Battle Tinker - Alpha Release Report

Sandro De Zanet, Fabio Zünd, Marco Rietmann

May 5, 2008

This report is structured into many sections each explaining what changes and improvements have been accomplished on Battle Tinker.

1 Game

1.1 New Structures

- The cockpit 'Earth One' has the problem that the junctures are inappropriately distributed among its surface. This results in not being able to build an easily steerable ship. We created a new cockpit 'Beetle' which is very small and has only four junctures - one on each side.

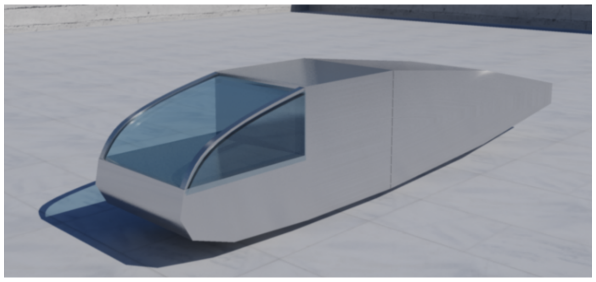

Figure 1: Cockpit Beetle

- In order to be able to give the ship a shape we created two new structures 'LeftWing' and 'RightWing'. The wings are already textured.

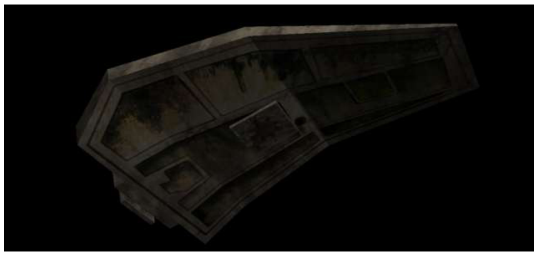

Figure 2: Wing

- A new structure 'Energy Supply 1' was created to be able to add energy storage and energy generation to the ship.

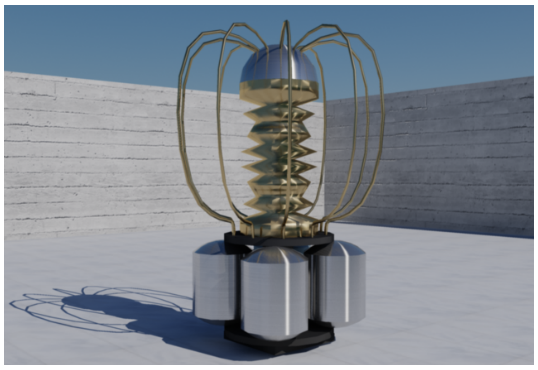

Figure 3: Energy Supply 1

- A new weapon 'MiniLaserTurret' was introduced to enrich the armory.

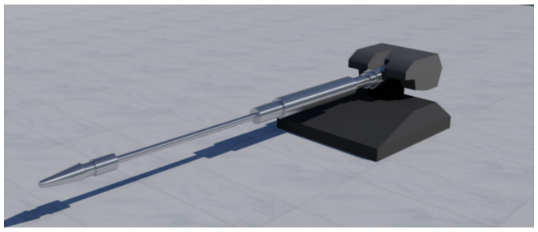

Figure 4: Mini Laser Turret

- We added one defensive structure "Shield". It is fully functional but we will use it not until we introduce shaders on the models to render the shield semi-transparent.

Structures have many parameters regarding health, damage dealing, thrust and energy. These have to be determined and evaluated during the play testing session.

1.2 Energy

We introduced the notion of energy to the game. Every structure now provides an amount of energy storage, continues energy consumption/generation, energy consumption on action and a 'how-much-energy-do-I-need-to-initiate-my-action' value. Energy Supply structures can store large amounts of energy while the cockpit can only store a small amount of energy and most other structures can store no energy at all. Energy Supply structures have a negative continues energy consumption in order to generate energy while e.g. engines have a positive continues energy consumption. The energy consumption on action value determines how much energy is consumed per second if the structure performs its action e.g. fires a weapon or an engine. A structure can only perform its action if there is enough energy left on the ship.

