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

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

- **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, found in LWJGL.
- **GLFW**
  - A library used to create windows, found in LWJGL.
- **OpenAL**
  -An audio library found in LWJGL.
- **HashMap**
  -A type of list containing paired objects, a key and a value. Given a key, a value can be extracted.
- **Maven**
  - Used to load any and all necessary library dependencies during the development process and for the finished .jar file.

2 System architecture

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 or classes(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

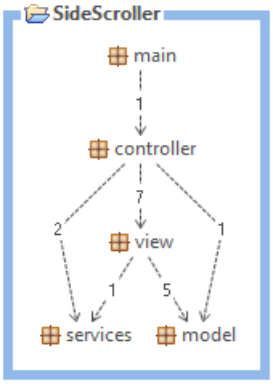
The application requires java version 1.8 and the SideScrollerDemo.zip file, containing a directory of game assets(images and sound), a SideScrollerDemo.jar file and a README.txt file.

How to run:

1. Install java.
2. Unzip SideScollerDemo.zip to where you want to execute the program.
3. Double click SidescrollerDemo.jar alternatively run ”java -jar SideScrollerDemo.jar” in the command prompt.

The program should run and can be turned of by pressing the quit button, pressing escape when in the main menu or by pressing the cross in the upper right corner.

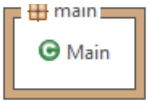
2.3 Packages



As well as several LWJGL libraries, The.jar package contains five class packages, separated into a MVC structure to control dependencies. This package is built using Maven, including all needed dependencies to the package.

3 Subsystem decomposition

3.1 Main



The Main package contains a single java class, Main.

3.1.1 Main

Author
Sebastian

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

Used by
N/A

Uses
Game

3.2 Controller

The controller package contains the class file, Game.

3.2.1 Game

Author
Sebastian & Eric

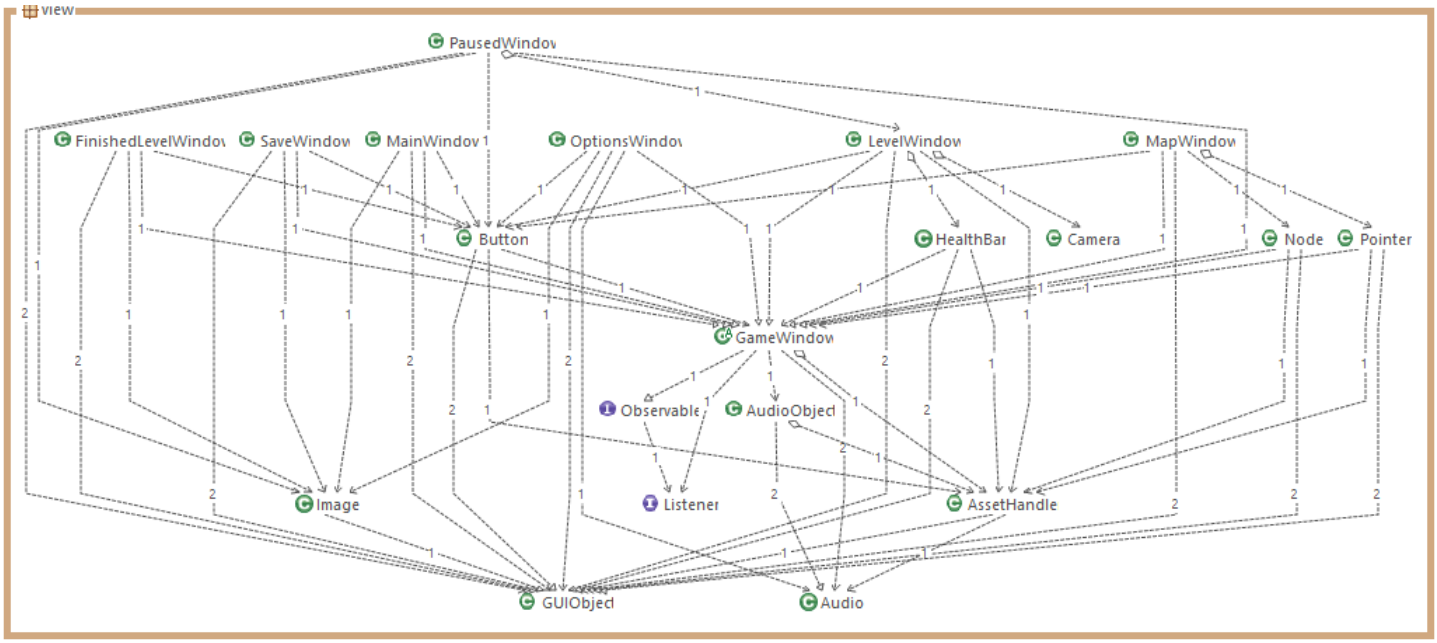
Responsibility
Acts as a controller for the program, containing both the active model and the active view. Game initializes the first view, the Main menu. It listens to the active view waiting to get notified to update it. Game will only create and use the model when creating a level view.

