Prototyping Walking Robots Using Genetic Algorithms: Specifications

Kevin Beck

Joseph Borsodi

Ryan Romanosky

Kristy Schworn

California University of Pennsylvania

Specifications Document

CSC 490: Software Engineering

6 November 2017

Instructor Comments and Evaluations

Table of Contents

Abstract

This paper explores the analysis of the requirements document and further details the specifications of the project details. This project, as previously stated in the requirements document, is the prototyping of walking robots using genetic algorithms. These algorithms provide unintuitive solutions to problems using iterative processes and rapid revision to develop the best solution. The software outlined in the previous document as well as this specifications document will provide a user with the ability to generate rapid solutions to various terrain types utilizing a genetic algorithm. This document specifically outlines the various constraints on the project, as well as the general responsibilities of structures within both the software and the team structure. This document explores potential complications and describes the application of classes and objects and how those will communicate to achieve the project's goal. Also described within this document are definitions of what will be considered a success of the project and how those will be determined in the future, upon the project's completion.

Keywords: Genetic Algorithm

Description of Document

Purpose and Use

This specifications document explores the topics discussed in the previous requirements documents and develops a foundation for which the design document will be constructed. Through various analytical processes, the requirements document will be deconstructed and reviewed for potential weaknesses or vagueness. The revisions of certain aspects of the requirements document will be expressed in this document as well as further discussions of the production plan the project will continue to follow. This document will provide a strong foundation upon which the design document, and ultimately the implementation of the project, will be set.

Intended Audience

For the understanding of the reader, a prior understanding of the requirements document is necessary. There will be continued revision and changes made to said document, however, the reader is assumed to have comprehended the requirements document in full. Throughout the document, revisions to previous editions will be indicated and explained. Any topics discussed in the requirements document, such as the background information or the exploration of the project domain, are considered to be unaffected by the text of this document. The reader is expected to have an understanding of the technical aspect of the document, as the text will delve into various areas of the project where such understanding is required. Though each topic of technical background will be further explained and explored, a background in computer science or other engineering field will allow the reader to fluidly navigate the topics explored in the writing of this text.

System Description

Overview

The creation of robotics through the use of simulation has many advantages as previously discussed. The project's environment has not considerably changed since the release of the requirements document. The use of a genetic algorithm to iteratively develop rapid prototypes of walking robot designs is the main objective of the project. The client/owner of the developed product will employ a user that navigates the software. The interface will be very user friendly with 4 main sections whose use cases are explored more completely later in the text. The user will operate the software, navigating through sequential tasks. The first task is setting up the terrain.

The software will allow the design of a terrain based on user specifications. There will be various obstacles, and terrain styles for the user to choose from and adjust. These types of terrains will be either entered in by hand, or loaded from a prior run of the program. After deciding on the terrain which will be used in the test, the user moves to the next section, robot specifications.

There will be various constraints upon which the robot will be designed around. Sliders and text boxes will be employed to enter in the data for which the robot is constructed. These constraints will be used by the simulation to maintain a resulting bot that fits the specifications inputted by the user. After designing the restrictions on the robot, the user will continue to the final pre-run stage, algorithm settings.

Throughout the process of developing the generations of robots in search of the solution, the simulation will utilize different methodologies to produce each individual. The factors upon

which these methods are built are managed in the algorithm settings portion of the product. The specifics of these settings are explored further in the text, however, after the user designs the basic algorithm, the user will be able to execute the run function of the program.

The running of the simulation represents the phase of the product in which the user will watch as a robot is designed and tested. This phase of the software is void of user interaction, other than the ability to change the rate at which the simulation is run, or to exit or save the current testing environment. Upon completion of the simulation, the user will either be given a design for the robot along with a message indicating the degree of success for which the robot performed. The following pages will thoroughly explore the development of this discussed application.

Environment and Constraints

End User Profile

The user of the software developed in this project will be required to have basic knowledge relating to the operation of a computer. The software will be user friendly, meaning the user interface will be easily navigated without special training, and the user will not need a thorough understanding of how the software functions. However, the user would benefit from understanding the software more thoroughly as this will allow the user to extract better results from the simulation. To facilitate the meaningful execution of the software, the user should read the manual for the project or have the manual on hand to explain potential difficulties within the technical aspects of the robotics, terrain, and algorithm portions of the software. Also beneficial is a more technical understanding of the construction or operation of the domain the robot is to operate in, as this will provide insight into the potential issues the robot will face with more

certainty. Likewise, a mechanical understanding of the functions the robot is to have is also helpful as it will eliminate potential misunderstandings when developing the constraints around how the robot will be constructed within the simulation. The user also could benefit from a better understanding of genetic algorithms as to best take advantage of the algorithmic offerings within the software. While these aspects of user understanding provide a potential advantage, none are needed for the useful operation of the software as the user interface will provide hover text that explains the functions in some detail, and the user will have visual clues for the sequencing of actions.

User Interaction

The user of the software will use basic computer input to navigate the software. A mouse, keyboard, and monitor are required in addition to a standard modern computer, defined further in the hardware constraints section of this text. Throughout the operation of the software, no additional input devices are needed or supported. Sole keyboard operation is also not available in the software. A mouse is a needed component of operation.

The main user interaction will be navigating a menu while entering/selecting data constraints. A user will boot up the program and be greeted with a main menu. A mouse will be the primary means of navigation to access the 3 mains areas of the menu. Whilst in these areas the data constraints will be entered using either a keyboard or mouse. Once the program has the required data from the user and the run button has been selected minimal interaction will be required for the rest of the program's running time. Mouse input will be required to view the results of the simulation.

Hardware Constraints

The current hardware constraints are unknown at this point, as this is dependent on the complexity of the functions of the software itself. There will be basic restrictions, but the software will run on most modern hardware without issue. Upon completion of various phases of the design document, these constraints will be better understood.

The program will be running physics simulations through the Unity3D engine which can be a very taxing on a computer even though unity is very well optimized. During the initial versions of the software only one simulation will be run at a time as a precautionary step to check performance and set a baseline. If it is determined that the simulations can be run on multiple setups with ease the number of simulations running at one time will be increased to help decrease the total running time of the program. It may be the case where an entire generation of robots can be generated and tested at one time on a moderately capable machine, which in return will allow the program to run as fast as it can without using a form of time acceleration within the simulation. Running multiple simulations at one time is a more effective solution to speeding up the programs overall run time because time acceleration can have negative side effects when calculating physics within the engine.

