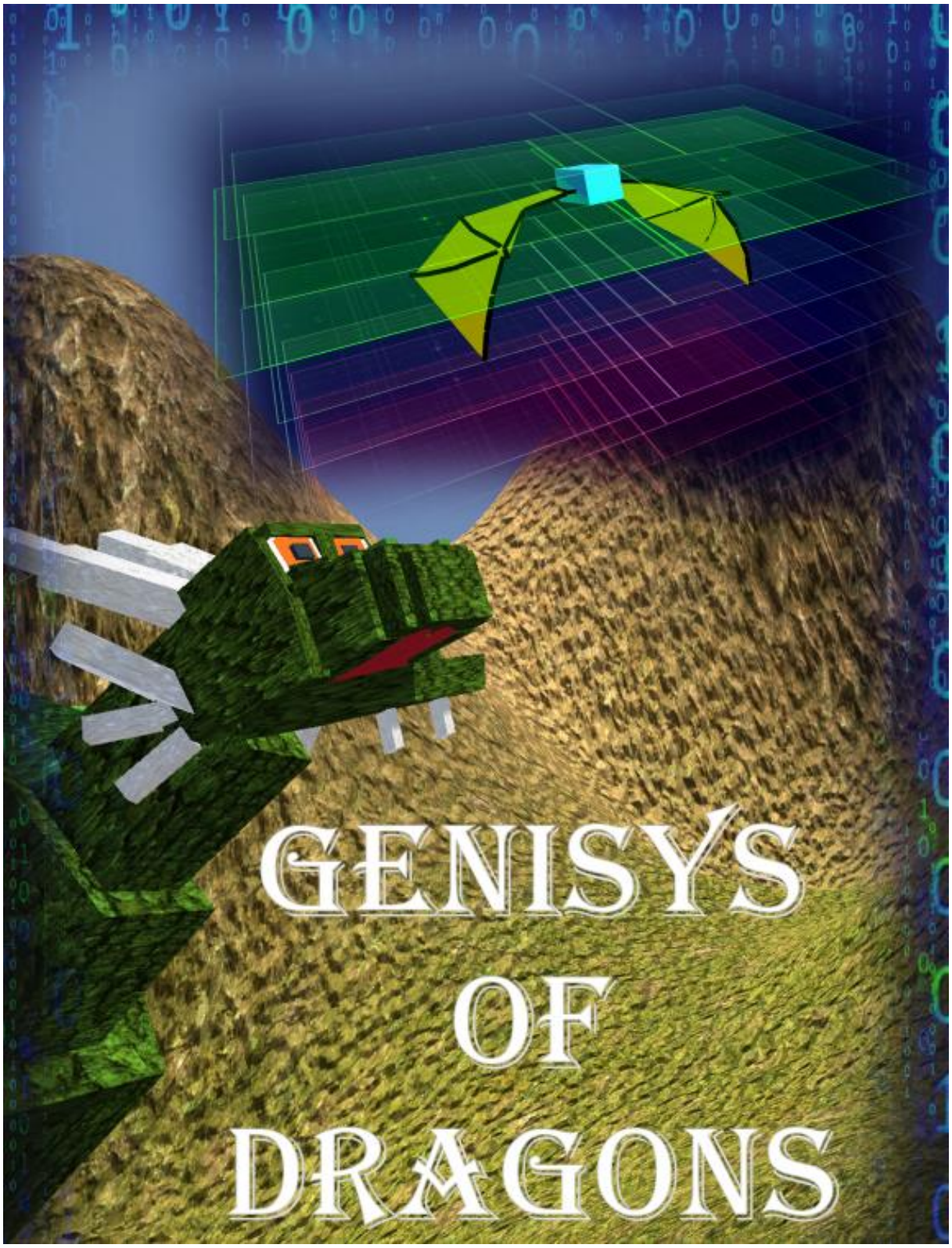

Ein Software-Projekt von Alexander Georgescu
Fachbereich: Künstliche Intelligenz
Betreuer: Prof. Dr. Johannes Maucher

Gliederung

1. Einleitung und Zielsetzung

Das Projekt soll zeigen wie künstliche Intelligenz in Verbindung mit virtuellen Simulationen zur Lösung von bestimmten Problemen in der Realität verwendet werden könnte.
Konkret wird ein genetischer Algorithmus verwendet um das Bewegungsmuster eines vorausgesetztes, physikalisch simulierbaren Modells eines simplen Flügelpaares zu generieren und zu optimieren.
Im Anfangszustand ist das Modell in der Simulation nicht flugfähig.

