Rapport de Projet
Développement d'un Moteur de Jeu 2D avec LibGDX

PCOO_Jeu2D

Daniel Carriba Nosrati

Équipe et Contributions

- Daniel CARRIBA NOSRATI [développeur du projet]

Section 1. Introduction

Ce projet a été réalisée pour l'UE « Programmation et conception orientée objet » du 5e semestre de la Licence Informatique de l'Université Côte d'Azur. Il s'agit d'un moteur de jeu RPG en 2D programmé en Java. Il est extensible, et donc possible d'ajouter de nouveau contenu et de nouvelles fonctionnalités sans modifier le code Java existant.

Section 2. Présentation du projet

Technologies et Outils Utilisés

- LibGDX : pour le développement du moteur de jeu. La version 1.13.0.0 de LibGDX a été utilisée.
- Tiled : pour la création et la configuration des cartes.
- Gradle : pour construire (compiler et exécuter) le projet.
- IntelliJ IDEA : IDE utilisé pour programmer.

Fonctionnalités implémentées

- En lançant le programme, l'utilisateur a le choix de créer une nouvelle partie en appuyant avec la souris sur le bouton « New Game », ou de charger la dernière partie sauvegardé en appuyant sur le second bouton « Load Game ».
- Si aucune sauvegarde n'est trouvée, un message « No saved game found! » est affiché, et l'utilisateur peut uniquement commencer une nouvelle partie.
- Une fois commencé, le jeu est automatiquement sauvegardé à la fermeture du jeu, dans le fichier « game_save.ser » situé dans le dossier « assets ».
- Le joueur peut se déplacer de tuile en tuile, vers le haut, la droite, la gauche et le bas avec les touches « fléches » du clavier.
- Le joueur peut courir en restant appuyé sur la touche « Left Shift » (la touche majuscule gauche).
- Le joueur ne peut ni traverser des obstacles (les maisons, les arbres, les décorations, etc.), ni traverser des ennemis.
- Le joueur peut tuer un ennemi à côté de lui, en le regardant et en appuyant sur la barre d'espace.
- Le joueur peut entrer dans les maisons ayant une porte. Une porte est représentée grâce aux tuiles suivantes :
   Le joueur doit se déplacer la droite, la gauche ou vers le bas pour entrer dans la maison
   Le joueur doit se déplacer vers le haut pour entrer dans la maison
- Lorsque le joueur entre dans une maison, une nouvelle carte (TiledMap) représentant l'intérieur de la maison est chargée et affiché.

- En commençant une nouvelle partie, le joueur commence sur la carte « city.tmx »
- Pour changer de carte, le joueur entre dans une des maisons situées au nord, à l'est, au sud, ou à l'ouest de la ville, et en sortant de la maison par la seconde porte, le joueur se retrouve sur les cartes « forest.tmx », « ocean.tmx », « beach.tmx » et « mountain.tmx » respectivement.
- Plusieurs ennemis sont présents sur toutes les cartes, et lorsque le joueur tue un ennemi, celui-ci disparaît et le compteur de « Enemies killed » affiché sur l'écran (le HUD) s'incrémente.
- Ce compteur affiche le nombre total d'ennemis tués sans faire de distinction de la carte.
- Un ennemi tué n'est plus jamais affiché sur la carte, même si le joueur quitte une carte et revient plus tard.

Configuration et Ajout de Contenu avec Tiled

Pour créer de nouvelles cartes il faut utiliser le logiciel Tiled.

Les cartes seront sauvegardées dans le dossier « assets/worlds » et les tilesets utilisables sont « exterior.tsx » et « interior.tsx » dans le dossier « assets/tilesets/worldtiles ».

Une fois la carte créer dans le bon dossier (« assets/worlds ») les tuiles peuvent être dessiner sur la carte. La carte devra avoir les couches suivantes (de la couche supérieure à la couche inférieure):

- enemies (couche d'objet)
- teleport (couche d'objet)
- obstacles2 (couche de tuile)
- obstacles (couche de tuile)
- ground (couche de tuile)

Les tuiles où le joueur pourra se déplacer doivent être dessiné sur la couche « ground ».

Les tuiles où le joueur est interdit de marcher doivent être dessiné sur la couche « obstacles ».

La couche « obstacles2 » est uniquement utilisé pour dessiner des tuiles au-dessus d'autres tuiles de la couche « obstacles » (pour apporter plus de détails graphiques par exemples).

