The agents also store information from their previous experiences, which is used for reinforcement learning. This includes adding agents, updating Cluster object that contains information and actions that a cluster would take. This includes adding agents, updating with various feature vectors to help shape behaviors. Finally, a visual representation of each tested model is viewable, showing the position and flight path of each agent. Other communication information is able to be displayed as well.

Methodology

One of the important features used is the agent image vector. The image vector is a three-by-three array that captures the states of the area around the agent position and is used as part of the input vector for the neural network for reinforcement learning. The agents also store information such as its vector and position as well as the closest agent's vector and imposition. The combination of features provides a solid base for modifications and improvements.

Reinforcement Learning

Reinforcement learning uses rewards to determine the optimal course of actions in a scenario. Whenever the agent is determined to have made a correct action, a reward is given to the agent. At the end of simulation, each agent will have its reward added to a total, which determines if the cumulative action of the agents is efficient or not. The goal of reinforcement learning is to gain the highest cumulative reward. Reinforcement learning is a perfect fit with decentralized clusters because it is able to reward agent for making optimal decisions whether or not they are in a cluster.

Markov Decision Process generates the proper rewards distribution for reinforcement learning. A Markov decision process is a stochastic learning process that uses four inputs: State, Actions, Probabilities, and Rewards. In this case, the agents and clusters take in the state of the environment, and based on the probabilities, determines rewards based on agent actions. Markov decision processes can help reinforce learning by calculating the most efficient actions within an environment and then creating a reward structure based on agent states and actions.

The clusters are assigned based on distances between agents. After the initial setup, the simulation will keep iterating. In each iteration, the fire will advance based on assigned patterns and probabilities. The agents will simultaneously move. Over time, the agents actions are based on the MDP, heuristic, and action sets available. As the agents move around, the flight paths recorded and displayed as colored lines based on the cluster each agent is in.

Analysis and Results

Once training is complete, the model is packaged and stored. In order to test the simulations, the packaged model is called and deployed in a test environment. The agents and fires are deployed randomly at the start of the simulation.

As the fire progresses, the agents will continue to monitor the spread of the fire. In this case, the agents will begin to look for the boundaries of the fire and start encircling it.

Summary/Conclusions

A simple event-based simulator was developed and modified in order to test and verify multi-agent clustering technique. The simulator was extended to track individual agents, clusters, and other metrics that would influence effective operation.

The features are modular, allowing for future modification and development based on user needs. Further, the simulator is lightweight and can run on different platforms without needing large changes.

Acknowledgements

I would like to thank Dr. Jonathan Ponniah for his guidance and advice during the course of this Project.

I would also like to thank Dr. Ravi Haksar for his previous work with multi-agent systems.