Evolving QWOP Gaits

Steven Ray
CSU Sacramento
stevenlray@gmail.com

V. Scott Gordon
CSU Sacramento
gordonvs@ecs.csus.edu

Laurent Vaucher
Google, Inc.
laurentvaucher@gmail.com

ABSTRACT
QWOP is a popular Flash game in which a human player controls a sprinter in a simulated 100-meter dash. The game is notoriously difficult owing to its ragdoll physics engine, and the simultaneous movements that must be carefully coordinated to achieve forward progress. While previous researchers have evolved gaits using simulations similar to QWOP, we describe a software interface that connects directly to QWOP itself, incorporating a genetic algorithm to evolve actual QWOP gaits. Since QWOP has no API, ours detects graphical screen elements and uses them to build a fitness function. Two variable-length encoding schemes, that codify sequences of QWOP control commands that loop to form gaits, are tested. We then compare the performance of SGA, Genitor, and a Cellular Genetic Algorithm on this task. Using only the end score as the basis for fitness, the cellular algorithm is consistently able to evolve a successful scooting strategy similar to one most humans employ. The results confirm that steady-state GAs are preferred when the task is sensitive to small input variations. Although the limited feedback does not yet produce performance competitive with QWOP champions, it is the first autonomous software evolution of successful QWOP gaits.

Categories and Subject Descriptors
I.2.6 [**Learning**]: Artificial Intelligence – *learning.*

Keywords
Genetic algorithms; games

1. INTRODUCTION
QWOP is a simple online game in which a human player controls a graphical representation of an Olympic sprinter. The player wins by successfully guiding the sprinter to the finish line. The only inputs are the keystrokes Q, W, O, and P, which control particular muscles in the sprinter's legs. For a game with only four inputs, the game has proven to be a difficult, unintuitive task for humans.

We attempt to evolve successful gaits using a genetic algorithm. We do this by building an API for QWOP, a genetic encoding for keystroke sequences, looping such sequences in real-time, and using the resulting score as the fitness function. We then test the efficacy of various evolutionary models on this application.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2662-9/14/07...$15.00.
http://dx.doi.org/10.1145/2576768.2598248

2. BACKGROUND
2.1 QWOP
QWOP is a popular computer game developed in Adobe Flash by Bennett Foddy, and is available for free on his site Foddy.net [3]. In QWOP, the player takes control of an Olympic sprinter running a 100-meter race. Upon reaching the 100-meter mark, the player wins the game. If the runner's head or one of his hands touches the ground at any point along the way, QWOP considers him fallen, and the game is over. The game quickly became notorious for its difficulty, despite what seems like a simple task. Playing QWOP well has been described as "a ballet of tiny corrections, any of which is likely to throw off the player's timing and may result in a fatal error" [11]. QWOP's difficulty is largely a function of two factors: the *control scheme* and the *physics engine*.

QWOP gets its name from the game's control scheme. To play a game of QWOP, the user controls the runner using only the Q, W, O, and P keys on the keyboard, each of which controls a specific muscle group in the runner's legs. Q and W move forward the runner's left and right thighs respectively. The O and P keys each map to his left and right calves. Achieving a realistic bipedal gait in QWOP is often soon abandoned by new players, who tend to resort to finding a repeatable pattern of inputs sufficient enough to scoot the sprinter to the finish line.

The QWOP website states: *"QWOP is a game where … you need to regain something many of us take for granted: the ability to walk."* Thus the game forces us to work out mechanically a task that most of us perform every day yet to which we do not devote much active thought. Users are precluded from relying on their own experience and knowledge of balancing and walking on two legs, because QWOP reroutes the motor skill of synchronized leg muscle manipulation to the user's fingers via the game controls. Players are also limited with regard to the amount and type of relevant sensory feedback available to tell how well they are doing and make necessary adjustments to their runner's gait, as the only available sensory feedback is the visual state of the runner.

The second factor that complicates QWOP is its *ragdoll physics engine*. Ragdoll physics is a method for procedurally animating characters based on a skeletal configuration of rigid bodies connected by joints and muscles in a simulated physical environment [6]. The physics system is an approximation but reasonably realistic simulation of a real runner's body with regard to physical laws like gravity, friction, and inertia. The player must constantly work not only to complete the race, but also to keep the runner from falling over with each step. Every slight movement of the runner's legs carries with it consequences of momentum and velocity compounded by the game world's gravity and friction. As a result, achieving QWOP's seemingly simple goal becomes extremely difficult, as the game is unforgiving to imprecise and even slightly poorly-timed movements.

