SAE 01.1 - Implémentation jeu "KiKéProche"

Ce document décrit le sujet de la SAE « *S1.01 - implémentation d'un besoin client* ». Le sujet se basera sur les contenus du TP10 (pour la gestion des tableaux avec des objets) et du TP11 (pour la gestion d'un paquet de cartes).

Consigne

- **Durée** : 12h de travail étudiant.
- **Format** : travail en binôme.
- **Rendus attendus** :
  - **Aprés 6h de SAÉ** :
    - une première version intermédiaire à déposer sur arche ;
    - avec vos classes `Carac` et `Carte`.
  - **Aprés 12h de SAÉ** :
    - un code commenté qui compile ;
    - des classes de test ;
    - un rapport qui présente votre réalisation.

Avant toute chose, nous vous conseillons de lire la section 7 décrivant le contenu attendu du rapport pour ne pas oublier des éléments lors de votre réalisation.

Consigne

N'oubliez pas de sauvegarder régulièrement votre projet (sauvegarde sur votre espace personnel, envoi par mail entre vous, utilisation d'outil de partage, ...) afin de ne pas perdre votre travail entre deux séances. Aucune perte de travail ne pourra justifier une absence de rendu de la SAÉ.

1 Présentation du jeu KiKéProche

L'objectif de la SAÉ va être de programmer le jeu « KiKéProche » inspiré des jeux « zéro à 100 » [1] et « Timeline » [2].

1. (édité par Le Scorpion Masqué - `https://www.scorpionmasque.com/fr/zero-a-100`)
2. (édité par Asmodée - `https://print-and-play.asmodee.fun/fr/timeline/`)

FIGURE 1 – Exemple de la carte grenouille (recto à gauche, verso à droite). Ici les caractéristiques sont la taille (8cm), le poids (0.02kg) et la longévité (8ans).

Le jeu « KiKéProche » est constitué de cartes représentant des animaux (mais tout autre type de carte est possible). Comme illustré figure 1, une des faces de la carte donne le nom de l'animal, l'autre face décrit différentes caractéristiques (par ex, son poids et sa taille).

Il s'agit d'un jeu **solitaire** qui se joue contre l'ordinateur. À chaque manche, le joueur tire un nombre de cartes au hasard dans sa main mais ne voit que les noms des animaux sans connaître leurs caractéristiques. Le jeu propose au joueur une caractéristique et une carte tirées au hasard (par exemple, la caractéristique poids et la carte animal `lama`) et le joueur doit sélectionner la carte de sa main dont la caractéristique demandée est la plus proche de celle proposée par le jeu (dans cet exemple, la carte correspondant à l'animal dont le poids est le plus proche du `lama`).

Le jeu montre les caractéristiques de la carte choisie par le joueur et vérifie si la carte choisie est la carte de sa main la plus proche de la carte proposée par le jeu (sur cet exemple, si le poids de l'animal choisi est effectivement le poids le plus proche du `lama`) :

- Si c'est le cas, le joueur remporte la manche et la carte est alors retirée de la main du joueur ;
- sinon, le joueur perd la manche. La carte choisie par le joueur et la carte correspondant à la bonne réponse attendue sont défaussées toutes les deux et le joueur en pioche deux nouvelles dans la pioche pour les remplacer.

Dans l'exemple de la figure 2,

- Si le joueur choisit la `marmotte` qui fait 5kg, comme il possède dans son jeu la carte `tamanoir` dont le poids de 42kg est plus proche de celui du `lama`, il perd la manche, défausse la carte `marmotte` et la carte `tamanoir` et en pioche deux nouvelles.
- Si le joueur choisit la carte `tamanoir` (l'animal avec le poids le plus proche du `lama`), il remporte la manche, défausse uniquement cette carte et réduit la taille de sa main.

L'objectif pour le joueur est de poser toutes ses cartes le plus rapidement possible.

FIGURE 2 – Quel animal a le poids le plus proche d'un `lama` ?

2 Organisation de la SAÉ

La SAÉ est organisée par étapes. Chaque étape consiste à écrire certaines fonctionnalités et les tests associés.