Sur la couche « teleport », des objets rectangulaire (bouton « Insert Rectangle ») peuvent être placé. Ces objets sont utilisés pour que le joueur puisse changer de carte. Ces objets doivent avoir la taille d'une ou de plusieurs tuiles (utiliser la touche « contrôle » pour que les rectangles dessinés s'adaptent aux tuiles). Ces objets devront également avoir les propriétés suivantes :

- « nextWorldMapName » de type String, ayant pour valeur le nom de la prochaine carte (par ex : « house_medium.tmx »).
- « nextWorldMapTileX » de type int, ayant pour valeur la coordonné X de la tuile de la carte référencé dans « nextWorldMapName », où le joueur sera placé après téléportation.
- « nextWorldMapTileY » de type int, ayant pour valeur la coordonné Y de la tuile de la carte référencé dans « nextWorldMapName », où le joueur sera placé après téléportation.
- « whenKeyPressed » de type String, ayant pour valeur « PlayerInputKeyUp », « PlayerInputKeyRight », « PlayerInputKeyDown » ou « PlayerInputKeyLeft ». Si la valeur est « PlayerInputKeyUp » alors le joueur devra aller vers le haut pour être téléporter vers la prochaine carte. Le principe est de même pour les autres valeurs. Aucune autre valeur différente de ces quatre sera autorisé.

Les coordonnées X et Y des tuiles est affiché en bas à droite dans le logiciel. La tuile de coordonnée (0, 0) se situe en haut à gauche de la TileMap.

Sur la couche « enemies », des objets rectangulaire (bouton « Insert Rectangle ») peuvent être placé. Ces objets représentent les endroits où les ennemis seront placés au chargement de la carte. Ces objets doivent avoir la taille d'exactement une tuile (utiliser la touche « contrôle » pour que les rectangles dessinés s'adaptent à la tuile). Ces objets doivent également avoir les propriétés suivantes :

- « spriteSheetImageName » de type String, ayant pour valeur le nom de l'image du spritesheet contenant les graphismes de l'ennemi (par ex : « orcs_1.png », ou tout autre image dans le dossier « assets/tilesets/characterstiles »)

Pour ajouter des propriétés à des objets : faire clic droit sur l'objet en question, cliquer sur « Object Properties… », cliquer le « + (Add Property) » et choisir le type ainsi que le nom.

Tout objets des couches « teleport » et « enemies » ayant des propriétés erronés, manquantes ou vides ne seront pas reconnus, et ignorés par le moteur de jeu.

Compilation et exécution

1) Prérequis :

Pour compiler et exécuter le projet, voici les prérequis nécessaires :

- Java 11 au minimum (J'ai personnellement utilisé Java 21)
- LibGDX 1.13.0.0 a été utilisé pour le moteur de jeu
- Gradle est utilisé pour compiler et exécuter le projet
- S'assurer que le fichier « gradlew » a les droits pour être exécuté

2) Étapes de Compilation :

Pour compiler, exécuter la commande suivante :

```
./gradlew build
```

3) Lancement du Jeu :

Pour lancer le jeu, exécuter la commande suivante :

```
./gradlew run
```

4) [Optionnel] Construire le jar exécutable :

Pour construire le jar exécutable, exécuter la commande suivante :

```
./gradlew jar
```

Le jar exécutable sera dans le dossier lwjgl3/build/libs/

Il suffit d'exécuter le jar et le jeu se lancera

5) [Optionnel] Supprimer les dossier « build » :

Pour supprimer les dossier « build », exécuter la commande suivante :

```
./gradlew clean
```

6) [Remarque] Vous pouvez également utiliser `./gradlew.bat` au lieu de `./gradlew`

Les instructions pour compiler et exécuter sont également dans le fichier README.md

Dépôt GitHub

Pour accéder au dépôt GitHub : https://github.com/dcarriba/PCOO_Jeu2D

Pour cloner le projet utiliser la commande suivante :

```
git clone https://github.com/dcarriba/PCOO_Jeu2D.git
```

Ensuite vous pourrez compiler et exécuter le projet.

Section 3. Présentation technique du projet et contributions

Architecture Générale du moteur de jeu

Le code source Java est situé dans le fichier core/src/main/java/

Il est structuré en plusieurs packages :

