CROS ARTHUR // Développement JAVA

Soutenance « Escape Game Online »

Table des matières

Le projet

Le but du projet est de réaliser un jeu sur le principe du « + ou – » afin de deviner, ou de faire deviner au programme, un code à chiffres d'une longueur variable prédéfinie, en orientant à chaque tour la réponse par l'utilisation des symboles (+), (-) ou (=). Le jeu se termine lorsque la combinaison a été découverte, le nombre de tours maximum (variable et prédéfini également) a été atteint, ou encore, pour un troisième mode unique, le joueur ou le programme a trouvé la solution avant l'autre.

Il existe trois modes de jeu :

- Le mode « DEFENSEUR », dans lequel le joueur saisit sa combinaison et le programme tente ensuite de le découvrir en X coups maximum.
- Le mode « CHALLENGER », dans lequel le joueur doit tenter de découvrir la combinaison générée aléatoirement par le programme en X coups maximum.
- Le mode « DUEL » dans lequel le joueur et le programme s'affrontent pour tenter de deviner la combinaison de l'autre jusqu'à ce que l'un des deux trouve cette combinaison.

A la fin d'une partie, le programme doit demander au joueur s'il souhaite relancer une partie ou non, et si oui, s'il souhaite le faire dans le même mode, ou encore dans un autre mode.

Le programme doit également pouvoir afficher la combinaison qu'il a générée aléatoirement en début de partie si une option « développeur » a été activée au préalable.

Les variables telles que la longueur du code, le nombre de tours maximum et le mode développeur doivent être présentes et modifiables dans un fichier de configuration externe, sur lequel le programme se basera pour gérer le jeu.

Problématiques, contraintes et solutions //

+ Notion d'objet / Polymorphisme / Modularité

L'acquisition du concept principal d'objet, et de ses notions d'abstraction, a été capitale pour simplifier la lisibilité et la rédaction du code de ce projet. Plutôt que de multiplier les classes en fonction des modes de jeu, il est devenu évident qu'il était plus simple d'instancier cet objet et de modifier son comportement en fonction du mode choisi par le joueur en début de partie. Cela était d'autant plus vrai qu'au début du projet, le cas d'un second jeu complètement différent (le Mastermind) pouvant être choisi au départ, se présentait.

L'objectif premier était également de pouvoir ajouter des modes de façon simple sans modifier la structure du projet. L'utilisation de classes abstraites a donc été un levier majeur pour la réalisation de cet objectif.

Une partie se détermine par le mode de jeu choisi au départ par le joueur. Ce choix entraîne l'appel de la fonction dédiée à ce mode dans l'objet **Game** généré au début du programme (méthode ***selectGameMode***).

////////// Main.java

```
[…]
private Game selectGameMode() {
    sc = new Scanner(System.in);
    try {
        printMenuMode();
        gameMode = sc.nextInt();
        if (gameMode == 1) {
            return gameChoice.defenderMode();
        }
        else if (gameMode == 2) {
            return gameChoice.challengerMode();
        }
        else if (gameMode == 3) {
            return gameChoice.duelMode();
        }
        else {
            sc = null;
            throw new Exception();
        }
    } catch (Exception e) {
        logger.info("Merci de sélectionner votre mode de jeu avec 1, 2, ou 3");
        return selectGameMode();
    }
}
[…]
```

Le mode de jeu est récupéré depuis la classe **Recherche** qui étend la classe abstraite **Game**. Ceci a été fait dans une optique de modularité du programme. Par ce moyen il est extrêmement simple de rajouter un jeu différent dans une autre classe qui serait elle aussi une extension de **Game**.