Consigne

Les étapes seront progressivement validées. Ne passez à l'étape suivante que quand une étape est validée (code complet et commenté et classe de test écrite et validée).

Important

Un premier dépôt est **à faire à l'étape 2** après 6h de SAÉ.

3 Étape 1 : Classe `Carac` (1h)

La classe `Carac` a pour objectif de représenter une caractéristique. Chaque caractéristique est décrite par :

- un attribut `nomCarac` de type `String` représentant le nom de la caractéristique (par exemple, `"taille"`) ;
- un attribut réel `valeur` représentant la valeur de la caractéristique (par exemple, `8.0` pour la taille de la grenouille).

La classe `Carac` possède un constructeur avec deux paramètres qui créé un objet de type `Carac` dont les attributs sont passés en paramètre.

Cette classe possède en outre une méthode `double getValeur()` qui retourne la valeur de la caractéristique et une méthode `boolean etreEgal(String nom)` qui retourne `true` si et seulement si le `nom` passé en paramètre est égal à l'attribut `nomCarac` de la caractéristique.

Question 1

Écrire la classe `Carac` (cette classe simple n'a pas besoin d'être testée).

4 Étape 2 : Classe `Carte` (5h)

4.1 Descriptif des cartes

Les cartes de la SAÉ sont légèrement plus complexes que celles du TP11. Chaque carte est décrite par

- un attribut `nom` de type `String` représentant le nom de l'animal (par exemple, `"Grenouille Verte"`) ;
- un attribut `boolean` nommé `visible` qui vaut `true` si et seulement si la face avec les caractéristiques de la carte est visible (`false` par défaut) ;
- un tableau d'objets `Carac` nommé `caracs` et correspondant aux différentes caractéristiques de l'animal (par exemple, pour la `grenouille verte`, le tableau contient les caractéristiques `("taille",8.0), ("poids", 0.02), ("longevite", 8.0)` - cf figure 3).

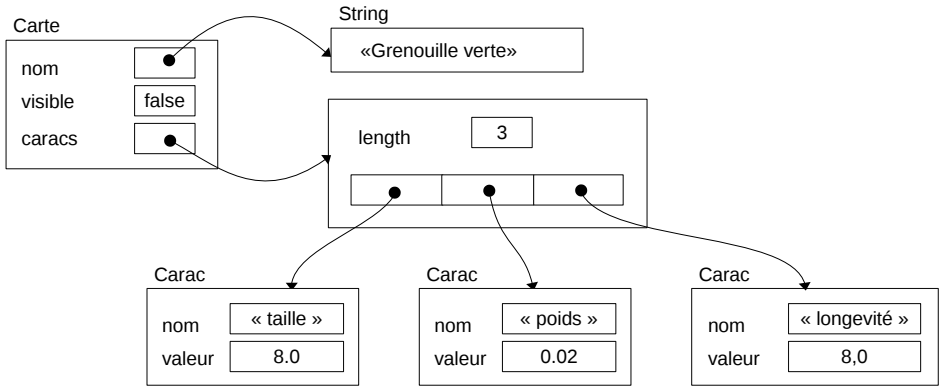

FIGURE 3 – Schémas mémoire représentant la structure interne de la carte `Grenouille Verte`

La classe `Carte` possède en outre

- un constructeur permettant de construire une carte à partir de son nom et ne possédant initialement aucune caractéristique (correspondant à un tableau `caracs` de taille 0) ;
- une méthode `void ajouterCarac(Carac c)` permettant d'ajouter une caractéristique à la liste des caractéristiques de l'animal (en créant un nouveau tableau) ;
- une méthode `double getValeur(String nom)` qui retourne la valeur de la caractéristique dont le nom est passé en paramètre. Si cette caractéristique n'existe pas, cette méthode doit retourner `-1.0`.

Question 2

Ecrire la classe `Carte`, compléter la classe de test `TestCarte` et vérifier que les méthodes de la classe `Carte` fonctionnent correctement.

4.2 Descriptif des fichiers fournis

Les cartes d'un jeu sont décrites dans des fichiers textes fournis sur arche. Ces fichiers sont au format CSV (*Comma-Separated Values*) : les valeurs sont séparées par des caractères `';'` et un tel fichier peut s'ouvrir avec un tableur (comme Excel ou LibreOffice).

