GameOfEvolution

Individuelle Praktische Arbeit

Kantonsschule Frauenfeld

Autor: Patrick Zürcher

Datum: 24.02.2022

Inhalt

1 Grundangaben

1.1 Aufgabenstellung

Game of Evolution soll eine einfache Simulations-Umgebung sein mit welcher mit verschiedenen Attributen wie Temperatur, Biom und Essen eine Selektion angestrebt wird.
Im weiteren Verlauf sollte auch noch eine KI ähnliche Selektion und somit Evolution stattfinden.
Diese Punkte sind jedoch schwer innerhalb des Zeitrahmens erreichbar und liegen somit ausserhalb des Umfangs des Projektes.
Zur Umsetzung soll C++ verwendet und damit verbundene Bibliotheken und Programmierumgebungen.
Zum grösstenteils wird die SFML Library verwendet.
Der Code wird in English beschrieben und geschrieben.

1.2 Projektorganisation

Vorgesetzter: Jean-Pierre Mouret

Arbeiter: Patrick Zürcher

1.3 Deklaration der Vorkenntnisse

- C++: Noch nie bevor genutzt
- SFML: Noch nie zuvor genutzt, erste library, die auf die Grundfunktionen zum Zeichnen zurückgreift.
- Visual Studio: Bereits während dem Erlernen von C# reichlich genutzt.

1.4 Deklaration der Vorarbeiten

Keine

1.5 Deklaration der benützten Firmenstandards

Keine

2 Zeitplan

Der Zeitplan ist in 8 Wochen aufgeteilt pro Woche sollte etwa über 5 Tage mit jeweils 2 Stunden gearbeitet werden.

