DESIGN DOCUMENT

All work Copyright 2013 by DeadFish Productions

Michael Griscom, David Klimek, Frans Kurniawan, Shitianyu Pan, Josh Ventura

Version # 2.5

26 April 2013

ABSTRACT

This document outlines *Pivot,* a single player puzzle-adventure game that is a twist on the concepts popularized by Valve's *Portal*. In *Pivot*, players have the ability to create portals, which act as wormholes allowing the player – and other objects – to transport across the level. *Pivot* carries the physics-altering mechanics further by allowing players to manipulate gravity, adding to the depth of the puzzles. As the player progresses through the game, the level difficulty increases, slowly introducing new game-play elements and forcing the player to avoid spiders, turrets, and other obstacles. In addition to the built-in levels, players have the option of creating their own custom levels, which can then be shared with others. The diversity of in-game objects allows for many possible designs, enabling the creation of a wide variety of puzzles.

dead fish productions

TABLE OF CONTENTS

Abstract 2

Overview 5

  Specifications 5

  Length of Gameplay 5

  Victory Conditions 5

Playing the Game 5

  Installation 5

  Main Menu 6

  Core Game 7

    Overview 7

    Controls 8

    HUD 8

    Game Elements 10

  Level Editor 14

    Controls 14

    Using the Editor 14

Technical Details 16

  Game State Transitions 16

Code Architecture 17

Saving 18

Sound 19

Artificial Intelligence 19

Graphics 21

Physics 27

Bibliography 29

OVERVIEW

SPECIFICATIONS

Pivot was designed for the Windows operating system. The game was created using C# coupled with Microsoft's XNA 4.0 platform [1]. The development took place over a span of 15 weeks.

LENGTH OF GAMEPLAY

A new player can complete each of the built-in levels in approximately half an hour.

VICTORY CONDITIONS

The player must get to the end zone of a level in order to successfully complete it. Completion of all levels results in beating the game.

PLAYING THE GAME

INSTALLATION

The One-Click installer can be used to install *Pivot* on a Windows computer. This installer bundles with XNA 4.0 dependences, thus the computer must meet the requirements for XNA, i.e., be running Windows XP, Vista, 7, or 8, and have a graphics card that supports Shader Model 1.1 and DirectX 9.0C [1].

MAIN MENU

Upon starting the game, the player has several options, as illustrated in Figure 1; these options can be selected through keyboard or mouse input. They can play the core game, unlocking new levels as they progress, or create their own levels using the level editor. There are also audio and visual options that can be adjusted.

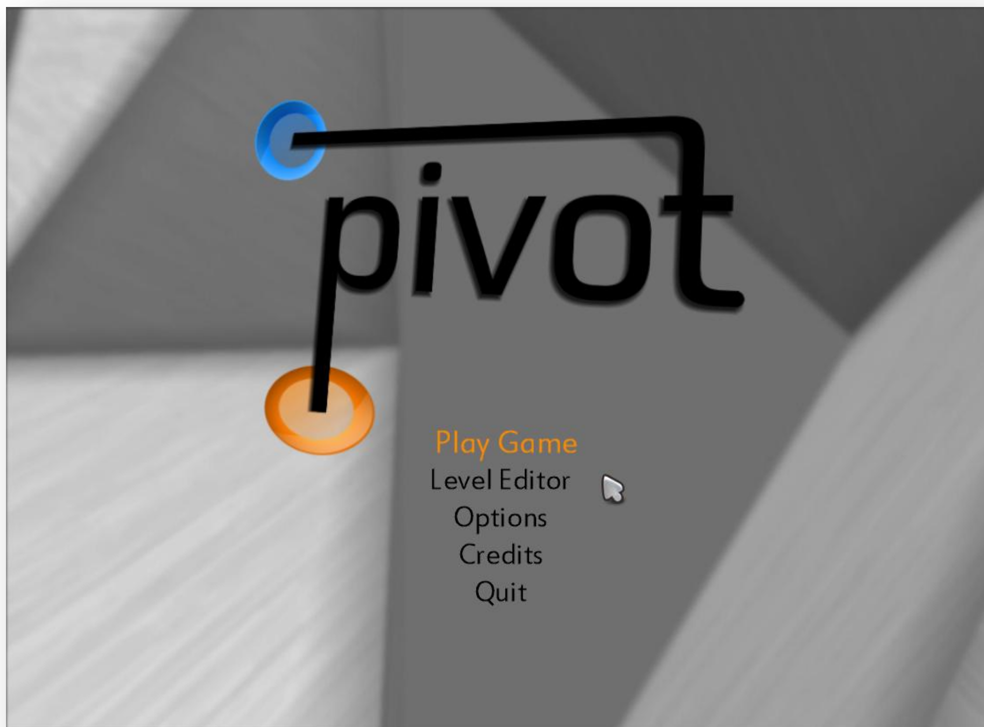

FIGURE 1: MAIN MENU SCREEN

CORE GAME

OVERVIEW

The goal of the game is to progress through every level. This is accomplished by reaching the "End Zone" (see Figure 2) through the use of tools at the player's disposal, while avoiding obstacles. The levels become increasingly difficult, slowly introducing new puzzle elements as the player becomes accustomed to the game mechanics.

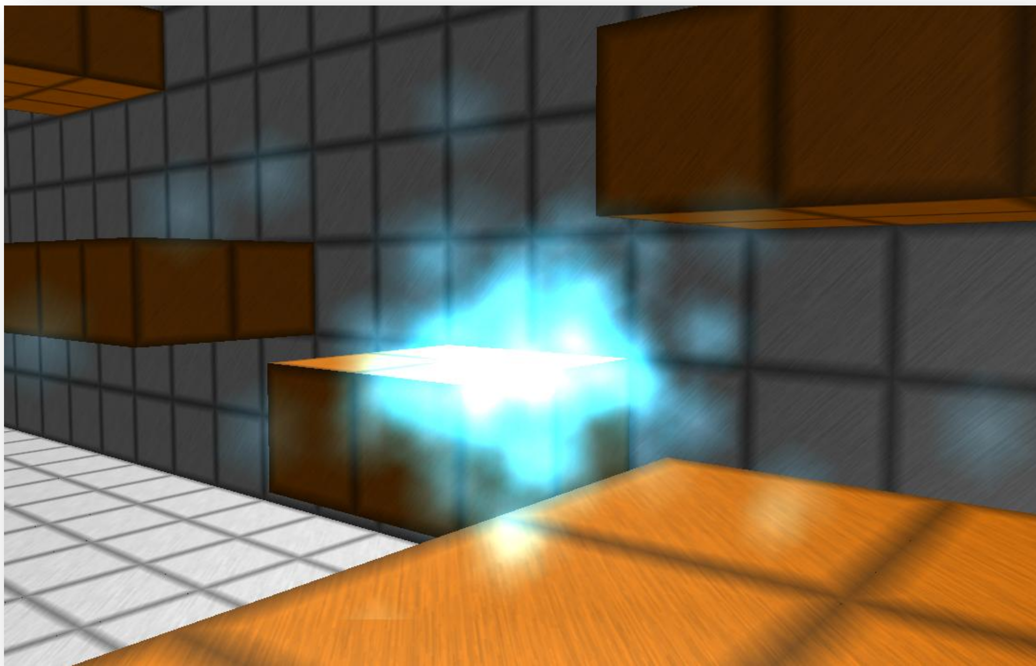

FIGURE 2: END OF LEVEL MARKER

CONTROLS

The controls for the core game are shown below in Table 1.