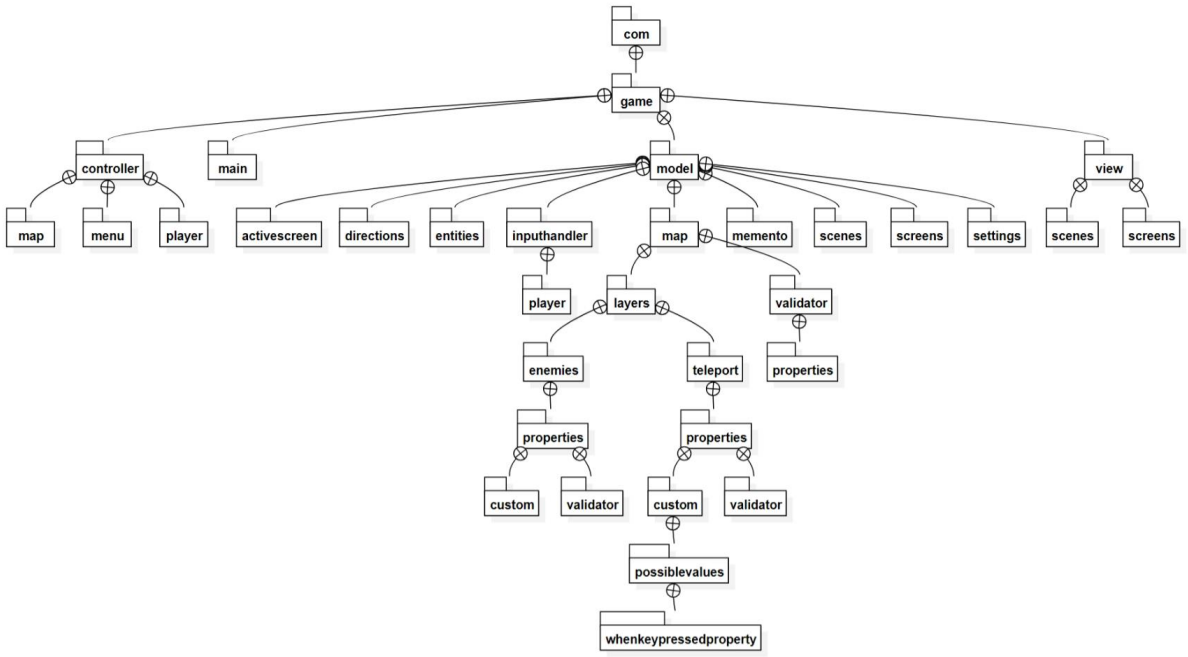
Le package com.game.main contenant la classe principale du jeu : Mygame

Le package com.game.model contenant plusieurs sous-packages, contenant chacun des classes d'Objets nécessaire pour le fonctionnement du jeu

Le package com.game.view contenant plusieurs sous-packages, contenant chacun des classes qui s'occupe de l'affichage des modèles.

Le package com.game.controller contenant plusieurs sous-packages, contenant chacun des classes qui réagissent à des événements de l'utilisateur et qui mets à jour le jeu en conséquent

Organisation des packages et sous-packages (sans les classes) :



Détails :

Le package com.game.model contient :

- com.game.model.activescreen :
    - La classe ActiveScreen, qui représente l'écran actuellement affiché. Cette classe prend un objet Screen et un objet ScreenView.
- com.game.model.screens :
    - L'interface Screen, qui définit un écran.
    - La classe PlayScreen, qui implémente Screen et qui représente l'écran du jeu. Un objet PlayScreen nécessite un objet Player et WorldMap.
    - La classe MenuScreen, qui implémente Screen et qui représente l'écran de menu affiché au début du jeu, où l'utilisateur peut reprendre la dernière sauvegarde ou commencer une nouvelle partie.
- com.game.model.scenes :
    - La classe Hud, qui représente le HUD affiché sur le PlayScreen.
- com.game.model.map :
    - La classe WorldMap, qui représente la carte où le joueur se déplace. L'objet WorldMap contient un objet TiledMap ainsi que des objets Enemy.
    - com.game.model.validator.properties :
        - L'interface PropertiesValidator, qui définit un validateur de propriétés d'objets, d'une TiledMap.
        - La classe PresencePropertiesValidator, qui implémente PropertiesValidator et qui valide si les propriétés d'un objet de la TiledMap sont présentes ou pas.
        - La classe NotNullPropertiesValidator, qui implémente PropertiesValidator et qui valide si les propriétés d'un objet de la TiledMap ont des valeurs non null et non vides.
    - com.game.model.map.layers :
        - com.game.model.map.layers.enemies.properties.custom contient la classe SpriteSheetImageName, qui représente la propriété « SpriteSheetImageName » qu'un objet de la couche d'objets « enemies » peut avoir.
        - com.game.model.map.layers.enemies.properties.validator contient le validator ValidateCustomPropertiesEnemies qui valide que les propriétés d'un objet de la couche d'objets « enemies » sont valides.
        - com.game.model.map.layers.teleport.properties.custom contient les classes NextWorldMapNameProperty, NextWorldMapNameTileX, NextWorldMapNameTileY et WhenKeyPressedProperty, qui représentent les propriétés de même nom, qu'un objet de la couche d'objets « teleport » peut avoir.
        - com.game.model.map.layers.teleport.properties.custom.possiblevalues contient les valeurs que la précédente propriété WhenKeyPressedProperty est autorisé d'avoir.
        - com.game.model.map.layers.teleport.properties.validator contient le validator ValidateCustomPropertiesTeleport qui valide que les propriétés d'un objet de la couche d'objets « teleport » sont valides.
- com.game.model.entities :

