Rapport de conception AP4B
Projet Turing Machine

The TUTU Project

William IMBERT Victor KRETZ Mattéo POURCINE Noé SENGEL

8 décembre 2023

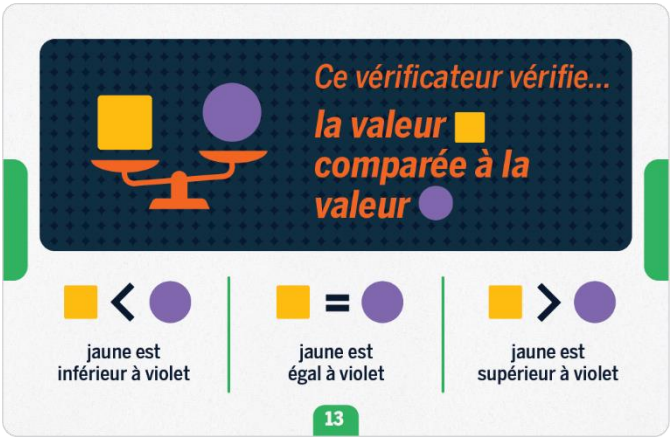
Table des matières

Introduction et but du projet

Dans le cadre de l'UE AP4B, nous avons réalisé un projet mettant en application les notions d'UML et le langage Java. Parmi les deux sujets proposés, nous avons choisi le projet « Turing Machine », un jeu de déduction où le ou les joueurs doivent interroger un proto-ordinateur afin de trouver un code secret. Le but du projet était donc de réaliser une version numérique du jeu de société s'inspirant du monde de l'UTBM. Le cahier des charges indiquait également qu'il était important de rester aussi proche du jeu que possible, tout en s'adaptant réellement au format numérique.

1. Fonctionnement du jeu

Le jeu Turing Machine, malgré l'apparente simplicité de ses règles, s'avère assez difficile à jouer. Dans une partie type, 1 à 4 joueurs tentent de déchiffrer un code secret en interrogant un proto-ordinateur à l'aide de cartes perforées. A chaque tour, les joueurs choisissent une sélection de trois cartes perforées ayant chacune un chiffre compris entre 1 et 5. Les joueurs vont ensuite interroger la machine à partir de ces trois chiffres à l'aide de cartes critères.



Carte critère typique

En effet, chaque « problème » comprend, en plus du code à deviner, 4 à 6 cartes critères parmi 48 disponibles avec le jeu. A chaque tour, les joueurs peuvent choisir jusqu'à trois cartes critère sur lesquelles tester leur sélection de cartes perforées. Pour chaque carte critère, un seul et même critère (fixé pour le problème sélectionné) est testé. En utilisant différents sets de cartes perforées, les joueurs pourront déterminer quel critère est le critère testé par la carte critère. Dans l'exemple ci-dessus, si le joueur propose un set de cartes perforées avec « jaune > violet » et que la carte critère retourne « vrai », cela voudra dire que le critère teste est bel et bien le critère « jaune > violet ». On saura alors que la valeur jaune du code secret est supérieure au violet.

Quand un joueur aura trouvé le critère testé pour chaque carte critère, il pourra ainsi déduire le code secret : en effet, il n'y a qu'un code secret respectant l'intégralité des critères d'un problème. Après chaque tour chaque joueur met son point au centre, avec le pouce vers bas s'il n'a pas trouvé le code et le pouce vers le haut s'il pense l'avoir trouvé. Si le joueur a bel et bien trouvé le code, il remporte la partie. S'il s'est trompé, il est éliminé et la partie continue. Si deux joueurs ont trouvé le code, le joueur ayant posé le moins de questions à la machine remporte la partie.