TABLE 1: MAIN GAME CONTROLS

Button	Function
W/A/S/D	Character movement
Space bar	Jump
Esc	Pause Menu
Mouse	Look
Left mouse button	Fire blue portal
Right mouse button	Fire orange portal

HUD

The HUD is depicted in Figure 3. A timer, visible at the top of the screen, adds an additional element of competition, allowing players to track their personal best times for each level. The crosshair serves multiple purposes, showing both where portal “bullets” will be fired in addition to whether a portal can be fired on a given surface, indicated by the crosshair’s color. Surfaces which can support portals are detailed in the Cube Types section of this document. The different crosshair colors are shown in Figure 4. Damage infliction is indicated on the HUD by a red overlay, shown in Figure 5. When the red overlay fades away, the player has returned to full health.

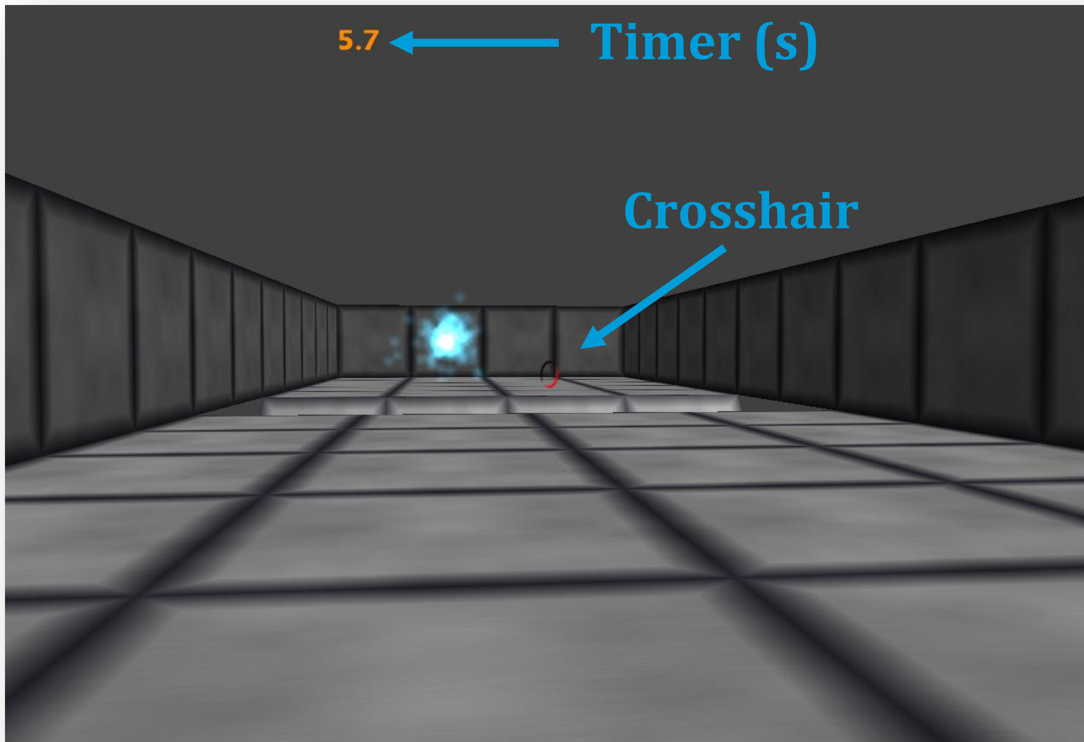

FIGURE 3: MAIN GAME HUD

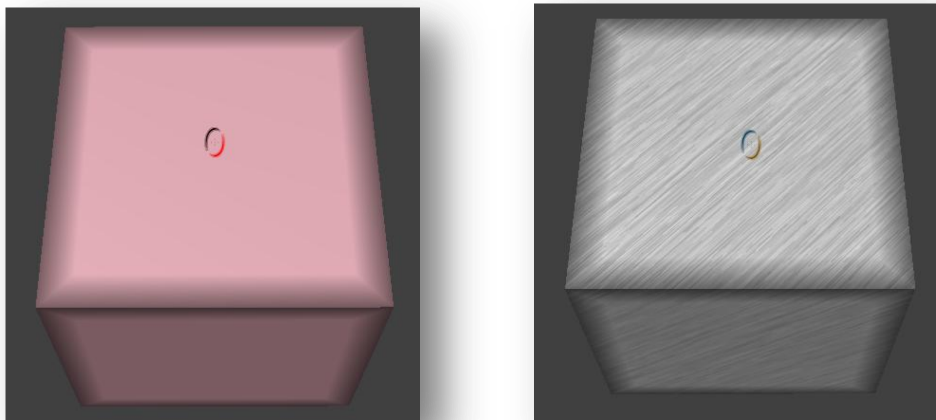

FIGURE 4: INVALID PORTAL SURFACE (LEFT) AND VALID SURFACE (RIGHT) CROSSHAIR INDICATORS

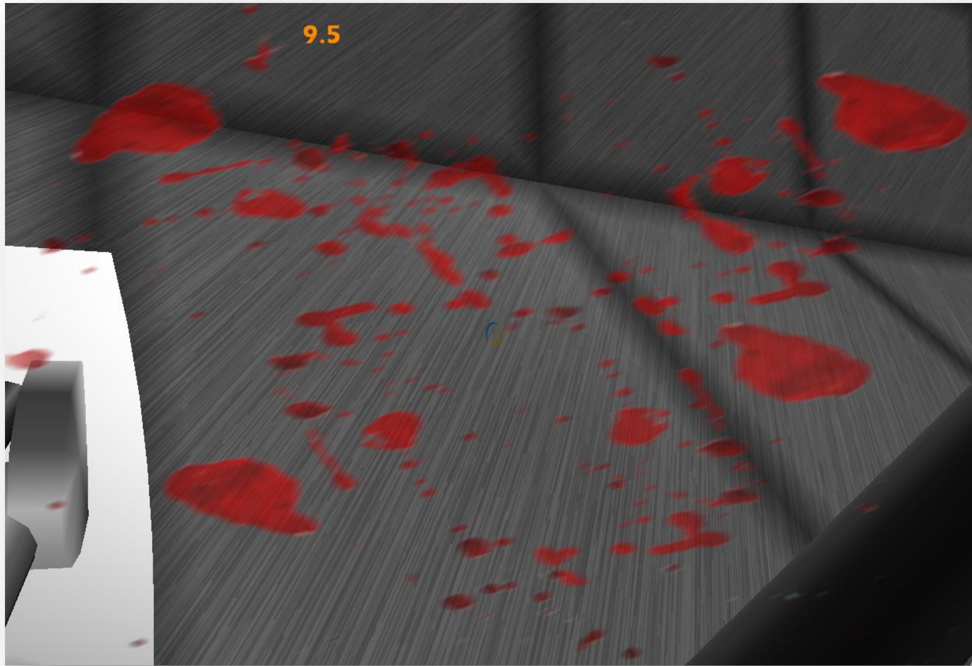

FIGURE 5: DAMAGE INFLICTION INDICATED BY A RED OVERLAY

GAME ELEMENTS

GRAVITY BUTTONS AND PORTALS

