System design document for The Grupp

Version: 1.0

Date: 22/05/2018

Author: Sebastian Lind El-Bahrawy, Eric Mossberg and Alexander Nordgren

This version overrides all previous versions.

Contents

1 Introduction **2**
1.1 Definitions, acronyms and abbreviation . . . 2

2 System architecture **2**
2.1 Top layer . . . 2
2.2 Software . . . 2
2.3 Packages . . . 3

3 Subsystem decomposition **3**
3.1 3.1 "...First software to describe" 3
3.2 "...next software to describe" 3
3.3 Main . . . 3
3.3.1 Main . . . 3
3.4 Controller . . . 3
3.4.1 Game . . . 4
3.5 View . . . 4
3.5.1 AssetHandler . . . 4
3.5.2 Audio . . . 4
3.5.3 GameWindow . . . 4
3.5.4 Menus . . . 4
3.5.5 LevelWindow . . . 5
3.5.6 GUIObject . . . 5
3.5.7 Camera . . . 6
3.6 Model . . . 6
3.6.1 Level . . . 6
3.6.2 Entity . . . 6
3.6.3 CheckCollision . . . 7
3.6.4 Weapon . . . 7
3.7 Services . . . 7
3.7.1 Loader . . . 7
3.7.2 SaveGame . . . 7
3.7.3 Texture . . . 7
3.8 MVC . . . 7

4 Persistent data management **8**

5 Access control and security **8**

6 References **8**

All red text is to be deleted once the SDD is finished.

1 Introduction

General info. What is this? What does it describe?
This document is a technical description of the program developed by the members of The Grupp for course TDA367/DIT212.

1.1 Definitions, acronyms and abbreviation

Some definitions etc. probably same as in RAD

- **STAN**
  - This is a structure analysis tool used in the process of developing this system. Used to avoid, find and elminate unwanted dependencies.
- **MVC**
  - Model-View-Controller, A design pattern used in Software development.
- **LWJGL**
  - Lightweight Java Game Library, severs as a collection of libraries useful when developing a game.
- **OpenGL**
  - A library used to render graphics.
- **GLFW**
  - A library used in OpenGL to create windows.
- **OpenAL**
  -An audio library used with OpenGL.
- **HashMap**
  -A type of list containing paired objects, a key and a value. Given a key, a value can be extracted.

2 System architecture

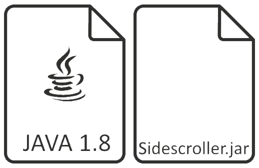
The most overall, top level, description of the system. Which (how many) machines are involved? What software on each (which versions). Which responsibility for each software? Dependency analysis. If more machines: How do they communicate? Describe the high level overall flow of some use case. How to start/stop system. An UML deployment diagram, possibly drawings and other explanations. Possibly UML sequence diagrams for flow. (Persistence and Access control further down) Any general principles in application? Flow, creations, ...

This section severs to describe and diagram the architecture of the program, starting with the very top layer and gradually progressing deeper into the system in the following sections. The software in itself has been developed in a MVC fashion to avoid issues caused by dependencies. There are no cyclical dependencies between packages(can be seen in STAN generated diagrams below.)

2.1 Top layer

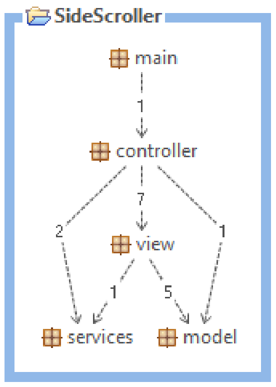
The program in its simplest form is a Game to be run on a computer by a single user. The program uses OpenGL as a tool to render 2D Graphics and therefore requires a GPU supporting this.

2.2 Software



The application requires java version 1.8 and is executed from the **Sidescroller.jar** file.

2.3 Packages



The.jar package contains five class packages, separated into a MVC structure to control dependencies.

3 Subsystem decomposition

3.1 3.1 "...First software to describe" ...

Recap: What is this doing (more detailed) Divide it into top level subsystems. An UML package diagram for the top level. Describe responsibilities for each package (subsystem). Describe interface. Describe the flow of some use case inside this software. Try to identify abstraction layers. Dependency analysis Concurrency issues.

If a standalone application
- Here you describe how MVC is implemented
- Here you describe your design model (which should be in one package and build on the domain model)
- A class diagram for the design model. else
- MVC and domain model described at System Architecture Diagrams
- Dependencies (STAN or similar)
- UML sequence diagrams for flow.

Quality - List of tests (or description where to find the test)
- Quality tool reports, like PMD (known issues listed here)
NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by.., uses ...

3.2 "...next software to describe" ...

As above....