Il existe trois difficultés de problèmes (facile, standard, difficile), régulés par la sélection de cartes critère. On remarque notamment que les cartes critère des problèmes de niveau difficile ont une majorité de critères non mutuellement exclusifs. En d'autres termes, pour ces cartes allant de 26 à 48, savoir qu'un critère n'est pas valide ne permettra pas de déduire que ce critère ne s'applique pas au code secret, mais simplement qu'il n'est pas testé par la carte critère. Cela rend les problèmes difficiles très perturbants pour les nouveaux joueurs et il est facile de tirer de mauvaises conclusions à cause d'une mécompréhension du fonctionnement du jeu.

Bien que nous ayons tenté d'exposer brièvement les règles et le fonctionnement du jeu dans cette partie, il est difficile de montrer l'étendu de sa richesse en termes de stratégies. Il peut être nécessaire de voir des parties jouées pour comprendre réellement de quoi il s'agit. Ainsi, pour plus d'informations vous pouvez vous référer aux différentes ressources qui nous ont aidé à comprendre en détail les mécaniques et stratégies plus avancées de Turing Machine.

2. Choix de conception

a. Réflexion sur l'adaptation du jeu au format numérique

Après nous être imprégnés du fonctionnement du jeu, la première étape de conception du projet a été de définir la manière dont le jeu doit être adapté au format numérique. En effet, celui-ci a été conçu uniquement comme un jeu de plateau, demandant des solutions ingénieuses pour implémenter certains mécanismes comme celui de la vérification des cartes critères. En effet, pour tester un set de cartes perforées, les joueurs doivent superposer ces trois dernières et les places sur une carte « vérificateur ». La superposition des cartes fera ressortir une croix rouge un une marque verte indiquant le résultat du test. Il s'agit d'une mécanique que nous n'avons pas eu à utiliser dans notre implémentation. En effet, notre version étant au format numérique, il est possible de tester ces critères de manière beaucoup plus simple en les considérant comme des conditions booléennes.

Nous avons également eu à nous pencher sur la manière de gérer les cartes critères, qui nous a donné du fil à retordre. Lors de la conception du projet, nous n'avions initialement

considéré que les cartes dont les conditions étaient mutuellement exclusives. Cela a grandement influencé notre vision de la vérification des cartes car nous avons construit notre réflexion autour du postulat que le résultat donné par une carte critère pouvait être déterminé uniquement à partir du code secret et du code testé. Cela est effectivement vrai pour les premières cartes mais ne fonctionne plus pour les cartes non mutuellement exclusives qui impliquent que le critère testé soit prédéfini avec le problème. Cela nous a donc poussé à revoir notre architecture sur les cartes critère (voir diagramme de classes).

