Evolving locomotion for virtual quadrupeds

by

Caleb Compton

B.S., Kansas State University, 2018

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2020

Approved by:

Major Professor

Dr. William Hsu

Copyright

© Caleb Compton 2020.

Abstract

Determining efficient gaits and walk-cycles for arbitrary body shapes is an ongoing problem that has a wide array of applications, from robotics to video game development and computer animation. Many different methods have been used in solving this problem, each with trade-offs in run-time efficiency, generality, and ease of implementation. The technique used in this project is Proximal Policy Optimization, a form of reinforcement learning in which efficient walk cycles can be learned and improved automatically. This technique will be applied to a quadrupedal agent, which will learn to walk to a target location in a simulated environment.

In addition, this project further optimizes the body of the agent over time for more efficient locomotion with genetic algorithms. In each generation 10 randomly mutated quadruped agents will be created, their performance evaluated, and the performance evaluations used to produce the next generation. In this way the agent's body and gait will evolve together to achieve the desired results.

Table of Contents

List of Figures v

List of Tables vi

Acknowledgements vii

Chapter 1 - Introduction 1

Chapter 2 – Related Works 3

Chapter 3 – Methodology 9

3.1 Training Environment 9

3.2 Reinforcement Learning for Locomotion 10

3.3 Quadruped Optimization 12

3.3.1 Quadruped Mutation 13

3.3.2 Genetic Algorithm 15

Chapter 4 - Analysis 17

Chapter 5 – Conclusions 23

References 28

List of Figures

Figure 1 - A default quadruped, with scale of 1 for every rigid body ... 9
Figure 2 - A single instance of the training environment ... 10
Figure 3 - Examples of mutated quadrupeds ... 14
Figure 4 – Results of training with different starting conditions ... 18
Figure 5 - Default quadruped slithering on the ground. Torso outlined in red ... 20
Figure 6 - Top performing body shapes for models 1 (A), 2 (B) , and 3 (C) ... 21

List of Tables

Table 1 - Curriculum Lesson Values 13
Table 2 - Time-steps required to learn each lesson in the curriculum 19

Acknowledgements

I would like to thank Dr. William Hsu for serving as my major professor on this project, and for all his advice and encouragement along the way. Additional thanks go to Dr. Mitchell Neilsen and Dr. Josh Weese for serving on my graduate committee.

I also wish to thank my wife Abigail Compton for all her love and support.

Chapter 1 - Introduction

When creating a 3D character in a virtual environment, such as animated films or video games, the character should move around that environment efficiently and believably. Although many different techniques can achieve this, the specific technique chosen often depends on the particular application. The two most commonly used techniques are motion capture (mocap) and hand animation. Both of these techniques can achieve very lifelike results, but are also quite limited. Hand animation can be extremely time consuming, and the resulting animations are usually limited to agents with similar proportions and body structure. Mocap is usually quicker but can only be used for characters with a humanoid body shape.

Both techniques depend heavily on the shape and proportions of the character being animated. If the character shape changes, previous animations must either be adjusted or entirely redone. For many applications this is not a major problem, because most characters in films and video games do not change shape over time, and character design is usually determined before animation begins. However, in some applications these techniques are infeasible. For example, if the body shape or proportions of the character can change over time, in response to user input or the environment, preparing animations using mocap or hand animation is not possible. Instead, a new technique is required that can develop new animations procedurally, one that can adapt to changes in the character's shape.

This report covers one such technique, in which the body style of the character adapts to perform a particular locomotion task. Specifically, in this project, a simple quadrupedal character consisting of several capsules connected by movable joints will

adapt its proportions to move more effectively through a 3D environment built in the Unity 3D game engine. The shape and proportions of this character are not fixed, so traditional animation techniques would not be effective for this application. Instead, the character should learn to move procedurally using reinforcement learning (RL). As the character learns to move, its body will adapt to move more effectively through genetic algorithms. In this way, the character's body and motion will be optimized simultaneously.

The use of RL to generate animations for the character has the additional benefit of allowing unique locomotion methods to be developed for different training environments. While only a single training environment was used for this project, the same technique could easily be adapted to a diverse range of different environments, which would result in different body styles and types of movement.

The remaining chapters will explain how this project was implemented and the results and conclusions. Chapter 2 will provide additional background on this topic by exploring related work and previous research in this area. Chapter 3 will explain in detail how this project was implemented, including the exact tools and techniques used. Chapter 4 will include the results of this project, and Chapter 5 will provide conclusions and recommendations for how this project could be continued and improved.

Chapter 2 – Related Works

The problem of simultaneously evolving the body and motion of a character is one that dates back to 1994, with the paper “Evolving virtual creatures” by Karl Sims [1]. The project Sims described used evolutionary algorithms for evolving both the shape of a creature and the neural structures that controlled its movements. The results of this project were block-like creatures that could perform tasks like walking, swimming, and protecting a green cube. Although this resulted in some very interesting body styles and movement techniques, the results were not very lifelike. Some of the results resembled existing animals, such as snakes, but most of the majority of the evolved creatures did not, and the movements that evolved did not resemble actual animal movements.

In 2000, Hod Lipson and Jordan Pollack expanded on the techniques pioneered by Sims in their paper “Evolving Physical Creatures”, in which these techniques were applied only to virtual creatures but also to physical robots [2]. This project evolved creatures to perform a locomotion task in a virtual environment, then automatically 3D printed high-performing models. To ease the transition to reality, Lipson and Pollack simplified the virtual models as much as possible. They did so by only allowing elementary building blocks in their models – the virtual creatures consisted only of bars connected by joints, with simple actuators serving as muscles. When 3D printing these creatures, the virtual representation was automatically converted to a printable form, and accommodations were included for linear motors which were manually added to the model by the researchers. The same neural network that controlled the virtual creature could control the robot by activating the motors, and the behavior of the robot faithfully reproduced the actions of its virtual ancestors.