The players HUD displays how much energy the ship can store altogether, how much energy the ship produces/consumes and how much energy the player has left to use at the moment.

1.3 Sound

Creating a sound engine for playing simple (non-3D) sounds is very easy in XNA. Also for 3D sounds there is one helpful tutorial. But achieve more advanced effects is very hard. There are only very few tutorials and help about XACT and XNA for sound on the web so far. We spent days to figure out how one can fade from one cue to another (The solution to this particular problem is to use categories with instance limiting and fading and to assign cues to this category). A sound engine for engine sounds was the first difficult task: The engine should behave like this:

- By default an 'engine idle' sound is played.
- When the user presses the button the 'engine idle' sound fades to an 'engine running up' sound and, if he still presses the button, the latter fades into a 'engine running loop' which loops as long as the button is pressed.
- When the user releases the button the 'engine running loop' or 'engine running up' sound fades into an 'engine running out' sound and then fades into the 'engine idle' sound again.

This behavior was successfully implemented and tested with sounds for an electric motor but is not used at the moment because we lack some engine sounds.

Concerning the sound engine we distinguish the following categories of sounds:

- Simple sounds: Sounds that play arbitrarily and create a new cue for each playing. (E.g. clicks in menu)
- Simple loop sounds: Sounds that loop certain times during the game. They use always the same cue. (E.g. alarms)
- 3D single sounds: Sounds that play arbitrarily but are assigned to doppler, distance and stereo effect and are played once for each player. The cue is also created each time when the sound plays. (E.g. laser fire)
- 3D loop sounds: Sounds that loop certain times during the game and are assigned to doppler, distance and stereo effect. They use always the same cue and are played once for each player. (E.g. engines)
- Engine sounds: As mentioned above.

For the voice warnings we recorded the voice of our native English speaking friend Hanni Hille and used the Vocoder in Cubase and some effects to generate the robot like voice.

Unfortunately, we had no time whatsoever to compose and add any music to the game so far. But, starting now, the priority for music will be one of the highest.

1.4 Storage

To be able to save ships to the HDD we created the notion of Ship Blueprints. Since the structure and ship classes are too complex to serialize we serialize the Ship Blueprint classes, which uniquely define the glue parameters to build a ship from scratch, and store them to the file system.

1.5 Fun with X-Wing

To test our structure engine, we built a fully functional Star Wars X-Wing. We simply downloaded the model of an X-Wing (TM) broke it down to its building blocks (cockpit, engines, turrets and wings) and created the structure classes in the project. The latter was done by inheriting from a few classes and defining the juncture positions. After gluing the structures together and assigning the buttons to the engines and turrets we could go on a joyride with the X-Wing.

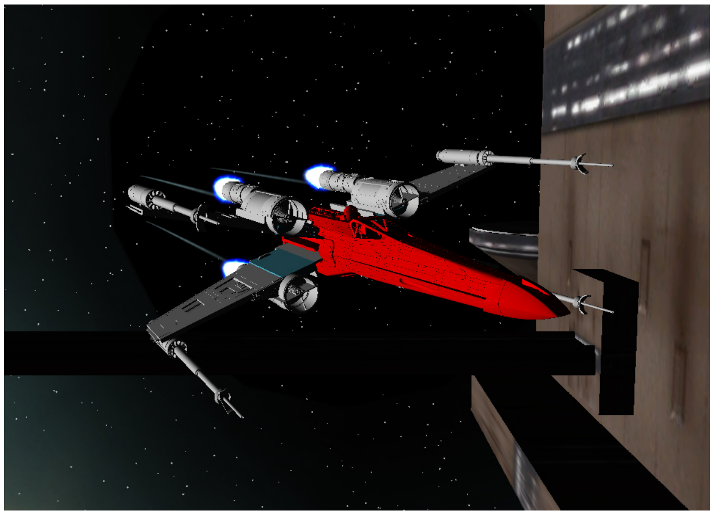

Figure 5: X-Wing

2 Battle Arena

2.1 Skybox

Finally, we achieved to create a skybox that meets our desires. In all previous skyboxes or skyspheres the stars were blurred, distorted and it was slow. Now we use a combination of a billboard content pipeline processor to generate star billboards on a sphere mesh and an ordinary skysphere to display the nebula. The stars are perpendicularly aligned with the looking direction of the camera so they never look distorted. The nebula texture was created in Photoshop.