////////// Game.java

```
public abstract class Game {

    public abstract Game challengerMode();
    public abstract Game defenderMode();
    public abstract Game duelMode();

    […]
}
```

////////// Recherche.java

```
public class Recherche extends Game {

[…]

}
```

+ Algorithme

Il a été indispensable de trouver la méthode la plus efficace dans la proposition d'une solution par le programme dans le cas des modes DEFENSEUR et DUEL. Mathématiquement, le plus pertinent est de proposer, à chaque essai, la moyenne de chaque chiffre composant le code, entre la proposition précédente et un maximum ou un minimum, déterminé par la réponse précédente. Au premier tour, la proposition sera systématiquement le '5' (un choix devant être fait entre le '4' et le '5' étant donné que la valeur médiane entre le '0' minimum, et le '9' maximum est de '4,5', cette valeur a été choisie pour simplifier le code de comparaison de l'algorithme).

Prenons par exemple le cas d'une partie avec un code à un seul chiffre. Si le code à trouver est le '1', le programme proposera en première solution '5', la réponse apportée sera donc '-'. Au second tour, le programme calculera la moyenne entre sa proposition précédente (à savoir '5') et le '0' (le chiffre à trouver étant inférieur), et proposera donc le chiffre '2' (2 étant la moyenne tronquée entre '0' et '5'). La réponse sera à nouveau '-'. Le programme calculera de même que précédemment la moyenne entre sa proposition précédente, le '2', et le '0', à savoir le '1' et proposera cette réponse qui sera effectivement la réponse adéquate.

L'utilisation comme chiffre de référence pour la moyenne ne pourra pas être autre chose que '0', '5', ou '10'. En effet seuls quatre cas peuvent se présenter :

- Si la proposition précédente était inférieure ou égale à 5, et la réponse était '-', alors il convient de faire la moyenne entre '0' et la valeur proposée précédemment ;
- Si la proposition précédente était inférieure ou égale à 5, et la réponse était '+', alors il convient de faire la moyenne entre la valeur proposée précédemment et '5' ;
- Si la proposition précédente était supérieure ou égale à 5, et la réponse était '-', alors il convient de faire la moyenne entre '5' et la valeur proposée précédemment ;
- Si la proposition précédente était supérieure ou égale à 5, et la réponse était '+', alors il convient de faire la moyenne entre la valeur proposée précédemment et '10'.

Cet algorithme est ensuite utilisé dans une boucle qui répète le processus pour chaque chiffre, autant de fois qu'il est nécessaire, afin de proposer un code comportant un nombre de chiffres équivalent à ce que la configuration réclame.

Cet algorithme ne se déclenche bien évidemment que lorsqu'il est nécessaire. Si la réponse précédemment apportée était '=', à ce moment-là, le programme reproposera simplement sa proposition précédente, en ne modifiant que les chiffres qui n'ont pas eu de réponse satisfaisante.

L'algorithme, représenté par la méthode ***tentativePc***, prend en paramètres initiaux, la réponse apportée au tour précédent compareUserResponse ('-', '+' ou '='), ainsi que la proposition précédente prevPropoPc. Il retourne ensuite la nouvelle valeur à proposer newPropoPc.

L'avantage d'effectuer ce calcul sur chaque chiffre composant le code, de manière indépendante, est qu'il suffit d'appeler cette méthode autant de fois que la configuration le réclame, et de reconstituer à la sortie, le code résultant, en mettant bout à bout chacune des propositions.

////////// CombinaisonRecherche.java

```
public int tentativePc(char compareUserResponse, int prevPropoPc) {
    if (compareUserResponse == '=') {
        newPropoPc = prevPropoPc;
    }
    else if (compareUserResponse == '-') {
        if (prevPropoPc <= 5) {
            newPropoPc = prevPropoPc / 2;
        }
        else {
            newPropoPc = prevPropoPc - ((prevPropoPc - 5) / 2);
        }
    }
    else if (compareUserResponse == '+') {
        if (prevPropoPc >= 5) {
            newPropoPc = prevPropoPc + ((10 - prevPropoPc) / 2);
        }
        else {
            newPropoPc = prevPropoPc + ((5 - prevPropoPc) / 2);
        }
    }
    return newPropoPc;
}
```

+ Variabilité de la configuration

Une des contraintes du programme était qu'il soit possible, au moyen d'un fichier de configuration, de modifier certaines règles du jeu, sans avoir à modifier le code pour autant. Il est alors devenu indispensable de créer un fichier de configuration simple, regroupant ces différentes règles, auquel le programme ferait appel afin de modifier son comportement.

