EECE 478 Final Report

Presenting…

Tank.This!

Joseph Au

Andrew Cheung

John Dong

Tricia Pang

Brandon Shi

Spring 2008

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

Tank.This! game is a turn-based tank shooter that allows for game-play in 2D and 3D modes featuring a unique terrain shifting that morphs the playing field from 3D to 2D modes. The inspiration for this game comes from the web-based game GunBound and the feature of terrain morphing was influenced by the game Fez. We incorporated elements from several sources for our models and music to present a very entertaining game that is requires a mixture of aggressiveness and strategy to excel in.

This report first presents the game-play of the game, which detail the controls and strategy of the game. We next show the Graphics User Interface (GUI) and detail the in-game menus. We then discuss the game engine, detailing such things as collision detection, camera movements, object loading, and game physics. We finally show how our models were created, from conception to final designs.

2. GAME-PLAY

Game-play for Tank.This! consists of Control Scheme, Firing Mechanics, Damage Mechanics, and Mini Strategy Guide.

2.1. CONTROL SCHEME

The next two sections details the keyboard and mouse input that controls the game.

2.1.1. KEYBOARD CONTROLS

The keyboard controls of this game change in behavior slightly when the perspective switches from isometric to 2D orthogonal view. However, it remains intuitive.

Button	Type	Description
W	Movement	Move the tank forward in Isometric view
S	Movement	Move the tank backwards in isometric view
A	Turning/Movement	Turn the tank to the left in isometric view; move the tank to the left of the screen in 2D orthogonal view.
D	Turning/Movement	Turn the tank to the right in isometric view; move the tank to the right of the screen in 2D orthogonal view.
E	Aiming	Aim the turret to the left.
Q	Aiming	Aim the turret to the right.
R	Aiming	Aim the gun up.
F	Aiming	Aim the gun down.
N	Turn Control	Skip a turn.
1	Ammo Selection	Select Ballistic Projectile Type
2	Ammo Selection	Select Laser Projectile Type
3	Ammo Selection	Select Missile Projectile Type
V	View Selection	Enable/Disable Projectile View
m	Sound Selection	Enable/Disable Music
ESC	Menu	Open the menus; close the menus if menu is already open. Also pauses the game.

Table 1: A List of Keyboard Controls

2.1.2. MOUSE CONTROLS

Button	Type	Description
Right Mouse Button/Movement	Camera Panning	In isometric view, shifts the camera left/right; in 2D orthogonal view, shift the camera sideways. In isometric view shift the camera up/down.
Left Mouse Button	Firing Control	Holding down the mouse key will gradually increment the power of a shot, effectively controlling how far the shot will travel. Releasing the mouse button fires the shot.

Table 2: A List of Mouse Controls

2.2. FIRING MECHANICS

To fire a projectile, a player generally has the do the following:

- Rotate the body to achieve a course direction angle adjustment
- Rotate the turret to achieve finer direction angle adjustments (Not available when in 2D mode)
- Elevate the gun to achieve the wanted firing angle
- Hold fire button down until desired firepower is achieved and let go to shoot.

All projectiles except for lasers are affected by weather, specifically wind, so their effects on projectile path have to be accounted for when aiming. Each type of ammunition consumes a different amount of energy to fire and this energy is recharged over time or when a player is able to pickup an ammo crate.

2.3. DAMAGE MECHANICS

Each weapon can inflict a maximum amount of damage bullet, being the weakest, laser being medium, and missiles has the greatest damage. Maximum damage is inflicted from a direct hit by one of these projectiles and lower splash damage is done when a projectile lands close to a player. Players should be careful when firing as to not hit themselves by firing straight up for example. When a projectile strikes

the ground, it will damage the ground and create a creator, which can be useful for hiding in. Users can gain more life points by picking up Health Crates located at random locations on the map. A player loses when they run out of life points.

2.4. STRATEGY GUIDE

An effective player has to fully utilize the terrain and objects in the game to excel in the game.

2.4.1. HOW TO SHOOT WELL

The key to getting direct hits is the try and hit from a steep angle because this exposes more surface area for a hit compared to aiming for the front. Use the energy efficient ballistic type weapons to help you gauge range and aim before you unload the big guns such as missiles or lasers as these consume a lot more energy. Also, keep an eye on the timer so that you have enough time to shoot and move to cover after the shot.

2.4.2. HOW TO DEFEND WELL

An effective player has to fully utilize the terrain and objects in the game to excel in the game. For example, a player can hide behind buildings to help shield them from others and only move out to shoot. This hit and run strategy can be very effective against a player who doesn't know how to do the same or counter it. Be careful not to overstay your welcome because terrain and building are damageable and you might find yourself expose from a well placed shot. A possible counter strategy for players who likes to hide is to utilize the 2D map to expose players that would otherwise be hidden in the 3D map. Always attempt to pickup crates throughout the game as these can turn the tides in a losing battle. For example, an ammo crate can give you enough energy to fire a laser to score a direct hit and inflicting massive amounts of damage.

3. GRAPHICAL USER INTERFACE

The graphical user interface is divided up into three major components. It includes: the Heads-up Display, the In-Game Status Texts and the Game Menu System. Each of these major components displays a certain amount of contextual information, all of which revolve around making the game more intuitive and playable to the player. The Heads-up Display includes status information such as player unit health, time left for the round, turn information, direction facing, and wind factor etc. The In-Game Status Texts component give more immediate feedback as to when a projectile weapon hits and does damage to the enemy unit, how much damage is inflict is displayed to the user right in the game playing field in the form of scrolling numbers. Other information such as misses, critical hits etc are also displayed this way. Finally there is the Game Menu System in which the user can use to easily start, setup games, choose units and tanks and playing the game, all without having to mess with command-line parameters. The Game Menu System also acts as assets loading and game initializing/resetting mechanism to allow new games to be started without having to restart the game executable entirely.

3.1. METHODS FOR IMPLEMENTATION

All of the Interface functionalities are encapsulated in the Interface class which keeps track of menu states, pointers to game play entities such as player information, timing and physics etc. The textures used for the GUI elements are also part of the class and are initialized and loaded at the start of the game.

Output parts of the GUI are fed through the glutDisplayFunc(display) function and user inputs are taken in through keyboard, mouse functions.

The actually “clicking” actions are handled using the glutMouseFunc() event and then a class method will decide if the user is clicking on a visible GUI element, triggering the relevant actions associated with

those GUI elements. Similarly, mouse hovering GUI elements triggers a “hover” effect, this helps making the GUI intuitive to the user and makes the interface less “static”. A class private member variable containing the button states is updated and then another class method is called to actually render these graphical changes to the GUI.

3.2. HEADS-UP DISPLAY

The Heads-up Display (HUD) is used to display on screen information to the user. The HUD is divided into two parts. The bottom part of the HUD is its own separate viewport in the orthogonal view showing the health bars of both of the players, the wind direction, and the power of the active shot, the score and buttons for weapon selection.

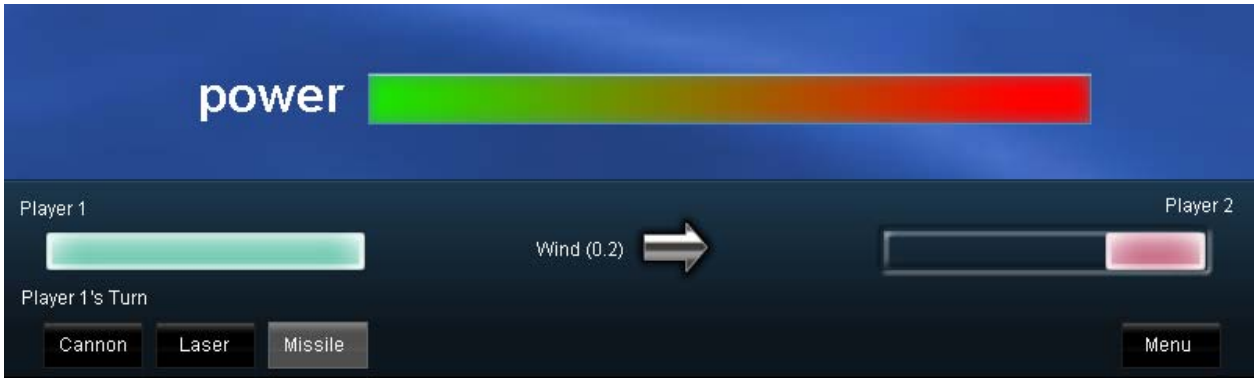


Figure 1: Heads-up Display

The second major part of the HUD’s functionality is the timer bar showing at the top of the screen. It displays the time left for the currently active player to make a move in prominent numbering.