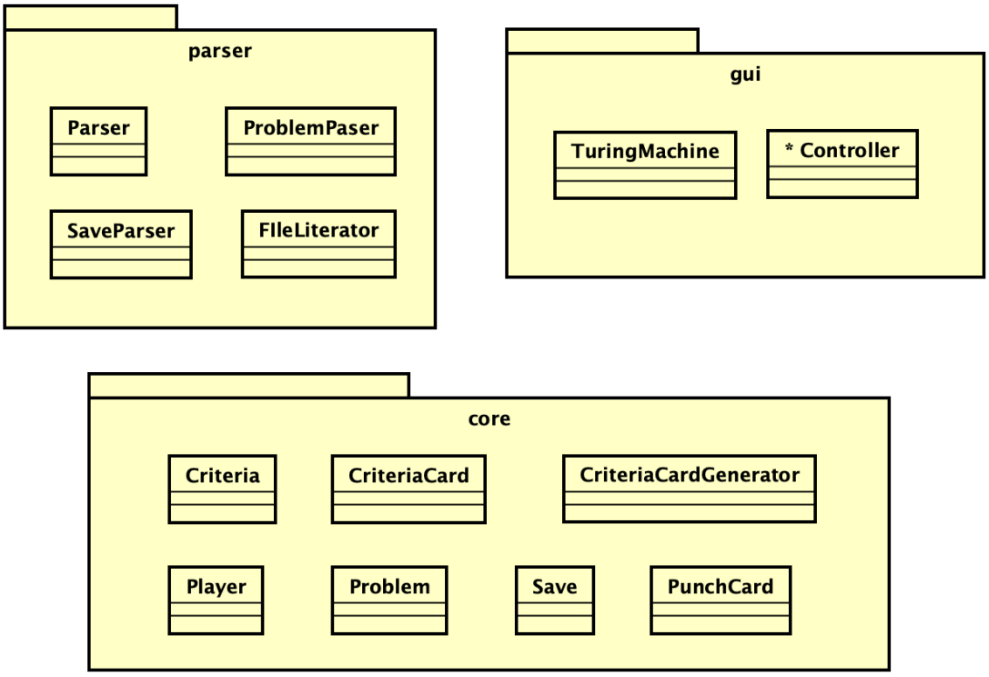
b. Adaptation à l'univers de l'UTBM

Le jeu Turing Machine étant très abstrait, il nous a semblé difficile de trouver une adaptation dans le gameplay du jeu à l'univers de l'UTBM. Pour satisfaire le cahier des charges, nous avons décidé de créer un scénario s'inspirant du monde de l'UTBM :

Nous jouons le rôle d'un élève de l'UTBM qui souhaite dérober des copies de finaux dans le but d'assurer son semestre. On arrive devant l'UTBM (menu du jeu) après un passage à l'ordinateur pour identification, on accède alors aux bureaux de trois professeurs. Une fois devant les 3 portes, c'est à nous de choisir quel bureau braquer. Dans chaque bureau se trouve un coffre contenant les sujets de finaux, ce coffre a 3 chiffres est notre défi, le dernier qui nous sépare des tant convoitées copies.