In order to beat certain levels, the player must make use of gravity buttons and portals. The gravity button is shown in Figure 6. In order to activate it, the player must jump on the button. The universal gravity will then be changed to any of the five other directions based on the given button. In other words, what was previously the ceiling or any of the four walls will now be the floor; an example of this is illustrated in Figure 7.

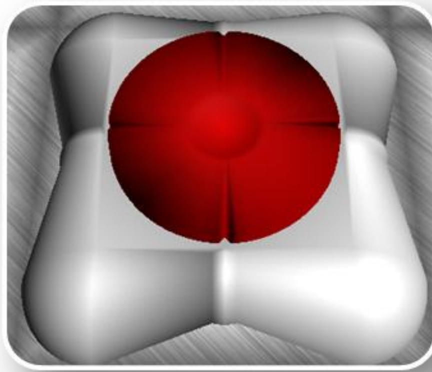

FIGURE 6: GRAVITY BUTTON

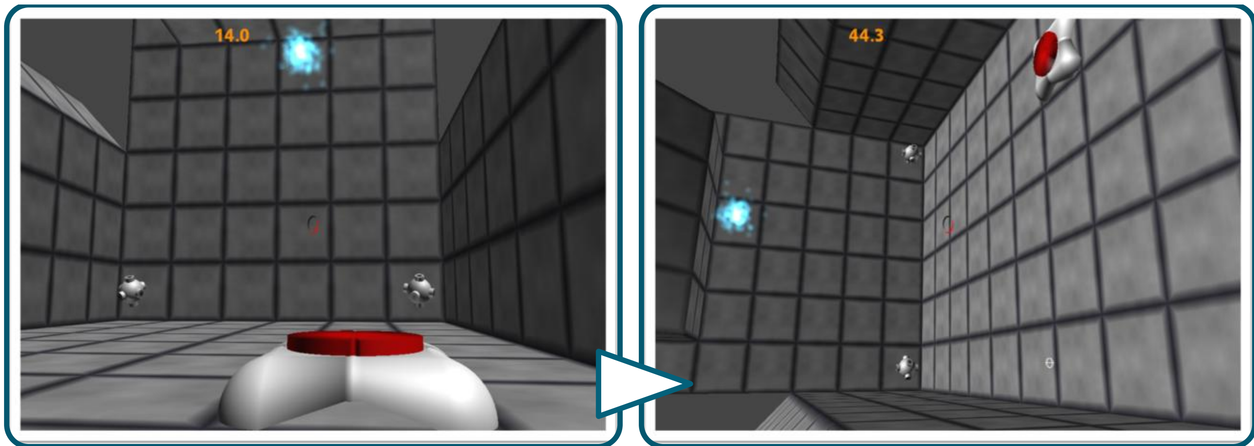

FIGURE 7: LEVEL REORIENTATION CAUSED BY GRAVITY BUTTON

There are two different types of portals: level portals which are permanent, and player portals which can be placed and removed in-game. These types are distinguishable by color, as shown in Figure 8. Static portal pairs share the same color, and the color of pairs varies. The player is able to shoot one pair of portals, that is, if a blue portal is fired, then firing a new blue portal will remove the previous one. Static portals have the added feature that they may alter the player's gravity. The

player's orientation after entering such a portal can be seen by looking through the portal. Other objects in the level, such as lava, are not affected by this change.

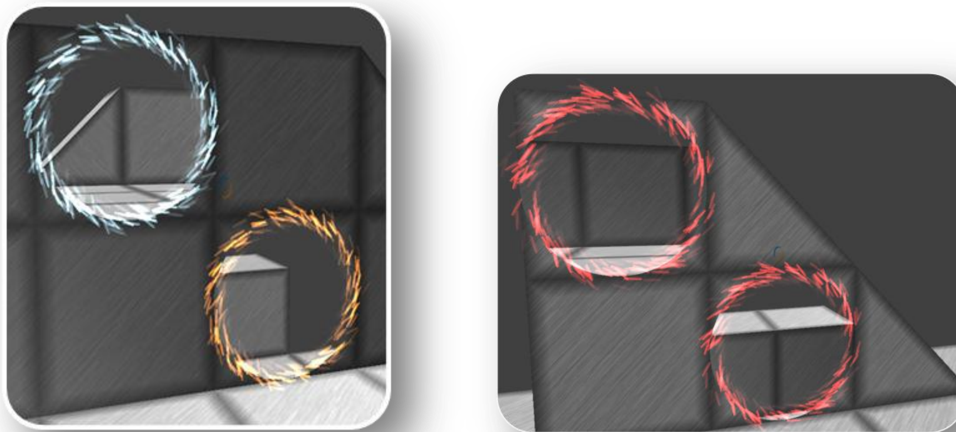

FIGURE 8: PLAYER PORTALS (LEFT) AND LEVEL PORTALS (RIGHT)

CUBE TYPES

There are five different types of surfaces: metal, rubber, concrete, brick, and wood. The differences between these are given in Figure 9. The last three types, concrete, brick, and wood, differ only in appearance.

