**Recovery**

**A FPS/Survival Horror Game**

**Made in Unity 3D with Maya and Blender**

**Roles**

**Bryan Spahr**: team lead, character voices, music, coding, texturing, modeling, level design, lighting, all level compilation

**Shanon Mathai**: coding, scripting, modeling, animations, bots, AI

**Adrian Campos**: coding, story, main level mesh, documents

**Babak Shahian**: object meshes

**Jessica Candebat**: object meshes, wall panels

**Kylie Adkins**: sketches, title screen

Original Concept?:

We wanted a wide array of enemy bots that would fight in different ways, such as shooting, punching, or picking you up and throwing you. We also planned to have a boss fight with a massive bot at the end of the game that would signify the end of the game.

End result:

We ended up with a game this is more of a FPS. We were originally going for a game where you should choose combat as the last resort. We settled for a more simplistic AI algorithm that made it too difficult to implement effective sneak around tactics where the penalties would be more harsh should the sneaking around fail. Now the player has to engage in combat at some points in the game and kill the enemies. We were hoping to use the NavMesh and NavMesh Agent that is provided by Unity to have a smarter AI, but NavMesh generation is a Unity Pro-only feature that we were not willing to invest in.

Running the game:

Download the application for your OS from the webpage: <https://sites.google.com/site/angryrobotstudio/home/download>

Or git pull this repo for the complete Unity project:

<https://github.com/pizzaboy314/recovery>

You can run the source in Unity by opening the Unity folder as a Unity project and loading the start.unity scene in the Assets folder. Pressing the Play button at the top of the Unity application to start the game.

Design details:

The map contains a number of rooms so that the player could search around the level and explore different areas. Theses rooms contain various models so the player can keep track of where they are in the level (should they go around in circles). There are a few signs on the walls to direct the player, given that they pay attention and get close enough to them. We utilized the Unity fog render (which really just obscures object a distance with a selected color based on distance. ***TODO***

The AI relies on predefined paths. The paths are defined by a consecutive series of empty game objects, whose transform will represents the set of point in the order of how the tracker will try to follow the points. The WaypointCircuit script is assigned to a parent GameObject (in Unity) to define an independent circuit of points for a path, where the children transforms defines the points to a single, looping track for the bots to follow, you would organize the children in the Inspector under the WaypointCircuit script until the order seems fit (Gizmo drawing in the script helps visualize the ordering of the track). This means, if the bot never finds the player within its lifetime, it will remain in the circuit for as long as the game is being played. The bot targets the next point if it hasn’t locked on to the player once it has come within a define distance to the current tracking point in the circuit. The bot locks onto the player based on the raw distance between the player and the itself. At the point that it locks onto the target and makes it best attempt to move directly at the target and follows the target until the target leaves the max follow distance (1.5 \* the attach distance). The AI is susceptible to getting stuck since it is trying to go straight at the player regardless of obstacles in between. The robots are able to attach to the closest WayointCircuit by finding the closest point that in each circuit and find which circuit contains the closest point. The robots attach to their closest circuit on start and when the detach from their target.

The scripts that work on the AI are thrown into a namespace called RobotAI so that it can refer to the CircuitRefs script that is used to find the closest circuit and the vice versa, which serves essentially as a singleton using only static fields and methods. Each WaypointCircuit will add itself to the ArrayList upon the call to Awake. This way each robot can ask the “map” (more specifically the CircuitRefs class) to find the closest WaypointCircuit using its own transform using the function getClosestCourse.

**Asset Sources**

spaceship sound

http://www.freesfx.co.uk/sfx/spaceship

various SFX

http://www.freesound.org/

black/metal textures

http://www.psdgraphics.com/textures/metal-textures/

tileable photoshop metal holes pattern

http://s0nkite.deviantart.com/art/SK-Metal-Grids-HD-Patterns-188624389

vector metal pattern

http://eps-ai.blogspot.com/2012/03/vector-metal-texture.html

all other textures

http://cgtextures.com/

gui health bar code

http://answers.unity3d.com/questions/17255/how-do-i-make-a-health-bar.html