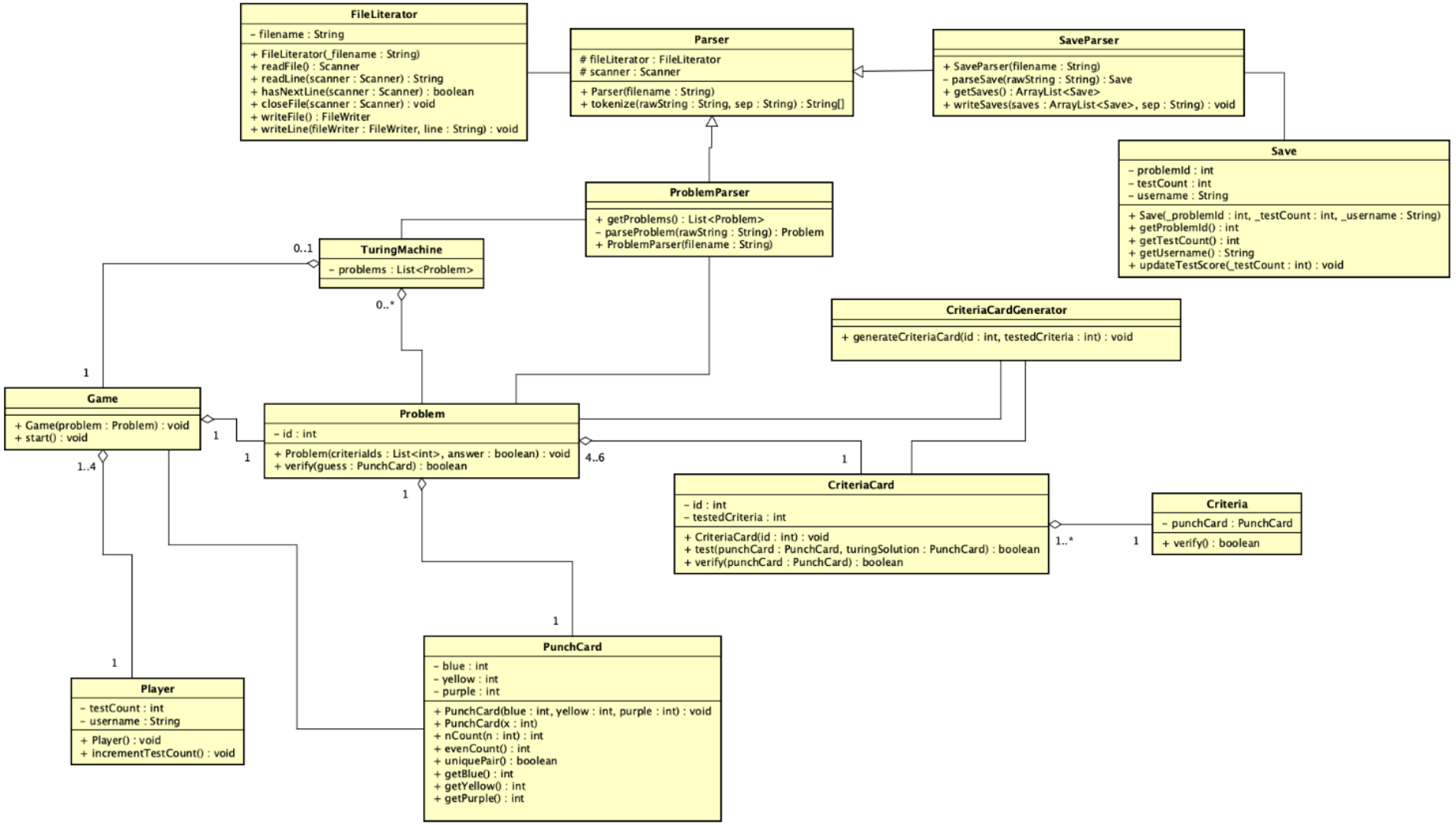
3. Diagrammes de classes

a. Packages



Le diagramme de classe ci-dessus illustre la structure organisationnelle de notre système en utilisant une hiérarchie de packages, le tout en étant plus visuel que si nous avions tout mis sur le diagramme de classe.
Le premier package, "core", est dédié au cœur fonctionnel de la machine de Turing, implémentant ses algorithmes fondamentaux. Le second, "gui", s'occupe de l'interface graphique, rendant le système visuellement attractif et surtout bien plus ergonomique qu'une simple console. Enfin, le package "parser" sert à l'analyse des problèmes de la machine de Turing et contient FileLiterator une classe File utile pour analyser justemment.
Nous détaillerons les classes contenues dans chaque package dans le diagramme de classe, sauf celle de la classe gui car elle est encore en développement.
Grâce à ce diagramme, on peut mieux appréhender l'organisation modulaire de notre système et comment chaque niveau contribue de manière spécifique au bon fonctionnement des domaines qui lui sont associés.

