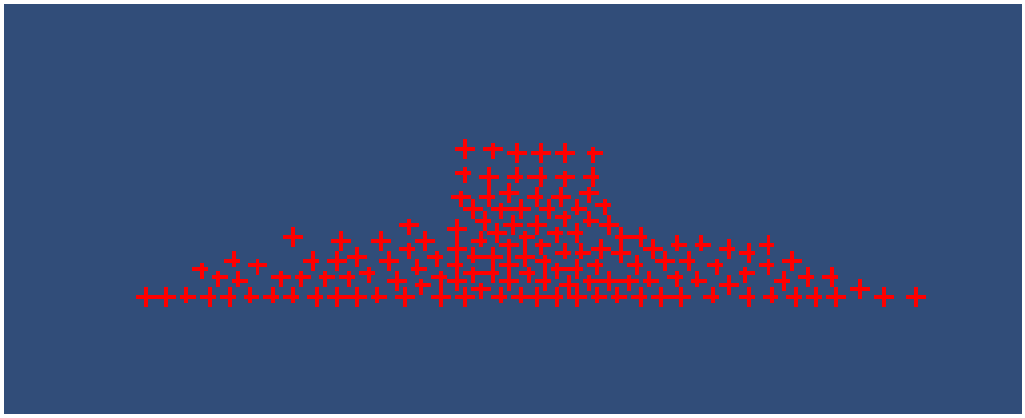
Interim Report

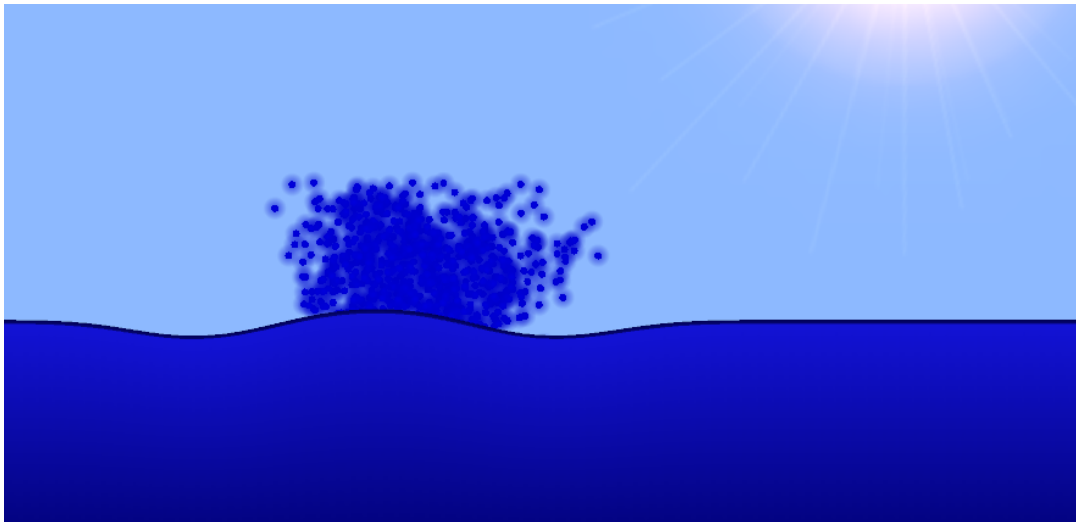
Since our first playable demo presentation, we have worked on refining our water simulation. As mentioned during the presentation, we decided that we would attempt to implement fluid simulation based on M. Müller-Fischer's paper "Position Based Fluids" in Unity. After a week and a half of hard work and debugging, we had a functional fluid. The only problem was, it did not perform well.