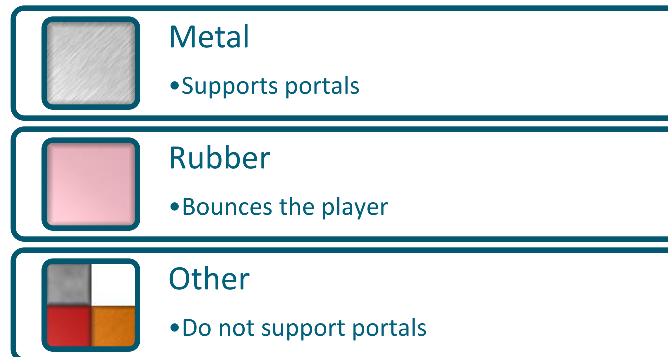

FIGURE 9: SURFACE TYPES

ENEMIES AND OBSTACLES

When navigating to the end zone, the route is often complicated by different obstacles which can damage and potentially kill the player. These are described in Figure 10.

FIGURE 10: ENEMIES AND OBSTACLES

LEVEL EDITOR

CONTROLS

The level editor controls are shown below in Table 2.

TABLE 2: LEVEL EDITOR CONTROLS

Button	Function
Mouse/Arrow keys	Translate cursor
+/-	Move cursor forward/backward
U/I/O/J/K/L	Alter cursor orientation
W/A/S/D	Translate Camera
Q/E	Zoom camera out/in
Ctrl+[add or subtract key]	Rotate camera
Ctrl+Arrow Keys	Rotate level
Space/Left mouse button	Place object
Right mouse button	Remove object
Ctrl+(Shift)+E	Alter object type
Ctrl+M	Alter object sub-type
Esc	Pause Menu

USING THE EDITOR

Players can create custom levels through the use of a built-in level editor. The user-friendly interface and file format allow for the quick creation and sharing of levels, as described in the Saving section of this document. The level editor also provides a drop-in feature, accessible through the pause menu (Figure 11), which allows immediate play testing during level creation.

FIGURE 11: LEVEL EDITOR PAUSE MENU

The level editor HUD is shown in Figure 12. The cursor, represented by a wireframe cube, indicates the location of the object to be placed. The arrow indicates the orientation of the object, which is used in setting the gravity direction of buttons and static portals.

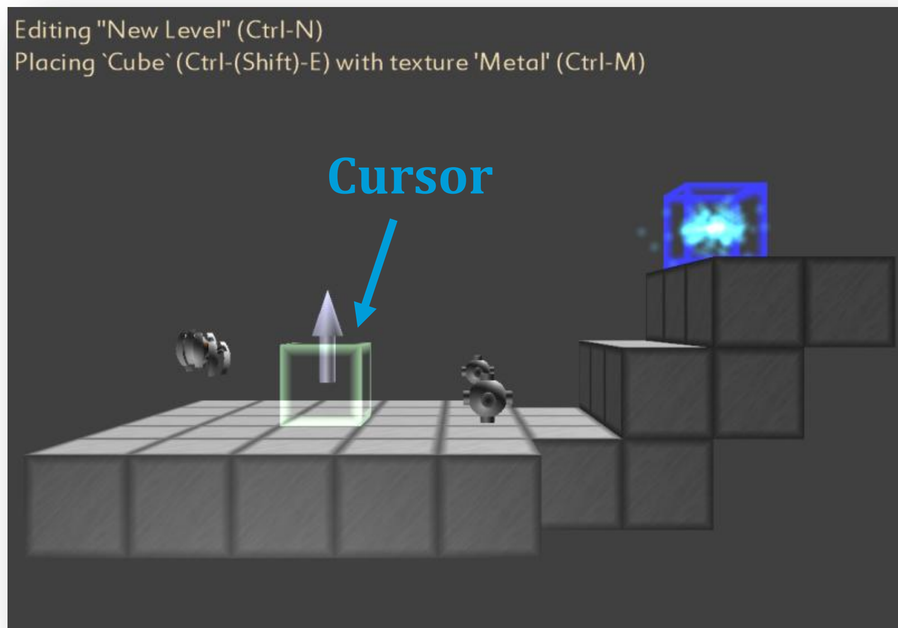

FIGURE 12: LEVEL EDITOR VIEW

TECHNICAL DETAILS

GAME STATE TRANSITIONS

Figure 13 illustrates the possible game state transitions.