2.2 Genetic Algorithm Models

Genetic algorithms (GA) are search algorithms based loosely on the principles of natural genetic evolution. They have been useful mainly for optimization, by applying simplified notions of selection, crossover, mutation, and survival of the fittest to an artificial population of encoded candidate solutions. Each member of the population is evaluated by a fitness function, and the GA iterates the genetic operations through generations until it reaches some terminating condition, such as a sufficiently optimized solution or a fixed number of generations.

The Simple Genetic Algorithm (SGA) is a *generational* model described by Holland [5], in which the entire population is replaced by a new population of individuals at each iteration.

A *Steady-state* GA does not follow a generational model, instead creating only a few new individuals at a time, which replace weaker individuals. In some versions, a few individuals are selected at random, using the strongest of those as parents, and the offspring replace the weakest only if they have higher fitness. The steady-state model has been shown to have advantages over the generational model in many problem spaces [12].

In a *Cellular* GA, each individual in a population selects a mate from those in its local proximity. This mechanic simulates isolation by distance within the population, and promotes niches of subpopulations that can improve diversity and help to prevent premature convergence [7]. Our version is based on a CGA described in a previous study [4].

2.3 Evolving Gaits

GAs have been explored for learning and optimizing gaits in hexapod [8], quadruped [2], and biped robots [13]. Biped robotic gait learning offers a slightly different problem domain from learning for robots with more than two legs, as walking with only two legs necessarily involves periods of time during which the robot is supported entirely by only one leg. Thus, bipedal gaits require more careful balance than gaits in which multiple legs are supporting the robot at any given moment. Most research on evolving bipedal gaits uses either physical or simulated robots with relatively high degrees of freedom compared to QWOP's runner, which has only four degrees of freedom.

Brodman and Volstad used reinforcement learning to achieve bipedal gaits in a stick-figure simulation of QWOP. Their program took a multi-dimensional feedback approach that moved the runner and adjusted his various limbs based on their horizontal, vertical and angular velocities, the body's calculated center of mass, and whether or not at least one foot was touching the ground. Their learning approach was able to achieve both a stable "shuffle-like" gait and a less stable but faster more standard gait, depending on the reinforcement learning model used [1].

3. Qwopper INTERFACE

In March of 2011, Vaucher implemented and described a Java-based controller program that could play QWOP through a web browser [10]. Unlike the work of Brodman and Volstad, which uses a model of QWOP, Vaucher's program attempts to play QWOP itself. Vaucher's QWOP controller program, named "Qwopper", uses the java.awt.Robot class to interact with the screen and play QWOP. It captures screen images and compares them with screenshots and known patterns and colors found in the QWOP interface to locate the game window. Once the game window is located, Qwopper gives it focus by sending a mouse event to the game. Once the game window has focus, QWOP is played entirely using keyboard commands sent by Qwopper. Qwopper monitors the current game score, consisting of the current distance traveled by the runner, by periodically capturing a rectangular image of the location on the game screen where the score is displayed. Qwopper then applies a thresholding function that converts the image to black and white, tokenizes each character in the image, and then compares against a set of reference images of each character to parse the score for each runner. If the runner falls over, Qwopper sends a spacebar command to restart the game [10].

