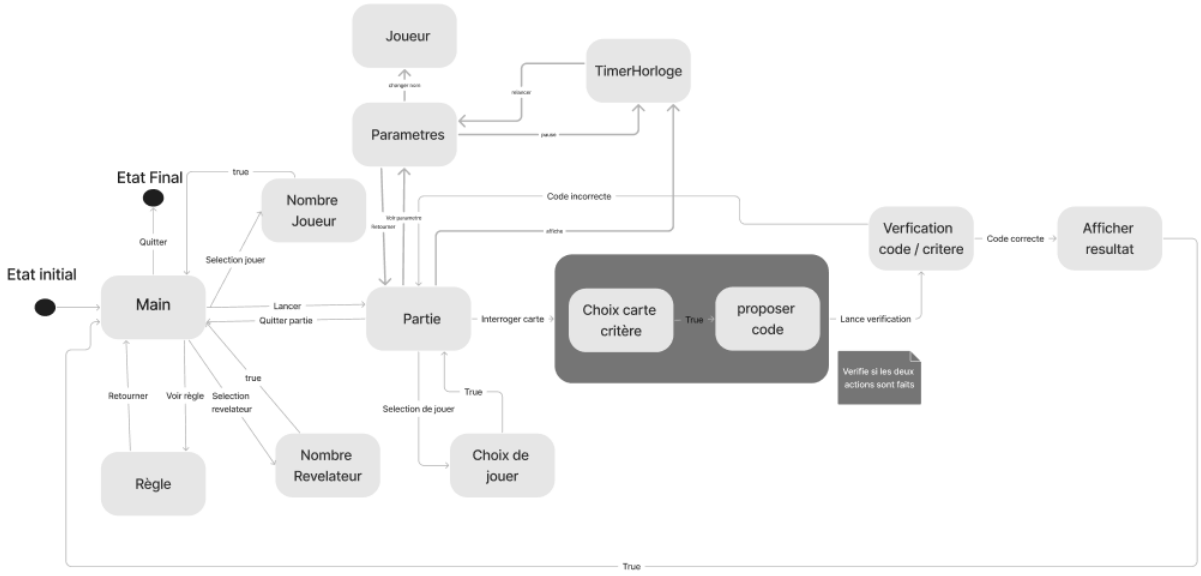
5. Le diagramme d'états-transition

Le diagramme d'états-transition décrit le comportement des objets d'une classe au moyen d'un automate à états finis. Chaque état est stable et durable ce qui veut dire qu'ils attendent des évènements pour changer d'état.



Au début, l'utilisateur commence dans la main, c'est-à-dire l'interface menu principale du jeu. On peut, si on le souhaite, visualiser les règles avant de revenir à l'interface principale, quitter le jeu ou encore lancer une partie. Si l'option lancer a été choisie, une partie de jeu est créée. À cet état, il est possible de sélectionner un joueur, puis d'interroger une carte critère. Pour passer à l'étape suivante, la vérification, il est nécessaire d'avoir choisi une carte critère ainsi que proposé un code à 3 chiffres. Si le code proposé est correct, on gagne le jeu et on revient à l'interface principale. Dans le cas contraire, la partie en cours continue.

Voici la nouvelle version du diagramme d'états-transition :



Ici nous avons juste rajouté une petite partie avec les paramètres possibles pour l'utilisateur, le reste du diagramme ne change pas.

Conclusion conception

A travers ce rapport, nous avons pu nous apercevoir que le travail de conception est très important avant de commencer la programmation d'un projet. Avec ces diagrammes fait, l'implémentation de ce projet en Java sera simplifiée et prendras surement moins de temps que si nous n'avions pas réaliser cette phase. A travers ce projet, nous espérons, pouvoir améliorer nos compétences, de travail de groupe, de développement, et de conception tout en apprenants un nouveau langage qui est le Java. Enfin nous pensons que ce projet nous permet d'apprendre une manière de gérer un projet et nous aideras dans nos futures carrières d'ingénieur.

IMPLEMENTATION

La seconde partie de ce rapport porte sur la phase d'implémentation et sur l'analyse post-réalisation de notre projet.