- le fichier `"animaux.csv"` décrit des animaux avec une taille (en cm), un poids (en kg) et une longévité (en année)
- Un second fichier, le fichier `"pokedex.csv"`, est fourni sur arche, décrit des Pokemons avec leur taille (en cm) et leur poids (en kg) et vous permet de tester votre application sur d'autres données.

Les deux fichiers fournis sont toujours structurés de la manière suivante :

- la première ligne décrit les noms des caractéristiques des cartes du fichier. Par exemple, dans le fichier `"animaux.csv"`, la première ligne est :
  `"nom;taille;poids;longevite;"`
- Les lignes suivantes contiennent une carte par ligne et sont structurées sous la forme `"<NOM>;<CARAC1>;<CARAC2>;...;"` où `"<NOM>"` désigne le nom de l'animal, `"<CARAC1>"` la valeur de la première caractéristique, etc... Ainsi, dans le fichier `"animaux.csv"`, la ligne 9
  `"koala;73;9;15;"`
  décrit la carte `koala` d'une taille de 73 cm (première caractéristique), d'un poids de 9 kg (seconde caractéristique) et d'une longévité de 15 ans (troisième caractéristique).

Le début du fichier `"animaux.csv"` contient les lignes suivantes :

```
nom;taille;poids;longevite;
lapin nain;19;1.30;7;
saint Bernard;80;92;8;
[...]
```

- La première ligne `"nom;taille;poids;longevite;"` décrit la structure du fichier. Elle indique que le fichier animaux contient 3 caractéristiques par carte (en plus du nom), à savoir, dans l'ordre, taille, poids puis longévité.

- la seconde ligne `"lapin nain;19;1.30;7;"` décrit une carte `lapin nain`. Un `lapin nain` a donc une taille de 19 cm (première caractéristique), un poids de 1.3 kg (seconde caractéristique) puis une longévité de 7 ans (troisième caractéristique).

4.3 Chargement des fichiers

La classe `LectureFichier` **fournie sur arche** permet de lire un fichier texte. Elle contient

- un constructeur `LectureFichier(String nom)` qui prend en paramètre un nom de fichier et vérifie si le fichier est bien présent. Si le fichier est absent, le constructeur renvoie une erreur et le programme s'arrête.
- une méthode `String[] lireFichier()` qui lit le fichier passé à la construction de l'objet et retourne un tableau de `String`. Chaque `String` de ce tableau correspond à une ligne du fichier lu.

Utilisé pour lire le fichier `"animaux.csv"`, le tableau retourné possède 58 cases car le fichier contient 58 lignes. La chaîne située dans la première case vaut

`"nom;taille;poids;longevite;"`.

Elle correspond à la première ligne du fichier et au descriptif des cartes du jeu.

Consigne

On considérera que le nom est **toujours** le premier élément de chaque ligne et on le traitera séparément du reste.

Question 3

Écrire une classe `MainLecture` qui charge le contenu du fichier `"animaux.csv"` et affiche les lignes une après l'autre.

4.4 Création d'une carte à partir de chaines

On souhaite pouvoir construire une carte en passant deux chaines de caractères en paramètre :

1. la chaine décrivant les caractéristiques des cartes. Par exemple, pour le fichier `"animaux.csv"`, la chaine `""nom;taille;poids;longevite;""`.
2. une chaine décrivant une carte. Pour la carte `lapin nain` du fichier `"animaux.csv"`, la seconde chaine serait `"lapin nain;19;1.30;7;"`.

Le constructeur de la classe `Carte` doit donc fonctionner selon l'exemple de code suivant et fournir une carte correspondant à un `lapin nain` avec les bonnes caractéristiques.

```java
String carac = "nom;taille;poids;longevite;";
String valeurs = "lapin nain;19;1.30;7;";
// construit une carte "lapin nain" avec les bonnes caracteristiques
Carte c = new Carte(carac,valeurs);
// par ex, un poids de 1.30
System.out.println(c.getVal("poids"));
```

Pour aborder cette question, on pourra utiliser la méthode `String[] split(String separateur)` de la classe `String`. Cette méthode permet de couper une chaine en plusieurs morceaux selon le paramètre `separateur` passé à la méthode et retourne le tableau contenant tous les morceaux obtenus.

