



# RIVERRUSH

## Team 3: StreetlampStudios

Rob van der Berg (4319427)  
Job Engel (4217187)  
Mitchell Olsthoorn (4294882)  
Thomas Smith (4316320)  
Martijn Steenbergen (4311442)

Instructors:  
Rafa Bidarra  
Fanny Lie  
Jesse Donkervliet

# Table of Content

1. Introduction	1
2. General Game Premise	2
3. Setup	3
4. Controls	3
5. Reflection on the product and process	4
6. Description of developed Functionalities	5
7. Interaction Design	5
8. Evaluation of the functional modules and the product	7
9. Outlook	9

## Introduction

Whenever someone goes to a lecture of some kind they often find themselves sitting in an auditorium, waiting for the event to start. Nowadays these people either socialize with the people they came with or they spend the downtime on their smartphones, often ignoring the people around them.

At the start of this project we set out to alleviate this problem by developing a game that can be played with an entire auditorium audience in order to promote interaction and socialization.

- To accomplish this we focused on the following:
- Ensuring that a large audience can play concurrently
- Promoting cooperation and competition
- Making people feel like they have meaningful and personal impact on the game
- Encouraging an energetic environment that encourages people to join

To this end we have developed River Rush.

## End-User Requirements

In order to lead us in this project, we've assembled several more in-depth requirements for the game to hold to. The requirements listed here is merely a small part of all the requirements we had set in our product planning. The requirements are listed in no particular order; all of these requirements share the same top priority.

### 1 - The game must support up to 100 people playing concurrently

In order to make this game playable in an auditorium we need to ensure that all attendees in the auditorium that want to play are able to do so. Due to implementation constraints however we need to limit ourselves to a certain audience we want to appeal to. To that end we've decided to set the upper bound of players to 100. We believe this number allows most smaller scale auditoriums, such as university lecture halls, to play comfortably without having to withhold people from playing.

## 2 - The game must have an active playing component through motion based controls

As said in the introduction, we want the game to encourage an energetic environment. In more practical terms that means we have to require the game to have active input; in our case through motion based controls.

## 3 - The game must not require users to download any sort of application

In order for this game to achieve the vision we had we need to ensure that every user is able to play this game without relying on user constraints such as the need for an Android or Apple Smartphone. Because our game is based on the idea that the audience will not only play this game for just a short duration, but also potentially only play the game once, we require that the game does not require downloading an application. This constraint also vastly increases the speed at which new players are able to join.

## 4 - The game must feature a playable character representative for each user

We want each user to feel useful and impactful. The first step in ensuring that the user does so is allowing them to have their own character which only they can control

## 5 - The game must feature multiple teams that can be divided based on the auditorium layout.

In order to realise our vision of a competitive/cooperative game we must divide the players in a certain number of teams. Additionally, we want our game to make full use of the context in which it is placed so we must make use of the unique environment an auditorium provides. As such we can divide the players into team through their position in the auditorium. This has the additional benefit of encouraging users to interact with the people around them.

## 6 - The game must be playable.

This requirement is very high level because it must be divided into several smaller requirements which pertain to our final product vision. First, the game must feature a game track that records the team's progress. Second, various obstacles must spawn at given locations on that track. Third, the player must simply be able to dodge these obstacles and, if they are not able to do so, be able to fall off the boat. Finally, teams must be able to win or lose the game based on their performance. These requirements are very much specific to our game premise which we will discuss in the next section.

# General Game Premise

River Rush is a cooperative and competitive game that pits two teams against each other in a race to the finish. The game features two boats filled with animals that are placed side-by-side on a river. The boat naturally progresses towards the end goal at a given speed, but its speed is determined by the performance of its players. During gameplay various obstacles will spawn to present danger to the boat and its crew. The players are able to dodge these obstacles if they give the correct input at the right time.