The approach to virtual evolution found in [1] was imitated by many projects throughout the 90's and 2000's [2]-[4]. However, as noted by the authors of "Unshackling Evolution" in 2013, the complexity and realism of virtual creatures did not significantly improve during these decades [5]. For example, the creatures produced in [4], published in 2011, were at a similar level of complexity to those that evolved in [1], written 17 years prior.

One of the factors hypothesized to be limiting the evolution of virtual creatures was a reliance on rigid bodies, which do not accurately represent the bodies of real-world animals [5]. To address this issue the authors chose to evolve voxel-based creatures made of four different materials – soft "flesh", hard "bone", and two different forms of "muscles". These soft virtual creatures were able to produce lifelike motions, many of which resembled the gaits of real-world animals. In addition, the creatures that evolved during this study outperformed creatures designed by humans using the same materials.

A different approach was taken by the development team working on the 2008 video game Spore. Creatures in Spore were designed by the user, not evolved for a particular task, and behaviors like walking and dancing were customized for each creature. Spore had the additional challenge that these behaviors and animations had to be performed in real-time without a long and computationally expensive optimization process.

The solution chosen by the Spore team was to retarget existing animations rather than produce new entirely animations [6]. The animations would begin with reference poses, called keyframes, designed by an animator. These reference poses were then stored in a generalized form that relied on body-independent position and rotation data that could

be easily rotated or mirrored. Additional constraints were placed on this data that limited the contexts in which the animation could be used. For example, a particular animation might only apply to graspers (a type specification) in the front half of the creature (a spatial specification). When needed, these animations could be converted from a general form to a more specific form based on the skeleton of the particular creature being animated. The result was creature-specific goal poses, which could then be reached using an IK solver.

Spore provided animations for user-designed creatures, but this retargeting technique could theoretically be applied to creatures that are procedurally generated or produced through evolutionary strategies. However, the technique is fundamentally a kinematic one. While this technique is ideal for performing real-time, user-controlled actions in a video game, it does not apply to physics-based characters in simulated environments.

Won and Lee, in their paper "Learning Body Shape Variation in Physics-based Characters" were explicitly inspired by the approach taken in Spore. They sought to develop a similar system that would control arbitrary user-designed characters without retraining the character controller [7]. Their primary approach included parameters that described the body shape as part of the character state specification, and used deep reinforcement learning (DRL) to train a controller on a variety of parameter values. The body parameters used in Won and Lee's project included length of limbs and width of torso but could not adjust attributes such as the number of limbs or joints. This is due to the constraints of DRL – specifically, the constraints required a fixed number of input

dimensions. Because of this, their controller allowed on-the-fly variation of the proportions of a character, but each controller is still restricted to a single body configuration.

Kevin Wampler and Zoran Popović tackled a similar problem in “Optimal Gait and Form for Animal Locomotion” [8]. This project used a Spacetime Constraints approach, in which various aspects of the desired motion (such as start and end positions), information on physical structure of the character, and constraints on timing (such as when each foot should contact the ground) formed a constrained optimization problem that could be solved to produce the desired action [9]. The Spacetime Constraints technique typically applies only to characters with a fixed body structure because body configuration is part of the problem specification. However, Wampler and Popović expanded this technique to allow for additional optimization of the character body structure by representing the body proportions as additional variables to be optimized rather than fixed constraints. The result of this technique is a closed-loop motion that satisfies all necessary constraints (such as start and end positions) while optimizing the desired criteria (such as minimizing energy expended). The drawbacks of this technique are that the user must provide a detailed specification of every movement problem to be solved, and every action must be solved separately.

Directly applying torques to limbs or joints is a common way to simulate movement in virtual creatures, as seen in [1],[7], and [8], but it is far from the only possibility. Another common solution is to simulate muscles directly, at varying levels of detail. A very simple muscle simulation was used in [5], and a much more physically realistic simulation is used in “Flexible Muscle-Based Locomotion for Bipedal Creatures” [10]. In this paper, the authors describe a complex muscle simulation that simulates muscle contractions and tendon connections between bones. They also specify a technique for automatically optimizing muscle geometry,

including placement, length and attachments. This is necessary because there may not be real-world data regarding the optimal muscle geometry for every body structure.

Muscle simulation can result in more realistic gaits, but at the cost of increased complexity. In addition, because my project allows varying body proportions, a complex muscle simulation would require an additional optimization step in order to optimize the muscle connections between each rigid body, and this optimization step would have to be repeated every time the character evolves. For these reasons, and because direct torques have been shown to produce physically plausible gaits in [7] and [8], I chose to apply torques directly to limbs in this project.

Previous work in this area used several different approaches to the body structure of virtual characters. Some techniques allowed wildly varying structures with no fixed number of segments [1]-[4], [6], while others produced minor variations on an existing body structure [5], [7], [8]. As mentioned in [7], traditional DRL techniques require the user to specify a fixed number of dimensions for the state and action representations, which would make it very challenging to have a variable number of body segments. Because I used RL to train the characters in this project, I took an approach to body structure similar to [7] and [8], which focused on adjusting the proportions of the body instead of generating entirely new body shapes.

Other projects have also taken many different approaches to evolving optimal body shapes. While evolutionary techniques are quite common [1]-[5], approaches including user-guided evolution [6], and constrained optimization [8], [10] have also been successful. However, in “Why virtual creatures matter”, author Sam Kriegman explains that true morphological optimization – “changing the number and placement of degrees of freedom, not merely tuning

the parameters of a predefined structure" – had only been achieved through evolutionary techniques [11].

This may be changing – in 2019, the paper "Learning to Control Self-Assembling Morphologies" presented a technique in which simple, single-segmented agents could link-up to form larger, multi-segmented agents [12]. In this project, the decisions to join together or break apart is determined by a neural network. This paper is the first time a gradient-based technique has been used to dynamically change the number of limbs instead of an evolutionary technique, and the technique performed well on a number of locomotion tasks.

Since the characters in my project have a fixed number of degrees of freedom, it may have been possible to use a gradient-based or constrained-optimization approach to determine optimal body shapes. However, analytical optimization usually requires a very clear specification of the task to be optimized, while a genetic algorithm allows the body to adapt dynamically to changing movement styles as the creature learns. Because the locomotion task is open-ended and constantly changing, I chose to use a genetic algorithm to optimize the characters in this project.