2.1 Soll-/Ist-Zeitplan

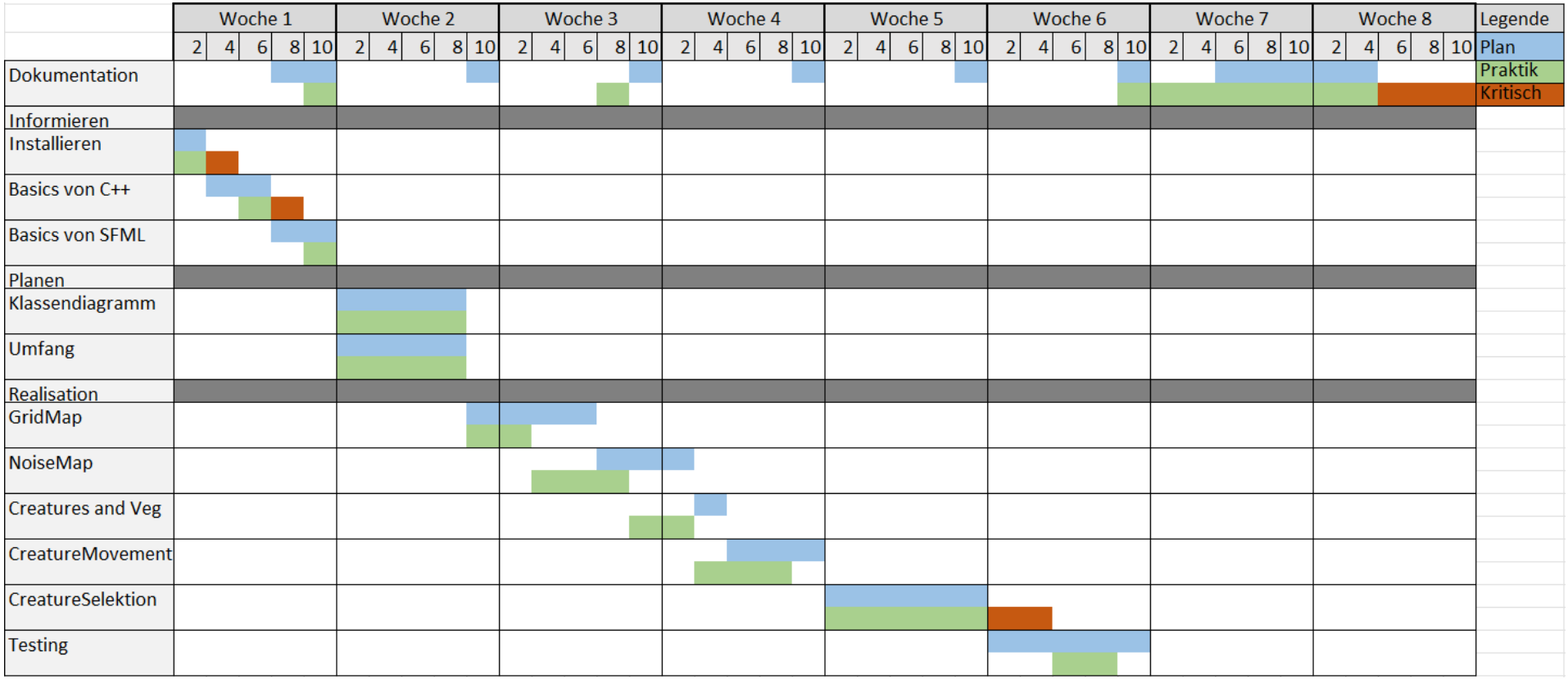


Abbildung 1 Zeitplan

3 Arbeitsjournal

3.1 Woche 1

Datum	Woche 1
Aktivität	
Soll	Installation der Programmierumgebung und Synchronisation mit GitHub. Testprogramme von SFML und C++ zum Testen der Neuheiten.
Ist	Programmierumgebung installiert. Synchronisation mit GitHub funktioniert. Testprogramme von SFML und C++ durch Tutorial erlernt.
Ergebnisse	
Erfolge	Die Testprogramme haben gut funktioniert und ich habe die Grobheiten von C++ schnell verstanden.
Misserfolge	Das Installieren hat sich durch ein kleines Problem verzögert und die GitHub Funktionalität hat zu Beginn nicht funktioniert.
Fazit	Durch ein wenig Überzeit konnte ich mich im Zeitplan halten destotrotz hat die erste Woche gut funktioniert
Überzeit	2H
Es wurden 2 Stunden mehr investiert, um ein wenig C++ auszutesten	

3.2 Woche 2

Datum	Woche 2
Aktivität	
Soll	Grundplanung der Simulation und wie viel diese beinhalten soll und darauf basierend Erstellung eines Klassendiagram.
Ist	Umfang geplant und Klassendiagram gezeichnet.
Ergebnisse	
Erfolge	Ich konnte eine relativ gute Planung der Simulation umsetzen.
Misserfolge	Das Klassendiagram ist wahrscheinlich noch ein wenig unzureichend, da die genaue Umsetzung via C++ noch nicht sehr bewusst ist.
Fazit	Die Woche lief gut und es war beinahe zu viel Zeit zur Verfügung.
Überzeit	-

3.3 Woche 3

Datum	Woche 2
Aktivität	
Soll	GridMap und NoiseMap umsetzung.
Ist	GridMap und NoiseMap umgesetzt
Ergebnisse	
Erfolge	Die Umsetzung der Map ging sehr Flott voran.
Misserfolge	Bei der Umsetzung der NoiseMap hatte ich eine längere Debug-Zeit als erwartet, da ich x und y Werte vertauscht habe. Ebenfalls war der Ladevorgang der Map sehr ineffizient und ich musste diesen überarbeiten.
Fazit	Neben der etwas Überzeit durch Debuggen lief alles gut.
Überzeit	1H
Ich hatte eine Stunde für extra Debugging.	