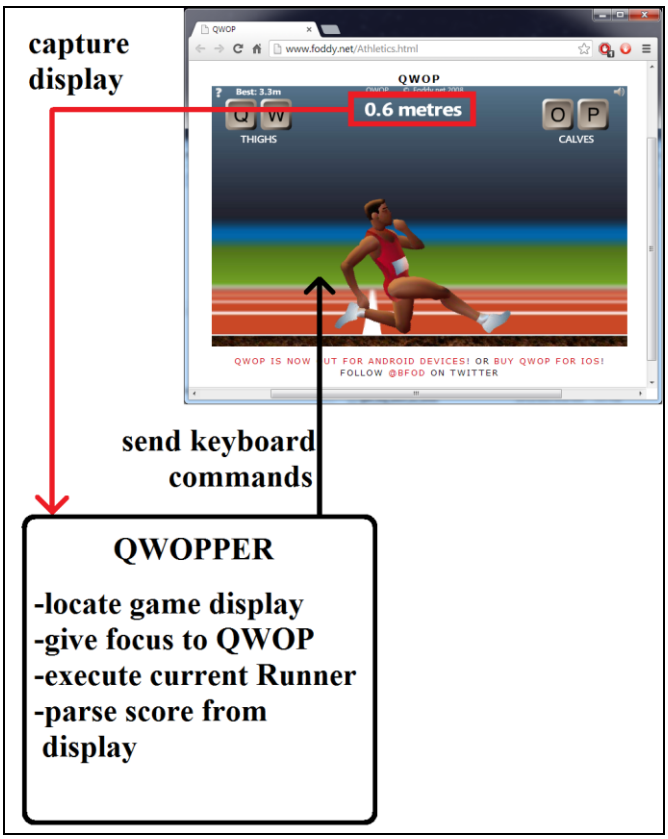

Figure 1. QWOPPER system interface

We use Vaucher's Qwopper software to evaluate the fitness of each runner in a GA population, by decoding each individual in the population into a sequence of runner keystrokes, and then playing QWOP according to each runner's genetic sequence, which is looped until the runner crashes, reaches the time limit, or wins the game. Generations of runners evaluated by Qwopper are evolved using a genetic algorithm and logged for analysis.

4. QWOP GAIT EVOLUTION

4.1 Genetic Encoding

Two different encodings were explored. The first encoding was described by Vaucher [10]. A second encoding was developed for this project to address disruptive effects of crossover and mutation discovered to be particular to QWOP gaits. The two methods are dubbed "encoding 1" and "encoding 2", respectively.

Vaucher's original encoding (*Encoding 1*) encodes a sequence of QWOP inputs as a string of characters. Each character represents either a key press, a key release, or a delay. An example individual is shown in Figure 2:

```
QO+qPW+wpo+QPW+wO+qp+P+Q+++
qp+QPW+wo+qp+POQ+q+W+Qp+qwo
```

Figure 2. Example runner using Encoding 1

A capital letter represents pressing that key on the keyboard, a lowercase letter represents a key release, and the '**+**' represents a delay in which the current state of inputs is maintained for a set length of time. The individual in the above figure translates to "Press Q and O, hold them for 150ms, release Q, press P and W, hold for 150ms, release W, P and O, wait…" and so on. An advantage of Encoding 1 is that it is easy to read and understand the sequence of inputs represented by the solution candidate.

Encoding 1 proved problematic when crossover and mutation were introduced, because the control state at any given point on an individual using this encoding is highly dependent on its context within the sequence. Crossover and mutation operations on individuals defined with Encoding 1 resulted in producing "non-coding DNA" or redundant commands such as consecutive capitals or lower-case letters. Encoding 1 was designed such that each letter indicates a change in the current input state. Each crossover and mutation operation had a high probability of rendering one or more characters in the chromosome redundant, which often dramatically altered the input sequence.

The second genetic encoding explored (Encoding 2) encodes input sequences using a 16-character alphabet, each letter of which represents one of the possible input combinations in QWOP. In order to describe Encoding 2, we must first describe the alphabet it uses, shown in Figure 3.

QWOP uses only four buttons, so there are 16 possible input combinations that comprise QWOP's input space. Each letter in Encoding 2 maps to one of these 16 input combinations. Unlike Encoding 1, input sequences built with this alphabet assume a small delay between each letter's execution, since each letter defines a distinct input state for all four control keys. Encoding 2 does not suffer from the context-dependence of letters in Encoding 1, since each letter here represents a distinct input state for *all four* keys. As seen in Figure 3, a key value of 0 indicates a key release command, and the key value 1 indicates a command to press the key. Redundant sequential press commands for a given key will simply continue to hold the pressed key continuously, while redundant release commands will just continue to refrain from pressing that key.

An example individual defined using Encoding 2 is shown in Figure 4. Each letter in the sequence is separated by an implied wait period of 150ms, like Encoding 1, during which the current input state is maintained, meaning that any keys pressed according to the current letter are held until a letter is encountered that releases that key. The individual in Figure 4 translates to "press Q and O, hold for 150ms, press P and release O while continuing to hold Q, wait for 150ms, release Q and O and press W, hold for 150ms, release W and press O, wait…" and so on.

Alphabet	Q	W	O	P
P	0	0	0	0
D	0	0	0	1
C	0	0	1	0
J	0	0	1	1
B	0	1	0	0
I	0	1	0	1
H	0	1	1	0
N	0	1	1	1
A	1	0	0	0
G	1	0	0	1
F	1	0	1	0
M	1	0	1	1
E	1	1	0	0
L	1	1	0	1
K	1	1	1	0
O	1	1	1	1

Figure 3. Encoding 2 Alphabet with Input States

```
FGBCHFELMIEFNGJCLHLEMCLJKJLNEKGHDGJDAJLE
```

Figure 4. Example runner using Encoding 2

The *mutation operator* used throughout the project, for both genetic encodings, was to randomly select and alter a single character in every child runner.