Figure 2: Example round timer

3.3. IN-GAME STATUS TEXTS

The in-game status texts are status change indicators whenever an event occurs. Any changes to the Player classes are reflected by using these status texts. Some examples of the status texts are shown in the following figure.

These texts are drawn as texture boxes in game and animates outwards from the screen by manipulating the coordinates of the quads on which these textures are painted. We found that the uses of animated bitmapped text slow down the game significantly, so instead textures were used.

Figure 3: Examples of Status Texts

3.4. GAME MENU SYSTEM

The game menu system features a multitude of selection screens which walk the players through the game starting process. Users can choose their playing units and maps by previewing the 3D models of those units and maps in each of the menus.

There are four menus: Main Menu, Player 1 Tank Selection, Player 2 Tank Selection and Map Selection.

3.4.1. MAIN MENU

The Resume Game option will only be the GameInProgress flag in the Interface class is active.

Figure 4: Main Menu Screen

3.4.2. PLAYER TANK SELECTIONS

The Player Tank Selection screens are painted with a 2D textured background and the player tanks will be rendered in 3D in the foreground. Pressing the buttons on the left and right hand sides of the screen will toggle through each of the available player units and hitting the “Select” button at the bottom center of this screen will complete the player unit selection and proceed to the next step.

3.4.3. MAP SELECTION

The Map Selection screen will show a scaled down miniature overview of the battlefield and the win/lose statistics between Player 1 and Player 2 during past battles on that battlefield. Similar to the Player Tank Selection screens, this menu can be interacted with by clicking the arrows on the left/right sides of the screen to shuffle through the maps. By hitting the "Start Game" key at the bottom center of this menu, a new game will start with the previously chosen parameters.

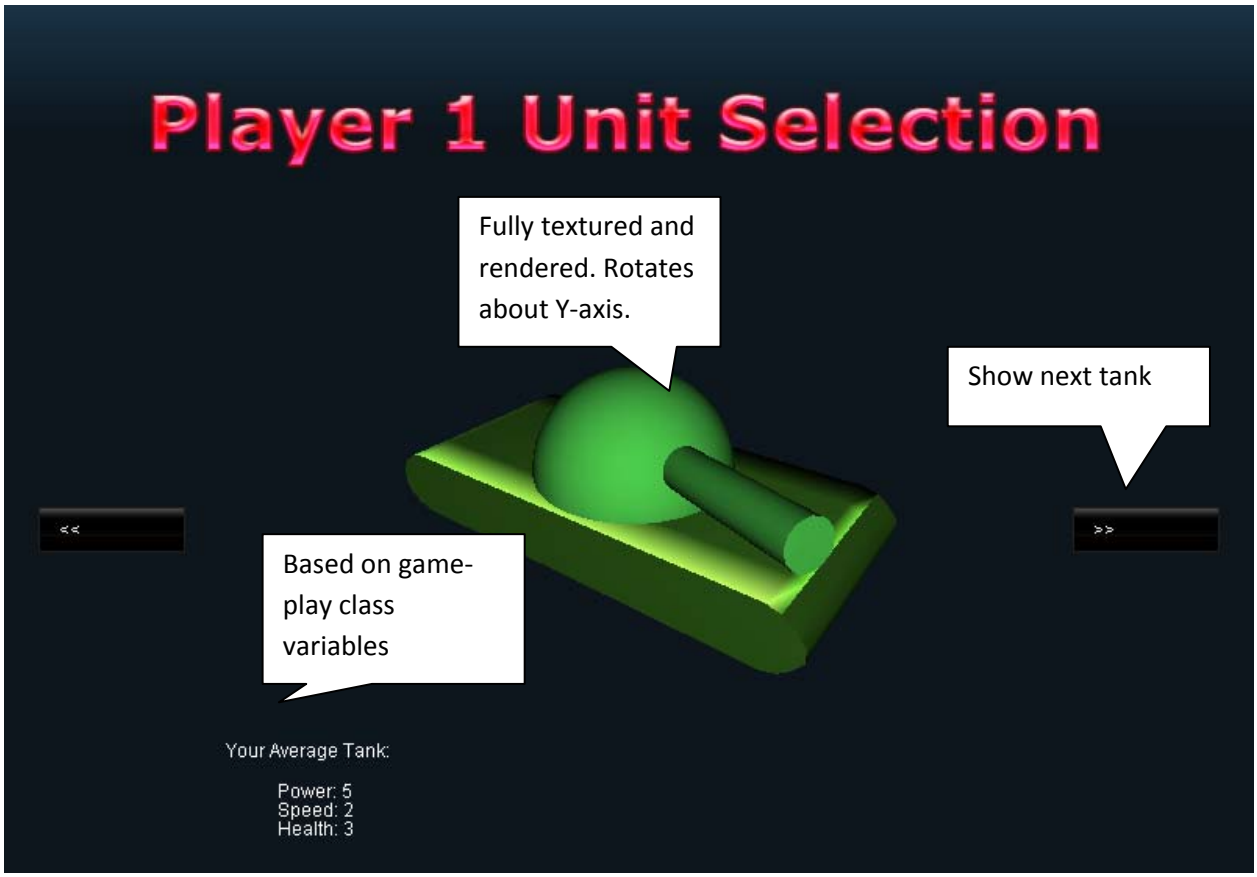


Figure 5: Example unit selection screen

4. GAME ENGINE

The Game Engine for Tank.This! consists of Deformable Shifting Terrain, Collision Detection, Camera Models and Movements, Physics and Rendering.

4.1. TERRAIN

The terrain in Tank.This! consists largely of islands and water. Tanks are able to traverse the land, and the ground is damageable by their bullets. A unique element to the terrain is the ability for it to morph into a two-dimensional view, allowing the user to more easily shoot another player, or travel between islands.

4.1.1. TERRAIN SHIFTING

The most unique feature of this game is the ability to compress the 3D ground plane into a 2D plane so that the tanks can more easily shoot and move. The following set of images show a simple example of this compression.

