Cahier des Charges

*Money Flop quiz*

Table des matières

[1. Contexte 2](#_Toc190883113)

[2. Fonctionnalités 3](#_Toc190883114)

[Gestion des données 4](#_Toc190883115)

[3. Spécifications fonctionnelles 5](#_Toc190883116)

[5. Interfaces graphiques 6](#_Toc190883117)

[Conclusion et Perspectives 10](#_Toc190883118)

[7. Retour d'expérience et amélioration du cours 10](#_Toc190883119)

Groupe numéro 7 :

Théo Fillon,

Bastian Fontaine-Tranchant,

Lucas Corompt

## 1. ****Contexte****

Ce projet consiste à créer une application de jeu télévisé interactif basée sur des quiz, inspirée de *Money Drop*. L'objectif est de permettre à un joueur de répondre à une série de questions à choix multiples tout en gérant un solde d'argent qu'il peut répartir sur les différentes réponses. L'application doit être réalisée en Java et inclure deux interfaces graphiques distinctes, l'une pour l'introduction et la roue de la fortune, et l'autre pour le quiz et le calcul des mises et gains.

L'application se compose de deux grandes parties :

1. **Interface d'accueil** : Le joueur entre son pseudo et fait tourner une roue qui lui attribue une somme d'argent de départ.
2. **Interface explicative :** Le joueur prend connaissance des règles et démarre le jeu.
3. **Interface de jeu** : Le joueur répond à 10 questions avec des choix de réponses, répartit son solde d'argent sur les différentes réponses et voit son solde mis à jour après chaque question.

## ****2. Fonctionnalités****

#### **2.1 Interface d'accueil (Page d'accueil)**

* Affichage d'un message de bienvenue.
* Champ pour entrer un pseudo.
* Une roue de la fortune qui génère une somme d'argent aléatoire pour le joueur.
* Un bouton "Jouer" pour valider le pseudo et passer à l’interface explicative.

#### **2.2 Interface explicative**

* Affichage d'un message de bienvenue
* Affichage des règles du jeu
* Un bouton pour lancer le jeu

#### **2.3 Interface de jeu**

* Affichage de la première question.
* Un timer de 2 minutes qui se réinitialise à chaque question.
* 4 réponses possibles sous la question, chacune ayant un champ où le joueur peut saisir une somme d'argent à miser.
* Affichage du pseudo du joueur en haut à gauche de l'écran.
* Affichage du solde du joueur
* Un bouton "Valider" pour confirmer la réponse et la mise.
* Un indicateur du numéro de la question (1/10).
* Calcul des gains en doublant la mise sur la bonne réponse et en perdant la mise sur les mauvaises réponses.

#### **2.4 Gestion des mises et gains**

* Lors de chaque question, le joueur peut répartir son solde sur les réponses proposées.
* Si la réponse est correcte, le joueur double la somme qu'il a misée sur la bonne réponse.
* Si la réponse est incorrecte, le joueur perd l'argent misé sur les mauvaises réponses.
* Le jeu comporte 10 questions, et le but est de finir avec le plus d'argent possible.

#### **2.5 Fin du jeu**

* À la fin des 10 questions, un message d'achèvement
* Affichage de la solde final du joueur.

## ****3.**** Conception UML

**Une image contenant texte, diagramme, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

#### 1. Fenetre

**Rôle :** Gère l'identité du joueur en stockant son pseudo.  
**Attributs :**

* pseudo : String → Stocke le pseudo du joueur.  
  **Méthodes principales :**
* setPseudo(String) : void → Définit le pseudo du joueur.
* getPseudo() : String → Récupère le pseudo du joueur.

#### 2. FenetreResultat

**Rôle :** Gère l'affichage des résultats et le suivi du quiz.  
**Attributs :**

* questionNumber : int → Numéro de la question actuelle.
* totalBalance : int → Score ou solde total du joueur.
* correctAnswerIndex : int → Index de la réponse correcte.
* questions : List<Question> → Liste des questions du quiz.
* currentQuestionIndex : int → Index de la question en cours.
* availableIndices : LinkedList<Integer> → Indices disponibles pour les questions.
* timerManager : TimerManager → Gère le temps des questions.  
  **Méthodes principales :**
* getCorrectAnswerIndex() : int → Retourne l'index de la bonne réponse.
* validateWagers() : void → Valide les mises des joueurs.
* checkWagers() : void → Vérifie si les mises sont correctes.