3.4 Woche 4

Datum	Woche 2
Aktivität	
Soll	Creature und Creaturemovement implementiert.
Ist	Creature und Creaturemovement implementiert. Struktur Überarbeitung.
Ergebnisse	
Erfolge	Die Umsetzung von Creature, Vegies und deren Movement war sehr einfach. Ebenfalls habe ich die Zeit genutzt den Code zu überarbeiten um eine klare Trennung zwischen Rendering und Coding zu definieren. Für eine vereinfachten Umsetzung neuer Creatures wurde die Klasse Species erstellt um einen Blueprint zu ermöglichen.
Misserfolge	Bei der Umsetzung der NoiseMap hatte ich eine längere Debug-Zeit als erwartet, da ich x und y Werte vertauscht habe.
Fazit	Neben der etwas Überzeit durch Debuggen lief alles gut.
Überzeit	1H
Ich hatte eine Stunde für extra Debugging.	

3.5 Woche 5

Datum	Woche 2
Aktivität	
Soll	Umsetzung Selektion zwischen Creatures
Ist	Umsetzung Selektion zwischen Creatures vereinfacht fertig. Jedoch treten noch kleine Fehler auf.
Ergebnisse	
Erfolge	Die Implementation einer Hitbox zwischen mehreren Creatures war sehr einfach.
Misserfolge	Das entfernen von Creatures funktioniert noch Fehlerhaft. Wahrscheinlich wurde ein Edge-Case nicht beachtet. Es wird in der nächsten Woche nochmals Zeit benötigt.
Fazit	Es gab viele Fehler und das Debugging war bisher noch nicht sehr erfolgreich. Die Simulation hat bisher auch noch einen recht einfachen Aufbau.
Überzeit	2H
Ich hatte zwei Stunden für erfolgloses Debugging.	

3.6 Woche 6

Datum	Woche 2
Aktivität	
Soll	DeltaTime implementieren & Selektion nachholen
Ist	DeltaTime implementiert & Selektion nachgeholt.
Ergebnisse	
Erfolge	Durch endloses Debuggen konnte ich den Fehler ausfindig machen und konnte die Simulation zum Laufen bringen. DeltaTime konnte sehr schnell implementiert werden, durch meine bisherige Erfahrung in Unity.
Misserfolge	-
Fazit	Selektion Fehler sind gefixt und DeltaTime war rasch implementiert.
Überzeit	-
-	

3.7 Woche 7

Datum	Woche 2
Aktivität	
Soll	Testen
Ist	Getestet
Ergebnisse	
Erfolge	Test
Misserfolge	-
Fazit	Beim Testen gab es keine grösseren Probleme.
Überzeit	-
-	

3.8 Woche 8

Datum	Woche 2
Aktivität	
Soll	Dokumentation Fertig
Ist	Dokumentation Fertig
Ergebnisse	
Erfolge	-
Misserfolge	-
Fazit	-
Überzeit	-
-	

4 Informieren

Im Folgenden werden die verschiedenen Möglichkeiten zur Umsetzung des Projekt analysiert.

4.1 Analyse Aufgabenstellung

Für das Projekt werden von mir selbst folgende Anforderungen gesetzt:

- Zufällig generierte Karte:
  Es soll möglich sein eine Karte automatisch zu erstellen. Die Grundidee ist die übereinanderlegen von verschiedenen NoiseMaps um eine einzigartige Map zu erstellen.
- Selektion von Creatures
  Die Simulation soll Kreaturen aufgrund von Variablen, die sich im Laufe der Simulation verändern selektieren und die Kreaturen Rauswerfen, die schlechter abschneiden.
- Darstellung der Simulation
  Es soll möglich sein die Simulation Grafisch darzustellen um so ein nützliches Nutzererlebnis darzubieten.
- Geschrieben in C++
  Das ganze Projekt soll in C++ geschrieben sein und somit die Besonderheiten dieser Programmiersprache beinhalten.

4.2 Benutzeranalyse

Da sich aus zeitlichen Gründe der Umfang der Simulation relativ klein haltet ist das Programm für normale Benutzer nicht sehr nützlich. Es sollte jedoch für Informatikinteressierte möglich sein das Projekt weiterzuführen und auf der Simulation aufzubauen. Somit ergeben sich folgende Ergebnisse.

