reb00t

Patrick Soueida *26767923*

Jordan Orsini *26471196*

Matthew Verrucci *26795528*

Alec Nepute *27285086*

Majed Alkhouri *26089046*

Project download link:

https://drive.google.com/file/d/0B-YiYGcy6iuXUEdkVVdPQUNvRFE/view?usp=sharing

Project walkthrough link:

https://www.youtube.com/watch?v=awULl7uzEGQ&feature=youtu.be

Table of Contents

Executive Summary

You are a computer virus that has infected a host computer. The goal is to reach a portal that leads to the computer's mainframe. It isn't that easy, however, as the anti-virus guards have been alerted of your infiltration and are looking for you.

Luckily, due to your elite hacking skills, you can bypass the guards with in two different ways. You can either camouflage into the surrounding environment to disguise yourself as one of the three coloured registers (Green, Blue, or Red), or you can use your Stun Gun to slow down the anti-virus guards. The anti-virus guards are guarding a total of 4 randomly generated switches that need to be activated in order to open the mainframe portal.

By using fast thinking and paying attention to the surroundings, you must activate all four switches and then reach the final portal without any of the anti-virus guards catching you. Or else, you start back from the beginning. Good luck, hacker!

Overview

Genre: Action-Stealth/Puzzle

Platform: Windows

Setting: Inside a computer's security fortress represented digitally (with inspiration from TRON)

Plays like: TRON meets Metal Gear Solid

Summary:

You play as a hacker, represented by a humanoid avatar, with the goal of accessing the main database of a computer. To reach your goal, you must bypass the anti-virus guards to deactivate the database's firewalls. You can bypass in two ways: either by camouflaging into the environment, or by using your stun gun. Once you deactivate all the firewalls, you can access the main database, thereby winning the game.

Mechanics:

- The stealth mechanic works by making the levels have color-coded sections, with the enemies patrolling their environment. The player can blend into the color-coded sections to disguise himself from the enemies.
- Each enemy can't see the player if the player blends in their respective color-coded section (e.g: player enters blue zone, enemy spots him, player must blend into blue zone before enemy catches him).
- With this, the player must think in terms of tactics as each time the player enters a zone, his color-coded ability to camouflage from other enemies depletes (e.g: when player enters blue-zone, his blue-camouflage has 5 seconds before it completely depletes, which forces the player to think tactically to avoid enemies and save his ability).

- The stun gun uses the same energy source as the camouflage and depletes it entirely, so the player must strategically decide which to use and when.

Related Games:

Portal, Valve Corporation.

Portal is a puzzle-platformer video game developed by Valve Corporation. Since Portal takes place in a testing environment, its levels purposefully have a "planned-out" feel to them. Much like our environment will be, the level design in Portal is not meant to feel organic.

Metal Gear Solid Series, Kojima Productions.

The Metal Gear Solid series is a famous and beloved action-stealth game franchise, focusing more on the stealth aspect than the guns-blazing action approach that many other games do. Our game will take this approach, making action and shooting secondary features.

Tron Franchise, Walt Disney.

The Tron franchise consists of multiple movies, games, a TV series, and more. The Tron environment takes place in a digital world with very computer-like, neon infrastructure. This vibrant environment in Tron is a huge aesthetic inspiration for our game.

Far Cry 3: Blood Dragon, Ubisoft Montreal

Released as a standalone DLC for the original Far Cry 3, Blood Dragon is a parody of 80s sci-fi action movies. Much like our game, the environment consists of bright, neon colors and polished textures.

