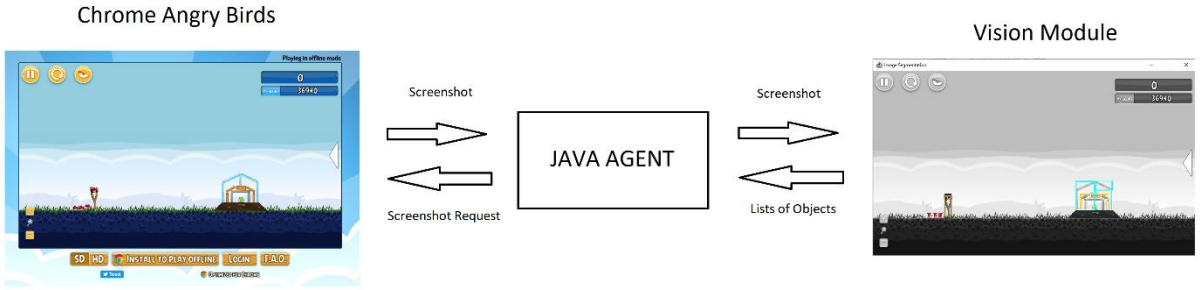
Rokos Award Internship

Developing a Reinforcement Learning Agent to play Angry Birds

This summer, I had the opportunity to help in the development of a reinforcement learning agent that would be designed to play Angry Birds. Reinforcement learning is a subset of machine learning, where agents are designed to interact with an unknown environment and improve on their behaviours based on the quality of their previous actions. Essentially, the agent will decide on an action based on the state it is currently in, take that action and have it rewarded based on a previously determined algorithm, and then use that reward to update its decision-making process should that same state be reached again.

Recording Agent

The first part of my internship was to design an agent that would record a human playthrough of an Angry Birds level. The aim was to record each shot taken by identifying the slingshot in the game window, and then finding the launch position of the bird relative to that slingshot. Rather than starting from scratch, I was provided with a java agent that would play through levels with a very basic shooting strategy. This agent connected to a chrome version of Angry Birds with a chrome extension, and would send actions (clicks and drags) to the game and in turn would receive screenshots of the game state. The agent had a vision module that would be able to identify significant objects in Angry Birds levels (birds, sling, pigs, blocks, etc.).



Whilst this provided a helpful starting point, to record human player actions, it would be necessary for the agent to record clicks and drags made by the user. I was able to modify the chrome extension so that any such action would be recorded, and then sent to the java agent. One of the other issues with the provided agent was that screenshots were only taken at regular intervals determined by how quickly the vision module could process the incoming images. However, this could result in actions being taken by the player in the gaps between screenshots. It was especially important that the recording agent had an accurate portrait of the game state when actions were being taken so that the recorded inputs were as accurate as possible. To ensure this, I modified the chrome extension so as to send screenshots of the game whenever a click was detected.

To store the recorded information, I used a MongoDB database. For each shot, the launch position of the bird in relation to the slingshot was recorded, as well as the game state (e.g. the number of birds/pigs remaining) after the shot had been taken.

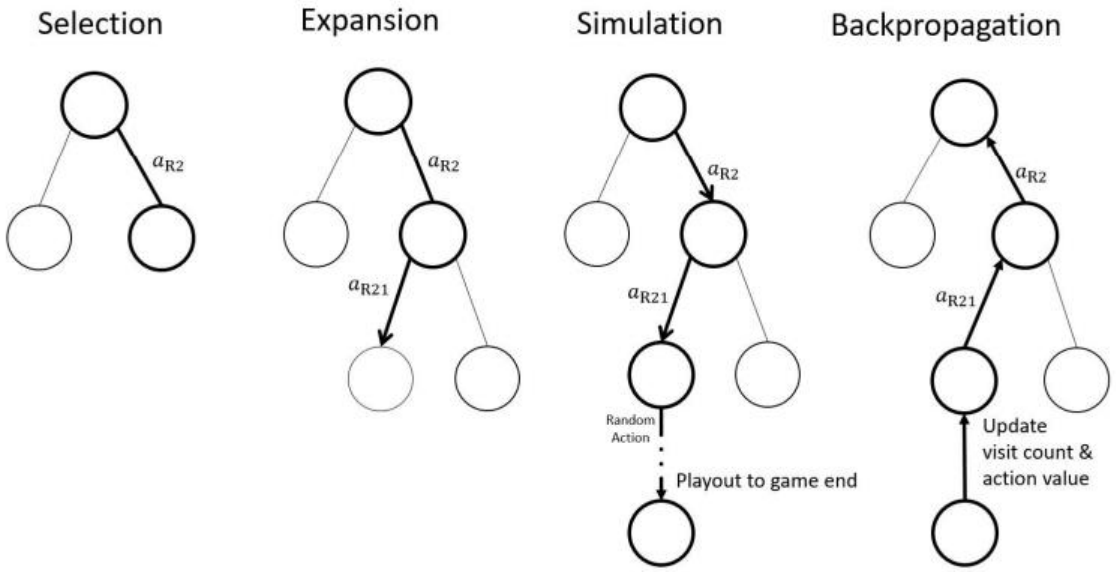
Reinforcement Learning Agent

The second part of my internship was to develop a basic reinforcement learning (RL) agent that would be able to play simply designed Angry Birds levels. RL is a method of machine learning where an agent is provided with no initial information about its environment (such as training data) but will be rewarded on the actions it takes depending on the quality of those actions. As such, an RL agent must balance both exploitation (choosing actions already known to give a high reward) and exploration (choosing actions that may have higher rewards, but whose value has not been precisely determined yet).

