Physical Character Animation using Machine Learning

Design Document

Team 4

Client/Advisor: Jim Lathrop

Rob Quinn - Project lead, Sim lead programmer, client communications
Joe Sogard - Web lead, Backend programmer
Joe Kuczek - Full stack web, SCRUM master
Luke Oetken - Simulation programmer, Status reporter
Andrew McKeighan - Simulation programmer
Kenneth Black - Simulation programmer, Machine Learning

sdmay18-04@iastate.edu

https://sdmay18-04.sd.ece.iastate.edu/

Revised: 2/20/2018

Table of Contents

Introduction **3**

Acknowledgement 3

Problem and Project Statement 3

operational Environment 3

Intended Users and uses 3

Assumptions and Limitations 3

Expected End Product and Deliverables 4

Specifications and Analysis **4**

Proposed Design 4

Design Analysis 5

Design Specifications 5

Testing and Implementation **6**

Interface Specifications 6

Hardware and software 6

Functional Testing 6

Non-functional Testing 7

Process 7

Results 8

Closing Material **8**

Conclusion 8

References 8

Appendices 9

List of definitions

- Animation - character movements for a game or media
- Keyed animation - animation made by interpolating between manually placed keyframes
- Keyframe - represents a character’s pose at a given frame
- Physical animation - animation where the character’s movement is driven by physics and forces
- Physically based animation - animation where physics were studied or used to create keyframes

1 Introduction

1.1 Acknowledgement

This project will receive necessary assistance from the university to provide us a dedicated server and database.

1.2 Problem and Project Statement

In modern video games there is an increasing need for high-fidelity physical and physics based animations for characters. These animations can be extremely time consuming and expensive to make using traditional methods such as hand-keying or motion capture, and require many professional animators. Also, most existing procedural animation tools only work on human skeletons.

This project creates physical animations for video game characters using genetic algorithms. Characters have realistic 3D physicality and learn coordinated muscle-based motion to satisfy a goal such as creating a walk cycle.

1.3 operational Environment

This is a virtual simulation rendered on the Unity 3d game engine (ref. 4.2.1). The engine creates most of the graphical interface, with our code being written on separate script components written in C#.

1.4 Intended Users and uses

Our intended users are game developers who need animations for their characters. Many games today use physical animations where the movement of the character is driven by physics. Other games that use keyed animations may use physics simulation to make their animations realistic. Animations and especially walk cycles are time consuming and may require a dedicated professional to make them. Our application will take characters and simulate them to learn physically plausible movements which game developers can then use in their game.

1.5 Assumptions and Limitations

Assumptions:

- The cost will be negligible considering we are not working for pay, using free tools, and the web server is provided.
- The simulation can be run faster than real time in order to get the data we need.

Limitations:

- Two semesters to work on project.
- Limited by the amount of training data.

1.6 Expected End Product and Deliverables

End product will be a virtual simulation of character movements using a genetic algorithm/machine learning implementation. The product will be delivered at the end of our second semester in May. We will have the core functionality completed in February.

2 Specifications and Analysis

2.1 Proposed Design

2.1.1 Genetic Algorithm Design

Our genetic algorithm simulates characters to learn physical animations.

2.2 Design Analysis

We researched different machine learning options:

- Genetic algorithms
  - Genetic algorithms are the easiest machine learning algorithm to get started with and should scale well for our purposes.
  - Genetic algorithms with k-best and crossover options
- Unity Machine Learning Agents (ref. 4.2.2)
  - Released in beta after we started, but being evaluated in case we want to switch
  - Uses TensorFlow and GPU processing (ref. 4.2.3-5)
  - May learn significantly faster

Physics

- Physics runs at a stable rate
- Simulation can be sped up safely by bottlenecking at physics (no framerate dependency)
- Muscles
  - Initially, muscles based on motor force
  - Converting to tension based muscles
  - Sinewave driven is a good solution, may add octaves

Website options

The primary options that we have to decide on regarding the website is mainly the language that we will be writing the front and back-end in. Options include Node.js with AngularJS, Python Flaskr, or jQuery with PHP. There are advantages and disadvantages of each, primarily what the developers have more and less experience with.

We implemented a genetic algorithm for the animals to learn to walk. They have muscles, a brain, and a genome to represent their unique muscle values.

The implementation works well and the animals learn within 5 to 100 generations how to walk, depending on the complexity of the animal (number of limbs and joints)

Overall the project is going well and should work with our algorithms. To take this to the next level we have been looking into Unity Machine Learning Agents which we may integrate into the project to make the learning faster and more robust. It uses a powerful machine learning library which we could not implement on our own for this project but may be useful to add.

Animals are currently very successful with their learning and we have found good settings for the physics that are realistic and stable.

The animals are very capable with our current algorithm and with more genetic options they will learn better. If Unity ML-Agents works well for our needs and outperforms our current algorithm we may integrate it.

2.3 Design Specifications

We are using Unity to simulate our project (ref. 4.2.1). We would only need a computer that has the specifications that would allow it to run the simulation in Unity.

We need to be able to simulate at least three different types of animal, each of which will result in a learned locomotion behavior. We also need to be able to simulate the physics up to 10-times faster than real time. It must simulate physics at a consistency for various scales, and run for 1000 generations without supervision. Finally the simulation should be able to take stable evolved genomes and use them in a new branch.

