System design document for 017

Version: 1

Date: 2016-05-29

Author: Philip Tham, Anton Josefsson, Mikael Ragnhult, Stefan Fritzon

This version overrides all previous versions.

1 Introduction

1.1 Design goals

We want to design the system in a modular way, with each module as loosely coupled to each other as possible. In particular, we want the application to be extendable and modifiable due to system requirements changing, specifically due to game design changes.

1.2 Definitions, acronyms and abbreviations

MVC - Model, view, control. A software architecture methodology for separating data, logic, input and gui.

2 System design

2.1 Overview

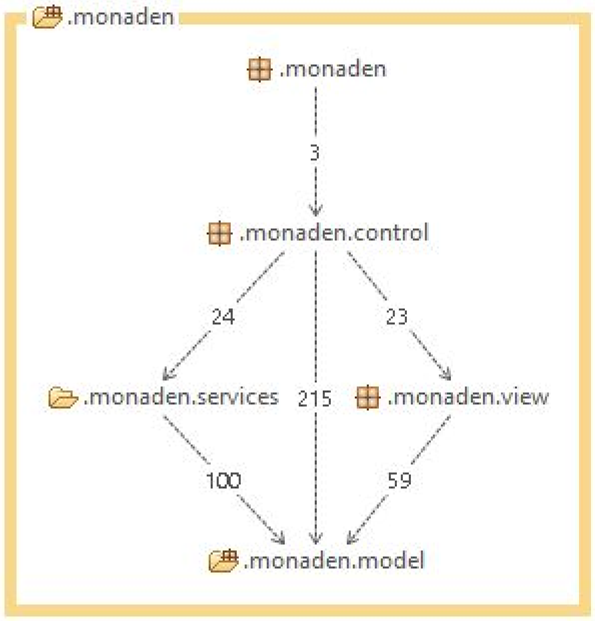
The application will use the MVC model to separate rendering, logic and input from each other. Generally, we want to use a pull strategy between the view and the model (i.e the renderer extracts necessary data from the model), and the controller should handle any user input and instruct the model to change as appropriate.

For necessary large changes in the model (such as transitions between levels) and in order to preserve modularity of certain services (such as audio, potential achievements) we will use the observer pattern, allowing the model to broadcast events to any potential interested parties without breaking the dependencies.

2.2 Software decomposition

2.2.1 General

Our application is divided into the following top level packages

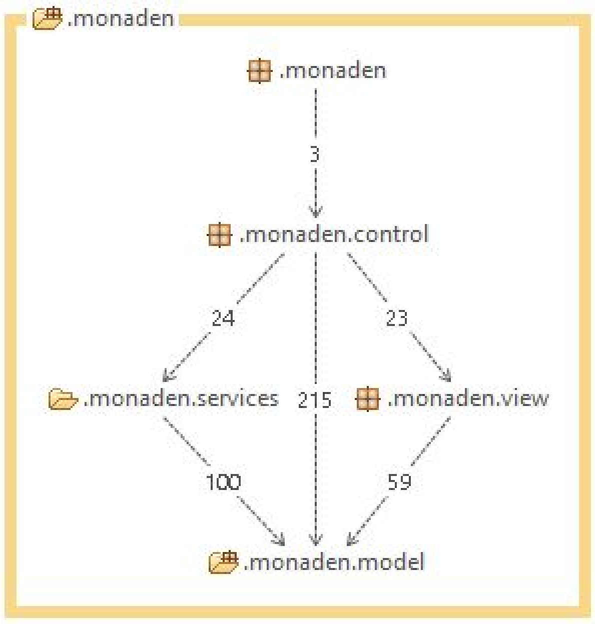


Main is the entry point of the program
Control contains input handling and logic for modifying model
View contains rendering code
Model contains classes that contain data
Services contains functionality such as parsing and audio playing