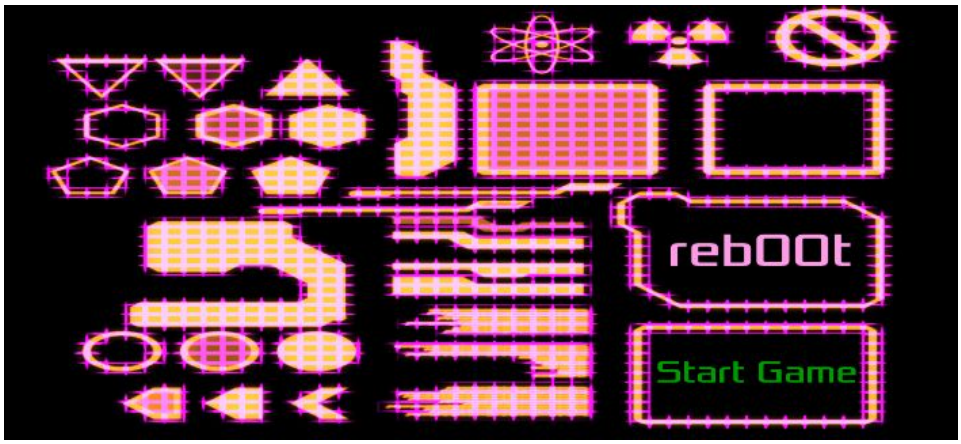
Game Characters

Player: The humanoid avatar for the player, a white-hat hacker by day and black-hat hacker by night. The player has the intention of infecting any computer they can and hacking into its mainframe. The avatar wears an archetypal set of space-marine armor and uses a stun gun to avoid raising any alarms or damaging the database he wants to hack.

Anti-virus guards: Following the retro-futuristic style of Blood Dragon as mentioned earlier, these are cyberpunk-style avatars for the anti-virus programs. They have funky hairstyles, retro sunglasses, and poker facial expressions. This shouldn't fool anyone, however, as they are specifically programmed to efficiently eliminate any threats from the system. They have simple AI, guarding critical sections of the system from any programs or threads that seem out-of-place. If they detect such an anomaly, they will lock onto and remove it upon contact.

User Interface Storyboards

Start menu:

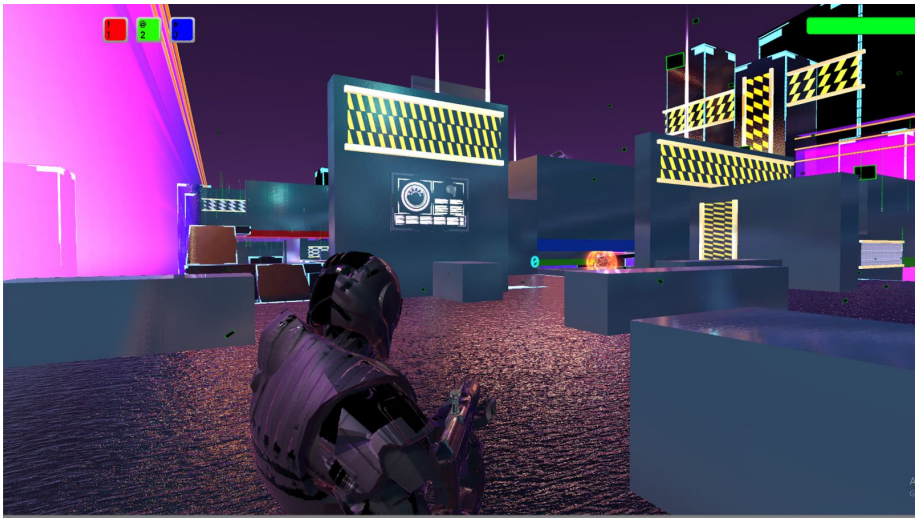


Relatively simple Start menu. Hovering over Start Game changes the color from green to red. Clicking "Start Game" brings up a short message prompt informing the player of the story's overview and the objective.

Screenshot of gameplay. Player in neutral (not camouflaged) state in the middle

- Top-right: player's energy. Green portion represents remaining energy (used portion is red)

- Top-left: buttons to press for camouflage and their respective colors (1 is to camouflage in red, 2 is to camouflage in green, and 3 is to camouflage in blue).



Screenshot of the player aiming before shooting:

- Added target crosshair in the middle (the blue zero), to perform greater target acquisition.