- Zielgruppe: Informatikschüler
- Alter: ca. 15-18 Jahre
- PC-Kenntnisse Mittel
- Programmierkenntnisse Mittel bis Fortgeschritten

4.3 Konkurrenzanalyse

Da es sich bei dem Projekt nur um eine kleine Form von Simulation handelt gibt es keine direkte Konkurrenz.

5 Planen

Im folgenden werden Umsetzungsmöglichkeiten in Betracht gezogen, mit welchen man das Projekt umsetzen könnte.

5.1 Must-Have

In den Folgenden Punkte sind die zwingend benötigten Attribute des Projektes um die Mindestanforderungen zu erfüllen.

- Map Generation:
  Die Map soll basierend auf eine NoiseMap Vollständig Autonom erstellt werden.
- Kreaturen selektion:
  Die einzelnen Kreaturen sollen sich Fortpflanzen und Abschlachten um, so eine interessante Simulation zu ermöglichen.
- Darstellung:
  Die Simulation soll in einer 2D Grid-Welt abgezeichnet werden.

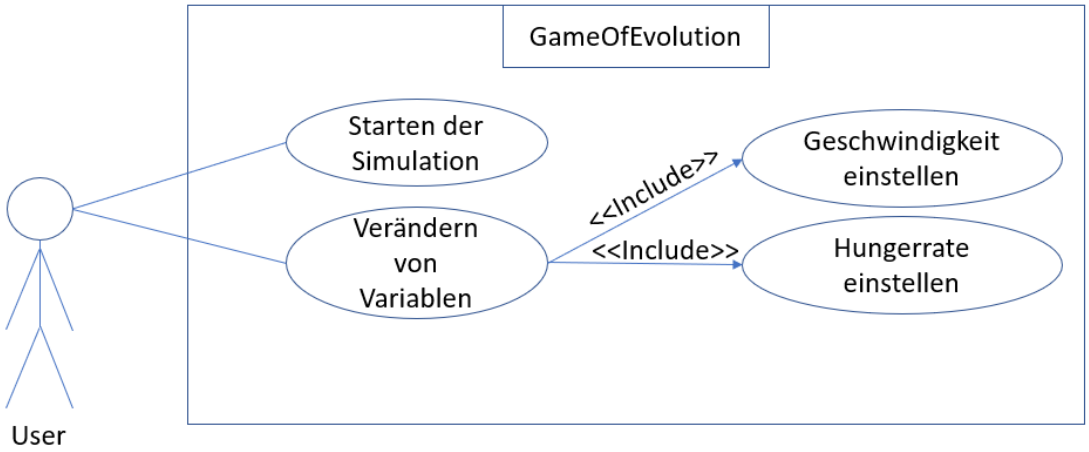
5.2 Nice-to-Haves

Im Folgenden mögliche Punkte aufgeführt, welche im Rahmen der SA womöglich nicht erfüllt werden können.

