AI for Games: Individual Project

## Git link: <https://github.com/rhodes167/AIProject>

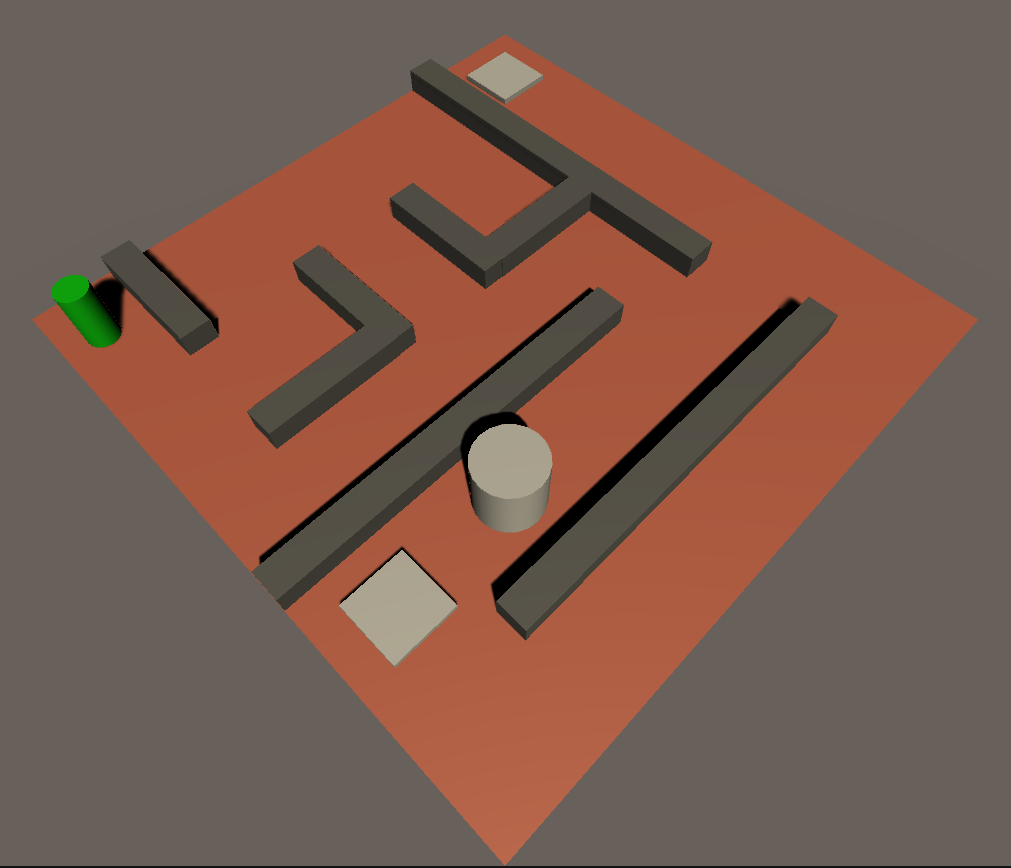
# Introduction

This project involves the production of a technical demonstration for a simple stealth game system. The game involves a small surface upon which two actors can move around. One of these actors is controlled by the player, the other is controlled entirely by the AI systems implemented. Also on the surface, are numerous walls forming a simple maze. The actors will be required to navigate around these obstacles in order to interact.

The player character must navigate these walls, and plot a course to a location selected by the player using the left mouse button. The other actor, filling the role of a guard, will patrol between two points in the maze. Upon reaching one of these points, the guard will wait before proceeding to patrol back towards the other location.

At any time during this patrol behaviour, if the player character gets close enough to the guard, the guard will become alerted. The alerted guard will then no longer follow the patrol route previously outlined, but instead pursue the player character. If the player manages to put enough distance between the player character and the guard, the guard will give up and cease chasing the player. At this point, the guard will return to its non-alerted state, and resume its previous patrol route.

A NavMesh was deemed necessary to ensure proper navigation around the wall obstacles. The AI controlled guard character will feature behaviour determined by a Finite State Machine. The choice to use a Finite State Machine was made due to the simplicity of the AI decision system. With only a limited number of possible AI agent states; Patrolling, Idle and Chasing, this AI would not require any more advanced system.