b. Diagramme de classes

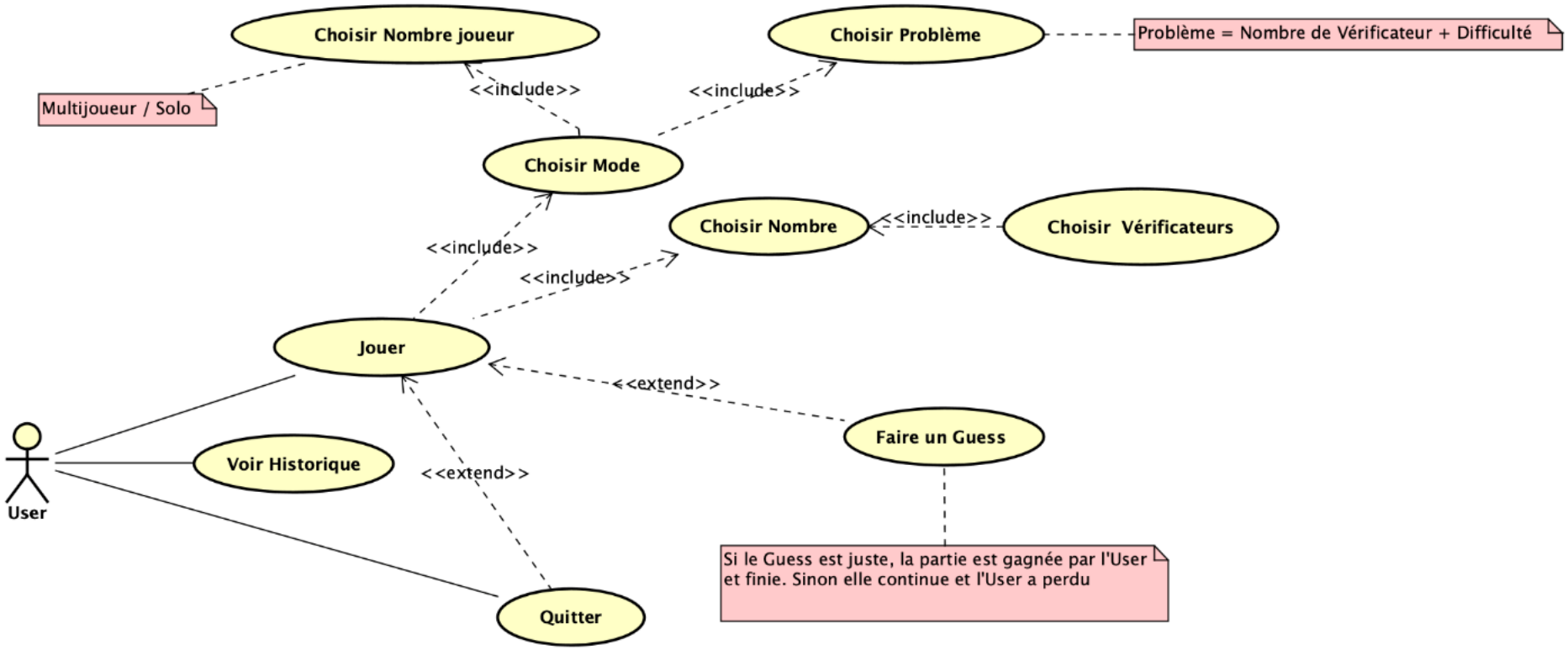


La classe *TuringMachine* peut être considérée comme la classe centrale du projet. C'est cette classe qui va se charger de mettre en relation les parties *model, view* et *controller* du programme. On retrouve sur le diagramme toute la partie *model*, contenant la logique du jeu à proprement parler. La liste des problèmes est parsée à l'aide de classes *ProblemParser* qui fera ensuite appel aux autres classes de parsing. Cela permet de passer d'un fichier texte contenant une liste de problèmes à une liste de problèmes dans le model. Ces problèmes comprennent la liste des critères qui leur sont associés (avec l'index du critère testé) ainsi que l'id du problème et la solution.

On voit également sur le diagramme qu'une sélection de cartes perforées est représentée par la classe *punchCard*. Les cartes critères sont représentées par une liste de *Criteria* qui contiennent la condition booléenne du critère. Pour générer les cartes critères du jeu, on a une classe utilitaire *CriteriaCardGenerator* permettant de retourner à la demande une des 48 cartes critères du jeu.

Les classes contrôleur étant nombreuses (une par page de l'UI), nous ne les avons pas représentées dans le diagramme. On note cependant qu'elles permettent de gérer les entrées utilisateur afin de communiquer avec la classe *Game* qui contient tout l'état de la partie en cours.