Les valeurs variables en question sont :

- Le nombre de chiffres composant le code, représenté par la variable size ;
- Le nombre de tours maximum autorisés pour découvrir la solution pour les modes DEFENSEUR et CHALLENGER, représenté par la variable tries ;
- La possibilité d'afficher dès le départ, ou non, le code généré par le programme, à des fins de contrôle et de débogage, par l'utilisation de la variable devmode.

Ce fichier de configuration, nommé 'config.properties', est la seule partie modifiable du programme et doit se répercuter automatiquement à chaque endroit nécessaire du programme de manière transparente.

////////// CombinaisonRecherche.java

```
size = 4
tries = 4
devmode = 1
```

Pour utiliser ces valeurs, de la manière la plus efficace possible et pour éviter toute redondance à travers le programme, une classe dédiée à la lecture de ce fichier, GetProperties, a été créée. Cette classe comprend une méthode permettant de récupérer les valeurs contenues dans le fichier de configuration ainsi qu'un 'getter' par variable. Il suffit alors de faire appel à ces 'getters' depuis n'importe quelle autre classe pour utiliser ces valeurs simplement.

////////// GetProperties.java

```
private void run() {

    try {

        Properties prop = new Properties();
        String propFile = "config.properties";
        InputStream inputStream = getClass().getClassLoader().getResourceAsStream(propFile);

        if (inputStream == null) {

            throw new Exception();

        }

        prop.load(inputStream);
        size = prop.getProperty("size");

        prop.load(inputStream);
        tries = prop.getProperty("tries");

        prop.load(inputStream);
        devmode = prop.getProperty("devmode");

    }

    catch (Exception e) {

        logger.info("Exception catch");

    }

}
```

La variable size est utilisée à chaque fois que le programme doit générer un code secret ou tenter de deviner celui par l'utilisateur. L'utilisation systématique d'un découpage du code en chacun des chiffres le composant permet d'utiliser l'algorithme de proposition, ou de génération de code, autant de fois que précisé dans le fichier de configuration en utilisant simplement une boucle, appelant ces fonctions pour chaque chiffre, pour un maximum de fois équivalent à la taille définie dans la configuration.

////////// Recherche.java

```
public Game defenderMode() {
[…]
    for (int i = 0; i < size; i++) {
        char splitUserResponse = userResponse.charAt(i);
        int splitPrevPropoPc = prevPropoPc[i];
        newPropoPc[i] = defenderRecherche.tentativePc(splitUserResponse, splitPrevPropoPc);
    }
[…]
    return null;
}
```

De la même façon, la variable tries est utilisée dans une boucle représentant un tour de jeu, pour les modes DEFENSEUR et CHALLENGER, afin de limiter le nombre de tours maximum autorisés pour deviner le code généré par le programme ou pour que le programme lui-même devine le code saisi par l'utilisateur.

Il est à noter que la boucle possède également une condition de sortie pour le cas où la solution aurait été trouvée dans le nombre de tours impartis, par l'utilisation de la variable responseFound. Ces deux conditions font que cette boucle se répètera tant que le nombre de tours maximum n'a pas été atteint et tant que la solution n'a pas été trouvée.