#### 3. WheelPanel

**Rôle :** Gère le fonctionnement et l'affichage de la roue.  
**Attributs :**

* timer : Timer → Gère le temps d’animation de la roue.
* angle : int → Angle actuel de la roue.
* values : String[] → Valeurs possibles affichées sur la roue.
* decelerationTime : int → Temps de décélération de la roue.
* startTime : long → Temps de départ de la rotation.
* isSpinning : boolean → Indique si la roue est en train de tourner.
* valueToDisplay : String → Valeur affichée après l'arrêt de la roue.  
  **Méthodes principales :**
* generateValidAngle() : int → Génère un angle valide pour la roue.
* isValidAngle(int) : boolean → Vérifie si un angle est valide.
* startSpin() : int → Démarre la rotation de la roue.
* stopWheel() : void → Arrête la roue.
* getValueToDisplay() : String → Retourne la valeur affichée.
* getColor() : Color → Retourne la couleur correspondant à la valeur.

#### 4. TimerManager

**Rôle :** Gère le temps imparti pour répondre aux questions.  
**Attributs :**

* timer : Timer → Timer utilisé pour la gestion du temps.
* timeRemaining : int → Temps restant avant la fin.
* onTimeOut : Runnable → Action à exécuter quand le temps est écoulé.  
  **Méthodes principales :**
* startTimer() : void → Démarre le timer.
* resetTimer() : void → Réinitialise le timer.
* stopTimer() : void → Arrête le timer.

#### 5. Question

**Rôle :** Représente une question du quiz avec ses réponses possibles.  
**Attributs :**

* question : String → Texte de la question.
* propositions : String[] → Liste des réponses proposées.
* correctAnswer : String → Réponse correcte.
* anecdote : String → Information complémentaire sur la question.  
  **Méthodes principales :**
* Question() → Constructeur initialisant une question.
* getId() : long / setId() : void → Gère l’identifiant de la question.
* getQuestion() : String / setQuestion(String) : void → Accède ou modifie la question.
* getPropositions() : String[] / setPropositions(String[]) : void → Accède ou modifie les propositions.
* getCorrectAnswer() : String / setCorrectAnswer(String) : void → Accède ou modifie la bonne réponse.
* getAnecdote() : String / setAnecdote(String) : void → Accède ou modifie l’anecdote.

## ****4 Gestion** **des données****

#### Présentation des données utilisées

Les données utilisées dans notre application sont issues de fichiers JSON contenant des questions, leurs réponses et des anecdotes associées. Ces fichiers ont été téléchargés depuis **Open Quizz** et nous avons combiné plusieurs quiz pour créer une base de données plus variée et complète.

Le format des données est structuré en JSON, avec une organisation sous forme de liste d'objets, chaque objet représentant une question avec ses différentes réponses et une éventuelle anecdote.

#### Récupération et sauvegarde des données

Pour ce faire nous utilisons le fichier json-simple-1.1.1.jar qui nous permet de lire les fichier json.

Pour gérer ces données, nous utilisons une **liste** qui contient l’ensemble des questions chargées depuis le fichier JSON. L’application utilise la fonction **shuffle()** de la bibliothèque random pour mélanger les questions afin d’assurer une expérience variée à chaque session.

L’interaction avec les données se fait en retirant la première question de la liste après qu’elle ait été posée à l’utilisateur. Cela permet d'éviter la répétition immédiate des questions et de simuler un tirage aléatoire dynamique.

Concernant la récupération des données, notre application charge le fichier JSON au démarrage et le convertit en une liste exploitable en mémoire. Il n’y a pas de sauvegarde des données en temps réel, car les questions sont simplement utilisées en mémoire pour une session donnée.

## ****5. Spécifications fonctionnelles****

#### **3.1 Scénario de jeu**

1. **Lancement du jeu** :
   * Le joueur ouvre l'application.
   * Le pseudo du joueur est saisi.
   * La roue de la fortune tourne et attribue une somme d'argent aléatoire au joueur.
2. **Règle du jeu** :
   * Le joueur prend connaissance des règles du jeu.
   * Le joueur lance le jeu.
3. **Début de la première question** :
   * Le timer commence (2 minutes par question).
   * La question est affichée avec quatre réponses possibles.
   * Le joueur doit répartir son solde d'argent sur les réponses.
   * Le joueur valide la réponse et la mise.
4. **Après chaque question** :
   * Le solde est mis à jour en fonction de la réponse donnée.
   * Le joueur peut voir son solde, son pseudo, et l'indicateur de question actuelle (ex. 1/10).
   * Le timer se réinitialise pour la prochaine question.
5. **Fin du jeu** :
   * Après les 10 questions, un message affiche le solde final du joueur et les gains/pertes réalisés.

