TELEMIS

Exercice de Développement Java/Web

Bowling Antique

Application Web RESTful

Technologies utilisées :

Java 17 • Spring Boot • Maven • JUnit 5
REST API • Swagger/OpenAPI • MVC Pattern

Auteur : Guillaume Alber

Date : 22 mai 2025

Table des matières

1 Introduction **2**
1.1 Contexte du Projet . . . 2
1.2 Objectifs Pédagogiques . . . 2

2 Spécifications du Jeu **2**
2.1 Règles Principales . . . 2
2.2 Système de Scoring . . . 2
2.3 Règles Spéciales . . . 2

3 Architecture Logicielle **3**
3.1 Phase 1 : Moteur de Jeu Core . . . 3
3.2 Diagramme UML . . . 3
3.2.1 Classe Game . . . 3
3.2.2 Classe Player . . . 3
3.2.3 Classe Frame . . . 3
3.3 Difficultés . . . 4
3.4 Phase 2 : Extension Web . . . 4

4 API REST **4**
4.1 Endpoints Disponibles . . . 4
4.2 Documentation Interactive . . . 4

5 Tests et Qualité **5**
5.1 Stratégie de Test . . . 5
5.2 Exemple de Test . . . 5

6 Technologies et Outils **5**
6.1 Stack Technique . . . 5
6.2 Justification des Choix . . . 5

7 Installation et Démarrage **6**
7.1 Instructions de Démarrage . . . 6
7.2 Accès à l'Application . . . 6

8 Évolutions possibles **6**
8.1 Améliorations Techniques . . . 6

9 Conclusion **6**

1 Introduction

Ce document présente l'implémentation d'un jeu de bowling antique fictif dans le cadre d'un exercice de développement pour Telemis. Le projet démontre une approche méthodique du développement logiciel, alliant les principes fondamentaux de la programmation orientée objet à des technologies web modernes sous Java.

1.1 Contexte du Projet

L'application simule un jeu de bowling inspiré de découvertes archéologiques africaines fictives, offrant une extrapolation du bowling traditionnel avec certaines règles spécifiques. Le développement s'articule autour de deux phases principales :

1. **Phase 1 :** Développement du moteur de jeu en Java pur
2. **Phase 2 :** Extension web avec API REST utilisant Spring Boot

1.2 Objectifs Pédagogiques

Ce projet vise à démontrer :

— La maîtrise des concepts de programmation orientée objet
— L'implémentation d'une architecture logicielle claire et maintenable
— L'utilisation des frameworks Java modernes
— Les bonnes pratiques de développement et de test

2 Spécifications du Jeu

2.1 Règles Principales

Le bowling antique se distingue du bowling traditionnel par plusieurs caractéristiques uniques :

Configuration de Base

— **Frames par joueur :** 5
— **Lancers par frame :** 3 maximum sauf pour la derniere f rame
— **Nombre de quilles :** 15
— **Joueurs minimum :** 2

2.2 Système de Scoring

Le calcul des points suit une logique spécifique adaptée aux règles du jeu antique :

Type de Frame	Calcul des Points	Maximum
Frame "Normale"	Somme des quilles abattues	14 points
Spare	15 + 2 lancers suivants	45 points
Strike	15 + 3 lancers suivants	60 points

TABLE 1 – Système de scoring du bowling antique

Score maximum théorique : 300 points par joueur

2.3 Règles Spéciales

— **Dernière frame :** Bonus de lancers supplémentaires en cas de strike ou spare
— **Maximum de lancers :** 4 lancers possibles dans la dernière frame
— **Validation :** Contrôles stricts sur le nombre de quilles abattues

3 Architecture Logicielle

L'architecture suit une approche en couches respectant les principes SOLID et les design patterns reconnus :

Patterns Utilisés

- **Singleton Pattern :** Initialement utilisé pour la gestion global du jeu, mais abandonné afin de permettre la gestion de plusieurs instances dans lors de l'extension web.
- **MVC Pattern :** Séparation des responsabilités dans la couche web