We tested two different crossover strategies: *single-point "cut-and-splice"* crossover, and standard *two-point* crossover. The "cut-and-splice" strategy selects a different crossover point for each parent, allowing for varying chromosomal lengths to be produced in offspring. The motivation for this approach was that the appropriate length of a good input loop in QWOP is not at all obvious, and we anticipated that this strategy could increase the search power of the algorithm with regard to the length of looped input sequences. By contrast, the two-point crossover strategy selects two crossover points for each parent at the same location on both parent chromosomes, thus maintaining chromosomal length in children and throughout all generations.

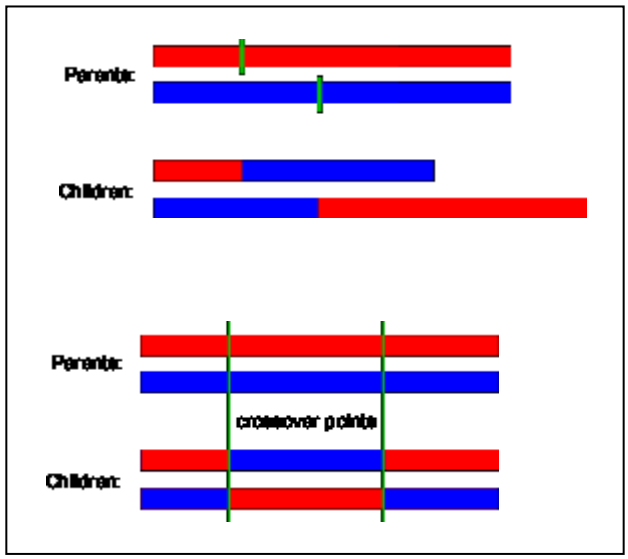

Figure 5. One-point "Cut-and-splice" crossover (above), and standard two-point crossover (below).

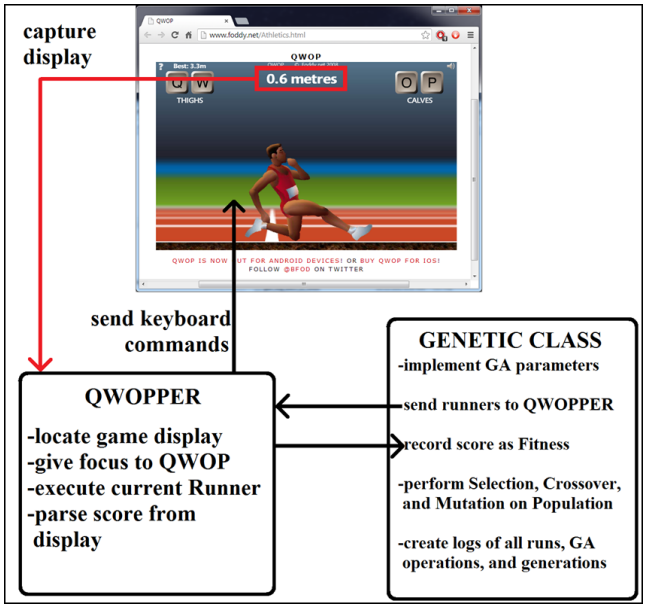

Figure 6. Genetic Algorithm real-time QWOP interface

4.2 Genetic Algorithm

We tested two different implementations of the evolutionary model: a standard generational model (SGA, described earlier) and a cellular model.

Our cellular GA implementation uses a local selection strategy that restricts the pool of potential mates for a given runner to those nearest to it in the population. Local selection was realized using a two-dimensional borderless "wrap-around" grid structure so that each individual has the same number of neighbors. Local selection was introduced to prevent, or at least slow considerably, premature convergence on a suboptimal solution due to loss of genetic diversity in the population. Children are only allowed to advance to the next generation (i.e., replace their parents) if they perform better than their parents. This both guarantees that average population fitness can only increase, and also introduces a form of elitism, whereby good solutions that outperformed their children would continue survive to the next generation.

In our cellular configuration, each runner is allowed to mate only with its fittest neighbor. The cellular configuration is synchronous, meaning that the algorithm proceeds from the top left individual through each row until every individual has mated. A temporary population is used to store the best runners produced for each index. After the entire population has mated, the current generation is replaced with the new temporary population.

