Lappeenrannan teknillinen yliopisto

School of Business and Management

Tietotekniikan koulutusohjelma

Finnish Game Jam: Report

Mika Savolainen 0438617

Oskari Liukku 0435843

How Gameday was made

Työn tarkastaja(t): Antti Knutas

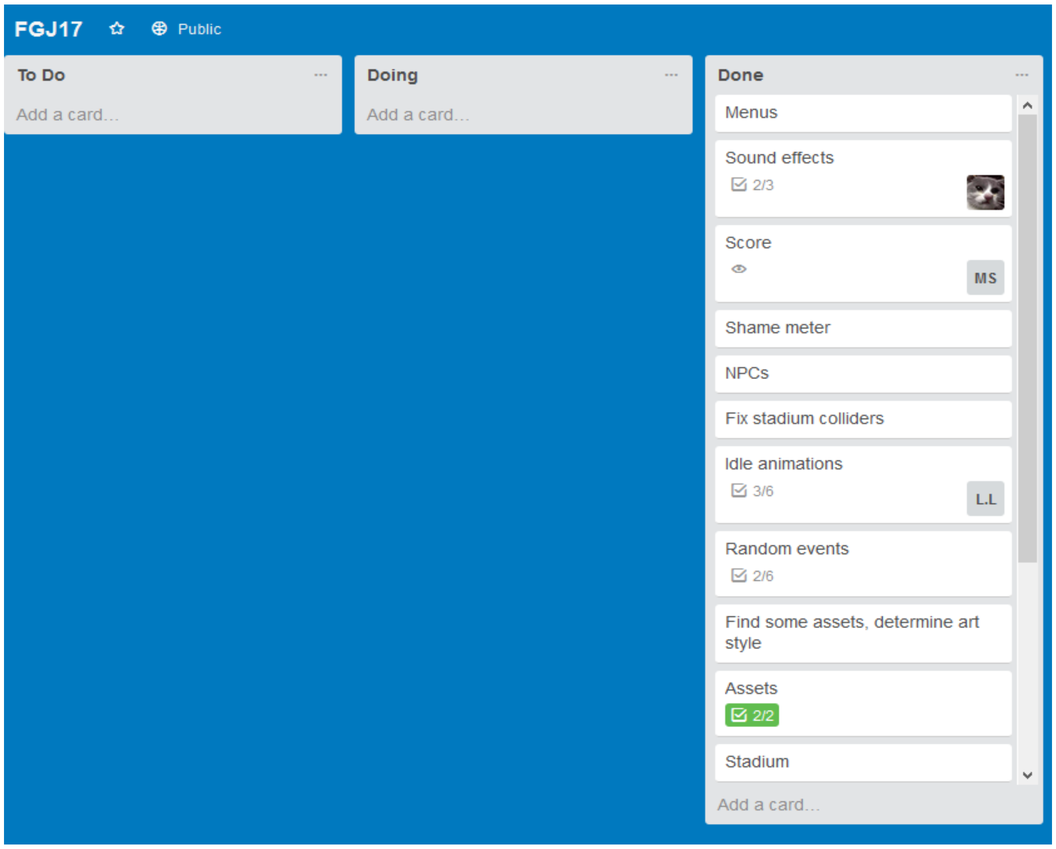
Työn ohjaaja(t): Antti Knutas

Intro

Our game jam group consisted of us two, Oskari Liukku and Mika Savolainen from LUT, with Oskari's friends Miika Kanerva, Lauri Lappalainen and Jussi Joki from Aalto University joining us. Both of us, Oskari and Mika, had attended Finnish Game Jam in Lappeenranta in both 2015 and 2016. Oskari, Miika and Lauri had previously taken part in Ludum Dare 34 as well as held a few private game jam weekends amongst friends. Their game from Ludum Dare lead them to start a company, Improx Games Oy, together with two more friends. The game Trimmer Tycoon ended up on Steam after further development. Mika had been developing games between jams as well, so we all had a respectable amount of game dev experience under our belts.

Except for Finnish Game Jam 2015, where we made a text-based adventure game using Python, Unity had been all of our tool of choice. Oskari learned C# and Unity not long after the game jam of 2015, having been inspired to start making games by himself. Mika learned Unity in 2015 as well and has been using it since. Lauri and Jussi both study Computer Science at Aalto, so coding wasn't unfamiliar at all to them. Miika studies Automation, and because of that isn't as experienced at coding. He's used Unity on and off, but mainly he's the art guy.