Software Constraints

The software created in this project will be developed within the Unity3D engine which has the ability to export the project to a plethora of different operating systems. Unity support on desktop spans across Windows, Mac, Linux, and SteamOS. The ability to build a version for modern consoles is a possibility, as well as mobile and Augmented Reality / Virtual Reality. This aspect of Unity3D is advantageous as it provides a larger number of platforms on which the

software will run. During the development process, all users will have access to the Unity3D development environment. Each seat of Unity3D is free for non-commercial use. This allows the development team to all work concurrently on the development of different aspects of the software.

During the creation of this software the team will learn to use the programming language C#. The main deciding factor to use this language was that it is compatible with Unity and will allow the use of object oriented programming. The software will be developed using many classes so it is vital the integration of these classes and the physics simulator is simple and straightforward, something C# provides.

To aid the development and creation of this software the team will learn and use Git through GitHub as a version control system. With the use of Git, it will be possible for team members to work on their individual portions and merge them back into the main "branch" of code easily without the hassle of sending individual files around. Version control will be seamless as new versions will automatically synchronize between the programming environments each member will have. In the event that a new version has faults and affects the overall stability of the software it can be easily reverted to a prior working version.

Time Constraints

This document, the specifications, will be completed no later than the 7th November 2017. The following text, named the Design Document will be constructed and completed by the end of the year 2017. Alongside the construction of the design document, the development team will explore the Unity3D environment. Under the direction of team member Kevin Beck, the team will learn the basics of the creation of UI elements, as well as physical simulations

within Unity3D. The mastery of the basic operations of Unity3D will precede any implementation of the project. Also, to be studied and evaluated is the understanding of the aspects of genetic algorithms utilized in the project. Beginning at the start of the spring semester 2018, the team will begin the implementation of the project. The strict deadline of the completion of the project is the end of the spring semester 2018.

Other Concerns

Concerns with specifics to the project itself consist of two main areas, implementation and continuation. The implementation of certain constraints within the software may prove too strict or the opposite may prove true. If the constraints are too strict, the design of the robot may not result in a diverse population of robots, whereas if the constraints are too loose, the population will be nonsensical and fail to achieve meaningful locomotion. It is the team's top priority to develop the software in a way that allows for these constraints to be adjusted throughout the stages of development. The other concern is the expandability moving into the future. Some aspects of the project will almost certainly require hard restrictions of the software. This ensures that any future development, or substantial changes to the existing software, will require a thorough rewrite of portions of the algorithm or interface. It is important moving forward that the team maintain a strong sense of maintainability in the construction of the portions of the software.

Acceptance Test Criteria

Testers

In order to best evaluate the success of the project, different groups of people will be testing the software, each with their own defined set of objectives. The 2 groups are as follows:

Group 1: The first testing group will be 25 individuals from non-technical majors on California University's campus. With no input from the development team, the individuals will be asked to test the software without any knowledge of the software's purpose. These individuals will attempt to navigate the product, successfully reaching the run page without issue. A poll of this group will be focused on the usability of the software from the perspective of a first-time, uninformed user. This will best test the intuitiveness of the interface as well as flow of the information seamlessly from the user's perspective.

Group 2: The second group will consist of 25 individuals with a technical background. A similar test will be performed as in group 1, however, group 2 will be informed of the software's intention. This allows the user to have a good grasp of what the functionality of the software is as well as the intended goal. After allowing the users from this group to test the software, they will have polled on their use scenario. A secondary score of whether or not a functional robot design was created within this group's testing time will also be indicated in this evaluation.

Criteria for User Acceptance

The criteria for user acceptance will differ based on group. Group 1 will be given polling questions related to the accessibility and simplicity of the programs design. The questions will be worded as statements and answered on a scale of 1-5 with 1 indicating disagreement, and 5 indicating full agreement. The four questions for group one are as follows:

Q1: The program was easy to navigate from start to finish.

Q2: The user interface was intuitive and easy to understand.

Q3: The program explained what each setting controlled.

Q4: I think anyone who uses a computer would be able to use this program.

A score will be calculated on the 4 questions, and the average score of each question will be no lower than 4 for any question. Any score that is less than 4 will indicate a failure to accomplish the goal of a simplistic and intuitive interface.

The criteria for Group 2 will be similar, but more extensive. The primary goal of this group will be the successful operation of the software as well as the successful creation of a robot. These users will use the software with the instruction to create a robot that will navigate a terrain of their creation. The users will operate within the constraints of the software to create the robots. If the testing results in the successful creation of a robot at least 80% of the time, the goal of creating robots that exhibit locomotion will be considered a success. A failure to reach an 80% success rate indicates a failure to properly constrain the user's choices to provide a platform on which successful robots will be designed. After the conclusion of the testing, the users will be given a similar questionnaire as group 1. The questions will consist of each of the 4 statements that group 1 was given, with the addition of the following 3 questions, all being evaluated on the same 1-5 scale.

Q5: The software is adaptable to meet the constraints of many different environments.

Q6: The software is self-explanatory and easy to understand.

Q7: The software did not have any technical issues, and functioned as I would expect.

These questions will also be evaluated with the same criteria of the average score being 4 or higher for each question. A score of less than 4 on Q5, Q6, and Q7 will indicate a failure to create an environment suitable to technical user use.

The testing constraints may be altered in the future to include other aspects of the project, but at the time of this writing best represent the goals of the development team.

Integration of Separate Parts and Installation

The installation for this program is fairly straightforward. On the project's website, separate links will be provided to download our software for the supported systems: MacOS, Windows, and Linux. The Windows version will include an executable (.exe) file and a Data folder, which contains the resources used in the application. MacOS user will receive the standard .app application bundle containing all of the relevant files. The Linux version will include a binary file (.x86) as well as a folder containing data resources. Although there will be a Linux version, the decentralized nature of Linux distribution will mean some users might run into issues running the binary. Unity is the platform being used, meaning any user running a Linux distribution and version fully supported by Unity should be able to run the software without any major issues. Similarly, older versions of Windows and MacOS/OS X might not be supported, either due to lacking Unity support, having an older, unsupported operating system, or simply due to hardware constraints. A list of supported operating systems and builds will be included on the same page as the download link. This should help alleviate issues arising from users inquiring about support for their OS. A benefit to using Unity is the fact that the program will not need any type of "installation process" aside from copying and pasting the relevant folder or .app file. This makes the installation of the program extremely easy and cuts down on errors that could emerge from updating or reinstalling the software. A user will be able to delete the entire application and as long as the Data folder—or wherever the user has chosen to save their files—is kept intact, no saved data will be affected.