Used by

N/A

Uses
Listener, FinishedLevelWindow, GameWindow, LevelWindow, MainWindow, MapWindow, OptionsWindow, PausedWindow & SaveWindow.

3.3 View



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

3.3.1 AssetHandler

Author
Sebastian

Responsibility
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 static version of this exists in GameWindow(used for graphics) and in AudioObject (used for sound and music).

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.

Used by
LevelWindow, GameWindow, AudioObject, Node, Pointer, Button & HealthBar.

Uses
Audio, Entity & GUIObject

3.3.2 Audio

Author
Sebastian

Responsibility
Mainly serves to limit dependencies for GameWindow, AudioObject and AssetHandler. Sets up audio Id's, inherits Thread and is inherited by AudioObject.

Used by
AssetHandler, AudioObject, OptionsWindow & GameWindow.

Uses
N/A

3.3.3 AudioObject

Author
Sebastian

Responsibility
This sets up the Audio object and is created with an ID, playtime and a Loop value(on/off). The object inherits Audio, which 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.

Used by
GameWindow

Uses
Audio, AssetHandler

3.3.4 GameWindow

Author
Sebastian & Eric.

Responsibility
Abstract definition of a game window. Defines the basic functionality of all windows, such as supplying them with vital methods. Each class that extends this must define their own render and and input logic.

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.

Used by
Game, MapWindow, OptionsWindow, MainWindow, LevelWindow, PausedWindow, FinishedLevelWindow, SaveWindow, Pointer, Button, Healthbar & Node.

Uses
Listener, Observable, Texture, Audio, AudioObject, AssetHandler.

3.3.5 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**

  Author
  Sebastian

Responsibility
The 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.

Used by
Game

Uses
Button, GUIObject, Image, GameWindow

- **SaveWindow**

Author
Alexander

Responsibility
A simple view containing three options to load a game.
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.

Used by
Game

Uses
Button, GUIObject, Image, GameWindow

- **OptionsWindow**

Author
Eric

Responsibility
Simple window that allows the user to modify program state. At the moment, only muting the music is supported, but additional functionality can be added should it be needed.
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.

Used by
Game

Uses
Button, GUIObject, Image, GameWindow, Audio

- **MapWindow**

Author
Sebastian

Responsibility

The menu holds 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.

Used by
Game

Uses
Button, GUIObject, Pointer, Node, GameWindow

- **FinishedLevelWindow**

Author
Alexander

Responsibility
This menu contains two button objects and an image object and loads when a level is won or the character dies.

- **Back to map button or Enter key**: Notify controller, create a map menu based on current save data.
- **Back to menu button or Escape Key**: Notify controller, transition to Main menu.

Used by
Game

Uses
Image,Button,GameWindow,GUIObject.

- **PausedWindow**

Author
Alexander & Eric

Responsibility
Window that is shown when game is paused while in the LevelWindow. Allows the user to exit the level or return to main menu.