As Unity was so familiar to all of us from previous projects and game jams, we didn't even think about switching. We used Unity as our game engine, Gimp as our art-creation tool and Trello for task and idea management. Trello is a great way to keep track of ideas and to-dos without the need for a whiteboard. We seemed to forget it for moments during the jam's excitement, but we did keep updating it, and at the end of the development time, all our tasks had moved to the "Done" slot! The classic pen and paper were heavily in use as well, for visualizing math problems quickly as well as concept art.



FGJ17 ☆ Public

To Do	Doing	Done
Add a card...	Add a card...	Menus
		Sound effects 2/3
		Score MS
		Shame meter
		NPCs
		Fix stadium colliders
		Idle animations 3/6 L.L
		Random events 2/6
		Find some assets, determine art style
		Assets 2/2
		Stadium
		Add a card...

Image 1. How we shared work in Trello

Development

This years theme was “Waves”. We talked before the theme was announced that we could easily spend the first few hours planning and brainstorming. Our first idea was a top-down castle defence game, where you’d have to blow up tanks and troops by causing earthquakes under them. We talked about the idea for a good ten minutes before Mika suggested a game about being in the audience of a football game and having to participate in human waves. We quickly fell in love with the idea: it was simple and had potential to be really fun. It reminded us of My Summer Car, and since that was such a successful game, surely this one would be too. We wanted to keep the graphics simple and sweet and focus on making the gameplay fun.

The finalized plan was to create a game where you have to be part of the crowd without embarrassing yourself too much. You’d have to stay with the rhythm of the rest of the audience doing waves. Another thing that inspired us was that this would work great in VR: maybe we could make a VR version later.

After around 30 minutes of planning we got to work. Jussi became the head of the "coding team" consisting of Mika and Lauri, while Oskari and Miika started working on the graphics. We wanted the graphics to be simple and blocky. We had very little experience with 3D modeling, and this would be our first 3D game. We made all the models inside Unity out of the standard cubes and cylinders. It worked out better than expected. Even the football stadium isn't a model, it's generated by code Oskari wrote out of small prefabs we created. Every object's position is calculated via vectors. This allowed us to change the size of the stadium at any time.

Once we got a working demo going (a few hours of coding to figure out how to make the wave move through the audience), we began testing it every once in a while. Granted, we probably should have tested in longer sessions to figure out how fun the game would be in the long run, but we didn't. Anyone who stopped at our desk got to see how our game worked and try it if they wanted to. We must've had at least five outside testers during the development, and a few more during the finals hours of bug fixing.