    - La classe abstraites Entity qui représente une entité (i.e. joueur, ennemi) et qui a besoins des objets suivants : EntityAnimation, WorldMap et SpriteSheet.
    - La classe Player qui étend Entity et qui représente le joueur.
    - La classe Enemy qui étend Entity et qui représente un ennemi.
    - La classe EntityAnimation qui crée l'animation de marche d'une entité. Cette classe nécessite d'un objet de type Direction ainsi qu'un objet SpriteSheet.
    - La classe SpriteSheet qui représente les graphismes d'une entité et qui contient, à partir d'une image, un objet Texture.
    - La classe EnemyFactory qui récupère les objets de la couche d'objets « enemies » de la TiledMap, et qui y génère les ennemis correspondants.
- com.game.model.directions :
    - La classe abstraite Direction qui représente une direction dans laquelle une entité peut regarder et se déplacer.
    - Les classes DownDirection, LeftDirection, RightDirection et UpDirection, qui étendent la classe Direction et qui représentent la direction vers le bas, la gauche, la droite et le haut respectivement.
- com.game.model.inputhandler.player :
    - La classe abstraite PlayerInputKey, qui représente une touche clavier pour le joueur ainsi que l'action associé.
    - Les classes PlayerInputKeyAttack, PlayerInputKeyUp, PlayerInputKeyDown, PlayerInputKeyLeft, PlayerInputKeyRight et PlayerInputKeySprint qui étendent la classe PlayerInputKey et qui représentent chacune une touche et l'action associé.
    - La classe PlayerInputHandler, qui contient la méthode handleInput() exécutant l'action associé d'un PlayerInputKey lorsque sa touche est appuyé sur le clavier.
- com.game.model.mememto :
    - La classe GameMemento qui implémente l'interface Serializable, et qui est utilisé pour créer la sauvegarde du jeu. GameMemento sauvegarde ainsi ses attributs de type Player et WorldMap. Les classes Entity, EntityAnimation, SpriteSheet, Direction et WorldMap implémentent donc aussi l'interface Serializable, pour pouvoir sauvegarder l'ensembles des éléments nécessaires.
- com.game.model.settings :
    - La classe Settings, qui contient les réglages du jeu suivants : la hauteur et largeur de l'écran. Cette classe est utilisée dans les classes Mygame, PlayScreen et Hud, qui doivent accéder à ces paramètres.

Le package com.game.view contient :

- com.game.view.screens :
    - L'interface ScreenView, qui définit un ScreenView qui doit afficher un objet Screen avec une méthode render().
    - Les classes PlayScreenView et MenuScreenView qui implémentent l'interface ScreenView et qui affichent respectivement un objet PlayScreen et MenuScreen.
- com.game.view.scenes :
    - La classe HudView, qui affiche le HUD sur l'écran de jeu.

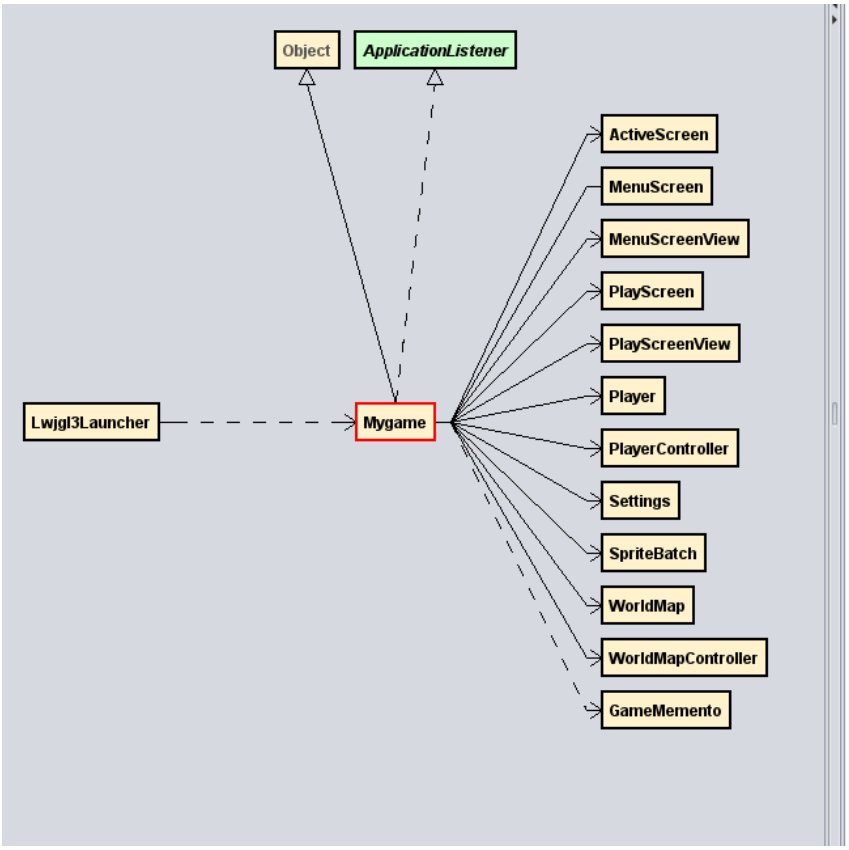
Le package com.game.controller contient :