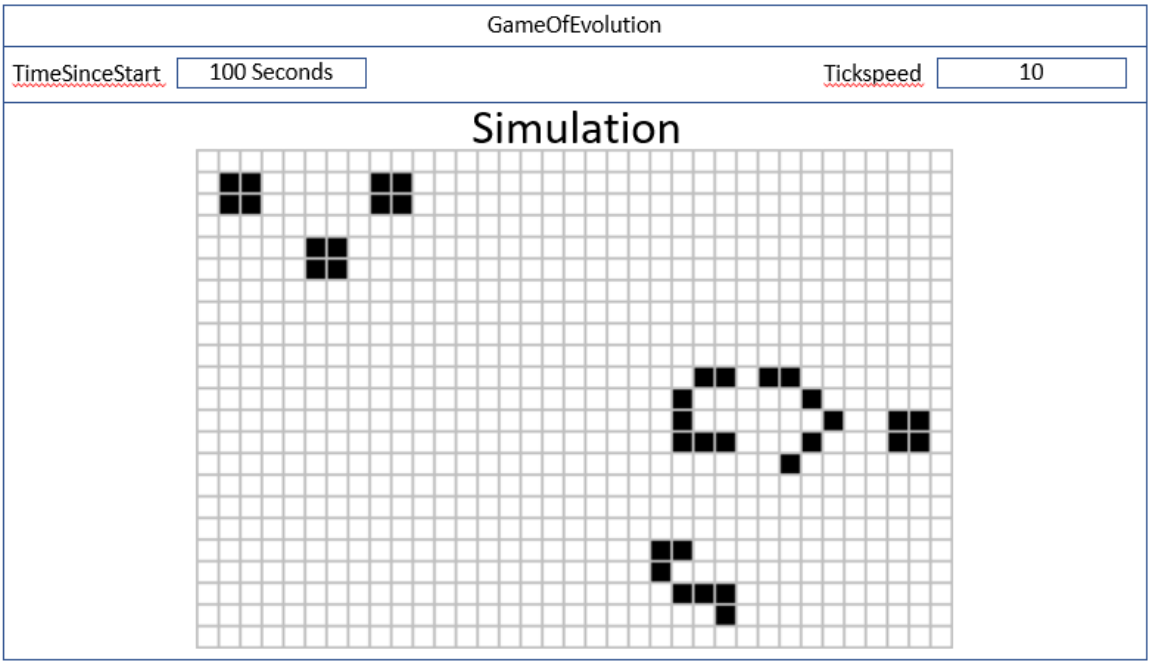
- Veränderbare Variablen während Laufzeit
  Da die Zeit für ein schönes Interface etwas
- Großartiges Interface
  Ein Interface soll es ermöglichen eine neue Welt zu starten oder auf einer bereits laufenden Variablen zu verändern.
- Sinnvolle Selektion
  Die Selektion könnte noch etwas sinnvoller und auf Grundlagen Realer Gegebenheiten basieren.

5.3 Use-Cases

Im folgenden Diagramm werden die Möglichkeiten des Users aufgelistet.



5.4 Mockup



5.5 Klassendiagram

5.6 Varianten

5.6.1 NoiseMap

Zur Umsetzung der Karte braucht es eine geeignete Karte es gibt hierbei 3 Möglichkeiten.

5.6.1.1 Eigene Noise Map

Eine Möglichkeit ist es, die Noise Map selbst zu schreiben, dies würde sehr viel Zeit brauchen und würde den Rahmen der SA schon bei sich selbst füllen.

5.6.1.1.1 Vorteile:

- Selbstgemacht
- Bringt viel Erfahrung

5.6.1.1.2 Nachteile:

- Möglicherweise Fehlerhaft
- Braucht viel Zeit

5.6.1.2 Perlin-Noise

Eine weitere Möglichkeit wäre eine bereits verfügbare Perlin-library zu verwenden. Perlin Noise ist eine Rauschfunktion die 1982 entwickelt wurde und Pseudo Gradienten Werte an Gitterpunkten.

5.6.1.2.1 Vorteile:

- Wenig Arbeit
- Funktioniert schnell und einfach

5.6.1.2.2 Nachteile:

- Nicht das Aktuellste

5.6.1.3 Simplex Noise

Simplex Noise ist die Aktuellste Möglichkeit eine Rauschfunktion auszuführen. Sie ist vergleichbar mit Perlin Noise jedoch mit geringerem Rechenaufwand.

5.6.1.3.1 Vorteile

- Schneller als Perlin Noise
- Wenig Arbeit

5.6.1.3.2 Nachteile

- Standardversion ist nicht Open Source

5.6.2 IDE

Zur Entwicklung braucht es ebenfalls noch eine passende IDE um das Codieren zu erleichtern. Hier kommen vor allem Visual Studio und Visual Studio Code in Frage.

5.6.2.1 Visual Studio

Visual Studio ermöglicht eine einfache Implementation, jedoch kann es auch Heavyweight im Vergleich zu Visual Studio und kann hier mehr Probleme machen.

