

PROJET : LES PIRATES

LES TESTS

BEN JAAFAR CHEDLI / RIBEIRO DUARTE
BAGDATLI OKTAY / MUNOS ENZO
GUICHARD LUCAS / ESSENGUE MATIS

Lundi 27 mai 2024



Université Paul Sabatier - 2024

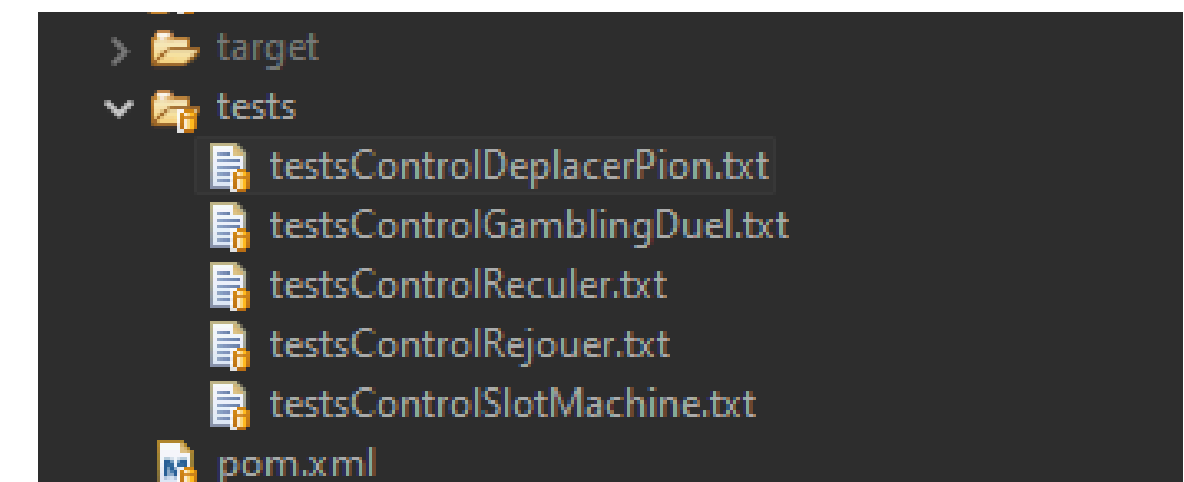
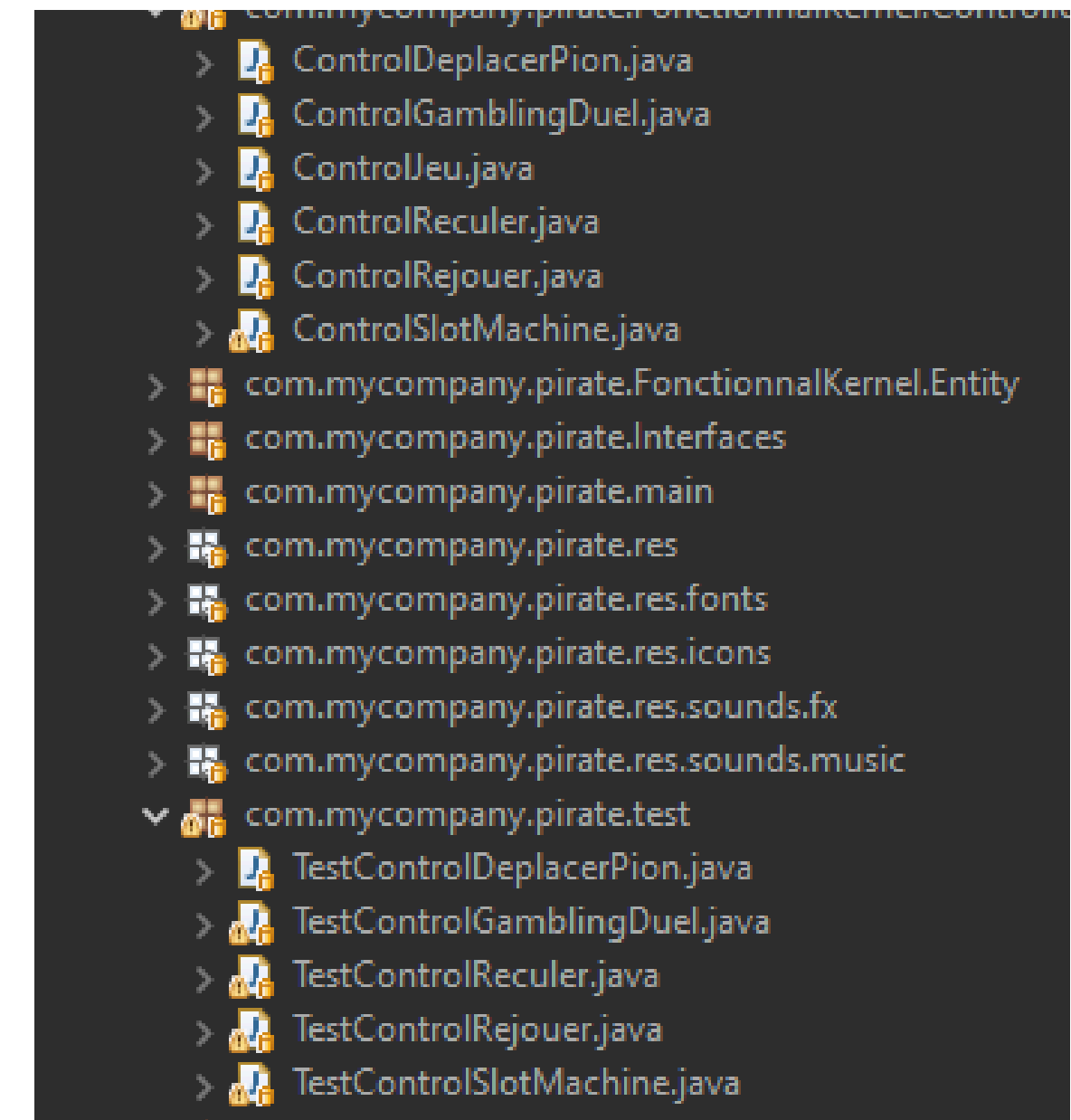
PARTIE DU PROJET TESTÉ • PRÉSENTATION TECHNIQUE • RÉALISATION DES TESTS • BILAN