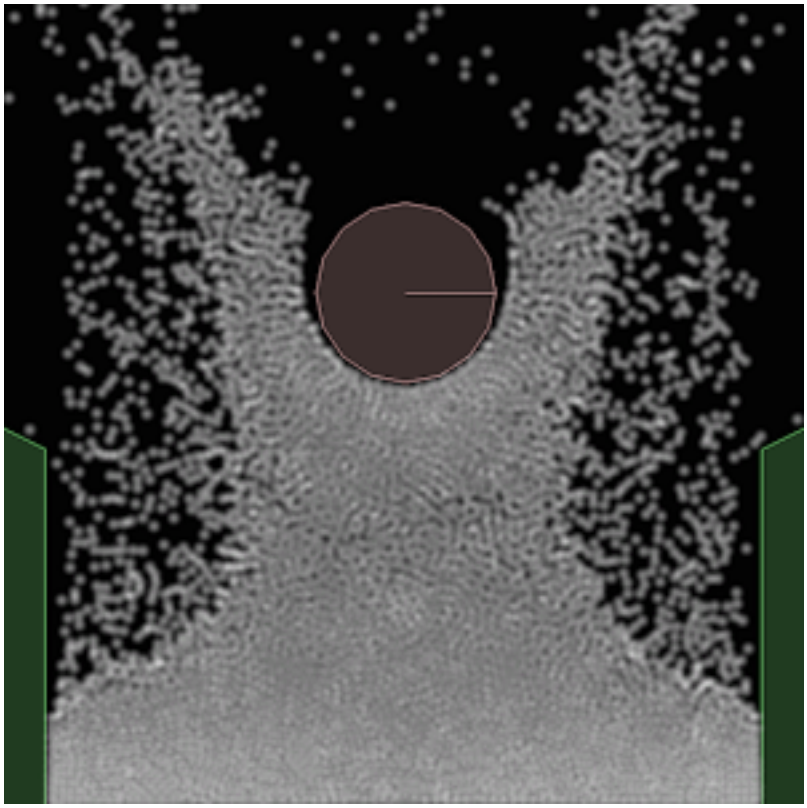
In the image above, you can see our particle simulation in Unity. The particles are represented in code as position coordinates and rendered on screen as small crosses for better performance. The image above has 144 particles and works at 50+ frames per second however we already run in to major slowdowns when we have 300 particles.

We attempted to find ways to increase the speed of the simulation. Using multithreading helped a bit, but not much. Also, we found that Unity provides a way to use the GPU through something called compute shaders however this would only work on windows, while our target device is an android. Finally, writing code directly in c++ and accessing it in Unity would also be possible but writing and integrating this code would be cumbersome and impractical with the projects restricted timeline.

We decided to attempt a backup plan. We had 2 options, which we mentioned at our first playable demo presentation. The first was to drastically simplify our simulation, just simulating the water surface with no particles as seen below.



The second was to use an open source portable cpp library for simulating fluids and rigid objects called liquidfun, as seen below.



We decided for the second option, since the water simulation would be more enjoyable to interact with, even though we were unsure how we would integrate this with Unity or if its performance would be good on the Unity framework. Also, the fact that it is open source allows us to modify or extend parts of it when needed.

We proceeded to try to integrate liquidfun into Unity. Unfortunately, there is not much documentation online on how to do this. Even though we were eventually able to access simple c++ code from Unity, to access liquidfun would be much more complicated and we were still unsure about the performance. Due to our time restriction, we decided to simplify things and access liquidfun directly through an Android project using java native interface invocations. This basically means we decided to scrap Unity.

To help us develop our Android game directly with the Android SDK, we adopted a lightweight open source game framework engine called AndEngine. This framework helps us structure our game code and allows us to reuse basic game elements. It is also simple enough to understand quickly and easy to extend.

In the past few days, we were able to integrate liquidfun into AndEngine and create our first playable demo with water simulation. Right now, we have sea simulation with 1500 particles and ships spawning on either side, fireball slingshot launches and a magical bird. All collisions are being handled and frame rates are around 50fps on an android tablet.

In the past weeks, we had major setbacks however, with the new frameworks we adopted, we feel that we are back on track. Developing directly on the Android SDK does involve writing more code, however we believe that we can make a much better game in shorter time without Unity.