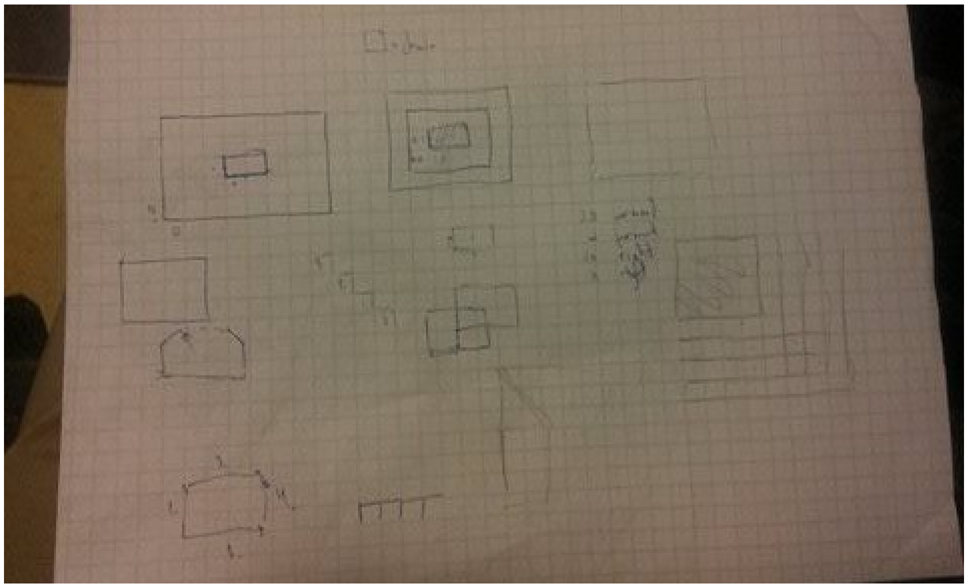


Image 2. Designing how wave would work

The most difficult part was figuring out how the wave was going to work. Oskari had this idea that every person in the crowd could send a message to the persons around him that they should be doing a wave too. We figured that since that would require thousands of colliders and position calculations, it would be quite laggy. Jussi ended up making the waves into classes that just held the waves position and a function for moving the wave. These wave objects could then be spawned and updated through a manager object, and each audience member could check to see if it was near a wave object. The wave sound is actually a separate, gameobject based solution. An invisible ball is moving in a circle on top of audience in sync with the wave and playing a sound of a crowd cheering.

In the end the game came very close to our original plan. We had a few more features planned (like being able to drink a beer while watching the game, or having a popcorn salesman pop up), but those we ended up not having time for. We managed to implement all the core features however, and the game is a great demo of our idea. We successfully figured out which of the features we had planned were part of the minimum viable product, and implemented those first. As we had a few extra hours, we managed to squeeze in some extra features, like the “simulation” of a football game on the field of the stadium, working scoreboards, and idle animations for the crowd.

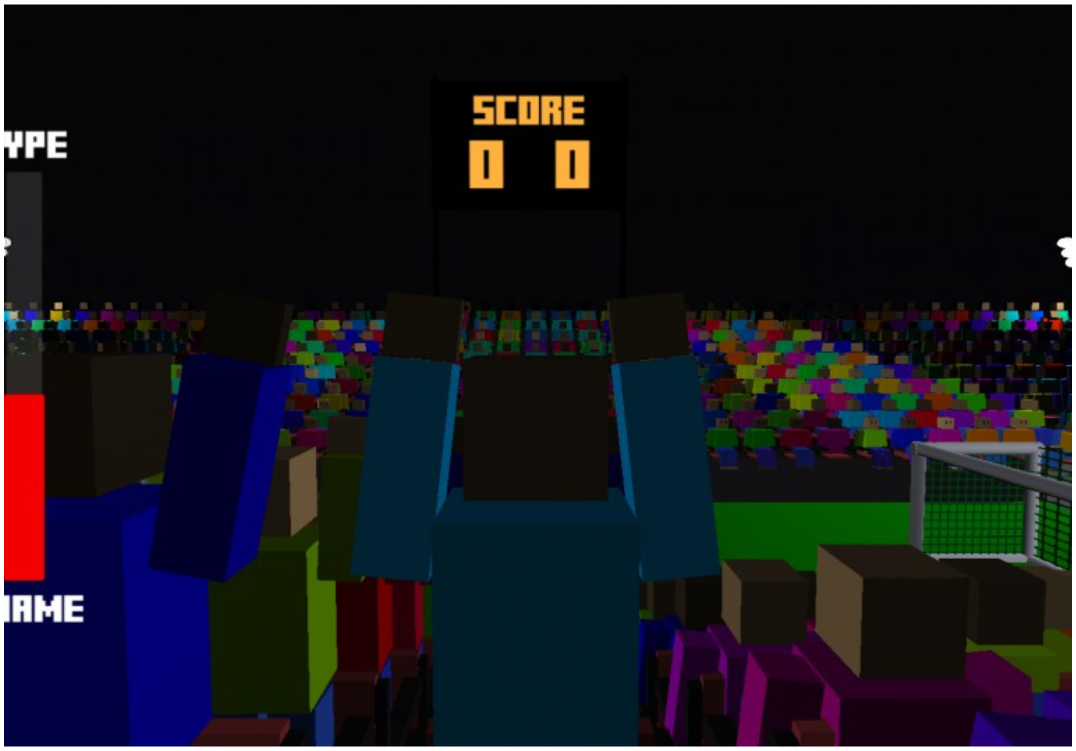


Image 3. Score for football game

Most of our game design was made in the start and we kept most things way we designed them. We had clear plan what we would create but we didn't have plan for how we would do it. We didn't have any specific architecture in our game and some parts of our code were a mess. We wanted to start working fast so we just shared tasks and started working on different components. This wasn't an issue till we need to use other people's code. Everyone was coding the way they are used to and sometimes it was hard to understand other people's code. In game jams most of the time you need to focus on the making game work and less time making the hidden parts pretty. Sometimes you need to hardcode features, because creating them properly would take too much time. This will make continuing the development harder.