Introduction

Nous avons commencé l'implémentation du projet assez rapidement après avoir finalisé nos diagrammes UML de conception, afin de ne pas être pressés par le temps avant le rendu. Pour cette phase de codage, nous avons essayé au maximum de nous tenir à la conception réalisée sans pour autant nous limiter, par exemple en adaptant les noms des fonctions pour en donner des plus logiques au moment du codage. Nous avions surtout pour objectif de garder au maximum la même structure.

I. Fonctionnement et fonctionnalités

D'abord nous arrivons à l'interface principale où nous pouvons retrouver des boutons qui font des tâches différentes. Par exemple, le bouton règle pour afficher les règles du jeu et bouton quitter pour quitter le jeu. Pour commencer le jeu, il faut choisir le nombre de joueurs et cartes critères souhaités et cliquer sur le bouton lancer. Un joueur et quatre cartes critères sont choisis par défaut. Le bouton lancer nous ramène à l'interface suivante qui est l'interface partie. Dans cette fenêtre, il y a un bouton paramètres qui donne le choix aux joueurs de donner leurs noms dans un champ texte. La notion timer est aussi implémentée pour mesurer la durée de chaque partie. Les joueurs ont le choix d'arrêter le timer et le redémarrer. A chaque tour, le joueur doit choisir son nom dans la liste des joueurs, choisir une carte critère, proposer un code pour vérifier la carte critère ou directement deviner le code réponse. Chaque joueur dispose de 3 essais pour interroger les cartes par tour. Le premier joueur qui trouve le code gagne !

II. Explication du code

Cette section a pour but d'expliquer le fonctionnent de notre code. Dans un premier temps de façon général puis de façon plus ou moins détaillée classe par classe.

Quand on lance le jeu, le main crée une instance de la classe MenuJeu et utilise sa méthode d'affichage pour afficher le menu. Dans le menu, on utilise plusieurs boutons qui possèdent chacun leur méthode, dont deux pour changer les paramètres du jeu, le nombre de joueurs et le nombre de cartes. Chaque bouton dans notre projet appelle des méthodes via des lambda expressions. Ensuite, lorsque le bouton lancer est cliqué dans le menu, sa méthode crée une instance de la classe Partie grâce à son constructeur, qui prend le nombre de joueurs et le nombre de cartes. De plus, dans le constructeur de la classe Partie, en plus d'initialiser les paramètres utiles à la partie, un Groupe de joueurs est créé à partir du constructeur de la classe GroupeDeJoueur, qui lui-même appelle n fois le constructeur de la classe Joueur. Enfin, un problème est créé dans le constructeur de la partie. Une méthode est utilisée pour cette création de problème selon le nombre de cartes souhaitées. Par exemple, si on est dans une partie à 4 cartes, la méthode nous permettra d'appeler le constructeur de la classe Problem_4_Cartes. Pour 5 et 6 cartes, on appellerait également le constructeur de leur classe respective pour la création de la classe Problem, étant donné que les 3 classes Problem_n_Cartes héritent de la classe Problem. Seules les méthodes d'affichage sont différentes pour chacune de ces sous-classes.

Une fois une instance de Partie créée avec son constructeur, sa méthode afficher est lancée, qui elle-même, en plus d'afficher ses propres éléments possédant sa propre méthode, lance la méthode d'affichage des sous-classes Problem par polymorphisme.

Dans la méthode d'affichage de problème, on utilise les 3 méthodes de la classe mère pour la récupération, de façon aléatoire, d'une ligne qui contient les ID des cartes et le code réponse qui devront être utilisés pour ce problème. Il existe 3 fichiers différents selon le nombre de cartes voulues. À partir des ID récupérés, la méthode d'affichage de problème va elle-même appeler la méthode d'affichage de chacune des cartes, en plus de gérer la sélection avec des boutons radio pour le groupe de cartes récupéré. Le code récupéré dans le fichier est stocké dans un attribut de la classe Problem.