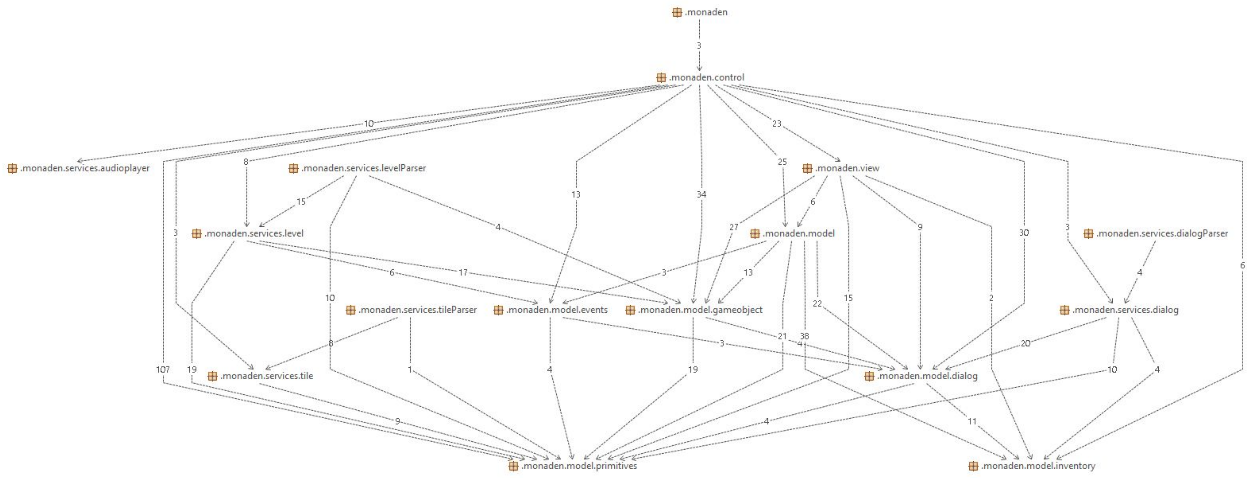
2.2.2 Decomposition into subsystems

The system currently has four services, audio as well as tile, level and dialog parsing. The tile parser is responsible for parsing an xml-file containing ids and what graphic file the id represents, which is used when rendering the levels. The level parser is responsible for parsing xml files containing what objects a level consists of, what characters there are in the level, what other levels you can move to from that level and what dialogs can be triggered from movement. The dialog parser parses xml-files containing dialogs, which can also trigger level transitions.

2.2.4 Dependency analysis



Top level dependencies.



Flattened dependencies.

The dependencies very clean on the top level and reasonably clean on the flattened level. Analysis was made with the STAN tool.