Installation on possible future OSes depends on each operating system's ease of publishing. Console versions will have to go through each respective console maker's app store

submission process. Mobile versions will have to go through a similar process for their respective operating systems. Once approved, installation should be as any app is to install (i.e., tap a button to install). However, as a small team, working out bugs in our current supported operating systems will take up a lot of resources. The universality of Unity and its ability to distribute on a wide variety of operating systems helps to ease this burden as the program scales up its supported OSes, but for now versions for other operating systems will remain in either alpha or beta builds, or only exist as a possibility. Performing maintenance at each time a feature is added may not be worth the small amount of resources available, especially for a platform that might make up less than one percent of the user-base.

A user manual will be included both on the download page as well as in the application folder itself. The manual will be designed for both technical and non-technical users of the application to take full advantage of its features and customizations. It will include a first-time setup installation guide so that new users can get to know the features of the program as they run it. A quick guide for each feature will also be included in the manual. For more technical users, the manual will go over how to finely tune each feature (e.g., how to adjust the settings to represent certain areas on Earth). A FAQ section will live at the end of the manual. This section should be as comprehensive as possible—without entering full technical support—to cut down on the amount of user support requests received. The manual should also include any relevant contact information as well as any information a user might need to troubleshoot installation issues (e.g., unsupported system issues).