The fitness function uses two parameters collected by Qwopper. The first is the final state at the end of a run, that is, either *stopped* or *crashed*. If the runner falls over, Qwopper considers the final state *crashed*, and in that case we assign a fitness of 0.0 (regardless of the distance achieved before crashing). If the run ends for any other reason, such as the time limit being reached or the race having been completed, the final state is *stopped*, and in that case we assign a fitness equal to the second parameter, which is the final score, in particular the *distance*, achieved when the game is stopped. Other combinations were explored through experimentation, but are not reported here.

The population was initialized in one of two ways. In the first method, a collection of hundreds of random runners were generated and fitness-tested. Each random runner played QWOP for 60 seconds or until it fell down. Generation 0 was then seeded with the best performing randomly generated runners. In the second method, Generation 0 was populated completely with untested randomly generated runners.

We used a population size of 16 individuals for the SGA implementation, and a population size of 30 for the CGA implementation.

5. RESULTS

The combination of the physics engine and restrictive control scheme make falling over in QWOP very easy, both for humans and, as it turned out, also for the GA implementations described. It is critically important for the player to achieve a stable gait. While the fitness function also selects for speed, this can only evolve after stability is achieved. Compounding the difficulty is the fact that some amount of randomness exists when the Qwopper program plays QWOP. Because of the ragdoll physics system, when the game starts, the runner is first initialized to a standing state, and then the physics rules are immediately applied, causing his body to "settle in" due to gravity. To illustrate the runner's instability, one can simply start a game and not press any buttons, then watch as the runner eventually begins to wobble and fall forward as the physics calculations appear to create a feedback loop that increases the wobble. This introduces some amount of random variability in the performance of *any* given solution sequence played by the Qwopper program. A single sequence of inputs, run multiple times, will rarely result in the exact same score. It was often observed in early generations that the same runner could score well in one trial and crash in the next. This is evident in the run logs, as many populations were initialized to the same generation of runners, and in each trial these runners achieved different scores. This inconsistency increases the importance of a stable gait, as the best surviving gaits were necessarily those that crashed the least despite this variability.

It is therefore no surprise that the primary trait that emerged was a sort of "stability gene," usually expressed as an opening sequence that resulted in a stable stance. There were two button combinations that almost always came to dominate the population. Namely, to simultaneously press Q and O or to simultaneously press W and P.

The 'QO' combination moves the left thigh backward and extends the right calf forward, causing the runner to step forward and drop to his left knee with the right leg extended in front. The 'WP' opening performs the inverse operation of bending the right leg back and extending the left leg forward, which drops the runner to his right knee. In both openings, the runner lowers his center of gravity and spreads his legs wide, as if trying to do the splits. Figure 7 illustrates both combinations.

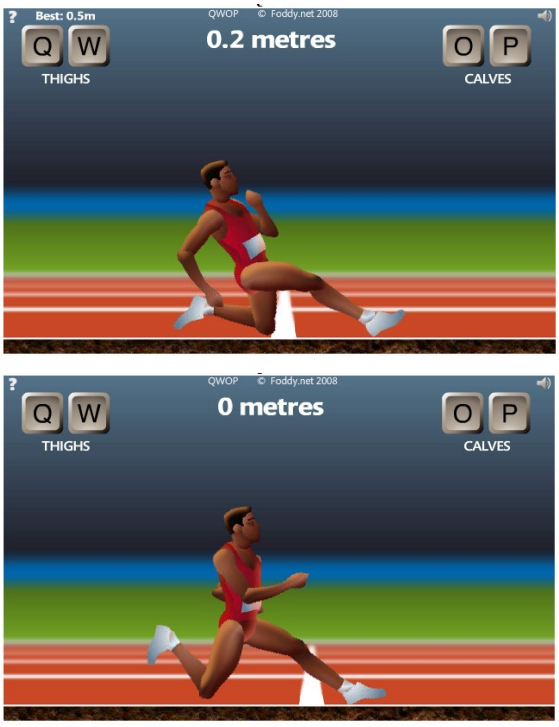

Figure 7. Opening moves 'QO' (above) and 'WP' (below)

The 'QO' and 'WP' openings dominated populations, even when using a local selection policy in a large population to slow premature convergence.

Notably, the opening consistently used by the fastest human QWOP scores documented is instead 'WO', which causes the runner to push off with the back foot and raise the knee of the other leg, as shown in Figure 8. The GA always evolves away from this opening because following it with a stable gait is likely impossible without the visual sensory feedback that the fastest human players enjoy, and that is not yet a part of Qwopper. Our GA implementation, which uses an essentially "blind" input loop, is inherently unable to react to visual feedback to adjust its gait during a run like a skilled and sighted human player can, so the 'WO' gene is quickly bred out of the population in favor of solutions with more stable openings.