Durch Simulationen mit jeweils anderen Bewegungsmustern und Flügelformen, den sogenannten Chromosomen werden die entsprechenden Muster auf ihre Leistung hin untersucht.
Mit Leistung wird in diesem Zusammenhang der Auftrieb den der Flügel durch Vorführung eines Bewegungs-Zyklus bewirkt hat.

Die klassischen Methoden der genetischen Informatik, wie Mutation, Kreuzung und Selektion dienen anschließend zur Errechnung neuer Chromosome für die nächste Generation.
Generation für Generation strebt der Algorithmus somit eine Erhöhung der Leistung an.

Der Praxisbezug ergibt sich aus der Übertragbarkeit des Konzepts auf zahlreiche, reale Objekte bzw. Maschinen.
Das Resultat sind lernfähige Roboter die ebenso wie das Modell, durch "Trial and Error" nach und nach entweder ein bestimmtes Ziel erreichen oder ihre Fähigkeit im Bezug zu einem Ideal optimieren.

1.1 Mitgelieferte Dateien

Das Archiv in dem sich diese Dokumentation befindet, enthält zwei vorkompilierte Varianten des Programms im Unterordner "Builds":
GenisysOfDragons_v1_Low.exe und GenisysOfDragons_v1_High.exe.
Der einzige Unterschied sind Starteinstellungen. Die zweite Variante hat eine Startpopulation von 48 statt 24 wie die Low Variante und empfiehlt sich somit eher auf leistungsfähigeren Computern mit dedizierter Grafikkarte.
Das Rendering ist jedoch in beiden Varianten aktiv. Um schneller Resultate zu erhalten, empfiehlt es sich, das Rendern zu deaktivieren und die Population weiter zu erhöhen.

Darüber hinaus gibt es den Ordner "GenisysOfDragons v1.0" welches die Projektdateien für Unity enthält. Im Folgenden Punkt wird beschrieben, wie man ein solches Projekt öffnet.

2. Programmstruktur von Unity

Die Basis des gesamten Softwareprojektes bildet die hoch entwickelte Spiele-Engine „Unity“.
Unter „Spiele-Engine“ oder „Game-Engine“ versteht man in diesem Zusammenhang, eine Softwareumgebung welche das Verwirklichen eines Projektes vereinfacht indem sie dem Entwickler viel grundlegende Arbeit abnimmt.

Darunter fällt z.B. die hardwarenahe Programmierung - Unity bietet gleich sehr weitreichende Hardwareunabhängigkeit und unterstützt zahlreiche Plattformen – aber auch die Berechnung physikalischer Objekte sowie die Darstellung in 3D inklusive Dinge wie Schattenwurf, Lichtspiegelung, etc.

Unity ist leider nicht Open Source (was zur Folge hat, dass gewisse tiefere Änderungen nicht möglich sind) aber als „Personal Edition“ welche für dieses Projekt ausreicht, frei verfügbar und kann von www.Unity3D.com heruntergeladen und installiert werden. Neben der dortigen Software, ist nur noch eine Version von Microsoft Visual Studio für C# notwendig.

Nach der Installation kann das Programm gleich gestartet werden. Es wird eine Registrierung angeboten, aber es kann auch offline gearbeitet werden. Oben rechts im anschließenden Fenster wird ein bestehendes Projekt geöffnet werden.
Dazu einfach den Ordner aus dem entpackten Archiv auswählen, welches zu dieser Dokumentation gehört.

Dies öffnet den eigentlichen, sogenannten Editor von Unity.
Dieser ist grob in 4 Bereichen eingeteilt:

- Links die „Hierarchy“ diese zeigt alle Objekte (in Unity, „GameObejct“ genannt) welche sich momentan in der sogenannten Szene („Scene“) befinden.