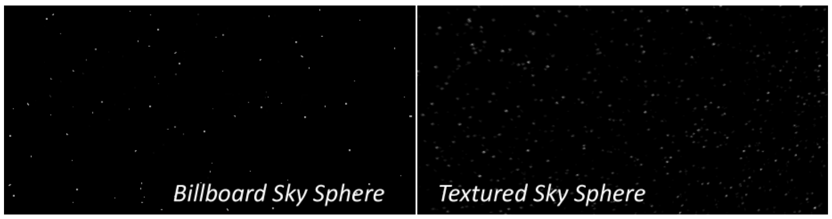

Figure 6: Comparison of skyboxes

2.2 HUD Indicator Arrows

After playing a while with the ships, we found out that it was very difficult to locate the other players once they're out of sight. So we introduced an indicator to the HUD to help the player find the other ships. For each enemy there is an individual indicator which points in the direction of the enemy ship. The indicator is aligned on a circle around the center of the viewport. If the enemy can be seen the indicator disappears. For the implementation we transformed the position vector of the enemy ship to view coordinates by applying the inverse view matrix to the position.

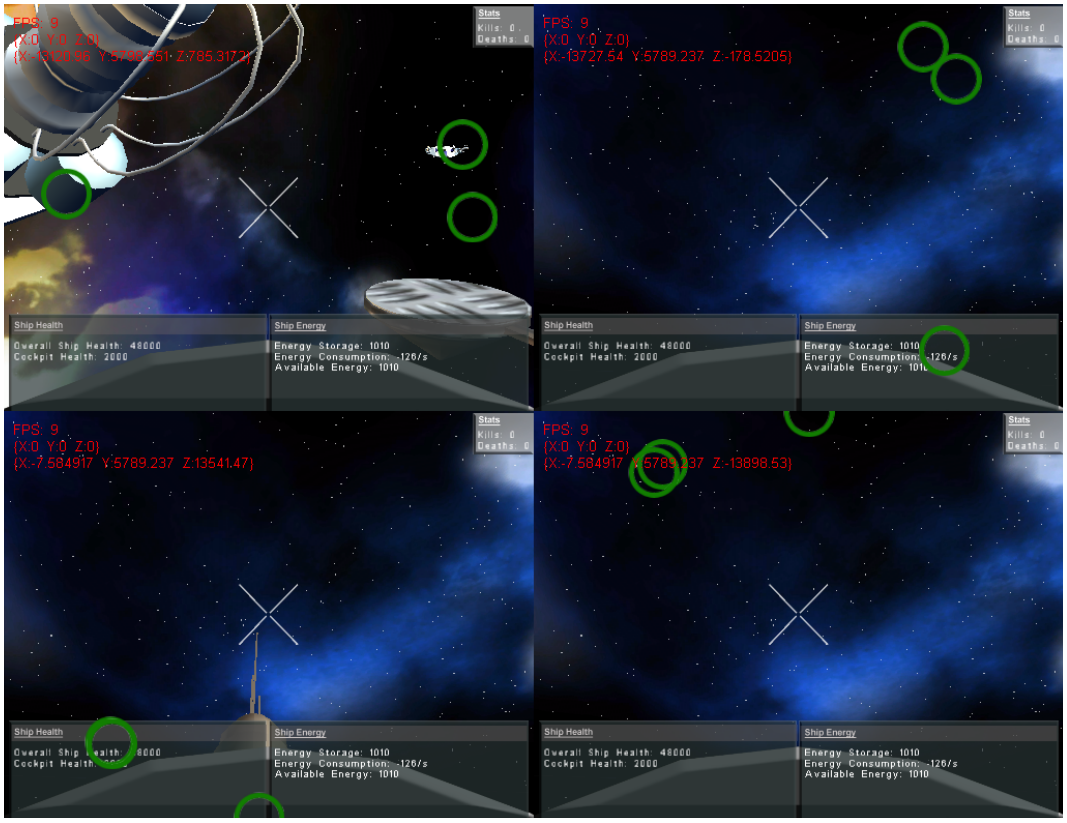
FPS: 9
(X:0 Y:0 Z:0)
(X:-13727.54 Y:5789.237 Z:-178.5205)
(X:-7.584917 Y:5789.237 Z:13541.47)
(X:-7.584917 Y:5789.237 Z:-13898.53)
Stats
Kills: 0
Deaths: 0
Ship Health
Overall Ship Health: 48000
Cockpit Health: 2000
Ship Energy
Energy Storage: 1010
Energy Consumption: -126/s
Available Energy: 1010