3.1 Phase 1 : Moteur de Jeu Core

Le cœur de l'application repose sur trois classes principales formant la logique metier :

3.2 Diagramme UML

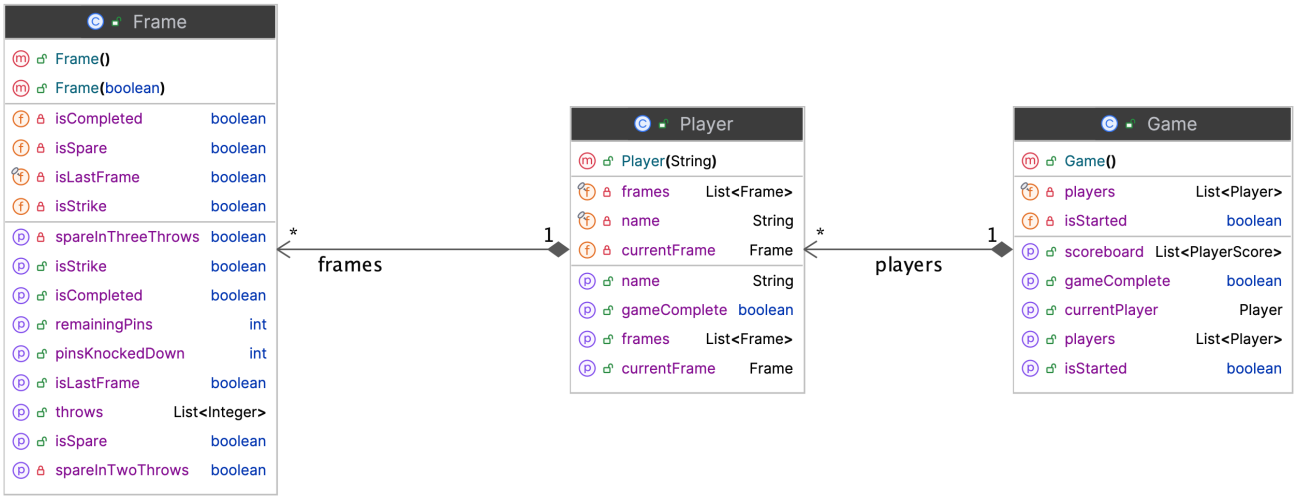

FIGURE 1 – Diagramme UML de l'architecture du moteur de jeu

3.2.1 Classe Game

Responsabilités :

- Gestion de l'état global du jeu
- Coordination des tours de jeu
- Validation des actions des joueurs

3.2.2 Classe Player

Responsabilités :

- Gestion des frames individuelles
- Calcul du score total
- Validation des lancers
- Gestions des

3.2.3 Classe Frame

Responsabilités :

- Gestion des lancers individuels
- Détection des strikes et spares

3.3 Difficultés

La principale difficulté a été la gestion de la dernière frame, plus complexe que les autres du fait des règles de bonus. En cas de strike ou de spare (y compris obtenu en trois lancers), le joueur peut bénéficier de jusqu'à cinq lancers. Il fallait éviter de passer au joueur suivant trop tôt et réinitialiser les quilles au bon moment. Cela explique la complexité de la méthode `getRemainingPins()`, qui centralise cette logique particulière.

3.4 Phase 2 : Extension Web

L'extension web suit l'architecture MVC de Spring Boot :

Couche Controller

- Expose nos endpoints REST
- Reçoit et transmet les requêtes au service
- Retourne les réponses HTTP/JSON

Couche Service

- Fait le lien avec le moteur de jeu
- Valide les données métier
- Gère les exceptions applicatives

Couche Modèle

- Contient les entités métier (Game, Player, Frame)
- Encapsule la logique métier centrale
- Ne dépend d'aucune technologie Spring

4 API REST

4.1 Endpoints Disponibles

L'API REST offre une interface complète pour interagir avec le jeu :