Screenshot of the pause menu (when you press ESC button):

- Resume: resumes the game state
- Quit: go back to main menu

Technology Plan

Unity assets:

- Sci-Fi Arsenal, https://www.assetstore.unity3d.com/en/#!/content/60519
- Animated Sci-Fi soldier, https://www.assetstore.unity3d.com/en/#!/content/80326
- Digital Environment Effects, https://www.assetstore.unity3d.com/en/#!/content/83441
- Neon Glow Displays Vol_1, https://www.assetstore.unity3d.com/en/#!/content/57561
- Neon Glow Displays Vol_2, https://www.assetstore.unity3d.com/en/#!/content/62168
- Unity Standard Assets https://www.assetstore.unity3d.com/en/#!/content/32351

Project discussion and general communication:

- Slack

Code management:

- GitHub
- GitHub integration for Slack

Documentation and presentations:

- Google Docs
- Google Slides

Minimum Hardware Requirements:

- OS: Windows Vista 32-bit / MacOSX
- Processor: Dual-core 2.0 GHz
- Memory: 2 GB RAM
- Graphics: DirectX 11 class GPU with 512MB VRAM (nVidia GeForce 8600 series, AMD Radeon HD 3600 series, Intel HD 4000 series)
- DirectX: Version 11.0c
- Storage: 1.0 GB available space
- Sound Card: DirectX9.0c or greater Compatible Sound Card

Software Architecture

Our game contains 4 main modules that interact with each other in order to give the player a fun and challenging experience. Each of these modules were done and tested alone and were then integrated together and tested to insure proper flow to the game.

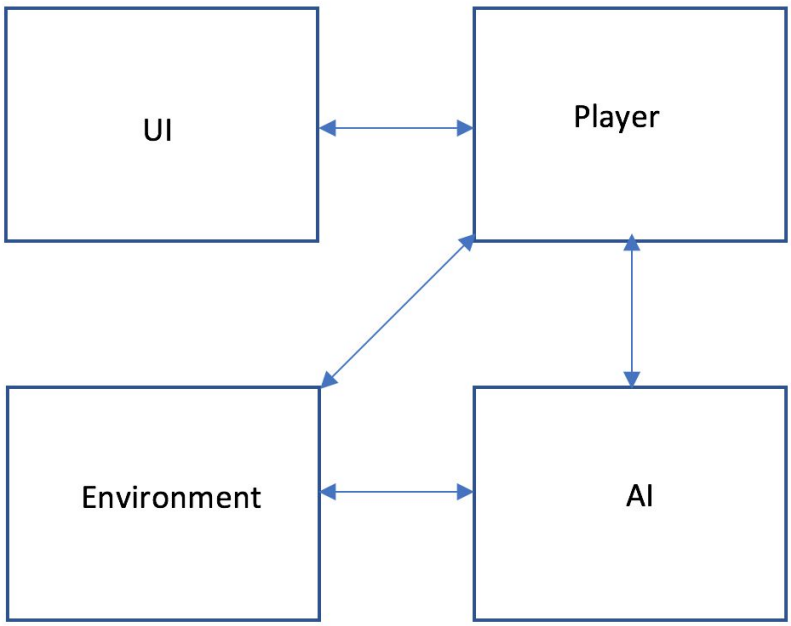
The first module is the player and player controller. It focuses on the physics behind the movement of the player as well as jumping and crouching. It also enables the different animations and different sounds needed for different actions (i.e. walking, running, crouching, shooting, interacting with a terminal). This module is also responsible for changing the color of the player character when pressing different camouflage buttons (see Controls) and spawns the gun bolt when firing, along with all the necessary calculations in order to have the player shoot forward. Respawning the player once they collide with an enemy is done in the respawn script.

Next is the level design/environment. This module is responsible for all the visual aspects and flow of the game as well as the interaction between the player and the environment. As mentioned previously, the aim of the game is to activate 4 switches that are randomly selected every time the game is played. Once these switches are activated, a portal opens and the player must escape. The random selection of switches is done in the switch script. This insures that the switches are randomly chosen every time. The level was designed to give the player freedom in choosing how to get to each objective. It was also fitted with different assets to bring life to the world.