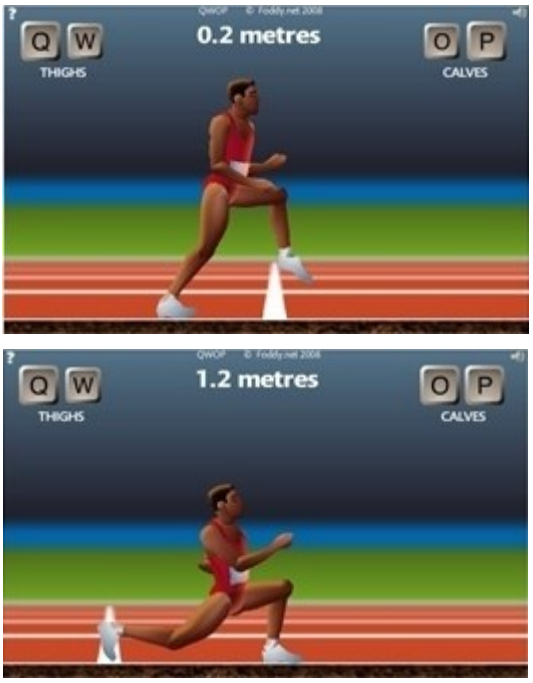

Figure 8. The 'WO' opening stride and resulting stance

Our best-evolved solutions transitioned from one of the two stable openings shown in Figure 7, into a gait that scooted forward while largely maintaining the runner's stable stance. Later generations gave way to the transition from scooting forward to more of a hopping gait. In these gaits, the runner would typically hop on one knee while kicking his free leg out in front of him to increase forward momentum. Even in the hopping gait, the legs remained spread apart, maintaining stability. Figure 9 shows the runner engaged in this evolved hopping gait. The gait is able to complete the QWOP race in about 2 minutes.

We posted a video of the evolved gaits, starting with some initial unsuccessful gaits and ending with the evolved hopping gait, at `http://www.youtube.com/watch?v=eWxFI3NHtT8`.

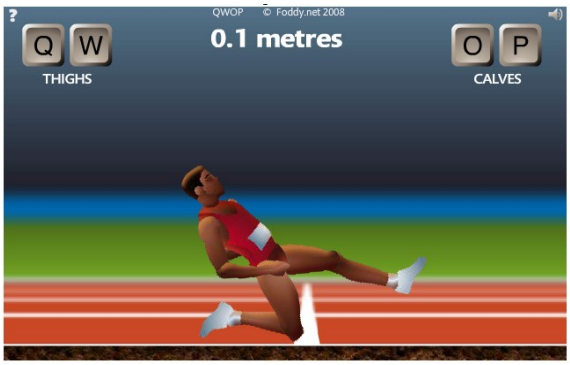

Figure 9. Gait involving hopping on one knee while kicking forward with the free leg.

5.1 Algorithm Performance

Three configurations of the GA models and parameters described earlier are tested. The first configuration uses a generational model, encoding 1, 3:2 tournament selection (three individuals are selected at random, the best two of which serve as parents), single-point cut-and-splice crossover, and a population consisting of the 16 best-performing runners from a pool of 390 randomly-generated individuals. As seen in Figure 10, this configuration fails to evolve solutions that played QWOP any better than the best randomly generated ones. This lack of improvement was attributed to a consistently observed high rate of failure in children (>50%). Because this GA configuration used a naïve succession policy wherein every child produced was guaranteed to succeed to the next generation, this high failure rate precluded the gene pool from improving with any significance.

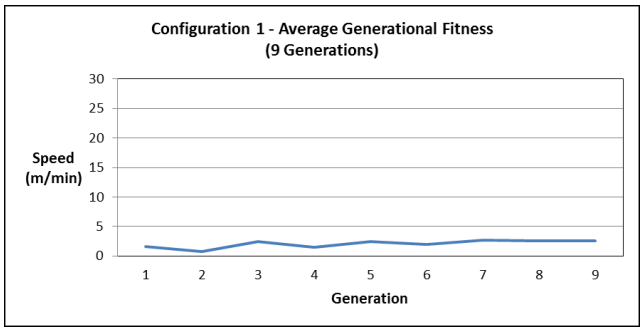

Figure 10. Fitness improvement in Generational GA model

The second configuration tested is identical to the first (including the same initial population), except that it uses a succession policy similar to that of the steady-state model. In this configuration, children only survive to the next generation if they outperform the worst of the three potential parents selected. This single modification results in a GA that produces positive results, as shown in Figure 11.