Chapter 3 – Methodology

3.1 Training Environment

The main goal of this project was to develop a system that allowed a character to evolve and learn in a variety of simulated environments, so I developed the project primarily using the Unity game engine. Unity provides a number of features like built-in physics engine, 3D rendering, and support for rigid bodies and joints, which allowed me to focus on training and evolving my characters rather than reimplementing these features.

The character I used for this project consists of 12 rigid bodies, which include a head, neck, torso, tail, four upper legs, and four lower legs. These rigid bodies are connected using Character Joints – a type of Unity joint that allows specified maximum and minimum rotations around three rotational axes for each joint [13]. Figure 1 shows a default quadruped that has not undergone mutation.

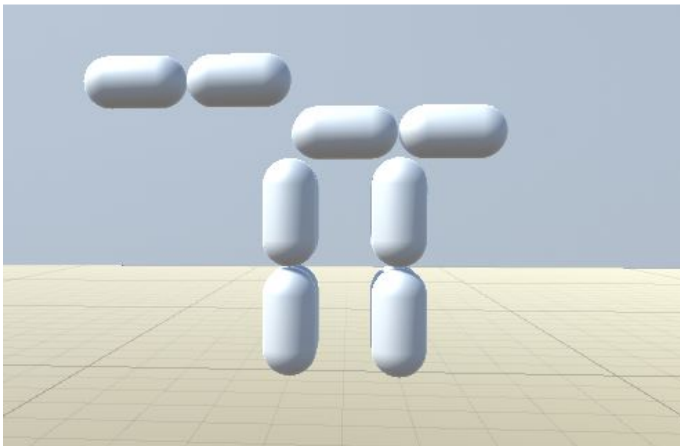


Figure 1 - A default quadruped, with scale of 1 for every rigid body

Each character was trained within an individual training area – an enclosed rectangular area containing the quadruped and a marked goal location, shown in Figure 2. During training 10 instances of the training area were used.

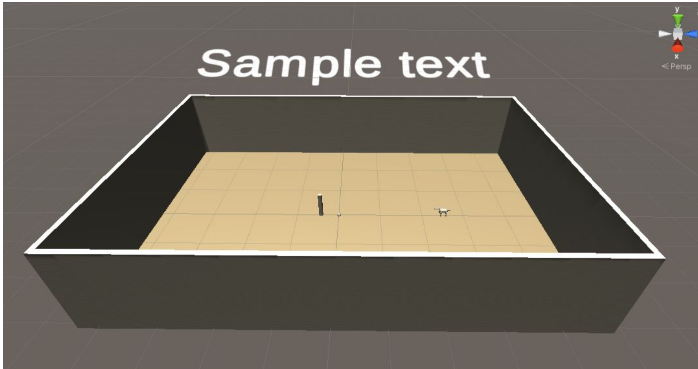


Figure 2 - A single instance of the training environment

3.2 Reinforcement Learning for Locomotion

In *Reinforcement Learning: An Introduction,* Richard Sutton and Andrew Bartow define RL as a way to map situations to actions so as to maximize a numerical reward signal [14]. In this project, the reward signal was provided by a reward function, R(s_i, a_i). The system maximizes this function by optimizing a policy π that, given a state s_i, will return an action a_i that will advance the system to state $s_{(i+1)}$. Over time, this policy function learns to choose the action that maximizes the expected reward for the agent. RL as a field consists of a variety of different algorithms that achieve this goal, each in different ways.

The particular technique used for this project is Proximal Policy Optimization (PPO), an RL algorithm developed by OpenAI that has outperformed most other RL algorithms in learning efficiency for continuous control tasks, including locomotion [15]. For this project, I did not develop my own implementation of PPO; instead, I used the Unity ML-Agents library, which provides an implementation of PPO and facilitates interaction between the Unity game engine and the TensorFlow machine learning platform [16]. Although I did not implement the PPO algorithm, I still provided specifications for states, actions, and the reward function.

At each time-step the character's state s_i consists of 61 different values representing the agent's current configuration as well as observations about its environment. These values include the character's current global position and velocity, the position of the goal, the agent's current forward vector, the target direction vector, the rotation of each of the agent's limbs, as well as which of the lower limbs are in contact with the ground.

Each action a_i consists of 11 vector values, each containing 3 floats. Each vector represents a torque that will be applied to one of the character's limbs, and each of the three floats represents the torque around a particular axis of rotation. No torque is applied directly to quadruped's torso, so there are only 11 values. Each torque is scaled based on the cross-sectional area of the limb, as well as a hand-tuned torque multiplier.

The reward function $R(s_i, a_i)$ for this character has four main parts. At each time-step the agent's current distance from the goal is compared to their previous distance, and the difference is used to determine their progress value. This progress value is then scaled by a hand-tuned value and added to the current reward. This encourages the agent to move closer to the goal, and discourages moving further away. If the agent reaches the goal (within an acceptable radius) it receives a larger reward, and the training area for that agent is reset. To encourage efficiency, the agent receives a small penalty based on the magnitude of the torque it applies at each time-step. This encourages a more efficient gait, and discourage unnecessary movement. In addition, the agent has a single failure condition that applies a large penalty and resets the training area. For this project, if the agent's head or body touch the ground, which indicates that the agent has fallen over, the failure condition is met.

This project uses curriculum learning to speed up the process of training the agent. In ML-Agents, a curriculum consists of a series of lessons that the agent works through one at a time. When the average score of the agents exceeds some minimum threshold (shown in Table 1), the curriculum will move on to the next lesson. In this way the same agent can initially be presented with a simple task, and the task can become progressively more difficult as the training continues.