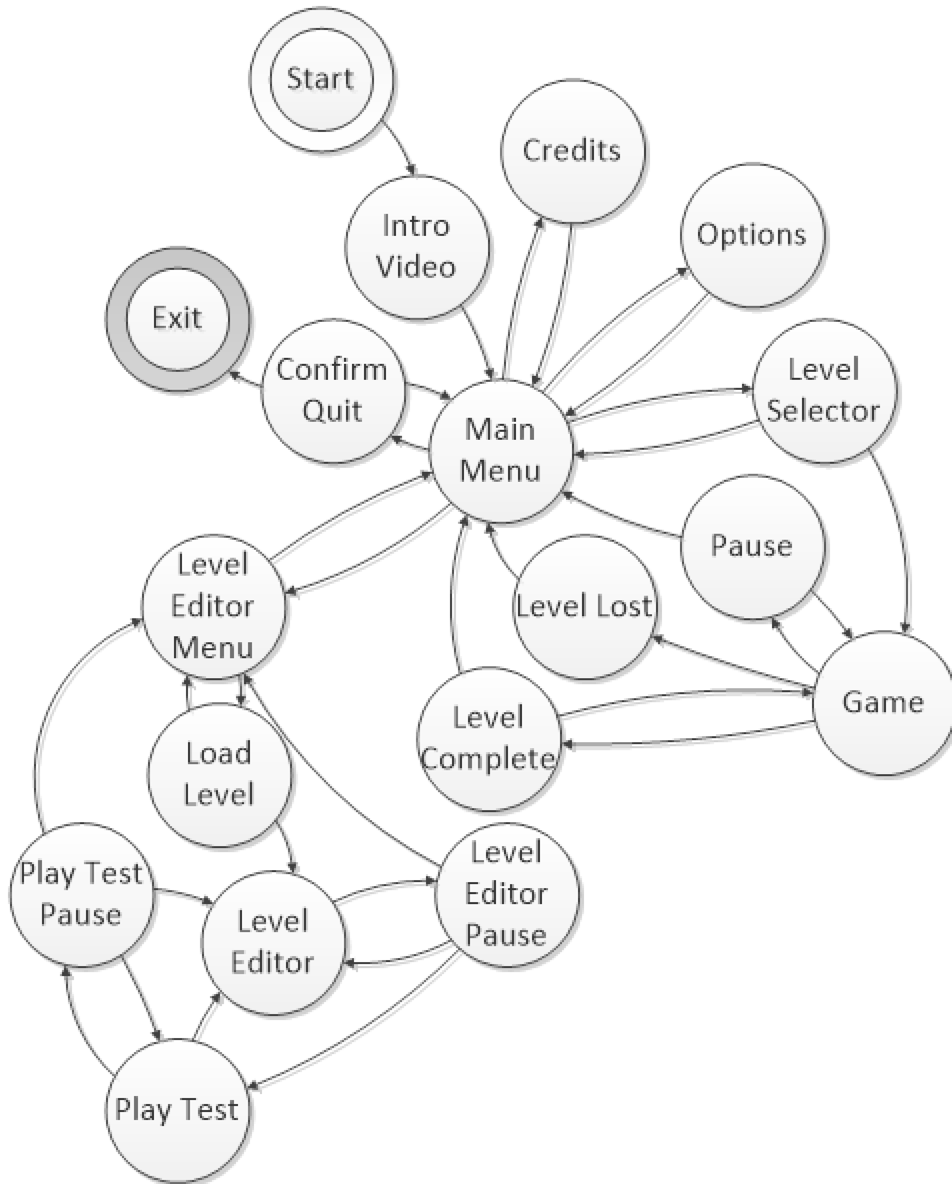

FIGURE 13: GAME STATE TRANSITION DIAGRAM

CODE ARCHITECTURE

The foundational organization of the codebase is designed using the Model-View-Controller (MVC) architecture, which provides a loose coupling between the game's business logic and its visual

manifestation. This design philosophy was leveraged across the codebase in order to create a flexible, maintainable application through the encapsulation of the various game components (e.g., the physics, graphics, and AI engines). Using the Visual Studio 2010 code metrics analysis, maintainability indices were calculated for each project of more than 500 lines. This index is a value between 0-100 designed to quantify the maintainability an application. Although this is a difficult characteristic to judge, particularly in an automated way, Microsoft considers values greater than 20 as “good” [2]. The results of this analysis are shown in Table 3.

TABLE 3: VISUAL STUDIO 2010 CODE METRICS RESULTS

Project	Maintainability Index	Executable Lines of Code
Framework	83	714
GameCore	91	1,335
GameState	84	1,658
Graphics	76	1,884
MapEditor	90	554
Physics	90	973
Weighted Average	**84.4**	

SAVING

The game progress (i.e., which levels have been beaten and the best time for each) is stored, along with the game settings, in the user’s AppData folder. Additionally, custom levels are serialized in an XML format within the same folder to allow for convenient saving, loading, and sharing.

```xml
<SpawnPoint>
  <X>0</X>
  <Y>0</Y>
  <Z>10</Z>
</SpawnPoint>
<EndZone>
  <Min>
    <X>-5</X>
    <Y>15</Y>
    <Z>-45</Z>
  </Min>
  <Max>
    <X>5</X>
    <Y>25</Y>
    <Z>-35</Z>
  </Max>
</EndZone>
<Tiles>
  <TileData>
    <Translation>
      <X>0</X>
      <Y>-10</Y>
```

```xml
<CompletedLevelData>
  <Name>level1</Name>
  <TimeToComplete />
  <TimeCompleted>2013-04-04T00:00:00</TimeCompleted>
</CompletedLevelData>
<CompletedLevelData>
  <Name>Platforms</Name>
  <TimeToComplete>00:01:37</TimeToComplete>
  <TimeCompleted>2013-04-04T00:00:00</TimeCompleted>
</CompletedLevelData>
<CompletedLevelData>
  <Name>Spiders</Name>
  <TimeToComplete>00:00:42</TimeToComplete>
  <TimeCompleted>2013-06-04T00:00:00</TimeCompleted>
</CompletedLevelData>
<CompletedLevelData>
  <Name>Gravity</Name>
  <TimeToComplete>00:02:47</TimeToComplete>
  <TimeCompleted>2013-06-04T00:00:00</TimeCompleted>
</CompletedLevelData>
<CompletedLevelData>
  <Name>StaticPortals</Name>
  <TimeToComplete>00:01:30</TimeToComplete>
  <TimeCompleted>2013-10-04T00:00:00</TimeCompleted>
</CompletedLevelData>
```

FIGURE 14: SAMPLE XML FOR CUSTOM LEVEL DATA (LEFT) AND GAME PROGRESS (RIGHT)

SOUND

The cross-platform Audio Creation Tool (XACT) library was used in order to play in-game music and sound effects [3]. The sound effects are generated in a virtual 3D space, meaning that sounds (e.g., that generated by the end-of-level marker) become louder or softer based on proximity, in addition to having differing left-right speaker strengths.

ARTIFICIAL INTELLIGENCE

The enemies of *Pivot* follow a state-machine model, where an entity's current state (e.g., "Idle") can transition to another state (e.g., "Chase") via a trigger (e.g., "Player Visible"). This allows for easy extension through the creation of additional, more complex states. Further, in the future, alternate AI personalities could be added via the use of different state machines, such as those emphasizing exploratory or aggressive behavior.

For the spiders, additional logic – in the form of path planning – is used to provide chasing mechanics. This is achieved through the creation of a navigation graph for the level, an example of which is shown in Figure 15. Note that in the image the wooden cubes are excluded from the graph, as spider locomotion is restricted to metallic surfaces. To determine paths, these graphs are searched via Dijkstra's algorithm, using the QuickGraph framework [4]. During locomotion, spiders are also encoded with flocking tendencies to create more realistic and varied behavior. These forces are detailed in Figure 16.