Note that average generational fitness does not always improve, because offspring are not required to be better than their parents.

The third configuration, and the one that yields the most stable and consistently efficient gaits, uses the CGA model, encoding 2, two-point crossover and a population size of 30 individuals. This population is initialized with the best individuals from a pool of 500 randomly-generated runners. Figure 12 shows the relative performance of configurations 2 and 3.

We took a closer look at the two encoding methods (Vaucher's original keystroke encoding, and the newer alphabet of keystroke combinations) to determine to what extent the quality of the initially-generated populations might be influencing the results. To this end, two pools of random runners were generated, one for each encoding method, and compared. Although the vast majority of randomly-generated runners for both methods crashed at or near the starting line, the top 30 runners from a pool of 500 using encoding 2 traveled, on average, over twice as far as the best 16 runners from a pool of 390 using encoding 1. The average fitness of the 30 fastest random runners from encoding 2 was 5.767 meters per minute, while encoding 1 averaged 2.033 meters per minute in its 16 fastest random runners.

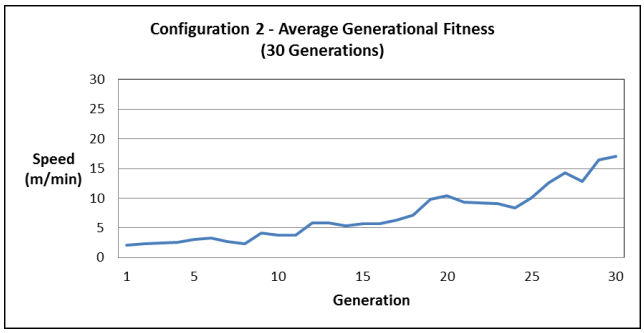

Figure 11. Fitness improvement in Steady-State model

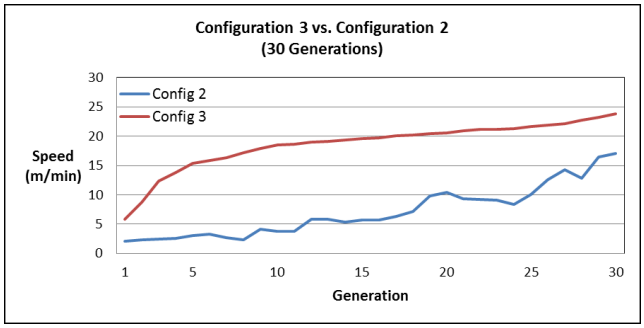

Figure 12. Fitness improvement in Config. 2 vs. Config 3

To control for this performance gap between the best random runners generated by the two different encodings, Configuration 3 was further tested using a different initial population of random runners, all of which had crashed outright when first tested by Qwopper. Despite a low average fitness of this initial population at 1.3 meters per minute, the average generational fitness accelerated just as quickly in early generations as did the population seeded with high quality random candidates. Regardless of the fitness of the initial population, Configuration 3 consistently performed better than Configuration 2, suggesting that the other GA parameters were largely responsible for the algorithm's accelerated evolution toward faster runners compared to the previous configurations.

The performance of configuration 3, initialized with fit versus unfit randomly-generated individuals, is shown in Figure 13.

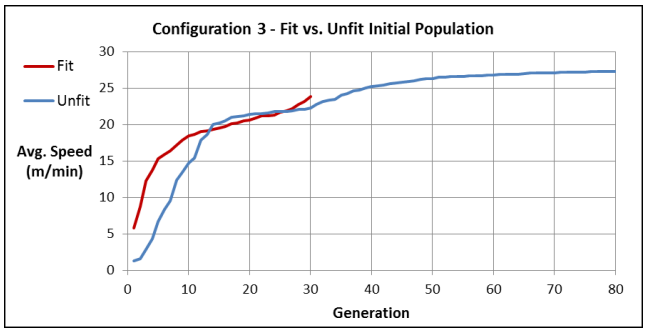

Figure 13. performance, fit vs. random initial populations

6. CONCLUSIONS

This project experimented with and compared many different flavors of the genetic algorithm and its components on the task of evolving QWOP gaits. Some of configurations were demonstrably better suited to solving this particular problem than others. This experiment found the greatest success with a Cellular GA.