- Die große Mitte nimmt eine inaktive Variante der Szene ein. Eine Szene kann man sich am ehesten als Level eines Spiels vorstellen. Es ist die vollständige Welt welche zu einem Zeitpunkt dargestellt werden kann. Mittels rechter Maustaste kann man sich darin umsehen. Mit dem schwarzweißen Pfeil über der Szene wird dies dann gestartet. Generell ist es aber wichtig zu wissen, dass das Ausführen einer Szene innerhalb des Editors, einen sehr großen Overhead im Vergleich zur Ausführung eines „Builds“, also einer Exe-Datei (auf Windows). Unter „File -> Build & Run“ kann mit beliebigen Szenen die Exe generiert werden.

- Unten gibt es die zwei Reiter „Projekt“ und „Console“. Die „Console“ zeigt Fehlermeldungen und Debug-Zeilen an. Unter „Projekt“ können alle Komponenten und Skripte des Projekts angezeigt werden. Zur Verzeichnisstruktur, siehe unter „1.1 Prefabs, Components und Scripts“.

- Die rechte Seite wird vom sogenannten „Inspector“ eingenommen. Dieser erlaubt es den momentanen Zustand der Objekte aus der „Hierarchy“ oder den Elementen aus dem „Projekt“ anzuzeigen und zu verändern.

2.1 Prefabs, Components und Scripts

Alle Bestandteile aus denen ein Projekt besteht, werden in Unity unter dem Begriff „Assets“ zusammengefasst. Entsprechend nennt sich so das Hauptverzeichnis welches man unten im Editor unter „Projekt“ aufrufen kann.

Die Arbeit dieses Softwareprojektes befindet sich ausschließlich im vormals leeren Unterverzeichnis „Resources“ sowie in „Scenes“. Die anderen Verzeichnisse enthalten Standarddaten welche zu Unity’s Demo-Projekten gehören. Daraus werden zur schöneren Darstellung, eine Handvoll Texturen bzw. Grafiken verwendet.
Es findet jedoch kein fremdes oder externes Skript Verwendung.

Um den Inhalt der Ordnerstruktur unter „Resources“ verstehen zu können, muss man die verschiedenen Asset-Arten kennen.
Wichtig sind zum einen die sogenannten „Prefabs“. Diese können als Prototypen für GameObjects verstanden werden (und werden später mit der Funktion „*Instantiate()*“ in ein solche umgewandelt). Generell kann man jedes Prefab einfach mit der Maus in eine Szene ziehen, so dass ein GameObjekt in der Hierarchie erscheint (wo es durch Rechtsklick auch gelöscht werden kann).
Prefabs selbst gibt es jedoch nicht in der Szene – es sind immer gewissermaßen Instanzierungen davon.

Klickt man ein Prefab an, z.B. den „DragonBody“ unter „Dragon -> Body“, erscheint sofort rechts im Inspector eine Detailansicht, welche die sogenannten „Components“ (hier meist Komponente genannt) untereinander beinhaltet.

Jedes Prefab hat standardmäßig die Komponente „Transform“ welche die Koordinaten, Skalierung und Rotation angibt. Weitere oft verwendete Komponenten ist der „Mesh Filter“, also ein 3D Modell (Vorschau im Icon des Prefabs) dessen Textur durch eine weitere Komponente vorgegeben und durch einen „Mesh Renderer“ in der Szene dargestellt wird wird.

Ebenso werden „Box Collider“ und „Rigidbody“ verwendet welche dem Objekt ein physikalisches Verhalten (Trägheit, Kollisionen, optional Schwerkraft usw.) geben.

Übrigens werden die Gelenke der Flügel im Projekt durch die Komponente „Hinge Joint“ realisiert. Diese werden jedoch vollständig zur Laufzeit generiert und sind deshalb jetzt noch nicht sichtbar.

Gleich neben dem „DragonBody“ befindet sich das zweite, wichtige Element in den Verzeichnissen, ein Skript bzw. eine Klasse. Diese können ebenfalls als Komponente an ein Objekt gehaftet werden. In dem Fall kann mittels „*GetComponent<SCRIPTNAME>()*“ die Instanz dieses Skripts (also ein einfaches C# Objekt) ermittelt werden. Statische Klassen und Funktionen dagegen sind auch ohne Verknüpfung an ein GameObjekt verfügbar.