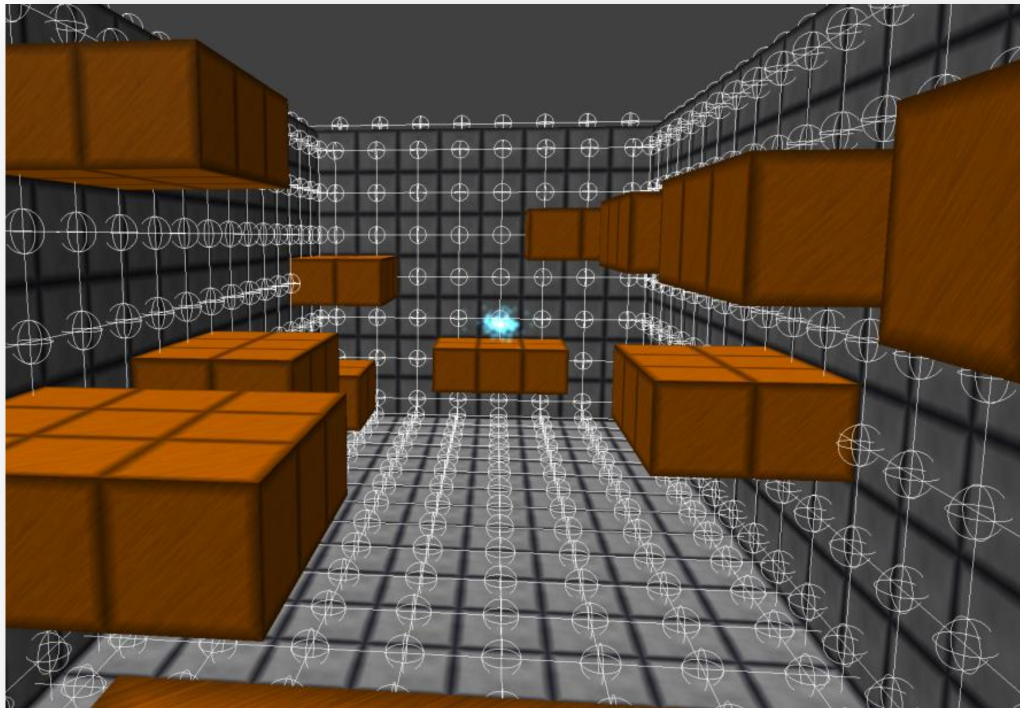

FIGURE 15: SPIDER NAVIGATION GRAPH FOR A LEVEL

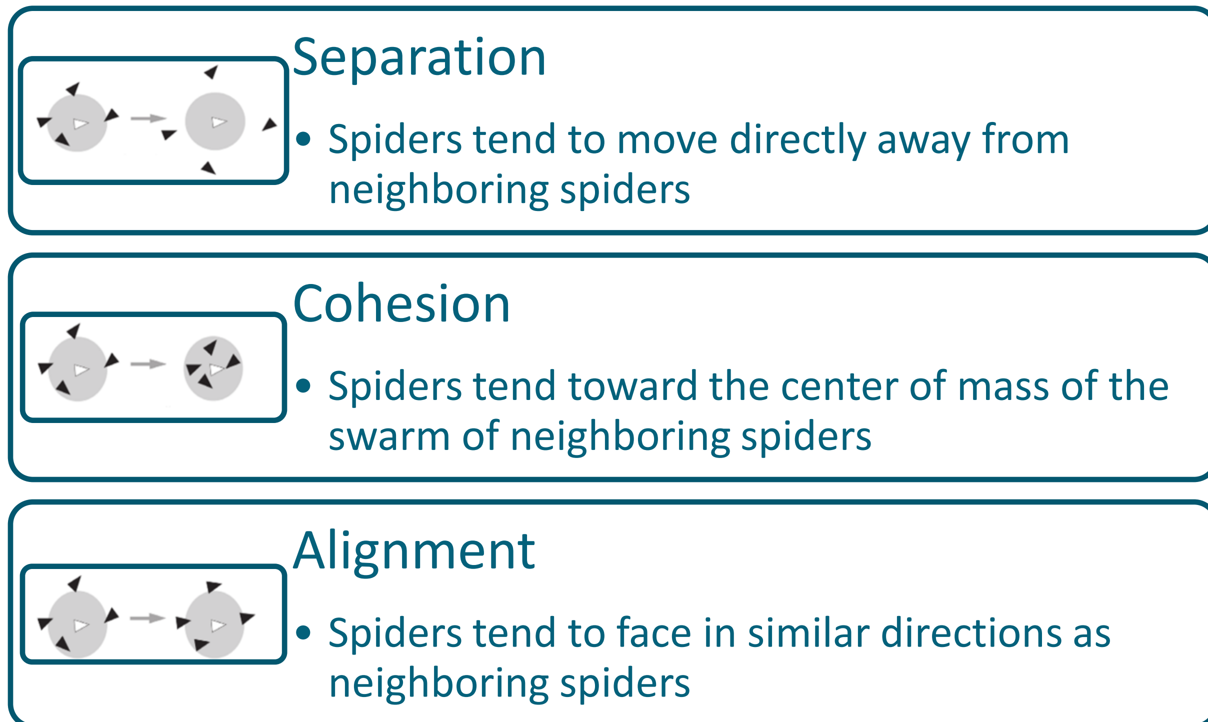

FIGURE 16: FORCES THAT CREATE FLOCKING BEHAVIOR IN THE SPIDERS (IMAGES FROM [5])

GRAPHICS

All of the in-game textures and models are created procedurally; that is, rather than utilizing modeling software, the functions that describe the appearance of objects are represented mathematically within the code. A simple example of this is the six-sided turret model. The model is created using a single sphere and three cylinders which are drawn along each axis from the outside of the sphere, through the center, to the other side of the sphere. The code for this is fairly simple:

```
var sphere = GraphicsUtil.SphereColored(radius, col);
verts.AddRange(sphere.verts);
inds.AddRange(sphere.inds);

GraphicsUtil.ModelColorNormal cylinder;
short co;
```

```
foreach (Vector3 v in new Vector3[] { new Vector3(maxRadius, 0, 0), new
        Vector3(0, maxRadius, 0), new Vector3(0, 0, maxRadius) })
{
    cylinder = GraphicsUtil.CylinderColored(barrelRadius, -v, v, col,
         Color.Black);
    co = (short)verts.Count;
    verts.AddRange(cylinder.verts);
    inds.AddRange(GraphicsUtil.Rebase(cylinder.inds, co));
}
```

The rotating turret is drawn in a similar manner. It comprises eight "pipe sections," that is, pieces of a cylindrical shell of a certain thickness between two angles, along with three colored cylinders. The primary code that generates it thus:

```
GraphicsUtil.ModelColorNormal[] models = new GraphicsUtil.ModelColorNormal[] {
        PipeSection(.6f, .8f, MathHelper.Pi*1f /16f, MathHelper.Pi*7f /16f, -length/2,
                length/2, new Color(200, 220, 255)),
        PipeSection(.6f, .8f, MathHelper.Pi*9f /16f, MathHelper.Pi*15f/16f, -length/2,
                length/2, new Color(200, 220, 255)),
        PipeSection(.6f, .8f, MathHelper.Pi*17f/16f, MathHelper.Pi*23f/16f, -length/2,
                length/2, new Color(200, 220, 255)),
        PipeSection(.6f, .8f, MathHelper.Pi*25f/16f, MathHelper.Pi*31f/16f, -length/2,
                length/2, new Color(200, 220, 255)),
        GraphicsUtil.CylinderColored(.7f, new Vector3(0, 0, length/2 + .1f), new
                Vector3(0, 0, length/2 + .3f), new Color(30, 35, 50), Color.White),
        GraphicsUtil.CylinderColored(.1f, new Vector3(0, 0, length/2 + .3f), new
                Vector3(0, 0, length/2 + 1f),  new Color(230, 235, 250), Color.Black),
        GraphicsUtil.CylinderColored(.2f, new Vector3(0, 0, length/2 + 1f),  new
                Vector3(0, 0, length/2 + 1.5f), new Color(30, 35, 50), Color.Black)
        };
```