The results confirm that steady-state evolutionary models outperform generational models when the fitness of potential solutions is highly sensitive to small changes in their encoding. This is intuitive considering the high failure rate of children observed throughout this experiment. Maintaining successful individuals and only replacing the weak ones with children of higher quality was critical to evolving successful gaits.

Through repeated trials we observed that certain traits, particularly traits that conferred a greater degree of gait stability, consistently emerged and eventually came to dominate the population. This observation held true even in a Cellular GA implementation which used a local selection policy in conjunction with a larger population size. If an argument can be made that the GA prematurely converged toward only solutions with the most stable opening moves, it is likely a consequence of requiring the system to learn to play QWOP without the ability to react to dynamic sensory feedback.

The results suggest methods that might evolve even better solutions to QWOP. Clearly the GA would benefit from a fitness function that provided more sensory information, such as the physical configuration of the legs or body of the runner at a given time. Another possible improvement would involve separating the opening stage from the looped stage, since it is possible that an optimal input loop may not contain the same sequence that starts the runner from his initial standing position, which is only needed once at the beginning of the race. There is also room for further exploration with regard to solving QWOP using other variations of the genetic algorithm that were not explored here.

One factor that limited the amount of experimentation we were able to do is the time that it takes to evaluate the fitness of an individual. Qwopper is a bot that must run each individual in a browser window in real time; a typical population of size 30, evolving for only 30 generations, often required over 20 hours of uninterrupted execution. Fortunately, the steady-state GA models that we used (including the CGA) have already been shown to be highly parallelizable [4], representing another potential avenue for improving the results.

Interestingly, the best solutions ultimately found by the CGA bear a striking resemblance to the way many people actually play the game. Most people who play QWOP do not post speed-run videos on YouTube and compete for the world record, which is currently held by Roshan Ramachandra at 51 seconds [9]. Finding a repeatable pattern and scooting to the finish line is a commonly-recommended strategy in online discussions about how to beat QWOP, which is the same type of solution that the GA consistently achieves.

Considering QWOP's volatile ragdoll physics system, its limited control scheme, and the fact that the GA has no mechanism for direct sensory feedback with which to respond and make adjustments, the gaits achieved by the GA implementations described in this paper serve as a proof of concept for robotic gait learning in a complex space with minimal inputs and feedbacks.

7. ACKNOWLEDGMENTS

We wish to thank Bennett Foddy for granting us permission to use the images of QWOP that appear in this paper.

8. REFERENCES

[1] Brodman, G. and Voldstad, R. (2012). *QWOP Learning*. Department of Computer Science CS 229 class project, Stanford University, 2012. Retrieved 4/13/2014 from http://cs229.stanford.edu/projects2012.html

[2] Clune, J., Beckmann, B., Ofria, C., and Pennock, R. (2009). *Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding*. Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC '09).

[3] Foddy, B. *QWOP* [Flash Game] (2008). Retrieved 11-6-13 from http://www.foddy.net/Athletics.html

[4] Gordon, V. and Whitley, D. (1993). *Serial and Parallel Genetic Algorithms as Function Optimizers*. Proceedings of the 5th Int. Conf. on Genetic Algorithms (ICGA-93).

[5] Holland, J. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, ©1975.

[6] Jakobsen, T. (2001). *Advanced Character Physics*. Proceedings of the Game Developers Conference (GDC-01).

[7] Mahnig, T. and Muhlenbein, H. (2002). *A Comparison of Stochastic Local Search and Population Based Search.* Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02).

[8] Parker, G., Braun, D., and Cyliax, I. (1997). *Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm*. Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC '97).

[9] Ramachandra, R. (2013). *Fastest 100m run, QWOP (flash game).* Guinness World Records. Retrieved 11-6-13 from http://challengers.guinnessworldrecords.com/challenges/160-fastest-100m-run-qwop-flash-game

[10] Vaucher, L. (2011). *Genetically Engineered QWOP (Part 1).* Slow Frog Blog [blog]. Retrieved 11-6-13 from http://slowfrog.blogspot.com/2011/03/genetically-engineered-qwop-part-1.html

[11] Wheeler, C. (2012). *QWOP and Simulation Design, Pt. 2. The Rules on the Field* [blog]. Retrieved 11-6-13 from http://therulesonthefield.com/2012/07/10/qwop-and-simulation-design-pt-2/

[12] Whitley, D. (1989). *The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best.* Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89).

[13] Wolff, K. and Nordin, P. (2002). *Evolution of Efficient Gait with an Autonomous Biped Robot Using Visual Feedback.* Proceedings of the 8th Mechatronics Forum International Conference (Mechatronics-2002).