Figure 7: Green circles indicate the direction to players

2.3 Collision Detection

We noticed that the collision detection is more important to our game than we thought at the beginning. First, we approximated the ships with bounding spheres to make collision detection easy. But it looks very strange when shots explode far from the actual targets. To make things better we needed more accurate collision detection. In the current version of the game, all objects are approximated with object oriented convex hulls. It took us quit a lot of time to code a suitable generator for these convex hulls. We didn't want to have too many triangles in the convex hulls, so we limited the number of vertices to 100. The Algorithm adds the first 100 vertices to a list, such that every added vertex is the one which has the biggest minimal distance to all already added vertices. In a second step we incrementally generate a convex hull around the point cloud in the list. For searching the collision, we use search trees based on bounding spheres. The leafs of the trees are convex hulls, which then give us a more accurate colliding point than just using bounding spheres.

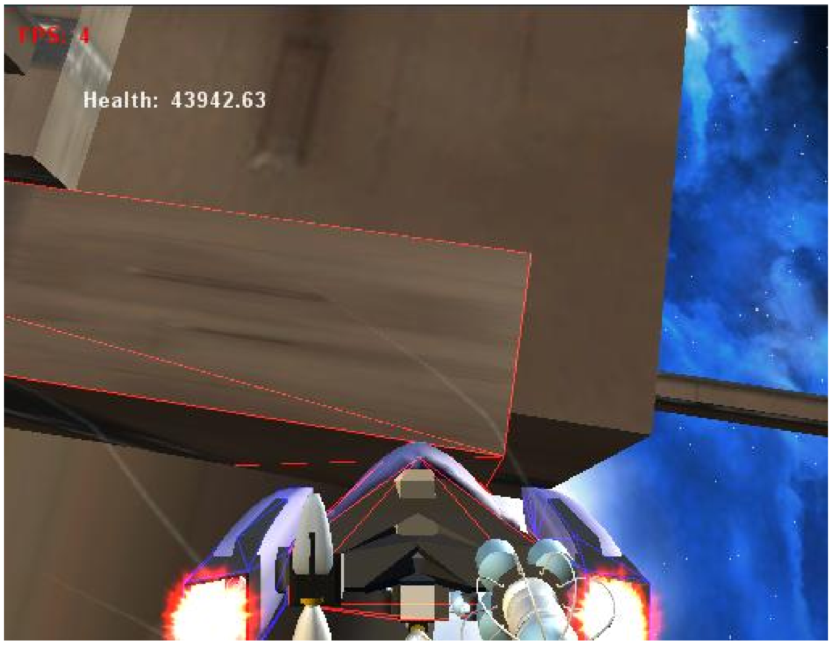

Figure 8: Red: Convex hulls that collide; Blue: Convex hulls that are taken into account for collision from the sphere based collision search tree

3 Editor

3.1 Menu

The whole interface had to be reconsidered, since the first implementation was not usable. We created a new circular menu with icons that implements all the existent and some new actions. The editor should be able to allow the following modifications on a ship:

1. Navigate on all junctures of the ship. We basically left this part unchanged. This means that the user can navigate with the direction pad on a juncture and go to its parent or its children by pushing up or down.