Diese Vorgehensweise ist z.B. in der Statischen Klasse „Global“ zu sehen welche die erwähnte Funktion benutzt um Objekte von verschiedenen Skripten durch variablen verfügbar zu machen.

Sofern eine Klasse von „MonoBehavior“ erbt, werden bestimmte Funktionen automatisch aufgerufen. Z.B. „*Start()*“, „*Awake()*“ und „*Update()*“ (ausgeführt während jedem neuen Rendern des Bildschirms) sowie „FixedUpdate()“ (ausgeführt wenn die Physik upgedatet wird).

3. Ordnerstruktur

Es folgt eine Übersicht über die verschiedenen Verzeichnisse unter „Resources“. Für Details, siehe die Kommentare am Anfang jedes Skriptes.

|**AI**| (Enthält die Bausteine für den genetischen Algorithmus sowie zwei Kontrollskripte)

|**Dragon**| (Enthält alle Objekte und Skripte die direkt einen Drachen bzw. dessen Flügel repräsentieren)

- |**CompleteDragon**| (Enthält die Komponenten für das vollständige Drachenmodell das für den Medianight Trailer verwendet wurde. Das Skript in diesem Ordner enthält noch die Aktionen für den Trailer)
- |**SimplifiedDragon**| (Der Körper in einer effizienter renderbaren Rechteckform, das Knochensegment der Flügel, sowie zwei Kontrollskripte)

|**DragonWorld**| (Die Bestandteile und Kontrollskripte der restlichen Umgebung in den Szenen)

- |**Camera**| (Objekte für die Kamera welche für die Darstellung gebraucht wird. Die Skripte kontrollieren auch Mausfunktionen.)
- |**Controlers**| (Wichtige Kontrollskripte und Helfer-Objekte)
- |**GUI**| (Alles für die 2D-Oberfläche zur Steuerung sowie die Anleitung)
- |**Surroundings**| (Weitere Objekte zur Darstellung der Welt)

|**Factories**| (Statische Klassen zur Generierung von Dingen wie Polygonen, den Drachen u.ä.)

|**GlobalClasses**| (Statische Klassen für globale Variablen bzw. Einstellungen)

|**Other**| (Sonstiges; Teils ohne Verwendung)

3.1 Scenes

Der Ordner „Scenes“ enthält drei Szenen die Bestandteil des Projektes sind. Wie bereits zuvor erwähnt, repräsentieren Szenen, gewissermaßen Startzustände des Gesamtprojekts.
Ein Doppelklick öffnet die betreffende Szene.

- |**CliffSimulation**| Stellt die Szene für den Trailer dar und enthält ein weites Terrain sowie ein Exemplar des vollständigen Drachens. Mit der Maus und den WASD Tasten kann man sich nach einem Doppelklick herum bewegen und Teile des Drachens ziehen und schieben.
  Für Näheres zur Steuerung, das Tutorial in der „MainScene“ lesen.
  Durch Drücken von STRG kann man schritt für schritt durch die Teile des manuellen Medianight Trailers durchgehen (Achtung, relativ fehleranfällig).
- |**MainScene**| Ist die eigentliche Anwendung welche den genetischen Algorithmus des Projektes darstellt und präsentiert. Es enthält ein selbsterklärendes GUI mit tooltips sowie ein Tutorial welches automatisch startet. Fast der gesamte Inhalt dieser Scene wird zur Laufzeit durch Code generiert, deswegen ist vorab im Editor fast nichts zu sehen.

|**TestingScene**| In dieser Szene wurden ein paar Physikelemente getestet. Sie ist eigentlich obsolet und nicht Teil des Projekts.

Das Projekt ist darauf ausgelegt sowohl mit der „CliffSimulation“ als auch der „MainScene“ starten zu können. Mittels der Taste <F> kann zur Laufzeit gewechselt werden. Im Fall eines „Builds“ müssen dafür jedoch beide Szenen in den „Build Settings“ übernommen werden.