Monte Carlo Tree Search

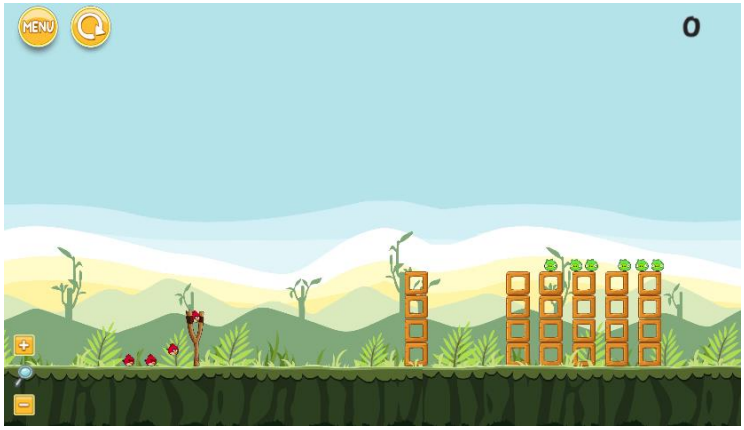
Due to the nature of Angry Birds, it would require a large time and computing cost to explore all possible states, so it is practical to use a heuristic algorithm. I chose to design a Monte Carlo Tree Search (MCTS) algorithm to learn to play Angry Birds. Rather than exploring every state first, an MCTS algorithm estimates the quality of certain states through repeated random play from that given state. An MCTS consists of four stages:

1. **Selection:** A path starting from the initial node is taken based on maximising a chosen value. This could be based entirely on the current expected reward of each action, or could also account for how certain the agent is on each reward estimation by also factoring in the number of times that action has been taken.
2. **Expansion:** Once a node is reached with insufficient collected data to properly assess future action values (usually because the node has not been encountered a sufficient number of times), a random action is taken, and a new node is created.
3. **Simulation:** The agent will play the game until completion with random actions being chosen. With multiple random playthroughs from the same node, the average of the rewards obtained should approach a good approximation of the actual potential reward of that node.
4. **Backpropagation:** On completion of the simulation, the reward obtained is used to update the expected rewards of the actions taken.



Science Birds

One of the issues I faced with just using a web browser version of Angry Birds was length of time required for the MCTS agent to learn the game and improve. As the agent must play each state it encounters potentially up to around a hundred times, and at normal speed, the learning process is quite slow. Science Birds is a unity clone of Angry Birds that has a number of benefits over the browser version of the game when implementing an RL agent. Primarily, Science Birds allows for the speed of gameplay to be increased (up to around 50 times normal speed). This allows for much faster playthroughs of each level which vastly increases the speed at which the agent can learn. Additionally, it is far easier to create new levels for the agent to learn on. The AI Birds level generation competition provides a baseline generator that can be modified to produce randomly generated simple levels. To simplify the game for the RL agent, only one block type would be used in level construction, and only red birds would be used. By only using one block type, the state recorded could simply be the location of the centre of each block and pig in the level, and the number of birds remaining. This also reduced the accuracy required for the vision module as any blocks identified would be known to be that one block type.



Example of Generated Level

MCTS Agent Design

One of the challenges with designing the MCTS for the Angry Birds agent is the slight randomness in the Angry Birds gameplay. Actions do not necessarily lead to exactly the same results, and so when considering the tree search, this means that actions do not necessarily always lead to the same future node. Rather than saving future expected rewards on nodes, and then assessing actions based on the nodes they lead to, it was much more practical to save expected rewards on the actions themselves. The agent performed the four stages of the MCTS as below:

1. **Selection:** A path starting from the initial node is taken based on maximising
$$SEL(a) = Q(a) + \frac{C}{1 + N(a)}$$
where Q(a) and N(a) were the expected reward of the action 'a' and number of times it had been taken respectively.
2. **Expansion and Simulation:** Once a node is reached that had not been reached enough times, actions would be taken randomly, although ensuring each action would be tried once before any actions were retried.
3. **Backpropagation:** On completion of the simulation, the reward obtained is used to update the expected rewards of the actions taken.

To limit the number of possible actions, the agent was limited to taking shots between angles of 10 and 120 degrees in increments of 10 degrees, and shots would be taken either with full power, or 60% power.

Overall, I felt that over the course of my internship, I learnt a lot about the basics of reinforcement learning. As I go into my third and fourth year of engineering, this foundation should be helpful if I choose to do any machine learning courses/projects. I also had the opportunity to improve my skills at coding in Java and javascript. I would like to thank Clarissa Costen and Bruno Lacerda for supervising my internship, as well as Nick Hawes for helping organise it.