- com.game.controller.map :

    - La classe WorldMapController, qui prend le PlayScreen, et qui change la TiledMap de la WorldMap, lorsque le joueur marche sur un objet de la couche d'objets « teleport ».
- com.game.controller.player :
    - La classe PlayerController, qui prend l'instance Player ainsi qu'une instance de PlayerInputHandler, et qui gère les déplacement du joueur en utilisant la méthode handleInput() du PlayerInputHandler.
- com.game.controller.menu :
    - La classe MenuScreenController, qui prend le MenuScreen et qui ajoute des Listeners au boutons du MenuScreen.

Le package com.game.main contient :

- La classe Mygame, qui est l'instance principale du jeu. Voici un Schéma UML détaillant les relations de la classe Mygame :



La classe Lwjgl3Launcher dans le package com.game.main.lwjgl3 dans le répertoire lwjgl3/src/main/java/ contient la fonction main() du projet, qui crée et exécute l'instance principale du jeu Mygame.

Utiliser et étendre la librairie du moteur de jeu

Le moteur de jeu peut être étendu en ajoutant de nouvelles classes.

Par exemple :

Une classe Npc qui étend la classe com.game.model.entities.Entity pour ajouter des Npc au jeu.

Des classes DiagonalLeftUpDirection, DiagonalLeftDownDirection, DiagonalRightUpDirection et DiagonalRightDownDirection qui étendent la classe com.game.model.directions.Direction pour ajouter des mouvements diagonaux .

Des classes InventoryScreen et InventoryScreenView qui implémentent com.game.model.screens.Screen et com.game.view.screens.ScreenView respectivement pour ajouter un écran d'inventaire.

Des classes qui étendent com.game.model.inputhandler.player.PlayerInputKey pour ajouter plus de contrôles pour le joueur.

D'autres classes Java peuvent également être ajoutés pour étendre le moteur de jeu.

Le jeu peut également être étendu avec de nouvelles cartes Tiled :

Il suffit que les cartes faites avec Tiled respectent les conditions citées auparavant (dans « Configuration et Ajout de Contenu avec Tiled » de la « Section 2. Présentation du projet »).

Répartition des Tâches

Le projet a été réalisé seul.

Section 4. Conclusion et Perspectives

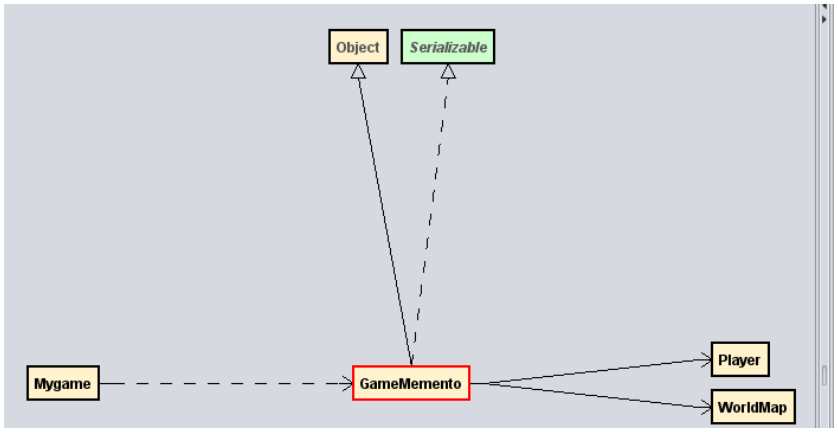
Pour conclure, ce projet est un jeu RPG en 2D créé en Java, qui est extensible avec de nouvelles classes Java, pour ajouter de nouvelles fonctionnalités, et qui est également extensible avec de nouvelles cartes Tiled.

Annexes

Voici des schémas UML supplémentaires des quelques classes suivantes : GameMemento, WorldMap, Entity, PlayerInputKey, PlayScreen, MenuScreen et ActiveScreen.