4. Grundprinzipien des genetischen Algorithmus

Genetische Algorithmen generell versuchen das Prinzip der Evolution aus der Natur zu übernehmen bzw. von dessen Prinzipien zu profitieren.
Konkret bedeutet dies, das wünschenswerte Ziel - in diesem Fall die Flugtauglichkeit eines Flügelmodells – durch gezielte Versuche von möglichen Werten bzw. Modellen zu erreichen.
Im Gegensatz zu einer rein zufallsbasierten Versuchsreihe, bilden die Ergebnisse der vorausgegangenen versuche, die Basis für die neue Versuchsreihe.
Dies hat die Absicht, zu ermitteln was, mit Hinsicht auf das Ziel, „gut“ war und was nicht, damit über einen Zeitraum hinweg, sich die Ergebnisse zunehmend dem Ziel nähern.

4.1 Wichtige Begriffe: Population, Phenotype usw.

Das was vorhin als Versuchsreihe bezeichnet wurde, heißt in der Welt der genetischen Algorithmen (sowie in diesem Projekt und dessen Code) „*Population*“.
Sie beschreibt somit die Gesamtheit aller „*Individuen*“ bzw. im Code meist „*Samples*“, welche ihrerseits jeweils ein einzelnes Exemplar der zu testenden Sache darstellen.

In der Fachsprache ist „Individuum“ und „Sample“ oft gleichbedeutend mit „*Phenotype*“, ein Begriff aus der echten Biologie welcher sich auf den genetischen Bauplan eines jeden Lebewesens bezieht. Genau an diesen Unterschied angelehnt, beschreibt im Code dieses Projekts, das „*Phenotype*“, einen konkreten Datensatz, der zur Erzeugung eines konkreten Individuums dient.

In der hiesigen Umsetzung besteht, wie in der Natur, ein solcher Phenotyp aus jeweils einer Reihe von sogenannten „*Chromosomes*“ (plus ein par zusätzliche Daten).
Bei der Generierung eines Samples aus dem gegebenen Phenotyp, bestimmen die Chromosome, die Beschaffenheit jedes einzelnen Flügel-Segmentes (im Programm als Flügelknochen sichtbar) sowie dessen lokales Bewegungsmuster.

Zusammenfassung:

Population: Gesamtheit aller Individuen/Samples. Siehe Skript „*PopulationControler*“.

Individuen/Samples: Konkretes Versuchsmodell. Siehe Skript „*DragonFactory*“

Phenotyp: Datensatz zur Generierung eines bestimmten Versuchsmodells. Siehe Skripte „*Phenotyp*“ und „*PhenotypFactory*“.

Chromosom: Austauschbare Daten-Komponente eines *Phenotyps*. Siehe Skripte „*Chromosome*“ und „*DragonFactory*“.

4.2 Fitness: Untersuchung eines Samples

Ein weiterer Kernbegriff der genetischen Algorithmen, ist die sogenannte „Fitness“. Sie beschreibt wie gut ein konkretes Individuum bzw. dessen zugrundeliegende Phenotyp, hinsichtlich dem Erreichen des Ziels, abschneidet.
Im Falle dieses Projektes geschieht dies durch eine physikalische Simulation des generierten Flügelpaares bzw. des vereinfachten Drachens.

Nach einer bestimmen Anzahl von Flügelschlägen wird ermittelt welche Höhe das Modell im Vergleich zum Startpunkt (immer 0) erreicht hat. Darüber hinaus werden weitere Werte festgehalten wie die Gesamtfläche der Flügelmembran.

Wie bereits erwähnt, wird das Modell immer aus den austauschbaren Chromosomen generiert. Entsprechend werden auch die Chromosome anhand ihres bewirkten Auftriebs, einzeln beurteilt und haben dadurch eine eigene „Fitness“ welche bei der Generierung von neuen Phenotypen relevant wird und mit in die Fitness des Phenotyps einfließt.

Zur Konkreten Berechnung der Fitness, siehe im Skript „*PopulationControler*“, die Datenstruktur „*EvaluatedPhenotype*“.