En ce qui concerne le système de cartes, la création ne se fait pas selon les ID récupérés, mais un grand nombre est créé de base dans une classe nommée BaseDeCartes. Donc, quand le jeu est lancé, toutes les cartes sont créées, mais seulement certaines sont utilisées selon les ID récupérés dans les fichiers. Pour la base de cartes, on utilise une HashMap, avec un ID (celui qu'on récupère dans les fichiers) et une carte qui est créée avec son constructeur. Dans notre projet, une carte est créée à partir de la classe CarteCritère, qui possède du texte et une instance de la classe Critère. Ces critères sont seulement composés d'un ID qui nous servira à vérifier plusieurs critères différents.

Enfin, lorsqu'une vérification est lancée à partir du bouton de la classe Partie, un objet de la classe Vérificateur est créé. À partir de cette instance, on appelle sa méthode de vérification du code entré en prenant en paramètre l'ID de la carte sélectionnée, ainsi que le code proposé. Cette méthode récupère la carte à partir de l'ID dans la base de cartes et appelle une seconde méthode de vérification qui prend une carte et un code, et qui va récupérer l'ID du critère de la

carte. À partir de celui-ci et du code proposé, elle va appeler la méthode de vérification de la classe Critère. L'instruction vérifier change selon l'ID du critère. Par exemple, un critère avec un ID 1 vérifie : v1 > 1, ou le 2 : v2 > v1…

De plus, en général, dans toutes nos classes, nous avons essayé, lorsqu'on l'estimait nécessaire, d'ajouter des "try catch" un peu partout dans le but de limiter au maximum le crash du jeu si une erreur non contrôlée se produit.

Voici donc l'explication de ce que nous estimons être la structure générale de notre projet. D'autres classes ou méthodes non présentées ici sont également présentes dans notre jeu, mais ne sont pas nécessaires à son fonctionnement de base ou à sa compréhension en général.

Cependant voici des explications supplémentaires pour chacune de nos classes :

Main

C'est la classe qui est appelée lorsque notre jeu est exécuté, elle ne fait que créer et afficher le menu du jeu.

BaseDeCartes

Cette classe est là où nous créons nos cartes critères avec pour chacune leur description ainsi que leur l'ID à l'aide d'un tableau Hashmap.

CarteCritere

Cette classe permet la création des cartes du jeu, elles sont composées de texte et d'une instance de la classe critère. De plus, elle possède une méthode pour afficher une carte de façon formatée (avec le texte au centre d'une carte possédant des bords arrondis).

Critere

Cette classe, qui est utilisée pour le système de cartes, contient une instance présente dans chacune des cartes. Le critère n'est formé que d'un seul attribut id. À partir d'une méthode, on peut interroger un critère à partir de son id. La règle à vérifier change selon l'id du critère. Par exemple, v1 > v2 ou v1 + v2 = 5… Des méthodes supplémentaires utiles aux règles mises en place sont également présentes dans cette classe, tel que pour compter le nombre de 1 dans un code donné…

GroupeDeJoueur

Cette classe permet la création d'une liste de joueurs de la taille souhaitée. Son constructeur, qui prend en paramètre le nombre de joueurs dans la partie, crée les joueurs à partir de la classe joueur et les ajoute à la liste à l'aide d'une boucle. D'autres méthodes nécessaires à la gestion du groupe sont également présentes, comme une méthode pour obtenir la liste des noms de joueurs, la vérification si tous les joueurs sont à 3 essais, et enfin la remise à zéro collective des nombres d'essais. Cette fonction nous permet d'éviter la création de joueurs un par un dans la partie, ainsi que de faciliter certaines gestions des joueurs, comme vérifier d'un seul coup si tous les membres ont atteint la limite d'essais...

Joueur

Cette classe nous permet de créer des joueurs qui sont constitués d'un nom, d'un id et d'un nombre d'essais. Des méthodes getter et setter nécessaires à leur gestion sont présentes, en plus de méthodes qui gèrent les essais des joueurs.

MenuJeu

Cette classe nous permet d'afficher l'interface principale du jeu. De plus, ici, nous avons mis des boutons et des comboBox pour choisir le nombre de joueurs ainsi que le nombre de cartes critères voulus. Nous avons également ajouté plusieurs boutons pour permettre à l'utilisateur de voir les règles, de quitter le jeu et, finalement, de lancer une partie. Des méthodes pour chacun de ces boutons sont également présentes dans cette classe.