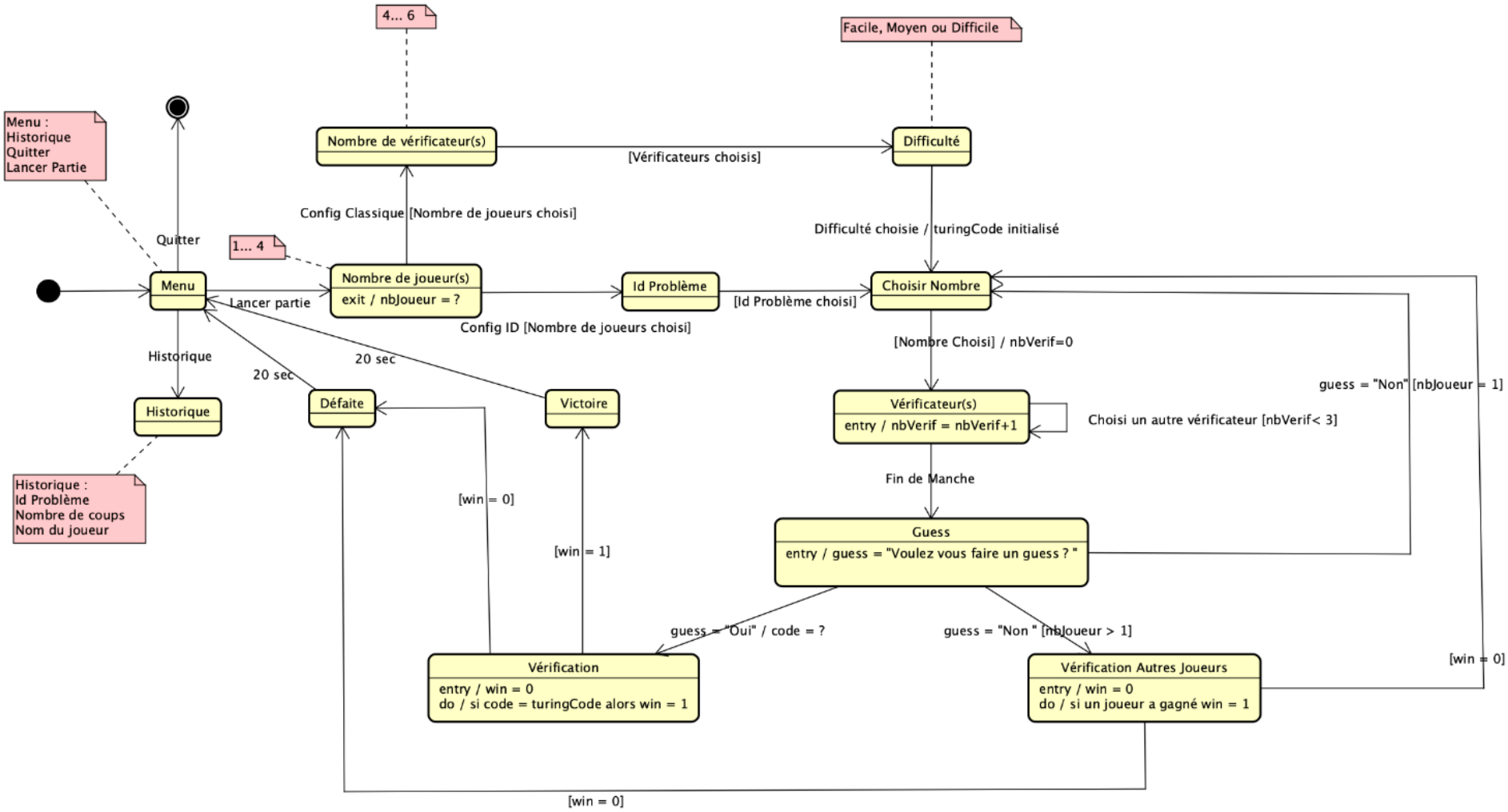
4. Diagramme de cas d'utilisation



Le diagramme de cas d'utilisation représente l'interaction entre l'acteur unique, "User", et notre jeu. L'acteur a la possibilité de jouer, quitter le jeu, ou consulter l'historique.
Le cas d'utilisation principal, "Jouer", encapsule le processus global de jeu, incluant les étapes de choisir le mode de jeu impliquant le choix de joueur et du problème et de choisir les vérificateurs à tester impliquant sélectionner un nombre en amont.
L'extension "Quitter" permet à l'utilisateur de quitter le jeu à n'importe quel moment pendant le processus