5. Erhalt der Population

Zwecks maximaler Flexibilität und Performance läuft die Generierung und Untersuchung (Simulation) der Samples asynchron ab. Einzig die Größe der Population ist fest.

Das Ende der *UpdatePhysics()* Funktion des WingSet (im gleichnamigen Skript) beinhaltet das Abbruchkriterium der Flügelsimulation und sendet die vorbereiteten Daten zur Analyse, an den PopulationControler mittels „*EvaluateSample()*“.

Dieser Controler führt die Analyse und Bestimmung der „Fitness“ durch und speichert das Resultat in eine, nach Fitness sortierte Liste (in C# ein „*SortedDictionary*“), zusammen mit den benötigten Daten. Anschließend wird das zugehörige Sample sowohl aus der Simulation als auch aus der Population entfernt.

Das Resultat ist somit eine freie Stelle in der Population (festgehalten in der Liste „*freeIndices*“). Entsprechend wird mittels „*GeneratePhenotypes()*“ das Signal an den „AIControler“ (Siehe gleichnamiges Skript) gewissermaßen der Auftrag gegeben, neue Phenotypen bereit zu stellen welche in eine weitere Liste, der „*readyPhenotypes*“ gespeichert werden. Zwecks mehr Details bezüglich der Generierung, Siehe Punkt 5.1.

Noch im Zuge der Evaluierung des beendeten Samples werden abschließend mittels „*AddPossibleSamples()*“ alle bestehenden Lücken in der Population gefüllt.
Enthält die „*readyPhenotypes*“ Liste nicht genug Einträge, werden Zufalls-Phenotypen verwendet. Auf diese Art und Weise wird die Population auch bei Programmstart oder bei manueller Änderung der Populationsgröße generiert.

Das Ziel dieser Entkopplung zwischen Sample-Generierung und Phenotype-Generierung liegt in der Flexibilität besonders mit Hinblick auf mögliche Anwendung auf verteilten Systemen oder MultiThreading.
Unglücklicherweise unterstützt die freie Version von Unity, nicht die benötigte Funktionalität dafür.

5.1 Generierung neuer Phenotypen

Phänotypen werden nur generiert wenn genug auf der Warteliste

Der "AIControler" enthält eine Reihe von Einstellungen für das Bereitstellen neuer Phänotypen was durch die Funktion „*GenereatePhenotypes()*“ geschieht.

Das Array "GenerationRules" enthält dafür eine Reihe von vorgefertigten Befehlen bzw. Regeln zur Berechnung entsprechender, neuen Phänotypen auf Basis der Fitness von gegebenen Datensätzen.

Zunächst wird per Zufallsgenerator („*Random()*“) je nach zugehöriger Wahrscheinlichkeit, eine der Regeln ausgesucht. Die Wahrscheinlichkeiten sind dabei so gesetzt, die Regeln welche hohe Fitness bevorzugen, häufiger anzuwenden. Besonders kleinere Fitness Werte sollen aber trotzdem zum Einsatz kommen, was mir aufgrund der sehr starken Abhängigkeit zwischen den Bestandteilen eines Phänotyps (also zwischen den Chromosomen) als sinnvoll erscheint.

Es wird nach Zufall mit einer bestimmten Wahrscheinlichkeit, entweder einer der besten Phänotypen gewählt, oder einer der schlechtesten. Zusammen mit den negativ orientierten Regeln soll dies eine höhere Varianz der Ergebnisse bewirken. Bei theoretischer Anwendung auf einer zweidimensionalen Gleichung, würde dies verhindern dass nur lokale Maxima gefunden werden.

Wird ein schlechter Phänotyp gewählt, wird mit einer weiteren Wahrscheinlichkeit, die Timesteps invertiert.

Je nach Anforderung der Gewählten Regel, werden entsprechende Phenotypen aus der getesteten Liste heraus gesucht und die Regel darauf angewendet. Da die Liste eine limitierte Größe hat, nach Gesamt-Fitness sortiert ist und die Auswahl eine probalistische Tendenz hat, die höhere Fitness auszuwählen, findet hier automatisch die Selektion statt.

