CMP304 Coursework
Project Report (50%)

Comparing a Finite State Machine and a Rule Based System as 'Drone Workers'

by: Austin A Flynn 1700403

1. Introduction

The basis for the project was an article by Fernando Bevilacqua [1] found whilst doing research for the Finite State Machine blog. The article used the 'brain' of an ant as an example, which inspired the creation of a drone-like artificial intelligence which would collect items placed in an environment. The most obvious AI choices for this project were Finite State Machine, from here on referred to as FSM, and Rule Based Systems, from here on referred to as RBS.

The overall goal was to test which of these two systems would run the most consistently and efficiently with similar code bases and tasks to achieve.

2. Methodology

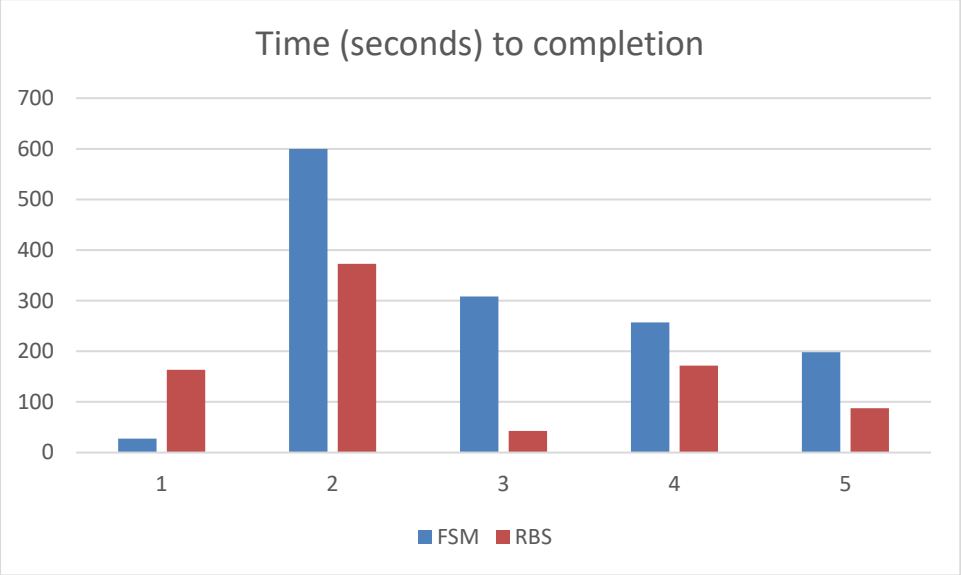
The first step was to decide on which of the potential systems to compare. The goal was not to create a learning AI, so concepts such as genetic algorithms and neural networks were removed from the equation. The deciding step was that of compatibility, as it would only be a fair test if both of the methods being used had similar, if not identical code apart from the parts directly relating to the method in question. The FSM and RBS were chosen for this reason, as the same functions that the FSM called based on the current state of the game could be used by the RBS, with the functions being called based on the condition of several Booleans as opposed to set states.

Before the beginning of this module the author had never used Unity before, or coded in C#, and as such a basis was found for the project to allow for a smoother approach vector and fewer issues caused by a lack of knowledge specific to the platform. The article in question was written by Lance Talbert [2], and covers the basics of AI in Unity, as well as important features such as collision detection using the engine. Despite this foundation the article has little on actual AI, however the 'Looker' components used by the AI actors are almost unchanged from this version, with only a change from being a collider to a trigger, and minor data passing upgrades. Using Unity version 2019.2.0f1 two identical arenas were created, both comprising of four key components in the same layout.

The first of these components is the floor, which acts as a navigable area for the AI, as they are only given permission to traverse these areas. Next is the 'Pickups', which are three green cylinders in each arena which the AI has to collect and return to the third component, the 'Dropzone'. The Dropzone is a trigger component which the AI must return to after collecting a Pickup, vaguely simulating an ant carrying food back to its nest before foraging for more. The final key component of each arena is the AI object itself, each with their corresponding 'Looker' component which acts as the eyes of the AI. The way the AI operates within these arenas is simple, they wander randomly until they 'see' a Pickup overlap with their Looker, at which point they make a beeline for the Pickup. Once the main AI body touches the Pickup it disappears, and the AI returns to the Dropzone before wandering away again. Once all three Pickups have been collected and the AI returns to the Dropzone, the time it took to complete the task is output to the debug menu for later use.

To ensure results that were as accurate as possible, the program was run five times, with the completion time being recorded each time. The time it takes for each of the different AI to have a full run-through of the Update function, essentially how efficient each system is, was also recorded.

3. Results



The first chart here shows the basic readout of the raw data, with the RBS consistently finishing its task faster than the FSM after the first run. The data is somewhat skewed however, as the FSM never completed the second run and was given a time of 600 seconds/10 minutes in an attempt to compensate for the glitch which caused it, which wasn't replicable as far as could be ascertained.

This information can be refined somewhat to make it more useful, as finding the Standard Deviation of each of these sample groups will display how consistent each of the two AI were in their ability to perform. For the FSM, the Standard Deviation was 208.6359 seconds, or almost 3½ minutes, meaning that its performance was erratic at best. The RBS was much more consistent at 126.6755 seconds, still having a notable discrepancy of around 2 minutes, but still more reliably average than the FSM.

Perhaps the most readily applicable result was that from the efficiency test. As both of the AI were running similar code outside of their AI specific functions, a comparison of the speed at which the computer can process a single pass through the 'brain' of the AI should give an indication of which better suited the project as a whole.

The faster of the two systems was the RBS, with 0.016668 seconds passing between each 'thought'. This is however only marginally faster than the 0.016743 seconds it took for the FSM to process each update, and as such a definitive victor for this specific project in this regard cannot be crowned.

4. Conclusion

To conclude, while the results show these two systems to be quite similar on a project of this scale, there are many questions which could do with more definitive answers. It is the author's personal belief that if the idea of an ant simulation were to be taken to the extreme, the slight erratic nature of a Rule Based System would make it feel more natural and reactive, and therefore the superior choice. However the Finite State Machine would much better suit a larger project, as the ability to neatly categorize each of the possibly outcomes from a scenario allow for a greater level of control both in regards to how a scenario will play out, and in managing resources. Ultimately, while both of the systems tested performed well at this scale, with larger projects selecting an appropriate AI could make or break a project, and this module has granted some insight into the selection process.

5. References

[1] Fernando Bevilacqua (2013) Finite-State Machines: Theory and Implementation [online] TutPlus, available from https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867

[2] Lance Talbert (2018) Creating a Simple AI with Unity and C# [online] Redgate Hub, available from https://www.red-gate.com/simple-talk/dotnet/c-programming/creating-a-simple-ai-with-unity-and-c/

[3] Unity Documentation (Unknown) Version 5.4 [online] Unity available from https://docs.unity3d.com/540/Documentation

[4] Unity Forums (2014) [online] available from https://forum.unity.com/threads/how-to-get-the-position-of-a-gameobject-through-the-collider.245130/

[5] Unity Answers (2011) [online] available from https://answers.unity.com/questions/140798/how-to-set-variable-value-of-a-different-script.html