The two types of obstacles that spawn are cannon balls and icebergs. Cannon balls fly over the boat in an attempt to hit animals on top of it. If the cannonball hits an animal the animal flies off the boat and into the water. When an animal is in the water they do not contribute to the boat's speed thus decreasing the maximum speed and giving the other team an advantage. In order to dodge the cannon ball the player must give the command to its animal to jump. If they jump at the right time the cannonball passes under them and the animal lands safely back on the boat.

Icebergs are more dangerous. An Iceberg flows along the river and is able to hit the boat itself. In order to dodge the iceberg players must work together in order to steer the boat in the right direction. Each player can signal the boat to move in a certain direction, but their personal input is proportional to the amount of animals on the boat. In other words, one animal signaling the boat to a given direction only moves the boat slightly. However, if players band together they can move the boat out of danger. If all fails however and the iceberg manages to hit the boat, animals who signalled to the wrong direction are thrown off the boat due to the impact, causing a potentially massive decrease in speed.

The game ends whenever one of the boat gets to the end of the track, declaring that boat the victor.

## Setup

The game consists of two components:

- A main screen which features the game itself on which the players can see the river, the boats, the animals, and the boat's progress on track
- The controller screens. These are featured on the player's personal smartphone that connects to our webapp. The controller features the player's animal, their actions (jumping or signaling), their team name, and their boat's progress on the track. It is here where the player can see where their animal is located on the boat.

## Controls

All controls are made through smartphones of the players. When a player is connected they are able to flick their phones up to let their animal jump. When a player flips their phone to the side the animal signals the boat to move to the given direction.

These controls are best suited for a smartphone, but the game functions on most other devices that have a browser as well. In that case the app will allow for touch screen controls through swiping, or simulated touch screen controls through dragging the mouse.

## Reflection on the product and process

*(from a software engineering perspective)*

Throughout the project we held to the principles of SCRUM. That means we held a short meeting every day (usually at 9:00 for about 15 minutes) as well as two long ones at the end of every sprint on Friday (usually starting at 9:00 and ending at 11:00).

During the short meetings we discussed our progress of the previous day and our planning of the following. During the long meetings we discussed the same on a sprint basis and created a sprint reflection and sprint planning. In the sprint reflection we documented our failures of that sprint as well as possible adjustments we could make. In the sprint planning we would attempt to make those adjustments for assigning tasks and time allocation.

Alongside these meetings we developed a roadmap during the start of the project to guide us on a week by week basis. Unfortunately we fell behind of this schedule very early on. We went through a lot of refactoring and architecture design changes that severely limited our progress throughout the early part of the project. When the time came where we had to deliver our so called First Playable Spikes we hadn't implemented several key pieces of functionality yet. This setback in development persisted throughout most of the project. In some cases we were only able to implement certain functionality 2 or 3 weeks after we had planned them in the roadmap. In the end the early setbacks meant a fair amount of functionality could simply not have been implemented.

Another issue we had throughout the sprint were unexpected loss of development time. Several times throughout the project we hadn't considered certain planned events in sprint plannings as well as during the development of the roadmap which meant we had considerably less time than we anticipated for. For example we were not aware that the beta build meant a feature lock. This meant that the features we had postponed in earlier weeks and the features we had planned in the roadmap after this point had to either be implemented in a limited form or not implemented at all.

Finally, we had some internal team issues. We realised early on that some team members were not putting in the hours we expected from them. This caused a significant loss in development progress and, more importantly, caused the team environment to suffer. Eventually we discussed these issues with the entire team which prompted those team members to increase their participation and productivity. However, we alerted these team members of the problems too late; the damage had already been done and the project was behind schedule and features had to be scrapped.

In the future we must anticipate these types of problems from the start. We should've designed an architecture we would use throughout the entire project and stick with it. Through this we could've avoided significant loss in time which could've given us the time to achieve the polish we wanted early on. Additionally, we should've anticipated planned events ahead of time and based our roadmap around it.

Finally, to resolve our internal conflicts we should've monitored each team member's performance on a week-by-week basis and alerted team members of their lackluster

performance as soon as the conflicts occurred, rather than wait to see if it resolves itself before taking measures against it.

