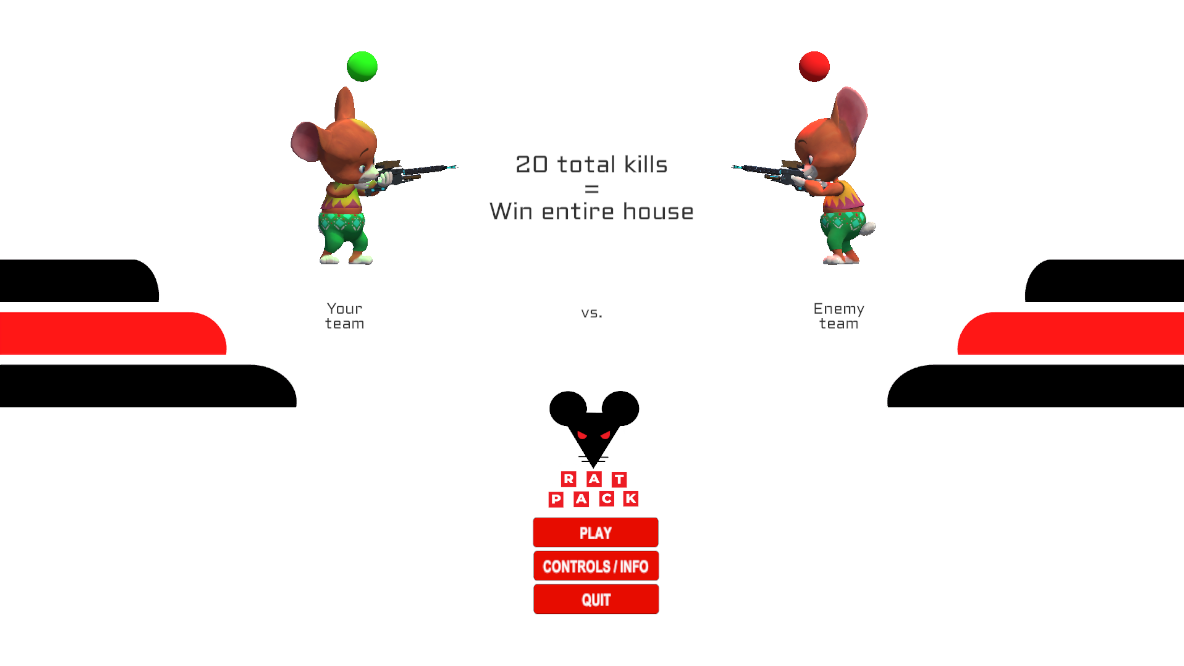
# **Intro**

Have you ever wondered what goes on inside your home after you have left the house? RAJAR Studios’ new game RATPACK welcomes you to find out but from a whole new perspective. This game allows you to take the form of a common rat and scurry around the house with your own rat clan, defeating all the members of the opposing clan and ultimately keeping all the food in the house for yourself. By utilizing objects in your environment and relying on your keen strategic skills, victory will come easily!



# **Team Members**

**Rohan Kadkol (Team Lead & Software Engineer)**

* + Implemented SCRUM project design
  + Responsible for integrating everyone’s work into our main scene
  + In-depth play testing
  + Player Controller
    - Movement (Walk, Run, Jump)
    - Shoot
    - Melee
  + Score Manager: Keep track of score and check if any team won
  + Player Models
    - Find character models for our rat characters (from mixamo.com)
    - Implement Mixamo animations on the character model
    - Link animation values with the player controller’s input
      * e.g. Pressing W triggers the walking animation
  + AI design
    - NavMesh, OffMeshLinks, AI logic
  + Troubleshooting
  + Website updates

**Annabelle Kanchirathingal**

* + Game Object Scripting
  + Documentation: final PowerPoint and final report
  + Game Map Design
  + Game Menu Design
    - Start Menu
    - Pause Menu
    - Controls
  + Playtesting

**Jacob Dickson**

* + Game Design
  + Map Design
  + Made colliders for map objects
  + Made ammo assets
  + Playtesting
  + Documentation: final PowerPoint, final report, OPPM

