The design for my final project is a patrolling enemy that will chase after you if you get too close, but can also be evaded. The enemy can also alert the other guards if it finds you. There is also a security camera system that'll alert the guards to pursue you. It works by checking to see if the player is within the view range of the enemy. If it is, the enemy will approach the character. The enemy moves slower than the player to allow the player to escape. If the player then moves outside of the view range, the enemy stops following and begins its patrol once again. The security camera works similarly to the enemies, where if you stand within its view range, indicated by a spotlight, it'll trigger the enemies to pursue you since they now know your location.

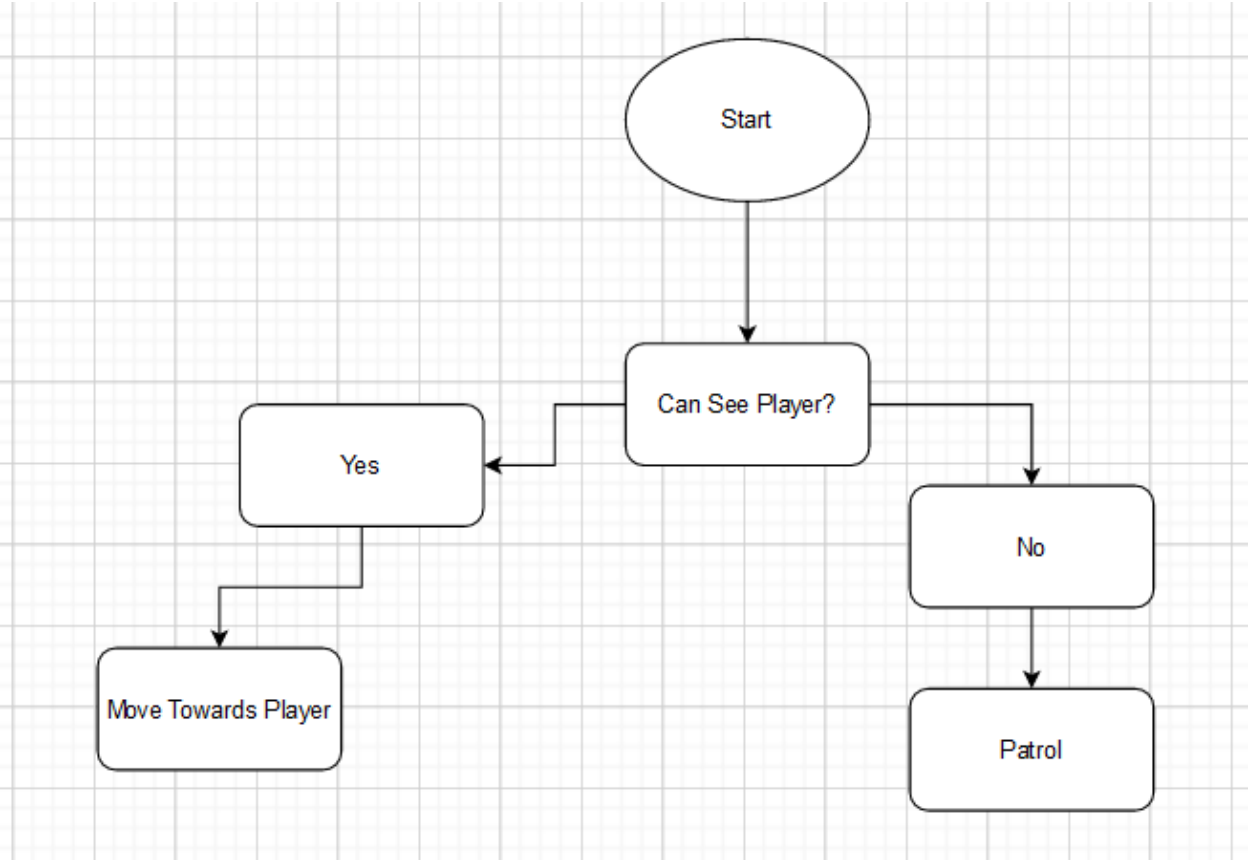
https://youtu.be/7dEaDeShANI

For the sake of viewer experience, the delay function was replaced since my laptop cannot run faster than <1 fps when using my custom delay method.

This sort of mechanic is versatile, but it would mainly be used in a stealth or horror based game, where evading enemy detection is crucial. Games such as Assassins Creed or Resident Evil come to mind. Whether it's intense chase sequences or completely avoidable chases, this will come in handy. This would also work if you placed down multiple enemies, as they would all search for the character and not each other. Another possible application would be enemy attack sequences. If you don't let the character lose sight, you could have the enemy begin attacking. In its simplest form, this is an idle enemy approaching the character once it sees the character. This aspect of the mechanism could be used in action games as well such as God of War or within the combat mechanisms of RPGs such as Genshin Impact.

