Rapport AP4B

Développement d'une application JavaFX en se focalisant sur la partie conception et programmation du coeur de l'application.

Antoine Laurant, Arnaud Michel, Antoine Perrin

Table des matières

Présentation

Projet Turing Machine - UTBM

Introduction

Ce projet rend hommage à Alan Turing, un génie des mathématiques et du cryptage qui a contribué à l'invention des ordinateurs. Inspiré du jeu de société Turing Machine, notre version numérique propose une adaptation unique et ludique pour les étudiants de l'UTBM.

Objectif du Projet

L'objectif de ce projet est de créer une version numérique du jeu Turing Machine, en s'inspirant du monde de l' **UTBM**. Dans cette adaptation, un **professeur** envoie un mail à ses élèves pour leur donner une **énigme** afin qu'ils puissent rentrer dans la salle d'examen grâce à un **mot de passe**. Les étudiants doivent proposer des mots de passe et l'application leur indique si les mots de passe proposés valident les critères de l'énigme. Pour éviter toute triche et pour rendre le jeu plus **stimulant**, les mots de passes seront à usage unique et généré en fonction de l'INE de l'étudiant. De plus lors du lancement de l'application, une difficulté est choisie par l'étudiant, ce qui influe sur la **complexité** de l'énigme et pour favoriser la prise de risque, un **multiplicateur** de points (variant de 1 à 1.8) sera appliqué sur la note de l'examen en fonction de la difficulté choisie.

Il est également possible de jouer à plusieurs, en **compétition** ou le **premier** qui trouve le mot de passe valide gagne. Dans ce cas l'INE n'est pas nécessaire.

Technologies Utilisées

Pour réaliser ce projet, nous avons utilisé les technologies suivantes :

- **Java**: pour la logique de programmation.
- **JavaFX**: pour l'interface utilisateur.
- **MVC (Model-View-Controller)**: pour structurer le code de manière modulaire et maintenable.

Fonctionnalités du Jeu

- **Connexion**: l'étudiant doit se connecter avec son INE pour accéder à l'énigme.
- **Choix de la difficulté**: l'étudiant peut choisir la difficulté de l'énigme.
- **Génération de mot de passe**: l'application génère un mot de passe unique en fonction de l'INE de l'étudiant.
- **Validation du mot de passe**: l'application valide le mot de passe proposé par l'étudiant.

Choix de conceptions

Afin de réaliser notre application, nous avons dû faire des choix de conception. Ces choix ont été faits en fonction des besoins de l'application et des contraintes techniques.

Model view controller

Nous avons choisi d'utiliser le **modèle MVC** pour la conception de notre application. En effet, cette architecture permet de séparer les différentes composantes de l'application, ce qui facilite la maintenance et l'évolution de l'application. De plus le modèle MVC permet de rendre l'application plus modulaire et donc plus facile à comprendre et à maintenir.

Création d'une unique solution

Pour créer une solution unique en fonctions de différents critères, nous avons choisi de modéliser les différentes étapes de la création de la solution grâce à la **composition de fonctions**.

Mathématiquement, nous créons un tableau de **125** éléments, chaque élément étant une **combinaison possible**. Nous appliquons ensuite une série de fonctions pour **filtrer** les combinaisons en fonction des critères choisis par l'utilisateur. Et nous continuons ce processus jusqu'à ce qu'il ne reste qu' **une** seule combinaison. Si au bout de 6 étapes, s'il reste plusieurs combinaisons, nous revenons à l'étape précédente et recommence le processus (donc, de manière récursive). Ce processus est développé dans un diagramme de séquence disponible **ici (Initialisation de la partie)**.

Modélisation mathématique du processus :

Initialisation :
$\Omega = \{(x_1, x_2, x_3) \mid x_i \in \{1,2,3,4,5\} \text{ pour } i = 1,2,3\}$
(ensemble de toutes les combinaisons possibles, taille 125)

Processus de filtrage :
$F = f_1 \circ f_2 \circ f_3 \circ f_4 \circ f_5 \circ f_6 : \Omega \to \mathcal{P}(\Omega)$
avec f_k : fonctions de filtrage appliquées séquentiellement

Résultat après chaque étape :
$\Omega_k = f_k(\Omega_{k-1}) \quad \text{avec } \Omega_0 = \Omega$
Si $|\Omega_k| = 1$ (une seule combinaison restante), le processus s'arrête.
Si $|\Omega_k| \neq 1$ (il reste plusieurs combinaisons) ou $k = 6$ (6 étapes atteintes), alors :
Retour à l'étape précédente avec une nouvelle fonction de filtrage.

Utilisation de JavaFX

Nous avons choisi d'utiliser **JavaFX** pour la conception de l'interface graphique de notre application. JavaFX est une technologie qui permet de créer des interfaces graphiques de manière simple et efficace. Par ailleurs, c'est une technologie qui est intégrée à Java, ce qui facilite son utilisation.