System Modeling

Functional: Use Cases & Scenarios

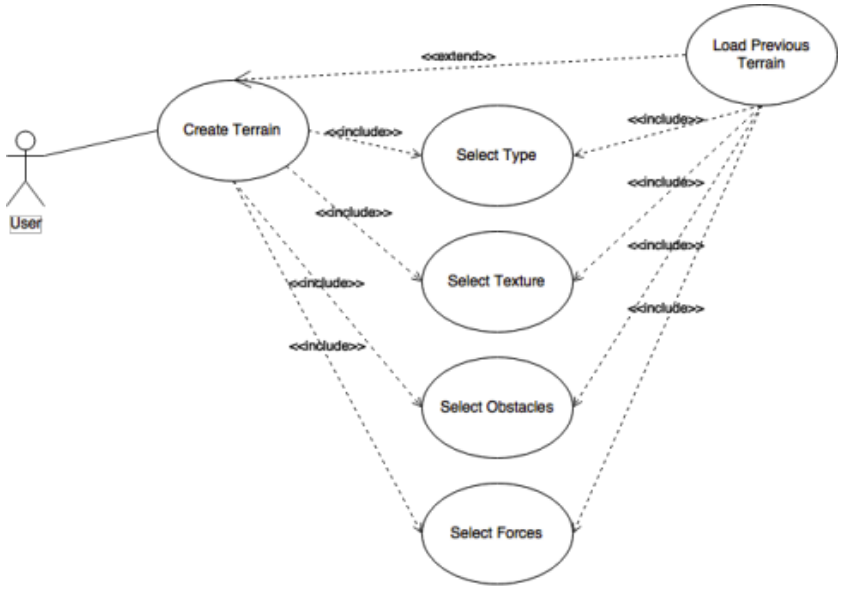

Figure 1. Terrain Settings Use Case

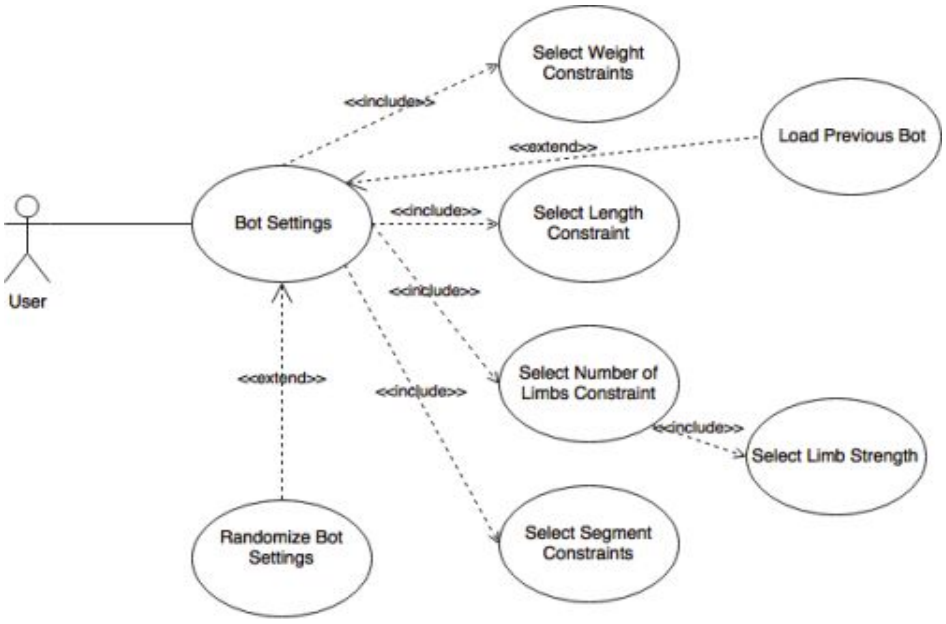

Figure 2. Bot Settings Use Case

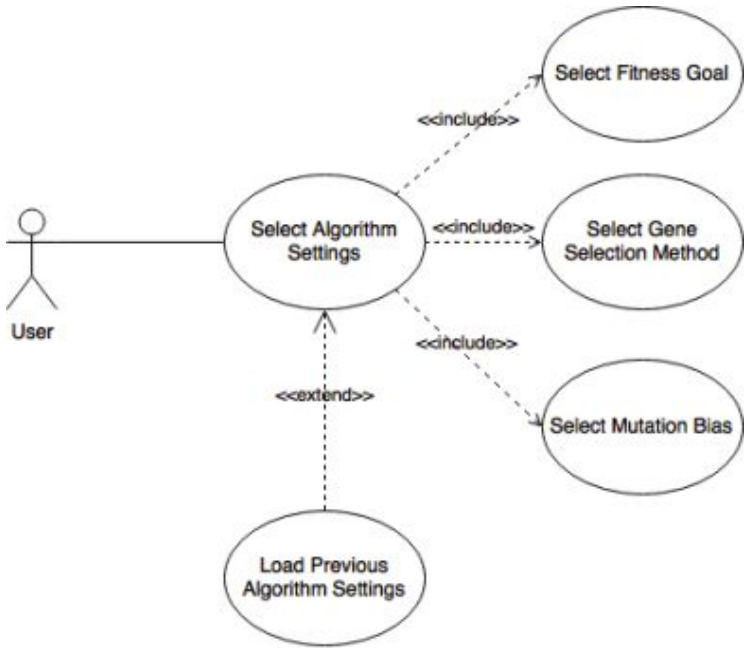

Figure 3. Algorithm Settings Use Case

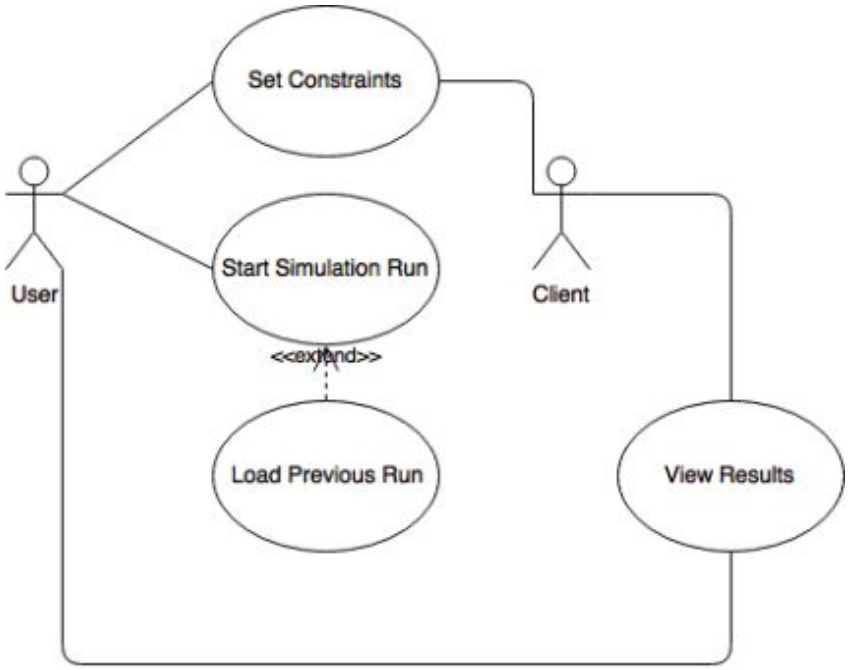

Figure 4. Simulation Use Case

Classes & Class Diagram

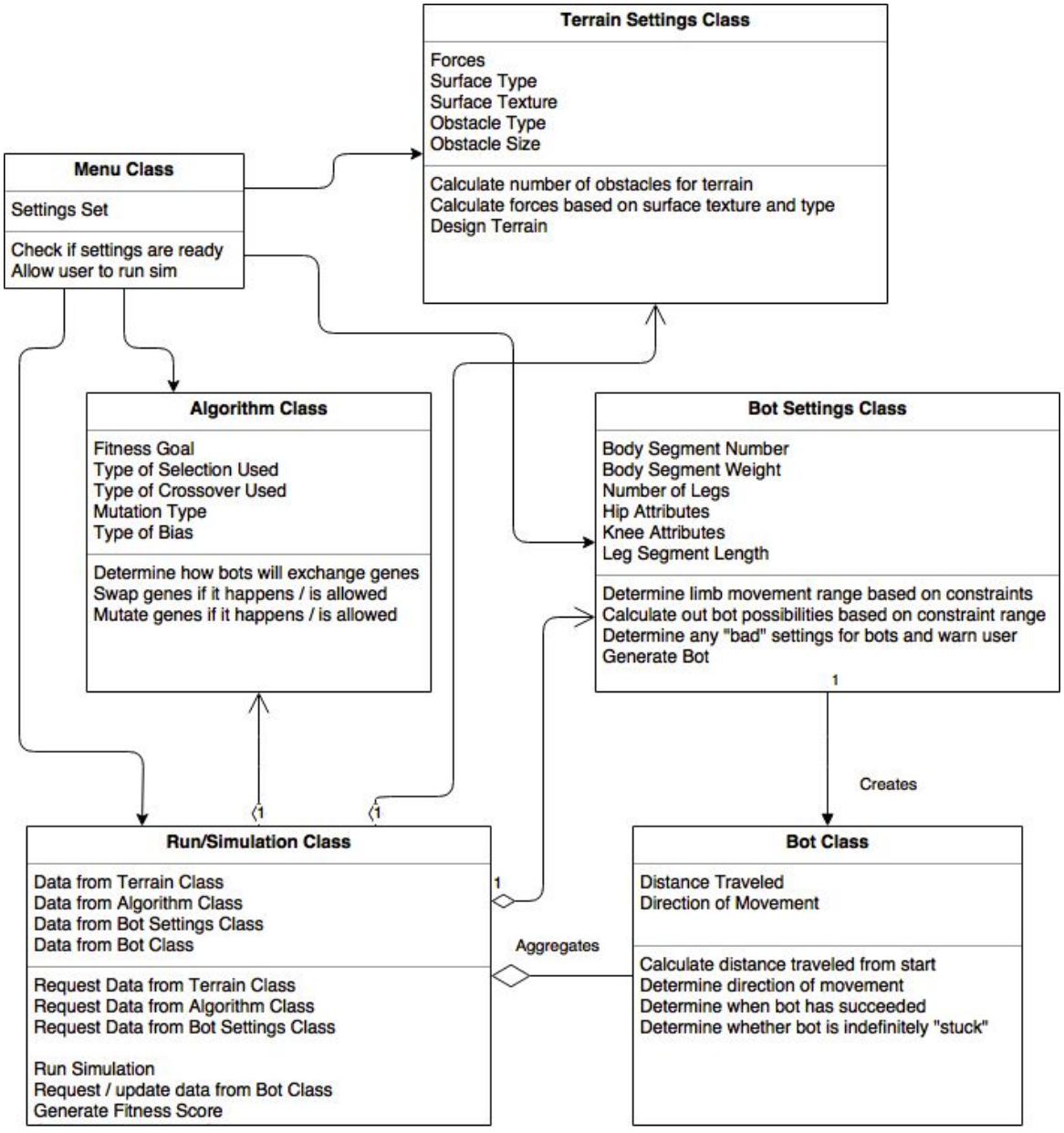

Figure 5. Class Diagram of Entities

Menu Class

In our current setup, the team is using six classes for the program. The first class, the menu class, contains all of the pre-setup selections. The user will have the ability to select, from the menu, the option to edit either the terrain, the bot, or the algorithm details. This class will check to see if the other classes have been self-validated before allowing the user to run the simulation.