5.6.2.1.1 Vorteile

- Einfache Installation
- Weniger Initialaufwand
- Vertraut

5.6.2.1.2 Nachteile

- Heavyweight
- Mögliche unlösbare Probleme

5.6.2.2 Visual Studio Code

Visual Studio Code ist im Vergleich zu Visual Studio sehr Lightweight jedoch brauch es einen höheren Initialaufwand und ich bin nicht so sehr vertraut mit Visual Studio Code.

5.6.2.2.1 Vorteile

- Lightweight

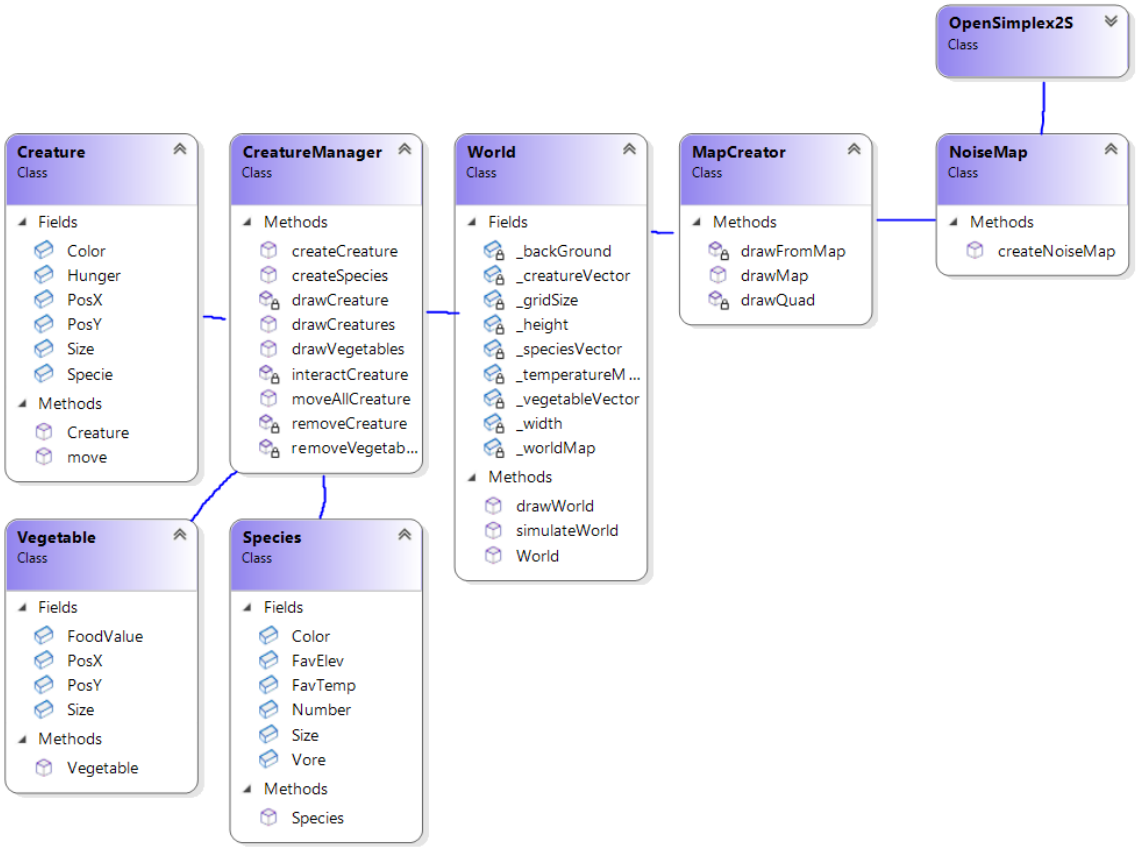
5.6.2.2.2 Nachteile

- Hoher Initialaufwand
- Wenig Vertrautheit

5.7 Gantt-Diagramm

Ein Gantt-Diagramm wurde bereits in der Zeitplanung aufgeführt.