- **Retun button or Escape Key**: Notify controller, create LevelWindow based on current existing model.
- **Back to map button or Enter key**: Notify controller, create a map menu based on current save data.
- **Back to menu button**: Notify controller, transition to Main menu.

Used by
Game

Uses
LevelWindow,Image,Button,GameWindow,GUIObject.

3.3.6 LevelWindow

Author
Sebastian & Eric

Responsibility
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 Pause menu.

Used by
Game, PausedWindow

Uses
GameWindow, Entity, GUIObject, Level, SaveGame, HealthBar, Button, CheckCollision, Camera, AssetHandler

3.3.7 GUIObject

Author
Sebastian

Responsibility
This object holds an ID, coordinates and a directory path value.

Used by
MapWindow, OptionsWindow, MainWindow, LevelWindow, SaveWindow, PausedWindow, FinishedLevelWindow, Pointer, Button, Image, Node, HealthBar

Uses
N/A

The following five classes inherit GUIObject. On creation these objects use its ID to gather a path from the Asset Handler.

- **Button**

  Author
  Sebastian

  Responsibility
  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.

  Used by
  MapWindow, OptionsWindow, MainWindow, LevelWindow, SaveWindow, PausedWindow, FinishedLevelWindow

  Uses
  GUIObject, AssetHandler, GameWindow.

- **Node**

  Author
  Sebastian

  Responsibility
  Used in map to symbolize a Level entry point.

  Used by
  MapWindow

  Uses
  GUIObject, AssetHandler, GameWindow.

- **Pointer**

  Author
  Sebastian

  Responsibility
  Unlike other GUI Objects, this hold functionality for updating position in relation to node position.

  Used by
  Map Window

  Uses
  GUIObject, AssetHandler, GameWindow.

- **HealthBar**

  Author
  Sebastian

  Responsibility
  Holds functionality to change what Value is used in asset handler for its ID, based on model.

  Used by
  LevelWindow

  Uses
  GUIObject, AssetHandler, GameWindow.

- **Image**

  Author
  Sebastian

  Responsibility
  Does not use asset handler, takes a directory path in its constructor instead.

  Used by
  OptionsWindow, MainWindow, SaveWindow, PausedWindow, FinishedLevelWindow

  Uses
  GUIObject

3.3.8 Camera

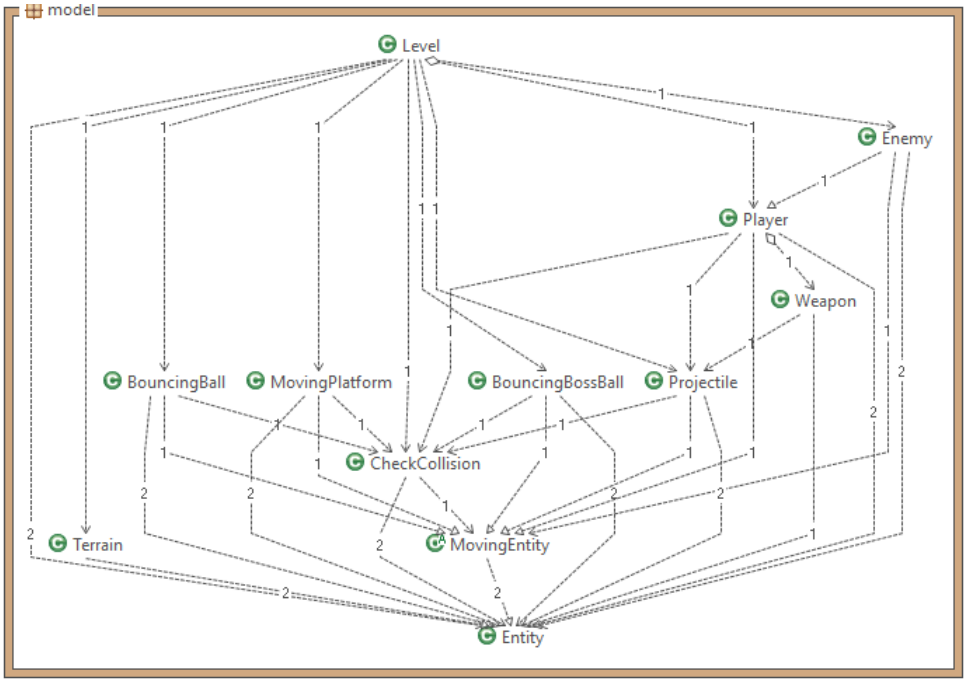
Author
Eric

Responsibility
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 sets what entities may get painted and how far their coordinated should be modified when painted.