Lessons become more difficult by adjusting the values of certain variables within the training system. In this project, the variables controlled by the curriculum are the distance to the goal, the acceptable radius around the goal, and the range of angles in which the goal can be placed. Initially the goal is placed quite close to the agent, so it can be reached even accidentally. Over time, the goal is placed further and further away from the agent, forcing it to walk longer distances to reach it. In early iterations, the goal is placed directly in front of the agent; in later iterations it can be placed in a range of up to ± 30 degrees off the agent's forward axis. Table 1 shows the values associated with each variable in each curriculum level. As you can see, the variable that controls the acceptable radius around the goal remains fixed. Initially this radius decreased over time, but when testing the project I found that certain quadruped configurations could not fit within a smaller radius. I also found that a larger radius was unnecessary in the earlier lessons, so I chose to keep this variable fixed.

3.3 Quadruped Optimization

The second major goal in this project was optimizing the shape of the quadruped to perform its locomotion tasks more efficiently. This was done in two steps: mutation and selection. The mutation function produced a range of possible quadruped configurations,

while the selection process determined which configurations the most effective. These selections were then used to populate the next generation of quadrupeds.

Table 1 - Curriculum Lesson Values

	Default	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5
Max Angle (degrees)	0.0	0.0	5.0	10.0	15.0	30.0
Goal Distance (meters)	10.0	15.0	20.0	25.0	35.0	55.0
Acceptable Radius (meters)	6.0	6.0	6.0	6.0	6.0	6.0
Score Required to reach next lesson	0	0	0	10	10	N/A

3.3.1 Quadruped Mutation

Each character entered training the same way: as the default quadruped shown in Figure 1. Before training began each character underwent mutation individually, resulting in a population of 10 unique quadrupeds.

The mutation function produced a mutation vector for each portion of the quadruped's body. Because the right and left legs should reflect each other, this function produced a total of 8 different mutation vectors. Each vector contained 3 float values between $\frac{1}{9}$ and 3. These float values each represented a scale factor applied to each body section along their three local axes.

Each vector applies only to a single body section, but they are not entirely independent. The vectors were subject to a number of constraints to ensure they produced

a valid quadruped configuration. The first constraint ensured that the front legs had the same length as the back legs (to ensure that both the front and back legs had valid joint connections to the torso, which is not guaranteed if they are different lengths). A second constraint guaranteed that the front upper legs have the same x-z cross-section as the lower legs, and the back legs had a similar constraint. This ensured valid joint connections between upper and lower limbs. In addition, the width of the neck, head, and tail should not exceed the width of the torso, and the torso should be wide and long enough to accommodate the legs beneath it. These constraints prevented unwanted collisions or intersections between the rigid bodies.

Once a valid configuration was reached each body part was scaled and then placed in its proper position relative to the other parts. Lower legs were placed so that the bottom of the leg was in contact with the ground, and they were positioned at the four corners of the torso. The upper legs were placed directly on top of the lower legs, and the torso was placed directly above the upper legs. The neck was placed directly in front of and above the torso, and the head was placed directly in front of the neck. The tail was placed directly behind and slightly below the top of the torso. Example configurations of mutated quadrupeds are shown in Figure 3.

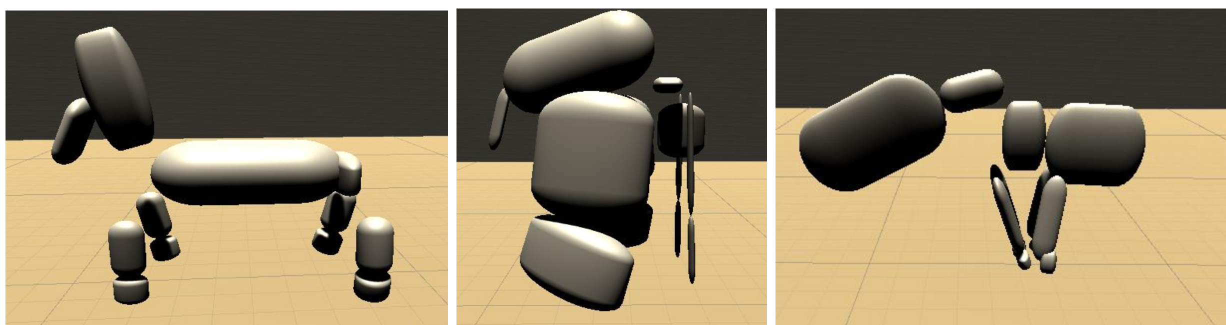


Figure 3 - Examples of mutated quadrupeds

3.3.2 Genetic Algorithm

Training for the quadrupeds consisted of multiple different rounds, with each round lasting a total of 5000 time-steps. Training was simulated at 10x speed, so a single round lasted about 8 seconds. During each round a total of 10 different quadrupeds were trained and scored. At the end of each round, a genetic algorithm was used to produce a new generation of quadrupeds to participate in the next round of training.

In *Genetic algorithms in search, optimization, and machine learning*, David Goldberg describes genetic algorithms as "search algorithms based on the mechanics of natural selection and natural genetics" [17]. In the same way that natural selection improves the fitness of a population of animals, genetic algorithms improve the fitness of a population of virtual entities. This is achieved by mimicking the biological processes of reproduction and mutation - "in every generation, a new set of artificial creatures is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure" [17].

Each new individual is produced from two parents from the previous generation, and these parents are chosen by running a tournament. Through these tournaments, a subset of individuals from the previous generation can be selected, compared based on their fitness values (in this project the fitness value is simply each individual's cumulative reward function for the previous episode), and the fittest individual chosen as a parent for the next generation.

Once two parents are chosen a new individual is produced by performing two operations on the parents: crossover and mutation. Crossover is performed by selecting attributes from each parent with a fixed probability; in this project, the probability is 50% because there is no reason to favor inheriting traits from one parent over the other. While

producing a new individual, each attribute also has a fixed probability of being mutated; in this project, that probability is 10%. Once this process is complete, the new individual must be verified to ensure that the combination of inherited traits and mutations is still a valid quadruped configuration.

The genetic algorithm used in this project also uses a technique known as elitism, which ensures that the best individual generated so far is always included as a member of the next generation [17]. This ensures that the fitness of the fittest individual never decreases from one generation to the next. According to Goldberg this technique tends to improve performance in systems with a single optimum, but increases the likelihood of getting caught in a local optimum in a system with several optima [17]. Due to the complexity of this project it is likely that there are several optima. However, elitism is used in this project to more easily track the evolution of the fittest individual.