3.3 Main

The Main package contains a single java class, Main.

3.3.1 Main

Responsible for initiating the program. It does so by simply creating and starting the controller, which then takes over.

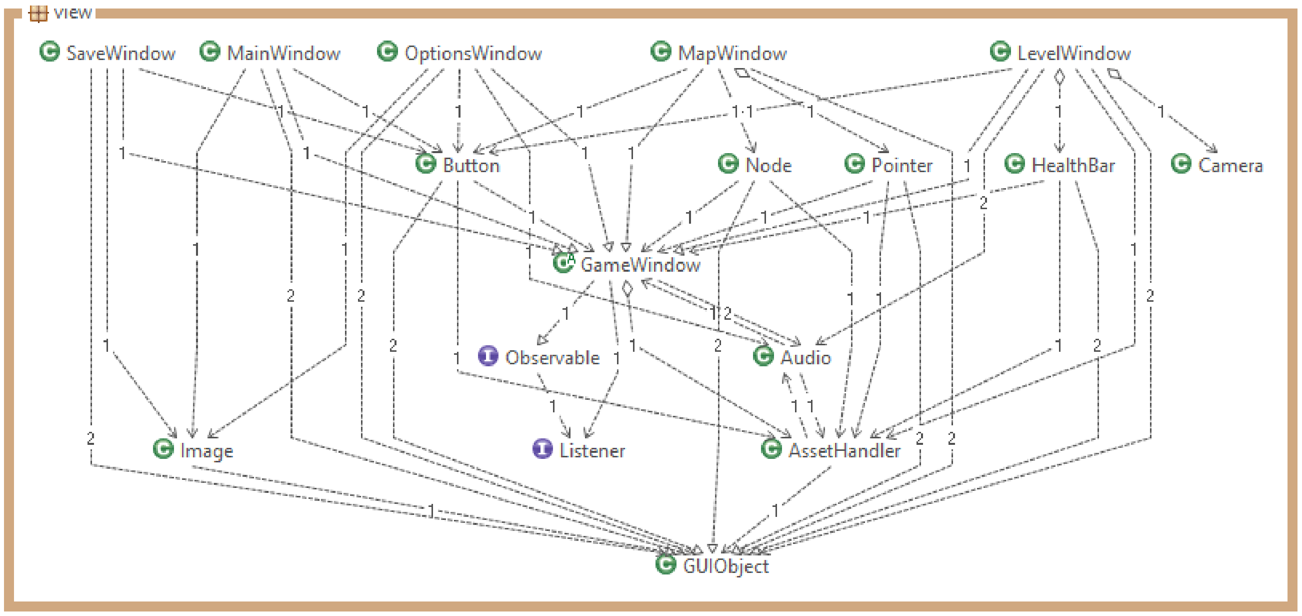
3.4 Controller

The controller package contains the class file, Game.

3.4.1 Game

Acts as a controller for the program, containing both the active model and the active view. Game initializes the first view, the Main menu. It listens to the active view waiting to get notified to update it. Game will only create and use the model when creating a level view.

3.5 View



The view package contains several classes used to construct the programs four menus and the visual part of the game, the level view.

3.5.1 AssetHandler

Functions as a collections of directory paths for game assets(paths to files used by the program). It holds three HashMap objects for images used in relation to the model object Entity, view object GUIObject and view object Audio. A single satic version of this file is only every used and is contained in GameWindow.

Entity, GUIObject and Audio objects all contain ID's, these ID's are used as a Keys in each corresponding HashMap and is paired with a directory path as its Value.

3.5.2 Audio

This sets up the Audio object and is created with an ID, playtime and a Loop value(on/off). The object extends to Java's Thread class to support running in its own thread once started. It however does not initiate this thread itself. A static list of all created audio objects, exists in GameWindow where the functionality for creating new threads for Audio objects, as well as, terminating these audio files is located.

3.5.3 GameWindow

Initializes the game window by using the GLFW library, contains assets, audio and renders textures. This is where the background music is created and run. It also sets up key callbacks, meaning, what happens when a specific key is pressed. All other Windows inherit this class.

3.5.4 Menus

The following four classes are menu classes and all hold a list of GUI Objects specific to that menu, as well as, specific instruction regarding what to do when the user clicks or presses a key.

If a key Callback is run for a mouse click, those coordinates are matched with GUI Object buttons for a button specific outcome. If a key is pressed, that key is matched for a key specific outcome. This outcome is often controller getting notified to make the current active view into another one.

- **MainWindow**: The initial starting menu is generated here. It contains three button objects and an image object.
  - **Play button or Enter key**: Notify controller, transition to the Save Menu
  - **Options button**: Notify controller, transition to the Options menu.
  - **Quit button or Escape key**: Notify controller, terminate program.