Another module is the enemy AI. It focuses on AI pathfinding, animations, states and seeking the player. The AI is an integral to this game and needed different scripts in order to work as we planned. AI movement was all done within the AI basic script. This script was responsible for making sure the AI

moved correctly and that the speed of the AI was acceptable. The AI camera script and AI sight scripts focus on the different behaviours of the AI and also detect when the player enters their field of view. With the help of NavMesh, the AI are able to navigate through the level with ease and seek out the player if the player has been detected.

Finally, the last module is UI which is responsible for the in-game HUD, the main menu, the cutscene, the intro and outro text and the pause menu. Our aim was to have the UI be simple and efficient so the player doesn't get confused with what each component means. With the integration of the energy bar and the buttons used to camouflage, this gives the player information on how to play the game. The HUD is done in the player controller and the AI billboards are done in the AI camera script. The cutscene used different camera vantage points in order to tell the player what needed to be done. It checks to see which switches are enabled and shows them accordingly.



Controls

Keyboard & Mouse control mapping:

- **W:** Forward movement
- **A:** Strafe-Left
- **S:** Backward movement
- **D:** Strafe-Right

- **Space:** Jump/Skip cutscene
- **E:** Activate nearby switch

- **1:** Activate red camo
- **2:** Activate green camo
- **3:** Activate blue camo
- **Tab:** Deactivate camo

- **Left control:** Toggle crouch
- **Left shift[HOLD] + W:** Sprint
- **Escape:** Toggle pause menu

- **Mouse:** Camera movement
- **Left mouse button:** Shoot
- **Right mouse button[HOLD]:** Zoom in + enables crosshair

Algorithm methods for key input:

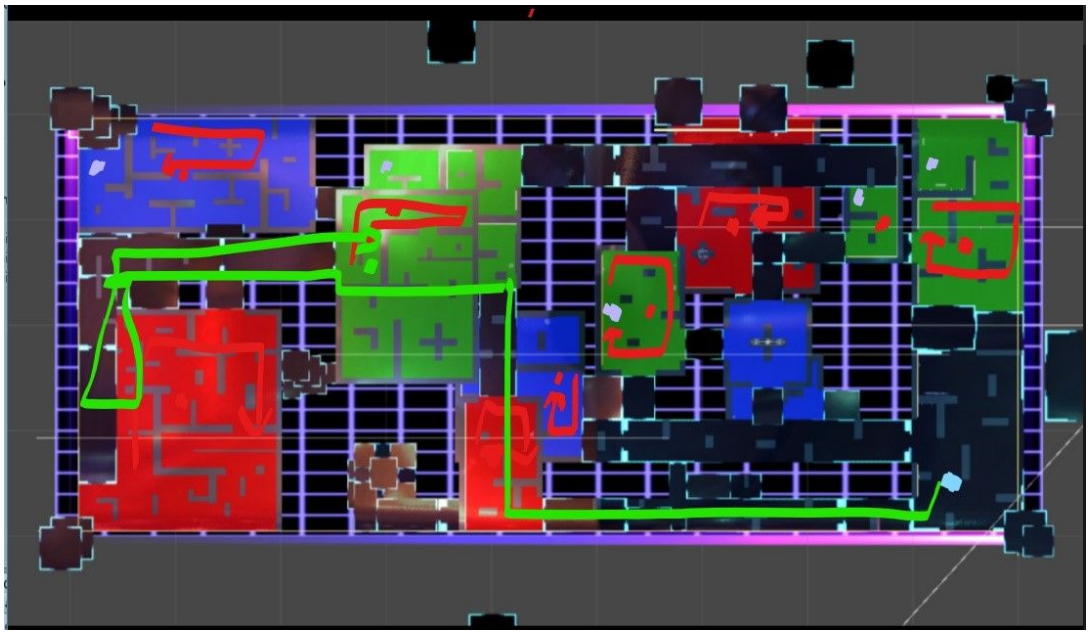
Our algorithms for input works by calling a decision tree (in the void Update() function) to check if an input keycode is detected. If yes, it either directly maps the input, or activates a boolean that is a gateway for the input.