**Andrew Drummond**

* + In game assets for the player weapon
  + Player UI
    - Implemented Blood Layer
  + Sound Engineering
    - Composed all in game music, menu song, in-game song, and post-game song
    - Created sound effects for jumping, player damage, powerup, melee hit, and melee miss using BFXR and SFXR
    - All other in game sounds found on unity asset store

**Ryan Courson**

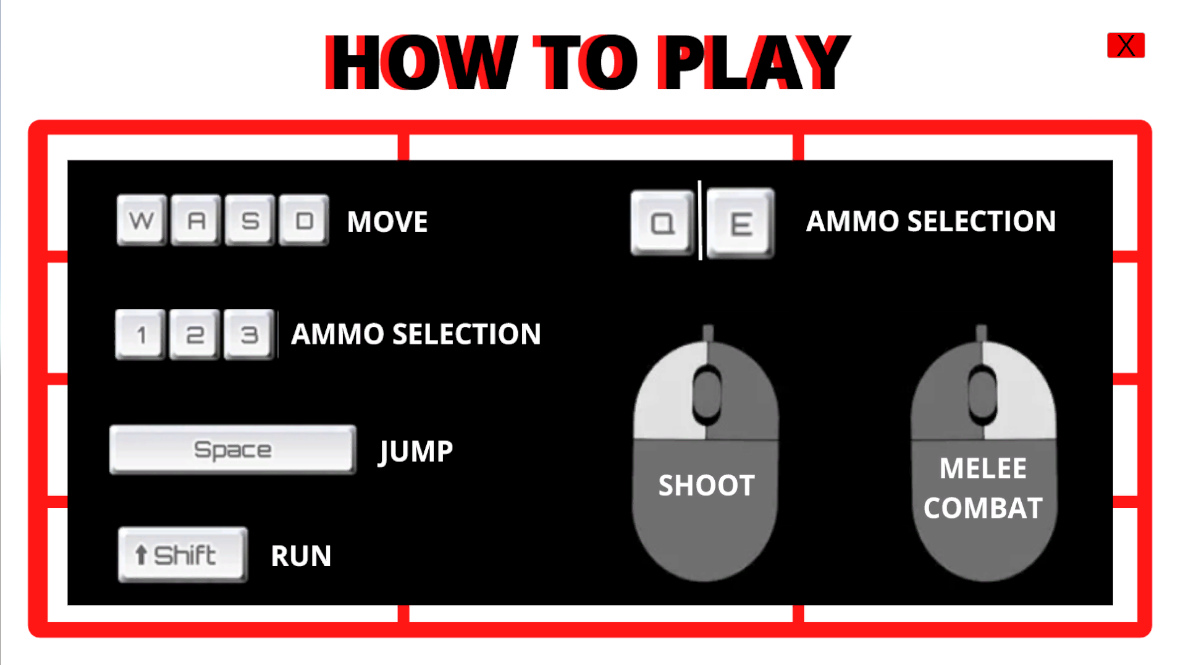
* + Implemented the HUD
    - Health bar
    - Team kills bar
    - Ammo Selector UI as well as scripting to change ammo type
    - Hit indicator
  + Documentation: Final Presentation
  + Item collection scripting

# **How to play the game**

**Controls**

The user can use the WASD keys on their keyboard to move the player back and forth & side to side. To make the player run, the user can hold down the shift key. To jump the user can use the space key. To shoot at their opponents the user can utilize a left click on their mouse. For melee combat, the user can utilize the right click on their mouse. To pause the game users can press down the ESC key on their keyboard. Lastly, ammo can be selected using the 1,2, or 3 number keys [for the three different ammo types] or by pressing Q/E [if the user wants to switch from the previous ammo or move on to the next]

* + Move: WASD
  + Run: Hold Shift
  + Jump: Space
  + Shoot: Left click
  + Esc: Pause game
  + Melee: Right click (new from after presentation)
  + Ammo selection
    - Use the 1, 2, or 3 numbers keys for the three ammo types respectively
    - Press Q or E to go to the previous or next ammo type respectively