RegleDuJeu

Cette classe est utilisée pour afficher les règles du jeu, qui sont créées avec l'aide de labels où l'on met du texte. Elle possède aussi un bouton de retour, créé pour disposer le frame et retourner à l'interface principale.

Partie

Cette classe gère les parties. Comme expliqué dans l'explication générale, elle crée et affiche le problème de la taille voulue et crée le groupe de joueurs. En plus d'afficher le problème, dans sa méthode d'affichage, elle affiche un timer de la classe TimerHorloge, ajoute un bouton pour les paramètres, qui appelle lui aussi la classe Paramètres, un bouton de sélection du joueur choisi

ainsi que l'affichage de son nombre d'essais, qui se met à jour dynamiquement, l'ajout d'un espace pour proposer un code, et enfin un bouton pour lancer la vérification. Chacun de ces boutons possède sa méthode (ou en appelle une d'une autre classe). Une méthode est également présente pour mettre à jour l'affichage des noms de la liste de joueurs proposée par le bouton de sélection de joueur. Cette dernière permet lors d'un changement de pseudo, de mettre à jour la liste avec le nouveau pseudo.

Parametres

À travers la classe paramètres, l'utilisateur peut changer le pseudo du joueur actuellement sélectionné. Le nouveau nom est saisi dans un textfield et ensuite récupéré et attribué à l'attribut nom du joueur en utilisant un setter. Elle offre également des fonctionnalités pour le timer, tel que mettre en pause et relancer le timer. Cela se fait via des boutons et des appels de méthodes de la classe TimerHorloge. Les paramètres sont affichés via sa méthode d'affichage. À noter que nous avons fait le choix que si l'utilisateur quitte l'interface paramètres avec le bouton quitter et que le bouton pause est pressé, le compteur se remet en marche. Cela permet, selon nous, d'éviter des "triches" sur le timer, comme proposer des codes alors que le temps, et donc la partie, sont en pause.

TimerHorloge

Cette classe implémente un chronomètre qui suit le temps écoulé en heures, minutes et secondes en utilisant les threads ainsi que la classe Timer de Java. Le chronomètre démarre automatiquement à la création de l'objet et incrémente chaque seconde, sauf s'il est mis en pause. Le temps actuel peut être récupéré avec la méthode getTemps. Le chronomètre propose des fonctionnalités de pause et de redémarrage, gérées par l'état booléen d'une variable "actif". Le TimerListener est utilisé pour pouvoir observer et réagir aux mises à jour du temps.

Problem et ses classes filles

Cette classe est utilisée afin de générer les problèmes à partir d'un fichier texte situé dans le dossier texteproblem qui contient pour chaque ligne, le numéro du problème, les IDs des cartes critères requis pour le problème et le code réponse. Cette partie est constituée par une classe mere abstraite problem et 3 sous classes qui gèrent l'affichage des problèmes avec diffèrent nombres de cartes critères. Le numéro de problème (la ligne dans le fichier texte à récupérer) est généré grâce à un générateur de nombre random. Cela permet à l'utilisateur d'avoir de nouveaux problèmes à chaque nouvelle partie. Les IDs et le code réponse sont ensuite récupérés et stockés dans des maps séparés pour les prochaines utilisations. Dans les 3 sous-classes, le fonctionnement est presque pareil, on s'adapte juste à la taille du probleme. De plus avec ces sous-classes nous avons introduit des boutons radios pour permettre la sélection des cartes et pour envoyer l'ID de la carte au vérificateur quand celui si en a besoin.

Vérificateur

Cette classe sert d'intermédiaire entre la partie et les cartes critères. Elle possède deux fonctions de vérification. On aurait pu les regrouper en une seule, mais de cette manière, la modularité est augmentée. La première vérifie le code proposé à partir d'un ID de carte et la seconde avec la carte elle-même. Pour la vérification à proprement parlé, on récupère simplement le critères des cartes et on appelle leurs méthodes de vérification présentes dans la classe, un switch selon l'ID qui permet d'avoir un critère différent pour chaque ID.

III. Rôle et avis de chacun

Dans cette partie, nous allons chacun exposer nos taches effectuées ainsi que donner nos avis sur le projet.