Used by
LevelWindow

Uses
Entity

3.4 Model



3.4.1 Level

Author
Sebastian, Eric & Alexander

Responsibility
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.

Used by
LevelWindow, Game

Uses
Entity, Terrain, BouncingBossBall, BouncingBall, Projectile, CheckCollision, Enemy, MovingPlatform, Player

3.4.2 Entity

Author
Eric

Responsibility
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.

Used by
Camera, AssetHandler, LevelWindow, Terrain, BouncingBossBall, BouncingBall, Projectile, CheckCollision, Enemy, MovingPlatform, Player, Weapon, MovingEntity, Level

Uses
N/A

All entities below inherit Entity.

- **Terrain**

  Author
  Sebastian

  Responsibility
  A simple object used for statically places Terrain, such as non-moving platforms, the ground or other obstacles.

  Used by
  Level

  Uses
  Entity

- **MovingEntity**

  Author
  Alexander

  Responsibility
  An extension of the Entity class. Defines behaviour for an Entity that is able to be in motion.

  Used by
  Player, Enemy, BouncingBossBall, BouncingBall, CheckCollsion, Projectile, MovingPlatform

  Uses
  Entity

  MovingEntity is inherited by the classes below.

  - **BouncingBall**

    Author
    Alexander

    Responsibility
    An object with custom functionality that regains its y-axis velocity on collision. This ball only travels on the y-axis.

    Used by

Level

Uses
Entity,MovingEntity, CheckCollision.

- **BouncingBossBall**

  Author
  Alexander

  Responsibility
  An object traveling on both axes, inverting its direction on collision unaffected by gravity.

  Used by
  Level

  Uses
  Entity,MovingEntity, CheckCollision.

- **MovingPlatform**

  Author
  Alexander

  Responsibility
  An object continually updating its x-coordinate to move right to left.

  Used by
  Level

  Uses
  Entity,MovingEntity, CheckCollision.

- **Player**:

  Author
  Sebastian & Eric

  Responsibility
  Holds core functionality for characters, such as health, movement, taking damage, dealing damage and dying.

  Used by
  Level, Weapon, Enemy

  Uses
  Entity, MovingEntity, CheckCollision,Weapon, Projectile.

  Player is is inherited by Enemy.

  * **Enemy**

    Author
    Sebastian & Alexander

    Responsibility
    Holds functionality unique to the enemy character, such as independent random movement.

Used by
Level

Uses
Entity, MovingEntity, Player

- **Projectile**

  Author
  Eric

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

  Used by
  Level, Weapon, Player

  Uses
  Entity, MovingEntity, CheckCollision.

- **Weapon**

  Author
  Eric

  Responsibility
  Used by player to define a Weapon with specific attributes. If the need for more than one weapon for the Player arises, this class can be slightly modified with minimal effort to accommodate that change.

  Used by
  Player

  Uses
  Entity, Projectile

3.4.3 CheckCollision

Author
Sebastian & Alexander

Responsibility
Used by Level to check for Entity collisions.

Used by
LevelWindow, Level, Projectile, BouncingBossBall, BouncingBall, Player, MovingPlatform.

Uses
Entity, MovingEntity

3.5 Services

3.5.1 Loader

Author
Sebastian

Responsibility
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.