## Description of the developed functionalities

In chapter 2, we gave an initial overview that describes general functionality. These are extended to provide a better game play experience.

### Drop-in / Drop-out

Players can join or leave the game at any moment they wish without affecting the playability of the game, or giving an advantage to any team. For example, when a player joins team A in the middle of a game this team gains an extra player, but no speed advantage.

### Team overflow protection

When a lot of players join a single team this team may get filled up. The game handles full teams by automatically assigning new players to another team. This way, players may not all be on the team they want, but as long as the total amount of players is less than 100, these players will still be able to play.

### Team balancing

It may happen that a game starts with unbalanced teams. That is, one team may have a lot more players than the other. The game's logic handles this seamlessly, as the speed of a team is relative to the amount of players on the team.

### Graphical hints

When a lot of players participate, it may become hard for a player to identify his or her character on the screen. We solve this problem by giving every player a different position on the boat as well as a different look for their character. Along with this, the smartphone module also provides individual feedback to clarify the player's position and state.

## Interaction design (development of the HCI module)

Due to the nature of our product we have decided to study the usability of our product through real user testing. In order to do so we have separated our testing in several parts:

- Developer Testing
- Functionality User Testing
- Small scale Playability Testing
- Large scale Playability Testing

## Developer Testing

These tests were performed throughout the entirety of the project by the developers themselves. These tests were focused on finding bugs and ensuring that given functionality functions properly. Considering the nature of this section we will not discuss these tests in detail. Suffice it to say that these tests have shown us whether or not the functionality works as we envisaged. These tests were purely done from a developer's perspective and thus were not intended to evaluate user interaction.

## Functionality User Testing

Nearing the end of the project we invited some people to our game to test certain functionality. These tests involved a single person in an informal environment where we simply gave the user a certain instruction to perform and evaluated how the user did so. For example, when we were developing our motion controls for the web application we had to research how users would perform their notion of a "flick" motion. Through these tests we discovered that a lot of users would simply make a lifting motion with their smartphone rather than the "flick" motion we had in mind. The tests of this nature we performed gave us valuable insights regarding how a user interacts with our product in a controlled environment and allowed us to calibrate certain features to accommodate for varying needs.

## Small Scale Playability Testing

Further on in the project we invited groups of people to test the game in its entirety. These tests involved around 5 people in an informal environment, but we gave limited instructions to these users and simply allowed them to behave as a typical user would. These tests were performed to test both functionality as well as user interaction. During these tests we would specifically observe the users to gain insight about certain shortcomings of our game. At the end of these tests we would encourage the users to be critical and possibly give feedback. These tests showed us for example that it was very difficult for users to find their representative character on the screen and this allowed us to fix issues like these before critical deadlines.

## Large Scale Playability Testing

At the end of the project we arranged for an actual auditorium with sizeable audience to feature our game before a lecture. These tests involved around 30 people in their "natural" environment. Before the tests we explained our game to our audience including instructions on how to connect to the game and how to play it. After that we started the game and gave the users free reign about how they would interact with it. At the end of these tests we asked the audience to fill in a short google form about their experience.

The results of these tests were middling. The first large scale test we did had some major performance issues as well as scaling issues. The original game was made on laptop with 1920 by 1080 resolution but most beamers have a much lower resolution, which caused our game to not display correctly. We had already made progress with this problem by the time we started these tests, however the feedback showed us we weren't as far as we had hoped.

The other feedback we got was about the framerate of the game. Because we wanted to use the recordings of these tests in our CHI play video we used video recording software to record the gameplay. This negatively impacted our game because the recording slowed down the game a lot and made the framerate stuttery.

The last feedback we received mentioned that the controls were not properly explained. One of the users we had invited to join the tests told us that tilting the phone did not change the signaling flag his character was holding. This made sense because to choose a flag you have to move the phone sideways. This could be fixed by adding a tutorial as an other person mentioned. This could have been done in the waiting screen, but unfortunately we did not have enough time to implement this.