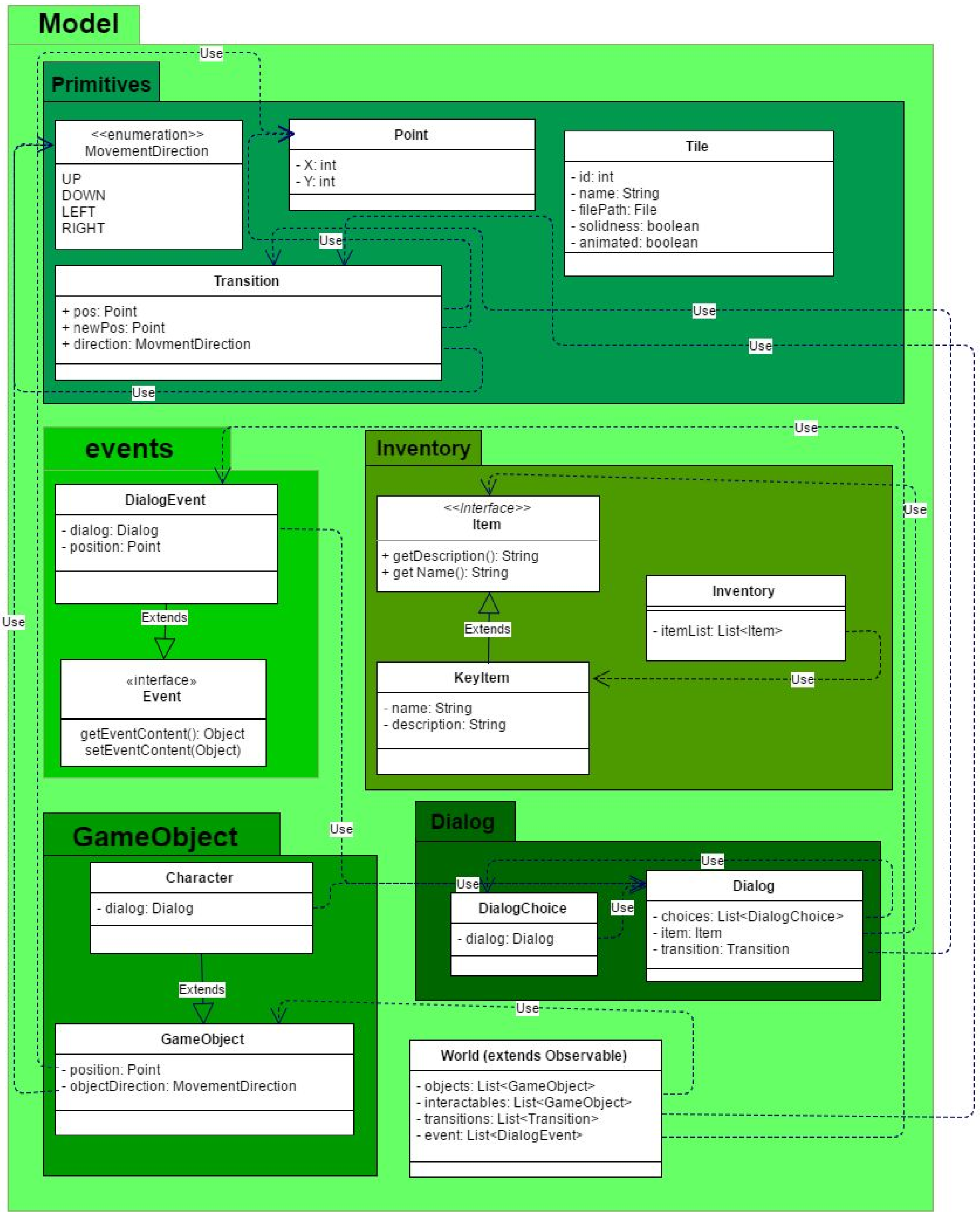
2.4 Persistent data management

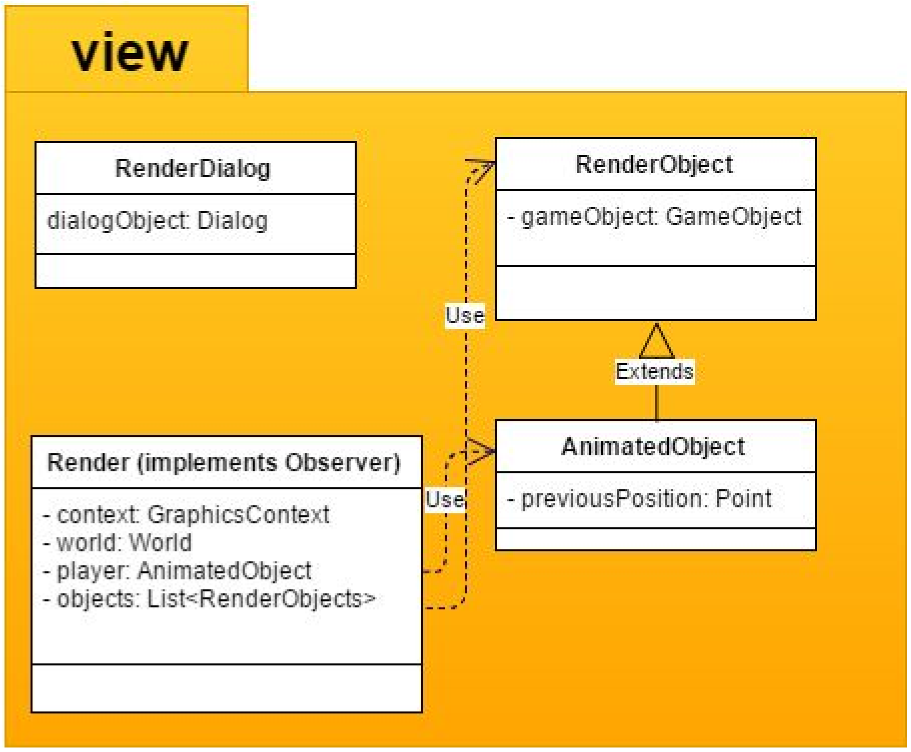
The application will contain a service for saving and loading a game in progress. The details of these services have yet to be designed.