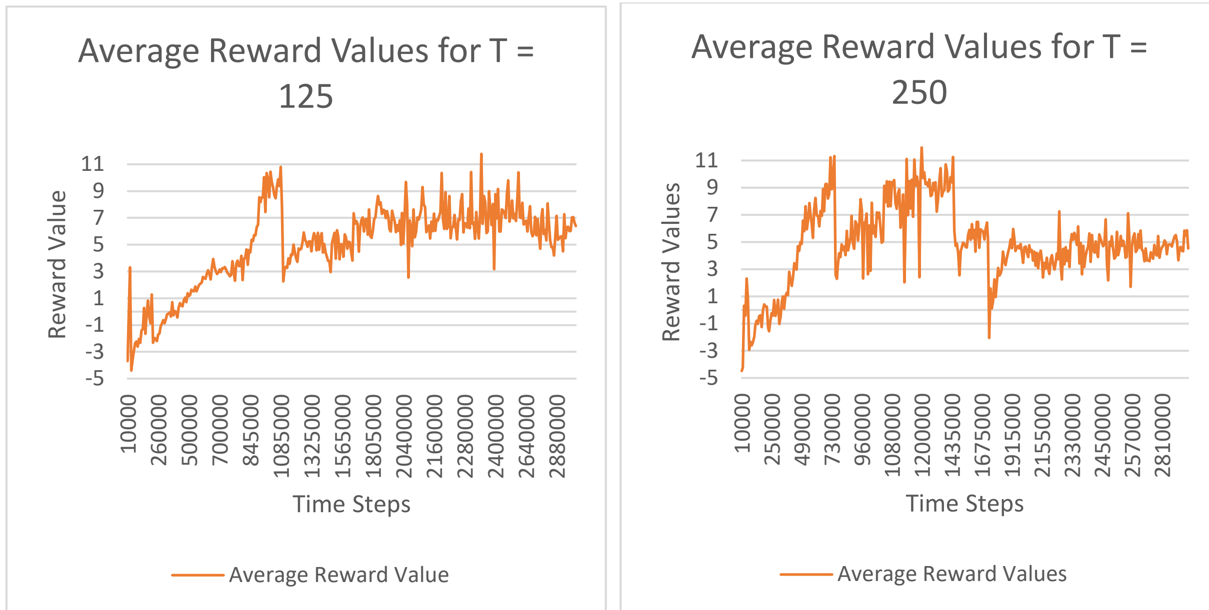
Chapter 4 - Analysis

After training models, I observed a number of interesting results. First, while the mutation function always resulted in creatures with a plausible quadrupedal body structure, these creatures did not demonstrate movements resembling those of real world animals. In many cases, the movement styles developed during training involved the character spinning, either around the up axis (yaw) or forward axis (roll), rather than propelling itself forward using its limbs.

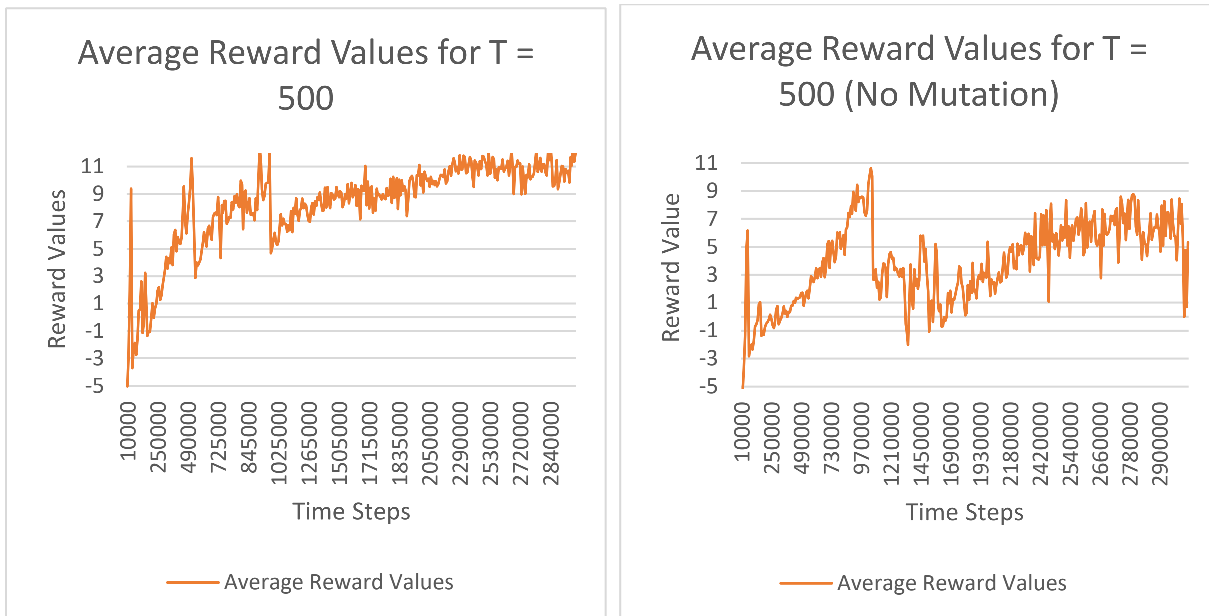
One parameter that had a strong effect on training these characters was the value chosen as a torque-multiplier T. At each time step a torque is applied to each body part (other than the torso), and the value of these torques in each direction is capped to the range [-1 , 1]. Because of this, each torque is scaled not only in proportion to the cross-sectional area of the limb, but also by a hand-tuned torque multiplier T. If T was too low, the torque would not be large enough to produce meaningful movement of the limbs or to overcome friction. If the T value was too large, it could result in extremely erratic movement or even cause the character to float because of the force exerted upon the torso by the limbs.