Pseudo-code examples for better illustration:

```
// Example 1
if (Input.GetKeyDown(KeyCode.W)){
      //do GetComponent<Rigidbody>().AddForce() }

// Example 2
if (Input.GetMouseButton(1)){ isAiming = true}
if (isAiming = true){// do aiming }
```

Level Design



- Turquoise dot: player start location
- Red dots: enemy locations
- Red arrows: patrol arrows
- Green arrow: player flow from start of the game to the end of the game (after flipping all the switches on).

The level was designed to give the player the freedom to choose how to play as well as where to go next. It was designed to give the player a sense of open world, where they can explore the different areas of the map while getting to their different objectives. Ramps were used to facilitate access between different rooms of the map for both the player and the enemies. Different assets and particle effects were used in order to give the level some life (i.e. digit storm, pixel storm, etc…). We also wanted to give the level some verticality so we decided to have rooms that overlap vertically. Doing so made the level more interesting to explore and navigate. Most rooms have many hiding spots and tight spaces, although some

rooms are open and have a minimal amount of cover. This was done to force the player to think tactically and manage their energy bar better. The switches were placed in the bigger rooms. This was done so we could have more AI patrolling the area near switches making it more challenging for the player and therefore, making it more fun. As for the decoration and look of the level, many assets were used to decorate it and make it look visually appealing.

Mechanics Analysis

Mechanics:

- Stealth: Focusing on stealth makes the game more accessible than a standard action game. Also it is an effective yet simple way to exhibit the AI for the enemy NPCs.

- Camouflage: After hitting many roadblocks for interesting ideas, our team decided we needed some kind of unique mechanic or gimmick. Eventually a chameleon ability was brought up, but would require an environment with primary colors (since realistic textures would be difficult to implement and add to a complex 3D model). This, on top of the creative freedom, gave us additional incentive to use a digital environment.

- Stun gun: Our focus for the game has always been more towards stealth than action, so we originally had little intention to include any weapons, especially not lethal ones. Throughout the course of development, we eventually decided on a stun gun to fit the stealth theme. Towards the end, we also decided that the stun gun would use the same energy source as the camouflage to offer balance and variety to the gameplay.

Balance:

- We decided to use the same energy supply for the camo and stun gun for simplicity as well as design. We wanted the player to have the option to stun a guard while making sure the camouflage was the still the primary focus.

- Related to the previous point, there needed to be further incentive to use camouflage rather than the stun gun. This led to our decision to make the stun gun fully deplete the player's energy

- Coloured sections of the level are small, and encounters with the guards are relatively short, so the camo was adjusted to last 5 seconds instead of 15.

