IGGI GD2 Lab Exercise: Rolling Horizon Evolution

Introduction

The purpose of this lab is to gain experience of using the Rolling Horizon Evolution agent described in the lectures, and given in the codebase (see below).

The lab script should be doable within a couple of hours.

In the lab you will:

- Check out the latest software from the github repo
- Run a RHEA agent to player Asteroids and CaveSwing
- Play with the CaveSwing game to see where the default RHEA agents succeeds and fails
- Set up a search space for a RHEA agent for CaveSwing, and use the NTBEA to optimise the RHEA agent.

Downloading and Running the Software (5 mins)

Download the software from the following Github:
https://github.com/ljialin/SimpleAsteroids

(we suggest you download the .zip file)

Open the project in your favourite Java IDE (we recommend Intellij Idea).

Asteroids and CaveSwing (30 mins)

Start by running the file: **asteroids.RunAsteroidsDemo** with its default settings.

You should see the agent play reasonably well (let's define that as achieving average scores in excess of 8,000).

In the main method you will see the lines:

```
Game.seqLength = 100;
Game.nEvals = 20;
```

Experiment with different numbers of sequence evaluations (also referred to as rollouts), and with different sequence lengths. The product of these values gives the number of game ticks used per decision. As you experiment with the numbers, there is no need to keep this product constant. For example, while keeping the same sequence length, how small can the nEvals be before performance steeply declines? (e.g. average score below 5,000).

As another experiment, try modifying the agent's parameters such as whether it uses a ShiftBuffer or not. *(note: the Asteroids code is a bit old now, and the way the agent params are set up is not as clean as it could be. See getEvoAgent() in asteroids.Game).*

Repeat the experiment for **CaveSwing**, by running **caveswing.test.EvoAgentVisTest.**

The default agent should play ok with these settings:

```
CaveSwingParams params = new CaveSwingParams();
params.maxTicks = 2000;

// todo: how does changing the parameter settings affect AI agent performance?
// todo: Can you settings that make it really tough for the AI?
params.gravity.y = 0.2;
params.gravity.x = -0.0;
params.hooke = 0.01;
```

Defeating the Default Agent (20 mins)

Continuing on from the above, experiment with the game parameters (by setting them as shown above, but to different values and also tuning additional parameters such as nAnchors). Can you find versions of the game playable that you can play, but for which the default AI fails?

Experimenting with long reward horizons.

Now set all the score related parameters to zero, apart from successBonus (leave this at its default value of 1000). Does the agent now fail where it previously succeeded? Try to find instances where the default rolling horizon player can still outperform the random player (check in *agents.dummy* for *RandomAgent*), despite the delayed reward.

Agent Optimisation (30 mins)

Now optimise an agent to play CaveSwing.
The code to do this has already been written, and can be executed using:

hyperopt.TuneCaveSwingAgent

Run this and observe the outcome of the runs: search for "Solution" in the std out. Do the same agent settings tend to be found each time?

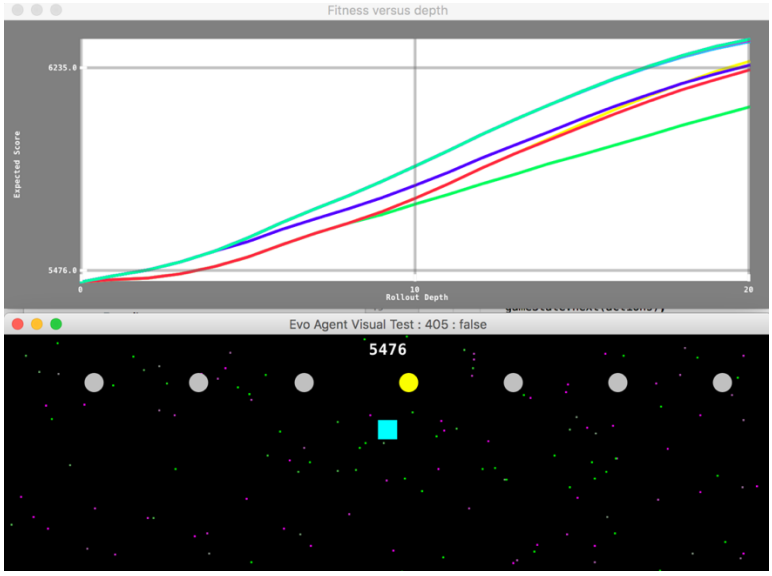
Questions

- What are the optimal values for sequence length?
- What is the impact of turning off the shift buffer?

The answers you get from the model have to be interpreted with caution. Why?

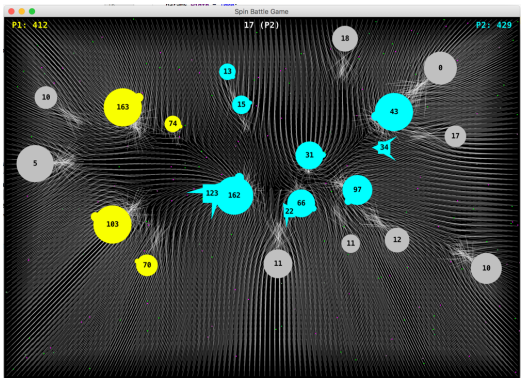
For the best solution you can find, test it with and without the shift buffer. What impact on performance do you observe now?

Run again **caveswing.test.EvoAgentVisTest** with the parameters suggested by NTBEA, the output should look something like this:



Rolling Horizon Evolution in Planet Wars

You'll find some Planet Wars code in the **spinbattle** package. Try competing against heuristic and rolling horizon agents: see code in the spinbattle.test and spinbattle.actuator packages. To play check the classes HumanSlingVersusEvoAgent and HumanCatapultVersusEvoAgent. Play with various game params such as the gravitionalFieldConstant and observe the effects on the game play. **How does changing the SimpleEvoAgent parameters affect it's play strength?**



Understanding optimality of SimpleEvoAgent

Refer to the Rolling Horizon Evolution 2019 lecture notes (slides 10 – 14 for this exercise)

Copy the code here into a Kotlin file and run it to simulate the OneMax version of the decision making problem faced by SimpleEvoAgent playing planet wars (i.e. use appropriate values for nActions, mutationRate and sequence length). Be careful, the mutation code needs to be changed to allow mutation between all possible values, something like Random.nextInt(nActions) will do. How optimal are the agents in OneMax terms?

```
package test
// or whatever package name you want

import java.util.*
import kotlin.random.Random

fun main() {
    val oneMax = SimpleOneMax(mutationRate = 0.2)
    oneMax.run(100)
}

class SimpleOneMax(val mutationRate: Double = 0.05, val n: Int = 20) {
    var current = IntArray(n) { i -> Random.nextInt(2) }
    fun run(nEvals: Int) {
        for (i in 0 until nEvals) {
            val mut = mutate(current)
            // note: would normally test ">="
            if (mut.sum() > current.sum()) {
                current = mut
                println("$i\t ${current.sum()}\t ${Arrays.toString(current)}")
                if (mut.sum() == n) {
                    println("Optimum after $i iterations")
                    return
                }
            }
        }
    }

    fun mutate(x: IntArray): IntArray {
        val mut = IntArray(n)
        for (i in 0 until n) mut[i] =
                if (Random.nextDouble() < mutationRate) 1 - x[i] else x[i]
        return mut
```

```
    }
}
```

Summary

We have now covered how to use the codebase to optimise both the game design, and an agent to play the game. These are useful tools that you can use to optimise many different systems.