**** This document contains the general outline that you should adopt for your Functional Specification ****

Multi-User Boids Game

Alexander Cahill – 15321711

Liam Ó Cearbhaill - 15384941

Functional Specification Contents

0. Table of contents

1. Introduction

1.1 Overview

The system to be developed is a multi-user game with boids making up the main component the users interact with. The aim of the game is to chase flocks of boids from your screen/environment to the other player's screen/environment.

Users will access the game through their web browser and be connected to each other through a web-socket server. The server will host the source code for the game and the client's (user's) machine will be responsible for rendering the graphics, namely the environment and the boids.
No rendering will take place server-side. The sever is responsible for connecting the clients and listening for relevant messages from the clients. Clients will send a message when a boid from their environment crosses the boundary to their opponent's environment. This is the main message the server is listening for, after it receives one, it will tell the other client that a boids has crossed over and that it needs to take responsibility for its rendering.

This client -> server -> client interaction will be the crux of the system.

1.2 Glossary

Boids: An artificial life program which simulates the flocking behaviour of birds. Developed by Craig Reynolds, 1986. The word "boid" stands for a shortened version of "bird-oid object".

2. General Description

2.1 Product / System Functions

The general idea for the functionality of the system is to create a casual game in which users compete to flock the on screen entities off their screens and on to their opponents. The user will open up the game on any supported browser, regardless of platform (phone/tablet/PC), and be able to participate. The “birds” within the game will be repelled from the mouse or area of touch. The system will provide entertainment to those who play it and will provide a practical medium for which we can implement a Boids algorithm.

2.2 User Characteristics and Objectives

Many of the user community will be other students in DCU and examiners.
We anticipate that they will be able to access the game using either their own pc/tablet or a lab machine.
A low level of technical expertise will be required to access and play the game but a short explanation of how to play and how it works will be included on the site.

2.3 Operational Scenarios

Use Case 1: Player visits webpage.

Description: A player visits the game's webpage, hosted on either a lab pc or AWS EC2 instance, and loads the homepage.

Success conditions: The player is granted access and loads the homepage.

Failure conditions: The player is not granted access and the homepage is not loaded.

Use Case 2: Players are match made.

Description: After a player visits the homepage and clicks "Play" they will be match made with another player also currently on the game's site. After this the game will start.

Success conditions: Players are paired together, the game starts and the players are able to chase boids from their environment.

Failure conditions: Players are not paired and the game does not start.

Use Case 3: Game is played.

Description: After players are match made the players will play a full game, lasting about 60 seconds.

Success conditions: The players are able to play a full game and a winner is chosen at the end.

Failure conditions: The players were unable to play a full game and a winner was not chosen.

Use Case 4: Game is restarted.

Description: After players play a full game and a winner is chosen they have the option to play another game against the same opponent.

Success conditions: The players choose to play another game, are match made with the same player and a new game starts.

Failure conditions: The players are not match made with the same opponent and a new game is not started.

Use Case 5: Game is quit.

Description: One of the players decides to quit the game before the timer runs out and is automatically the loser.

Success conditions: The player who presses “Quit Game” is made the loser and the game is ended.

Failure conditions: The game is not ended and the player is not made the loser.

2.4 Constraints

Performance: One of the constraints we may have is the capacity of the users to render the graphics. Since all the rendering will be done client side we cannot fully anticipate the hardware the capabilities of the users. To tackle this, we will decide on an assumed baseline of computing power slightly less than the lab machines.
Our aim is to achieve stable rendering for these machines firstly then work on optimisation for lower spec devices (such as tablets or smartphones).

There’s no library for consistently detecting a machines capability but one of the ways we can detect ourselves is timing how long it takes a client to render some graphics. For example, if it takes a client less than 850ms to render the environment then we can determine they will be able to render the higher quality graphics. If they take longer than 850ms to render the environment they will be served the lower quality graphics.

The servers’ capability to listen for messages from the clients may be a constraint, depending on where we decide to host it and the optimisation of our code. The server will have to recognise relevant messages and deliver them to the client in a timely fashion so that there is little or no noticeable latency. We hope to alleviate some of the latency issues by using an AWS EC2 instance as their reliability and speed is reputable.

Mobile Support: Because this is browser based we plan to support mobile devices. Again we plan to support devices with the baseline capabilities as defined above. One of the areas we may have difficulty with is the touchscreen aspect, although there are libraries that support this.
Another mobile property we may have difficulty supporting is the use of accelerometers as a way of interacting with the game, again there are libraries to support this but as neither of us have worked with them before this may take a lot of time to implement.

Game Design: One of worries is that we spend too much time designing and implementing the game rules. If they are too complicated, we may get sucked into trying to get them to work rather than focusing on the functionality of the system. To overcome this, we plan to implement a simple set of mechanics; that the users must chase they boids from their environment to their opponent's and that after 90 seconds the user with the most boids on their screen loses. After we meet this baseline we can start added more complexity such as different types of boids/animals or changing environment types.

Time: The timeline is an obvious constraint that we are aware of, as we have winter and summer exams. We have to be conscious of the balance study, other project work and work on our final year project.

