Documentation du Jeu Ghost Run 2D

Introduction

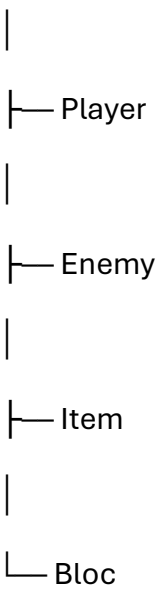
Ghost Run 2D est un jeu de plateforme développé en Java en utilisant le framework LibGDX. Il propose des fonctionnalités classiques comme le déplacement, le saut, la collecte de pièces, l'interaction avec des ennemis, et des transitions entre différents états (Game Over, Victoire). Cette documentation décrit en détail les classes principales du projet et leurs relations.

Structure des Classes

Voici la structure des classes avec leurs héritages et leurs relations :

Diagramme UML Simplifié

Entity (Classe abstraite)



├── Player

├── Enemy

├── Item

└── Bloc

Description des Classes

Classe Abstraite : Entity

Description : La classe Entity est une classe abstraite qui représente tous les objets du jeu ayant une position, une texture, et des interactions potentielles.

Attributs principaux :

- float x, y : Position de l'entité.
- float velocityX, velocityY : Vitesse de l'entité.
- Texture texture : Texture de l'entité pour l'affichage.

Méthodes :

- update(float deltaTime) : Mise à jour de la position de l'entité.
- render(SpriteBatch batch) : Rendu graphique de l'entité.
- getBoundingRectangle() : Renvoie la zone de collision de l'entité.

Classe : Player

Hérite de : Entity

Description :

Représente le joueur (Mario) dans le jeu. Il peut se déplacer, sauter, collecter des objets, et interagir avec des ennemis.

Attributs spécifiques :

- boolean onGround : Indique si le joueur est au sol.
- int coins, health, score : Statistiques du joueur.
- boolean isDead : Indique si le joueur est mort.

Méthodes :

- update(float deltaTime) : Gère les déplacements et les interactions du joueur.
- checkCollisionWithItem(Item item) : Vérifie et applique les effets des objets.
- checkCollisionWithEnemy(Enemy enemy) : Gère les collisions avec les ennemis.
- checkCollisionWithBloc(Bloc bloc) : Gère les interactions avec les blocs.
- checkCollisionWithPipe(Rectangle pipeBounds) : Gère les interactions avec les tuyaux.
- adjustHealth(int amount) : Modifie les points de vie du joueur.
- die() : Définit l'état de mort du joueur.

Classe : Enemy

Hérite de : Entity

Description :

Représente les ennemis dans le jeu. Les ennemis suivent des patterns de mouvement et interagissent avec le joueur.

Attributs spécifiques :

- String type : Type de l'ennemi (ex : Tortue, Champignon).
- int health : Points de vie de l'ennemi.
- List<Float> movementPattern : Décrit le pattern de déplacement.

Méthodes :

- update(float deltaTime) : Gère les déplacements selon le pattern.
- checkCollisionWithPipe(Rectangle pipeBounds) : Gère les interactions avec les tuyaux.
- takeDamage(int amount) : Réduit la vie de l'ennemi.

Classe : Item

Hérite de : Entity

Description :

Représente les objets que le joueur peut collecter, comme les pièces.

Attributs spécifiques :

- ItemType type : Type d'objet (ex : Coin).

Méthodes :

- applyEffect(Player player) : Applique un effet spécifique au joueur (ajout de score, santé, etc.).

Classe : Bloc

Hérite de : Entity

Description :

Représente les blocs dans le jeu qui peuvent interagir avec le joueur (collisions).

Attributs spécifiques :

- Aucun attribut spécifique.

Méthodes :

- Aucune méthode spécifique au-delà de celles héritées de Entity.

Relations Entre les Classes

1. **Player Interagit avec :**
    a. Item (pour collecter des objets et augmenter les statistiques).
    b. Enemy (pour infliger ou subir des dégâts).
    c. Bloc (pour détecter les collisions et permettre le déplacement).
    d. Rectangle (représentant les tuyaux dans le jeu).
2. **Enemy Interagit avec :**
    a. Rectangle (les tuyaux pour restreindre leurs déplacements).
3. **Item Interagit avec :**
    a. Player (pour appliquer des effets).

Architecture Technique

1. **Framework LibGDX :**
    a. Gère le cycle de vie du jeu : create, render, et dispose.
    b. Utilise SpriteBatch pour le rendu des entités.
    c. La classe principale (Main) coordonne les interactions et le rendu.
2. **Gestion des Collisions :**
    a. Utilisation de Rectangle pour détecter les collisions entre entités.
3. **Design Responsive :**
    a. Caméra (OrthographicCamera) ajustée pour s'adapter dynamiquement aux différentes tailles d'écran.

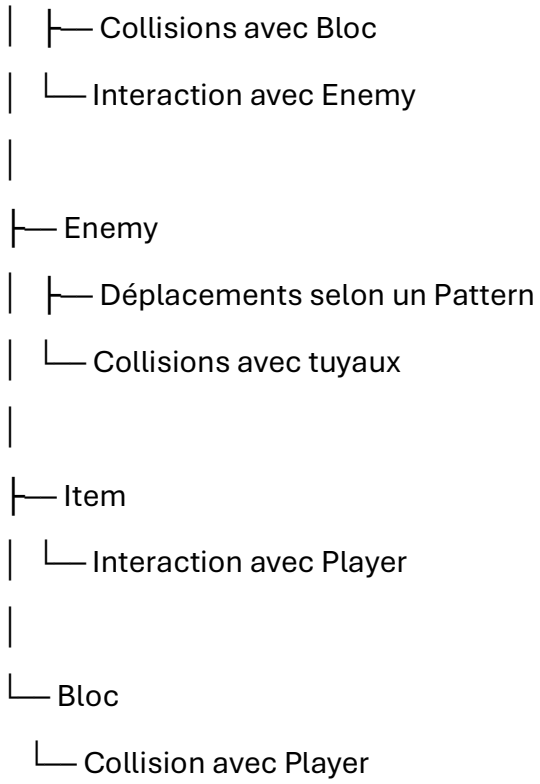
Diagramme Simplifié

Main (Point d'entrée)

│

├── Player

│ ├── Collecte d'Item



Conclusion

Le jeu Ghost Run 2D repose sur une architecture modulaire et extensible, facilitant l'ajout de nouvelles fonctionnalités (niveaux, ennemis, objets). Les principes de programmation orientée objet (héritage, encapsulation) ont été appliqués pour maintenir un code structuré et maintenable.

Ce projet met en avant des compétences clés en développement logiciel et démontre une bonne compréhension des mécaniques de jeu.