Mike Dunnegan and Jonah Fidel
COSC 435 Computer Graphics
Professor Fourquet
4/4/16

Final Project Proposal

Statement:

Our goal for this project is to create a game with the following characteristics.

- The player will control a spaceship
- The spaceship will be continually moving through the scene
- The player will be able to accumulate points by flying through rings
- The player will be unable to backtrack. Once a ring is missed, it is no longer collectable
- The game continues until the player crashes into one of many obstacles

We believe that between the spaceship and the terrain, we will have many opportunities to demonstrate visually pleasing graphics, and enjoyable gameplay.

The environment will be a very immersive desert. We will use blender to create a rugged terrain, and export it to our site. We will then use Three.js to apply color and texture to the terrain. We will use bump mapping to make the sand look realistic. We will model other desert objects in Blender and export those too. We will create pyramids, ruins, and oases.

The player's goal is to fly through hoops in a continuous environment. The hoops will be simple torus objects, and we will implement collision detection to register points scored. We will also use collision detection to determine when the player has crashed into a tree, building, or pyramid.

We will implement an algorithm that continually adds a segment of environment to the world when the player gets near. There will be 4 different types of segments. 2 of these are forest and desert. The other 2 are transition segments, from forest to desert, and vice versa

Technical Outline

<u>The infinite runner algorithm</u>

We will implement an algorithm that generates terrain and obstacles every time the player moves forward a number of units. It will terminate when the game terminates.

<u>The Crash Detection Algorithm</u>

We will find a physics engine, or implement our own simple collision detection to determine when the player has crashed, and when the player has flown through a ring.

Random Obstacle Placement Algorithm

We will implement an algorithm that will generate random X and Y coordinates for numerous objects, depending on the current location of the player. This will be in the context of the infinite runner.

Bibliography
We have found tutorials for some of our features

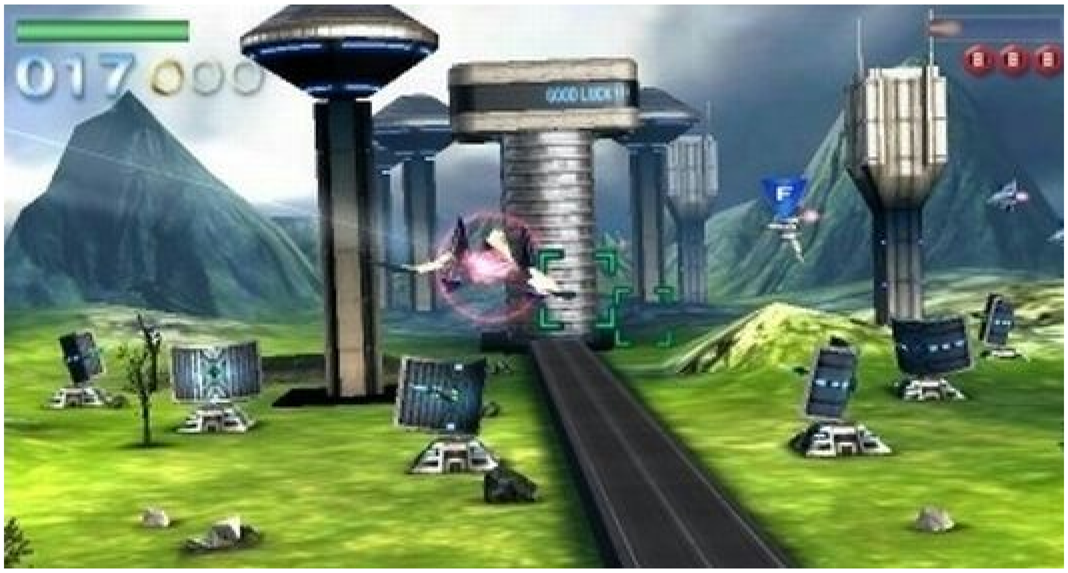
- Texture mapping in Three.js
  - http://solutiondesign.com/blog/-/blogs/webgl-and-three-js-texture-mappi-1/
  - http://learningthreejs.com/blog/2012/05/21/sport-car-in-webgl/
- Bump mapping in Three.js
  - http://solutiondesign.com/blog/-/blogs/webgl-and-three-js-creating-a-real-scene