Par exemple, le code suivant permet de couper la chaine `chaineACouper` selon les caractères `';'`.

```java
// la chaine qu'on souhaite couper en morceaux selon ';'
String chaineACouper = "pomme;banane;fraise;ananas;";

// generation du tableau avec split selon les ";"
String[] morceaux = chaineACouper.split(";");

// morceaux est un tableau de taille 4 (car il contient 4 morceaux)
System.out.println(morceaux.length);

// le premier morceau (indice 0) est "pomme"
System.out.println(morceaux[0]);
// le troisieme morceau (indice 2)  est "fraise"
System.out.println(morceaux[2]);
```

On utilisera aussi la méthode `double Double.parseDouble(String s)` permettant de convertir un `String` en `double`.

```java
String chaine = "3.14";
double val = Double.parseDouble(chaine);
```

Question 4

Écrire ce constructeur dans la classe `Carte`. Vous supposerez que les chaînes passées en paramètre sont bien formées.

Question 5

Compléter la classe `TestCarte` pour valider que votre constructeur fonctionne correctement et créé des cartes avec les bonnes caractéristiques.

4.5 Affichage d'une carte

Une carte s'affiche différemment selon qu'elle est censée être visible ou non.

- Si l'attribut de la carte `visible` vaut `false`, les valeurs des caractéristiques ne sont pas visibles (face verso) et la carte doit s'afficher sous la forme
  `"<NOM> -> *<CARAC1> ???* *<CARAC2> ???*"`
  où `<NOM>` désigne le nom de la carte et `<CARAC1>`,`<CARAC2>`, ... les noms des caractéristiques. Par exemple, pour la carte `lapin nain`

```
lapin nain -> *taille ???* *poids ???* *longevite ???*
```

- Si la carte est visible, ses caractéristiques sont visibles et la carte doit s'afficher sous la forme
  `"<NOM> -> *<CARAC1> <VAL1> *<CARAC2> <VAL2>*"`
  avec les bonnes valeurs. Par exemple, pour la carte `lapin nain`

```
lapin nain -> *taille 19.0* *poids 1.30* *longevite 7.0*
```

Question 6
Écrire la méthode `toString` correspondante.

4.6 `MainCarte` pour valider

Afin de valider vos méthodes, écrire un `main` dans la classe `MainCarte` qui

1. charge le fichier `animaux.csv` et stocke le tableau de `String` ;
2. puis, pour chaque chaîne de ce tableau (sauf la première ligne),
   - construit une carte en passant la première ligne et la ligne parcourue,
   - l'affiche,
   - la change de côté (recto / verso),
   - l'affiche à nouveau.

Question 7
Écrire la classe `MainCarte` et toutes les méthodes utiles de la classe `Carte`.

4.7 Tests

Question 8
Compléter la classe de test `TestCarte.java` chargée de vérifier que les différentes méthodes écrites fonctionnent.

Consigne
Valider complètement cette partie avant de passer à la suite.

Important
Une fois cette première partie validée, **déposer sur arche cette version correspondant à la version 1** de votre projet. Ce dépôt doit être effectué à l'issue des 6 premières heures.

5 Étape 3 : Gestion de la pioche et de la main du joueur (3h)

Vous disposez maintenant de cartes. L'objectif de cette partie est de pouvoir disposer de paquet de cartes pour représenter la pioche et la main de cartes du joueur.

5.1 Constructeur, ajout et suppression de carte

Le TP11 vous a proposé d'écrire une classe destinée à gérer un paquet de cartes. Cette classe doit posséder

- un constructeur vide ;
- un constructeur à partir d'un tableau de cartes ;
- une méthode `ajouterCarteFin` ;
- une méthode `retirerCarte`.

Question 9
Adapter et finir le TP11 pour disposer d'une classe `PaquetCarte` permettant de manipuler un paquet de cartes. Vous penserez à changer la classe `Carte` utilisée.

