Ted's Entertainment

Ted's Quest
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
<29/dez/19>	<1.0>	<initial description>	<Michaela Fleig>

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This document applies to the development of the game “Ted’s Quest” by Ted’s Entertainment. Affected by the SRS and the use cases.

1.3 Definitions, Acronyms, and Abbreviations

TQ	Ted’s Quest, name of the game

1.4 References

SRS	SRS.md, date, Ted’s Entertainment for Software Engineering Course, linked in same folder at GitHub, like this document.

1.5 Overview

This document describes the architecture of the software application “Ted’s Quest”.

2. Architectural Representation

As software architecture we planned to use the classical MVC-model. This should make the handling and maintaining of the source code easier and more understandable. By using the Unity Engine, it will be more complicated to implement, since Unity is providing most of the functions. MVC is represented by the three parts: model, controller and view. Model collects all data. In case of TQ, it’s the players local account, the scores and the local settings. The controller provides the data flow, the logic of the application. Applied to TQ this means the interfering points between the model and the user interface. The user interface is provided by the view. The view renders the data that the controller provides.

3. Architectural Goals and Constraints

Since we are saving the users login data on his local device, the architecture is designed for not publishing any information. If the user is signing up with its Google account, we cannot guarantee the same security, since Google is collecting its own information about its accounts.
The system should be running on computer and Android phones. Which is easy to implement since there is a service from the Unity Engine to build the same application for both devices. This and the used architectural model make the reuse of the source code uncomplicated.
Since we are not copying user data to our own server, it is not necessary at all to use a network connection.

4. Use-Case View

Until this moment there are no significant scenarios the use cases. The classes will be divided into the architectural model as follows:

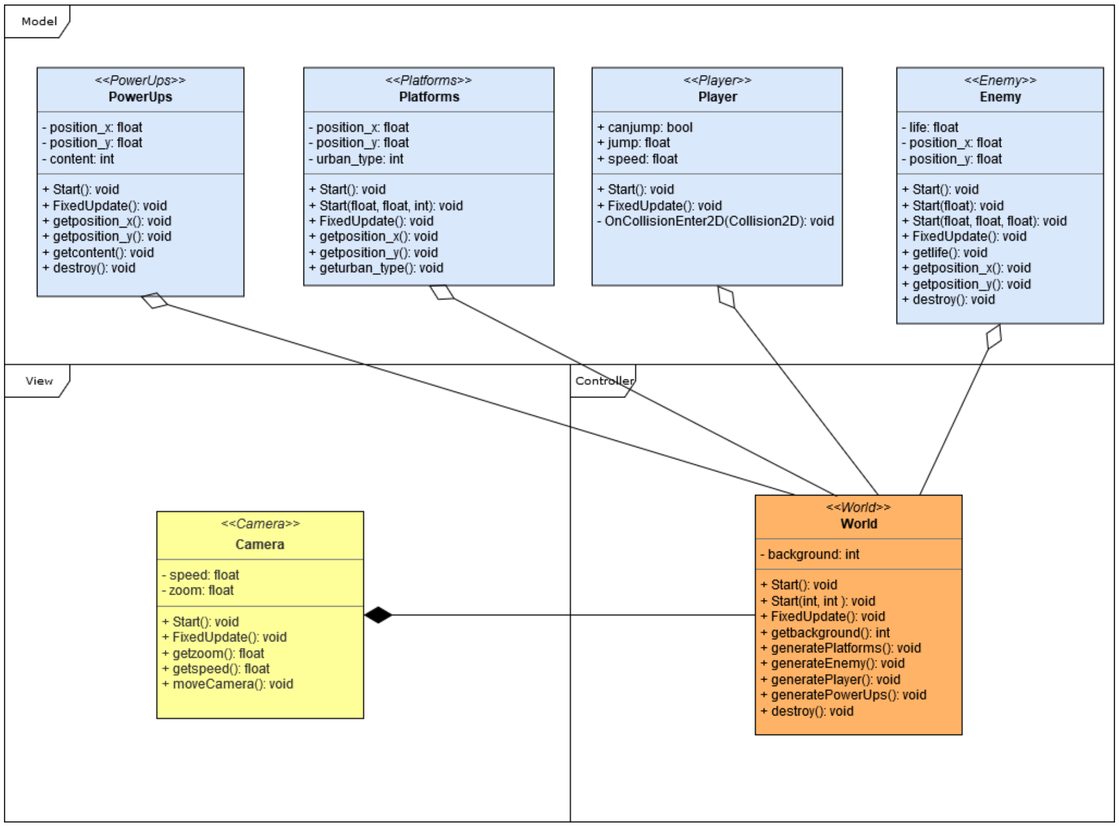

Abbildung 1: Some classes to demonstrate the MVC-model.

The blue section describes the "Model" components, the yellow class describes the "View" and the class "Word" is orange and belongs to the part "Controller".

4.1 Use-Case Realizations

The "Camera" is the interface of the game to the user. Controlled by the class "World" which is controlled by the user input, the data of the "Player" e.g. can jump or speed up. The demo will show these actions for the main character and can be found in the GitHub-Repository.

5. Logical View

Significant for the architecture is the separation of the source code into the parts model, view and control. The source code are kept simple and structured.
Therefore, classes for the user interface are structured together. They have a way to communicate with the control but not with the model. So only control can communicate with all logical views.

5.1 Overview

(n/a)

5.2 Architecturally Significant Design Packages

(n/a)

6. Process View

At this point, there is no real process view possible due to the fact, that there is a too small amount of classes that the developers can create themselves that could interact.
The user is interfering with the interface-class "World" that passes all valid information to the data storage.

This section passes the data back to the interface if necessary and now, the “World” passes this information to the view-part “Camera” for the user to see.

7. Deployment View

There is no delpolyment view for this project possible due to the fact, that there is no hardware connection existing. The developer will not have access to the hardware.

8. Implementation View

Significant for this architecture is the division of the classes into logical groups. Unity makes this harder to implement by taking a lot of orders away from the developer. This implementation will be remarkable for its best effort architecture that relies to the MVC-model.

8.1 Overview

(n/a)

8.2 Layers

(n/a)

9. Data View (optional)

(n/a)

10. Size and Performance

(n/a)

11. Quality

The chosen software architecture contributes to safety from internal data corruption and privacy implications. Portability and reliability are mainly contributed by the Unity Framwork, that takes most of the hardware based programming.