3. Functional Requirements

Web Socket Server

Description – The web socket server is the central part of the system that facilitates communication between the clients. It listens out for messages from clients telling the server that a boid has crossed over a border to outside of their environment. The client will have to send the coordinates (*x, y, z*) to the server, which will then send these coordinated to the other client.

Criticality – Very Important. This component of the system is critical to its functioning as the project is multiplayer focused and depends on bi-directional communication between the server and clients.

Technical Issues – A technical issue we may run into is the amount of messages the server can handle and process on time.

Dependencies with other requirements – Much of the other components on the system rely on the server's functionality, namely the users' ability to access the game and the clients' ability to communicate with each other so that crossovers to and from environments are properly rendered.

Boids Algorithm

Description – The boids algorithm describes the movement of the flocks to be rendered in the clients' environments. The algorithm is an artificial life algorithm and mimics the flocking behaviour of animals (like birds). The boids will have to be able to flock towards a common heading as a group. Individual boids will have to be able to join large flocks as they encounter them. They will also have to be able to avoid obstacles, in the case of the game this will be some of the walls of a clients' environment and a client's cursor.

Criticality – Very Important. This component of the system is critical to its functioning as it is a central aspect of the game and the flock's movement

Graphical Rendering

Description – The boids will modelled using JavaScript and the Three.js external library. They will be 3D entities whose behaviour is controlled by the boids algorithm and rendered by the clients in their respective environment.

Criticality – Very Important. The graphical rendering of the boids is also critical to the functionality of the game. Without the rendering the users will have nothing to interact with.

Technical Issues – The graphical rendering may present some technical issues to us depending on the computing power of the clients. There may up to a couple of dozen boids required to be rendered in each of the clients' environments at any one time. If the rendering is not stable enough we will have to do optimisation of the rendering by decreasing the polygons and quality of the modelling of the boids. Way may also have to limit the number of boids that spawn throughout the game.

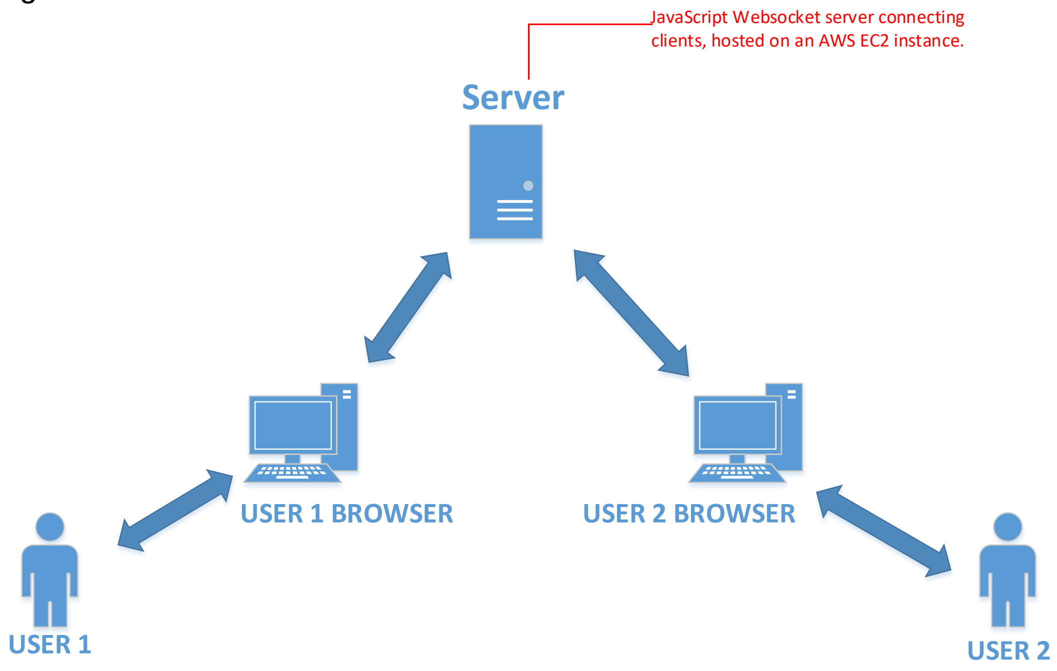
4. System Architecture

The only 3rd party architectural components we plan to utilise will be an AWS EC2 instance to host the server code.
We do not plan to use any other AWS tools, nor do we plan to use any AWS tools to create the operational logic.

The server will host the NodeJS code, HTML, CSS and supply the JavaScript to the user.
It will use mainly SocketIO, a NodeJS Library to implement websockets and handle the networking and matchmaking. When the game is in progress it will also be responsible for listening to crossover messages.
The clients will communicate to each-other through the server.

5. High-Level Design

The below system architecture diagram shows the interaction between the clients and servers as described in the overview. The AWS EC2 instance will simply be used for hosting the code.
The diagram shows how the users will use their browsers to access the game through their pc/tablet and how the server will listen and send relevant messages between them.


JavaScript Websocket server connecting clients, hosted on an AWS EC2 instance.
Server
USER 1 BROWSER
USER 2 BROWSER
USER 1
USER 2

6. Preliminary Schedule

Link to GANTT chart outlining our provisional timeline.