+ Regex

Pour les modes CHALLENGER et DUEL, il a fallu trouver une solution la plus simple possible afin que le programme puisse déterminer si la solution qu'il a proposée est bien celle attendue par l'utilisateur. Plusieurs possibilités ont été envisagées :

- Créer une boucle pour chaque chiffre du code vérifiant si la réponse pour ce chiffre était bien le caractère '=' ;
- Attribuer une variable booléenne qui vérifierait l'exactitude de ce chiffre qu'il serait donc simple de vérifier (si la réponse pour chaque chiffre est 'true', alors la proposition est bien la bonne) ;
- Trouver une règle simple qui vérifie l'intégralité de la séquence de réponse en une fois et qui retourne une valeur confirmant la proposition.

Dans un souci de clarté, la dernière solution a été retenue. Pour que le programme puisse vérifier la réponse apportée, il fallait alors trouver une fonction qui lui permette simplement de regarder « si la réponse apportée ne contient que des symboles '=' », et, si c'est bien le cas, de retourner un message de victoire.

La solution à cette problématique a été l'utilisation d'une **Expression Régulière** ou '**Regex**'. Cette dernière représente une chaîne de caractères 'possibles' pour une condition. Il suffit alors de composer syntaxiquement une regex explicitant le fait que la réponse apportée ne doit contenir exclusivement que des caractères '='.

////////// Recherche.java

```
[…]
if (pcResponse.matches("\\={" + Integer.valueOf(size).toString() + "}")) {

    responseFound = true;
    logger.info("\r\n\r\nBravo ! Vous avez trouvé ma combinaison !");
    break;

}
[…]
```

+ Gestion des exceptions

Pour la robustesse du programme, un des aspects importants était aussi de pouvoir gérer toute saisie incorrecte du joueur, et ce par un apprentissage de la gestion des exceptions. Toute rubrique nécessitant une saisie de la part du joueur se trouve ainsi inclue dans une instruction **try** (…) **catch**, permettant, si la saisie ne correspond pas à celle attendue, de renvoyer un message d'erreur personnalisé et de redemander une saisie, cette fois-ci correcte.

////////// CombinaisonRecherche.java

```
try {
[…]
}
catch (Exception e) {
    logger.info("Une erreur est survenue ! Rappel, votre combinaison secrète doit comporter "
+ size + " chiffres uniquement.");
    return inputUserCode();
}
```

Une fois le concept acquis, il est devenu indispensable de l'utiliser systématiquement lorsque l'utilisateur doit effectuer une saisie.

Les conclusions

L'écriture de ce programme était finalement régie par des contraintes autant explicites (configuration variable pour la taille du code secret ou pour le nombre maximum de tours de jeu, l'existence d'un mode « développeur », la possibilité de choisir parmi trois modes de jeu différents) que liées à l'exécution même du programme (création d'un algorithme, utilisation de regex).

Les contraintes explicites dirigeaient pleinement le projet vers une direction et une nécessité d'utilisation de fonctions spécifiques à la gestion de ces problématiques (notion d'objet, polymorphisme, modularité). Ces fonctions sont le fondement de la logique de Java et sont les bases de la programmation dans ce langage.

Les problématiques plus implicites ont quant à elles nécessité une réflexion au-delà du langage lui-même, pour y trouver une logique mathématique, ou syntaxique de manière plus générale (algorithme de résolution, ou de génération de code, regex utilisée pour simplifier grandement un code pouvant être lourd et indigeste).

Un des points importants reste le fait que le programme est parfaitement adaptable dans sa forme actuelle. Il est aisé d'y rajouter un mode de jeu supplémentaire du fait de sa modularité extrême, ou même d'y ajouter un tout autre jeu (ce qui était une part du projet initialement, car il devait être possible à l'utilisateur de choisir un second jeu au départ, le Mastermind). Il est envisageable de réutiliser ce projet pour approfondir les notions d'algorithme (pour des jeux plus complexes tels que le Mastermind justement), ou de le faire évoluer vers une interface graphique plus agréable à l'œil. Ce programme est au final extrêmement simple dans sa conception, mais ouvre tout un panel de possibilités pour une possible évolution future.