## Comparable State Of The Art

The First Person Shooter genre is extremely popular with franchises such as Call of Duty and DOOM. Both games fundamentally deal with the same problems handled in this project, including obstacle and bullet collision, as well as enemy AI. With Call of Duty having a larger focus on realism, and DOOM, being a part of the “Arcade” FPS genre, with less focus on realism. In this project the geometry the player traverses is extremely simple only consisting of a handful of different shapes. Both DOOM and Call of Duty handle more complex terrain, including slopes, stairs and natural phenomena such as mountains. Each respective game handles bullet detection, but in addition have several variants on weapons, with each variant requiring different behavior for collisions, a feature not present in this project.

## Implementation Compared To State Of The Art

In Call of Duty Black Ops 4, when in close quarters, bullets use the algorithm known as hitscan(3). Hitscan sends a ray in a direction and checks if it comes into contact with an object. For far away shots, projectile mechanics are used, which better simulate physical interactions such as bullet drop(2). This project uses a similar method for achieving hit detection. The primary difference in this project’s algorithm is the line of the bullet is not infinity, however, the level is constructed to where this limit could not be reached in a way that would directly affect the player. As no enemy will appear out of a player's range of fire. DOOM is not focused on realism and structures the gameplay, and AI of the enemies with the concept of push forward combat, having the player always moving. To do this, the AI in DOOM uses Hierarchical Finite State Machines(HFSM)(4). HFSM represents the agent’s current state, and what states the agent can transition to, by chaining together states, unique and dynamic AI behavior can be achieved. In order to handle large state spaces, HFSM, clusters states together, and allows some states to enter a new cluster(4). Additionally, to keep the flow of gameplay, enemies actively try and appear in open spaces to avoid clunky battles where enemies are stuck behind cover(4). DOOM’s environment is extremely complex as compared to this project. In this project, the bot enemy is in an empty square room and never goes out of bounds. It does not contain any states and is always, using random points trying to make contact with the player, using epsilon A*.

## Key Algorithms

Several Key algorithms were created in order to achieve a playable first person shooter. The two most important algorithms would be how the camera functions and collision from the player's gun to entities. The camera was adapted from the camera originally provided by Liam, it was changed to allow for a more traditional first person movement, such as jumping, rather than being able to move up and down freely. This required the camera to be in either an AIR or LAND state to determine if a jump was available for the player’s use. The next important algorithm was detecting sphere and line intersection. This was used for creating the hit boxes for all enemies. This was done by using the Circle-Line Intersection equation and extending it to 3D(1). With these two equations, the player was able to interact with several objects, such as

the targets or wasp enemies. This collision check, unlike other First Person Shooter games, does not check for obstacle collisions for bullets. This means the player is allowed to hit enemies behind a wall, as if one was not there. For the level design should not be too noticeable by the player. The only time this is apparent is in the corridor section fighting wasps, as the player is able to shoot through the door.
The player has two enemy types to take down. The first one, the wasp uses the boids equation and additionally has an attraction force towards the player. This causes the enemy to have simple behavior, without stacking up on one another.
The second enemy, the bot, randomly generates points, puts its position as the start, and the players point as the end goal. It then conducts epsilon A* search from the bot's position to the player. Every second the bot's and player's position is updated, connected to the same random points and a new path is generated. The random points are only generated in the boss room, this was set directly in the boss class, it would require additional updates before a more general use could be used. The amount of random points and which points to connect to were acquired by parameter tuning.
The player's environment is broken up into three different types of obstacles, rectangle, quad, and circular(cylinders). The rectangles and circular obstacles have collision checks for the y axis, which check if the player is above the obstacle currently and the next position will be inside or below the obstacle, if this is found, the players position is set to the top of the obstacle and the player's state is set back to landed. All obstacle types have collision checks. The rectangles and circular obstacles check if the current position is outside and does two seperate checks for the x and z axis. If it is found adding only the x axis would not result in invalid player location, the x axis will still be updated and vice versa for the z axis. This makes sure that if a player runs into a wall, the player's movement does not fully stop. Quad obstacles are set either north to south or east to west, if the player is within the rectangle and tries to cross a line they are stopped.
Finally two basic particle systems were implemented, one for the smoke for when the pistol is fired, spawning many translucent circles in front of the player. The second particle effect is when a wasp enemy has been shot, green circles explode out from within and the circles then fall to the ground.

## Computational Bottlenecks

This first computational bottleneck of my program is how obstacle collisions are checked for the player character and wasp enemies. This issue is lightly handled in the current version. Where after entering the boss fight, most obstacles from the target room are no longer checked. However, this is programmed just for this level, for a more general approach, one could check only nearby obstacles or have a grid system and only check in current/adjacent grid blocks. This same issue would occur for the pistol weapon, when firing. When the gun fires all entities are checked for a collision. If many entities were spawned, the bullet collision check would take too long, creating an unresponsive game.
There are several audio limitations, the majority of audio clips in the game are loaded into memory upon starting the level, this reduced performance issues compared to when files were

accessed on a need only basis. If a large number of sound files were to be used for a particular level, loading the audio at different points would reduce an initial lag when starting. There are current limits to audio use in the game. If too many audio files are played at once, all begin to lag. This could be due to the underlying audio library, or the way the audio is being played, however, I am not sure what a possible fix would be at this time.

## Future Work

Every aspect of this game could be improved. First, the hit detection of bullets should be updated to handle obstacle collision, and possibly handle only colliding with a single enemy. In modern First Person Shooters this is done with the Hitscan algorithm. The geometry the player interacts with is not varied. To increase the complexity of level design, adding the ability to rotate obstacles could be used. The audio of the game can at times use too many samples, this caused issues trying to record audio, in future work, using either a new library, or the current library in a new way could be done to have better quality audio. Finally, having more enemy types and having the epsilon A* algorithm enemies have to traverse around obstacles.

**Sources:**


**(1)**
**Weisstein, Eric W. "Circle-Line Intersection." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Circle-LineIntersection.html**

**(2)**
**Hitscan. Call of Duty Wiki,from https://callofduty.fandom.com/wiki/Hitscan**

**(3)**
**Jung, Tristan (2018-07-14). "How Do Bullets Work in Video Games? - Tristan Jung". Medium. Retrieved 2019-07-18.**
**https://medium.com/@3stan/how-do-bullets-work-in-video-games-d153f1e496a8**

**(4)**
**Thompson, T. [AI and Games] (2018, August 5) *Cyber Demons: The AI of DOOM (2016)* | *AI and Games* https://www.youtube.com/watch?v=RcOdtwioEfI**