Ces schémas UML donnent plus de détails sur les relations que ces classes possèdent au sein de projet.

Ces schémas UML ont été crée grâce au logiciel « Class Visualizer » (https://www.class-visualizer.net/index.html).

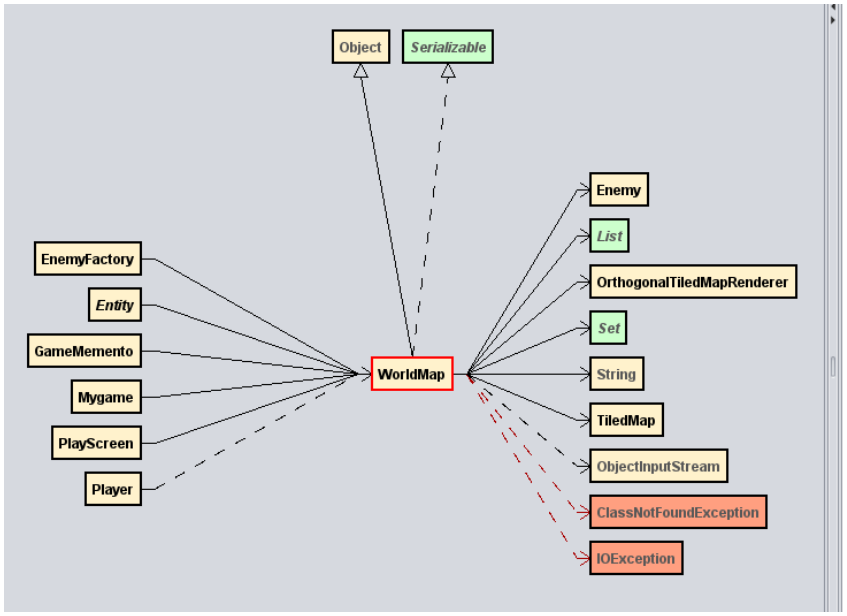


Object

com::game::model::memento::
GameMemento

Properties
«final» «readOnly»
+ player : Player
«final» «readOnly»
+ worldMap : WorldMap

Constructors
+ GameMemento(Player, WorldMap) : void



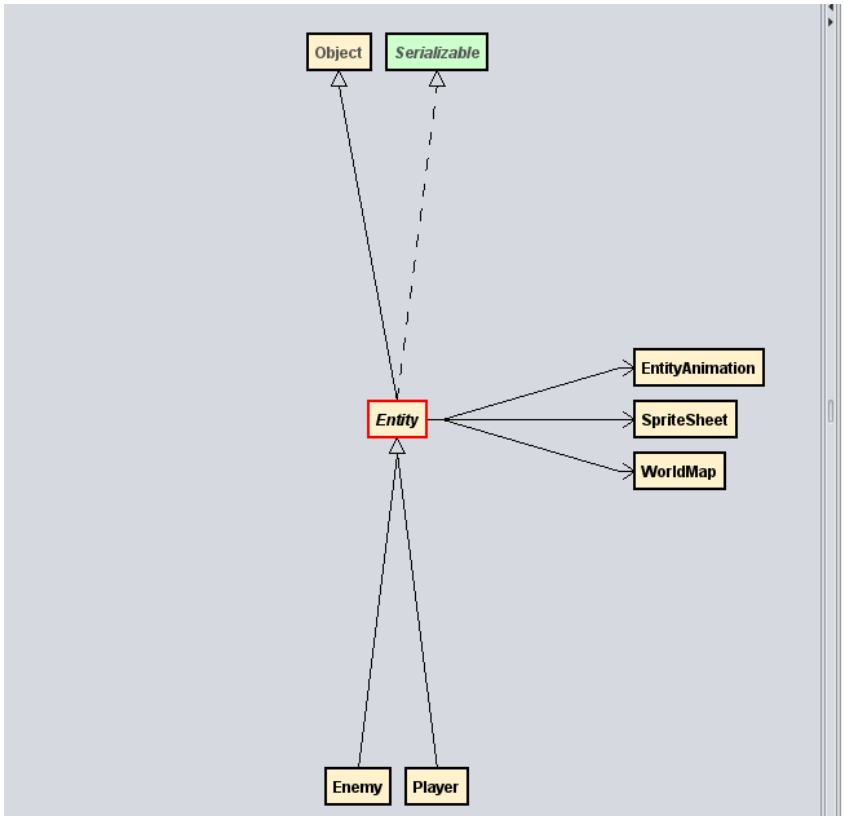
Object

com::game::model::map::
WorldMap

Properties
«readOnly»
+ enemies : List<Enemy>
«final» «readOnly»
+ killedEnemies : Set<String>
«transient» «readOnly»
+ renderer : OrthogonalTiledMapRenderer
«readOnly»
+ tileHeight : float
«readOnly»
+ tileWidth : float
«transient» «readOnly»
+ tiledMap : TiledMap
«readOnly»
+ tiledMapPath : String