5.8 Klassendiagramm



Das Klassendiagram wird unter Realisieren noch erläutert.

6 Entscheidung

In diesem Schritt wird zwischen den verschiedenen Varianten von der Planungsphase entschieden.

6.1 IDE

Als IDE entscheide ich mich für Visual Studio, da der Initialaufwand sehr wichtig für mich ist und ich die IDE schnell zum Laufen bringen will. Damit braucht es weniger Zeit, bis ich mit Programmieren beginnen kann.

6.2 Noise Map

Für die Umsetzung der Noise Map werde ich eine Simplex Noise Rauschfunktion verwenden. Da die Standard Library nicht open source ist, verwende ich eine Open Simplex Noise Library. Dadurch kann ich die Library so verwenden wie ich will.

7 Realisation

In diesem Teil wird der Programmierprozess dokumentiert.

7.1 Installation

Um mit dem Programmieren zu beginnen, müssen wir zuerst eine passende Programmierumgebung einrichten. Da wir uns für Visual Studio entschieden haben installieren wir dieses mit SFML. Wir folgen dem folgenden Tutorial, um SFML ins Visual Studios zu installieren. https://www.sfml-dev.org/tutorials/2.5/start-vc.php.

7.2 Aufbau

Das Programm soll so aufgebaut sein, das Rendering und Kalkulation klar voneinander getrennt sind. Dementsprechend halten wir in unserem Container World.cpp zwei Funktionen.

- drawWorld():
  Dient der Darstellung der Welt. Sie nimmt das Window und zeichnet die Spielwelt aufgrund von Parametern auf das Window.
- simulateWorld():
  Dient der Kalkulation der Welt. Hier werden die einzelnen Parametern bearbeitet, so dass eine interessante Simulation entsteht.

7.3 Kalkulation

7.3.1 Noise Map

7.3.1.1 Open Simplex Noise

Ein grosser Teil der Simulation ist die Umwelt. Hierzu verwenden wir eine Noise Map. Welche es ermöglicht ein komplexes Terrain ohne grossen Aufwand zu generieren. Durch die Überlappung mehrerer solcher Noise Maps können wir sogar verschiedene höhen und verschiedene Temperaturen implementieren. Welche die Simulation durch interessante Attribute erweitern kann.

Es gibt verschiedene Arten von Rauschfunktionen. Wie im vorherigen Teil deklariert habe ich mich für eine Open Simplex Library entschieden, welche den Vorteil hat das sie Open source ist.
https://github.com/deerel/OpenSimplexNoise

7.3.1.2 Implementation

Mit der eingesetzten Library ist es sehr einfach eine Noise Map zu erstellen. Man muss lediglich dem Rauschfunktion einen Seed geben, von der aus eine neue Map erstellt wird und nun mit einer doppelten For-Schleife durch den Vektor gehen und die Funktion nutzen um jeder Variablen den zugehörigen double Wert. Der double Wert kommt als eine Zahl zwischen 1 und -1. Nun muss der Wert noch in eine Zahl zwischen 1 und 0 konvertiert werden, indem wir die Zahl -1 und durch 2 Rechnen.

Durch Multiplizieren und Addieren der x und y Koordinate kann die Noise Map entsprechend angepasst werden. Wenn wir nun also die Koordinaten mal 2 rechnen, zoomen wir heraus. Diese Interaktionsmöglichkeiten werden im Unteren Artikel nochmals speziell erläutert.

https://www.redblobgames.com/maps/terrain-from-noise/

Um die Map noch etwas komplexer zu machen benutzen wir ebenfalls noch einen Temperatur Map, welche wir ebenfalls in die Simulation mit einfliessen lassen.

7.3.2 Creatures

Für die Kreaturen erstellen wir erstmal einen Vektor, in welche wir beim erstellen von World Kreaturen hinzufügen.

Die Kreaturen werden über die Klasse Spezies miteinander verbunden. In der Klasse Spezies werden Grunddinge wie Farbe und ID der Kreaturen gespeichert, um so eine Gruppe von Kreaturen zu erschaffen.