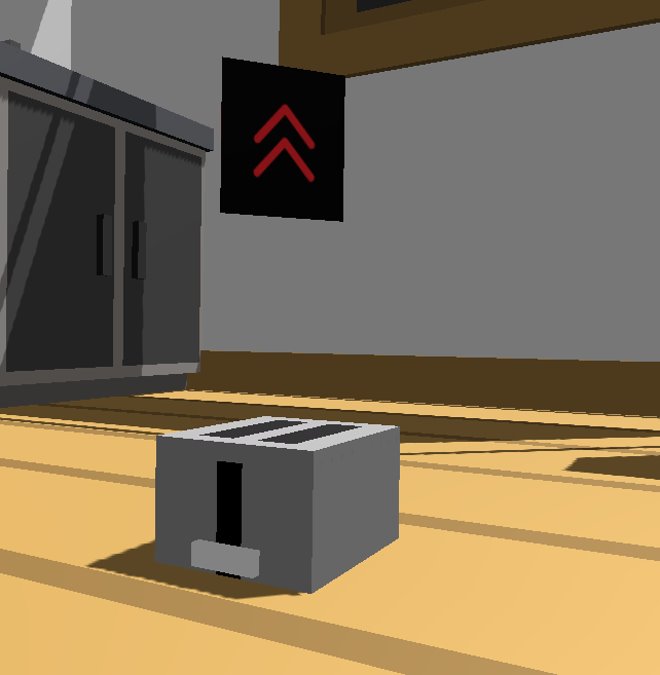


**Objective**

The first team to reach 20 kills wins the game. The main objective of the game is to try and take out as many of the opposite teams' players, without getting killed yourself. Utilizing the objects in the game environment, such as the soapy sponge or toaster, could give the player a great advantage over the opposing team; ultimately leading them to a victory!

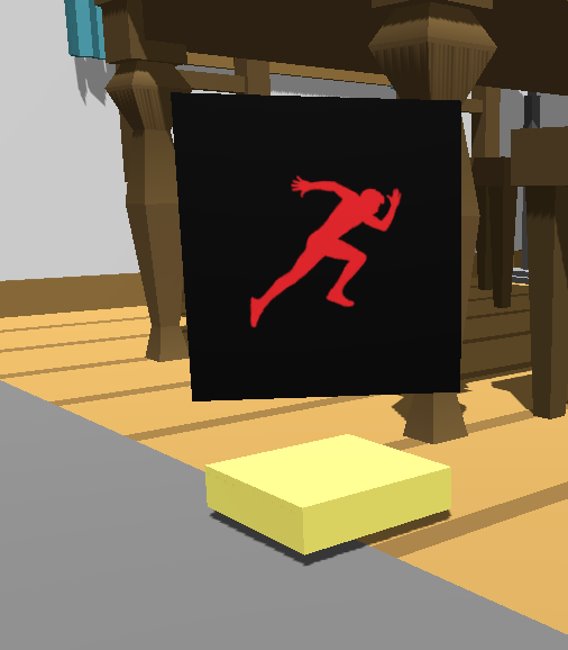
# **Game Rules and Mechanics**

**Toaster**



Allows the player to have a height advantage over their opponents.

**Soapy sponge**



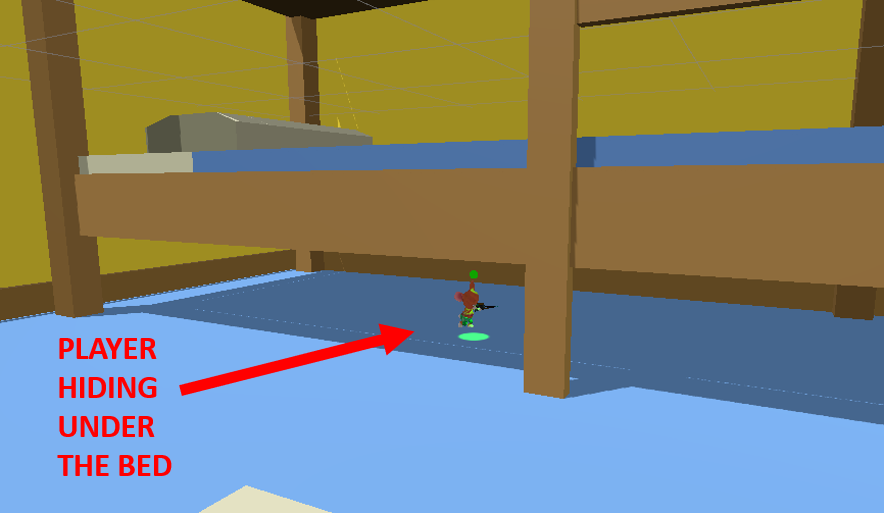
Reduces friction, allowing the player to walk/run faster & escape gunfire with ease.