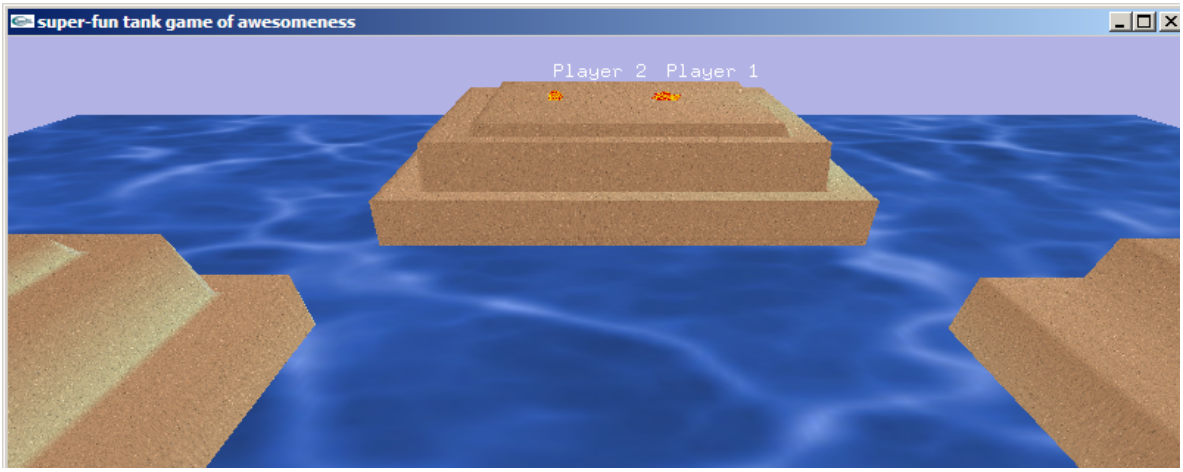


Figure 6: ISO View Before Shifting

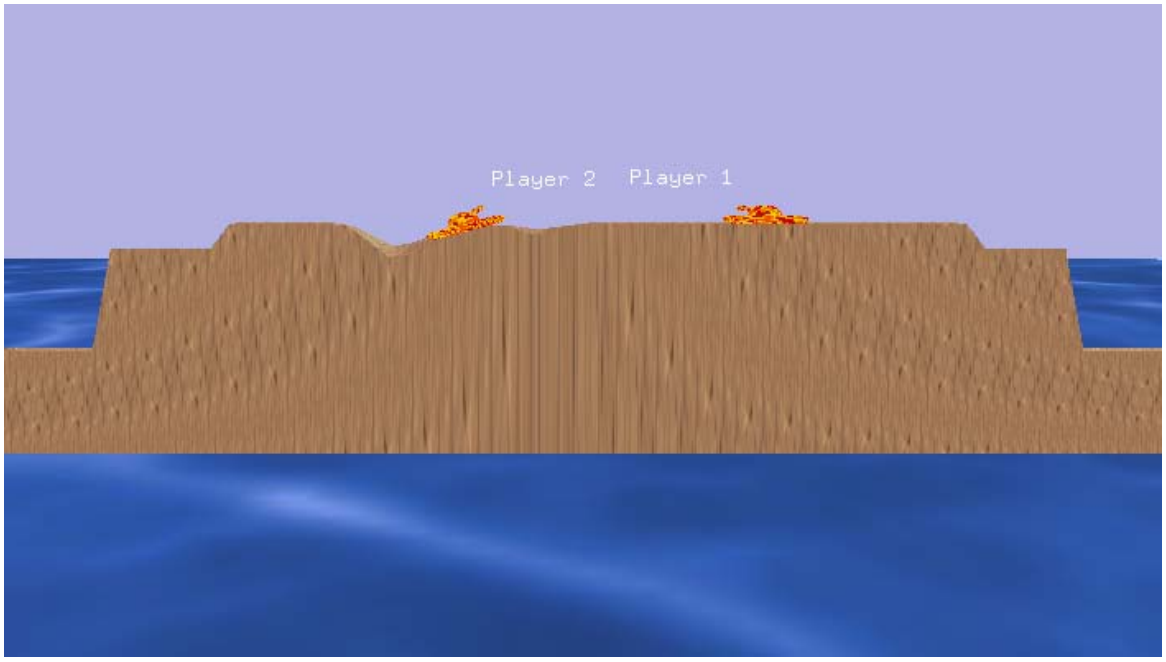


Figure 7: Equivalent flat view (on same island)

In the simple example above, the tanks are located on the same x-axis. As expected, the cross section of the land mass is displayed, with the tanks shown at the same distance from the camera. This 2D view shows the side profile of the tanks and allows the user to more easily shoot the other player.

The following example is less intuitive – the players are on different islands of the map. Upon further investigation, from a 2D perspective, the land masses align when viewed from the z-axis. The 2D view reflects this, and allows the player to traverse from one island to another. Although the movement from one island to another is impossible in an ISO view, the flat view enables the player to cross the body of water.

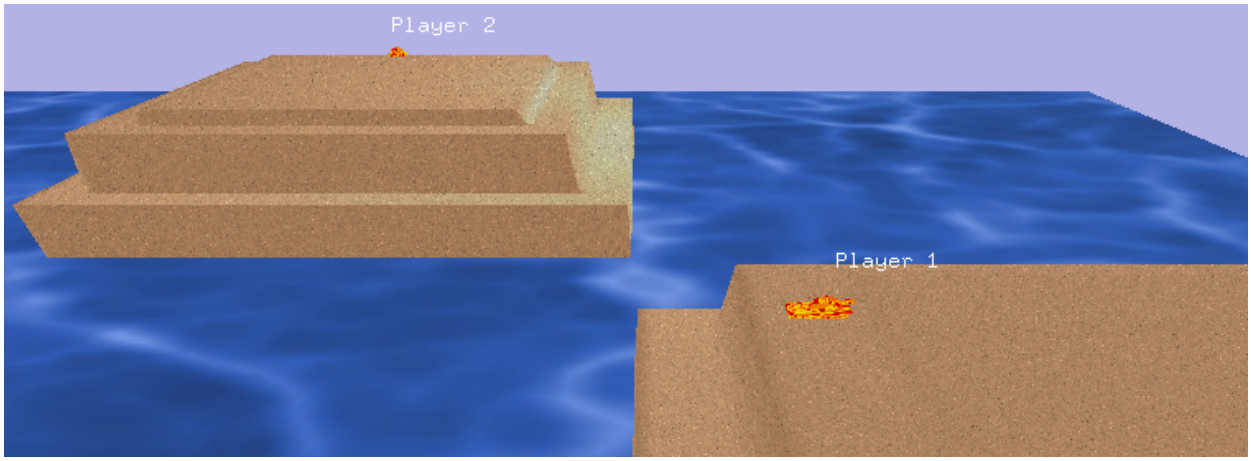


Figure 8: ISO View Before Shifting

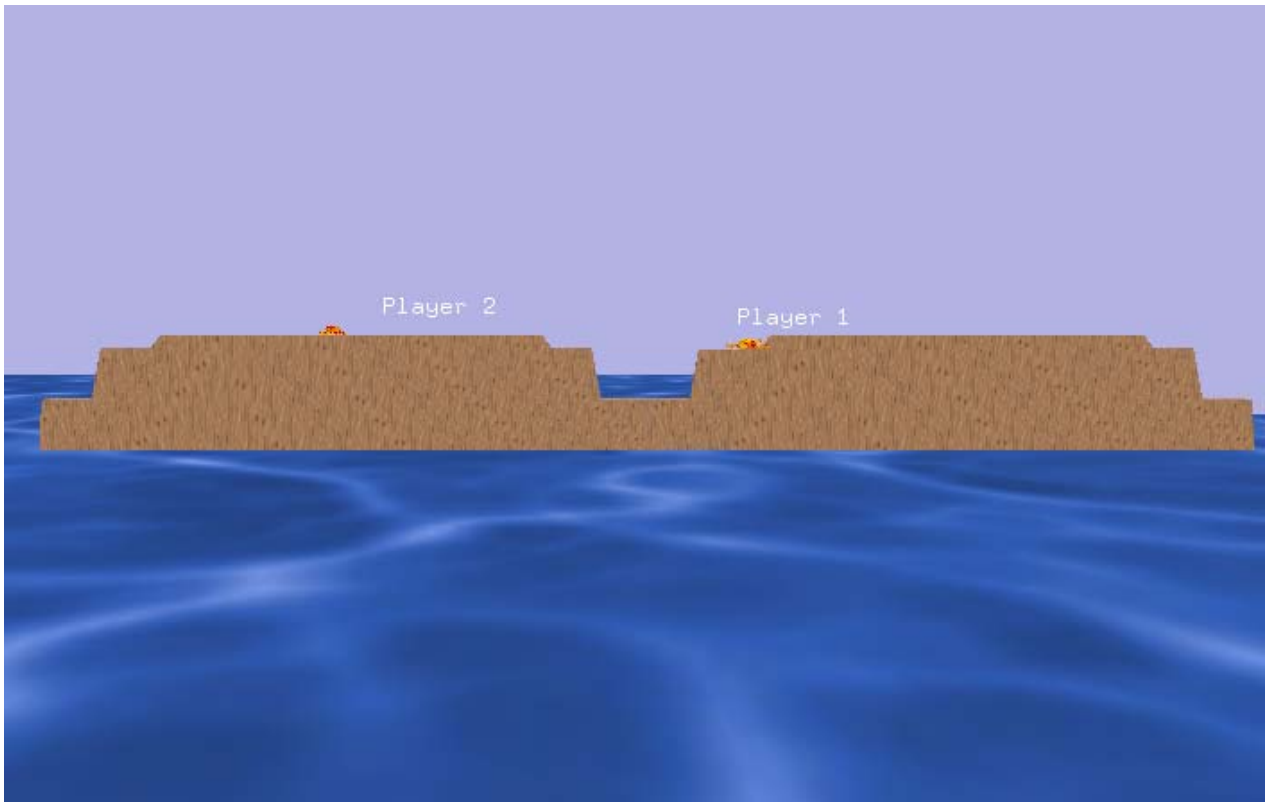


Figure 9: ISO view and its equivalent flat view (on different islands)

The algorithm used for this 2D compression is demonstrated in the following diagram, which shows an overhead view of the map (x-z plane).

