Game AI Documentation – Mappeinnlevering

Football Game

My solution for the Game AI exam was to have a game with 2 players, one real and the other as an AI that serves as a keeper. The main objective is for the player to shoot the ball into the goals of the opposing team to score a goal. The simple AI is capable of tracking the ball, collecting the ball in the field and giving the ball back when the player misses.

This solution makes use of two Finite State Machines, one on the keeper and a very simple two state machine on the ball. The A* algorithm was also implemented successfully but the calculation of the fastest path seems odd in my opinion.

Process

Working with this exam has proved to be a fun challenge with many highs and lows. When I started on the project I initially had in mind creating a whole 5 v. 5 FIFA like AI v. AI simulation but that proved to take too long to make in time for me to finish so I had decided to go with having a keeper and a player. I completed the project only through trial and error and had to spend a decent amount of time tuning the Keeper's state machine to make it work how I'd like. The physics of the ball was also a complicated thing for me to achieve how I'd have liked so I ended up locking the ball's position to the player when the player "picks up" the ball and then applying a force on the ball based on where the player is heading.

Features Included

- Finite State Machine
- Steering behaviors
    - Agent movement, Seek, Flee, Arrive
- A* Algorithm
- Pathfinder Editor (Special Feature)
    - With this editor, you can plot a path anywhere and make blockages and set the start/end points for the algorithm and then view a rendering of the path.

Known Bugs

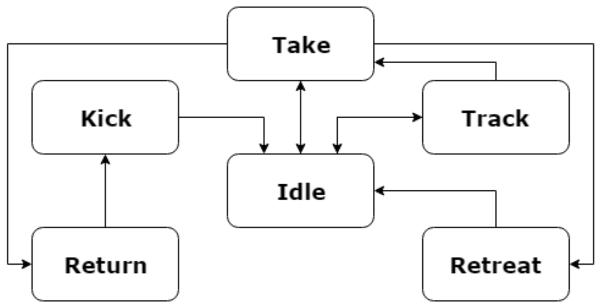
- A* Algorithm
    - I'm not exactly sure whether this is a bug or just the algorithm working as it should but when plotting some paths you'll see that it makes some sort of pyramid like movement path which is a bit odd.

Keeper Finite State Machine

Our keeper has six different states that it switches between:

- Idle: The keeper will return to its original goal position and defend
- Take: The keeper will retrieve the ball if it is in its vicinity and not in the player's hands
- Track: The keeper will stay at the goal but track where the ball is headed
- Retreat: The keeper will return to the goals to defend if it has attempted to collect the ball in the field unsuccessfully
- Return: The keeper will return to the goal with the ball in hand
- Kick: The keeper will kick the ball back into the field

Diagram of the Keeper State Machine



FSM v. Rule Based

I chose to go with using a Finite State Machine instead of creating a Rule Based system, in my solution. The reason why I chose to use a FSM was because it seemed to be a cleaner way of managing all the different states that an AI would have instead of having multiple if-statements in the system. Having an FSM means having a simple variable that holds the current state and creating an enumerator that contains all the states.

A* algorithm

My solution does not make a great use of the A* algorithm as I would have liked since I had a bit of trouble implementing this and making it work properly (I didn't manage to implement this until towards the due date). My solution does contain a "pathfinder editor" that allows you to create a path and then change starting/ending points and viewing the path that the algorithm had chosen.

Sources

- A* algorithm sources
  - https://en.wikipedia.org/wiki/A*_search_algorithm
  - http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
  - https://brilliant.org/wiki/a-star-search/
- Custom Unity Event System
  - https://unity3d.com/learn/tutorials/topics/scripting/events-creating-simple-messaging-system
- Learning about steering behaviors
  - https://www.red3d.com/cwr/steer/gdc99/