5.2 Constructeur à partir d'un fichier

On souhaite pouvoir construire un paquet de cartes directement à partir d'un nom de fichier et de la classe `LectureFichier`.

Question 10
Proposer un constructeur supplémentaire capable de créer un paquet de cartes en donnant uniquement un nom de fichier et en chargeant les cartes décrites dans ce fichier.

5.3 Piocher une carte

On souhaite disposer d'une méthode permettant de piocher une carte aléatoirement dans un paquet. Cette méthode ne prend aucun paramètre, retourne la carte tirée au hasard et la supprime du paquet. Si le paquet est vide, la méthode retourne simplement `null`.

Afin de sélectionner une carte au hasard, on pourra utiliser la classe JAVA `Random`. Cette classe :

- nécessite d'importer le package `java.util.Random` tout en haut de la classe `PaquetCarte` ;
- possède un constructeur vide ;
- possède une méthode `int nextInt(int bound)` qui retourne un entier choisi de manière aléatoire et compris entre `0` et `bound-1` (compris).

Le descriptif complet de la classe est disponible à l'adresse `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html`.

Question 11
Ajouter une méthode `piocherHasard` à la classe `PaquetCarte`.

5.4 Chercher la carte la plus proche

Pour connaître la bonne réponse d'une manche, on souhaite avoir une méthode capable de savoir quelle carte d'un paquet est la carte la plus proche d'une autre carte.

La méthode `int trouverCarteProche(Carte c, String nomCarac)` parcourt les cartes d'un paquet et retourne l'**indice** de la carte du paquet qui est la plus proche de la carte `c` passée en paramètre selon la caractéristique nommée `nomCarac`.

Question 12
Écrire la méthode `trouverCarteProche` et créer les tests pour la valider.

5.5 Méthode toString

Question 13
Compléter la méthode `toString` proposée dans le `TP11` pour pouvoir afficher toutes les cartes d'un paquet de cartes.

5.6 Tests

Question 14

Compléter la classe de test `TestPaquet.java` chargée de vérifier que les différentes méthodes écrites fonctionnent. Vous pourrez vous inspirer des tests proposés dans la classe `TestPaquet` fournie dans le TP11.

Consigne

Valider complètement cette partie avant de passer à la suite.

6 Étape 4 : Mise en place et exécution du jeu (2h)

Vous disposez désormais de tous les éléments pour écrire la classe `Jeu` chargée d'exécuter le jeu.

La classe `Jeu` contient

- un paquet de cartes représentant la main du joueur ;
- un paquet de cartes représentant la pioche des cartes ;
- un attribut `nomFichier` correspondant au nom de fichier utilisé pour récupérer les cartes ;
- un attribut entier `nombreCarte` correspondant au nombre de cartes initiales du joueur.

6.1 Mise en place du jeu

L'installation du jeu consiste à

- charger toutes les cartes dans la pioche à partir du nom de fichier ;
- tirer au hasard à partir de la pioche le nombre de cartes nécessaires pour remplir la main du joueur.

6.2 Déroulement de la partie

La partie se déroule manche par manche. À chaque manche,

- le jeu tire au hasard une carte de la pioche de cartes ;
- le jeu rend la carte visible et l'affiche au joueur ;
- le jeu choisit une caractéristique au hasard parmi les caractéristiques de la carte (vous pouvez modifier la classe `Carte` si besoin) et la présente au joueur ;

- le jeu demande au joueur le numéro de la carte qu'il pense être la plus proche et qu'il souhaite jouer de sa main (entre 0 et `tailleMain-1`) ;
- le jeu joue ce coup en fonction des règles du jeu, informe le joueur si la carte proposée est la bonne carte ou non et met à jour l'état du jeu en fonction de la réussite ou non du joueur (cartes défaussées, etc...).

6.3 Fin du jeu

Le jeu s'arrête sur une des conditions suivantes :

- le joueur a joué toutes les cartes de sa main : c'est une victoire.
- la pioche est vide : c'est une défaite.