Mögliche Regeln (mit einigen Variationen) sind:

--> P1 * P2 --> Zufällige Kreuzung
--> P1 + P2 --> Maximierende Kreuzung (Chromosom größerer Fitness hat Vorrang)
--> P1 - P2 --> Minimierende Kreuzung (Chromosom kleinerer Fitness hat Vorrang)

--> INVERT_TIMESTEP(P1 - P2) --> Minimierende Kreuzung (Chromosom kleinerer Fitness hat Vorrang) mit Invertierung
--> P1 / P2 --> Kreuzung zur Hälfte von P1 und Hälfte P2

--> MAX(P1, P2, P3, …) --> Maximierende Kreuzung über die Chromosome von beliebig vielen Phänotypen
--> MIN(P1, P2, P3, …) --> Minimierende Kreuzung über die Chromosome von beliebig vielen Phänotypen

Anschließend wird abhängig von den Einstellungen, eine Anzahl an Mutationen angewendet. Mögliche Mutationen:

--> MUTATE_FLAPTEPS(P) --> Variiert die flapsteps (Zeitpunkt der Muskelaktion)
--> INVERT_ FLAPTEPS (P) --> Invertiert flapsteps

--> MUTATE_FLAPLENGTH(P) --> Variiert Dauer eines vollständigen Flügelschlags

--> MUTATE_LENGTH(P) --> Variiert Segment-Länge

--> FULL_RANDOMIZE(P, 0.5) --> Variiert alles mit jeweils 50% Wahrscheinlichkeit
--> FULL_RANDOMIZE(P, 0.05) --> Variiert alles mit jeweils 5% Wahrscheinlichkeit

6. Herausforderungen und Probleme

Das Projekt hat mich vor einigen Herausforderungen gestellt.

Mein Wissen bezüglich den genetischen Algorithmen war relativ begrenzt, aber dank zahlreicher Online Ressourcen konnte ich mir schnell das grundlegende Verständnis aneignen.

Alles weitere geschah dann Schritt für Schritt und experimentell, allerdings ohne konkretes Code Vorbild für den Zweck des Projektes.

Die Engine selbst bot durch seine Grenzen ebenfalls Probleme die es zu lösen galt. Im Bereich der 3D Darstellung musste ich selbst die Engine ergänzen, da das System kein einfaches System zum Darstellen von flexiblen Polygonen enthielt, wie es für die Flügelmembran notwendig gewesen wäre. Das Resultat war das "PolySet" im gleichnamigen Skript.

6.1 Physik und Simulation

Die größte Herausforderung war die Umsetzung der Simulation selbst. Zwar nimmt die integrierte Physik Engine von Unity einiges an Arbeit ab, birgt aber auch selbst auch Probleme da sie nicht intern veränderbar ist.

So schlug ein Ansatz mit sogenannten "Springs" welche gewissermaßen die Flexibilität von Gelenken simulieren, fehl. Als Verbindung zwischen den einzelnen Flügelsegmenten führten sie beim Anwenden der notwendigen Kräfte, zum Zerreißen des Modells.
Dies ist ein bekanntes Phänomen dieser Art von Engines.

Als Alternative waren daher nur starre "Joints" verfügbar. Dazu gibt es mehr Informationen in den Code-Kommentaren.

Weiterhin führt die Physik zur Selbszerstörung des Modells. Dies wird jedoch vom System gehandhabt und die entsprechenden Samples zerstört und durch neue ersetzt.

Ein weiterer Makel solcher, für Games optimierter Engines, ist das Fehlen von echtem, flächenabhängigem Luftwiderstand.

Objekte ("Rigidbodies") können zwar durch eine generelle Reibung, als "drag" bezeichnet, verlangsamt werden, würden aber die Selben Kräfte erfahren, egal in welchem Winkel der Flügel steht, was das Ganze für diesen Anwendungszweck im Projekt unbrauchbar macht.

Stattdessen musste ich, wie im Skript "WingSet" zu sehen, selbst diese Kräfte berechnen lassen.

Herausgekommen ist zwar nicht etwas das akkurat den Gesetzen der Aerodynamik folgt - das würde auch extrem viel Leistung beanspruchen - aber es scheint für den Zweck zu reichen.

