AI Report

# The Game

The game I Designed the AI for was primarily a 2D fighter, this was because I wanted the AI to provide an engaging combat encounter for the player. The chosen style also meant that ideally the AI could be repurposed for other 2D games that might require engaging combat such as combat heavy platformers.

# AI Design

This Chosen Genre meant that at the most basic level the AI would require a Decision tree, and I also decided to implement a state machine for the AI to try and make it seem more realistic. Because the game is in real time this meant that the AI techniques had to be simple so that the computer does not slow down when trying to run the game.

In order to achieve this, I had a root function for the AI the purpose of which was to determine the state the AI should be in, this was called twice every second as supposed to every frame, and while this meant the AI reacted slower, the required computing power was less.

# Finite State Machine

As it is the highest level of the AI and the first thing that is calculated, it makes sense to talk about the finite state machine used by the AI. In this case the FSM was quite simplistic, only differentiating between two states (aggressive and defensive) and only had one condition for switching between the two, and due to the mechanics of the game there was no way for the AI to satisfy the conditions to switch back.

Furthermore because of the structure of the code, the state machine is checked twice every second, even though due to its simplicity it might make more sense to determine the state when the AI takes damage as this is the only point where it may change, the issue with this however is that the AI may take damage multiple times a second, so while the average computational requirements will be lower the spikes would be higher, which could potentially be more problematic for overall performance.

# Decision Tree

Depending on the state the AI will then go through the rest of the decision tree, the tree itself is relatively simple and shallow. The main function of the decision tree is to make sure that the AI is at the correct distance from the player so that it can connect with its attacks.

# Action Execution

The actions the AI had access to were fore the most part identical to those available to the player, to prevent the feeling of being cheated.

In terms of the code itself actions that weren’t a continuous process (Attack, Block, knife throw etc.) were written as functions that the decision tree would call when the criteria were satisfied.

However, for actions that were continuous (Approaching, Running away) the decision tree would change a bool variable that was used in the fixed update function to actually move the AI closer or further from the player. The result of which meant that these actions acted more as a separate state machine than as individual actions, and whilst this makes little difference to the end user experience, I would have liked to have implemented a more consistent structure to the actions.

# Event System

There is currently a very simple event system, so simple in fact that it can hardly be called an event system. It consists of calling a function from the game controller (that in turn calls a function from the AI) when the player throws a knife, the idea of which being that the AI will block the projectile.

# Further Work

One of the first things I would like to improve is my action execution, firstly I would create an action struct in unity, thus creating a standardized format for my actions

# The Game