# QWOP-timization II: Can policies be inferred from many examples?

Matt Sheen
Cornell University

**1 minute summary vid link**

QWOP is a "running simulator" video game which grew virally infamous for its difficulty. We set about making a robot which plays the game. This is difficult since the game has black-box dynamics with unobservable states, among other issues. We found a method to generate long sequences of open loop actions by doing a kind of tree search on the controls on a simulated version of the game. When replayed on the 'real game,' these trajectories fall apart after a couple steps. Armed with numerous trajectories from our tree method, we are attempting to infer a control policy. However, it's possible that all our trajectories are sampled from different, incompatible policies. More broadly, it is possible to have an near-infinite number of successful examples of a task from which *nothing* can be learned. While trying to control QWOP, we want to address this broader question also, which we think will become very relevant as more people try to transition machine learning from object recognition to controls problems.

## I. Introduction

To the general public, the problem of walking coordination seems like it must be intuitive. We know that locomotion control is difficult, but this can be hard to demonstrate in a hands-on way. Several years ago, a "running simulator" game called QWOP exploded on the internet ( 30 million hits), becoming notorious for its difficulty despite seemingly simple controls. It received considerable media attention and spawned Guinness World Record competitions (speed to run 100m). We attempt to beat the world record time by finding a policy for controlling QWOP (or at least post a competitive time!). This game poses a few interesting problems including: locomotion controller design with black box dynamics, control optimization of a system which may only be forward simulated from a fixed initial condition, very low-dimensional control of a high-dimensional system, dealing with unusual types of noise (e.g the game's internal clock cycles), and dealing with numerous unobservable states inside the physics engine.

So far, we've found a method for producing arbitrarily-long sequences of running actions. However, these open-loop trajectories can't be, say, put on a QWOP-playing robot. Hence, the next steps is to find a policy.

## II. About QWOP

QWOP is a web browser-based (Adobe Flash) ragdoll physics game. The goal is to make a 2D cartoon athlete run 100m on a track (although most players won't get more than a few meters). The player uses the Q, W, O, and P keys on the keyboard to apply coupled torques on the legs. Q and W apply opposite torques to the thighs; O and P apply opposite torques to the calves. Arms and ankles are coupled to these actions.

## III. QWOP 'simulations'

We've made/used several versions of the game:

1. **Real thing**: The real game played in the web browser using the keyboard.
2. **Hacked game**: Our hacked version of the game which can receive external input at precise physics steps.
3. **Fast QWOP**: The game recreated in Java using the same physics engine and many extracted parameters. It runs 1000x faster.

In a sense, the real game is analogous to a real robot, with our hacked version being a high-

fidelity, noise-free simulation, and the recreated version being a lower-fidelity, but fast simulation.

## IV. Progress so far

*Note: Some of this was presented at the Ohio State Dynamic Walking.*

So far we've considered the 'trajectory optimization' portion of the problem: finding a sequence of actions for running. Closing the loop is the work coming up. Due to the black-box nature of the game, the hidden states, and the discrete timesteps & controls, we didn't try the more tradition approach of finding a periodic gait – likely periodicity would require too many steps. Instead, we look at the game as a search through the space of possible controls. This approach requires a lot of weeding down since there are close to $10^{1200}$ possible games in the time of a world-record-pace game ( 50s). We reduce this space greatly by parameterizing sequences by *delays between transitions* rather than the actions at every timestep. By watching videos of the pros, we further reduced it to a single order of actions which is parameterized by 4 integer delays per gait cycle. For reasonable delays, the search space is down to $10^{77}$, still too big to directly brute-force. Instead we build trees out of the control actions, and strategically explore these on "Fast QWOP." Note that this is different from *Rapidly Exploring Random Trees (RRTs)* in that our trees don't encode the runner's state, only control actions. The software we developed is visual and interactive, allowing us to inspect different regions of the tree and find trends for successful behavior, while pruning other areas. Likely the most important heuristic is: falling forwards is better than falling backwards. In QWOP, falling backwards is something that usually occurs over the course of many actions, whereas falling forwards is very recent error. A simplified version of the algorithm is to build a tree randomly for awhile, find the most promising failures, go back a few actions from the failure and keep randomly building again, and repeat. With this, we can find a random 100m trajectory in roughly 5-10 minutes.

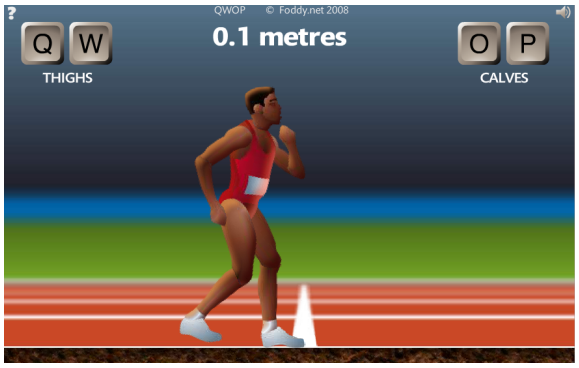


**Figure 1:** *QWOP screenshot.*

## V. Next steps and the big questions

These long trajectories don't work on the real game. Slight timing issues along with small differences between the simulator and the real game cause them to quickly fall apart. The goal is to infer a policy from our library of trajectories, or be able to conclude that it is impossible. A candidate method is to use a function approximator, like a neural network, to represent the policy. Another approach might be to split sections of actions into 'motion primitives' and string these together. But, it is possible that no method can infer a working policy from our trajectories. If each trajectory is sampled from a different, incompatible policy, then we cannot generalize. Imagine trying to swing a pendulum up, and we're given many trajectories. One swings the pendulum directly up from the left. Another to the right. A third pumps an extra swing before going up. A fourth pumps two times. And so on. You could imagine that if we are semi-randomly generating trajectories, like we're doing with QWOP, we might end up with trajectories like this. So, a broader question is: can we recognize cases in which trajectories can't be generalized, and can we find tricks to avoid or fix these cases. We think this is particularly relevant as more machine learning experts turn to controls problems now that classification is getting boring.