Used by
Game

Uses
N/A

3.5.2 SaveGame

Author
Alexander

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

Used by
Game, LevelWindow

Uses
N/A

3.5.3 Texture

Author
Sebastian

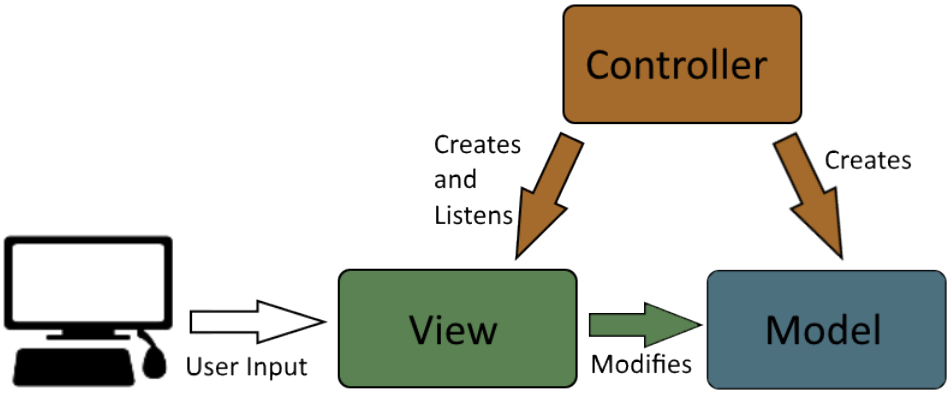
Responsibility
Translates images into data to be rendered.

Used by
GameWindow

Uses
N/A

3.6 MVC

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.



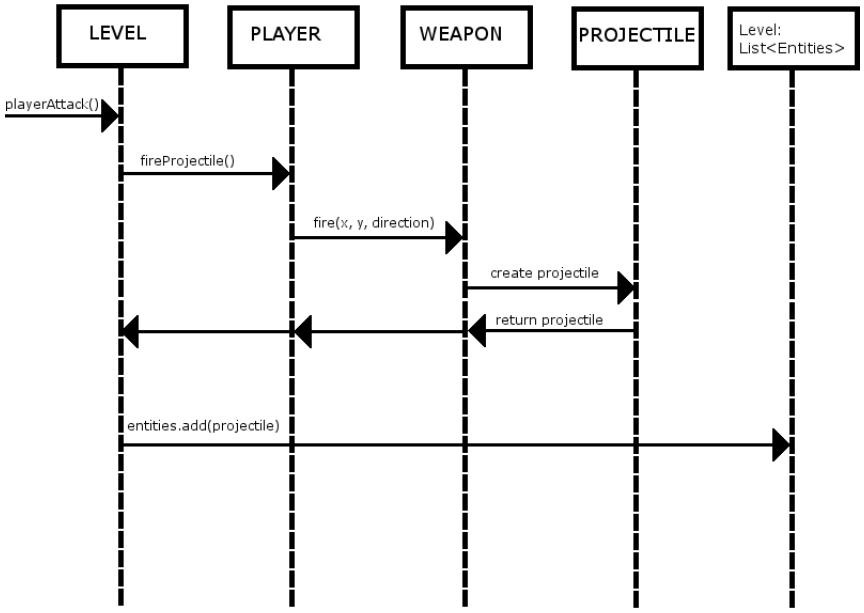
3.7 UML Class Diagram

A complete and very large class diagram of the program can be found <<here>>. Even though the full class structure is covered in this document, this diagram gives a solid overview of the structure and individual class dependencies across packages.