Objectives

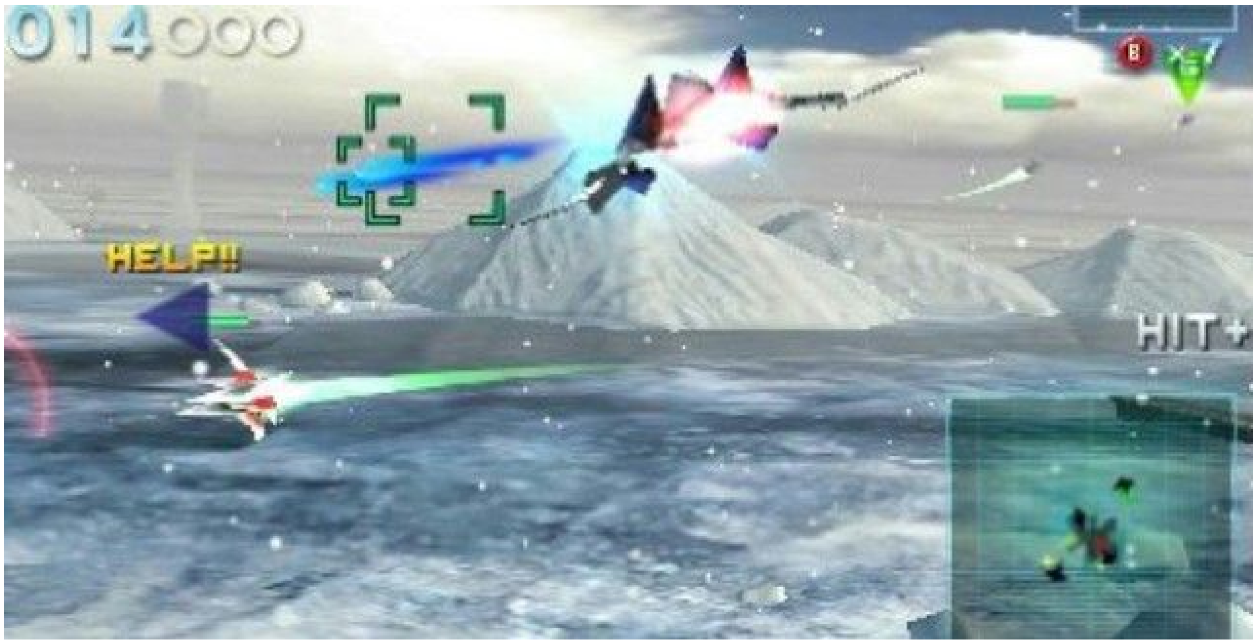
1. Create detailed models for a spaceship, two different types of pyramids. For the terrain texture, we will follow the example of https://github.com/srchea/Terrain-Generation.
2. Get texture assets for our objects online, and use Three.js to map them to our own objects. We will create our own bump map texture for the terrain and pyramids.
3. Build a particle animation for flames in Three.js and apply it as the exhaust of the spaceship
4. Add a physics engine, which will have the responsibility of handling plane crashes. The plan will crash differently depending on how it hits objects. When the plane crashes, the page reloads.
5. Find/develop an algorithm for the random placement of objects, and apply it ..
6. Implement lighting with a sun (main source), and add shadows for terrain, ship, and obstacles. The spaceship will have a light attached to it.
7. Add ring objects to our environment that award points when the player successfully flies the spaceship through the ring. We will use collision detection to know when this happens
8. Build a UI based on StarFox 64, which keeps track of score
9. Add flying functionality to our spaceship (vertical movement through space in addition to the horizontal movement found in the car example)
10. (Updated): the environment has invisible boundaries that the player CANNOT fly out of. When the player hits one of these boundaries an arrow appears onscreen directing the player back toward the middle of the screen.

Examples:
https://www.youtube.com/watch?v=FU-a6ZkF0io

Star Fox 64 3D for the Nintendo 3DS (2011)



Star Fox 64 for the Nintendo 64 (1997)



The original StarFox on the Super Nintendo Entertainment System (1993)

×2
SHIELD
LOOK, LOOK!!