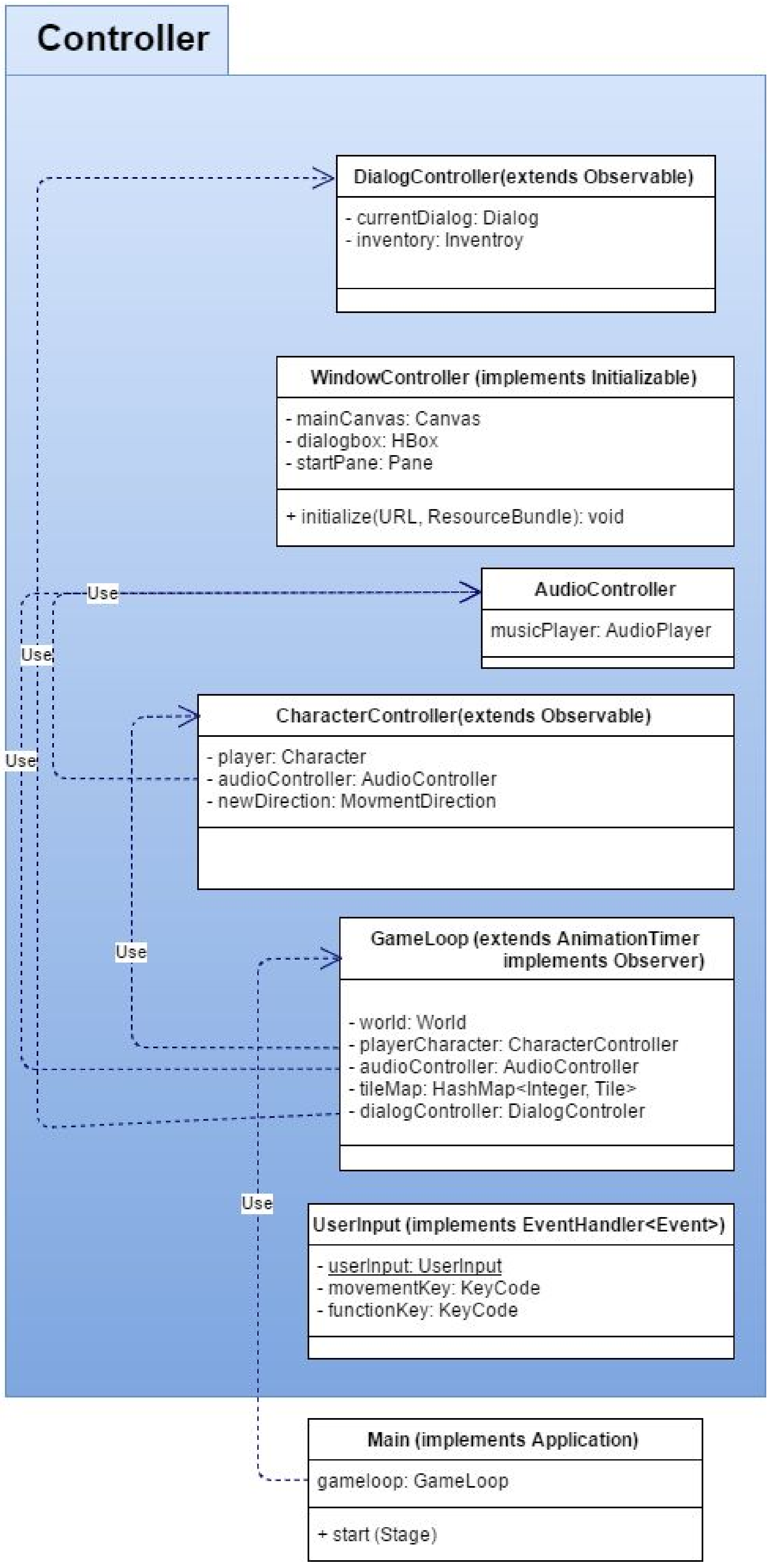
3 References

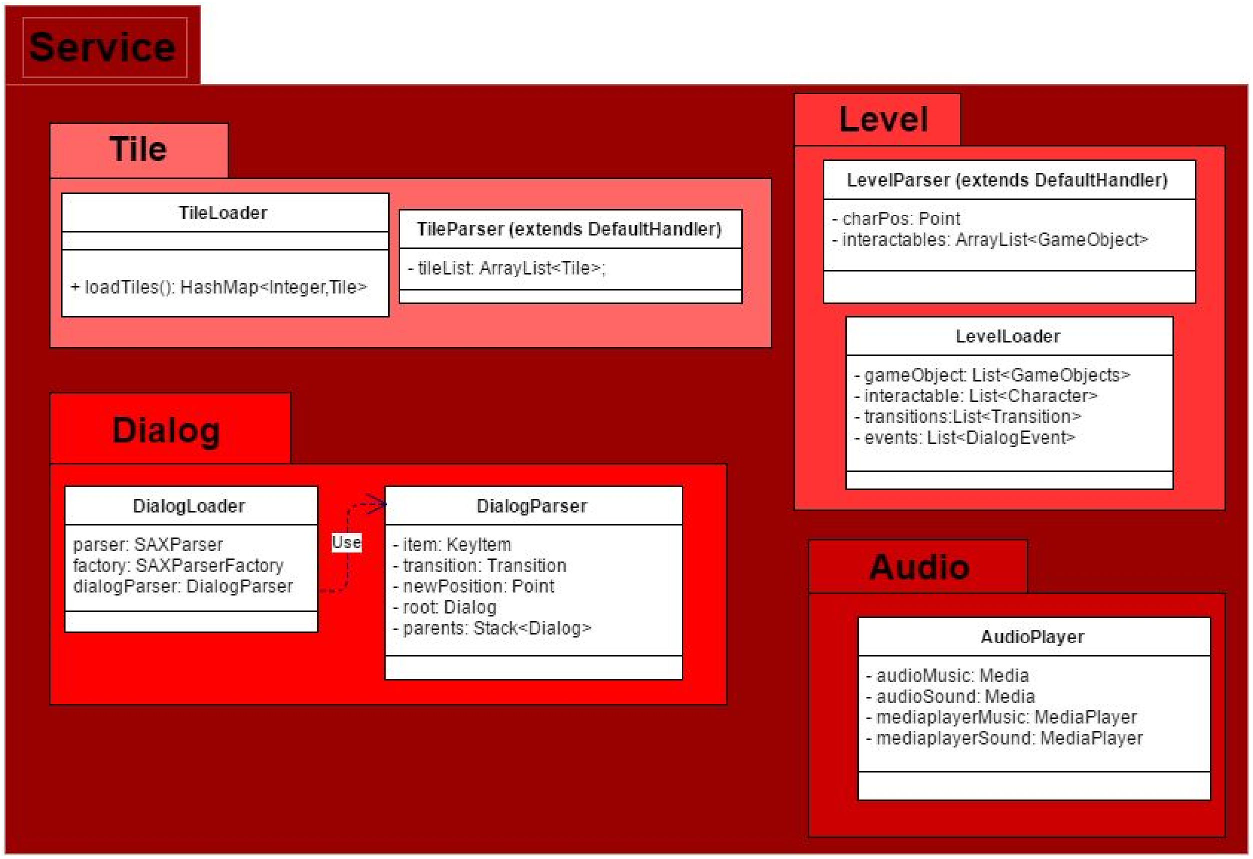
APPENDIX

Model Package UML
View Package UML
Controller Package UML
Service Package UML


Model
Primitives
<<enumeration>>
MovementDirection
UP
DOWN
LEFT
RIGHT
Point
- X: int
- Y: int
Tile
- id: int
- name: String
- filePath: File
- solidness: boolean
- animated: boolean
Transition
+ pos: Point
+ newPos: Point
+ direction: MovmentDirection
events
DialogEvent
- dialog: Dialog
- position: Point
Extends
«interface»
Event
getEventContent(): Object
setEventContent(Object)
Inventory
<<Interface>>
Item
+ getDescription(): String
+ get Name(): String
Extends
KeyItem
- name: String
- description: String
Inventory
- itemList: List<Item>
GameObject
Character
- dialog: Dialog
Extends
GameObject
- position: Point
- objectDirection: MovementDirection
Dialog
DialogChoice
- dialog: Dialog
Dialog
- choices: List<DialogChoice>
- item: Item
- transition: Transition