**Tables for height**



Players can go on top of the table to get a height advantage or go under the table to hide from some of the gunfire.

**Go under bed**



While players can take the high road to sneak up on the enemy from above, players can also go underneath objects to attack from a well-covered spot. An example of this is the bunk bed in the bedroom

**Damage System**

We have three different types of ammo: chocolate chips, salt crystals, and sugar crystals. Chocolate chips cause the highest damage, sugar crystals cause medium damage and salt crystals cause the least amount of damage.



**Design Tradeoffs**

**Player classes**

In the midterm presentation and report, we mentioned the implementation of player classes. These could include classes such as Standard, Bruiser, and Scout. The original idea to differentiate the classes was to have some ammo types restricted to certain classes. We felt that this went against our original game design of ammo being scattered across the map to be picked up and used with the universal weapon. Not being able to pick up ammo despite being told the weapon is universal would not feel good for the player. Another way for us to implement player classes would be to make each class have unique abilities. Creating unique abilities that are not only fun to use, but also balanced would take a lot of time that we did not have. Because of this, we opted to not implement player classes.

**Minimap + Team Kills**

Previously we had mentioned our plans to implement a minimap and include team kills for each room. However, we realized that including a minimap on the player screen would cause the HUD to get overcrowded and make overall gameplay less fun. So, we opted to leave it out. We had also planned on including the team kills as a bar instead of writing the team kills for each room. This just seemed impractical for our current game requirements.

**Single Player**

Initially, the game was created under the guise of a multiplayer game. Where each clan would be made of individual players, kind of like Clash of Clans. The objective of the game would have mostly been the same with the same game mechanics. For this project, we decided to pause on the multiplayer aspect and keep it solely single player since that is what the class required us to do.

**3 maps**

Instead of 3 maps of the same room, we decided to opt for one game map with 3 different rooms. Since there’s only so many ways in which the living room and kitchen could be laid out, we decided to incorporate the two other rooms to keep gameplay exciting for the user. Using a game map with 3 rooms instead of having 3 different maps with one room made more sense as our game is supposed to take place inside of all parts of a home.

# **Play testing**

**Reduce player size**

This was a major concern we received from most of the players who tested our game. They felt like the original character was far too big to realistically recreate the POV of a rat inside a home. We used this feedback to dial down the size of the player. This allowed the player to have a more realistic viewpoint and made our game more practical.



**Show feedback when taking damage (blood effect)**

Originally, we did not consider to have an effect that showed our player taking damage. However, it made sense to include a blood splatter effect whenever the player took damage as it made the game look more realistic. It also made it more obvious to the user when the player took a hit if they weren’t able to see it clearly on the HUD while playing.

**Allow to go underneath the table**

This was another concern we received that allowed us to focus on making our game more realistic. Now our rat player has the ability to go under tables and even beds, giving the user a tactical advantage against their opponents. This increases gameplay and allows for more strategic thinking.



**Reduce jump height**

We decided to reduce the jump height on our player after some users felt like the original jump height was too high for that of a rat. A reduced jump height will lead the players into using some of the objects in their environment to help them jump to higher vantage points.

**Gun feels weird, doesn't show the arm & is too far right**

Since our game is from the first-person shooter point of view, it made no sense that the gun would be visible to the right of the player. To fix this issue we moved the gun towards the center of the player and made it so that our player was able to hold the gun. We also added a hand holding the gun so that it didn’t look like the gun was floating in the air.



**Fix the bug of going through the couch**

During playtesting we realized the player was able to go through the couch, which was not ideal when playing the game. To fix this bug, we added a collider on to the player and the couch which made it so the player could jump on the couch without going through.

**AI was too knowledgeable**

Originally our AI was able to detect the player even while not being able to see the player. This gave the AI an unfair advantage over the player. To keep a level playing field, we created a more complex AI which was only able to recognize the player once they come into view.