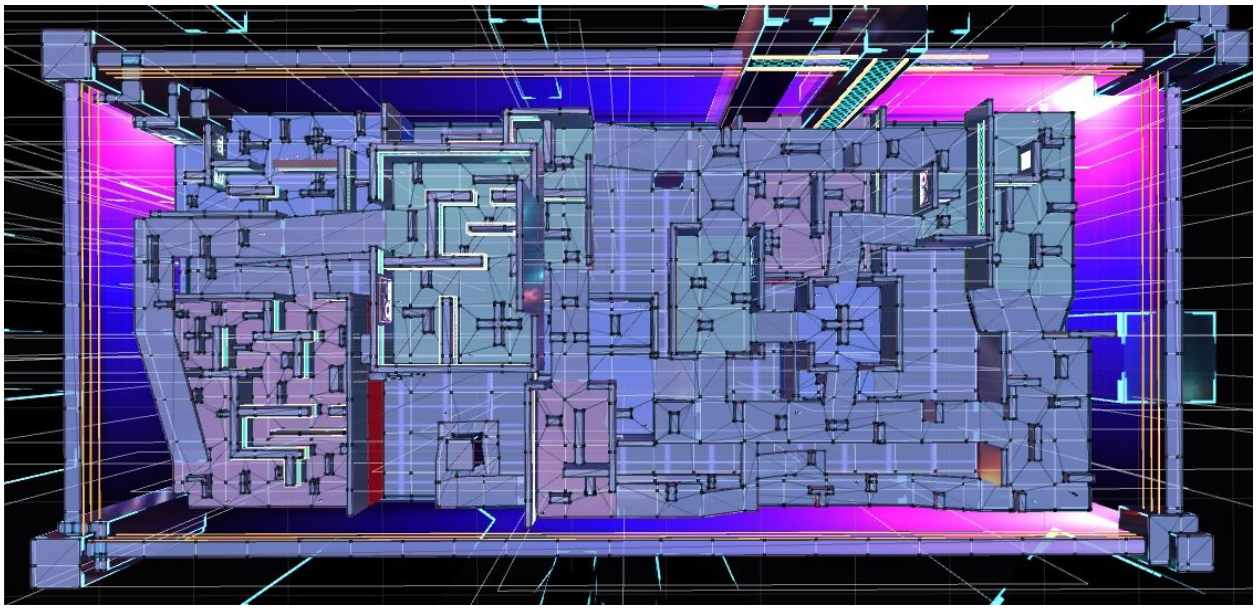
- Most of the level takes place in small rooms or on narrow platforms, so the idea of powerups to replenish energy was replaced by a slow regeneration.

- Other relatively small tweaks were made to improve the flow of gameplay, such as (but not limited to) increased movement speed and guard sight distance.

Artificial Intelligence

reb00t has a simplistic artificial intelligence that it is not too difficult for the player to avoid while also putting up a challenge and make the player pay attention to their surroundings.

The AI uses kinematic movement for its speed and rotation. By using the rigidbody's velocity, the AI can travel to different points and rotate accordingly when it turns around. To travel, the AI uses A* pathfinding on a Navmesh. Our original intent was to create a grid to calculate the A* algorithm but as development continued, we found the grid really hurt performance as well as hindered level design. We wanted a multi levelled area for the player to traverse which is not ideal for an A* grid graph. This is why we decided a Navmesh was the best way to implement pathfinding in our level. Below is the final Navmesh used for the AI's pathfinding:

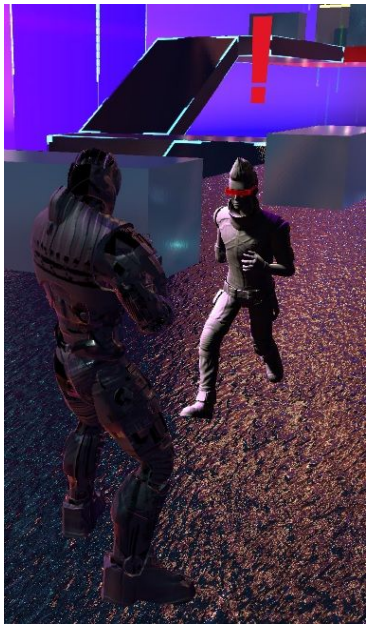


As mentioned before, the AI is simplistic in its decision making. In a game like reb00t, having an AI that is too intelligent can really frustrate the player and sometimes even make the game unbeatable. With that in mind, the team decided to have the AI make their decisions using a Finite State Machine (FSM).

All AI begin in a PATROL state where they are able to detect the player while they walk between a set of points unique to each of them. Each point from their set is decided randomly as the next point so the AI will never go in a constant pattern. This adds more strategy to the game as the player needs to make real time decisions when avoiding the AI instead of memorising their patrol patterns.