de jeu. De même, l'extension "Faire un Guess" offre la possibilité à l'utilisateur de faire une supposition pendant le déroulement du jeu.
Ce diagramme illustre de manière concise les actions principales que l'utilisateur peut entreprendre et comment ces actions interagissent dans le contexte du jeu.

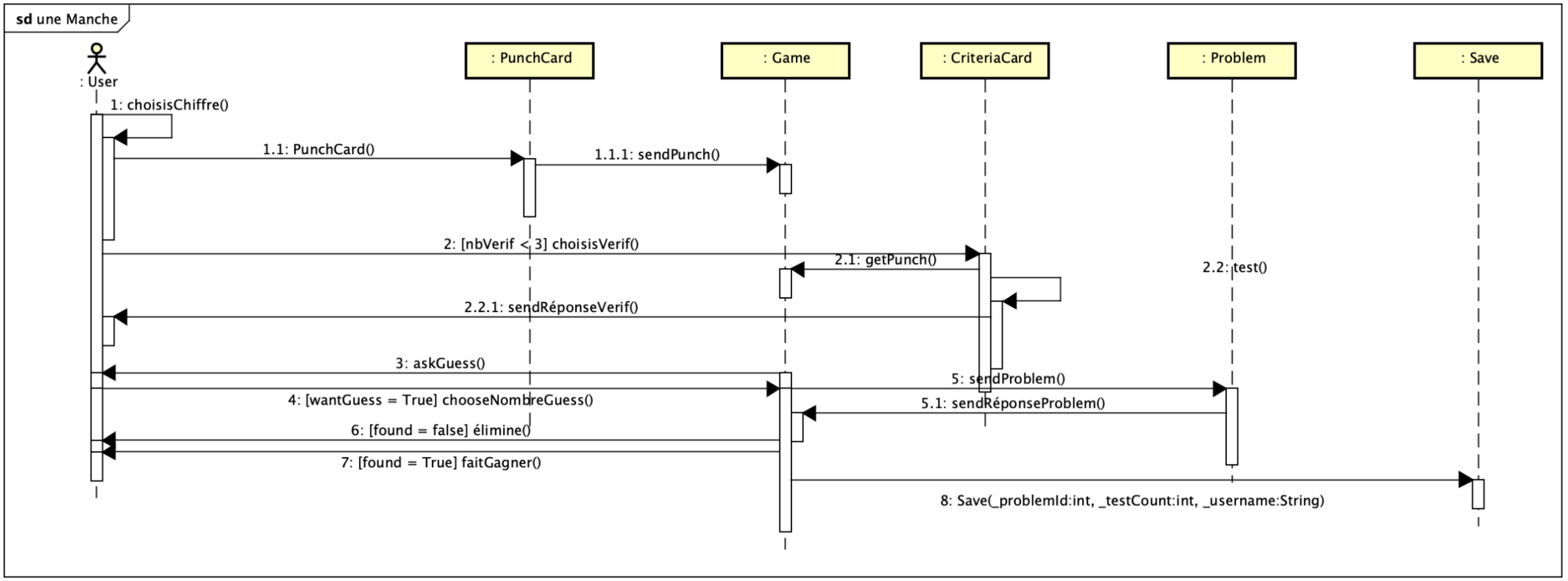
Cependant, il est difficile de représenter l'entièreté d'un système car on ne sait pas vraiment comment s'enchaîne les cas d'utilisations. En effet il manque l'aspect temporel c'est pourquoi il nous faut réaliser d'autres diagramme pour comprendre au mieux notre système.