# **Overall Software Design**

## **State Machines**

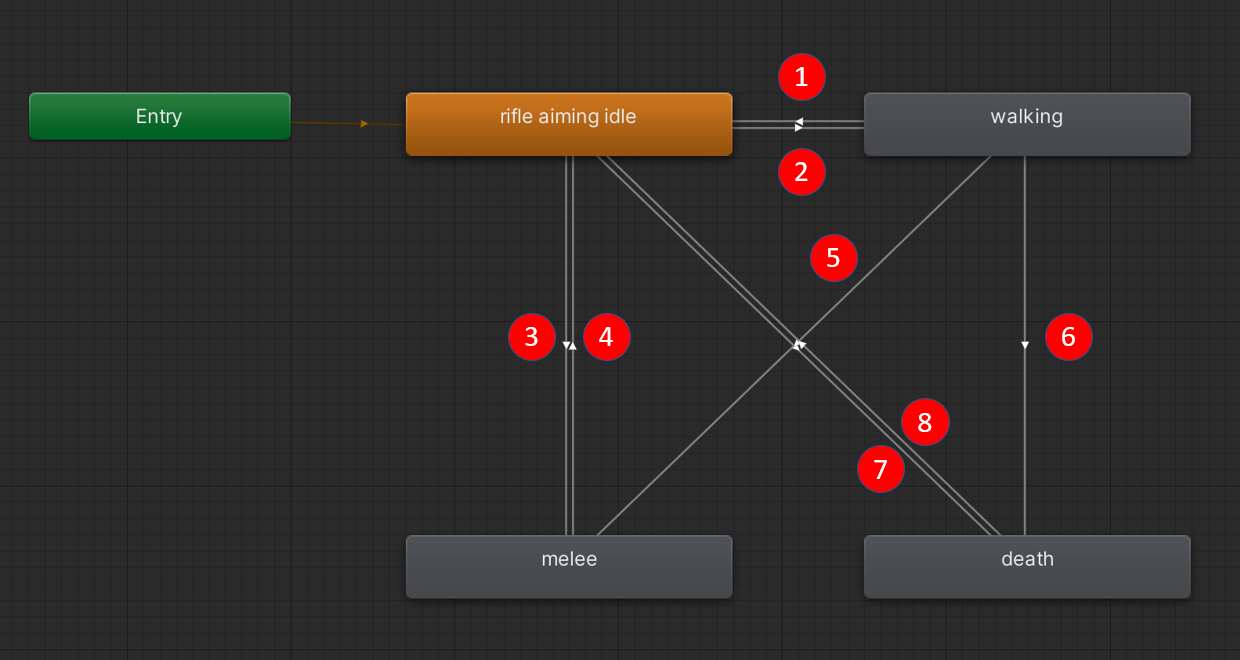
We used a state machine to decide which animation to play for each player model depending on the state of the player model

States:

1. Walking (bool)
2. Melee (trigger)
3. Death (trigger)

Transitions (labeled on the diagram):

1. Walking = false
2. Walking = true
3. Melee = triggered
4. Melee = after melee animation completed
5. Melee = triggered
6. Death = triggered
7. Death = triggered
8. Death = reset player back to alive