Our website should be able to hold up to 500 different generations in the database. It also should be able to show an updated graph of the fitness score over time. The website must also be visually appealing.

3 Testing and Implementation

3.1 Interface Specifications

Testing of this project will be done on the animal simulation itself and on the client-side website. On the simulation we will manually test to ensure that the algorithm is working as intended. Since we plan for the animals to do poorly initially then learn to do better, it will only be necessary to ensure that they are initially able to move and that data is recorded.

Regarding the website there will be more in-depth testing. Initially there will be unit testing by developers on how data is entered into the database. There will be manual testing on how the website looks, and there will also be unit testing on how data is interpreted on the front end. The majority of these tests will be manual or will take place during development.

3.2 Hardware and software

Testing of this project will be done on the animal simulation itself and on the client-side website. On the simulation we will manually test to ensure that the algorithm is working as intended. Since we plan for the animals to do poorly initially then learn to do better, it will only be necessary to ensure that they are initially able to move and that data is recorded.

Regarding the website there will be more in-depth testing. Initially there will be unit testing by developers on how data is entered into the database. There will be manual testing on how the website looks, and there will also be unit testing on how data is interpreted on the front end. The majority of these tests will be manual or will take place during development.

3.3 Functional Testing

We will have a test suite that will make sure that the algorithms are performing correctly. It will make sure that the animals are being evaluated against the correct criteria. We can also use tools to test the website to make sure that we are getting the correct functionality for layout of the pages.

To test if the machine learning algorithm is making notable progress, we will let the simulation run until the fitness scores remain approximately the same for 200 generations, indicating a plateau. Then we look at the graph to see any areas of positive increase in the score. If there are no areas of improvement then the animal and the test will be reviewed to think about if the animal may be performing to the best of its ability but is physically unable to make progress (for example, a flightless bird trying to fly to 100 meters high).

3.4 Non-functional Testing

To test that the application has consistent physics simulation at various time scales, we will run the simulation at the highest allowed time scale, and again at real time, and compare where the animals end up. If they end in the same position and pose then the physics are consistent.

To test if the simulation can run without supervision for 0 to 1000 generations we will leave the simulation running for however long it takes to reach 1000 generations. For example, a 20 second generation at 10x real time speed will take 20 * 1000 / 10 = 2000 seconds. At the end of the 1000 generation we will see that there are no errors and the algorithm has made progress.

To test if genomes are stable enough to reuse for future branches of generations, we will save the genomes, stop the simulation, and load the genome. The simulation should start where it left off and receive approximately the same or better fitness score.

To test if the data in the site is up to date, we will run a simulation to upload a new genome, then refresh the website and see that the data is there.

To test if the website is visually pleasing and good to compare data, we will populate the data with genomes and compare two genomes we know what to expect as the result, and see if the result is visually apparent in the graph.

3.5 Process

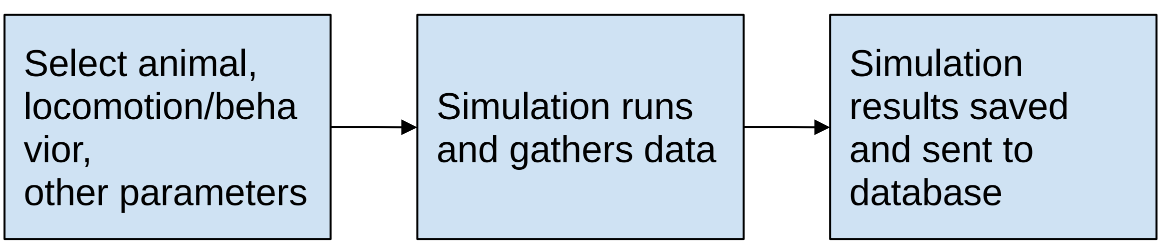
The animal simulation will be tested in the application by

- selecting a character
- select locomotion action
- letting the genetic algorithm run until termination condition
- Get animation and results

The back-end of the website will be tested manually during development. The developer will run the animal simulation for a time and ensure that the appropriate data is being sent to the database.

The front-end of the website will also be tested manually during development. Developer will store fake data in the database then ensure that it is properly reflected on the front end.

Flow diagram



3.6 Results

- Simulation
  - Locomotion
    - The animals learn how to walk, run, turn, sit, and more.
  - Behavior
    - The animal learns what to prioritize to maximize its lifespan
  - Success:
    - Learning makes progress at a reasonable rate and reaches desired behavior within 1000 generations.
    - Desired behavior varies by animal but it is when the animal appears to perform the action to the best of its ability.

   - Failure:
     - The animal fails to learn the behavior or make progress in 1000 generations

- Website
  -

4 Closing Material

4.1 Conclusion

Our project will simulate animals with genetic algorithms for the purpose of research and exploring the capabilities of machine learning within the context of animal locomotion and behavior. We will deliver a simulation app that can run different animals in the environment and a website to upload and analyze the data. Our team has experience with Unity, machine learning, and web, so we are qualified to make this project a success.

4.2 References

4.2.1 Unity

https://unity3d.com

4.2.2 Unity Machine Learning Agents

https://github.com/Unity-Technologies/ml-agents

4.2.3 Nvidia Cuda

https://www.geforce.com/hardware/technology/cuda

4.2.4 Nvidia cuDNN

https://developer.nvidia.com/cudnn

4.2.5 TensorFlow

https://www.tensorflow.org/

4.3 Appendices