That array is then placed into a single vertex buffer in a manner very similar to the one shown above for the six-sided turret. Clearly, most of the work (e.g., the trigonometry and index buffering) is done by the `PipeSection` and `CylinderColored` methods. Additionally, it can be seen that only four pipe sections appear in that array; the other four are packed into a separate vertex buffer to be drawn rotating the other direction. The rotation is simply done via matrix manipulation.

A less simplistic example of this is the spider model code and animation code. The entire trigonometry for this model is handled within the generator code. The spider does not use a texture; its eye is created by assigning random color values to each vertex, between yellow and red. The torso of the spider, the leg cylinders of the spider, and the foot curve for the spider are each stored in a separate vertex buffer. This allows for more easily animating the spider.

The animation is accomplished by creating eight points around the spider: one for each foot, as can be seen in Figure 17. The spider is given a sine wave to control its foot movement. When the wave is positive, the first set of four legs move to a new position nearer the spider's current position. When the wave is negative, the other four feet move to their corresponding position. If the spider moves during the elapsed time, the foot points do not move with it, which creates a more realistic dragging effect. To draw the legs, then, a triangle is calculated using the distance from the base of the spider to the foot position as the first side, and the length of each limb as the other two sides. Angles are then calculated, from which modelview matrices are calculated, and the legs are drawn according to those.

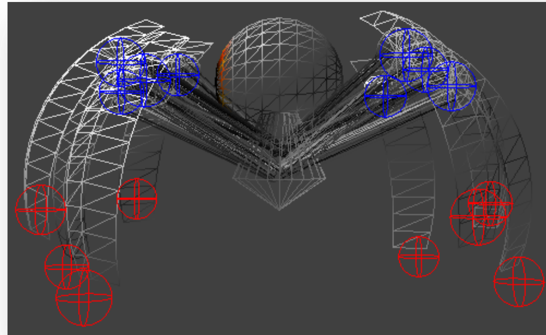

FIGURE 17: SPIDER JOINTS USED IN MOVEMENT

By extracting a function to generate the correct modelview matrices, creating an aesthetically pleasing explosion effect also becomes possible. When a spider is destroyed violently, the same matrices that were last used to render its limbs in the correct position are placed into a particle array. At the same time, an angular velocity matrix generated for each limb, along with a velocity vector. For each frame, the matrices are then transformed by the angular velocity matrix, as well as a new translation matrix created for their individual velocities. The result is shown in Figure 18.

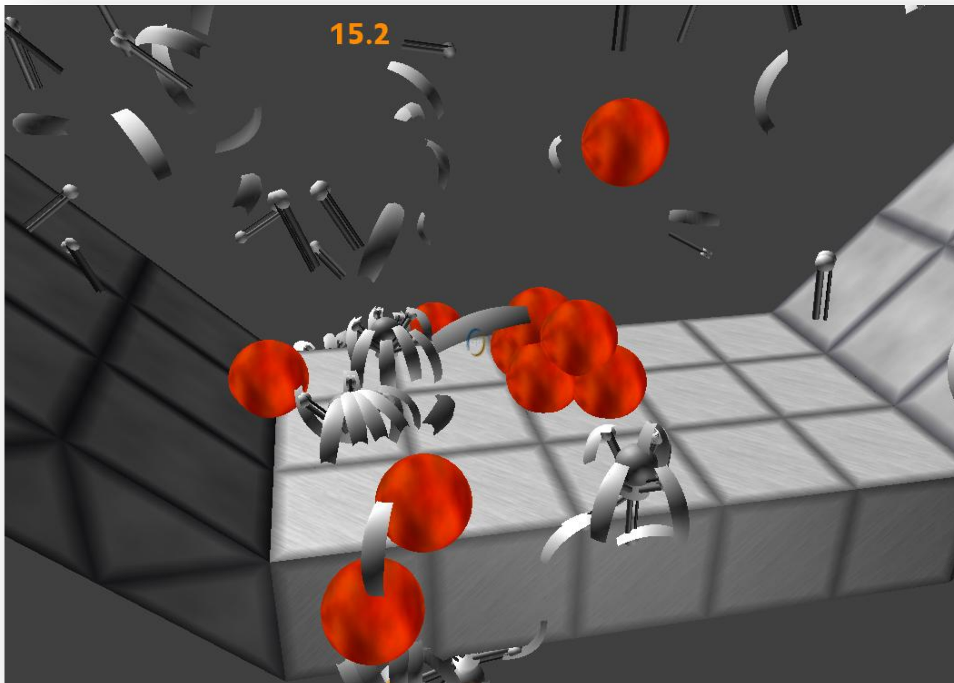

FIGURE 18: SPIDER EXPLOSION EFFECTS

Particle effects around portals are created using an array of roughly 300 particles, each having a unique position and velocity. Each particle is two triangles, graded from black to the color of the portal (mixed with white) and drawn additively. The end result of this is shown in Figure 19, an enlarged version of Figure 8.

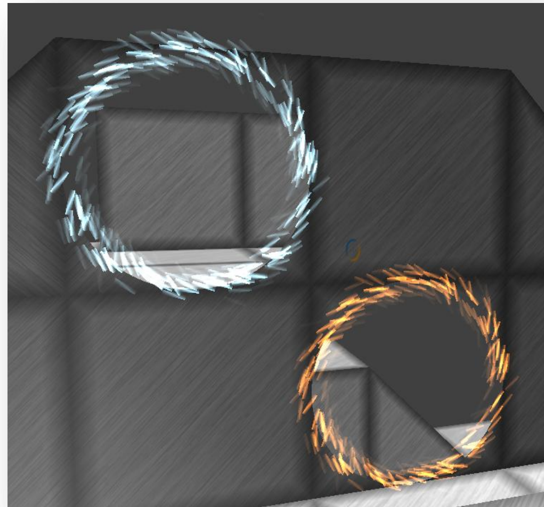

FIGURE 19: PORTAL EFFECTS

Portals are rendered by creating a "stencil" which informs the graphics device which pixels to render, then re-drawing the world in that stencil. The fill rate is optimized through the use of scissor rectangles, which enable the consideration of only those pixels which might lie in the portal stencil. These rectangles are illustrated in Figure 20. Portal nesting is accomplished recursively, and is optimized using back-face culling (indicating not to draw the scene), followed by frustum culling and occlusion testing.