Durant notre projet, le client nous a demandé de suivre une architecture ECB (*Entity-Control-Boundary*). Cette architecture permet aux contrôleurs de constituer le cœur fonctionnel de notre application, chacun d'eux étant responsable d'une partie spécifique du comportement du jeu. Ils effectuent les calculs et, dans le cadre de notre jeu, enregistrent les changements des états des variables pendant la durée du jeu. C'est pour cela que nous avons décidé de nous focaliser sur les tests des contrôleurs.

Nous listons donc cinq contrôleurs :

- ControlDeplacerPion : *gère le déplacement de nos pions.*
- ControlSlotMachine : *gère la génération des valeurs de notre machine à sous, qui nous donne la valeur de déplacement.*
- ControlRejouer : *gère la case spéciale nous permettant de rejouer un tour.*
- ControlReculer : *gère la case spéciale obligeant le joueur à reculer.*
- ControlGamblingDuel : *gère la case permettant de jouer au duel de "gambling" contre le robot.*

Nous ne testons pas ControlJeu, car celui-ci est un peu particulier étant donné qu'il ne contient que la boucle principale, qui ne fait qu'utiliser les autres contrôleurs. En testant nous garantissons que chaque aspect du jeu fonctionne correctement de manière indépendante




PARTIE DU PROJET TESTÉ • PRÉSENTATION TECHNIQUE • RÉALISATION DES TESTS • BILAN

Les techniques que nous avons décidé de suivre pour tester notre programme sont la méthodologie des *tests aux limites* et la méthodologie des *tests par partitions*. Pour rappel, la méthodologie des tests aux limites consiste à concevoir des cas de test en se basant sur les valeurs limites. La méthodologie des tests par partitions, quant à elle, découpe les tests en partitions et crée des classes d'équivalence, qui sont des ensembles de valeurs traitées de la même manière.

==> Nous allons prendre en exemple le test de *ControlDeplacerPion* en détaillant les cas de test.

Cas de test	Concretisation	Valeur attendu
Déplacement en avant sans dépasser le plateau	positionInitiale = 10 deplacement = 5	Position attendue: 15
Déplacement en arrière sans descendre en dessous de 1	positionInitiale = 10 deplacement = -3	Position attendue: 7
Déplacement en avant avec dépassement du plateau	positionInitiale = 30 deplacement = 10	Position attendue: 36
Déplacement en arrière avec descente en dessous de 1	positionInitiale = 5, deplacement = -10	Position attendue: 1
Déplacement en arrière exactement à la limite inférieure	positionInitiale = 3, deplacement = -2	Position attendue: 1

 Tests aux limites

Cas de test	Concretisation	Valeur attendu
Test de limite minimale	positionInitiale = 1 deplacement = -1	Position attendue: 1
Frontière au début du tableau	positionInitiale = 1 deplacement = -5	Position attendue: 1
Test de limite maximale	positionInitiale = 35 deplacement = 1	Position attendue: 36
Limite à l'extrémité du tableau	positionInitiale = 35 deplacement = 5	Position attendue: 36