Through testing, I found that T values around 250 resulted in reasonable results: strong enough to move limbs, but too weak to cause floating or other erratic behavior. To see which T value was the most effective and how different values affected the results, I trained the model using three different torque multipliers: 125, 250, and 500. Each of these training sessions lasted 3 million time-steps (about 83 minutes), and contained 600 generations. This was enough for the performance of each generation to plateau. Figure 4

shows the results of each of these runs. In addition, Table 2 shows the number of time-steps required for each run to learn each lesson in the curriculum.



(A) (B)



(C) (D)

Figure 4 – Results of training with different starting conditions

Table 2 - Time-steps required to learn each lesson in the curriculum

	Model 1 (T = 125 with Mutation)	Model 2 (T = 250 with Mutation)	Model 3 (T = 500 with Mutation)	Model 4 (T = 500 without Mutation)
Lesson 1	35,000	55,000	35,000	55,000
Lesson 2	180,000	150,000	80,000	100,000
Lesson 3	160,000	90,000	40,000	140,000
Lesson 4	725,000	450,000	380,000	760,000
Lesson 5	N/A	705,000	415,000	N/A

In addition to training with different torque multipliers, I also compared the results of training with a genetic algorithm to training without an algorithm. A model was trained where every quadruped was the default, un-mutated quadruped shown in Figure 1. I wanted to compare the results with and without the genetic algorithm to see if the mutation actually improved performance; [7] argued that non-parameterized RL would not be effective for training characters with different proportions. I chose to train one additional model (model 4) with no mutation and T = 500. Figure 4 (D) shows the results of this training.

The charts in Figure 4 and the data in Table 2 reveal some interesting patterns. The first immediately obvious pattern is that the character learned more quickly as the T value went up; model 1 could not even make it to Lesson 5, and model 3 reached lesson 5 much more quickly than model 2. Clearly, a higher T value is roughly the equivalent of more powerful muscles in a living organism, and stronger muscles are likely to result in quicker movement. However, once T is more than 500 it begins to produce physically unrealistic behavior.

What is more striking than the difference in T values, however, is the difference in performance with and without mutation. Training without mutation was significantly worse than model 3 (which shared the same value of T), but actually performed very similarly to model 1 with a T value of 125. This shows that the genetic algorithm significantly improves performance of a population of quadrupeds.

In addition, the movement style developed by model 4 differs significantly from the movement that evolved from any other model. Rather than twisting (either around its roll or yaw axes), the movement developed more of a slither, in which it lies as flat as possible (without it's body touching the ground, triggering a failure condition) and wiggled from side to side. Figure 5 shows this slithering motion in-progress.

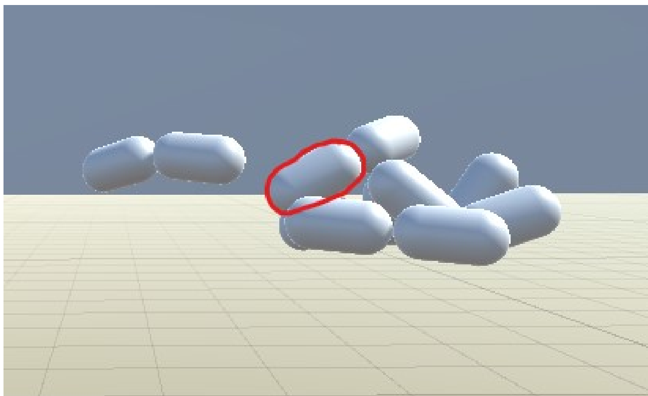
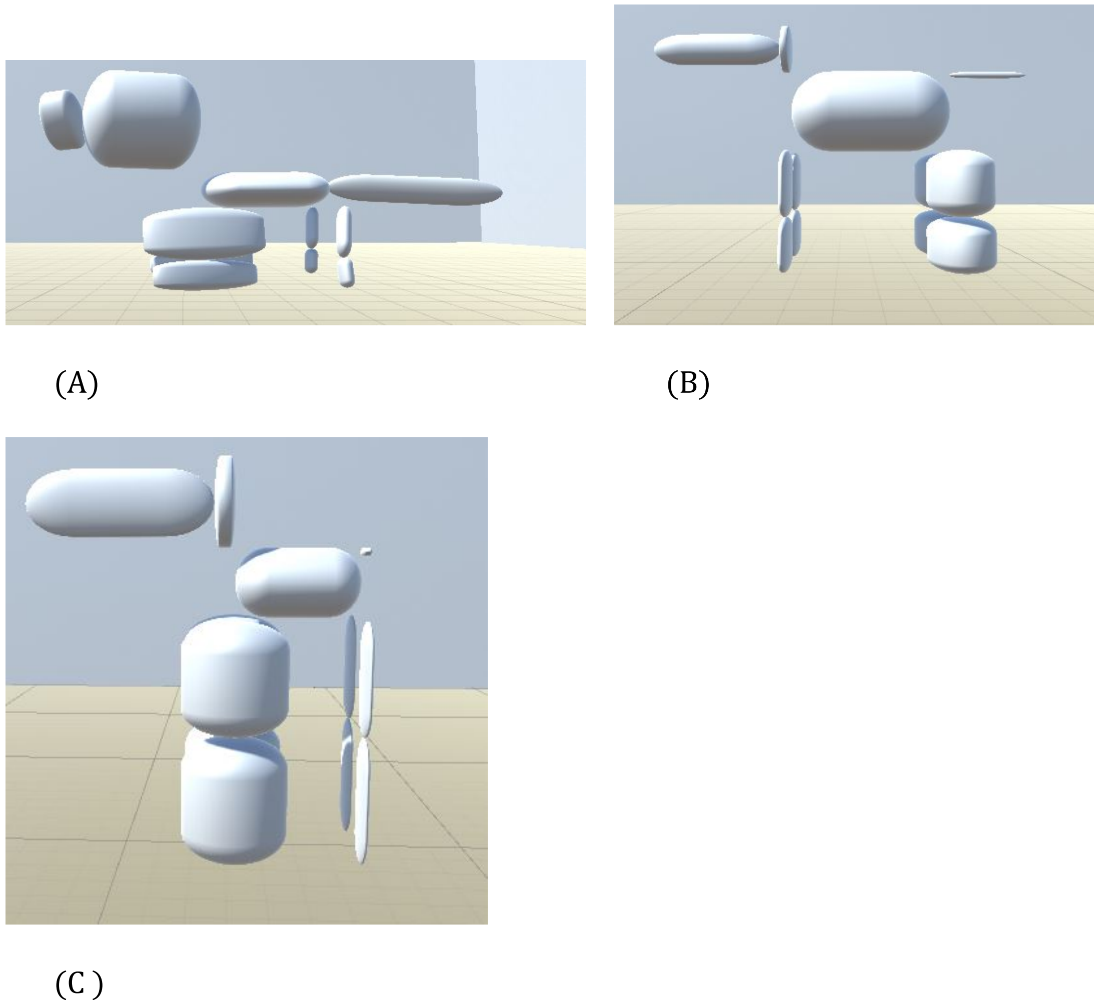