## Evaluation of the functional modules and the product

The developed product is comprised of three separate modules: RiverRush-Screen, RiverRush-Server and RiverRush-Mobile. All three modules must be used in order to run a playable game. A description of them is given below:

### RiverRush-Screen

This module provides the game with a main screen that must be displayed using a projector in the auditorium. When the module runs, it will create a network connection with the RiverRush-Server module. All changes that happen during gameplay will be received by this module and displayed to the entire auditorium. It is possible to run multiple instances of the Screen in order to “broadcast” the game to multiple screens.

### RiverRush-Server

This module takes care of all game logic. It must be run on a computer with open access to the internet or the auditorium LAN. When the Server starts it waits until at least one instance of theScreen module initiates a connection. From that moment on, players can use the RiverRush-Mobile module on their smartphones, tablets or laptops to connect to the Server. Only one instance of this module can be run at a time.

### RiverRush-Mobile

The Mobile module is essentially a web page that provides controls to the users to play the game. After a user loads the web page, a connection is initiated with the Server module. The player can then choose a team that he or she wishes to participate in. When the game starts, the Mobile module can be used to control the game.



These components connect using the WebSocket protocol. Network messages are turned into events that are dispatched to correct sub-components using an event driven architecture. A graphic of this architecture is given below.

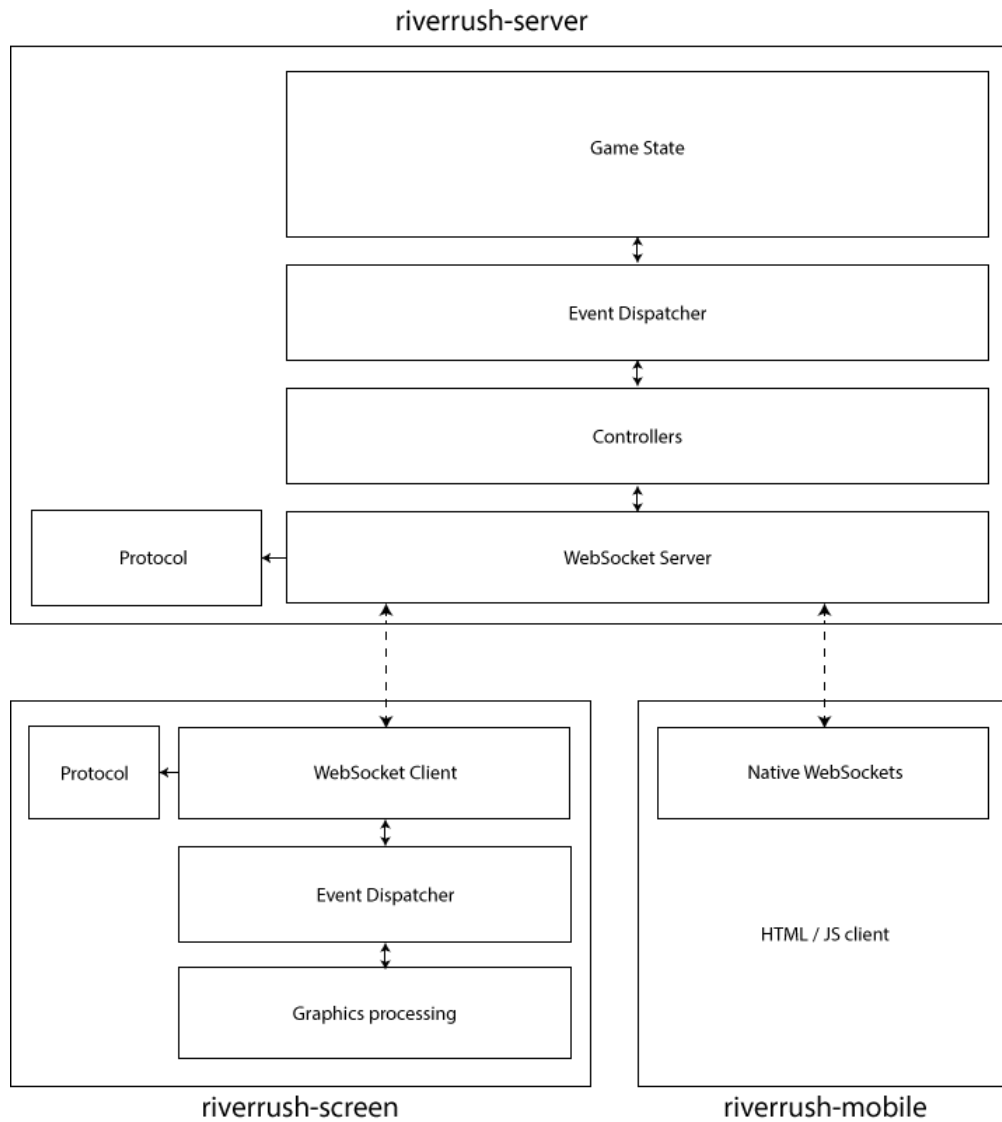


Figure 1: This graphic shows the general architecture of the game. Each module written in Java uses events to communicate things that happen in the game to correct components. All events go through the EventDispatcher class that maintains a list of events and listeners. The game state is maintained using the State pattern in order to avoid tricky branch-constructs.

## Failure Analysis

During the development of the game, we had to make certain design decisions that may impact playability. Potential problems are enumerated below.

### Player count limit

The amount of players that can participate in the game at the same time is currently limited by 100. This is because when only one shared screen is available, the amount of players that can be represented on the screen is physically limited.

### Network quality

The game relies heavily on networking. Therefore, the game should be run on equipment with decent quality networking components. If the network in the auditorium somehow fails, the game will be unplayable.

## Outlook

As mentioned in section 5, we were not able to implement various features in the given time span. Therefore adding the following features would be our foremost concern:

### Difficulty modes

Difficulty modes could allow us to expand our target audience immensely. If we could make a hard mode our game could appeal to more experienced gamers while an easy mode could make our game more accessible to users who are less experienced with games or even younger children.

### Splitting the audience based on their position in the auditorium through position detection

During this project we really wanted the audience to be engaged in the game. Splitting the audience in teams based on their actual location was a major part in doing so. Our plan was to implement splitting the auditorium through allowing users to input their seat number. With that information we could develop an algorithm that translates this to relative coordinates on the screen. For now we simply kept to allowing the users to tell us at which side of the auditorium they were.

### Penalties and boosts based on performance

At the moment, a team could theoretically fall behind early and, if the other team performs well consistently, never recover. We feel this could make for a frustrating playing experience and thus we wanted to implement boosts and penalties to alleviate this issue. An example idea for a boost is if all players in any sector all jump at the same time the boat would be granted a temporary increase in speed. An example for a penalty would work in the same way: If all players in any sector fail to dodge an obstacle the boat could get a noticeable decrease in speed for a small amount of time.

## Idle and malicious player detection

We wanted to implement a mechanic that automatically removes players from the game if they either idle, meaning they do not play for a while, or are malicious, meaning they try to intentionally sabotage the game or their team. We could implement a system where the app automatically sends a notice if the player hasn't done anything for a given amount of time to counter idle players. We could also implement a system that detects unusual gameplay to counter malicious players, though that would require a lot of playtime to discover what we could interpret as unusual gameplay.

Additionally, we are not yet satisfied with our graphics:

## Better sprites

Right now, the animals are rather small and depending on the screen it's displayed on, lose detail. Additionally our river and our river banks are very monotonous. Adding some more flora and fauna to the river would greatly increase the diversity of the track.

## More animals

Even though our initial plan was to have only monkeys, at some point we wanted to fill the boats with all sorts of different animals and make the game's theme be a sort of non-religious noah's arc.

## Animations

Even though our river can flow, our boat can move, and the animals can jump, animations are still very lackluster. The boat gives no indication that it is actually moving, and collisions give no indication of it aside from either the boat slowing down or the animals falling off.