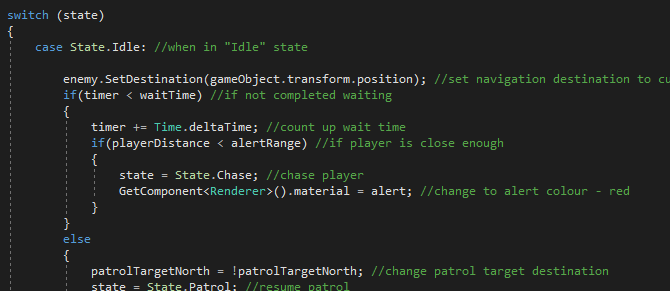


# Implementation Report

# The project was undertaken in the Unity game engine. This provided a good framework for a simple technical demonstration like this. The NavMesh was achieved through Unity’s NavMeshAgent AI system. This is a shortcut that was used so as to allow for greater focus on the primary AI system, the Finite State Machine.

Implementation of the FSM allows for transition into the Chasing state at any time. However, transitioning out of the Chasing state will always return to the Patrolling state. This was decided upon to prevent idling in arbitrary positions on the game surface, and cause the guard to begin travelling to its previous destination immediately.

The Finite State Machine itself was structured around using a switch statement and an enumeration. This approach, while not the most expandable, is sufficient for a small project such as this.



Decisions regarding the changing of states was handled through if statements within the switch statement. While in larger projects, this would quickly become cumbersome, it was selected here due to the relative simplicity and limited scope of the project. There were also no checks undertaken to ensure appropriate behaviour upon changing FSM states. In industry, this is often included so as to avoid errors and aid in debugging. This was not included here as the triggers for changing state are based on easily determined variable comparison, for example distance between game objects. As such, it is unlikely to cause problems in this way.

Originally, a behaviour tree was considered as opposed to the finite state machine. This was deemed too complex for this project, as all desired functionality could be achieved through a simple FSM.

# Discussion

The behaviour observed in the finished demonstration matches the desired outcome. Further expansion in functionality could be easily added through the implementation of a vision cone for the Guard non-player character. This would allow for the player to traverse closer to the guard, as long as the player remains out of the guard’s line of sight, for example behind it.

An alternative implementation of the finite state machine would cause the guard to enter the idle state in the position at which the player character escapes. This could be used to better simulate the guard “searching” for the player. The decision not to act this way was to allow for a faster paced demonstration with a more prompt “reset” condition. This idle behaviour could be combined with the above suggestion of a vision cone by having the guard “look around”, further personifying the character.

Furthermore, the modification of the checks to ignore sightings through walls would allow for more engaging gameplay. The current implementation models the walls as “waist height”. While a common trope for video games, this was selected as a simplification consideration, as opposed to one that benefits the product. One method through which this could be implemented would be a raycast from the guard character, projected in its line of sight. If it collides with a wall prior to the player, detection would not be triggered.

More bespoke pathfinding could have been used, as opposed to the current in-built Unity system. Examples for this could include the use of A\* to determine an optimal route across nodes. This was avoided to add simplicity and allow for focus to be put on the finite state machine component. The Unity implementation is also both effective and efficient for this project’s purpose.

Behaviour trees could have been used in place of the finite state machine. Finite state machines saw great popularity in early games and, in particular, those following the release of Valve’s Half-Life. Behaviour trees are much more popular in more recent games, approaching the de facto choice for AI behaviour. While behaviour trees allow for more flexibility and goal driven action, the added complexity is not necessary for the purposes of this project, and adds nothing to the functionality in this use case.

# References

Technologies, U., 2021. *Unity - Manual: Unity User Manual 2020.3 (LTS)*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/index.html> [Accessed 2 June 2021].

Technologies, U., 2021. *Unity - Scripting API: AI.NavMeshAgent.SetDestination*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.SetDestination.html> [Accessed 2 June 2021].