Figure 5 - Default quadruped slithering on the ground. Torso outlined in red

For those models that did involve mutation, Figure 6 shows the final body style from each training session. Among these body styles, the configurations resulting from models 2 and 3 are quite similar to each other; they both have short necks, minimal tails, long heads, one pair of narrow legs, and one pair of thicker legs. The primary difference between the two is that model 3 is much taller.

Model 1, on the other hand, is quite different. It is much shorter, with a large neck and tail. This may be due to differences in movement styles between models. Model 1 rolled around its forward axis, while the other two spun around their up (yaw) axes. This may also explain model 1's large neck (to protect its head while rolling), and the larger tail may serve to counterbalance the heavier neck. Like models 2 and 3, model 1 also has one pair of legs that is much larger than the other.



(A) (B)

(C)

Figure 6 - Top performing body shapes for models 1 (A), 2 (B) , and 3 (C)

One interesting pattern to note is that the models get progressively taller as the T value increases, possibly because each body part's weight increases proportionally to

volume, but muscle strength increases based on cross-sectional area. As the T value increases, this additional strength may make longer, heavier limbs feasible for the models. This could also explain why model 1 has very short limbs, but the front legs have a large cross-section; this combination helps maximize the strength of the front limbs. This may also explain why each model develops one pair of larger legs and a second pair of thin, almost vestigial legs. The models rely only on one set of limbs, thus reducing the total amount of weight that each model must carry. This could also explain why the tails in models 2 and 3 withered away to almost nothing; they were not contributing to the creature's motion, so minimizing the weight as much as possible was most efficient.

The reward function may also contribute to this phenomenon, particularly applying a small penalty proportional to the magnitude of the torque applied to the limbs, which was intended to encourage more efficient movement. In this project, the penalty depends not on the size of the limb, but on the unscaled magnitude of the torque being applied to each limb. Therefore, to minimize the penalty, the quadrupeds may have evolved to minimize the number of limbs that receive a torque at every time-step.

Chapter 5 – Conclusions

The primary goals for this project were to develop a system that allows a virtual character to evolve its body and motion in response to its environment. This goal was achieved by using reinforcement learning to govern movement, and a genetic algorithm for evolving the character's body over time. This combination was more effective in training the character to move from a starting position to a goal position than RL alone, although the resulting gaits and proportions did not resemble those of real-world animals.

While this project achieved most of its implementation goals, there is still much room for expansion. The most important area for future work, and the one implementation goal that could not be completed on time, would be to test this technique in several different training environments, not just one, and to observe how different environments affect not only the training process, but the resulting movement styles and the evolution of the character. As Chapter 4 shows, the evolution of the creature and the movement style are intrinsically linked. When you restrict a quadruped to a single body configuration, the result is a completely different movement style. Training quadruped characters in different environments would result in different body styles and, therefore, different movement methods as well.

One of the original goals for this project was to train the agent not only on land but also in aquatic and amphibious environments to see how this affects the evolution of the character. While training in these environments would not require any changes to the genetic algorithm or the RL system, implementing a physically-plausible water physics system would be necessary. This would have required two separate systems. First, a

buoyancy system would allow the character to float in the water, and second, a viscosity system would allow the creature to push against the water while swimming.

The formula for buoyancy is $B = \rho * V * g$, where ρ is the density of the fluid, V is the volume of the displaced fluid, and g is the force of gravity. Exerting a buoyant force simply requires approximating the volume of the character that is submerged at each time step, multiplying this volume by the weight of that volume of fluid, and applying this force upwards on the character either at the center of mass, or at several places on the character's body. For simplicity, the character and fluid could both be treated as if they have constant density, so a value of ρ would be required that produces physically plausible results.

To actually swim within the fluid requires modelling the viscosity of the fluid, so when the character's limbs exert a force upon the water, an equal force is applied to the character. To implement this would require an approach similar to [1], where the opposing force of a fluid was simulated by calculating the surface area of each moving surface in the direction of motion, multiplying this by the velocity of the limb and a hand-tuned viscosity constant, and applying the resulting force to each moving surface in a direction opposite the motion of the limb. Although it is a very simplified model, it was sufficient to simulate physically plausible motion in water [1].

In addition to training in aquatic and amphibious environments, many other aspects of the training environment could be adjusted, likely resulting in significantly different outcomes without needing to adjust the underlying systems. These changes could include uneven terrain like slopes, gaps, or obstacles.

On top of changing the learning environment, the character itself could also be changed to see how these changes affect the resulting body type and movement styles. There are four main ways the character itself could be changed: 1) The character could be trained with different (but still fixed) body styles; 2) the character could be modified to allow a variable number of body segments; 3) the current body structure could be modified to make it more biologically realistic; and 4) the technique could be applied to a skinned mesh instead of a collection of rigid bodies.

The first possibility is the simplest - the character could start with a different initial shape, perhaps bipedal instead of quadrupedal. Changing the initial shape of the character would require some changes to the mutation function and the RL parameters, but the general approach need not be changed.

