Battle Tinker - Interim Report

Sandro De Zanet, Fabio Zünd, Marco Rietmann

April 14, 2008

During the last three weeks Battle Tinker has made a big leap forward. We focused on functional features such as the rigid body solver, building a ship in the editor and collision detection/response in order to be able to 'play' as soon as possible. Regarding our project schedule, there are some tasks left to do that lie in the low target. Other tasks, on the other hand, that lie in the desired target are already finished. As expected, we have some trouble sticking to the initial schedule since some issues take enormously more time to complete, other issues come for free, and some tricky ones appear from nowhere. Basically, we think we make good progress.

1 News

Since space is visually boring, besides the skybox, the sun and the moon, we wanted to introduce a space station. It consists of a high, stretched building with pads, where the players would start. It represents the middle of the arena and would act as a reference for the players to better orient themselves. We modeled a very simple station to begin with and to see how it acts.

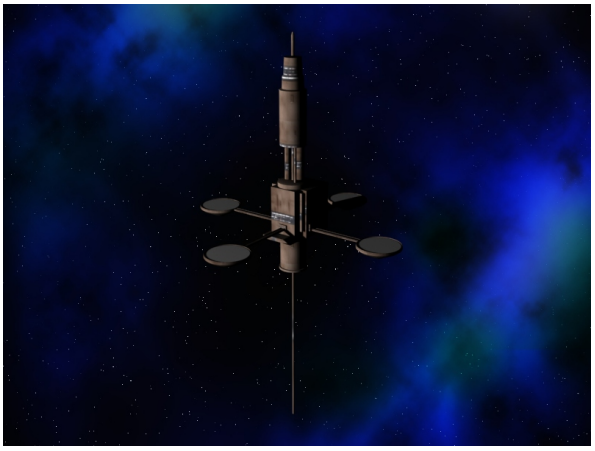

Figure 1: Space Station

Another idea was to implement a Racing Mode. Maybe some player would like to concentrate on flying instead of shooting. We could introduce a Space Racing Mode where players would have to fly around the arena through large hoops. They could still be shooting each other while doing that.

2 Progress

2.1 Battle Arena

2.1.1 Rigid Body Solver

We decided to abandon our ship physics and implement a complete rigid body solver from scratch according to literature. This threw us back on the schedule and took a lot of time but it turned out to be a good idea. Particular problems were that the implementation uses a rotation matrix instead of a quaternion and we spent some time finding an adequate matrix orthonormalization algorithm.

2.1.2 Collision Detection

For our spaceship battle, it is important, that we are able to detect collisions between shots and ships. But we wanted to collide also the ships between each other. For efficiency, we decided to approximate the ships with bounding spheres and test collision among them. It was our first idea to use spheres as well for the collision between the shots and the ships. But it turned out to be better using rays to represent the shots because the whole area between the last position and the current position of the shot needs to be checked.
Regarding the space station, the problem is that it is too big to be approximated with spheres only, without making a big error. So we decided to use polygon based collision detection on an approximated model of the space station. But we have not finished this at the moment.

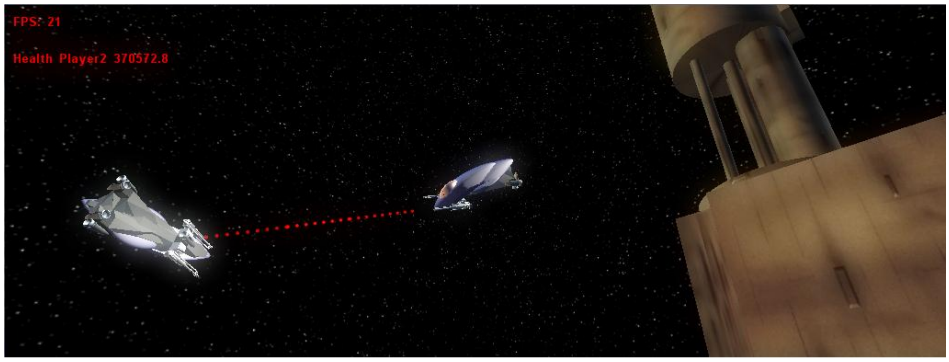

Figure 2: Collision Detection

2.1.3 Collision Response

It was difficult to find a good interface between our collision detection and the collision response. At first, we wanted only to apply force on a point on the ship on each collision. But we had to change that interface as we needed more information about the other colliding part. Finally, the collision response was programmed by the same person who has programmed the collision detection, such that we didn't lose too much time defining new interfaces. The collision response now works well, and we are lucky that we did not get more troubles with it.