Terrain Settings Class

The terrain settings class is the class that contains all information relevant to the creation and editing of the user-selected terrain. The forces applied to the bots-at-travel, such as gravity, friction, and wind velocity, will be edited, stored, and calculated in this class. The terrain settings class will also keep track of four important variables: terrain type, terrain texture, obstacle type, and obstacle size. Terrain type and terrain texture will be used in conjunction with applied forces to change the speed at which the bot can move. Obstacle type and obstacle size can also be edited here. Once the user has selected their terrain constraints, the class will send a validation to the menu class.

Bot Settings Class

The Bot Settings Class contains all of the information regarding constraints for "building" a bot. Although the Bot Settings class doesn't actually contain the bot object itself, it does contain the building block or genes to create it. These settings—number of limbs, limb length, number of segments, segment weight, hip attribute, knee attributes, and limb strength—will be used to actually generate the bot object. This class will calculate the ability of the bot to move its legs, both for simply walking and for walking *over* obstacles based on the

stored variables. It will also notify the user if the constraints selected will produce a "null bot" (i.e., one that has no chance of working out) and give the user the ability to change the constraints if they wish. Once the bot settings class has completed its functions, it will generate the Bot Class, discussed in the next paragraph.

Bot Class

The Bot Class is the bot object itself. It is created by the Bot Settings class and will keep track of the distance traveled by the bot as well as the direction traveled by the bot. This information will be used and accessed repeatedly by the Run/Simulation class to check if the bot is moving the correct or incorrect direction and how far the bot has traveled. Using the stored distance traveled data, it will be able to determine when it reaches the end of the terrain and thus, determine success. It will also keep track of whether the bot has failed so that the run/simulation class can check for validation.

Algorithm Settings Class

The Algorithm Settings Class will keep track of data relevant to the algorithm. This includes the fitness goal, the methods of gene swapping and mutation, as well as the type of bias used in the algorithm. Based on the constraints selected by the user, it will determine how each successful generation of creatures can pass on their genes. It will also determine how often the genes can and will randomly mutate.

Run/Simulation Class

This class is the most comprehensive of the six classes. The Run/Simulation Class aggregates data from the bot settings class, the terrain settings class, and the algorithm settings class. From the bot settings class, it receives the bot constraints. From terrain settings, it receives

the calculated terrain data and model. From the algorithm settings, it receives the completed modified algorithm. Using this data, it begins mapping out progress of the creature. The Run/Simulation class also repeatedly aggregates data from the bot class. This data is used to inform the sim class whether the bot is still moving in the correct direct—forward—and how far the bot has moved. It will check for errors, such as the bot moving backwards and traveling a negative distance, and will take corrective measures—such as ending the sim early—so that computer resources are not wasted on faulty bots that never had a chance. At the end of each run, the class will generate a fitness score of that bot to allow the user or client to compare different bots and their respective runs. The data from the run can be exported out for the client.

Menu Flow Concept Diagrams

Note: Menu flow diagrams were developed within the team as a means to help better visualize the system for the client and designers.

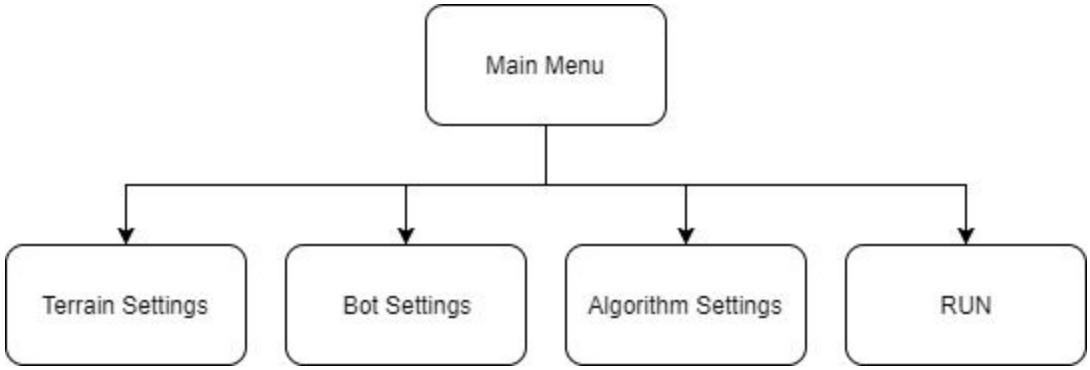

Figure 6. Main Menu

Whenever the user first opens the program they will be greeted with the main menu. As shown in Figure 6, the user will have 4 selections to choose from: Terrain Settings, Bot Settings, Algorithm Settings, and Run. Terrain Settings will be the first area the use will enter to select details of the terrain their bot will traverse. After the terrain has been chosen the next area the user will visit is Bot Settings. Once in the bot settings menu they will choose the constraints that the robot must follow to fit their needs. Finally, the user will have the ability to enter the Algorithm Settings to choose how the program will create and design robots. Initially the Run button will be greyed out until the user has properly selected all settings required to begin testing. If the user tries to run the program whilst missing some settings, they will be notified of all unfulfilled requirements. Once all requirements to run the program have been created the user may then run the program.

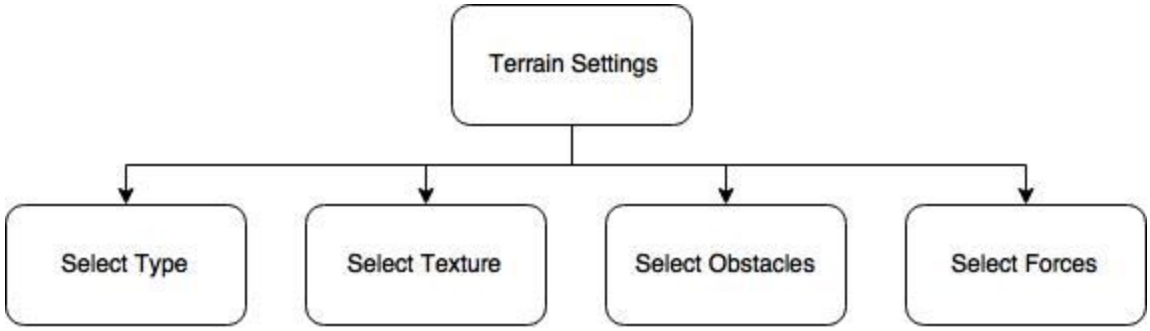

Figure 7. Terrain Settings

The first settings area the user will enter once opening the program will be the terrain settings which will allow them to specify the environment that the robot must traverse. Once in this area four settings must be selected: Type, Texture, Obstacles, and Forces (See Figure 7). The first terrain constraint to be specified is "Type" which represents the geometry of the terrain the robot must travel forward on. Examples are as follows: A flat terrain, A hill with a 45-degree incline, or a stair like pattern with 1-foot landings. After a type of terrain has been selected a texture for the ground will be chosen. The next setting is obstacles that the bot will run into while traversing the terrain. The user will select the type of obstacle followed by the size and frequency of these objects. Finally, the fourth and final setting for the terrain category will be forces the environment will have on the bot. These settings include drag, gravity, and wind forces that the bot must overcome. During run time these settings will be used to generate a random environment for each generation to traverse.