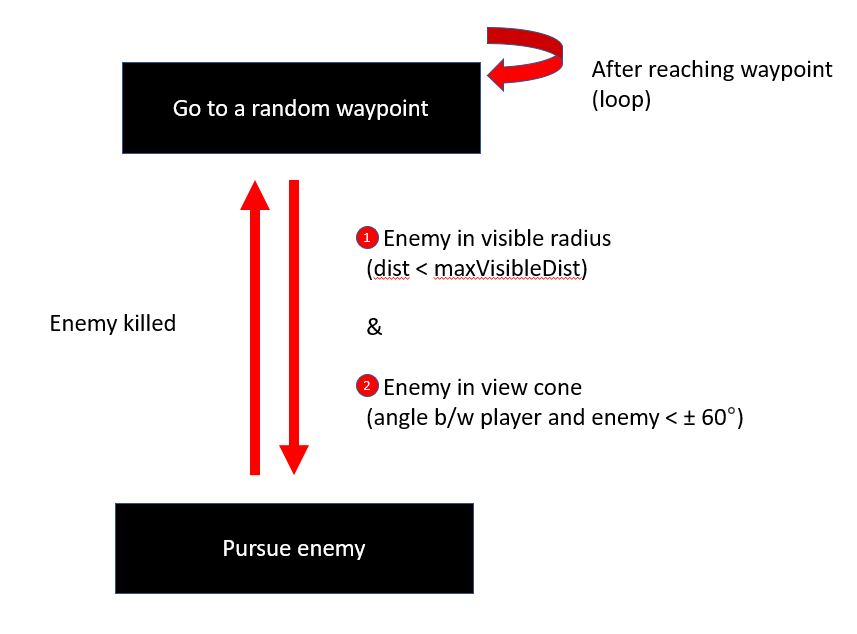


## **AI**

Our AI logic from the midterm presentation was simple. The NPC was completely knowledgeable of the enemy team member’s position at all times. Thus, the NPC would just follow the enemy players even though the NPC cannot realistically see them.

**Shortcomings of** **old system:**

* NPC blindly follows the opponent team’s players
  + Sneak attacks are impossible
* NPC can only traverse the ground level
  + We can easily go to higher ground and escape the NPC enemies



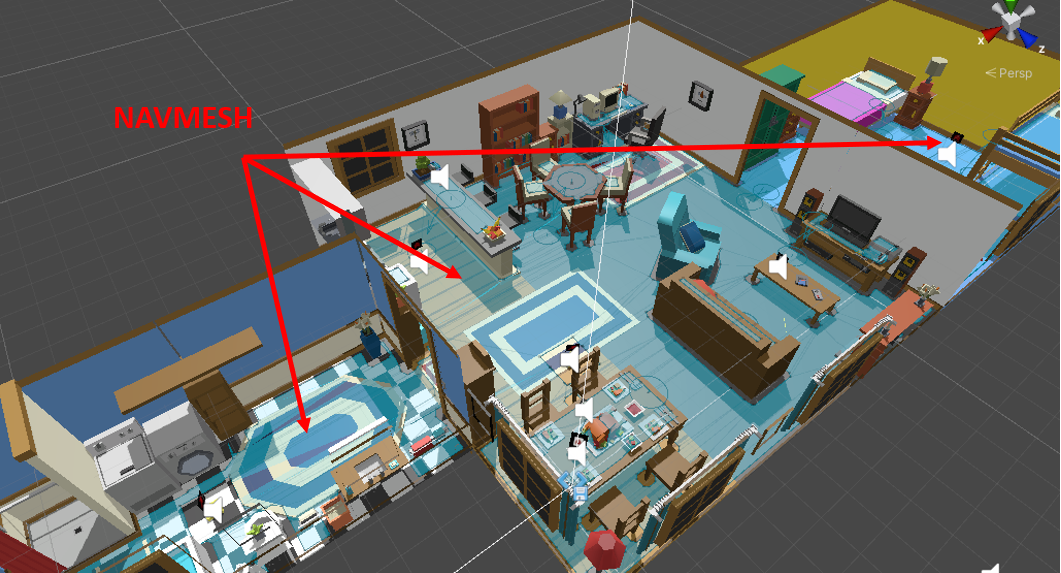
In our new system, we use the following:

1. Go to a random waypoint on the map
2. When spot an enemy, pursue to kill that enemy
3. After the enemy is killed, go back to step 1.

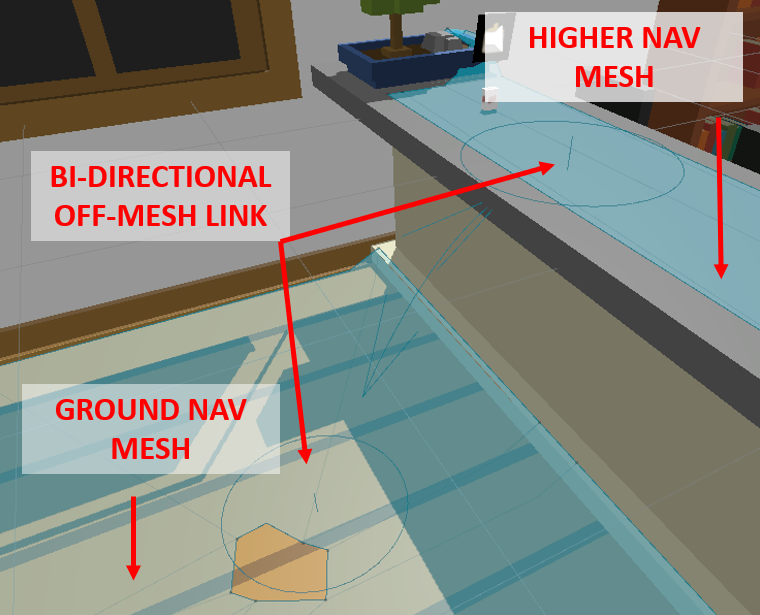
We define spotting an enemy if it satisfies two conditions:

1. Is the distance between the NPC and the enemy lesser than the maximum visible distance (dist < maxVisibleDist)
2. Is the enemy in the NPC’s view cone (angle between player and enemy < ± 60 °)

All the movement across the map is powered by Unity’s NavMesh and NavMeshAgent components. We first created NavMeshes across our three rooms and then attached the NavMeshAgent component on our NPC agents to traverse this NavMesh.



To enable our agents to traverse even higher levels, instead of just the ground level, we used bi-directional off-mesh links to connect two disjoint NavMeshes.



## **Visual Effects**

## **Bullet tracer**

We spawn yellow bullet tracers to visualize the path of the bullet



## **Audio Effects**

**Music:**

All music was originally composed using and is primarily guitar connected to Garage Band, along with other computer-generated instruments from the app.

**Sound Effects:**

Sound effects for player damage, powerup, melee hit, and melee miss were all made using BFXR and SFXR. All other sound effects were found from the unity asset store.

We used SFX to provide auditory feedback for the following tasks:

* Restock on ammo when the player collides with an ammo pack
* Health up: When the player colliders with the health up cheese bowl
* Melee: We used two different SFX to provide auditory feedback to confirm a melee kill or a melee miss
  + Melee miss
  + Melee successful kill
* Toaster Spring: When the player colliders with the toaster
* Soapy sponge wet feet: When the player collides with the soapy sponge
* Fire ammo
  + Normal ammo (Salt)
  + Heavy ammo (Sugar and Chocoloate Chips)
  + No ammo left sound

## **Credits**

Game Map Asset: <https://assetstore.unity.com/packages/3d/props/interior/simple-house-interiors-cartoon-assets-38027#description>

Mixamo Rat Player Model and Animations: <https://www.mixamo.com/>

Player Weapon Asset: <https://assetstore.unity.com/packages/3d/props/guns/sci-fi-rifle-low-poly-99671#description>

Footstep, Weapon, Healing, and Game Object Sound Assets: <https://assetstore.unity.com/packages/audio/sound-fx/classic-footstep-sfx-173668#description> , <https://assetstore.unity.com/packages/audio/sound-fx/shooting-sound-177096#description> , <https://assetstore.unity.com/packages/audio/sound-fx/cartoon-game-sound-5438#description>