6.4 Exemple

Voici un exemple d'exécution du jeu (le premier coup) tel qu'affiché dans la console :

Affichage console

```
---------------------------------CARTE ET CARAC---------------------------------
homard -> *taille 50.0* *poids 4.0* *longevite 40.0*
caracteristique testee :longevite
--------------------------------------------------------------------------------

---------------------------------MAIN DU JOUEUR---------------------------------
 0. hibou grand duc -> *taille ???* *poids ???* *longevite ???*
 1. coq -> *taille ???* *poids ???* *longevite ???*
 2. lapin -> *taille ???* *poids ???* *longevite ???*
 3. eccureuil roux -> *taille ???* *poids ???* *longevite ???*
 4. coccinelle -> *taille ???* *poids ???* *longevite ???*
--------------------------------------------------------------------------------

-------------------------------------CHOIX--------------------------------------
quelle carte de votre main est la plus proche ?
0
--------------------------------------------------------------------------------

---------------------------------CARTE JOUEE------------------------------------
hibou grand duc -> *taille 72.0* *poids 2200.0* *longevite 24.0*
-------------------------------------REPONSE------------------------------------
bien joue, y avait pas mieux
--------------------------------------------------------------------------------

---------------------------------CARTE ET CARAC---------------------------------
coyote -> *taille 87.0* *poids 16.0* *longevite 11.0*
caracteristique testee :taille
--------------------------------------------------------------------------------

---------------------------------MAIN DU JOUEUR---------------------------------
 0. coq -> *taille ???* *poids ???* *longevite ???*
```

```
 1. lapin -> *taille ???* *poids ???* *longevite ???*
 2. eccureuil roux -> *taille ???* *poids ???* *longevite ???*
 3. coccinelle -> *taille ???* *poids ???* *longevite ???*
--------------------------------------------------------------------------------

-------------------------------------CHOIX--------------------------------------
quelle carte de votre main est la plus proche ?
1
--------------------------------------------------------------------------------

---------------------------------CARTE JOUEE--------------------------------
lapin -> *taille 42.0* *poids 2.5* *longevite 7.0*
-----------------------------------REPONSE---------------------------------
eh non, il y avait une carte plus proche:
coq -> *taille 65.0* *poids 3.2* *longevite 14.0*
on les retire et on ajoute deux nouvelles cartes
--------------------------------------------------------------------------------
```

6.5 Travail à réaliser

Question 15

Décider des méthodes et programmer la classe `Jeu` pour pouvoir créer et exécuter une partie.

Question 16

Pour simplifier les affichages, n'hésitez pas à définir une méthode capable de faire une ligne de `'-'` de taille constante en insérant un texte au milieu, comme

```
"-------------------------------------CHOIX--------------------------------------"

"---------------------------------MAIN DU JOUEUR---------------------------------"
```

Question 17

Écrire la classe `ProgJeu` permettant de créer et de lancer un jeu. Le nom du fichier texte à utiliser pour construire la pioche sera stocké dans une variable pour pouvoir facilement le changer et passé en paramètre à la construction du jeu.

Consigne

Valider complètement cette partie avant de passer à la suite.

7 Rendu attendu

7.1 Version 1

La version 1 de votre projet a du être déposée à la fin de la classe `Carte`. Elle ne constitue qu'une étape intermédiaire de votre rendu.

7.2 Version 2

La version 2 de votre application est une version complète de votre projet. Elle doit contenir

- votre code complet (classes, classes de test, classes main attendues) ;
- un code commenté (commentaires internes, javadoc) ;
- un petit compte-rendu qui présente votre travail.

Ce compte-rendu devra préciser

- les noms, prénoms et groupe des membres du binôme ;
- l'avancée de votre projet (à quelle partie en êtes vous) ;
- un déroulé de votre travail (combien de temps pour chacune des parties réalisées) ;
- les difficultés éventuelles rencontrées lors du travail (éventuellement les questions que vous n'avez pas faites) ;
- la manière dont vous avez géré les différents problèmes rencontrés ;
- quelques explications sur la manière dont vous avez programmé la classe `Jeu` ;
- ce que vous avez appris en effectuant cette SAÉ.

L'objectif du compte rendu est de prendre le temps de détailler vos difficultés et de montrer que vous avez bien compris le sujet. Il ne s'agit de faire des discours abstraits mais d'expliquer en détail ce que vous avez découvert sur CE projet et les éléments qui vous ont pris du temps sur CE sujet (et pour quelle raison).