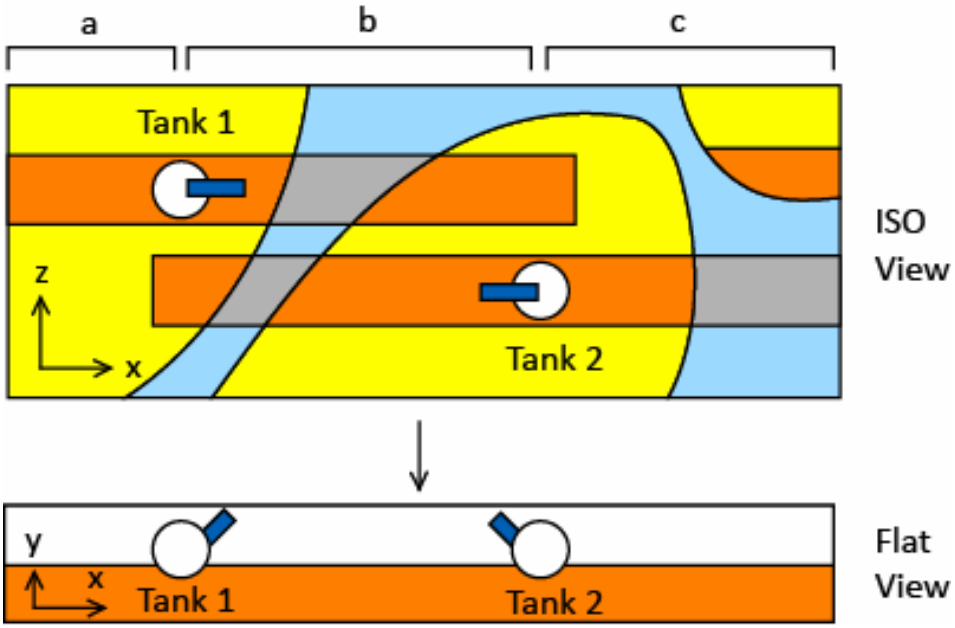


Figure 10: Ground compression algorithm

In a compression as viewed from the negative z axis, the strip of land along the x-axis:

At section ‘a’ is the land at Tank 1’s y-z plane

At section ‘b’ is the average height of the land at Tank 1 and Tank 2.

At section ‘c’ is the land at Tank 2’s y-z plane

An exception to this is the case that the ground calculated from the previous definition is water instead of land. In this case, the closest land along the same z-axis is selected and brought into the 2D view. If there is no land at any point along that z-axis, it remains as water.

When drawing the Flat View, by default the ground and tanks are both translated to a specified point, a certain distance from the camera.

4.1.2. TILES

The ground plane resides in the Grid class, and consists of Tile objects. Tiles are x-z coordinates of varying height. The Tile object also contains information about the face between the vertices (x,z), (x+1,z), (x+1,z+1), (x+1,z). Since a unique feature of the ground plane is the existence of islands, the Tile object also indicates whether the face consists of land or water.

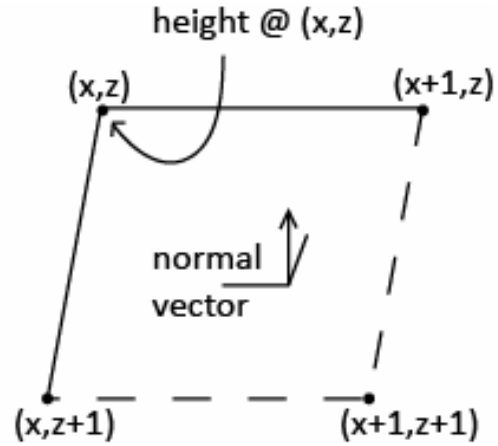


Figure 11: Tile object structure

The entire grid is drawn as multiple GL_QUAD_STRIPs (one for each z-axis coordinate), and their vertices are specified by the Tile coordinates. A texture is also vertex-mapped to each vertex, and the face normal is retrieved from the Tile object.

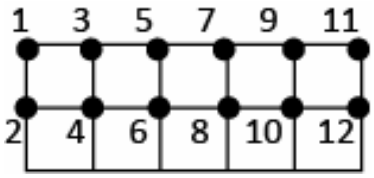


Figure 12: Vertex ordering when drawing ground plane using GL_QUAD_STRIP

4.1.3. IMPORTING MAP

The ground map is imported from a formatted file. This file consists of hundreds of data points specifying the height of the ground plane at each (x,z) location on the map. The imported map files are created in Microsoft Excel, which is a useful tool for viewing the heights of each point in their physical locations and making modifications. For example, a map can easily be generated by using the random function in Excel, as shown below:

Figure 13: Randomly generated ground map

After importing the height at each coordinate, the normal (n) of Tile (x,z) is calculated. Using the heights given by the surrounding coordinates, the cross product of the two vectors (v_a and v_b) that cross that tile is computed and normalizing. This is described by the figure below.

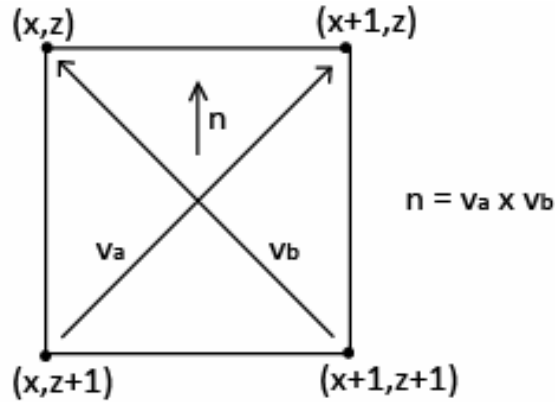


Figure 14: Calculation of tile normal

4.1.4. TANK MOVEMENT RELATIVE TO GROUND

When the tank moves, it adjusts its height according to the height of the plane at that particular location. The height between tile coordinates was calculated by interpolating the height of surrounding points. To find height at point (x,z), the equation of plane was calculated through tile coordinate (x_t,y_t,z_t) with normal (x_n, y_n, z_n) stored in the tile object.

$$n \bullet t = 0$$
$$x_n(x - x_t) + y_n(y - y_t) + z_n(z - z_t) = 0$$
$$height = y = \frac{x_n}{y_n}(x - x_t) + \frac{z_n}{y_n}(z - z_t) + y_t$$

When the tank moves, it adjusts its rotation so that it remains parallel to the plane at that particular location. The rotation was calculated by computing the angle and axis of rotation between the upwards vector (0,1,0) and the normal vector (x_n, y_n, z_n) stored in the tile object, and then applied to the model using glRotate().

$$\cos\theta = \frac{a \bullet b}{|a||b|}$$

Rotation about x axis: $$\theta = \frac{\cos^{-1}\left(\frac{y_n}{\sqrt{y_n^2 + z_n^2}}\right)}{2\pi \times 360}$$

rotation about z axis: $$\theta = \frac{\cos^{-1}\left(\frac{y_n}{\sqrt{x_n^2 + y_n^2}}\right)}{2\pi \times 360}$$

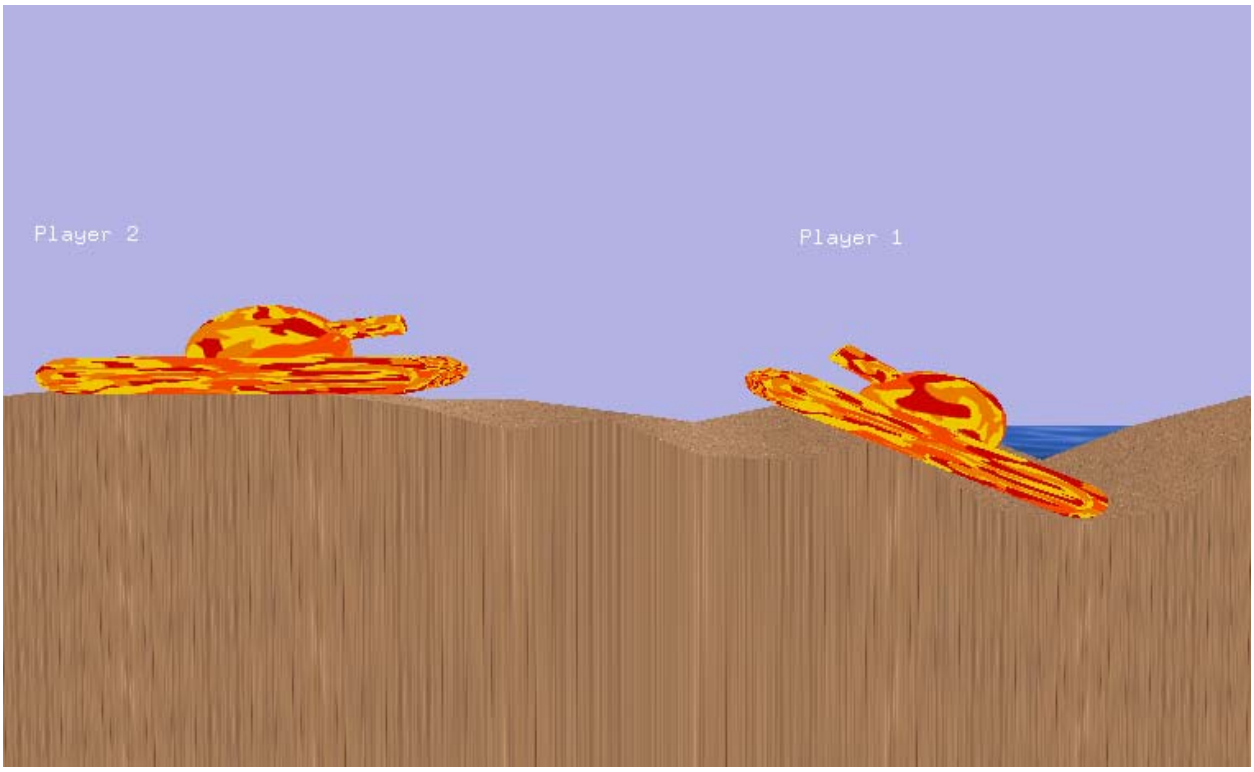


Figure 15: Tank movement relative to ground plane

Furthermore, the tank's movement is bound by the edges of the ground plane. On every move, the tank verifies that its next move is indeed ground (and not water) before it moves to that location.