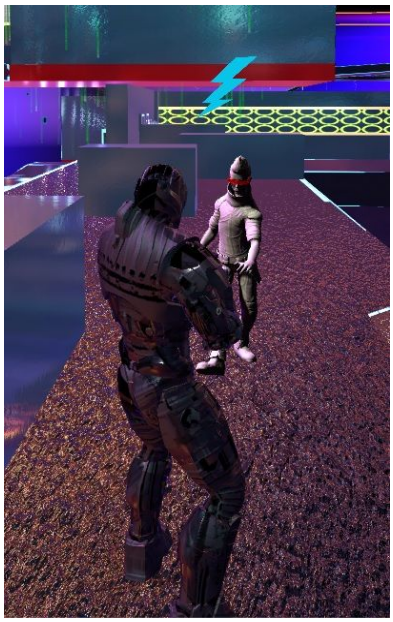
Each AI has a field of view (represented by a camera), that detects the player if he enters it. Once the player is detected, the AI casts rays out in front of him in many directions from the camera's position. If the player collides with any of these rays, the AI sees the player and enters a CHASE state (the player is not seen if he collides with a ray while hiding behind a wall or obstacle). This state makes the AI's next target position the same as the player's position and also increases the AI's speed so he can reach the player. If the player is caught or successfully camouflages, the AI returns to its PATROL state.

The final state an AI can have is the STUNNED state. This happens when the player successfully hits an AI with their gun bolt. Once the bolt and AI collide, the AI leaves whatever other state it was in and enters the STUNNED state. This makes the AI immobile and unable to see the player. The player has around three and a half seconds to bypass the AI without being noticed (Note: the player is still sent to the start if he collides with the AI while in this state).. However, once the AI leaves the STUNNED state, he re-enters the PATROL state and can once again detect the player. The stun is designed to be used as a last resort to bypass the AI if the player is not on a camouflaged floor.



CHASE

PATROL



STUNNED

The AI work independently. They do not notice if another AI is chasing the player since they will only be alerted of the player's presence if he collides with their specific rays. This was implemented to keep the difficulty at a reasonable level for the player so that they would not be surrounded with chasing AI.

Physics

In terms of collision resolution we use Unity's built in solutions. The player has an attached Rigidbody component and capsule collider. In order to move the player we apply forces of different magnitudes depending on whether the player is crouching, sprinting or walking normally.

In reb00t typical collisions consist of:

-Player and level

No action is explicitly called when the player collides with an object in the level. We let Unity's colliders and physics ensure that the player is not able to pass through level objects such as walls and floors.

If the player collides with a game object with the layer "Ground" we set that the player is grounded. When the player is grounded we allow the player to jump. If the player is not grounded we trigger the jump animation in the animator.

If the player collides with one of the colored ground objects and the player's color corresponds to the color of the ground object he is colliding with, we then set that the player is camouflaged. If the character is camouflaged he is invisible to the AI. Any AI in the alert state transitions into the patrol state.

-AI and level

No action is explicitly called when the AI collides with an object in the level. We let Unity's colliders and physics ensure that the AI is not able to pass through level objects such as walls and floors.

-Player and AI

When the player collides with an enemy AI we call the player's Respawn() method, resetting the player's position to the start of the level and replenishing the player's energy. All AI's that were previously on alert get their state reset to patrol.

-Player and switch

When a player collides with a switch the ability to press 'E' to activate it becomes available. Normally pressing 'E' doesn't perform any action but when colliding with a switch and pressing 'E' we trigger a player animation as well as activate the in game switch.

-Player and portal

When the player collides with the end game portal, an outro UI is displayed, the TimeScale in Unity is reduced to 0 while this UI is being displayed in order to prevent the player from being caught by the AI with it on screen. The UI then calls the load screen to load MainMenu.unity.