Consigne

IMPORTANT : Avant de passer à la suite, sauver une version 2 de votre application dans un fichier zip et déposer cette version sur arche.

7.3 Version 3

Une fois votre version 2 déposée sur arche, copier cette version et la renommer en une version v3 pour aborder les questions optionnelles qui arrivent ensuite. Cette nouvelle version servira de base à la suite de votre travail pour éviter les confusions.

Si vous répondez à des questions optionnelles, vous penserez à déposer votre version 3 dans le second dépôt arche.

Important

Les questions optionnelles pourront vous donner quelques points bonus si vous avez fini les parties précédentes. **Ne débutez pas cette partie, si votre projet n'est pas complet ni correctement finalisé.** Les parties suivantes ont pour objectif de vous permettre de continuer à travailler sur la SAÉ si vous avez fini votre projet avant les 12 heures.

8 Étape 5 : Options de jeu (partie optionnelle)

Une fois la version 2 complétée, différentes extensions sont possibles pour améliorer votre jeu. Si vous avez le temps, vous pouvez développer une (ou plusieurs) de ces extensions

8.1 Gestion du score

Désormais, l'application compte en plus le nombre de coups nécessaires pour que le joueur finisse la partie. Ce nombre de coups correspondra au score du joueur. Plus ce score est faible, meilleur le joueur est.

À la fin d'une partie, le score de l'utilisateur est affiché à l'écran et le jeu demande si on souhaite faire une nouvelle partie.

Question optionelle 18

Ajouter la gestion du high-score à l'application.

8.2 Gestion des meilleurs scores obtenus

Le jeu mémorise les 5 meilleurs scores obtenus par le joueur. Dés qu'un meilleur score est obtenu à la fin d'une partie, le jeu demande le nom du joueur qui vient de faire la partie et associe ce nom au high-score.

À la fin de chaque partie, avant de redemander si le joueur veut lancer une nouvelle partie, le jeu affiche les high-score mémorisés.

Comme le nombre de coups dépend de la main initiale du joueur, on utilisera au début les high-score d'une taille de main standard (6 cartes) puis on pourra stocker des high-score pour chaque niveau de difficulté (taille de la main initiale).

Question optionelle 19

Pour faire cela en tirant parti de la programmation orientée objet, gérer les high-score à l'aide d'une classe dédiée nommée `HighScore`.

8.3 Sauvegarde des high-scores

On souhaite sauver et charger les high-score dans un fichier texte pour pouvoir mémoriser les meilleurs scores entre deux lancements de l'application.

La classe `EcritureFichier` est fournie sur arche et permet d'écrire dans un fichier. Elle fonctionne en plusieurs étapes

- on créée d'abord un objet `EcritureFichier` en lui donnant le nom du fichier en paramètre ;
- on ouvre ensuite le fichier avec la méthode `void ouvrirFichier()` ;
- on écrit dans le fichier ligne par ligne grâce à la méthode `void ecrireFichier(String ligne)` ;
- enfin, on ferme et on sauve le fichier avec la méthode `void fermerFichier()`.

Par exemple, le code suivant crée un fichier "test.txt" et ajoute 10 lignes à l'intérieur

```
// creation de l'objet pour ecrire
EcritureFichier fichier = new EcritureFichier("test.txt");

// ouverture du ficheir
fichier.ouvrirFichier();

// ecriture de 10 lignes
for (int i = 0; i < 10; i++){
  fichier.ecrireFichier("ligne " + i);
}

// fermeture et sauvegarde
fichier.fermerFichier();
```

Question optionelle 20

A l'aide de la classe `EcritureFichier` fournie sur arche, gérer la sauvegarde et le chargement des high-score (pensez au préalable au format du fichier).

8.4 Fusion de paquets de cartes

On souhaite pouvoir fusionner deux paquets de cartes pour mélanger des cartes issues de deux fichiers différents.

Question optionelle 21

Modifier l'application pour demander au lancement plusieurs fichiers de carte et fusionner les paquets de cartes pour n'en faire qu'une seule pioche.