Requirements and Analysis Document for Gentlemen’s Dodgeball

Table of Contents

Version: 1.2

Date: 23/3 2011 (Revised 26/5 2011)

Author: Karl Bristav, Erik Sikander, Gustav Olsson, Viktor Åkerskog

This version overrides all previous versions.

1 Introduction

Gentlemen’s Dodgeball is a multiplayer 2D game, intended to play as a variation of the popular sport dodgeball on a computer. The game will be very simplistic in its design; The goal is to achieve deep gameplay with simple rules and actions.

1.1 Purpose of application

Gentlemen’s Dodgeball is entirely for entertainment purposes.

1.2 General characteristics of application

This application runs in real-time, whether the user provides input or not, and provides a rich graphical experience. Because of the multiplayer nature of the game, events may occur in the game world that are outside of the scope and actions of some players (as opposed to normal desktop applications, where all events are normally initiated by a single user, and no events occur unbeknownst to the user).

1.3 Scope of application

The game is built to be played by up to two players using the same computer and keyboard. It does not save any statistics and interrupted game sessions will be lost.

1.4 Objectives and success criteria of the project

The objective is to have a runnable prototype of the game where it is possible to play against other people on the same computer. A player should be able to move around in the game world, collide with obstacles, pick up balls, throw balls, capture flags and knock out players on the opposing team.

1.5 Definitions, acronyms and abbreviations

Game instance - A copy of Gentlemen’s Dodgeball running on a computer.
Game world - A virtual world that is visualized by the game instance.
Player - A person interacting with the game instance. A player has a representation inside the game world that other players can see and interact with.

Team - A set of players working together. The game will initially feature a total of two teams that are identical in every aspect except graphical representation. Each team will initially consist of one player.
Level - The arena on which the game is played.
Flag - An artifact that belongs to a team.
Ball - A team-neutral entity used by a player to knock out other players by means of throwing.
Knocked out - A player who is temporarily incapacitated, unable to pick up flags.
Scoring - An act which awards a team with score points. Examples include the act of knocking out another player, capturing the opposing team's flag and returning the own team's flag.

2 Proposed application

In this section we propose an application.

2.1 Overview

In the game, each player is assigned to a team. A player can pick up balls scattered about the game world and then throw them at the opposing team. If a player is hit by a ball travelling at a sufficiently high speed, he will get knocked out

At any point in time, a player may capture the opposing team's flag. He will be able to pick up or throw balls in this state and if he manages to bring the flag to the own team's flag at it's home position he will score.

2.2 Functional requirements

Players should be able to:

a. Start the game
b. Be able to move around
c. Pick up balls
d. Throw Balls
e. Pick up enemy flag
f. Capture enemy flag
g. Return team flag

The game world should:

a. Simulate all game objects
b. Prevent game objects from moving into each other when they collide
c. Generate friction in the game world, so the balls and other things thrown/moved will stop moving after a while

2.3 Non-functional requirements

2.3.1 Usability

It is important that it is easy to do fundamental tasks within the game (e.g. movement, picking up a ball, throwing a ball) when a person is experienced with the controls of the game, while it is of very little importance (at this stage) that the game is easy to learn.

2.3.2 Reliability

N/A

2.3.3 Performance

It is important that the game keeps a steady frame rate (number of image frames presented) at least above 30 frames per second. All actions a player do should result in an instant visual feedback.

2.3.4 Supportability

N/A

2.3.5 Implementation

The game will be realized using the Model-View-Controller design pattern. See the SDD for more information.

The game will be written in Java and use the Java Runtime Environment (JRE). Because of this, the JRE will have to be installed on all computers before the application can be launched.

The game will use two external libraries, LWJGL and JBox2D. LWJGL handles hardware accelerated graphics and input devices. JBox2D handles physics simulation and is responsible for collision detection and response.

2.3.6 Verification

Functions critical to the game engine and game model will be automatically tested. Automatic testing of the model is prioritized above testing of the controller.

2.3.7 Packaging and installation

Executable .jar file

2.3.8 Legal

N/A

2.4 Application models

Here we present an analysis of the domain and the functionality of the application.

2.4.1 Scenarios

The use cases are relatively simple. No need for scenarios.

2.4.2 Use case model

See 2.2 and the use case diagram in the appendix

2.4.3 Static model

See class diagram in the appendix

2.4.4 Dynamic model

See sequence diagrams in the appendix

2.4.5 User interface

The user interface will mainly consist of the visualization of the game world. Traditional graphical user interfaces will be avoided. Examples of elements in a traditional GUI are, amongst others, components like dialog boxes, buttons and text fields. Possible ways to interact with the application without these GUI components are using keys on the keyboard.

2.5 Possible future directions

Future network support is a long-term goal and the software architecture will be designed with this goal in mind. The game will be extendable and new features may be added after the course deadline, such as player abilities, more/better levels, vehicles, strategy elements and/or sound.

2.6 References

APPENDIX

- The use case diagram is located at: **gentlemen/documentation/usecases/Use cases.pdf**
- The class diagram is located in: **gentlemen/documentation/classdiagrams/**
- The sequence diagrams are located in: **gentlemen/documentation/ sequencediagrams/**

- A screenshot of the GUI: