Evolving Virtual Creatures

Ishmail Qasim – 180021745

Computer Science BSc

08/2021

Contents

Acknowledgement

I would like to thank my supervisors for being supportive during this project. To Dr Peter Lewis, for helping me learn to learn and enjoy the process, and thinking innovatively to address issue seemingly unfathomable. To Dr Megan Robertson, for helping me decide on a project to invest myself in and for helping me materialising a feasible project under the stresses and feeling directionless.

I would also like to thank the University for preparing me with the expertise, mindset and knowledge to enable the progress of this project.

Abstract

Evolving Virtual Creatures (EVC) is a project designed to explore the process and function of evolution by simulating a virtual work space to design, implement and experiment with evolution through fitness evaluation and assignment redefinition. This project is inspired by Karl Sims', Evolving Virtual Creatures project and will investigate his implementation, understand his reasoning and trial alternative evolutionary concepts. This is in order to understand and develop a workspace that can effectively trial designs from innovators and determine their effective fitness whilst returning far less costly solutions. All discussion will reference the research of Karl Sims, and the provided software documented below.

Introduction

Source Repository: https://github.com/Iqasim94/180021745-FYP-EvolvingVirtualCreatures
Contains: Source Code, Diary Report, Demo Run&Results, Project Poster, Project Definition Form, Term 1 Report, Final Report.
Test Classes: jbox2d-testbest > src/main/java > org.jbox2d.testbed.tests
Main Class: jbox2d-testbest > src/main/java > org.jbox2d.testbed.framework > TestbedTest.java
Evolver Class: jbox2d-testbest > src/main/java > my_Code > Evolver.java
Code Originality: The source code for the physics engine falls under the ownership of Daniel Murphy, licensed under the BSD-2-Clause. All authors are correctly identified for the code written, with adjustments made or code copied where the author is not Ishmail Qasim.

EVC is about experimentation with computational intelligence in order to design a device (creature) that may iteratively improve efficiency towards completing a task. Former prospective development goals focused on enquiring into the efficiencies of human mechanics and bodies in order to complete tasks. Is the human body the most efficient implementation of a body that can carry out tasks that are becoming invariably more complex and difficult in the human world? Are the current technologies surrounding the production and industrial world, the best use of electricity or can complete tasks with the lowest cost (cost will be referred to by fitness from this point on)? What about AI in interactive software or games? Are they the most realistic and competent they can be?

Due to time and complexity restraints, the scope of experimentation has faced reduction and focuses solely on the development of a vehicular object and aims to develop on the idea of producing a counterpart to a creature that will simulate evolutionary cycles in a modifiable test space. Later objectives would strive to return to the original aim. This will be discussed further in this report.

One of the many faces of this multi-faceted project, is to produce a virtual product that proves the usefulness of the evolutionary mechanisms that will be used and further discussed later in this report.

The nature of this software is designed to provide dialogue to the concept of evolution and morphology as discussed in Karl Sims' research project (Sims, 1994) inspiring this one, though a far simpler project.

Generally the main intention of this eventual software is to enable researchers from any industry to design a concept and test the usefulness. These could be engineers from production lines looking to move away from traditional assembly line robots with singular use and move towards multi-purpose, highly efficiently, highly versatile and reliably safe robot designs. This could also be construction companies looking to test structural concepts for new buildings and the tools uses to produce these concepts.

The state of the software at the time of this report will not be able to accommodate this function. This is due to the lack of physical, logistical and schematical (i.e. understanding how a creature should move but not mechanically how without an engineering insight.) data due to the nature of the graphical simulations that harness pre-existing, non-specialised physics engines.

Originally, the proposed end product would have been a 3D virtual space exhibiting what was briefly mentioned above. This included creating a virtual testing space in order to trial the creatures; the creatures that will inhabit this virtual test space with the ability to initially move along the x and y plane, with later developments including the z plane; A variety of simple tasks the creatures will be subject to in order to observe their efficiency and the improvements where they exist. All relying on producing an evolutionary algorithm to house the virtual development of a virtual organism that is based on the principle of random search – with effective implementation, this will ensure that no single strain of genetics will be repeatedly generated per case.

This will be demonstrated by the virtual creature's performance on 3 predetermined tasks; Travelling from point A to B successfully and in a timely manner; tracking efficiency of the creature as the time frame shortens. Once complete, obstacles will be introduced to make travelling from A to B more difficult but will ultimately display iterative improvement in path finding and creature evolution. Once these two tasks are demonstrated, the final task will be for the creature to collect an object from point B and travel to point C. This in concept will hybridize the creature in order to create an efficient multipurpose creature without compromising on speed or efficiency i.e. path finding and object transportation in place of one of the two in this concept.

Once project development was underway, designing a software that completed any predetermined set of tasks in a 2D space alone, would prove far too complex within the scope of the project. Many compromises would have to be made that would affect the realism of the tests due to ignoring potential genetic components. For this reason, the project scope was reduced to solely evaluating concepts in a 2D space. It made more sense in terms of exploring evolution to flesh out a 2D development cycle than a rudimentary 3D one within the available time frame. This will be discussed further later in the report.

In the end, the extent of the project devolved into experimenting with fitness evaluation of a car object; genetically evolving specific components including the wheels, contact forces, friction and more. This was done by manipulating the existing virtual work surfaces of jbox2d in order to design a virtual testing map with the car beginning on one side of the map and an objective waiting at the other side. The car will have to drive there and contact the objective as soon as it can, in currently any fashion that gets the job done, including through aggressive backflips. The fitness in this program is that of time; the faster, the fitter. The other mechanisms of iteration that exist so that a run does not

continue infinitely without evolving is by completing the run if double the best time has been reached by the timer. It is assumed that if a creature requires double or more of time to complete the course, then it is not a competitor. However, previews of the program show that having the highest speed is not the best outcome in its current state. Often times, what is previewed is that plausibility is traded for speed, completely in tangent to Sims' approach.

Although extremely rudimentary, this serves as a bouncing board to further develop understanding on the effectiveness of the chosen evolutionary processes and develop on them, iteratively evolving more components with vaster connotations on the eventual designs on the creature.

Background

Mentioned above, the inspiration to this project was taken from Karl Sims', 'Evolving Virtual Creatures' project and accompanying research paper (Sims, 1994) which also discusses the use of genetic algorithms in separate muscular and neural systems, morphology and optimization.

In Sims' EVC project, the arguments made were that developed complex virtual creatures that imitate plausible motion in a virtual space is highly difficult to achieve and maintain control over when the approach is via a static, predefined route. In fact, it was stated that the higher the complexity of the creature, the lower the control. This meant that although it is possible to predefine creatures via a static process, it would be incredibly difficult and produce unrealistic results.

In order to achieve complex yet plausible creatures, Sims, opted not to design creatures, but instead to design behaviours and have creatures design themselves around how much they fit the functional requirements. The 4 behaviours implemented were swimming, walking, jumping and following. With these behaviours, Sims, acclimates creatures to a specific function and consequentially narrows the range of randomness in creature design. However, this introduces a problem of under/overfitting evolution cycles because the definitions of a creature is continuously undergoing redefinition and a static evolution system will be unable to competently accommodate the new creature. This means that in order to further generate specialised evolutions, more than just the genetic parameters need continuous change. The algorithm for evolution also needs constant evolution to keep up with complex environment it is in.

When it comes to discussing creature structure, implementation and evolution, the discussion of genotype and phenotypes repeatedly appears with the definition; genotype are genetic encoding of a specific feature of the creature, while phenotypes are the genetic results produced by that genetic encoding of the genotype. To explain, when deciding on what colour of iris a person should have, we can break these genetic encoding that goes into that iris as blue(B) and black (b) (Campbell, 2020). The genotype of a 2 string long array that combines these genetic codes as either BB, Bb/bB or bb. The by-product of these results is that there are 3 particular variants of genotypes with one containing only blue codes, one containing only black codes and the final containing a combination of the 2. The phenotype aspect of this is that any possible by-product of the genotype will always be either blue or black. This discussion of geno/phenotypes led to Sims, hypothesising separate neural and muscular systems with the former being responsible for mathematical processing and gene parameters while the latter, forming the body and the behavioural output.

This is where the genetic structure of Sims' EVC and the proposed project gather inspiration and differ. Sims, operated his evolutionary process as pre-generating a set of genotypes and a set of phenotypes with the idea that limb production, mechanics and instructions would develop as a result

of copying existing phenotypes and applying adjustments and mutations. The process of 'copying' is less a random procedure and more a by-product of evolving a creature by a process called 'mating'. Mating is a generalization for 1 of 2 evolution processes. Crossover and Grafting. Where the genetic build-up of 2 parents are recombined by either swapping selected genes or creating a new set from splices of the parents. Once again in the process of genetic recombination, the recombination in question is that of the phenotypes with mutated genomes.

The mutations in Sims' model were also weighted to control the variation of evolution in each generation and ensure the acclimations towards a behaviour type would be uncompromised. This meant changes across genomes followed a gaussian pattern with variations in gene expression usually being minor and rarely drastic. The weights also ensured a lack of compromise in data accuracies. This means that values in the genome that were large, generally were not adjusted with a fine comb whereas small values were extremely accurate in the size of their changes. It was described as the size of the change was equivalent to the size of the value. I.e. if the torque of a limb in Sims' virtual space was 500 horsepower, it would not dramatically change the system if this was then changed to 450 or 550 horsepower. Whereas, if the degree of freedom of a limbs rotation was 12.6 degrees, a mass fluctuation in the same fashion as the torque would be too dramatic of a change and it would be likely the genetic algorithm would not be able to accommodate such a massive leap.

Much like the environment used in the proposed project, Sims, had to also follow the rules of physics that exist in the real world. It was paramount in his study that the creatures of his software moved plausibly and obeyed rules that intrinsically make sense to human minds. Sims, made this work by using a combination of body dynamics (rules that bodies should abide by), collision detection, collision response, fraction and fluid effects. Much of which exists in the provided system.

Further background includes that of module content from the CS3910 – Computation Intelligence (CI), most relevant are the teachings of Local Search, Evolutionary and Genetic programming. Namely in under the use of solving the Travelling Salesman Problem (TSP).

The teachings of these modules bring further understanding to Sims' approach to evolution, the design choice of crossover and graft and built the foundations for the evolution system that exists in the provided system.

Expectations

What was expected of this project was the ability to produce a system that was capable of approaching pre-set tasks and throwing any conceptual design at the task. The project should be able to highlight the effectiveness of a rudimentary evolutionary algorithm within the context of its design and any viable improvements. The project should also be able to prove that evolution is documentable whether by evolving a biological creature or by evolving a mechanical idea such as a car.

The evolutionary algorithm is expected to improve the virtual creature irrespective of how efficient the algorithm is, with the final output documented in the accommodating log showing the most effective genetic combination of that cycle. The process of evolutionary improvement should begin initially rapidly with successful regenerations slowing down as time progresses, as is the nature of improving the functionality of a design. A best must exist somewhere.

The expected design should allow a user to place any concept, irrespective of viable functionality into the system and it will quickly highlight whether the concept works first of all, followed by an eventual

creature that is highly optimized to the task at hand. All potential generations, as Sims demanded, functioning plausibly. This omits the viability of the course or problem the creature is introduced to, and how effective a creature may be in a virtual environment that is not plausible either.

It is also understandable how these 'expectations' may seem undecided or ambiguous but that is because it is difficult to categorically define what the expected or ideal outcome of this project is without understanding that evolution is still a concept under development and that both the scope, complexity and output will continue to grow as refinements are made. Sims himself stated *"A control system that someday actually generates "intelligent" behaviour might tend to be a complex mess beyond our understanding. Artificial evolution permits the generation of complicated virtual systems without requiring design, and the use of unbounded genetic languages allows evolving systems to increase in complexity beyond our understanding."* [(Sims, 1994)-9 Conclusion].

The only plausible measurement, would be to say that what is expected of this project is to uncover at least 1 new thing about evolution that was not already learnt through Karl Sims and the teachings of CS3910.

Evolutionary Algorithm

When approaching the conceptual design of the proposed software, there was a firm idea of which algorithm would be used as the evolver in this software and the reasons for why it, in hindsight, was the most optimal for this application. The prospective design would have set out to make use of the standard evolutionary algorithm as taught in CS3910 and supplementary textbooks (AE Eiben, 2003). Genetic crossover.

Genetic evolution has always conceptually maintained the idea that there should always exist a generational process of taking 2 or more uniquely identical objects as the seed (in this case, creatures) and use the existing genomes in order to manufacture a new subset genome containing a set of data from each seed. From this, a set of new creatures can be manufactured taking potentially the best traits of each seed and tackling the problem with an improved or worsened fitness. This process is harnessed in order to serve the ideals of survival of the fittest. Fitness being a concept of effectiveness/competency towards the problem it is tackling.

Described above is the fundamental process of the crossover evolutionary process as inspired by Sims, and this process is what was originally proposed in the development of this project. Though it was never specified in the project design that improved fitness was the objective of the software - this is an inherent concept in evolution. The only variation to any crossover is the number of crossovers and how to approach a fitness loop (where an improved fitness cannot be found due to being stuck in a feedback loop).

Although crossover is the ideal candidate for evolutionary processing, this is not what currently exists in the provided software. Instead what is currently being used is a likely hybrid variation of a perbutative greedy tour with aspects of evolution. Perbutative greedy tours, work as a way of randomly changing values or combinations of a genome from a function combination; evaluating the fitness and repeating the process on that genome if it is of higher fitness, or on the previous genome, if worse. The concept of splicing genetic code from multiple seeds is not involved here. The current variation of the greedy tour algorithm is the existence of a sleeping seeder, or in other words, a seed that is not in use unless of a specific criteria.

A problem that existed in the TSP was that at some point in crossover, the evolver would reach a hump and continually cycle among combinations that were all locally optimal but not the absolute optimal combination. Typically this is due to the limitations of the native evolutionary algorithm and fail safes need to be put in place to ensure that something takes the evolver out of a constant loop. Typically this fail safe is a sleeping seeder i.e. an existing combination or a vastly different regeneration loop that will run for a single generation to refresh the genomes.

A similar feature to this is implemented wherein, if a current combination cannot complete the course at a higher fitness, a random genome will be selected from the past runs and run that through the evolver to generate a different combination. This is done as it can be hypothesised that 2 higher optimal car configurations are of vastly different builds. For instance, a car with small wheels, high torque and extremely low frequency dampening can rapidly fly to the target while alternatively a vehicle with giant wheels and high torque and frequency dampening may also complete the course, at a similar fitness, but in a far more plausible fashion. Using a seeder allows both variations to become generated and compete for optimality.

To simplify the fitness evaluation process. A creature's objective is to move from its beginning position, to the target position which will be a box waiting for a collision. There are 2 instances where a regeneration occurs; when this target has been collided with; when double the previous best time has been met. In both of these cases, an evolution of the new combination occurs when the creature meets the finish line in a faster time than the current best. Alternatively, an evolution of the previous best occurs in all instances where double the time has surpassed, or when the target has been met at a slower time than the current best. In every instance where a combination meets the target but does so in a slower time, that combination becomes the new potential seeder. After every failure for a new combination to evolve, a counter is kept to determine the number of cycles that have run to determine whether a local optima has been generated, once 10 failures are reached, the seeder is activated and the cycle begins again.

Some aspects of the greedy tour and crossover are naturally similar, especially in the case of being greedy. One of the major flaws of greedy tour is that, unlike crossover, greedy tour is not optimally directional like crossover. In nature, crossover responsibly selects the best parents (seeds), to create the best children that will compete to determine who the best of the offspring are and whether these offspring contain any improvement over the fitness of their parent seeds. Due to complexity constraints, regeneration of new creatures has been left to this variant of the greedy tour with future developments upgrading to a crossover evolution cycles. Or much like Sims, a combination of crossover and grafting.

Process

Term 1:

My Final Year Project (FYP) supervisor (soup), and personal tutor, Dr Peter Lewis, and I mapped out a range of ideas before deciding on the current design prospective, as my initial ideas were seemingly elaborate and ostentatious; desiring for my EVC to carry out complex tasks before learning to crawl. At this stage, I had wanted to complete a rudimentary 2D design and delve immediately into 3D, unappreciative that even designing a virtually evolving object in 2D would be a complex process, and that transitioning to a new physics engine would prove a host of new issues that I later learnt I was not equipped to deal with.

We then began discussing how to implement this project and what tools may be useful to utilize. As the initial project definition involved creating a 3D EVC and my preference for programming language was Java; Peter, recommended looking into jBullet (jezek2, 2010), for an easily integrable 3D Java physics engine. Peter, also suggest that it would be simpler to implement the logic required for an EVC in a 2D realm before transitioning over to 3D. The rational perspective for this was that, provided the language and variables largely remained the same, the only difference that the 3D engine introduces is that I would need to plug a few extra tests into the program to accommodate the extra variables that would be created as a result of working with a 3rd plane. During this conversation, Peter recommended looking into Box2D (Catto, 2020) as this is a previously used and liked 2D physics engine, as the way to implement early stages of this software.

As mentioned above, one of the choices of modules chosen for the final year was CI. Unknown to me that this would be the best choice of modules I would choose as the intelligent logic and algorithms of multiple approaches would be taught to me. Once I was able to understand how to completely produce these algorithms, I would be able to simply take an algorithm and make adjustments for this project. Much of the work done this term was to focus on logic, plan and document my progress and to eventually be at a stage where I was able to begin the implementation of the first task.

Soon after my initial meeting with my soup, I began looking into forums to see whether Box2D is well supported and that any issues flagged were easily resolvable. In the process of this, I learnt that Box2D was a C# engine and began to look for alternatives. My assessment of Box2D is; while Peter recommended it with a great deal of confidence, I could either learn to code in C# and then look for an alternative 3D physics engine, or I could look for a java port. Multiple developers since the release and popularisation of Box2D ported multiple versions of multiple languages on Box2D and I pulled a version from GitHub called JBox2D (Murphy, 2013), developed in 2013 with ongoing support. However, due to my overconfidence in Peter's recommendation, I had overlooked evaluating existing documentation and support for this port and I would later regret this.

For around 2 weeks, dependency issues persistently appeared when importing JBox2D into the Eclipse IDE (the chosen IDE for this project). The package is of a Maven base, which involves creating dependencies with the Maven compiler. Maven is a tool that is used to make creating dependencies in java code easier for you in the sense that, creating the class or object dependencies is not required since Maven will do this for you in place of importing new libraries and integrating them. In order for Maven to do this, the program will have to install a helper tool in order to manage these dependencies automatically. The issue I faced was simply that the connector I was using to fetch the helper tool was either corrupt, or as many suggested on the related GitHub repository, the link to the Maven Catalogue was outdated and no longer supported. After multiple attempts to reinstall the entire project and update the catalogue link to no avail, I instead opted to uninstall eclipse and try again from scratch. I am unable to explain why, but reinstalling Eclipse alone was the only troubleshooting that was required in the end. JBox2D was now fully integrated into my Java IDE.

JBox2D was an incredibly difficult physics engine for one to use, especially one with so little confidence to approach this as I. Between the lack of documentation, comments or tutorials, the largest chunk of this year was dedicated not only to learning about the existing features, components, joints, mechanics and so on. Much of the time was dedicated to working out how the vast number of classes integrated with each other, with an aim to understand what the need for these connections were, what was manipulable and essential to my use and what could go. The trouble was that, with a lack of documentation, it was impossible to understand how these components interacted with each other and how to solve unidentifiable compiler errors due to a lack of time. However, one of the

largest perks of JBox2D was the testbed. For all the flaws that existed, the absolute need to learn everything was made unnecessary as featured in the engine was a pre-existing test space, designed to demonstrate what features exist to make up for the lack of supporting documents.

I began testing the pre-existing testbed functions in order to see what functionality exist already in the engine and find places I could perhaps manipulate code for my own use. Unfortunately, due to other academic commitments, I placed further testing and learning of JBox2D on hold. This was near the reading week mark with the first of the CI deadlines fast approaching. I focused for the next 2 weeks on completing the sign off tasks for the coursework; understanding and implementing random search, 2-opt swap and swarm intelligence. Unfortunately I was unable to complete the evolutionary algorithm in time for the sign off but this was not terrible as I had wished not to rush this algorithm and know that the implementation was absolutely fleshed out and correct as this would enable me to produce a far better designed and well understood evolutionary algorithm.

Near the end of the first term, I produced a Gantt chart for the rest of my project scope. The reason for developing the Gantt chart now and not in the past was because, I had an understanding of what I had to do in order to realise my FYP but not the confidence in my expertise in order to achieve them within a reasonable allotted time. Programming the local search algorithm may be a 1 day job for a confident and skilled programmer yet for me it took 3 days. How was I able to assess a reasonable amount of time to allot to a task? Much like what was written above, my soup helped to visualize a time frame each task may take, given what the finer details of those tasks may involve and to assess from those whether the task itself is complicated or not. Another useful thing Peter mentioned was to work backwards in milestone of what needs to be done. With these things in mind, An initial Gantt chart was completed and so began the rest of my tasks (Figure 4).

Term 2:

Pressing on through into the 2nd term with the encouragement of my supervisor and peers, in the process of finding my way back to the project, I began looking into other examples of similar projects that, much like I was doing with JBox2D, could dissect ideas from. The sources where any references or inspiration gathered were from a very similar project to the scope called ‘BoxCar2D’ (Ryan, n.d.).

While further dissecting jbox2d to evaluate what features were available to me, I modelled my understanding of the engine dominantly through the test named ‘Car.java’. Further code written in ‘CarDemo.java’ that is used to demonstrate the current state of the project progress is modelled originally from ‘Car.java’ with adaptions made to suit the needs of this project.

Through ‘Car.java’, the essential forces that dictates the creatures interaction within the virtual space were made available to understand (Shiffman, n.d.). The essential forces needed to produce any valuable insight to this project were mass (total mass non-reliant on gravity), force (the change in energy i.e. speed and direction | Force = mass*acceleration (powerelectric, n.d.)), torque (the available rotational energy of an object) and impulses (the forces produced on impact). These 4 fundamental forces are the ingredients that dictate the dish. These are represented in the code as m_speed, m_torque, m_hz/dampening frequency, m_zeta, fd.density and fd.friction. These resemble much of the same environmental dynamics displayed in Sims’ version of EVC and allow any moving object to bind itself to the laws of physics as experienced in the real world.

To obey the laws of physics, one must understand the concepts of physical forces and how creatures navigate through life. When we think about friction, a layman may picture friction burns on their hand or losing grip on the ground and scraping the sole of their shoe across the floor, but perhaps not that

friction is a constant force that never ceases, only changes. That even standing still, we are exerting frictional force on the ground in order not to slide around, even if that friction is not felt. Inertial forces exist everywhere and deliver a fashion of equilibrium. As Einstein once said "For every action, there is an equal and opposite reaction". This can be illustrated in the real world as the display of 2 cars driving at 30mph. One car may drive into a large concrete wall while the other may drive into a large elastic band. The 2 acting reactionary forces in this example are both the wall and the elastic band and while the cars will drive all the same, the inertial force of the reactionary forces will respond differently.

In jbox2d, to replicate these conditions, the actions and reactions of the objects within the virtual space are decided by the forces mentioned above. Density will determine how heavy or tensile an object is. The friction will determine how much downward force the creature will be able to apply and exert in the reactionary response. m_hz will determine the responsiveness from impulses such as the suspension in a car. As the car oscillates downwards due to a downward directional force or bump, the impulse response will ensure that the suspension will bring the car back towards equilibrium and so forth. Understanding these concepts helped me to identify candidates for evolution. Providing evolutionary regenerations that have potential applications to the real world. For example, the friction force that exists from tire rubber in the real world in some car regenerations might be too grippy for an extremely fit creature. Though that is conjecture, hypothetically an extremely grippy vehicle with high torque and speed should have the highest fitness in concept. More on that when discussing findings.

Term 3 and beyond:

As time was growing short due to confusion built surrounding jbox2d and battling burn out from the situational educational environment, it was decided that deferral would be the best option to complete this software, and following that, some compromises.

When approaching the concept of evolving a virtual creature, different ideas came to mind and in early concepts, the ability for new limbs and appendages would be generable, moving away from using fixed structures and into a direction more akin to true evolution, i.e. adapting to its environment to improve fitness. The complexity involved in this route, coupled with the now reduced time made this infeasible.

Another complexity to deal with in understanding jbox2d and how to tackle generating a first stage creature was to understand and manipulate one of the many existing 'joints' in the engine. These joints determine the behaviour, mechanics and limitations of movement. For instance, a revolute joint is one that revolves and has attached appendages that react in suit. For instance, a door handle is a revolute joint as the turn of the handle rotates a rod inside the door, which reacts with a connected lever to a drag a bolt in the direction of rotation. Another joint that exists are pistons. Pistons are innovative joints that will pull or lift a level, using inverse compact force. This means that a piston will oscillate between a compressed and non-compressed position, using the force exerted from each state to return to the other state, imparting motion.

The consensus given the shorter time frame, was to simplify the scope and restrain the creature to only making use of wheel joints. Wheel joints are self-explanatory. They are joints that rotate circularly and the concept of torque, speed and dampening frequency is much easier to understand as these are things we regularly use in the real world to measure the specifications of road vehicles. Using wheel joints was advantageous due to the sheer simplicity that it could be set up quickly and allow us to begin testing the use of evolutionary algorithms to fine tune the level of control that the

evolver should have. The original intention of this was to set up a minimum viable product, a sort of bouncing board to the bouncing board. Using wheel joints and finalizing a creature design in the configuration of a car resulted in producing a canvas that could begin to be built on. This 2 pronged task was intended to give us the foundation of something that could expand into something larger as deeper levels of control were formed, enhancement to evolution technology and so forth. Although denying to use of it at this stage, once time is given to extensively familiarising myself with more of jbox2d's technology, these new features would also be able to have some integration and provide valuable variations and further scope.

At this stage, new limitations and compromises are made and development is underway. Due to the lack of helpful resources for jbox2d and some consultation with my new personal tutor, Dr Megan Robertson. The perfectly viable consensus was, rather than to learn a large physics engine that lacked documentation, that it would be more feasible to use existing classes and technologies and refine them for the use of the project. The objective at the onset was never to produce a fully-fledged, self-programmed software, but to understand the concepts of evolution in a viable environment and the foundation for these things had already existed in jbox2d. Developing the idea was far more valuable than building the foundational environment. This meant no more having to deal with uninvestigable compiler errors and compatibility issues as the technology already existed in an environment and had been proven to work. This resulted in removing several potential levels of error, leaving only my incompetence as a programmer, but this was something I could change.

Armed with the confidence of the engines inner workings and existing code, I began experimenting with the 'Car.java', test class. This was the stage at which my understanding of conceptual forces were solidified as these were manually adapted, limits were sought and the modifications were made where they were potentially viable or had some contextual use in the sense of evolution. Though it was early stage, one of the mysteries of this engine or the fixture or joint definitions was an unexplainable wheel limitation. Through experimentation, a limit of 3 wheels was achieved with any larger numbers incurring compiler errors. There is no understanding as to why this issue occurs, though this may be due to conflicting joint definitions rather than a limitation of jbox2d. This did not appear as much of an issue due to the scope of the project at this stage.

There would be no need at this stage to be able to work out how to regenerate combinations with new appendages if this feature was extremely unique to the generation of a 3rd wheel. Though if this were to be implemented. The application of the genetic algorithms would turn into a 2 stage genome state, or a 2 dimensions genome. In this configuration, the 1st dimensions would be responsible for tracking the number of appendages or "phenotypes" in order to define the 2nd dimension's size. For instance, the current genome is a list that is 12 objects long and this is responsible for genetic regeneration of 2 wheels. Of these 12 objects; 4 objects are responsible for quantifying fitness; 4 objects for the genetic build of the first wheel; and 4 object for the genetic build of the second wheel. Adding a new dimension would be able to harness the number of wheels and generate a new list or dimension based on the genome of each object. Though without implementing this, it is difficult to state whether this is a highly effective or highly convoluted process.

It is possible in theory to adopt Sims', construction in this fashion as Sims' concept, involved generating predefined phenotypes with adjustable genotypes. In the above context, a wheel can act as a predefined phenotype and expanding this would only be a simple process of defining more phenotypes that can be called upon generation of the 1st dimension.

The ‘Car.java’ test, initially is an interactable test where the user is given total control of the vehicle, with freedoms to accelerate, brake and many other features that will be omitted in this report due to the lack of relevance to the project. Fortunately, it was simple enough to remove keyboard scanner features and automate the acceleration of the vehicle.

The next roadblock faced was in attempting to regenerate the animation and creature autonomously. Providing a state in the machine at which the fitness evaluation can take place and an evolution to occur. The code structure of these tests are interesting as they all exists as children to the testbed space. Consider the Kodak Carousel, in the era of film photography, Kodak produced a projector that accepted photo slides and allowed the user to project these onto a surface and rotate the carousel or swap slides inside the device to show different photos. The testbed in this context is analogous to the Carousel machine, while the photos are analogous to the variety of tests that can be carried out in this space. In this respect, the testbed class acted as a parent, calling the ‘Car.java’ test through a method called ‘step’. This is confusing as every test class has a separate main method that does not run the program but initiates the step function which acts as sort of a runtime manager. Ascertaining at which stage a regeneration should occur was tricky, should this be done inside the main method or the step function. Both options came with their own hurdles, the largest of either being that recompilations in the main method would engage the object generator on repeat or refuse to allow genome configurations to update. The largest issue of the step function was that any variable saved, amended or deleted simply just did not exist or was accessible to this. For some reason, variables instantiated in the main method or globally were readable but in the separate “launch” class, these elements seems to be generated but not stored accessibly.

The results of this roadblock indicated that data needed to be placed somewhere accessible to all methods while functions enacted upon them needed to be placed somewhere that did not initiate recursive loops and improper data storage. This resulted in a class designed to define variables globally accessible to all methods, placing initiators in the main methods and placing modifiers inside the runtime manager so that initiations and modifications do not conflict with each other at the same stage. This lead to the eventual class design (further explanation in Design).

Once regeneration was complete, deciding on an integrated evolution method or an openly accessible class became the next big question. During the CS3910 module, all computations occurred within the same class as it was feasible for a small scale problem, but if this project intends to develop, have applications in multiple contexts, it is logical to provide the evolutionary algorithms with their own space. The existing evolver class is original.

Having a creature, a task, a regeneration process and an evolver built the fundamentals of evolution. The next step was to find a way to evaluate the effectiveness of the current algorithm to the current problem or to preview whether any fitness improvement was occurring. For this, another class was taken from my CS3910 module that I had written and this was simply a method that recorded the how many runs have occurred, the fitness to compete with, the actual fitness, how many evolutions have occurred, followed by the genome construction of the phenotypes. Recording these allowed us to evaluate which combinations of genetic information are proven to have higher fitness.

Project Management and Learning Styles

To come to the product as infantile as it is, was a very windy road. With approaching the project

design, project management, dealing with a lack of confidence related to my abilities to take on a programming intensive task and approaching the task in a vastly unfamiliar pandemic climate.

After establishing a connection with Peter, and frequently keeping him up to date on my progress and findings. The first thing that was discussed was, What did I want to take home this year from my final year project? I had an initial concern of dread, feeling that the project I chose; while I thought it was incredibly interesting, was far out of my abilities. After much discussion, I was left with a sense of confidence that it was better to do something incredibly interesting, sufficiently rather than something boring, very well.

Throughout the initial term and the earlier half of the second, the persistent issue of programmer confidence was prevalent. This was enforced through bottom of the barrel, sufficient, 'good enough' grades in java development and vast amount of confusion when approaching coding prior to this project. At one stage, the instruction "Initialize the variables" would leave me in terror as I also did not fully understand whether initialization was the variable you instantiate globally inside the class and not just something that needs to be done inside the main method of that class. Through many conversations with Peter regarding this, the established mindset to approach to dealing with questioning your competency in a task is to plan. I was assured through my soup that part of the whole process of this project is learning to come up against the many roadblocks that were erected in this project and navigate around them and one of the most effective ways to do that was through planning. In respect to programming, this meant taking a pen to paper and visualizing what it was that I wanted.

This method of visualization was something I learnt to be a fundamental aspect of this entire project in materialising the concepts and software discussed in this report. A lack of planning leads to chaos and in chaos, doubt is sewn (Guzdial, 2020).

Progress in the second term came to a startling halt as the next roadblock appeared (this is illustrated in the Diary Report between the entries dated "13/12/2020 – 06/02/2021" to "14/02/2021 – 03/07/2021") (Qasim, 2021). At this point, the burgeoning exhaustion from studying through what felt like infinite lockdowns began to accumulate in burn out. Though I learnt many new learning techniques and experimented with work styles to find one that suited me, the learning process felt akin to a job that you lacked an affection for. One of the most inspirational things Peter had told me was "Learning should feel like messing around in a shed full of tools and looking outside to see the day is already over".

Often brought up in discussion was learning styles. In conversations, what became established was the idea of 'learning to learn'. University is an environment intended to ensure that a student searches for the ability to learn with the method that works best for them, and this FYP was the perfect environment for any learning style to bloom or change.

From ignoring Dr Nick Powell's advice on writing physical notes instead of typed to tackling this entire project, drawing and writing on paper, the foundational concepts required in this development and planning what direction the to take the code and how. Learning in the process that the act of making physical tactile notes are a far more visceral experience of imprinting information. This also helped to dispel the assumption that the only good programmer is the one that could write a program without needing to stop and plan as illustrated in Guzdial's declaration of what success in computing looks like (Guzdial, 2020).

From begrudgingly tackling the problem in a specific fashion, remaining obsessed with the idea that the written code is closer to the solution than starting again and rethinking the strategy. During the CS3910 module, when learning how to build the local random search algorithm, for weeks I had faced the same complicated error that conditional statement were either overbearing or too loose with many placed, designed to strangle the data processing. In discussions, Peter never stated whether my approach to this was correct or otherwise. Instead it was advised to revise my approach and see if it really was the best approach to the problem. This led me to redevelop a completely functional program with far less complexity and this was due entirely to what possible Sims eluded to in that, the result is not always the quantifiable measurement of success, but controlling the behaviour may produce better results. The lessons became fundamental to the realisation of this project and in dealing with the shock of lack of preparedness of managing a project of this scale.

Prior to this project, my fundamental experience in project management was entirely small scale and within small teams. Externally, my job is as a freelance photographer and often the projects undertaken are far smaller in scale, often lesser in complexity and usually come with sometimes an unclear set of instructions but clear measurements of success over a short time scale. However, the FYP is a long term project that is entirely defined by myself with a scale unknown to me. At least as a freelance, there was experience in the process involved in undertaking a project, where alternatively, learning the process was a fundamental aspect of the FYP. As such, the techniques that proved effective as a photographer were not translatable into this project and this took a great deal of difficulty to accommodate.

The approach to this problem was an attempt to break the large project into subsets of small projects and deal with them in a similar fashion to a photography project, however, this probably is not the most effective management process but a sufficable compromise.

Design

Expanding on the details provided in the above sections with supplementary but not exhaustive diagrams. As stated above, most of the code and structure is pre-existing and modifications were made. There are many cases of the use of stack overflow (various, n.d.) and an IDE AI, tabnine when in search of understanding compiler errors or jbox2d specific data conversion issues (Dror Weiss, n.d.). All original classes are place in their own folder and author illustrated. The basis of ‘CarDemo.java’ is built based on the structure of ‘Car.java’ and the general adaptation is the separation of the car building functionality from the main method and placed into the launch method, as illustrated in Figure 1. The original code responsible for input into the testbed to control the vehicle would also exist here but is now replaced by automation.

The most dramatic and noticeable difference if looking between ‘Car.java’ and ‘CarDemo.java’ (other than the removal of tags) is that, other than the process dubbed “Continue running test” in Figure 1, all of the following processes are new and this is largely where the evolution will occur. This is where the parameters of activation of the fitness evaluations is housed.

As explained above, these criteria are either the contact of the target box or double the best time is surpassed. Each of these completions have further levels and connotations attached, also explained above.

Moving onto the evolver, the process of evolution (also not exhaustive) is illustrated in Figure 2, where the genome is fed into from the step method of the ‘CarDemo.java’ class. The process of

evolution in this structure is very particularly structured which in the original concept of evolution, is far too unique and situational, however a great building block for further genetic evolution. The algorithm words by randomly looking for a value inside the genome and replacing it with a randomly generated value that is within range. The current ranges for these are as follows:

- Wheel Radius: 0.01f – 1.0f
- Wheel Speed: -200.0f – 0.0f
- Wheel Torque: 0.0f – 100.0f
- Frequency Dampening: 1.0f – 10.0f

These ranges were decided upon after testing to determine reasonable limits to ensure that extremely unplausible and lacking of value regenerations were reduced in likelihood and ensured that the creature could continue to function and produce results. One of the most dramatic problems with requiring limits is having wheels that have a large radius than 1.6f as there were 2 responses from this build. Either the frequency dampening was too low and the joint magnetism was not enough to prop the car body above the wheels to move. Or the frequency dampening was too high and would demonstrate explosive collision forces against the ground. Controlling factors like these are essential to generating valuable data.

In testing these limits, it was found that an old gene was unnecessary and removed. This 5th gene determined whether the wheel motors were activated or not but under consideration, though activated, a wheel with a speed or torque of 0.0f were effectively deactivated and led to the realisation that too many components regarding wheel activation was showing some bias towards changing this phenotypical response.

As for the CSVWriter, this class is a standard file writer class with the addition of a data conversion mechanism to change Object objects to String objects in order for the data to be saved.

Sample Demo: https://1drv.ms/u/s!Agy77BMmbQXLgt1qi77hjjA_Ztx53w?e=rLccEq
Sample Results: https://1drv.ms/u/s!Agy77BMmbQXLgt0cCmLD8P5a1eH7ew?e=auevkq

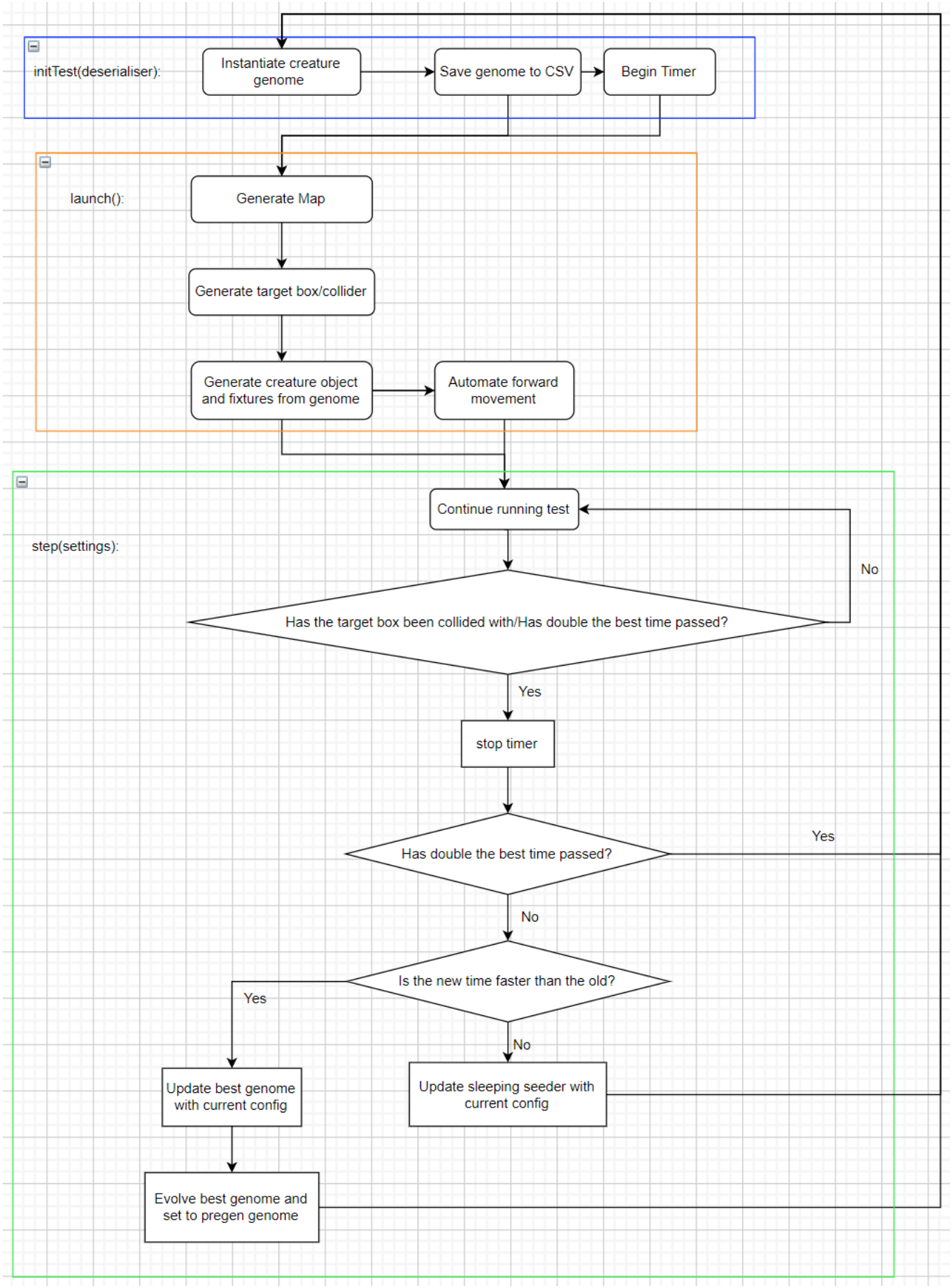

Figure 1: CarDemo.java structure

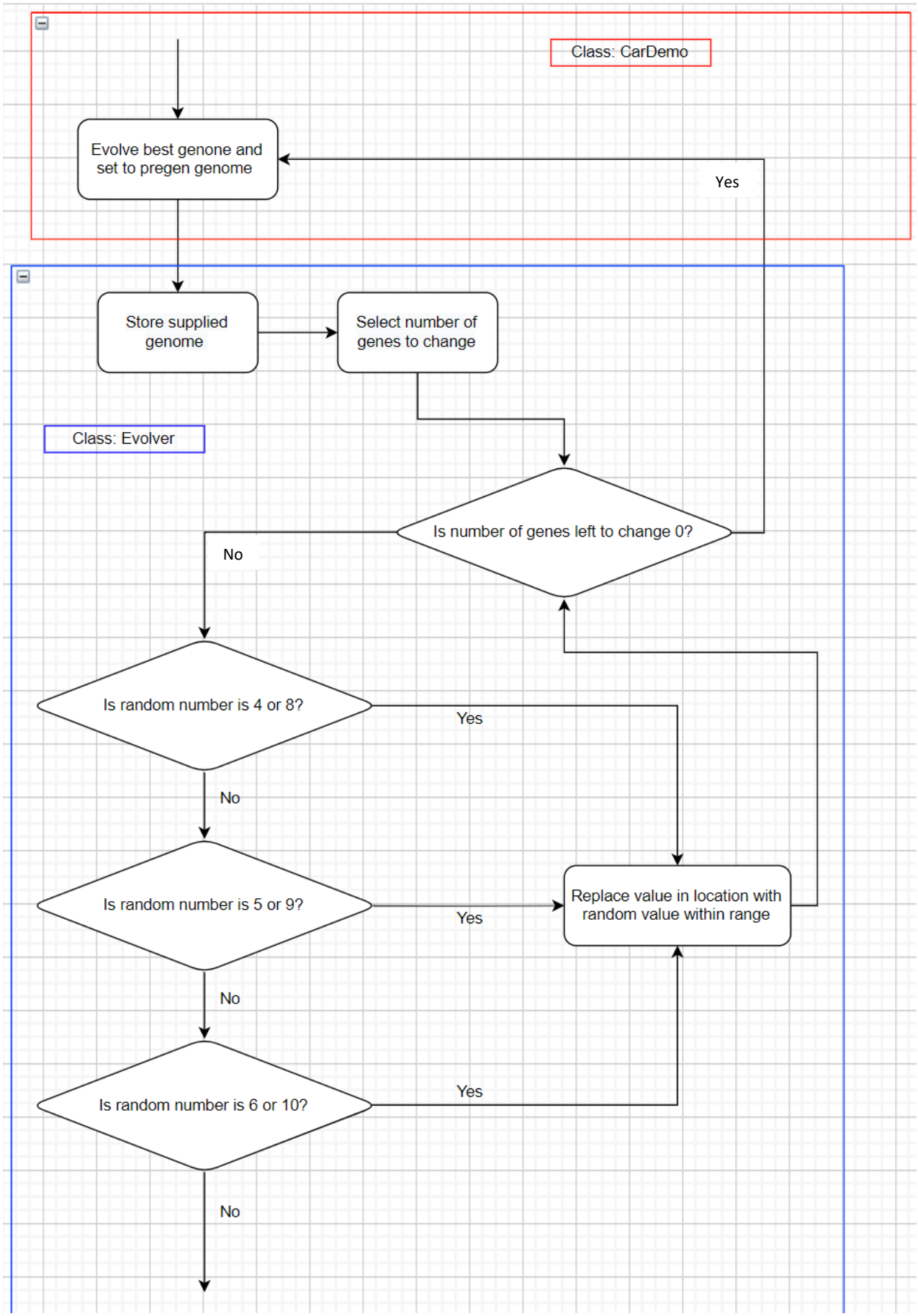

Figure 2: Evolver.java structure

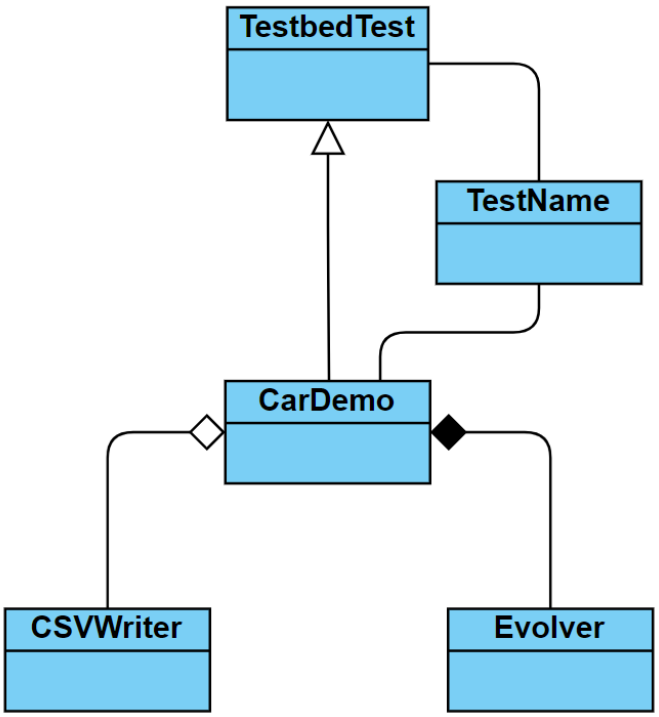

Figure 3: Class structure

Results:

The findings we can deduce from the current software is a lot of potential. In first the possibilities that exist from correcting the flaws that exist in the evolutionary algorithms, seeders, a wider range of autonomy on genome mutation or implementation of phenotype recombination.

In the current software build, the assumption was made that in theory, a vehicle with high torque and low to medium speed will produce a better fitness for the creature. However, these expectations omitted the input of efficiency factors and any possibility that too much of something is a bad thing. For context, torque is the driving force in pushing a mechanism into motion. The higher the torque, the faster the rate at which peak motion and frequency can be reached, however this explanation omits the mention of friction forces that act as traction for the vehicle. It is possible that too much torque is self-defeating as the inverse impulse factor may be too weak to return an equal reactionary force in order to maintain traction. Think of tire spin. In fact, often in regenerations that displayed high torque with high to mid speed, the impulse force was so great that the creature would be stuck performing infinite back flips, oddly enough, the highest fitness in the demo was performed valiantly by a creature stuck in an aggressive backflip loop. This is good in terms of fitness but bad in terms of plausibility.

While developing this software and studying the fundamentals of the physical factors, the scope at which these forces affect motion and interaction in the virtual dimension were far more complex than what was envisioned. When describing above the omission of efficiency factors, these were due to a lack of expectation and not a choice decision.

The key take away from this project is that, although the project scope had shrunk in complexity, the provided software proves that it is possible to evolve a virtual creature/object, albeit extremely primitively and far from any viable use in context to the possible later use of the software. It has

illustrated that just regenerating genomes with different values is not enough to produce a reasonable output, or one with real world viability.

In this case, it is clear now why Sims opted to genetically regenerate creatures based on phenotypes rather that genotypes (Sims, 1994). If Sims opted alone to regenerate genetic combinations in the fashion that has been done in the provided software, then perhaps he never would have been able to create any walking creatures that can display any sense of direction or contact the green cube displayed in the latter half of the referenced video or develop creatures to a level of complexity where they can modestly compete for fitness.

Of course the provided method of evolution is far too simplistic and progress from this would have been to move forward into the crossover method but due to time constraints, this was not possible. However this demonstration has made it clear that evolution cannot be so easily replicated in this fashion.

Future Developments

Reflecting on this project, with what has been illustrated above, this project is far too deep into its infancy. The solution to evolution is far too simplistic and technically does not even count as an intelligent algorithm. The seeder function works but not optimally in a way that demands the absolute best fitness. The variability in the creature is far too minor to really demonstrate the potential behind evolution.

Another detrimental problem from this program development is the evolutionary algorithm not being directional enough. Or containing any sense of direction at all. The fitness of the creature could jump to a genotype that has incredibly high efficiency with the next being vastly inefficient with very little influence on whether fitness improvements were due to this aspect of "survival of the fittest" or improvement much like in the real world concept of evolution. Instead the fitness improvements at this stage are entirely down to luck. A fish without a tail fin should not evolve to have one based on luck, it should evolve based on its behaviour, environment and survivability.

With that said, immediate developments would involve firstly adding a mechanism for stopping the creature. The measurement of fitness currently is based on being able to accelerate fast enough that the target can be met but not so much that it is missed, but this is ineffectual because the true results have no applicable value other than perhaps a racecourse. That is why a mechanism for stopping would be implemented next as arguably the beginning of behaviour development. After this would be the implementation of a crossover evolution system that can be further built on, as flaws arise. Eventually once these limitations are addressed, implementations can begin to involve the use of generating more wheels or appendages as the crossover algorithm evolves and more joint definitions are learnt and understood.

Improvements to the map will be difficult but possible. The idea of the map is to allow the developer to produce a 'problem' and then send the creature at it, in order to find the best response, fitness or build from it. The current map for all intents and purposes is just designed to make the course extremely difficult yet prove that the concept checks out.

Late stage developments would likely display far more complexity and follow a structure akin to Sims' approach with predefining phenotypes and parametrising behaviour boundaries (though this is likely to be done through choice of joints and controls to degrees of freedom). The model of a vehicle

would ideally be replaced with that of a bipedal model (such as a human) or in order to pay homage to the concept of trialling designs for use in industry, an armature model much like those found in industrial lines that produces vehicles.

Conclusion

Overall, the EVC project is an extremely deep in depth project in terms of understandability but incredibly interesting. Despite being in its early stages, understanding the multi-faceted complexity of the applicability of evolution within a virtual space for a pseudo living creature is rich with aspects that can only be revealed through exploration of the process in action. Understanding the approach to the process that Sims chose and trying to replicate or work towards that functionality with knowledge I was equipped with through my studies was extremely eye opening as discovering the greatness of the innovation Sims worked with and the multiple ‘aha’ moments brought to me through tutoring from Dr Peter Lewis produced a greater sense of appreciation for the concepts at play. This project moving forward has been a big lesson in technical and abstract skill; in programming ability; problem solving; developing learning styles; and concept building. It will be incredibly exciting to see how evolutionary concepts evolve from now on.

References

AE Eiben, J. E. S., 2003. Natural Computer Series. In: *Introduction to Evolutionary Computer.* s.l.:Springer-Verlag Berlin Heideberg, p. 300.

Campbell, M., 2020. *Technology Networks.* [Online]
Available at: https://www.technologynetworks.com/genomics/articles/genotype-vs-phenotype-examples-and-definitions-318446
[Accessed 11 2020].

Catto, E., 2020. *Box2D.* [Online]
Available at: https://box2d.org/
[Accessed 10 2020].

Dror Weiss, B. J. N. M. e. a., n.d. *tabnine.* [Online]
Available at: https://www.tabnine.com/?utm_source=search-web
[Accessed 06 2021].

Eiben, A., n.d. *evosphere.* [Online]
Available at: https://evosphere.eu/
[Accessed 10 08 2021].

Guzdial, M., 2020. *Communications of the ACM.* [Online]
Available at: https://cacm.acm.org/blogs/blog-cacm/247372-students-need-to-know-what-success-in-computing-looks-like-starting-from-realistic-expectations/fulltext
[Accessed 11 2020].

jezek2, 2010. *jBullet - Java port of Bullet Physics Library.* [Online]
Available at: http://jbullet.advel.cz/
[Accessed 10 08 2021].

Murphy, D., 2013. *GitHub - jbox-2.2.1.1.* [Online]
Available at: https://github.com/jbox2d/jbox2d
[Accessed 10 2020].

powerelectric, n.d. *Power Electric.* [Online]
Available at: https://www.powerelectric.com/motor-resources/motors101/speed-vs-torque
[Accessed 02 2021].

Qasim, I., 2021. *GitHub.* [Online]
Available at: https://github.com/Iqasim94/180021745-FYP-EvolvingVirtualCreatures/blob/main/Diary%20Report.docx
[Accessed 08 2021].

Ryan, n.d. *boxcar2d.* [Online]
Available at: http://boxcar2d.com/index.html
[Accessed 07 2021].

Shiffman, D., n.d. Chapter 5. Physics Libraries. In: S. Fry, ed. *The Nature of Code.* s.l.:s.n.

Sims, K., 1994. *Evolved Virtual Creatures.* [Online]
Available at: https://www.karlsims.com/evolved-virtual-creatures.html
[Accessed 10 08 2021].

Sims, K., 1994. *Evolving Virtual Creatures.* [Online]
Available at: https://www.karlsims.com/papers/siggraph94.pdf
[Accessed 10 08 2021].

Sims, K., 1994. *YouTube.* [Online]
Available at: https://www.youtube.com/watch?v=RZtZia4ZkX8
[Accessed 11 2020].

various, n.d. *stackoverflow.* [Online]
Available at: https://stackoverflow.com/
[Accessed 06 2021].

Figure 4: First stage Gantt plan