World (extends Observable)
- objects: List<GameObject>
- interactables: List<GameObject>
- transitions: List<Transition>
- event: List<DialogEvent>
Use


view
RenderDialog
dialogObject: Dialog
RenderObject
- gameObject: GameObject
Use
Extends
Render (implements Observer)
- context: GraphicsContext
- world: World
- player: AnimatedObject
- objects: List<RenderObjects>
Use
AnimatedObject
- previousPosition: Point


Controller
DialogController(extends Observable)
- currentDialog: Dialog
- inventory: Inventroy
WindowController (implements Initializable)
- mainCanvas: Canvas
- dialogbox: HBox
- startPane: Pane
+ initialize(URL, ResourceBundle): void
AudioController
musicPlayer: AudioPlayer
CharacterController(extends Observable)
- player: Character
- audioController: AudioController
- newDirection: MovmentDirection
GameLoop (extends AnimationTimer implements Observer)
- world: World
- playerCharacter: CharacterController
- audioController: AudioController
- tileMap: HashMap<Integer, Tile>
- dialogController: DialogControler
UserInput (implements EventHandler<Event>)
- userInput: UserInput
- movementKey: KeyCode
- functionKey: KeyCode
Main (implements Application)
gameloop: GameLoop
+ start (Stage)
Use
Use
Use
Use
Use


Service
Tile
TileLoader
+ loadTiles(): HashMap<Integer,Tile>
TileParser (extends DefaultHandler)
- tileList: ArrayList<Tile>;
Level
LevelParser (extends DefaultHandler)
- charPos: Point
- interactables: ArrayList<GameObject>
LevelLoader
- gameObject: List<GameObjects>
- interactable: List<Character>
- transitions:List<Transition>
- events: List<DialogEvent>
Dialog
DialogLoader
parser: SAXParser
factory: SAXParserFactory
dialogParser: DialogParser
Use
DialogParser
- item: KeyItem
- transition: Transition
- newPosition: Point
- root: Dialog
- parents: Stack<Dialog>
Audio
AudioPlayer
- audioMusic: Media
- audioSound: Media
- mediaplayerMusic: MediaPlayer
- mediaplayerSound: MediaPlayer