Matthieu :

Rôle :

Implémentation du système des cartes (Classe Critère, CarteCritere et système de BaseDeCartes), du menu du jeu avec la classe MenuJeu et règle, l'implémentation d'un timer, du système de joueur (groupe de joueurs et joueur) et beaucoup de la classe partie. En plus d'avoir aidé pour la vérification et fait pas mal de rapports.

Avis :

Ce projet a été très enrichissant. C'était la première fois que je faisais du Java et que j'implémentais une interface graphique. Le projet était très intéressant. Implémenter un jeu de société dans sa version numérique permettait de bien comprendre l'objectif final du projet et aidait à mieux appréhender l'aspect orienté objet du projet.

Amir :

Rôle :

Implémentation de la classe problem et les classes filles pour générer le problème du jeu.

Aidé à améliorer l'interface de partie.

Création de la vidéo de 5 minutes pour présenter le jeu.

Fait le rapport.

Avis :

Ce projet était très intéressant à faire et m'a donné une expérience globale de travailler en équipe pour réaliser un projet. J'ai pu enrichir ma connaissance de java et le pratiquer directement sur le code.

Theo :

Rôle :

Implémentation de la méthode pour vérifier les différents critères (Vérificateur) ainsi que l'interface du bouton vérifier. J'ai reçu de l'aide sur cette méthode. Implémentation des critères ainsi que de la modification des cartes critères (BaseDeCarte, CarteCritère et Critère) pour les problèmes à 4, 5 et 6 cartes.

Avis :

Ce projet m'a permis d'apprendre la gestion du travail dans un groupe. De sectionner un problème pouvant sembler difficile, comme faire un jeu de société, en plusieurs problèmes plus facile à aborder qui par la suite formeront notre jeu. Le projet représentait quelque chose de concret i.e. implémenter un jeu physique en un jeu virtuel et donc cela était très intéressant à expérimenter

IV. Amélioration possible

Nous pensons que notre jeu à divers axes d'amélioration et de nouvelles fonctionnalités pourrait améliorer l'expérience de l'utilisateur.

Au niveau du code, on pourrait améliorer la création des cartes. Actuellement, toutes les cartes sont créées lorsque le jeu est lancé. On pourrait trouver un système pour créer seulement les cartes voulues ou bien les récupérer à partir de fichiers texte. De plus, il aurait été bien de diviser le projet en plusieurs packages même si le projet n'est pas très complexe.

En termes de fonctionnalités, on aurait aimé ajouter la gestion d'indices via le *timer*. Au bout d'un certain temps, si le code n'est pas trouvé, on affiche un indice à l'utilisateur. Mais aussi, nous aurions voulu, avec plus de temps, afficher l'historique des codes proposés ou encore faire un système de classement entre les joueurs via leurs pseudos sur toutes les parties. Et bien sûr ajouter de nouvelles cartes et de nouveaux problèmes.

Enfin, de manière plus complexe, faire le jeu en ligne où deux ou plusieurs vrais utilisateurs s'affrontent.

Conclusion générale

Par rapport à la conception, en comparant nos diagrammes d'avant et après implémentation, nous pensons que notre structure générale a très peu changé, même si le contenu de nos classes dans le diagramme des classes a un peu évolué. Nous pensons surtout que ce dernier a pas mal évolué en raison du fait que c'était la première fois que nous réalisions un projet avec une interface graphique et que nous ne savions pas comment gérer les composants. Il y a également eu des oublis ou un manque de réflexion, mais nous sommes tout de même satisfaits de notre phase de conception, qui nous a beaucoup aidée dans la phase d'implémentation.

Et pour le projet en général, nous sommes satisfaits d'avoir obtenu un jeu fini et fonctionnel avec quelques petites options supplémentaires par rapport au jeu de base. Ce projet a été très enrichissant pour nous tous. Nous avons appris à réaliser une phase de conception, à utiliser Java et à implémenter des interfaces graphiques, en plus d'avoir renforcé nos compétences en travail d'équipe.

Sources

- https://turingmachine.info/
- https://www.scorpionmasque.com/sites/scorpionmasque.com/files/tm_rules_fr_24nov2022.pdf
- Cours d'AP4B