The second option would be to increase the range of possible mutations to not only change the proportions of the character but to allow the evolution of entirely new body shapes. Such an approach would require a different underlying representation of the character. The directed graph representation used in [1] is a possibility. The comparison of mutated and un-mutated results in Chapter 4 shows that allowing the character to mutate resulted in more effective locomotion overall, so it is likely that increasing the range of possible mutations would produce even more effective results. The downside to this approach, as mentioned in [7], is that RL requires a fixed number of input parameters, and you cannot change the state representation during execution. One way to mitigate this limitation would be to expand the number of input parameters to allow more body parts and include a placeholder value for unused body parts. This would allow variation in the number of limbs, although an upper limit would still be necessary.

The other option would be to stop using RL entirely, and explore other ways to control a body with variable segments. One possibility is the dynamic graph network described in [12], which was able to adapt to multiple simple segments joining together to form complex body structures. Another possibility would be to avoid centralized control entirely, and allow each limb to control itself independently. This decentralized approach was used in [1] and [4].

The third way to modify this character in future work would be to address some of the less biologically realistic aspects of the current model. These aspects include unrealistic restrictions on the cross-sections of limbs, overly simplified muscle behavior, and the tendency to evolve unnatural movement styles.

This project required the character's lower legs to have the same cross-section as the upper legs, but real-world animals do not have this restriction. In addition, while each limb is scaled independently on each axis, the scale remains uniform along that axis – there is no tapering. This is not biologically realistic – animal limbs do not have a uniform width. To more closely simulate animal limbs, I could modify this project to scale the top and bottom of each limb independently. Rather than constraining the cross-section of the upper and lower limbs, I could instead only constrain the portion of the limbs that are directly connected by a joint. This would not only produce more realistic looking limbs, but would also increase the number of degrees of freedom that can be optimized.

The technique used to move the character's limbs is also unrealistic. In this project, limbs are moved by directly applying torque to each limb at the center of mass. This model is very simplified, and does not capture the complexities of animal movement. For example, the strength of the torque applied to a limb does not depend on the direction of the torque

– it is equally strong in all directions. This also means that the character's "muscles" are capable of both pushing and pulling, and are equally strong at both. This does not reflect the behavior of muscles in the real world. To solve this problem, a more realistic muscle simulation, like the one used in [10], could be applied to this project. This would add additional complexity, but would only require mild adjustments to the RL specifications.

Modifying the reward function could also produce more biologically realistic results. The reward function used in this project is very simple – it rewards the character for moving towards the goal, but provides no guidance on how the character should move. Because the locomotion task is intentionally left very open-ended, the character often evolved very unnatural movement methods. Additional penalties could be added to the reward function to encourage more realistic movement. For example, a penalty proportional to the angle between the character's up vector and the global up vector could be added. This would encourage the character to remain upright by punishing rotation around the roll axis. Similarly, a penalty proportional to the angle between the character's forward vector and the goal would encourage the creature to face towards the goal while moving.

Finally, the character itself could be represented as a skinned mesh, rather than a series of rigid bodies. Implementing this change would be relatively straightforward. Instead of mutating rigid bodies, the bones within the skinned mesh would be mutated. This technique could be used to develop characters for a video game or animated film where characters would be represented as skinned meshes anyway.

References

[1] K. Sims, "Evolving virtual creatures," *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH 94*, 1994.

[2] H. Lipson and J. Pollack, "Evolving Physical Creatures", *Nature* Vol. 406 p. 974 – 978, August 2000

[3] G. S. Hornby, H. Lipson, and J. B. Pollack, "Generative representations for the automated design of modular physical robots", *IEEE* Trans*. on Robotics and Automation*, Vol. 19 No. 4 p. 703–719, 2003.

[4] J. Lehman and K. O. Stanley, "Evolving a diversity of virtual creatures through novelty search and local competition", In *Proc. of the Genetic & Evolutionary Computation Conf.*, N. Krasnogor, Dublin, Ireland, July 2011, p. 211–218

[5] N. Cheney, R. MacCurdy, J. Clune, H, and H. Lipson, "Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding", in *Proc. Genetic and Evolutionary Computation Conference,* C. Blum, Amsterdam, The Netherlands, July 2013, p. 167 – 174

[6] C. Hecker, B. Raabe, R. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real Time Motion Retargeting to Highly Varied User-Created Morphologies" *ACM Transactions on Graphics* Vol. 27 No. 3 p. 1 – 11

[7] J. Won and J. Lee, "Learning Body Shape Variation in Physics-Based Characters", *ACM Transactions on Graphics* Vol. 38 No. 6 207: 1-12, November 2019

[8] K. Wampler and Z. Popović, "Optimal gait and form for animal locomotion," *ACM SIGGRAPH 2009 papers on - SIGGRAPH 09*, 2009.

[9] A.Witkin and M. Kass, "Spacetime Constraints" *Computer Graphics* Vol. 22 No. 4 p. 159 -168, August 1988

[10] T.Geijtenbeek, A Panne, and M. van der Stappen "Flexible Muscle-Based Locomotion for Bipedal Creatures", *ACM Transactions on Graphics* vol. 32. p. 1-11, November 2013

[11] S. Kreigman, "Why virtual creatures matter", *Nature Machine Intelligence,* vol. 1, p. 492, October 2019

[12] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. Efros, "Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity", in *Conference on Neural Information Processing Systems,* Vancouver, Canada, November 2019, p. 1-14

[13] Unity Technologies. "Unity – Manual: Character Joint", Unity3D. https://docs.unity3d.com/2020.2/Documentation/Manual/class-CharacterJoint.html (accessed October 30, 2020).

[14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction,* 2nd ed. Cambridge, MA, USA: MIT Press, 2017, ch. 1, pp. 1-18

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov "Proximal Policy Optimization Algorithms", *ArXiv,* 2017

[16] Unity Technologies. "ml-agents", Github. https://github.com/Unity-Technologies/ml-Agents (accessed October 30, 2020).

[17] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning,* 13th ed. Boston, MA, USA: Addison-Wesley Professional, 1988, ch. 1, pp. 1-26