QWOP Bot Progress Report

Written by Matthew Oros, Sonny Smith, and Michael Terekhov
Baldwin Wallace University
275 Eastland Road
Berea, Ohio 44017

Abstract

Abstract Here

Content Here

Problem Description

QWOP is a simple 2-dimensional game with the goal being to make the main character run to the right as far as possible without falling. The challenge of the game is its physics simulation. The character is controlled using four keyboard keys: Q, W, O, and P. Q and W control the thighs [TODO: Check if this is the right way around] and O and P control the knees by either rotating the joints one way or the other. As soon as an upper-body limb contacts the flat ground, the game is over and is restarted. The score of the game is based on the distance traveled by the character.

The game is intentionally difficult as the physics simulation makes movements hard to predict and require a quick reaction time as well as understanding of how the combinations of key pressed translate to a desired movement on screen.

Proposed Implementation

The proposal of the simulation is an agent that is able to learn on its own to control the character to play the game of QWOP. Two main implementations that will be considered and attempted will be both based on neural networks [TODO: Make sure we are still using neural networks for Deep Q]. The first method is a genetic algorithm approach to learning while the second approach will be a deep Q method of learning.

Consideration of Backpropagation

A traditional method training a neural network is to use a method called backpropagation. Backpropagation is done by providing the network with input data and labelled output data where the term labelled refers to the desired output of the given input. The input is fed forward through the network in its current state and its output is compared to the labelled output. The weights are then adjusted from back to front based on how incorrect the network's output is.

The issue with this approach for this domain of problem is that there is not a viable method of obtaining such labeled data in this problem. The game of QWOP is difficult for humans and such the goal of the simulation is for the learning process to generate a solution from its fitness feedback and environment rather than by example from human playthrough. Thus the two methods that were chosen incorporate some form of random exploration of the state space to converge on a desired solution.

[TODO: Review this paragraph]The deep Q-learning approach that we use in our project differs from a common approach to this problem since we cannot use an example as a reference. Our approach is model-free off-policy learning. It also uses an off-policy learning strategy that allows the agent to learn from any data regardless of the behavior policy. In our approach, the neural network is modified based on the previous expriances. The previous experiances are captured in the memory which are broken down into several parts. The parts are reward memory, action memory, state memory, new state memory, and terminal memory. Each memory in broken down into batches, once the batches hare filled up the neural network is updated based on the rewards that were recieved given the states and the states after them. The memory is broken down into batches so the neural network can be modified more frequently.

Current Progress

In order to create a suitable learning environment for the agent, the game was reimplemented. The original game is browser-based and would require interfacing code to grab relavent data and to simulate keypresses. The logistics of this are not relevent to the desired outcome of the project and thus the approach of reimplementation of the game was chosen.

The programming language that was chosen was Python. Python was chosen due to its simplicity, flexibility, and plentiful collection of libraries, especially related to machine learning. For the simulation itself the graphics library chosen is Raylib. Raylib is a native C library but has various bindings to other languages such as Python. It was chosen due to its simplicity where it is not a graphics engine but rather abstracts OpenGl drawing calls to functions. This helps to decouple the simulation from the graphics. The physics library chosen is PyMunk. It is a simple 2D physics library where physics bodies are created with certain shapes and properties and added to a simulation space. PyMunk was also chosen for its simplicity and flexibility.

The initial progress on the project started with a basic simulation of physics bodies where the joints that connect them were actuated by a key press. This basic simulation later evolved to created legs and then hips, and then the upper body. Various tweaks were needed to produce a correct feeling simulation based on the original game. In order to accomplish this, the ability to directly control the character with the keyboard was added even though it would not be used in the final result as rather the simulated agent would be controlling the character.

Once the progress on the physically-simulated character was acceptable, work begun on creating a neural network. The neural network was [TODO: originally?] created without use of any libraries. A neural network consists of input neurons that connect to hidden neurons which then connect to output neurons. The strength of the connections between neurons are called weights. Each neuron accepts a single or multiple floating point values which then get summed and normalized using an activation function. The activation function chosen was the sigmoid function as it is very common choice.

In order to utilize a neural network, the inputs that are fed to the network must be determined. These inputs would denote which aspects of the environment the agent would be informed about. If too little information is forwarded to the network, then it may not be able to converge on an acceptable solution. If too much data is sent, the network may also have difficulty converging with having to weight the numerous inputs. The current implementation sends the global x and y positions of each limb to the network. We hypothesize that this would give the network enough information to determine how it should move its limbs when in a certain physical configuration.

Future Considerations

Some future considerations for the inputs of the neural network would be to send information regarding velocity of the character. This might better inform the network. However, it may not be necessary to input velocity for each individual limb and so an average velocity or velocity of just the center of mass might be sufficient. However, this could only be determined through experimentation. Another idea might be to have a memory neuron which would be an output neuron whos output is connected to an input neuron. This could potentially allow for the network to retain some type of memory about its computational state. This may be a helpful addition as the problem being solved is highly temporal and so the ability for the network to retain some kind of information about the previous frame to the next could prove to be beneficial.