- **SaveWindow**: This menu contains four button objects and an image object.
  - **Save button, one, two or three**: Notify controller, create a map menu based on corresponding save data.
  - **Return button or Escape key**: Notify controller, transition to Main menu.

- **OptionsWindow**: This menu contains two button objects and an image object.
  - **Toggle Music button**: Terminate or start the background song.
  - **Return button or Escape key**: Notify controller, transition to Main menu.

- **MapWindow**: The menu hols two button objects, several node objects and a pointer.

  This class is generated based on save and map data and contains as many nodes as the map data instructs, these nodes are either locked or unlocked based on the save data. It holds a pointer object, this object is rendered above a node and is moved given a key press. How far the pointer may move is limited by either the end of the row of nodes, or if the next node is locked.
  - **Enter Level button or Enter key**: Notify controller, the level corresponding to the current pointer position should be created and the view should transition.
  - **Return button or Escape key**: Notify controller, transition to Main menu.
  - **Right or Left key**: Update pointer position.

3.5.5 LevelWindow

This is the core view of the game. The class is created based on the Model and all containing Entities. It has a single button and a health bar GUI object and renders these, as well as, the model entities. This class continually re-renders based on the ever changing model.

To animate the health bar, the character and the enemy this view checks with values taken from the Model to change what path is used in the Asset Handler for these objects.

- **Space key**: Updates model.
- **Left key**: Updates model.
- **Right key**: Updates model.
- **Return button or Escape key**: Notify controller, transition to Map menu.

3.5.6 GUIObject

This object holds an ID, coordinates and a directory path value. On creation this ID is used to gather a path from the Asset Handler. The following five classes inherit GUIObject.

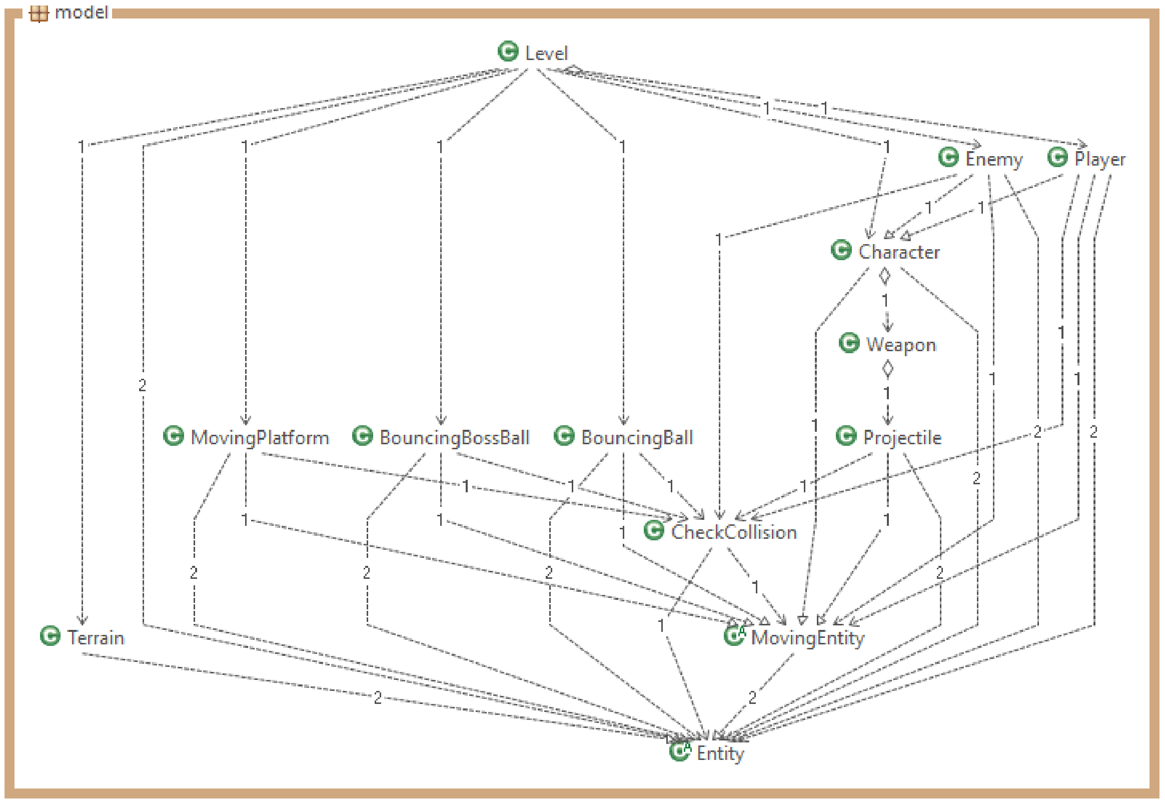
- **Button**: In addition to the inherited values, this class holds a height and a width, as well as, functionality using these values to calculate if a coordinate exists on the button.
- **Node**: Used in map to symbolize a Level entry point.