2.1.4 Graphic Effects

We use particles for most of the effects in the game. We found some really nice examples on the web, e.g. how to use the graphics card to calculate the particle system. This makes it efficient for even large numbers of particles. We had a first version of shooting smoke already on the first presentation even though it is actually part of the desired target. Now, we use one particle system for engine fire, one for a stride behind the engine and on for the laser shots.

Figure 3: Particle Effects

Initially, the new SkySphere looked nice. We need to create .dds cube map files for the texture. This can be done in the DirectX Texture Tool which comes with the DirectX SDK. A problem is that there are six textures loaded into one dds file (one for each cube side) and for 1024x1024px textures, the dds file gets rather big, as for 2048x2048px textures (that would result in a very nice, sharp starfield), the frame rate drops to only 2fps. Also, the stars appear distorted near the viewport borders. It does not look like space; it looks like white points that are painted on a rotating sphere around the player. We will need to work on that but with less priority.
We added a new lens flare effect that represents the sun. It will need a bit of fine tuning later.

2.1.5 Battle Arena Schedule

Functional Minimum: Complete! Low Target: Regarding that we didn't plan to add a space station, we have finished the low target. But we really want to add this space station to the game, so we will invest a weak more time for that. Desired Target: We have already finished most of the graphics effects.

3 Space ship model

To be able to pass around spaceship between the editor and the battle arena we had to define a general model for all space ships. This managed to be quite difficult, since there are many different requirements on both sides which had to be met in a coherent way.
A tree structure was quite straightforward and was the best way to represent the spaceship with all its substructures for the editor. All juncture coordinates and their normals had to be defined. But as we already had a rigid body simulation, we had to merge them together and recompute basic data for the rigid body, every time the ship model changed.

Since attributes were added for the various parts of the game, there are still redundant parts left, which should be refactored and unified.
As the functionality is already in the structure, the low target is met entirely.

4 Editor

The first implementation of the editor was able to glue structures arbitrarily to other structures. This had some consequences for the space ship model. For instance we had to define all junctures on the ship and their normals. The child structures had to be turned and translated to their right places according to their own junctures and the ones of their parent structures. We introduced pre-calculated transformation matrices for a faster displaying.
A minor challenge was the navigation on the junctures of the structures. We now navigate with an arrow which shows the actual normal on which a structure would be added. This seems not quite user-friendly and needs some more thought.
With the basic implementation we could build some basic ships, but we rapidly noticed that we couldn't build ships that were able to be controlled. One problem was the open space in the battle arena which had no friction. This problem was easily solved by adding some magic friction to the rigid body solver. The friction can be used to define the game difficulty. The other problem thought was that we couldn't turn the structures on the juncture. So obviously we had to implement the rotation on the juncture which proved to be quite challenging to implement, since there were some nasty details to it.
Mainly because of these problems we didn't manage to finish the editor yet. The association of the buttons to the various structures still needs to be implemented. Luckily, the model for assigning buttons is already available, so this task will only be a matter of displaying the right screen. With these last steps taken, we would have accomplished the low target for the editor.

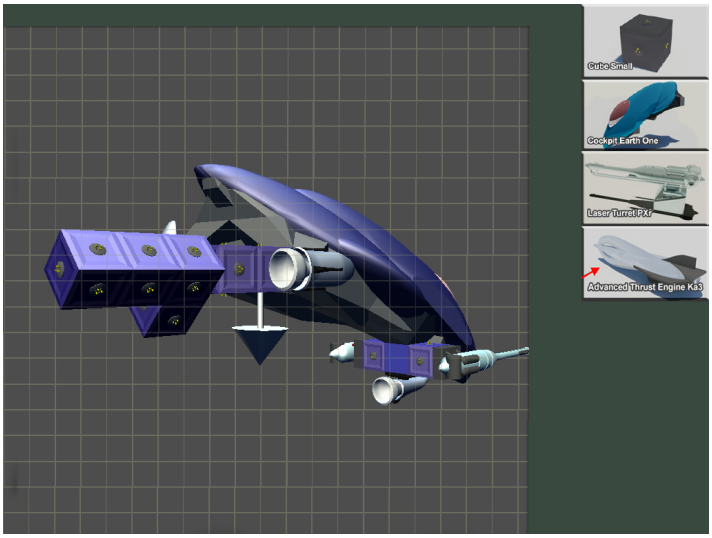

Figure 4: The Editor

5 Future Work

The main tasks for the next three weeks will be to create textures for all models, to create new structures such as energy supply units and to introduce sound. We have already built some sound testing application. Hence, the bare sound engine should not be, as we hope, a large undertaking, more so the creation of music and sound effects.
And, of course, collision detection, some graphical issues (e.g. GUI's), and especially the editor are to be refined further on.