The design is centric upon the enemy's and security camera's pawn sensing ability. This ability within Unreal allows the enemy to have a line of sight, as well as a listening radius. For this mechanic, we're only using the line of sight. Every tick, the enemy and security camera checks its respective lines of sight to see if it can see the player. If it can see the player, it sets its "Can See Player" boolean to true, and slowly approaches the character. As long as the "Can See Player" boolean is true, the enemy will approach the player. Once the player leaves the enemy's line of sight, the "Can See Player" boolean is returned to false, and the enemy goes back to its random patrol. Once "Can See Player" has been set as true for one guard, all other guards will also begin their pursuit.

The random patrol starts out by making sure the player isn't visible. As long as the player isn't visible, the game will randomly pick a point within the specified radius (in this case it is 500m, but this is modifiable), wait for a specified amount of time (in this case 2-5 seconds, but this is also modifiable), and will move to the point. This process repeats every tick until the player becomes visible to the enemy. If the random point picked is not able to be walked to, the enemy will not move for that turn of the process.



HomingEnemyCharacter
int mMinWaitTime
int mMaxWaitTime
int mRadius
float mDelay
float myRandomInRange(const int& min, const int& max)
void myDelay(const float& time)
FVector myGetActorLocation(const AActor target)*
FVector myGetRandLocationInRadius(const FVector& origin, const int& radius)

SecuritySystem
bool mCanSeePlayer

SecurityCamera
float mDelay

The security camera functions very similarly to the patrolling enemy. If it detects the player is nearby using pawn sensing, then it sets the "CanSeePlayer" variable to true, which triggers the enemies to start chasing you.

I started out with the functions that utilised the "Random" function within c++. This includes MyRandomInRange and MyGetRandLocationInRadius. Given a range and a

minimum value, these two methods are quite similar. The main difference is the random location randomises three points, whereas the random in range only randomises one. Delay utilised CTime to calculate the elapsed time. It checks the time each frame to see how much time has elapsed. If the specified delay time has elapsed, then it terminates. Lastly, myGetActorLocation utilised the Unreal function "GetActorLocation" to get the location of a given target.

I started out with a very different design, and a simpler one, than what was the final product. Initially, the enemy didn't patrol and only chased when the player got close. This worked by using a method I created which first checked to see how far the player was from the enemy. If the player was within a range specified by a variable, then the enemy would approach using the "AI Move To" node. Since the distance was checked every tick, and the enemy only moved when the if statement was true, it naturally would also stop chasing once the player got too far. I also didn't initially have the security camera feature at all.

Now, the system is more complex, as it utilises the Pawn Sense feature. This creates a line of sight for the enemy and security cameras, which is more specific than just standing within a radius. With this feature, we instead check to see if the player is within the enemy's or camera's line of sight each tick rather than the distance between the enemy and the player. We still utilise the "AI Move To" node, however it now serves two functions since the enemy is patrolling. While patrolling, the enemy will move to a random location within a radius on a random within range timer. Once spotted, the node is used to track down the player. Once the player has left the enemy's line of sight, it's back to patrolling for the enemy.

I had to do quite a bit of research to get the security camera and "alerting the guards" features working. I implemented what is called a Game Instance, which acts as a singleton to store game variables. By utilising this, I could set one variable to be true for all enemies in one fell swoop rather than attempting (and failing) to set the variable individually for every enemy. I also did research into the spotlight feature within Unreal, since I found it would be helpful to have a visual indicator to show the view radius for the security cameras. Once I had sufficiently fiddled with all the settings, I felt happy with how the security camera turned out.

Once the blueprint was done, it was time to work on converting the blueprint into c++. I started with the variables, as I found that to be the easiest piece to convert. The only variable I didn't convert initially was the "reference to self," as that is a more complex variable. From there I went to where I felt most confident in my ability, which was the methods involving random integers. Those went by fairly easily, except for a slight hitch with converting the random reachable point within radius. I had no way to detect if the ground was reachable, so I left it as is and continued on. From there is where things got the most difficult. I didn't have much experience with time, and this was my first interaction with Unreal Engine, so I wasn't sure how to convert the two different delay functions as well as the "AI Move To" node. In the end I asked for help from the professor on how to do the delay method, and his solution worked perfectly. I ended up deciding to leave the "AI Move To" node alone, as it was too complex for me to recreate.

The delay method took a lot of trial and error as well as consulting with friends. As I'm very inexperienced with working in time, I knew I would need help but certainly not to the degree I did. I tried numerous methods for creating my own timer, but none seemed to

work. The delay method also caused significant lag when playing, however it still worked. All attempts ended in the enemy spinning like crazy as it went from point to point in rapid succession. As previously stated as well, there were some complications with getting a random reachable point, as I couldn't figure out a way to check if there was actually ground there to walk on. The complexity of the "AI Move To" proved very difficult to translate, and in the end I left it as it was. I couldn't figure out how to make the enemy move, as the "AI Move To" node required use of the character controller.

https://www.youtube.com/watch?v=qHnRYr01fCw < blueprint for HomingEnemyCharacter derived from this video, code was done on my own

https://www.codespeedy.com/how-to-create-a-timer-in-cpp/ < code for delay function derived from this