5. Diagramme d'états-transitions



Ce diagramme d'état-transition offre une représentation visuelle claire et structurée du comportement dynamique de notre système. On peut ainsi facilement comprendre les différents états que l'utilisateur peut occuper pendant le jeu en plus des transitions entre ces états.
Par exemple notre diagramme se compose de deux phases principales, la première ou l'user paramètre la partie représentée par les états Nombre de joueurs, Nombre de vérificateurs et Difficulté ou Id du problème directement pour accéder aux problèmes en question. Ces états sont liés par le choix du paramétrage précédent.
On passe ensuite à la phase de jeu ou l'utilisateur choisis son nombre à 3 chiffres (état Choisir Nombre), une fois ce nombre choisi on passe au choix des différents vérificateurs à tester (au nombre de 3 maximum d'où la condition pour continuer d'en choisir).
Ensuite la manche se fini et on demande au joueur s'il veut tenter de faire un Guess de code, si non il doit attendre que les autres joueurs fassent de même puis il retourne au début de la manche, sauf si un joueur a trouvé le bon code (état Vérification autres joueurs).
S'il veut en revanche faire un Guess il ira dans l'état Vérification ou son Guess sera récupéré puis l'utilisateur sera jugé en conséquence : soit éliminé, soit victorieux (respectivement états Défaite et Victoire).
Enfin, une fois jugé, il sera renvoyé à l'état initial : Menu, ou il peut soit quitter soit aller voir l'historique.
Cette analyse approfondie permet donc non seulement de déchiffrer les interactions entre les états, mais également d'identifier les éléments et conditions déclenchant ces transitions.

6. Diagramme de séquence



Le diagramme de séquence ci-dessus offre une représentation détaillée des interactions entre les différents éléments d'un système au fil du temps, les éléments étant ici l'utilisateur avec User et les instances des classes provenant du diagramme de classe présenté en amont.
Nous avons choisi de représenter uniquement le déroulement d'une manche de jeu, c'est à dire la séquence allant de "Choisir Chiffre" aux Vérifications si on se réfère au diagramme d'états-transitions car c'est la seule nécessitant vraiment réflexion. En effet, le nom de la séquence paramétrage de partie se suffit lui-même pour savoir ce qu'il s'y passe, idem pour la séquence qui concerne la consultation d'historique.
Parlons du jeu donc, l'entité principale est assez logiquement de la classe Game qui vient centraliser les informations provenant de l'utilisateur : le nombre utilisé pour tester les vérificateurs provient de la classe PunchCard, les vérificateurs de CriteriaCard, le problème de Problem et la sauvegarde de Save.
Ce qui se passe entre les instances de ces classes est assez logique, surtout après avoir vu le diagramme précédent sur les différents états. Nous avons un peu peiné à représenter les liens entres les instances car le diagramme de séquence est loin d'être évident.
Cependant, nous tenions à nous y frotter et avons fait de notre mieux pour que ce soit cohérent. Ce diagramme capture donc de manière séquentielle les échanges de messages entre les objets ou acteurs du système, fournissant ainsi un aperçu dynamique du "flux" d'exécution.
En regardant ce diagramme, on peut discerner l'ordre chronologique des activités, les classes impliqués, ainsi que les messages échangés entre eux. Chaque lifeline représente l'évolution temporelle d'un acteur ou objet spécifique, tandis que les messages, illustrés par des flèches, indiquent les interactions entre ces entités.

Sources

Compréhension du jeu

Site du jeu : https://turingmachine.info/

Règles du jeu : https://turingmachine.info/files/rules/rules_FR.pdf

Astuces avancées : https://turingmachine.info/files/rules/TuringMachine_ParadoxeFaux_FR.pdf

Stratégies de jeu avancées : https://www.youtube.com/watch?v=0tndiaAxWu4&t=3104s