2. Turn structures on a juncture by 45°. This was also left unchanged. The user has to press the LB button to turn the structure on the selected juncture

3. Remove a structure. This action was newly added in the menu.

4. Select and add a structure. This action as well was newly integrated in the menu. The user can navigate through categories of structures and finally select the desired structure to add.

5. Assign buttons to structures. This is also integrated in the new circular menu.

6. Save/Load Ship. The Editor is now able to save a ship to HDD and load a ship from HDD.

7. Go to arena. The menu provides a button to leave to editor and join the other players in the arena.

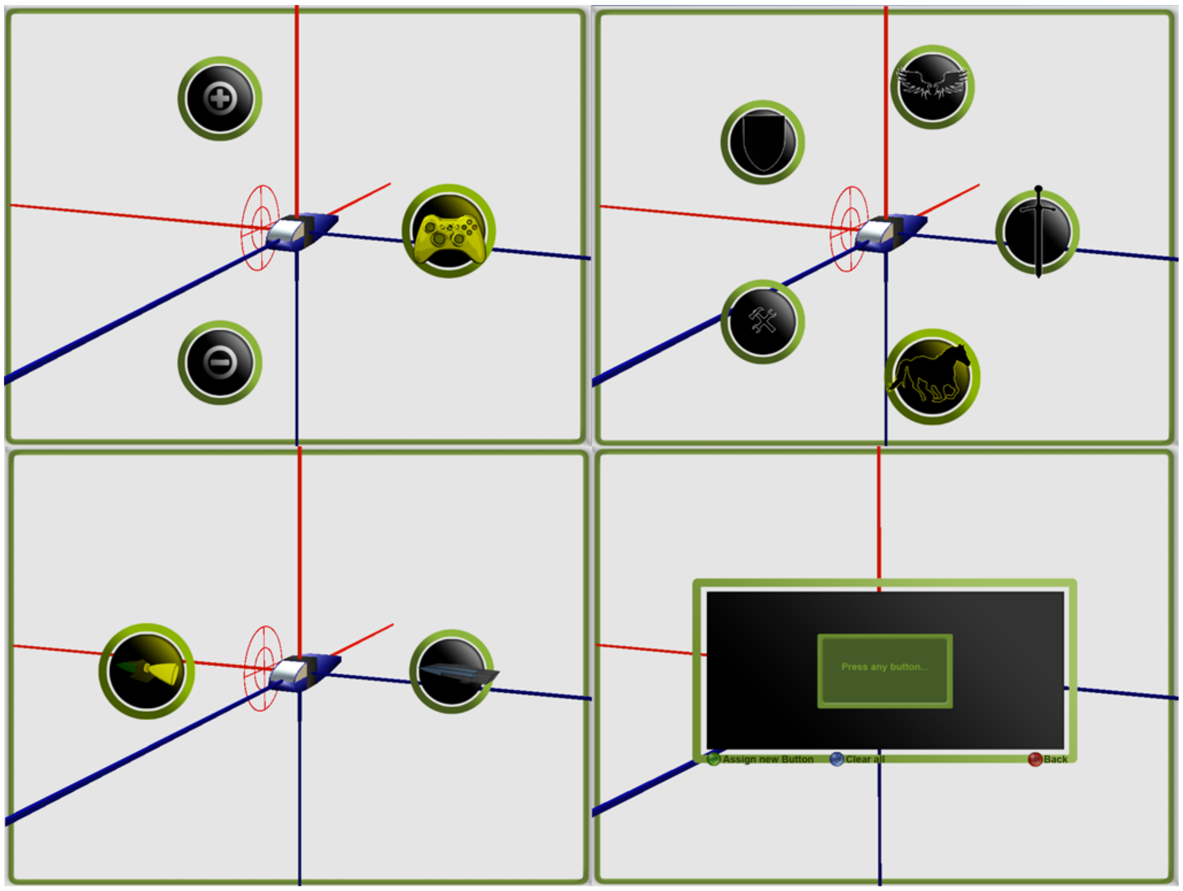

Figure 9: Editor with new circular menu