#### **3.2 Gestion des mises**

* Le joueur doit pouvoir saisir des montants différents pour chaque réponse.
* Le solde total doit être réparti sur les réponses choisies par le joueur.

#### **3.3 Chronomètre**

* Chaque question a un timer de 90 secondes.
* Le timer doit être réinitialisé à chaque question et visible en haut à droite de l'écran.
* À la fin du timer, si le joueur n'a pas validé sa réponse, la mise est annulée.

## ****6. Interfaces graphiques****

#### **5.1 Interface d'accueil**

Une image contenant capture d’écran, texte, Graphique, cercle

Le contenu généré par l’IA peut être incorrect.

* Un champ de texte pour entrer le pseudo du joueur.
* Une roue visuelle qui affiche la somme obtenue après rotation.
* Un bouton "Jouer" pour démarrer le jeu.

#### **5.2 Règle du jeu**

Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

* Un message de bienvenue.
* Les règles du jeu.
* Un bouton "Jouer" pour démarrer le jeu.

#### **5.3 Interface de jeuUne image contenant capture d’écran, texte, diagramme, conception Le contenu généré par l’IA peut être incorrect.**

* Une question au centre de l'écran.
* Un timer au milieu.
* Quatre réponses sous la question avec des champs de mise.
* la solde du joueur.
* Un bouton "Valider" sous les réponses pour soumettre les mises et la réponse.

#### **5.4 Interface de fin**

* En cas de victoire

#### **Une image contenant collage, capture d’écran Le contenu généré par l’IA peut être incorrect.**

* En cas de défaite

Une image contenant chien, mammifère, animal domestique

Le contenu généré par l’IA peut être incorrect.

## 7.Conclusion et Perspectives

Bilan du projet  
Le développement de Money Flop Quiz a permis de concevoir une application interactive alliant réflexion stratégique et gestion du risque. L’intégration de Java swing pour l’interface graphique a offert une expérience immersive et fluide, tout en assurant une gestion efficace des mises et des gains.

#### Difficultés rencontrées et solutions

* Gestion du fichier json : problème de synchronisation des questions.  
  Solution : implémentation d’une liste pour lire les questions de manière random à l’aide d’un shuffle
* Répartition des mises : éviter que le joueur ne mise plus que son solde.  
  Solution : vérifications et ajustements dynamiques avant validation.
* Mécanique de la roue de la fortune : difficulté à assurer un tirage fluide.  
  Solution : utilisation des angles préenregistrés pour garantir un arrêt précis.

#### Améliorations possibles

* Ajout d'un mode multijoueur pour des défis en ligne.
* Variété des questions avec un système de difficulté adaptatif.
* Effets sonores et visuels avancés pour renforcer l’immersion.
* Intégration d’une base de données pour sauvegarder les scores et suivre la progression des joueurs.

## 8. Retour d'expérience et amélioration du cours

Dans ce projet, nous avons appris à mieux structurer nos applications en découvrant des concepts comme Flat Laf et en approfondissant notre compréhension de l'importance de la modélisation pour mener à bien le projet.

La mise en œuvre de Flat Laf nous a permis de comprendre comment personnaliser l'interface utilisateur pour améliorer l’expérience visuelle. De plus, grâce à l’utilisation d’UML, nous avons vu à quel point il est crucial de bien modéliser une application avant de commencer à coder, ce qui nous aide à mieux organiser le projet, à anticiper les besoins et à améliorer la communication au sein de l'équipe.

Le projet a été à la fois pédagogique et ludique, ce qui a rendu l'apprentissage de la POO et de la COO particulièrement intéressant. En appliquant ces concepts dans un cadre concret, on a pu comprendre l’importance de structurer le code de manière modulaire et réutilisable. Cela nous a permis de saisir les avantages de l’encapsulation, de l’héritage et de la gestion des composants.

## 9. Assistance IA

https://chatgpt.com/share/67b61593-097c-8007-bbc6-a688b033796b