- **Pointer**: Unlike other GUI Objects, this hold functionality for updating position in relation to node position.
- **HealthBar**: Holds functionality to change what Value is used in asset handler for its ID, based on model.
- **Image**: Does not use asset handler, takes a directory path in its constructor instead.

3.5.7 Camera

The camera is the core of the side-scrolling mechanic. Its used when LevelWindow paints entities from model, based on how far the character has moved the cameras coordinates changes and alter what entities may get painted and how far their coordinated should be modified when painted.

3.6 Model



3.6.1 Level

Uses a level file to generate a customized level model. This model defines everything existing on the Level, as well as, functionality to modify the model.

3.6.2 Entity

This is an existing physical object in the level. It is defined during the levels creation and holds, dimensions, coordinates, an ID and collision detection functionality.

All entities below inherit Entity.

- **Terrain**: A simple object used for statically places Terrain, such as non-moving platforms, the ground or other obstacles.
- **MovingEntity**: This adds directional functionality to the entity object and is inherited by the classes below.
  - **BouncingBall**: An object with custom functionality for an object that regains its y-axis velocity on collision. This ball only travels on the y-axis.

- **BouncingBossBall**: An object traveling on both axes, inverting its direction on collision unaffected by gravity.

- **MovingPlatform**: An object continually updating its x-coordinate to move right to left.

- **Character**: Hold core functionality for characters, such as health, taking damage, dealing damage and dying.

  Character is inherited by the following two classes.

  * **Player**: Holds functionality unique to the player character, such as weapons.
  * **Enemy**: Holds functionality unique to the enemy character, such as independent random movement.

  Projectile: Used by the Weapon class, is simply a directional object continually updating coordinates until collision.

3.6.3 CheckCollision

Used by Level to check for Entity collisions.

3.6.4 Weapon

Used by character to define Weapon specific attributes.

3.7 Services

3.7.1 Loader

Loads Level and Map files by translating the file text into and array of Strings. This is used by the controller when creating the level model and the map view.

3.7.2 SaveGame

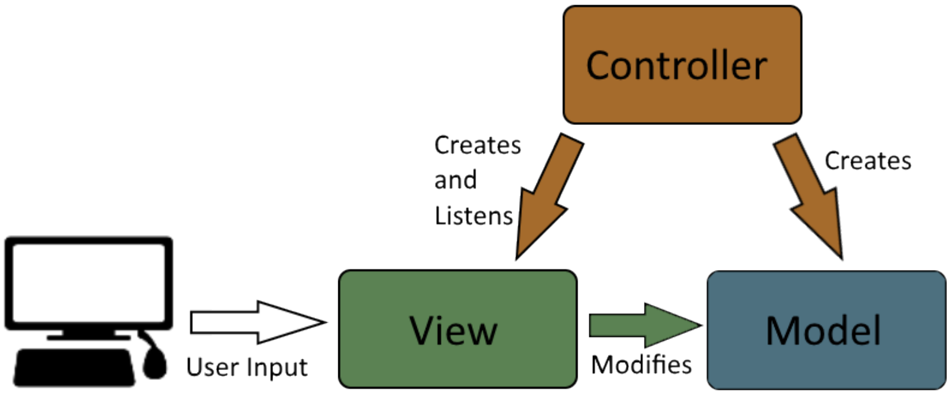
Writes and reads save files. This is uses by the controller when creating the map view, with the purpose of limiting or increasing player access.

3.7.3 Texture

Translates images into data to be rendered.

3.8 MVC

For each identified software above (that we have implemented), describe it ...



The Controller creates and listens to the created view, it observes events for new views to be created. Some views are created based on a Model, a model is then only created during the creation of such a view.

The View receives user input and may modify the model or create events read by the controller.

The Model is continually modified my the view and used by the view when rendering.

4 Persistent data management

How does the application store data (handle resources, icons, images, audio, . . .). When? How? URLs, path's, . . . data formats. . . naming..

The program uses several .png images and .oog sound files stored in the Assets directory. The paths leading to these files are stored and accessed by the view through the AssetHandler java class as described in the View Section.

5 Access control and security

Different roles using the application (admin, user, . . .)? How is this handled?
As our software is a standalone application to be used by a single person at a time, security concerns havent been much of a priority.

However if cheating would be a concern, encrypting save, map and level files so they may not be modified by the user could be something to look into. Maybe even adding a password to allow entry into a specific save to protect a Users progress.

6 References