Development Report

Game Development - Gade7321

Francesca Fitzgerald ST10143178

2024

## Changes From Part 2

The game state representation described in part 2 of this project is still evident in the architecture of this POE. With the implementation

## Chosen Behaviour

**Chosen Behaviour & Motivation:**

I decided to use a Monte Carlo Search Tree (MCST) algorithm as my ad-hoc behaviour in this game. Because my game does not have finite game states or game moves, using a MCST algorithm is far more effective and efficient. Using a minimax algorithm would have required me to calculate and record every possible game state in a data structure, requiring a lot of memory. I chose to use the MCST method because it can simulate the game and access the necessary information to make a decision, each time the AI needs to move or evaluate the game state.

The game can go in indefinitely, much like *Chess*, because of how the game is structured. This is the main reason I decided to use the Monte Carlo method. I have done research on what games use MCST verses Minimax, and an MCST algorithm is often chosen due to the level of strategy required in chess-like games (Medium, 2024).

**Implementation:**

In part 2, I derived a utility function to be used to influence my AI’s decisions. However, because I used an ad-hoc behaviour algorithm (not a machine learning algorithm), I did not need to implement this utility function.

The selection phase of the MCST algorithm requires the implementation of the Upper Confidence Bound (UCB 1) formula:

The first term evaluates exploitation, and the second term evaluates exploration. The code in my program uses each game state or node’s wins vs loses and how many times that