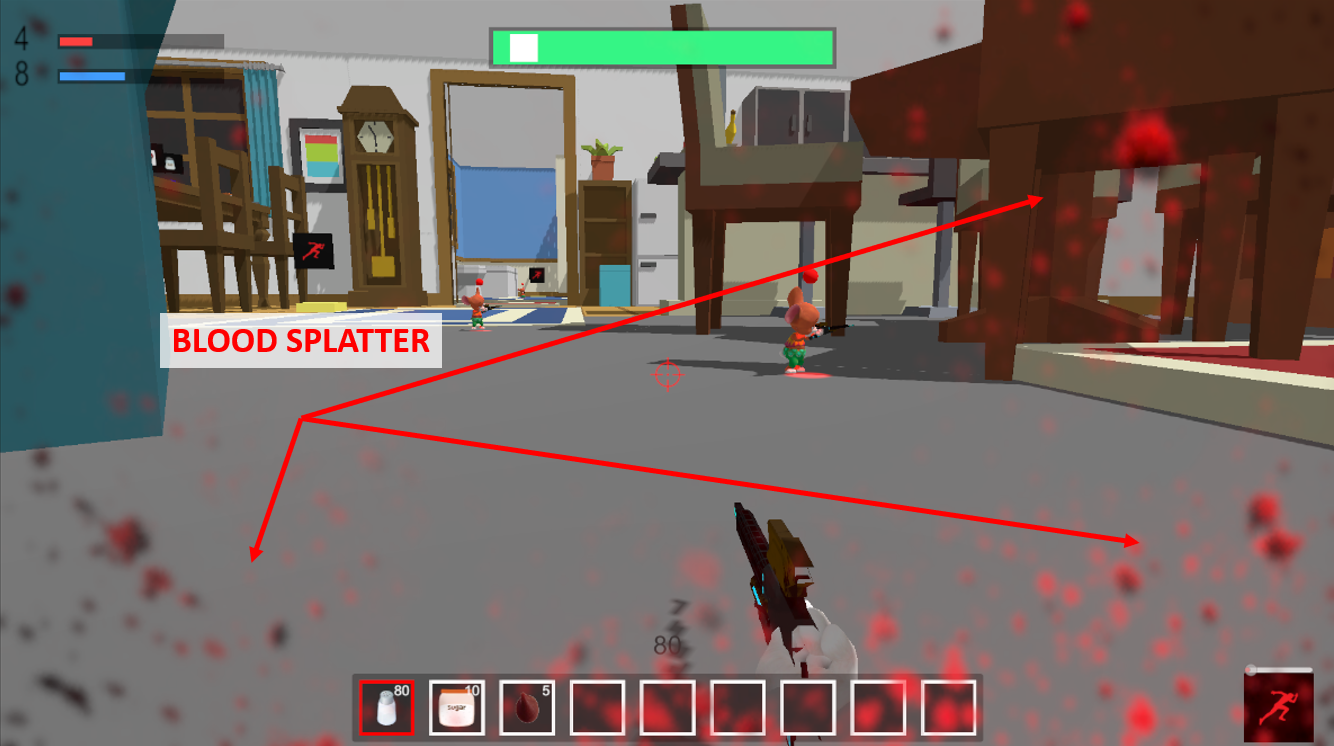
UFPS Package (for the blood splatter effect): <https://assetstore.unity.com/packages/templates/systems/ufps-ultimate-fps-2943>

Ammo Selector Image Assets: <https://www.canva.com/icons/MAEYJZZ2Axc-transparent-salt-shaker-with-metal-cap/> , <https://www.canva.com/icons/MAEQZUUu5xs-sugar-in-jar-cartoon-style-illustration/> , <https://www.canva.com/photos/MAB8OWvIriw-chocolate-chips/>

# **Other Major Components**

## **Blood Effect**

## The Blood Effect is an invisible layer, linked directly to player health gradually increasing the visibility of the splatter the lower the player's health. This is beneficial as it gives a noticeable signifier as to when the player is hit and is near death rather than having to look towards the health bar.



## **Maps**

Our game map has three different rooms for the player to roam. Each room has a different layout and different game mechanics incorporated to keep gameplay fun and exciting for the user. These rooms also have different variations of height/tactical advantages for the user to take advantage of.

*Living Room*  *Bathroom*



*Bedroom*

## **Ammo Pickups and Assets**

There are ammo pickups across the map for each of the three ammo types

* + Salt
  + Sugar
  + Chocolate Chips



Upon colliding with an ammo pickup, the player’s ammo reserve for the respective ammo is filled. Models for each of the ammo pickups were made in unity using the basic 3D shapes. The PNGs used to texture the models were found online, links to them are below

* + Salt base: <https://www.fontmirror.com/salt>
  + Sugar: <https://www5.flamingtext.com/net-fu/jobs/27709231777608741.html>
  + Chocolate chips: <https://www.vecteezy.com/vector-art/3406428-chocolate-text-effects>

## **Firing Mechanism**

**Hit Scanning**

To improve the performance of our firing mechanism, we decided to implement hit scanning instead of using bullets as game objects and detecting collisions. We did this by shooting a raycast from the camera center (tip of the gun) to the camera’s forward direction and noted where it hit. If the raycast hit an enemy, we deduct health from the enemy depending on the ammo type the player used.



**Camera Shake Effect**

We used a camera shake effect to simulate the gun vibration and recoil.

## **HUD**



The in-game HUD is used to show players their health, current selected ammo, amount of ammo for each ammo type, the team kills bar (to show your team’s progress in the match), the crosshair (to indicate where the player is aiming), the speed up indicator (when we touch the soapy sponge), and finally, our weapon.