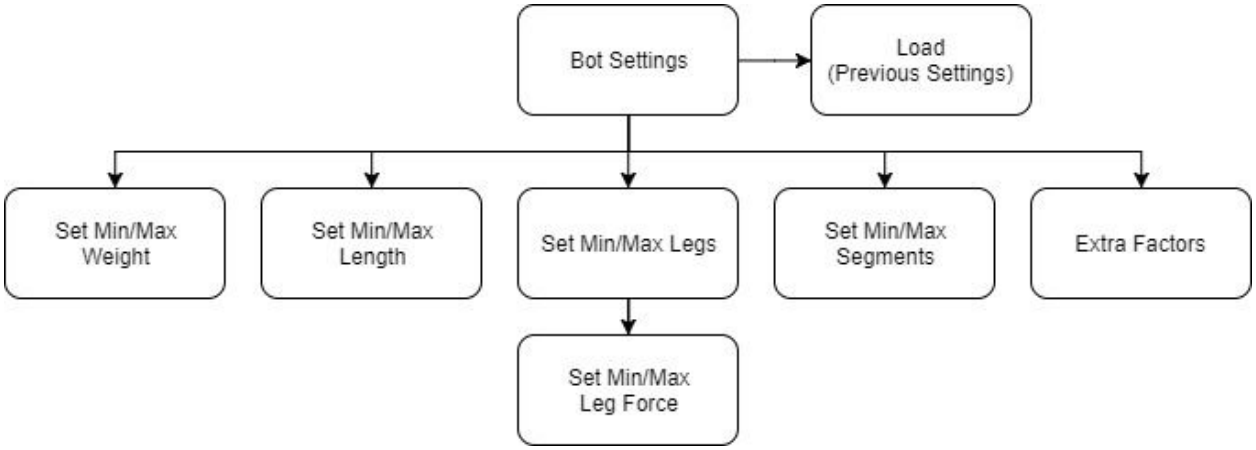

Figure 8. Bot Settings

The second settings area the user will be directed to is Bot Settings (See Figure 8). The intended use of this area is for the user to select the constraints that the program must follow while creating designs during run time. If desired a previous settings page can be loaded to rerun a previous simulation for different results. During regular usage, new bot constraints will be selected each time the program is run. Bot settings will be easy and fast to select. All settings on this page will use sliders along with checkbox buttons to increase ease of use.

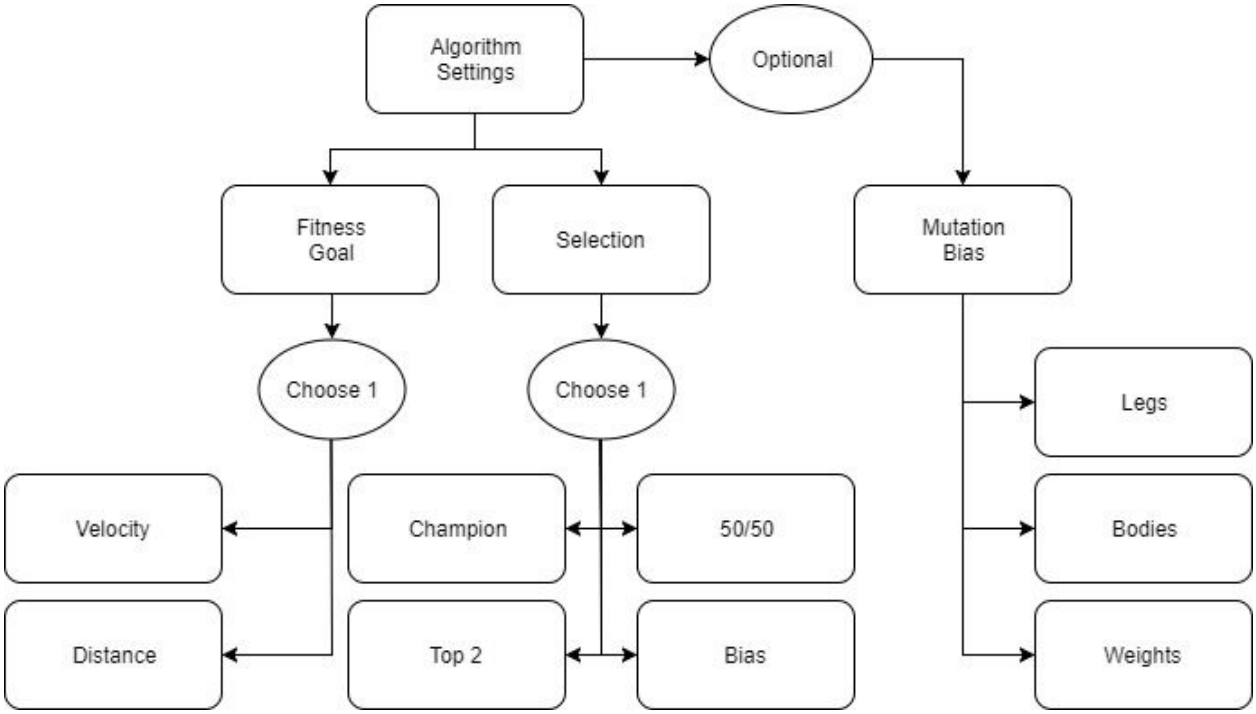

Figure 9. Algorithm Settings

The final and perhaps most intimidating area is Algorithm Settings (Figure 4). To help make this area user friendly only two settings must be defined, how the bot is to be ranked (fitness) and how bots are chosen to be seeded into the next generation. A third optional setting will be included that will allow the user to select bias during mutation. During runtime each bot will be given a fitness score to determine if it was successful or not. Fitness can be defined as either average velocity or maximum distance traveled but not both. After a generation has been created and ranked the next generation must be seeded. To determine which bots will be used as seeds as selection processed will be defined. Four choices will be given to the user and with the ability to select one.

- Champion - Only allow the best bot from each generation to be seeded into the next.
- Top 2 - Take the top two bots and seed them into the next generation.

- 50/50 - Take only the top 50% of the population to seed the next generation
- Bias - Preferences to certain body parts or values to be favored in fitness

A third optional setting will allow the user to select a mutation bias that will occur during runtime. Each robot in a generation is created randomly by following the constraints set in the bot settings area. Mutation Bias allows the users to push a bot towards a specific design such as awarding more fitness to designs that have longer legs than those who do not.

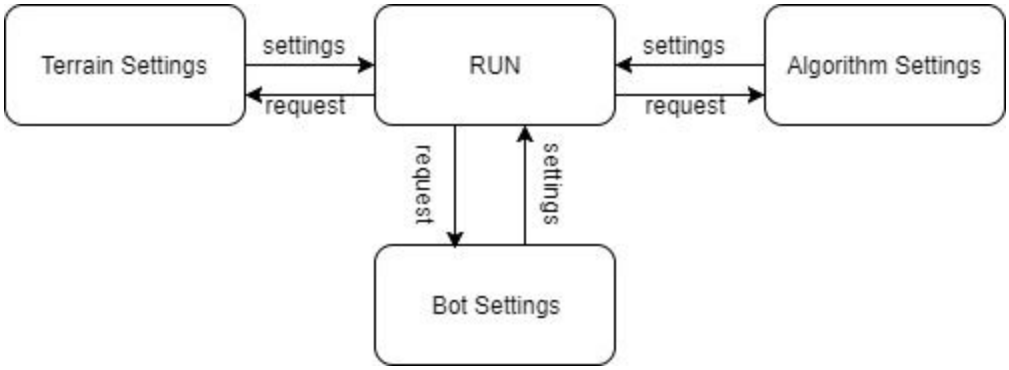

Figure 10. RUN

The status of the run button depends on the three settings areas (initially disabled). Once the user has visited and defined all settings the run button will be enabled allowing the user to begin running the simulations. As soon as the run button is pressed it will send a request to the three settings areas to collect their current settings and begin simulations (Figure 5). Note that once the run button has been selected the user will no longer have access to any of the settings areas and the following simulations will be run with current settings. A warning window will be displayed to inform them all settings are final once simulations begin. If the user accepts the warning windows continue button the program will transition into the simulations phase.

Dynamic: Statechart

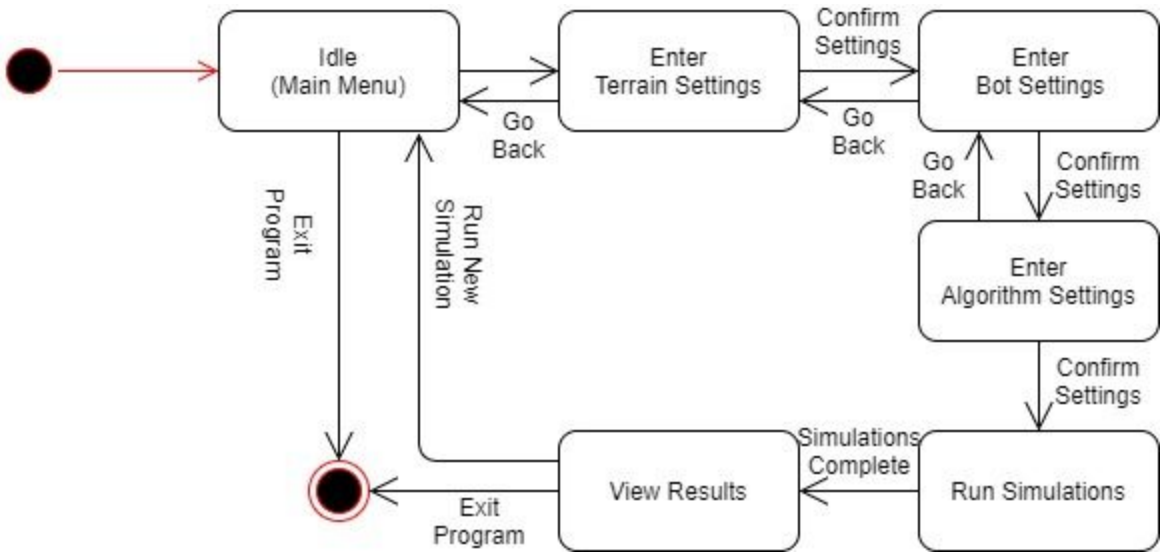

Figure 11. State Diagram

States

Idle- Initial state of the program (Main Menu), software will idle here until the user decides to begin entering constraints or exits the program.

Enter Terrain Settings- Second state of the program, user will remain here until they enter all settings relevant to the terrain and confirm or decide to go back. If the user decides to go back all settings will be lost.

Enter Bot Settings-Third state of the program, like the “Enter Terrain Settings” state the user will remain in this state until all settings are entered, and they confirm their selection. This state will also have the ability to go back to the previous state; however, all settings will also be lost.

Enter Algorithm Settings- Fourth state of the program, exactly the same as the previous two states. Algorithm settings will be entered during this state until the user has confirmed the settings. This state will also have the ability to go back to the previous state; however, all settings will also be lost.

Run Simulations- Fifth and most important state of the program, during this state the program will pull in all the previous data entered by the user and begin running simulations in the 3D physics engine Unity. The simulations will continue to run until the final generation of robots have been created and tested. At the end of this state it will automatically transfer to the "View Results" state.

View Results- The final state which will be automatically entered once the simulations have finished running. The user will have the ability to view the results from the simulations in this state. When they are finished they can either exit the program which is the final state or return to the main menu to run another set of simulations with new settings.

Events

Confirm Settings- User has confirmed their settings, program will move to the next state.

Go Back- Returns the user to the previous state, used for changing previous settings.

Simulations Complete- Automatically triggered once all simulations have completed.

Exit Program- Exits the program and returns the user to their desktop.

Transitions

"Confirm Settings" and "Go Back" are user activated events that trigger the transition between settings states. "Simulations Complete" is triggered automatically to transition the program to the view results state. The "Exit Program" event will cause the program to end.

Dataflow Diagrams

Components / Tools Needed

Any functioning, modern day computer will have the ability to run this program. If the computer is updated and virus free, no problems will occur with the program. Any desktop or laptop that is up to date will have no difficulties running the program. The team will also need a monitor, mouse, and keyboard or trackpad to develop and implement the project. Development of the program will require the team to use a physics engine.