Constructors
+ WorldMap(String, float, float) : void

Methods
+ dispose( ) : void
+ getEnemy(int, int) : Enemy
+ hasEnemyBeenKilled(Enemy) : boolean
+ isTileBlockedByEnemy(int, int) : boolean
+ isTileNotBlocked(int, int) : boolean
+ killEnemy(Enemy) : void
- readObject(ObjectInputStream) : void
+ updateWorldMap(String, float, float) : void



Object

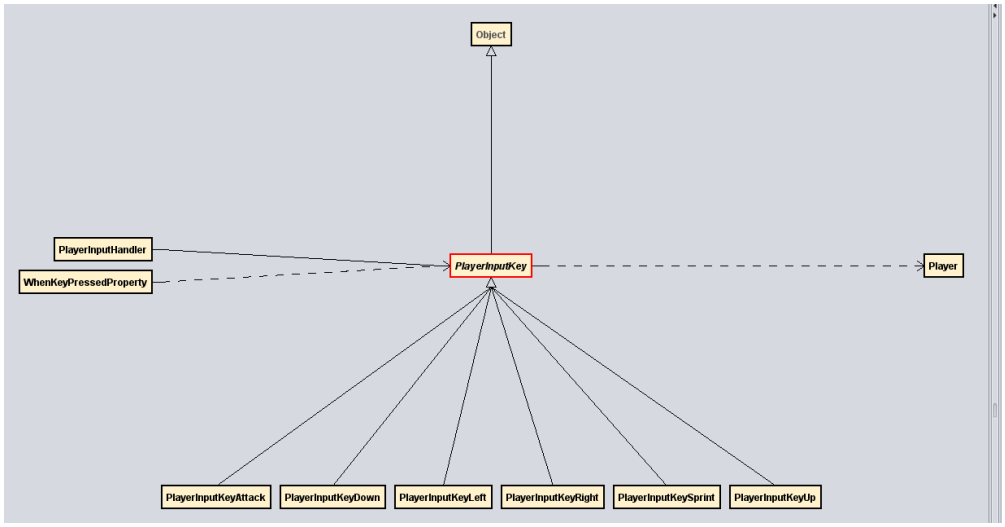
com::game::model::entities::
Entity

Fields
- isMoving : boolean

Properties
«final» «readOnly»
+ entityAnimation : EntityAnimation
+ moveSpeed : float
«final» «readOnly»
+ moveSpeedDefault : float
«readOnly»
+ moving : boolean
«readOnly»
+ notMoving : boolean
«readOnly»
+ positionX : float
«readOnly»
+ positionY : float
«final» «readOnly»
+ spriteSheet : SpriteSheet
+ targetX : float
+ targetY : float
+ tileX : int
+ tileY : int
«final» «readOnly»
+ worldMap : WorldMap

Constructors
+ Entity(int, int, SpriteSheet, WorldMap) : void

Methods
+ dispose( ) : void
- moveToNextTile( ) : void
+ setIsMoving(boolean) : void
+ setOnTile(int, int) : void
+ update( ) : void



