<Company Name>

<Ted's Quest>
Software Architecture Document

Version <1.0>

[Note: The following template is provided for use with the Rational Unified Process. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

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.
[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

4.1 Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality.]

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

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

5.2 Architecturally Significant Design Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

6. Process View

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

7. Deployment View

*[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]*

8. Implementation View

[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

8.1 Overview

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers.]

8.2 Layers

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

9. Data View (optional)

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

10. Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

11. Quality

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]