Communication throughout the entire project is extremely important to the success of the project. The team will have frequent face to face meetings throughout the entire process to share ideas, make modifications, and to further develop the project. When ideas need to be shared prior to an in person-meeting, the team will use the communication application Discord. This application allows for private groups to be created. Each member is part of the group created. Discord allows for group messaging and group calls. This will be utilized each time a meeting cannot be done in person. Discord is mobile friendly, can be downloaded, or used straight from the web. This makes it easily accessible to the whole team under most circumstances. Email will also be utilized as necessary throughout the creation of the project. Another major part of the collaboration process is where the coding will be done.

Every group member will use visual Studio 2017 Community to write the code for the program. This edition is free for students. Each group member will install Visual Studio to their desktop or laptop. This will be the main application used for writing code by every member, as it is easier for the whole team to be familiar with the same application. Small pieces of code may be written in any general text editor, but will be implemented into Visual Studio before it is

expanded on or edited by the team. While the coding will be primarily done using Visual Studio, the physics of the project will require the use of a physics engine.

The team will be using Unity as the physics engine to develop and implement this project. All four team members will use Unity to create the program. Each team member will have their own installation to work with. This will allow for every team member to dedicate to the project as a team as well as individually. The free edition of Unity is what each team member will use, as this edition is all that the project requires, unless it is being sold. The team will also need a way to share new thoughts, ideas, and creations to the other team throughout the development of the project.

In order to collaborate as a team from different locations, the hosting application GitHub will be used. GitHub works directly with Visual Studio for the purpose of collaboration. Each team member has downloaded and created an account with GitHub. A private group project was created on GitHub and every team member is a part of it. Using a private group ensures that only the four team members creating the program will have access to any code that is shared. GitHub will allow the team to send edits made to the program to each other. Before a change or advancement is finalized, each team member will have to approve the changes within GitHub. This will keep all team members aware of any progress made to the program. Furthermore, GitHub will be used to detect errors within the project. If a flaw is found, a team member can send in a request to change the code, notifying all group members of the problem. GitHub will allow different version of the project to be stored and kept in order should they need to be referenced or used in future developments. While this is a beneficial tool for code sharing, the team will also need ways to collaborate on documents.

Several documents will be created for this project. The team will be using Google Docs as a way to collaborate on these documents when unable to meet in person. Every team member is able to edit the document at the same time. Changes made can be tracked and comments can be added to any section written. Google Docs allows users to see who is working on what section and when the document was last changed. The Google Doc may only be accessed by users who have obtained the link to the document. Once a team member creates the document, they share the link with the rest of the team. Using Google Docs is the ideal way for the team to work on documents together as changes made can be seen in real time. The document can be edited anywhere that has internet connection and can even be accessed using mobile devices.

It is vital to the success of the project that each team member has a modern day computer. The computer will be needed for every process involved in creating this project. With a computer the member will be able to communicate with the rest of the team, write and share code, and install any applications needed. Once every application needed is installed and documents are written, the team will be able to start coding. The team will stay in contact throughout the entire process, making changes and advancements as a whole.

Appendix I : Glossary of Terms

Bias: Preferences to certain body parts or values to be favored in fitness

Environment: In the terms of this project the environment will be the generated terrain the robot traverses.

Fitness: The measure by which a robot is scored within the genetic algorithm. Depending on the application, this could be the distance or speed the robot covers or any number of other measures decided by the user of the software.

Genetic Algorithm: An algorithm in which some form of evolution is applied to a genetic code leading to iterative generations.

Mutation: The alteration, addition, or removal of a gene from a genome. This occurs between different generations of robots.

Physics Engine: An environment in which physics and collisions are solved and simulated given certain parameters.

Unity3D: Unity is a specific physics engine capable to provide a platform on which the software will be constructed.

Walking Robot: In this domain, a walking robot is represented by any number of robotic designs, all of which utilize at least 1 pair of legs. These robots have various applications and the term is intentionally vague to include all potential applications of this project.

Git - A version control system used within a group to aid software development.

C# - An object oriented programming language.

Appendix II : Team Details

As the workflow leader for this specifications document Joseph Borsodi was responsible for planning team meetings around the schedule of the team. Frequent meetings were needed to make sure everyone was properly working towards the goal as well as being kept up to date on what was required for this document. Joe was also responsible for creating the Menu Flow Concept Diagrams related to the main menu and the description of each, figures 6-10. The software relies on the user's ability to easily navigate the menu and settings areas easily without confusion. The menu is one of the most important parts about this project, without it the program would be unusable. A proper description of the menu and its parts was crucial to clarify its intended purpose to the client and designers. He was also responsible for creating the dynamic state chart Figure 11 along with the description of the states, events, and transitions. Joe also helped discuss the software and hardware constraints of the project.

Kristy went into detail concerning the components of the project. She covered topics such as needing up to date hardware and software. It was made clear that any modern-day computer would be useable for every step of the project. The tools needed for communicating, sharing, and creating were expanded on. Every installation that the team needed was explained. How each application will be utilized throughout the process was brought to the reader's attention. Grammar and spell check errors were corrected as the paper was written. The whole team worked together to give the document a final revision.

Kevin initiated the document and handled the writing of the description of the document including the Purpose and Use and Intended Audience sections. He created the abstract and also outlined and created the System Description sections. These sections describe the overall

direction of the project as well as the steps taken in the past. The Environment and constraints sections describe potential downfalls to be anticipated and avoided moving forward. He described the testing conditions for determining the successful outcome of the project, as well as devised the questions to be asked for surveys. These surveys will be deployed after the project's completion and will be used in the future to determine the levels of success of the project.

Ryan created the class diagram for the entities and the use case UML diagrams used in this document (Figures 1–5). He also wrote the descriptions of each class in the program, how they interact with each other vis-à-vis the class diagram, as well as the functions of each class and how the attributes are stored. The Integration of Separate Parts and Installation was also handled by Ryan. This section describes how the program can be easily installed on many operating systems and why Unity is a good platform on which to distribute the program. It also covers the user manual for this program. A good program starts with a good user manual, and this should be no exception. Given that the program will be used by both technical and non-technical people, having the user manual give walkthroughs for both basic and more fine-tuned customizations will greatly increase the usability of the program.

Appendix III : Workflow Authentication

_____	_____	_____
Printed Name	Signature	Date
_____	_____	_____
Printed Name	Signature	Date
_____	_____	_____
Printed Name	Signature	Date
_____	_____	_____
Printed Name	Signature	Date