Soutenance projet n°3

DA JAVA

Projet “Escape game online”

par Arnaud Laval

Date de soutenance prévue : 30/04/2019

Mentor évaluateur : Serge Coudé

Code source : https://github.com/Nanooneg/P3

Sommaire :

Contexte

A. Mission : Réalisation d'un jeu, le "Plus ou Moins"

3 modes de Jeu :

Challenger : Le joueur humain joue le rôle d'attaquant et doit trouver la combinaison du joueur IA (le défenseur).

Défenseur : Le joueur humain prend le rôle du défenseur et soumet une combinaison que le Joueur IA tente de découvrir.

Duel : Le joueur humain et le joueur IA jouent à tour de rôle pour découvrir leurs combinaisons respectives.

A chaques fois qu'un joueur propose une nouvelle réponse, on lui retourne un modèle sous la forme : "+-+=" où les "+" remplacent les chiffres de sa réponse qui sont trop petits, les "-" ceux trop grands et les "=" les bons. Le joueur a un nombre limité d'essai pour trouver la bonne réponse.

B. Déroulement d'une partie :

Lancement du programme

↓

Chargement des paramètres : X chiffre(s) dans la combinaison et Y essai(es)

↓

Le joueur humain choisit un mode

↓

1-Challenger 2-Défenseur 3-Duel

↓

Le joueur humain choisit le mode 1

↓

Le joueur IA génère une combinaison aléatoire de X chiffre(s)

↓

Le joueur humain propose une réponse

↓

L'IA lui renvoie le modèle ("+-+=")

↓

Ces deux dernières actions se répètent tant que le joueur humain ne trouve pas la combinaison de l'IA ou que le nombre d'essai(es) maximum(s) Y est atteint

↓

Si le joueur humain a trouvé : il gagne

↓

On propose au joueur humain de :

↓

Rejouer/Retourner au menu/Sortir

Démonstration

À faire lors de l’entretien...

Points importants

```java
public Map<Integer,String> genererReponse(int coupRestant, int coupMax, String modele, Map<Integer,String> memoire) {
    logger.trace( s: "début de demande à l'IA de générer une réponse");

    String temp;
    char[] caractereModele = modele.toCharArray(); 1

    int i, min, max, taille;
    if (memoire.get(0).isEmpty()){
        for (i = 0, taille = memoire.size(); i<taille; i++) {
            historique.ajouter(memoire,i, ajout: "5"); 2
        }
        System.out.println("J'ai " +coupMax+ " coup(s) pour trouver : ");
    } else {
        for (i = 0, taille = memoire.size(); i<taille; i++){
            switch (caractereModele[i]){
                case '+' : 3
                    historique.ajouter(memoire,i, ajout: "+"); 4
                    min = historique.lireMinimum(memoire.get(i)); 5
                    max = historique.lireMaximum(memoire.get(i));
                    do {
                        temp = String.valueOf(min + (int) (Math.random() * ((max - min) + 1))); 6
                    }while (!historique.parcourir(memoire,i,temp)); 7

                    historique.ajouter(memoire,i,temp); 4bis
                    break;
                case '-' : 3
                    historique.ajouter(memoire,i, ajout: "-"); 4
                    min = historique.lireMinimum(memoire.get(i)); 5
                    max = historique.lireMaximum(memoire.get(i));
                    do {
                        temp = String.valueOf(min + (int) (Math.random() * ((max - min) + 1))); 6
                    }while (!historique.parcourir(memoire,i,temp)); 7

                    historique.ajouter(memoire,i,temp); 4bis
                    break;
            }
        }
        if (coupRestant == 1)
            System.out.println("Dernier coup !!");
        else
            System.out.println("Il me reste " +coupRestant+ " coup(s) : ");
    }

    logger.trace( s: "fin de demande à l'IA de générer une réponse");
    logger.debug( s: "map renvoyée : " +memoire);
    return memoire;
}
```

A. Processus de génération de réponse par l'IA

1.Génération aléatoire pertinente

Cette méthode travaille avec une Map qui contient l'historique des réponses et le modèle généré après comparaison de ces dernières :

- la première réponse sera composée de X chiffre(s) '5' qui sont ajoutés à l'historique 2.

- pour les autres réponses, la méthode commence par découper le modèle 1 pour travailler indépendamment sur chaque chiffre 3, puis leur ajoute le signe du modèle correspondant 4 dans l'historique. Elle récupère en suivant les minimums et maximums les plus pertinents 5 pour générer aléatoirement 6 une nouvelle réponse qu'elle n'a pas déjà proposée 7.

Elle retourne l'historique mis à jour. Une autre méthode se chargera de récupérer la dernière réponse, pour la soumettre au comparateur.

A suivre

```java
public int lireMinimum (String valeurs){
    int minimum = 0, i;
    logger.debug( s: "chaîne lue : " + valeurs);

    if (valeurs.equals("")) {
        minimum = 0;
        logger.debug( s: "nouveau Min : " + minimum);
    }else {
        for (i = 0; i < valeurs.length(); i += 2) {
            if ((valeurs.substring(i, i + 1)).equals("+")) {
                logger.debug(valeurs.substring(i, i + 1));
                logger.debug(Integer.parseInt(valeurs.substring(i + 1, i + 2)));
                if (Integer.parseInt(valeurs.substring(i + 1, i + 2)) > minimum) {
                    minimum = Integer.parseInt(valeurs.substring(i + 1, i + 2));
                    logger.debug( s: "nouveau Min : " + minimum);
                }
            }
        }
    }
    return minimum;
}
```

2.Définition des bornes

Cette méthode renvoie le minimum le plus pertinent. Elle prend en paramètre l'historique : Les valeurs associées à chaque chiffre ainsi que le signe qui leur est assigné (voir 4/4bis dans la vignette précédente). Elle retourne la plus grande valeur accompagnée d'un signe '+' (la plus grande des réponses trop petites). La méthode qui renvoie le maximum fonctionne sur le même principe.