7. Fazit

Das gesteckte Ziel der Visualisierung eines genetischen Algorithmus in der Simulation eines Scenarios mit Realitätsbezug wurde meiner Meinung nach erreicht.

Allein die Zuverlässigkeit der Software ist nicht wirklich zufriedenstellend. Zwar schaffen es viele Samples, Auftrieb zu erzeugen, ob aber ein ansehnliches Ergebnis dabei rauskommt, hängt sehr vom Zufall ab und ist manchmal auch mit längerer Laufzeit nicht gegeben.
Zudem produziert die Physiksimulation von Unity im Zusammenspiel mit den „Joints“ teils unvorhersehbare Ausreißer in den Resultaten.

Auf jeden Fall habe ich dabei äußerst viel lernen können. Nicht nur über den Umgang mit Unity sondern vor allem über die Entwicklung des genetischen Algorithmus, sowie entsprechender Software drum herum.

Es gab einiges an Herausforderungen. Einerseits die Recherche zwecks der richtigen Ansätze, aber andererseits auch der Umgang mit den Einschränkungen der Game Engine Unity.
Alles in allem war es sehr interessant mich mit solch einem Thema auseinander zu setzen und kann so etwas nur empfehlen.
Eine andere Engine als Grundlage zu nehmen, könnte jedoch sinnvolles ein wobei die Möglichkeiten da leider generell eher eingeschränkt sind, sofern man auf Echtzeit-Rendering setzen will.

7.1 Denkbare Weiterentwicklung

Das Projekt könnte mit mehr Zeit und Ressourcen in einigen Richtungen verbessert werden.

Die Physikberechnung von Unity scheint generell, ebenso wie die Schnittstellen zu Skripten, single-core-optimiert zu sein. Auf aktuellen Maschinen könnte man das Programm also theoretisch, mindestens 4 mal starten und ohne Einbußen gleichzeitig laufen lassen (bei deaktiviertem Rendering).

Was fehlt ist somit nur ein Austausch-Mechanismus welcher die Informationen zwischen den Instanzen übermittelt. Dazu wurde die asynchrone Generierung von Phänotypen (Siehe Punkt 5) bereits vorbereitet.

Eine Globalisierung der beiden, dort erwähnten Listen bzw. Datenstrukturen, würde den gewünschten Effekt erzielen.

Der Kern des gesamten Projektes sind die „Generation Rules“ aus Punkt 5.1 welche für den eigentlichen Fortschritt des genetischen Algorithmus verantwortlich sind.
An diesen könnte noch viel Optimierungsarbeit zum schnelleren Finden eines „guten“ Ergebnisses geleistet werden. Andererseits bieten die Regeln auch eine gewisse Universalität und ermöglicht das Lösen der Problemstellung eventuell auch bei einer modifizierten Simulation.

Selbstverständlich kann die Visualisierung ebenso verbessert werden, besonders für die Vorführung beim Testen nach der Lernphase. Die momentane Version nutzt kein richtiges 3D Modell sondern

wurde der Einfachheitshalber, aus Kisten konstruiert. Die Verbindung mit einem sogenannten „rigged 3D model“ wäre denkbar und ein sauberes Resultat zu erhalten.

Darüber hinaus könnte die Physik bzw. Simulation optimiert und akkurater bzw. realitätsnäher gestaltet werden.

Ein paar dieser Punkte werde ich eventuell in Zukunft für eigene Zwecke angehen.

8. Links und Quellen

Als Quellen diente mir hauptsächlich das Unity Manual: http://docs.unity3d.com/Manual/index.html

Sowie das entsprechende Forum: http://forum.unity3d.com/

Die Basis für die Umsetzung des genetischen Algorithmus war der Kurs über Künstliche Intelligenz an der Hochschule der Medien, so wie die Recherche in der Vergangenheit zu diesem Thema für eine „Wissenschaftliche Arbeit“.
Bitte sehen deswegen in das zugehörige Quellenverzeichnis:
https://dl.dropboxusercontent.com/u/69344499/Genetische%20Algorithmen.pdf

Zu weiteren Fragen bezüglich der Umsetzung oder Dokumentation bin ich jederzeit bereit.