Diagramme de Classe

Diagrammes Use Case

Diagrammes de Séquence

Initialisation de la partie

Description de la manche

Description

La manche est une partie du jeu. Elle est composée de plusieurs éléments :

- Un **Round**: le model de la manche.
- Un **RoundController**: le contrôleur de la manche.
- Un **RoundView**: la vue de la manche.

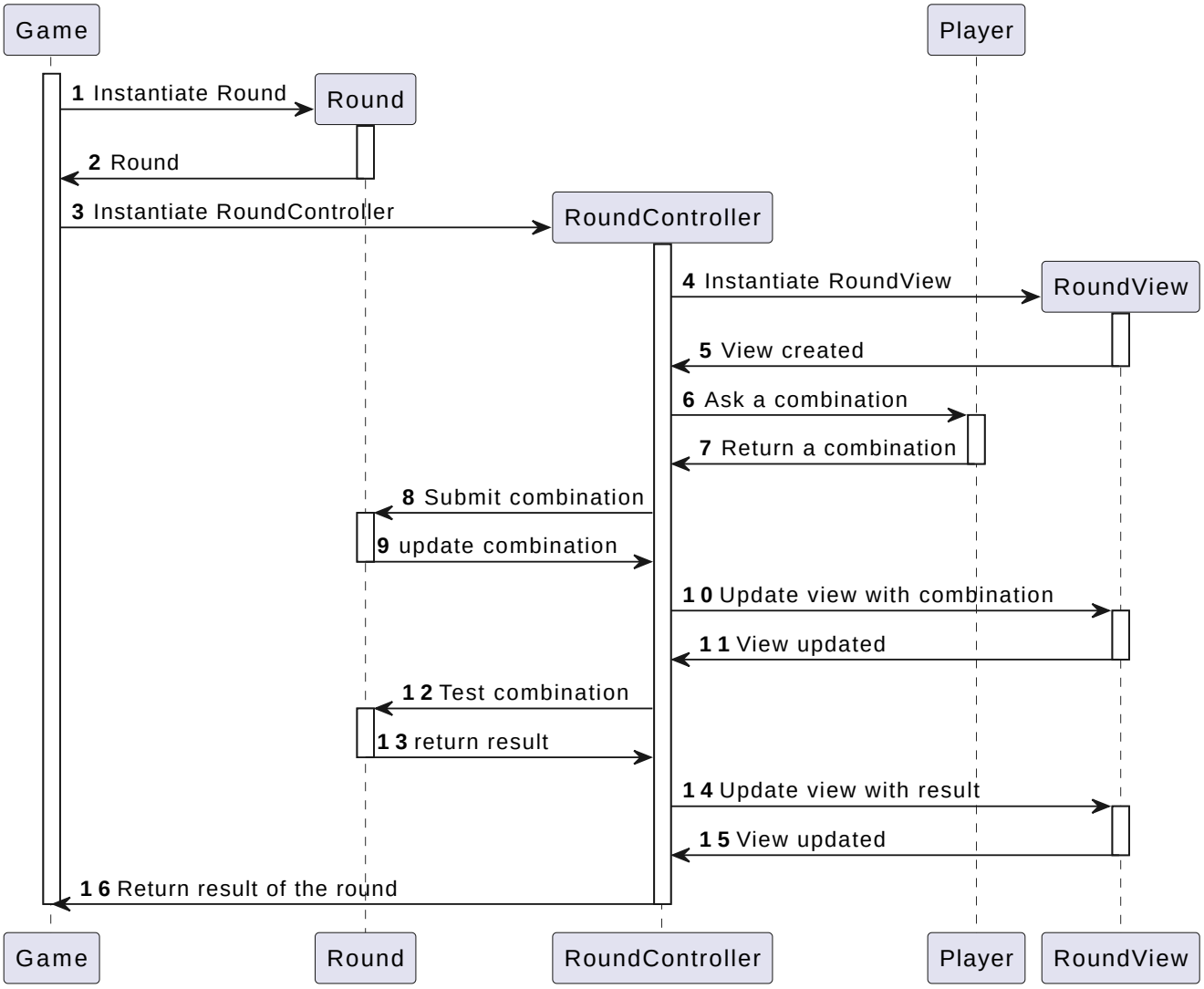
A chaque manche, un joueur doit proposer une combinaison. Cette combinaison est testée et le résultat est affiché à l'écran. Puis, le résultat de la manche est retourné au jeu.

Exemple

Un joueur propose la combinaison `123`. Pour un critère nombre de `1` le test renvoie `validé` (true), pour le critère le nombre d'index `3` est le plus grand, le test renvoie `non validé` (false), et ainsi de suite pour le nombre de critères de la partie. Enfin, le résultat de la manche est retourné au jeu.

Diagramme de séquence

Manche


Game
Player
1 Instantiate Round
Round
2 Round
3 Instantiate RoundController
RoundController
4 Instantiate RoundView
RoundView
5 View created
6 Ask a combination
7 Return a combination
8 Submit combination
9 update combination
10 Update view with combination
11 View updated
12 Test combination
13 return result
14 Update view with result
15 View updated
16 Return result of the round
Game
Round
RoundController
Player
RoundView

Conclusion