-AI and bullet

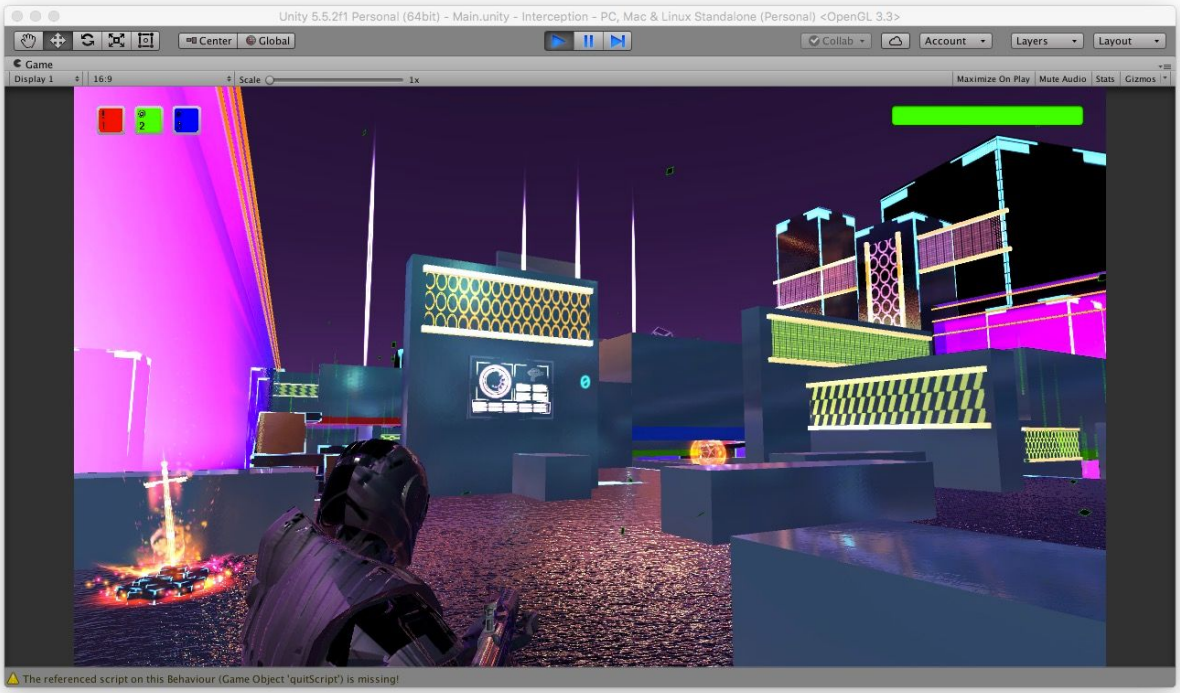
When a player bullet/projectile collides with the enemy AI, the AI transitions into a stunned state. In this state the AI is unable to change states for 3.5 seconds and is essentially disabled. Once the stun time has expired the AI transitions back into its patrol state.

Results

To ensure our program was working properly we playtested it extensively. After each new feature added we used the Unity editor in order to test that it was working as intended.

Once the game was finalized we built the project and played the game to completion multiple times in order to ensure quality of the final product.

To get a feel for the final product we've included a video walkthrough of what a player can expect playing through reb00t from start until completion. The video can be found linked on the title page.



User Manual

Running reb00t from a .exe:

The user will require the .exe file of the game as well as the reb00t_Data folder which contains all the necessary resources and assets used within the game.

In order to run the built version of the application, please run reb00t.exe located in the "Executable" directory.

NOTE: In order to experience the game as intended please ensure a 16:9 aspect ratio is chosen and graphics quality is set to "Fantastic" before launching the game.

Running reb00t from Unity:

The user will require all the Scripts and Scenes within the project as well as all the Assets listed in the Technology Plan section of this report.

In order to launch the application in Unity, run the MainMenu.unity file located in the "Interception/Assets/Scenes" directory.

Controls:

The player will require a keyboard and mouse to be able to enjoy reb00t.

W: Forward movement

A: Strafe-Left

S: Backward movement

D: Strafe-Right

Space: Jump/Skip cutscene

E: Activate nearby switch

1: Activate red camo

2: Activate green camo

3: Activate blue camo

Tab: Deactivate camo

Left control: Toggle crouch

Left shift[HOLD] + W: Sprint

Escape: Toggle pause menu

Mouse: Camera movement

Left mouse button: Shoot

Right mouse button[HOLD]: Zoom in + enables crosshair