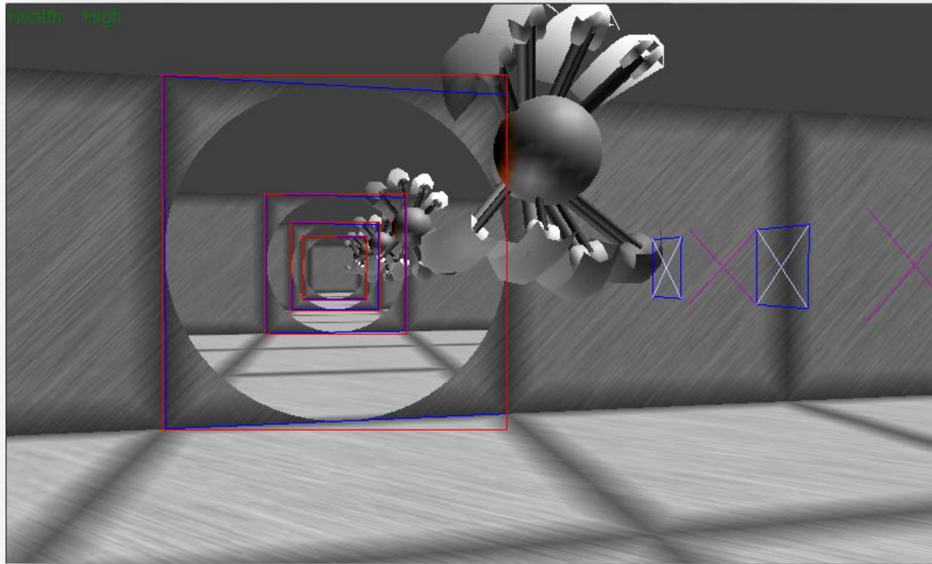

FIGURE 20: SCISSOR RECTANGLES INVOLVED IN NESTED PORTAL RENDERING

PHYSICS

The physics engine of *Pivot* is composed of the following objects: lava balls, spiders, cube tiles, ramp tiles, portal bullets, turret bullets, and a main player. The lava balls and main player are represented by spheres. Cube tiles and ramp tiles are represented with triangles. Portal bullets and turret bullets are represented with rays. To handle collisions, triangles are first extracted from every cube and ramp tile and placed into a binary spatial tree. This allows for quick access, $\log(n)$, to triangles near a given point or volume. The lava balls and main player bounce off the triangles upon collisions. A wireframe version of a level can be seen in Figure 21.

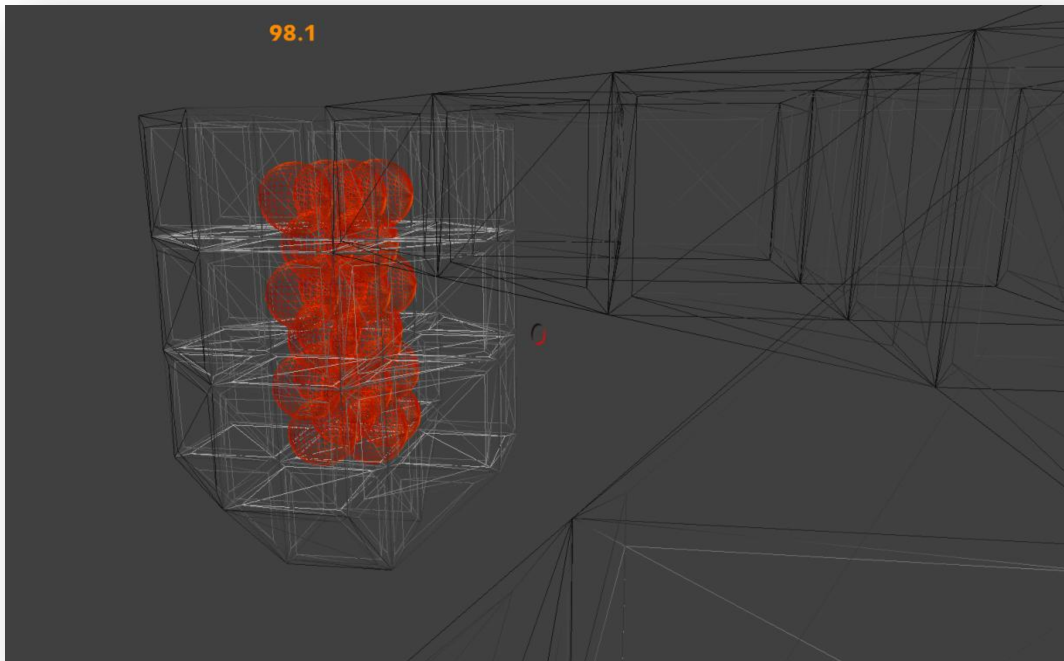

FIGURE 21: OBJECT WIREFRAMES

The triangles are also used by the spiders. When the game starts, the spiders are placed on the triangle that is directly beneath them. From that point on, they will always be positioned on a triangle. When a spider comes to an edge of a triangle it will detect if there is an adjacent triangle touching the edge of the triangle it is currently on. If there is, the spider will be moved to the adjacent triangle; otherwise, it will remain on the same triangle. Portals are accomplished by removing the triangles underneath them upon placement. If the portal is later removed, the triangles are replaced.

BIBLIOGRAPHY

[1] Microsoft, "Microsoft XNA Game Studio 4.0," [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=23714. [Accessed 25 April 2013].

[2] Microsoft, "Code Metrics Values," [Online]. Available: http://msdn.microsoft.com/en-us/library/bb385914.aspx. [Accessed 25 April 2013].

[3] Microsoft, "Using Microsoft Cross-Platform Audio Creation Tool (XACT)," [Online]. Available: http://msdn.microsoft.com/en-us/library/ff827590.aspx. [Accessed 25 April 2013].

[4] QuickGraph, "QuickGraph, Graph Data Structures And Algorithms for .NET," CodePlex, [Online]. Available: http://quickgraph.codeplex.com/. [Accessed 25 April 2013].

[5] M. Buckland, Programming Game AI by Example, Sudbury, MA: Wordware Publishing, 2005.