4.1.5. EFFECTS OF TANK SHOOTING ON GROUND

On a tank's shot, the shot either explodes when it contacts the ground, or disappears if it enters the water. On explosion, the terrain is damaged and the height of the ground is decreased in the following manner to emulate realistic damaged terrain. Similar to a Gaussian function, the height decreases the most where the shot contacts the ground, and the height of surrounding areas are decreased a smaller amount.

		-.02		
	-.03	-.05	-.03	
-.02	-.05	-.10	-.05	-.02
	-.03	-.05	-.03	
		-.02		

Figure 16: Decrease in height of ground due to tank's shot

When the terrain is deformed in the ISO view, it affects the terrain when the view is switched into the flat view. Similarly, when the terrain is deformed in the flat view, the terrain in the ISO view is affected. The second situation is more complicated from the first – the terrain damaged is based loosely on the origin of the ground tile that is damaged. However, there are cases in which a shot made in the flat view will land between the two tanks. As explained in the Ground Compression Algorithm, the height of the terrain damaged was calculated as the average of two different heights. Since this location is imaginary in the real ISO view, a line is drawn between the two tanks, and tile damaged is the one that lies along that line, at the same x-coordinate, regardless of whether it is land or water. This is described in the image below.

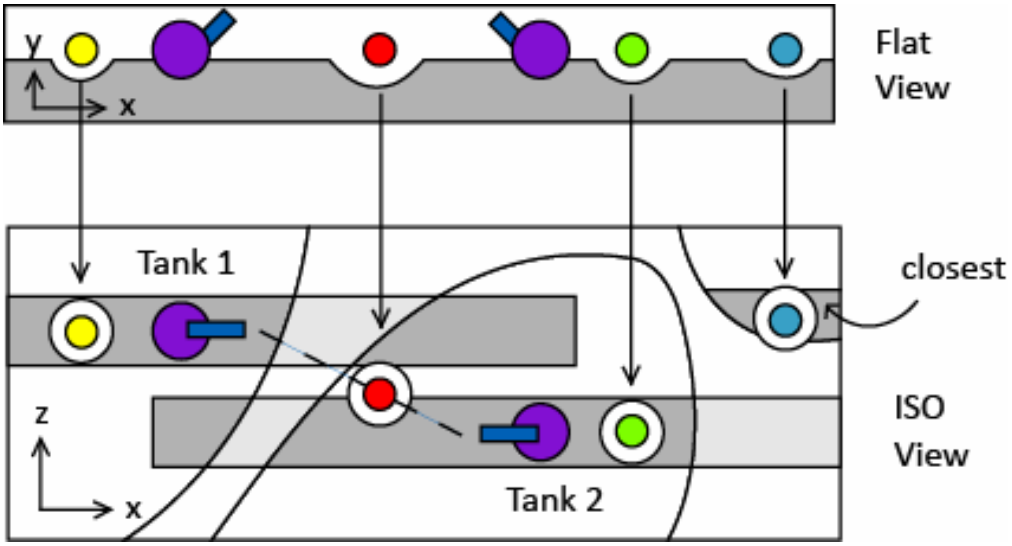


Figure 17: Terrain damage in flat view translated back to ISO view

4.2. COLLISION DETECTION

Initially, we planned on implementing collision detection by way of bounding volumes. This consisted of calculating the boundaries of all objects near a point of interest and determining if any object's boundary crosses each other. The calculation for 2D boundary collisions are easily modeled using bounding rectangle [1]. This method of boundary detection is dependent of the orientation of the object and we found that this method was too complex for 3D models of random orientation. For example, for every 3D object, there are five boundaries that needed to be checked for collision conditions. Instead, we decided to use spherical approximations of the volume to speedup and simplify our calculations. This meant that every model would have a set of spheres of varying radii that best estimates the volume. The collision models then becomes just simple calculations of distances between spheres regardless of object orientation. For example, if we have two objects (A, B) with dimensions of (Ax, Ay, Az, Ar) and (Bx, By, Bz, Br) respectively, then the equation for collision detection becomes:

Error! Objects cannot be created from editing field codes.

The accuracy of our collision detection is then dependent on only the number of spheres we choose to estimate our object and this number is varies from model to model depending on the shape of the object. The figure below illustrates how spheres are used to estimate the volume for one of our models. As you can see, we allow for

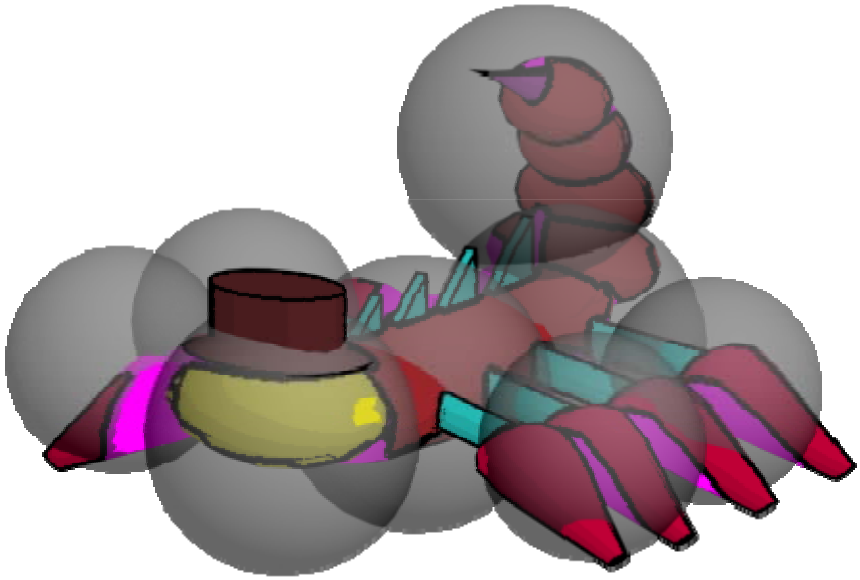


Figure 18: Spherical Estimation of 3D Models

4.3. MOVING CAMERA

Our camera model was implemented to allow for smooth camera panning in the 3D and 2D views as well as provides a smooth transition from 3D perspective to 2D perspective and vice versa. The mechanics of the moving camera model relies on a stack algorithm, which contains an array of camera views where the current camera view is at the bottom of the stack. The camera would smoothly animate to the next camera view above it and pop it from the stack when the transition is complete. This process occurs until all the camera view on the stack has been read and transitioned to. The stack follows a FIFO scheduling scheme which controls which camera view is read next. This camera model also allows for rapid viewpoint shifting for when you want to jump to a view without smooth animation.

For example, we implemented a camera view where we trace the path of the projectile using a combination of rapid camera movements to switch the view behind the projectile and then using smooth animation to follow the projectile path.

Our camera model allows for rotating about the terrain map, which also shifts our model orientation along with control orientation. The camera tracks of which axis it is viewing to allow for transforming our controls to always perform the correct operations regardless of camera orientation. For example, a user would press 'w' to move forward no matter where the camera is located. This is essential for our controlling scheme when in the 2D view because orientation of the axis matters in this mode.

4.4. GAME PHYSICS

Game Physics consists of projectile models using gravity for three distinctive ammunition types specifically bullet, laser, missiles.

4.4.1. PROJECTILE

The projectile class is used to draw the ammo as it is traveling through the playing field. To do so, it has to keep track of the ammo's location at each point in time that it is traveling. We model this by using equations of projectile motion. Since there are three different types of ammo, each type has a different set of equations to allow the ammo to have different traveling paths. All three types require the initial settings of position and velocity. For the ballistic ammo, we model this with the well known ballistic projectile equations with special modification to the time variable to suit our game purpose. As shown below, for each game time unit, it is multiply by 0.03, so that the projectile's position is incremented every 30 ms, same as the screen frame rate. The horizontal velocity remains constant in flight while the vertical velocity increases due to gravitational acceleration.