 Tests des partitions

Afin de réaliser les tests sur notre jeu, nous avons décidé de créer des scripts lisant des entrées et des oracles dans des fichiers externes, comme nous l'avons étudié lors de nos travaux pratiques (TP). Chaque contrôleur dispose donc de son propre script de test, accompagné d'un fichier texte contenant les entrées et les oracles pour chaque test. Le script lit les résultats attendus ainsi que les arguments de la fonction à exécuter pour lancer le test, puis affiche les résultats des tests en indiquant s'ils ont réussi ou échoué.

Pour le cas de notre *ControlSlotMachine*, nous avons créé un second constructeur comme suit :

```
public class ControlSlotMachine implements IControlSlotMachine {
    private final Random random;
    private int compteurSpin = 0;
    private IDialogue dialogue;

    public ControlSlotMachine(IDialogue dialogue) {
        this.dialogue = dialogue;
        this.random = new Random();
    }

    //Overload pour les tests
    public ControlSlotMachine(IDialogue dialogue, Random random) {
        this.dialogue = dialogue;
        this.random = random;
    }
}
```

L'intérêt de cette surcharge du contrôleur est de nous permettre d'initialiser, dans nos cas de test, un *ControlSlotMachine* avec un générateur de nombres aléatoires personnalisé. Cela nous permet de contrôler les valeurs générées par la méthode *randNextInt()*. Ainsi, nous pouvons lancer des cas de test avec des entrées et des oracles spécifiques afin de vérifier le bon fonctionnement de notre contrôleur.

```
Random fauxRandom = new Random() {
    private int callCount = 0;
    @Override
    public int nextInt(int bound) {
        int result;
        if (callCount == 0) {
            result = spinValuesInitiales[callCount];
        } else {
            result = spinValuesInitiales[callCount] - 1;
        }
        callCount++;
        return result;
    }
}
```

Nous utilisons ce système de *ControlSlotMachine* modifiée dans tous les contrôleurs qui utilisent une *ControlSlotMachine*, notamment pour les contrôleurs des cases spéciales.

PARTIE DU PROJET TESTÉ • PRÉSENTATION TECHNIQUE • RÉALISATION DES TESTS • BILAN

Nous avons réalisé des tests sur les contrôleurs de notre jeu en suivant une approche basée sur des scripts de test et des fichiers d'entrées/oracles. Chaque contrôleur dispose de son propre script de test et d'un fichier texte contenant les données nécessaires pour chaque cas de test. Les résultats ont été analysés pour évaluer la conformité du comportement des contrôleurs par rapport aux attentes.

Les tests ont été conçus pour couvrir les principales fonctionnalités des contrôleurs, notamment le déplacement des pions, la génération de valeurs pour la machine à sous, la gestion des cases spéciales telles que la possibilité de rejouer un tour, reculer ou jouer un duel de "gambling" contre le robot. La portée des tests s'étend donc à l'ensemble des fonctionnalités essentielles du jeu.

Les tests réalisés sont hautement réutilisables grâce à l'approche modulaire adoptée. Chaque contrôleur dispose de son propre script de test et de son fichier d'entrées/oracles, ce qui permet de les exécuter séparément et de les réutiliser facilement lors de modifications ultérieures du code. De plus, la surcharge du constructeur pour le contrôleur `ControlSlotMachine` afin de permettre l'utilisation d'un générateur de nombres aléatoires

personnalisé renforce la réutilisabilité des tests en offrant une flexibilité accrue lors de la validation du comportement de ce contrôleur dans différentes situations.

Pour atteindre un niveau de test encore plus complet, il serait pertinent d'envisager la validation de la partie interface utilisateur (UI) de notre code. Actuellement, cette partie n'est pas du tout testée, principalement parce qu'il y a peu d'interactions directes avec l'utilisateur dans notre jeu. Cependant, étant donné le nombre d'animations et l'utilisation de threads dans notre application, la validation de l'UI pourrait être importante. Les tests d'interface utilisateur pourraient révéler des problèmes potentiels liés à l'affichage des éléments graphiques, à la synchronisation des animations ou à d'autres aspects liés à l'expérience utilisateur.

En résumé, les tests réalisés offrent une bonne couverture des fonctionnalités des contrôleurs, tout en étant hautement réutilisables grâce à une approche modulaire et à la mise en place de structures facilitant les tests dans divers scénarios.