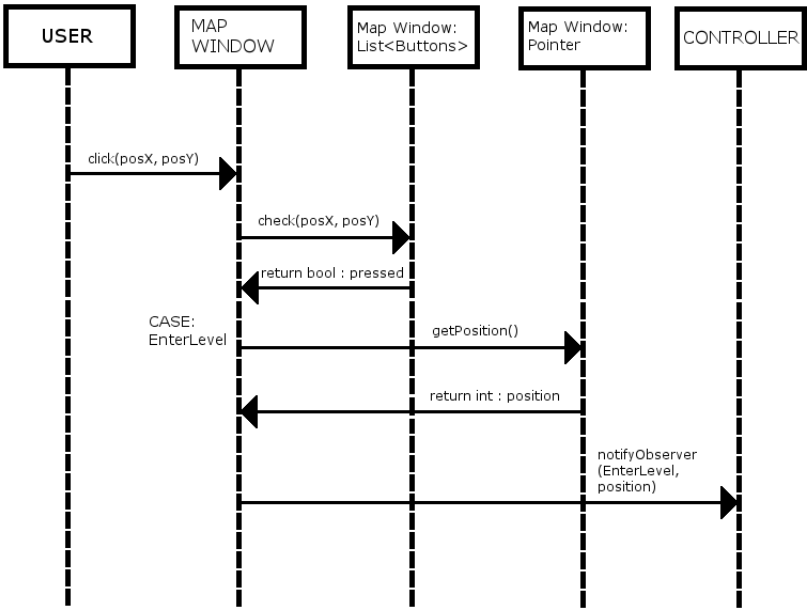
3.8 Sequence diagram

This sections severs to give some examples of sequences of calls happening in the program.

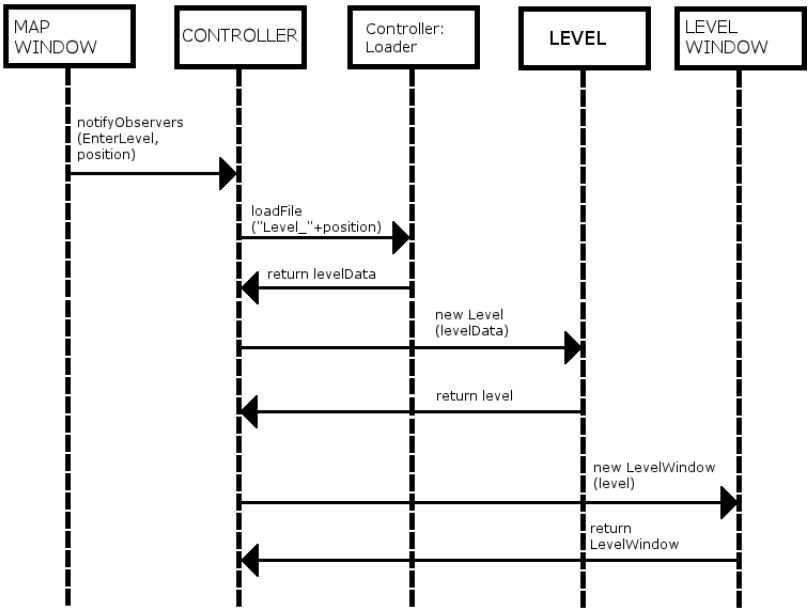
This diagram shows the sequence of calls that that place from the time the model learns that the player has attacked until the projectile is added as one of the models entities.



The user clicking enter the enter level button on the map view.



Controller updates current window to a new LevelWindow



3.9 Quality Assurance

To assure our program runs as smoothly as possible, the code analyzer PDM was used find and eliminate possible issues. Several critical "violations", as the diagnostic tool calls it, where addressed. Mainly focusing on performance hindering problems bus also on unnecessary code, such as, unused methods and variables and imports.

We did however find PDM heavily pedantic, and chose to ignore several "violations" found to be trivial, focusing on larger issues.

3.10 Tests

Documented test cases for the model can be found in **Game/src/model/model_tests**.
They test mainly the Player and Level class, while also ensuring that inherited functionality like the methods Player inherits from Entity works correctly.

4 Persistent data management

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

5 Access control and security

As our software is a standalone application to be used by a single person at a time, security concerns haven't 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

Some references to the concepts, libraries and tools covered in this document can be found here.

1. LWJGL - <<Link>>
2. STAN - <<Link>>
3. PMD - <<Link>>
4. MVC - <<Link>>
5. UML class diagram - <<Link>>
6. Sequence diagram - <<Link>>
7. Maven - <<Link>>