Biggest feature we wanted to add was VR. Our game would have been perfect VR game, but we didn't have VR glasses so we stuck on traditional game. Most of our laptops couldn't run the VR game either so the development would have been limited.

One other feature we wanted to add, but technology limited us was to have every person have their own audio source so crow and wave would feel more alive, but Unity limits amount of audio sources based on the computer. Because of this we made game object with audio source that follows the wave so player would know when wave is coming near him.

One of our biggest issue was that everyone had different Unity version. At first we didn't think this would be an issue, but at one part our project stopped working on the old Unity versions. Luckily just updating the Unities fixed the problem but it wasted a lot of time and prevented few persons from working mean while.

Our development was straightforward and we didn't agree on any systematic software development methodology. At the start we made a list about features we wanted in the game and we kept adding features to the list when we thought of something we could do.

If you used some word to describe our development methods, it would be iteration. Every time we added a feature we would test it and make little improvement to make it feel better. For example you gain score more faster than you lose so standing up one second too late doesn't feel too pushing.
Everyone in our team prefer game engines because it is easy to start working with them. Sure creation your own engine gives you more freedom, but we are not professional enough to take advantage of the freedom. Even some “bigger companies” don't make their own game engines, because it saves them time. You don't need to reinvent wheel every time.
For game jammers engines are great asset because it helps you to iterate fast and spend less time on all the non important parts of the game creation like setting up update loop. With game engine you can focus on the creation and designing.

We had many features we wanted to add but time was limiting us. We realized our game got boring pretty fast and it kinda easy to just not lose.

If we continued the development of the game we would add more interaction to the game for example buying food and throwing bottles to other people. Game also could have different stages so player would visit different stadiums and try to not shame himself. We also thought it would be fun to make game bigger and let player leave the stadium and move around the city.

We wanted to stay true to the theme so we focused on the waves. If we redid the game we wouldn't focus on the waves. The waves would be just one feature of the game and the stadium would be more interactive. We would let player yell, start fights and buy drinks and food.

As usual game jam was a great event. It was fun to have break from school work and more “serious” programming and just create a game. All of us love game development and having time limit prevents procrastinating and forces you to focus on developing.

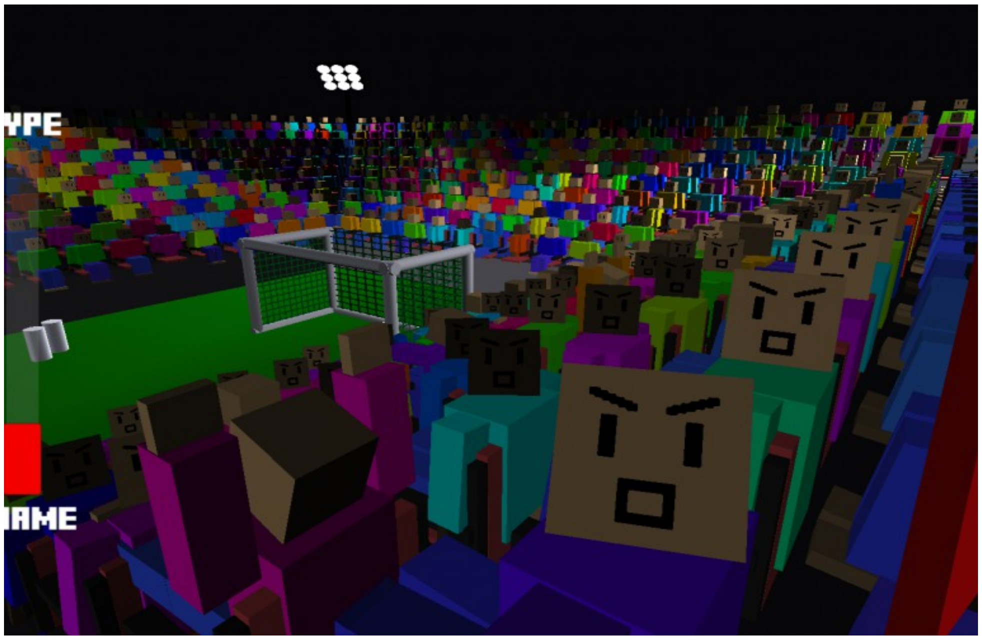


Image 4. Small details to make game feel better