Time, t — Horizontal Direction

Displacement, d $$d_x(t) = v_x(t)(0.03) + d_x(0)$$ (1)

Velocity, v $$v_x(t) = v_x(0)$$ (2)

Acceleration, a $$a_x(t) = 0$$ (3)

Time, t — Vertical Direction

Displacement, d $$d_y(t) = \frac{1}{2}(-9.81)t^2 + v_y t + d_y(0)$$ (4)

Velocity, v $$v_y(t) = (-9.81)t + v_y(0)$$ (5)

Acceleration $$a_y(t) = -9.81$$ (6)

For the laser ammo, horizontal velocity and vertical velocities are constant in flight, and this creates an effect that resembles a laser pointer. For the last ammo type, the missile, it is different than the ballistic ammo in that the missile will have a horizontal and vertical acceleration in flight up to a certain point in time. This causes the horizontal and vertical velocities to increase. Once the acceleration stops, the missile ammo will travel in flight like a ballistic ammo. The equations below show the missile projectile calculations while missile acceleration is non-zero.

Time, t — Horizontal Direction

Displacement, d $$d_x(t) = \frac{1}{2}(missile\ accel.)t^2 + v_y t + d_y(0)$$ (7)

Velocity, v $$v_x(t) = (missile\ accel)t + v_x(0)$$ (8)

Acceleration, a $$a_x(t) = missile\ accel$$ (9)

Time, t — Vertical Direction

Displacement, d $$d_y(t) = \frac{1}{2}(-9.81 + missile\ accel)t^2 + v_y t + d_y(0)$$ (10)

Velocity, v $$v_y(t) = (-9.81 + missile\ accel)t + v_y(0)$$ (11)

Acceleration $$a_y(t) = -9.81 + missile\ accel$$ (12)

4.5. GLM AND MODEL

The model class is responsible for importing models and providing information about the models to support various game physics including player movement, firing an ammo, and collision detection.

The model class is used to import models created from 3dsmax. We use a glm class, which is included in the glut distribution by Nate Robins. This glm class can read in .obj files and generate a display list, which can be used by opengl to draw the model. Moreover, the glm class can import .ppm files, which can be used for texturing our models.

We have four models since we have four players. Each model has three parts: body, turret, and gun. It is necessary to separate each model into these three parts, because the turret can rotate about the body, and the gun can rotate about the turret. As such, the model class has to provide information about the relative position of the turret and gun to the body.

In addition, gun length information must also be provided, so that the ammo can be drawn properly to fire from the tip of the gun.

Moreover, for proper collision detection, the dimension of the models must be known so that we know when a player has been hit by a projectile or any other moving objects.

5. ASSET MODELS

Asset Models includes descriptions for various objects used in the game including vehicles and projectiles and the details our process of model creation from initial art designs to final modeling in 3DS Max.

5.1. BULLET TYPES

There are three different weapon types available for player usage, with a different model used for each of the four characters. The bullet type is the default type, and it flies at a moderate speed and is affected by gravity. The laser type moves at a significantly higher speed and inflicts higher damage and only moves in a straight line. Laser type weapons are the easiest to use but consumes the most amount of energy. The final weapon type is the missile, which moves up in an arc trajectory with initial acceleration and also inflicts the most damage. The models for all of the weapon types can be seen below:

EA78 Iverson Tank

Bullet Laser Missile

Figure 19: Ammunition for Tank

01NK Mobile Artillery

Bullet Laser Missile

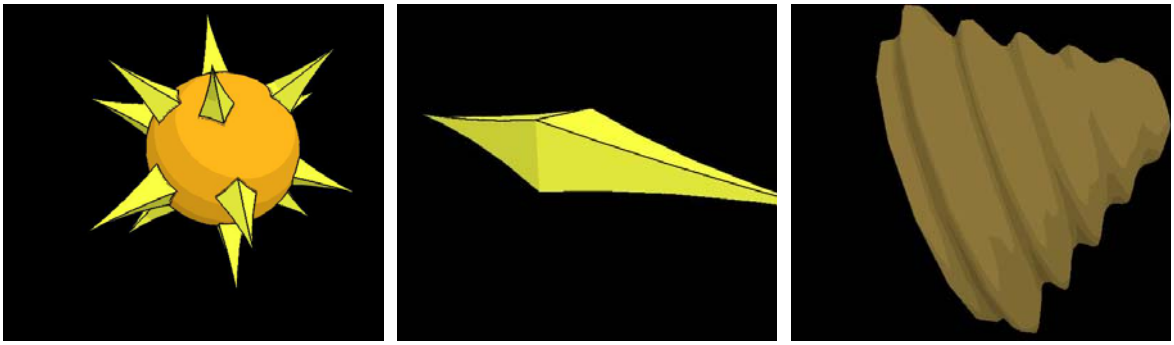


Figure 20: Ammunition for Artillery

HMS Mediterranean

Bullet Laser Missile

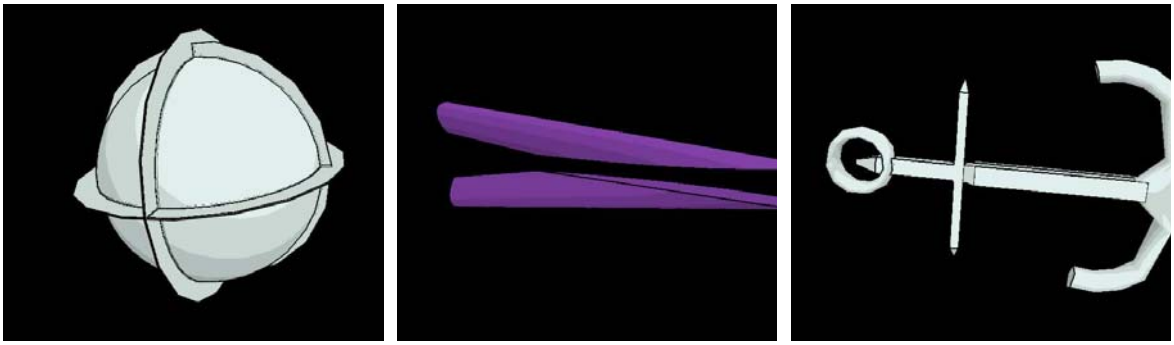


Figure 21: Ammunition for Battleship

“Old man” Alien Scorpion

Bullet Laser Missile

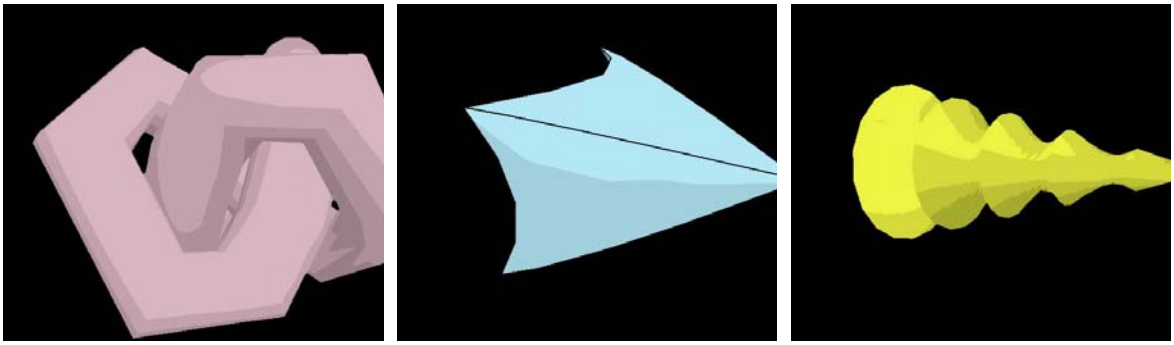


Figure 22: Ammunition for Scorpion

5.2. POWER UPS

There are two types of power-up crates located around the map that the user can use to gain an edge on the opponent. The figure below shows The Health-Up crate which adds 300 health points and the Ammo-Up create gives the user 25 additional rounds of ammo.