Méthode	Endpoint	Description
POST	`/api/games`	Création d'une nouvelle partie
GET	`/api/games/{gameID}`	Récupération de l'état du jeu
POST	`/api/games/{gameID}/throw`	Soumission d'un lancer
GET	`/api/games/{gameID}/scoreboard`	Récupération du scoreboard

Table 2 – Endpoints de l'API REST

4.2 Documentation Interactive

L'API est documentée avec Swagger/OpenAPI, accessible via le serveur web une fois le projet compiler et executer disponible sur le port par default 8080 :

`http://localhost:8080/swagger-ui.html`

5 Tests et Qualité

5.1 Stratégie de Test

La stratégie de test couvre plusieurs niveaux :

Couverture des Tests

- **Tests Unitaires :** JUnit 5 pour le moteur de jeu
- **Tests d'Intégration :** Spring Boot Test pour l'API
- **Tests de Validation :** Mockito pour les composants web
- **Tests de Cas Limites :** Strikes, spares, dernière frame

5.2 Exemple de Test

```
@Test
  void testStrikeScore() {
      System.out.println("DEBUG: Start testStrikeScore");
      // First frame - Strike
      player.addThrow(15);
      player.addThrow(3);
      player.addThrow(4);
      player.addThrow(5);
      assertEquals(27, player.calculateScore(1)); // should be: 15 + (3 + 4 + 5) = 27
      assertEquals(39, player.calculateScore(2)); // should be: scoreFrame1 + (3 + 4 +
          5) = 39
      System.out.println("DEBUG: End testStrikeScore");
  }
```

Listing 1 – Exemple de test unitaire issu de PlayerTest

6 Technologies et Outils

6.1 Stack Technique

Composant	Technologie/Version
Langage	Java 17
Framework Web	Spring Boot 3.2.3
Build Tool	Maven
Tests	JUnit 5.10.2 + Mockito
Documentation API	Swagger/OpenAPI

TABLE 3 – Stack technique utilisée

6.2 Justification des Choix

- **Java 17 :** Fonctionnalités modernes (records, pattern matching)
- **Spring Boot :** Écosystème mature et productif
- **Maven :** Gestion simplifiée des dépendances
- **JUnit 5 :** Framework de test de référence

7 Installation et Démarrage

7.1 Instructions de Démarrage

```
# Cloner le repository ou telecharger le zi
git clone [url Repo]

# Se placer dans le r pertoire du projet
cd telemis-bowling

# Compiler le projet
mvn clean install

# Lancer l'application
mvn spring-boot:run
```

Listing 2 – Procédure d'installation

7.2 Accès à l'Application

Une fois démarrée, l'application est accessible aux adresses suivantes :

- **Application web :** `http://localhost:8080`
- **Documentation API :** `http://localhost:8080/swagger-ui.html`

8 Évolutions possibles

8.1 Améliorations Techniques

1. **Communication Temps Réel**
   - Intégration WebSocket pour une vrai gestion du multijoueur
2. **Interface Utilisateur SPA**
   - Développement d'un frontend React/Vue.js
   - Interface responsive et intuitive
   - Gestion d'état côté client
3. **Persistance des Données**
   - Intégration base de données (H2/PostgreSQL) pour la persistance des scores
   - Gestion des profils utilisateurs
   - Historique et statistiques des parties

9 Conclusion

Ce projet illustre une démarche structurée, en combinant les bases de la programmation orientée objet avec une mise en œuvre web via Spring Boot. L'architecture claire et les tests unitaires assurent une bonne maintenabilité et facilitent les évolutions.

J'ai veillé à respecter les bonnes pratiques Java et Spring, tout en gardant le code lisible et modulaire. Ce travail constitue une fondation stable pour aller plus loin. J'aurais aimé poursuivre certaines fonctionnalités, notamment l'intégration de React avec Axios pour améliorer l'expérience côté client.

Livrable : Application web fonctionnelle avec API REST documentée, tests unitaires complets, et architecture extensible respectant les standards de l'industrie.