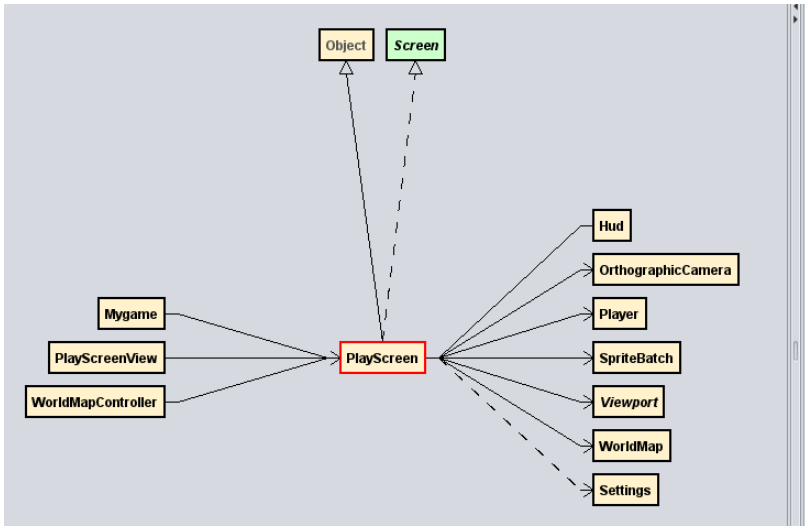
Object

com::game::model::inputhandler::player::
PlayerInputKey

Properties
+ key : int

Constructors
+ PlayerInputKey(int) : void

Methods
+ *action(Player) : void*



Object

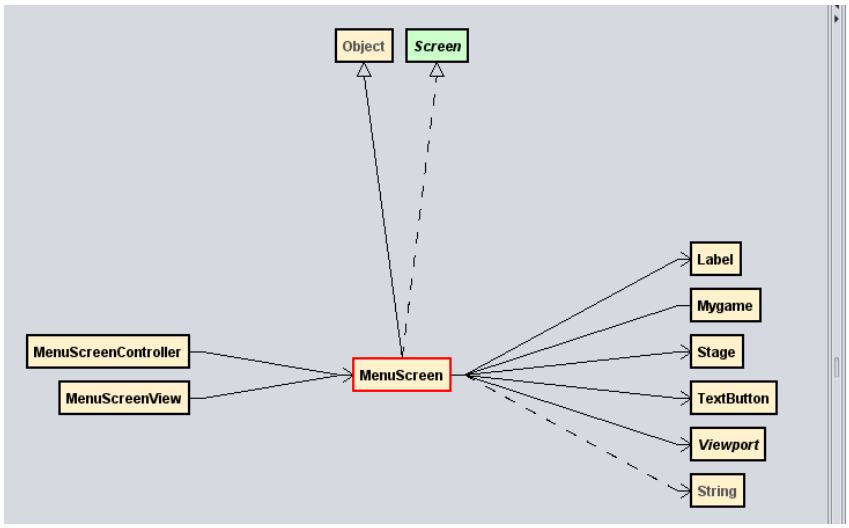
com::game::model::screens::
PlayScreen

Fields
«final»
- viewport : Viewport

Properties
«final» «readOnly»
+ batch : SpriteBatch
«final» «readOnly»
+ gamecam : OrthographicCamera
«final» «readOnly»
+ hud : Hud
«final» «readOnly»
+ player : Player
«final» «readOnly»
+ worldMap : WorldMap

Constructors
+ PlayScreen(SpriteBatch, WorldMap, Player, Settings) : void

Methods
+ dispose( ) : void
+ resize(int, int) : void



Object

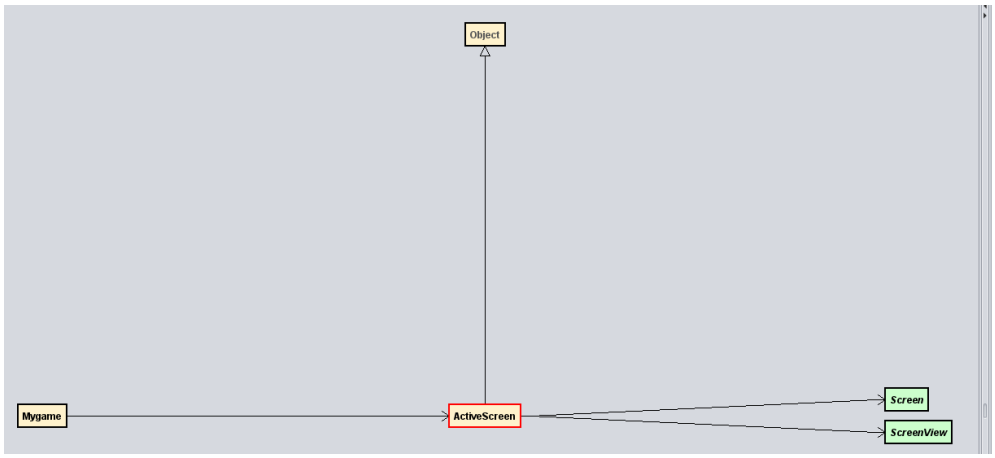
com::game::model::screens::
MenuScreen

Fields
- noSaveGameLabel : Label
«final»
- viewport : Viewport

Properties
«final» «readOnly»
+ game : Mygame
«readOnly»
+ loadGameButton : TextButton
«readOnly»
+ newGameButton : TextButton
«final» «readOnly»
+ stage : Stage

Constructors
+ MenuScreen(Mygame) : void

Methods
- createSimpleButton(String) : TextButton
- createUI( ) : void
+ dispose( ) : void
+ resize(int, int) : void
+ showNoSaveMessage( ) : void



Object

com::game::model::activescreen::
ActiveScreen

Properties
«final» «readOnly»
+ screen : Screen
«final» «readOnly»
+ screenView : ScreenView

Constructors
+ ActiveScreen(Screen, ScreenView) : void