3.Récupération de la réponse

Cette méthode s'occupe de récupérer la dernière réponse générée en concaténant les 1er caractères de chaque valeur associée au chiffre correspondant.
Elle retourne une chaîne de caractère qui sera comparée à la combinaison du défenseur pour générer un nouveau modèle.

```java
public String lireReponse (Map<Integer,String> memoire){
    logger.trace( s: "début de lecture dans  la map");

    String reponse ="";
    int i;

    logger.debug( s: "état de la map : " + memoire);
    for (i = 0; i < memoire.size(); i++)
        reponse += memoire.get(i).substring(0, 1);


    logger.debug( s: "réponse renvoyée : " +reponse);
    logger.trace( s: "fin de lecture dans  la map");
    return reponse;
}
```

```
do {
    try {
        choix = sc.nextInt();
        logger.debug( s: "Saisie utilisateur : " +choix);
        saisieOk = (choix >= 1 && choix <= 3);
    } catch (InputMismatchException e) {
        sc.next();
        logger.error( s: "Erreur de saisie !!");
        saisieOk = false;
    }
    if (!saisieOk) {
        this.couleurPolice( cas: 4);
        System.out.println("Erreur de saisie");
        this.couleurPolice( cas: 3);
        System.out.print("Que souhaites-tu faire (1/2/3) : ");
    }
}while (!saisieOk);
```

B. Gestion des erreurs

Ce bout de code est utilisé pour demander une saisie à l'utilisateur humain et gérer les cas où il ne répond pas ce que l'on attend de lui.
Le try/catch gère les cas où la réponse est d'un autre type qu'un nombre (exceptions) et après la saisie, on vérifie que le nombre donné est dans l'ordre de grandeur voulu 1.

C. Comparaison

Cette méthode compare la réponse générée avec la combinaison du défenseur.
Elle prend donc, en paramètres, les deux chaînes de caractères et commence par les découper dans des tableaux de caractères pour les comparer indépendamment. À chaque comparaison elle ajoute un '+', un '-' ou un "=" dans une troisième chaîne de caractères "modèle" qu'elle renvoie.

```
public String comparer (String combinaison, String reponse){
    logger.trace( s: "début comparaison des combinaisons");

    int nombreChiffre = reponse.length();
    String modele = "";
    char[] chiffreCombinaison = combinaison.toCharArray();
    char[] chiffreReponse = reponse.toCharArray();

    int i;
    for (i=0; i<nombreChiffre; i++) {
        if (chiffreCombinaison[i] < chiffreReponse[i])
            modele += "-";
        else if (chiffreCombinaison[i] > chiffreReponse[i])
            modele += "+";
        else if (chiffreCombinaison[i] == chiffreReponse[i])
            modele += "=";
    }

    logger.trace( s: "fin comparaison des combinaisons");
    logger.debug( s: "combinaisonA : " +combinaison+ " combinaisonD
    return modele;
}
```

D. Mise en forme

J'ai rajouté des changements de couleurs de police dans le terminal d'IntelliJ : Bleu, quand le joueur humain joue, jaune pour l'IA et rouge, quand il y a une erreur de saisie.
Cette fonctionnalité n'est pour le moment pas compatible avec les autres terminaux de commandes.
Elle pourra être développée, si cela est nécessaire, plus tard.

```java
public void couleurPolice (int cas){

    switch (cas){
        case 1:
            System.out.print("\033[34m");    //bleu
            break;
        case 2:
            System.out.print("\033[33m");    //jaune
            break;
        case 3:
            System.out.print("\033[30m");    //blanc
            break;
        case 4:
            System.out.print("\033[31m");    //rouge
    }
}
```

```java
public void decor (String cas, boolean retourAvant, boolean retourApres) {

    switch (cas) {
        case "double":
            if (retourAvant)
                System.out.println("\n");
            System.out.println("==================================");
            if (retourApres)
                System.out.println("\n");
            break;
        case "simple":
            if (retourAvant)
                System.out.println("\n");
            System.out.println("----------------------------------");
            if (retourApres)
                System.out.println("\n");
            break;
    }
}
```

J'ai également utilisé, à plusieurs reprises, des lignes de caractères en guise de séparateur.
Ces deux méthodes ne changent pas le fonctionnement du programme et servent seulement à l'ergonomie de l'affichage et le confort de jeu. Voir *démonstration*

IV.Pistes pour améliorer/optimiser le programme :

1. Utiliser des énumérations pour les modes de jeu et pour les éléments de mise en forme (couleurs et lignes) pour faciliter d'éventuelles futures implémentations.
2. Ajouter la possibilité de récupérer des arguments au lancement du programme pour changer les paramètres de fonctionnement, initialement chargés depuis le fichier config.properties :
    a. Nombre de chiffre(s) de la combinaison.
    b. Nombre de coup(s) maximum(s) alloué(s).
3. Rendre compatible l'affiche des couleurs de polices sur tous les supports.
4. Gérer les affichages de solution différemment pour alléger le code. Peut-être avec des énumération également.