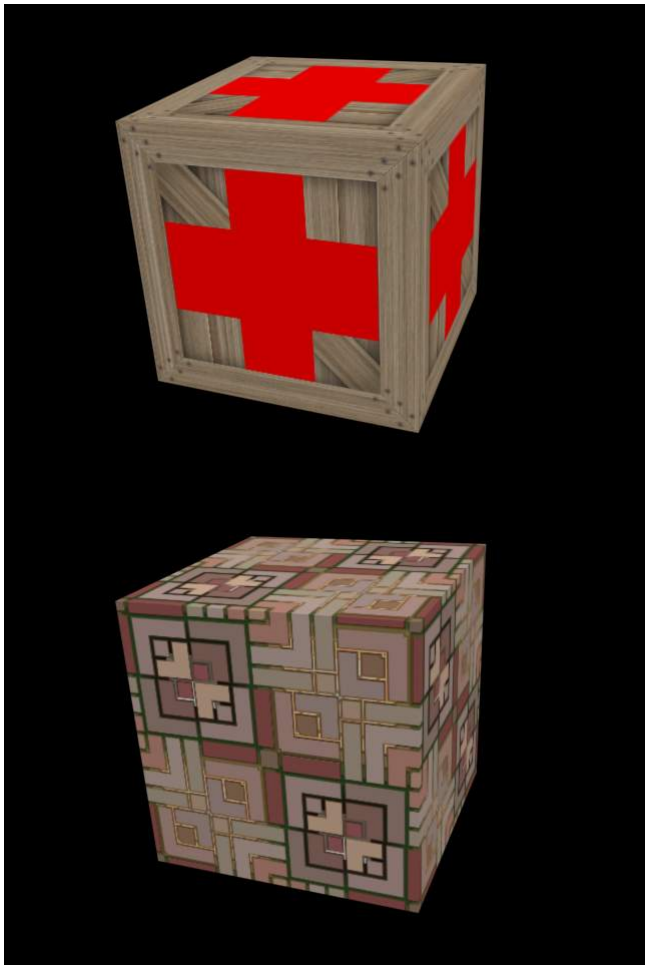


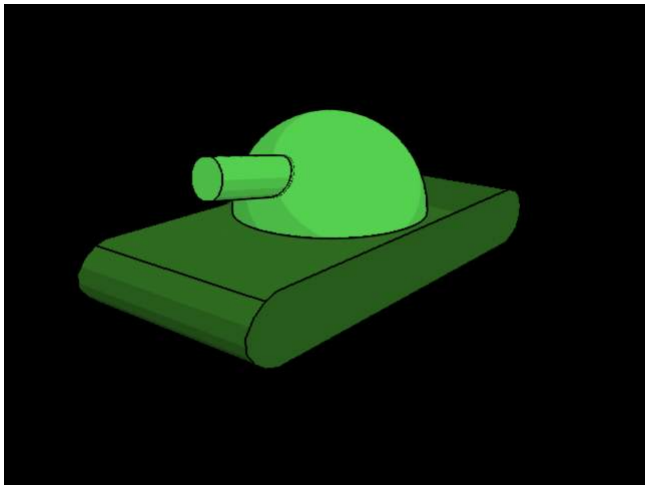
Figure 23: Health and Ammo-Up Crates

5.3. CHARACTER TYPES

Table 3: List of Playable Vehicle and Stats

Weight: 44.3 tons Length: 32.04 ft (9.77 m) Width: 12.3 ft (3.83m) Height: 8.5 ft (2.68m) Cannon: 120 mm M256 Smoothbore Cannon	**EA78 Iverson Tank**

HP: 1000



01NK Mobile Artillery

Weight: 72.4 tons

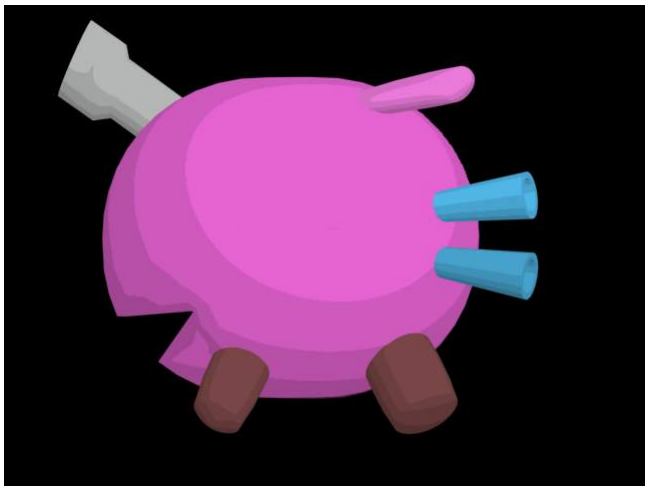
Length: 40.5 ft (12.3m)

Width: 50.4 ft (15.1m)

Height: 55.0 ft (14.3m)

Cannon: 155mm M198 Howitzer Cannon

HP: 600



Weight: 840 tons

Length: 110.6 ft (50.3m)

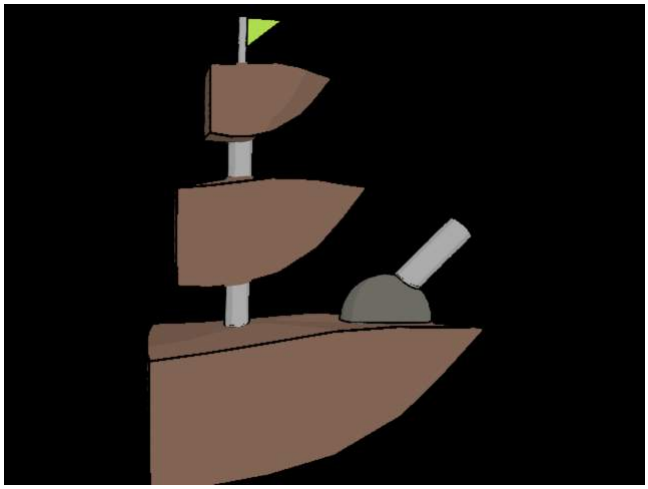
Width: 33.5 ft (10.1m)

Height: 430 ft (130.7m)

Cannon: 381mm BL15 Naval Gun

HP: 800

HMS Mediterranean



“Old Man” Alien Scorpion

Weight: unknown

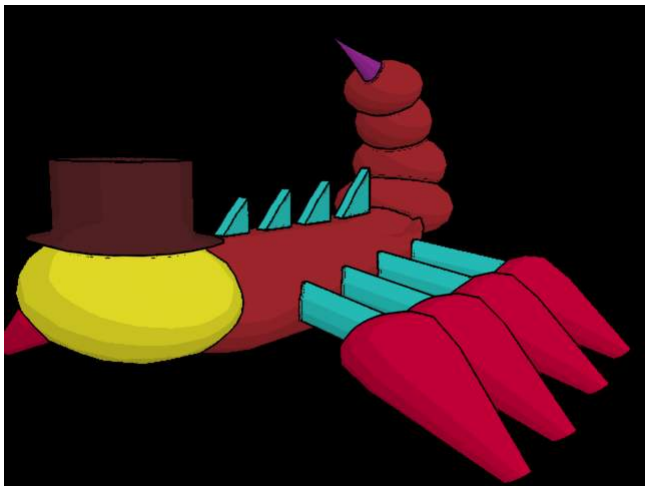
Length: unknown

Width: unknown

Height: unknown

Cannon: Tail Spike

HP: 800



5.4. MODELING

We started out with a series of hand-drawn concept art for the main characters in the game to provide a target during the modeling process, as seen in the next two figures.

