Emergent Architecture Design
StandUp Game

f 9th May 2015

Nick Cleintuar, 4300947
Martijn Gribnau, 4295374
Jean de Leeuw, 4251849
Benjamin Los, 4301838
Jurgen van Schagen, 4303326

Contents

1 Introduction

This document will provide insight into the architecture and the design of our product. It will provide goals that we seek to for fill and explanation of how our product is built.

1.1 Design Goals

In this section you will find the goals we seek to achieve in the design of our product. Throughout the development of our product we will try to maximise these design goals.

1.1.1 Reliability

Our product should be reliable, it should always work.

We want to guarantee the reliability of our product by testing. This will be in the form of unit testing, integration testing and real life testing. Further we will seek reliability by preventing mistakes by programming in pairs, using Pull request and static analysis tools like check style.

1.1.2 Manageability

Our product should also be manageable. It should be possible to change or add functionality to our product in the future, without unnecessary hassle.

We are going to keep it manageable by using design patterns. This provides more structure to our product. Another way we are going to achieve this is by KISS. To try to keep our implementations simple to make it easier to understand in the future.

1.1.3 Usability

Our product should be easy to use, have a low learning curve. This is important, because it will increase the likely hood that new players will keep on playing.

To increase the usability of our game we will let real users test our game to give as insight into what works and what does not. This gives us a chance to change it earlier or perhaps even to rethink the concepts of our product.

2 Software Architecture Views

This section will provide insight in the architecture of our product. It is divided into four paragraphs. In the first paragraph we will talk about how the subsystems of our product work together. In the second paragraph we will explain how the hardware and the software of our product work together. The third paragraph will provide insight into how the data is stored and handled. In the last paragraph we will discuss where collisions between subsystems might occur and how we prevent them.

2.1 Subsystem decomposition

Our product consists out of multiple subsystems. A subsystem in charge of the graphical user interface. A subsystem in charge of the assets. A subsystem in charge of the timers. And a subsystem in charge of the game mechanics.

2.2 Hardware/Software mapping

A user needs a device with access to a gyroscope and an accelerometer to complete tasks and to measure activity. Further it will need Bluetooth to find and connect to nearby users and Internet to access group specific data.
The system architecture will consist out of a server to let users access group specific data.

2.3 Persistent data management

Users can form groups and play the game as a group. Data that is group specific must be stored on a server to make it accessible for all the users in the group. Data that is user specific might also be stored on a server to prevent it from being accessed by third parties.

2.4 Concurrency

Collisions between subsystems can occur on multiple levels throughout our product. On server level two users might try to change group specific data at the same time, this will be mediated by the database management system. Collisions might also occur within the application, when two subsystem would like to access the same recourse.

Glossary

KISS Keep It Simple Stupid. 2

Pull request Pull request is a way to inform others of the changes you have made on a git repository and provides the opportunity for others to review your work and suggest changes. 2