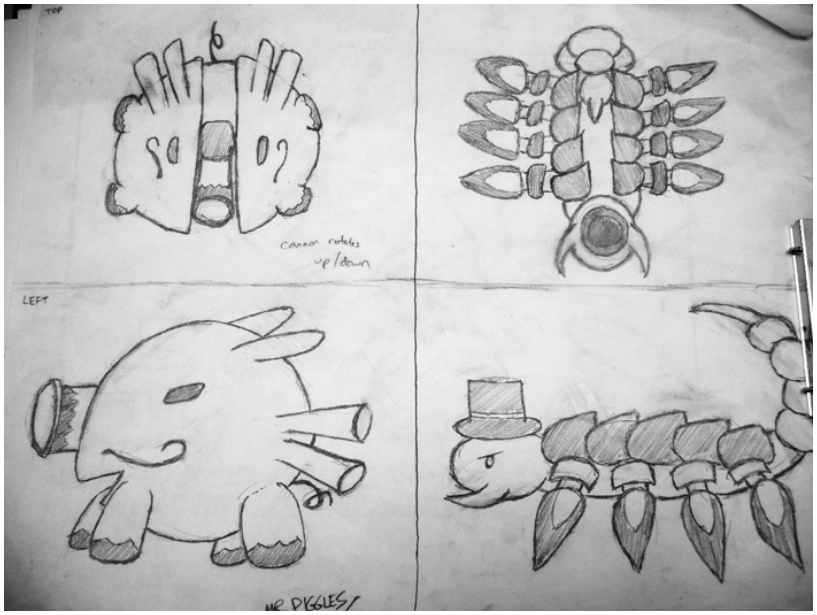


Figure 24: Concept Art

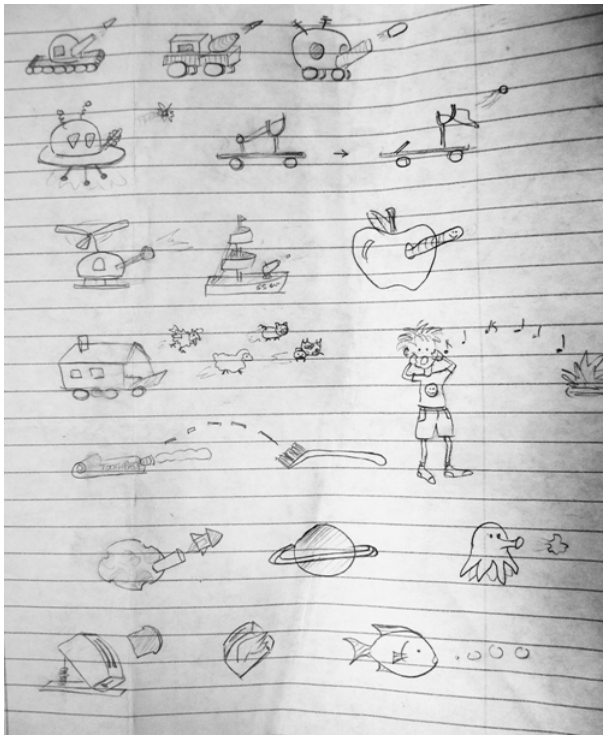


Figure 25: Draft Designs for Various Characters

Modeling was done using Autodesk 3D Studio Max 9. The general technique to create most of the more complicated shapes is to convert the object to an Editable Mesh, then select individual vertices and scale/move them around to the right location (as the figure below shows). This technique, used in combination with general modifiers such as Taper and Slice proved to be very effective in creating the desired shapes.

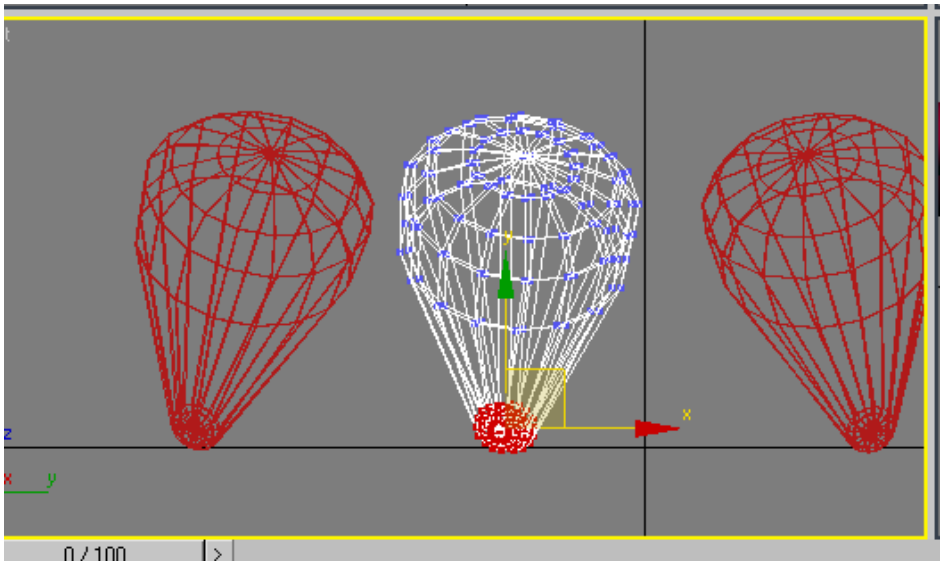


Figure 26: Selecting Individual Vertices of Mesh

Another good modeling technique involves turning the object into an Editable Poly and selecting individual faces of the object, then using the “extrude” and “taper” functions to create more faces that could be individually modeled (as the figure below shows). This process of building an object one “brick” at a time results in a single final object that can be more easily textures and will not have “seams” when differing pieces intersect with another.

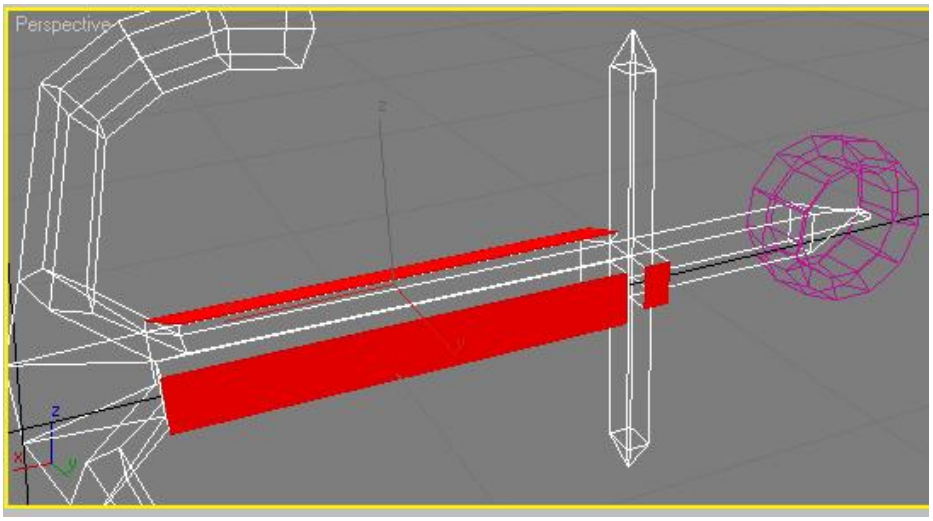


Figure 27: Selecting Individual Faces for Extrude

Each character model consists of 3 parts – the base, the turret, and the cannon. These are set up as separate objects so that the turret and the cannon can be individually rotated and the cannon can be moved up/down as required for game-play. The “pig” character does not have a separate turret --- it moves its entire body to aim its cannon. Once modeling is complete, the model is then resized and rotated to better fit the standard 1.0-based coordinate system in the OpenGL code.

5.5. TEXTURES

Texture files for each of the models were found on the Internet on public-use sites such as www.filterforge.com. A cell-shading technique was investigated (and used in the images of various models in this report) but found to be too processor-intensive for the computers in the labs.

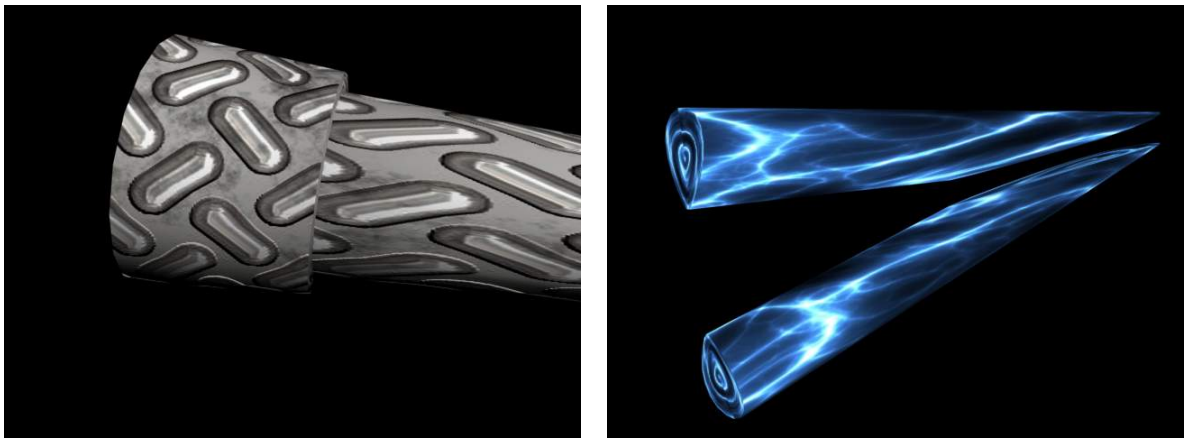


Figure 28: Sample Textures used for Ammunitions

5.6. SOUND

Sound is implemented using the SDL_MIXER library, which supports a large number of simultaneously playing audio channels, as well as one dedicated channel intended for background music. The sound effects are called through events in the game, and the music is enabled and disabled with a user button press. The SDL_MIXER library also provides functions to mix sounds and music together. The general technique to add sound to a game using the SDL_MIXER library can be summarized in the following 4 steps.

- Open up the audio device
- Load samples into memory
- Play them when necessary
- Clean up

The sounds in our game are downloaded from various free sound samples sites such as www.sound-effect.com and www.soundamerica.com. There are a number of sound effects in this game ---- a partial list can be found below.

- Main Menu Select
- Main Menu Back
- Character-related sounds when the user makes a character selection
- Individual firing sounds for each of the 12 ammo types

6. REFERENCES

[1] M. Barker, “3D Theory - Collision Detection,” 2008. [Online]. Available: http://www.euclideanspace.com/threed/animation/collisiondetect/index.htm [Accessed April 13, 2008].