7.3.3 Selektion

Kommen wir nun zum interessanten Teil der Simulation.
Um in der Simulation einen interessanten Ablauf zu erhalten, müssen wir eine Art von Selektion einbauen.

Zuerst machen wir eine simple Funktion für Movement. Da ein komplexer Algorithmus für die Spurenwahl sehr aufwendig ist und nicht zum Projekt gehört machen wir einfaches Movement in eine zufällige Richtung. Wir nutzen also die Funktion rand(). In der passenden Formel erhalten wir somit eine Zahl zwischen 1 und -1 .

```
(float)rand() / RAND_MAX * 2 - 1
```

Da eine Selektion sehr schnell sehr chaotisch werden kann halten wir uns an zwei einfache Variablen: Size und Hunger. Size ist eine Variable, die von der Spezies vererbt wird und die Stärke als auch die Fortpflanzung Geschwindigkeit beeinflusst. Hunger ist für jede Kreatur einzigartig und beeinflusst ebenfalls die Stärke und bietet den Schwellenwert, ab welchem sich eine Kreatur fortpflanzen kann.

Hunger ist standardmässig auf 0.5 und ab 1.5 tritt Fortpflanzung ein, welche denn Wert wieder auf 0.5 heruntersetzt. Size ist eine Zufällige grösse zwischen 7.5 und 15. Hunger soll pro tick einen bestimmten Wert abnehmen aufgrund der Folgenden Parametern: Auf der falschen Höhe und falschen Temperatur. Die Temperatur und Höhe entnehmen wir aus dem Vektor und somit haben wir einzelne Attribute.

Nun brauchen wir noch eine Funktion um Hunger zu generieren. Hier kommen zwei Ideen. Interaktion zwischen Lebewesen und zufällig verteiltes Essen über die Map.

Zuerst implementieren wir eine Kampffunktion für die Selektion zwischen Lebewesen. Hierzu muss zuerst detektiert werden, wie weit zwei Lebewesen voneinander entfernt sind. Für den X Wert kann man das Vereinfacht mit dieser Formel herleiten.

```
abs(cre1.x - cre2.x) <= cre1.Size / 2 + cre2.Size / 2
```

x1 und x2 sind die Mittelpunkte der Kreaturen. abs() verwandelt negative zahlen in positive und lässt positive stehen.
Die Gleiche Formel kann man auch mit Y verwenden womit wir somit eine einfache Detektierung haben.

Nun brauchen wir eine Kampffunktion mit welcher zwei Kreaturen miteinander verglichen werden und die andere Kreatur somit den Hunger der anderen erhält. Wir benutzen wie bereits gesagt die Size und den Hunger Wert hier. wobei Size und Hunger einfache Multiplikatoren sind um die Kampfstärke zu erhalten.

Ebenfalls brauchen wir nun noch Essen auf der Map wir erstellen also die Klasse Vegetables und spawnen sie mit jedem tick auf die Map. Nun brauchen wir noch eine Funktion um diese mittels Kreaturen zu essen. Wir nutzen die vorher benutzte Formel nochmals und implementieren so Essen.

Damit das Essen ebenfalls nicht unendlich auf der Karte verweilt machen wir eine rot Funktion, in welcher mit jedem Zug ein Teil des Hunger Values entfernt wird. Am Ende despawned das Essen.

7.4 Rendern

7.4.1 Grid Map

Wie bereits erwähnt müssen die Daten in eine geeignete Darstellung umgewandelt werden. Da SFML sehr nahe an openGl ist müssen wir in diesem Prozess vieles selbst machen. Somit beginne ich erstmal die einzelnen Quadrate via Koordinate zu Zeichnen und benutze diese Funktion dann um ein Grid zu erstellen. Sobald wir damit fertig sind, haben wir eine Funktion um da ganze Grid zu erstellen und den Vektor auszulesen.

Um unseren Vektor mit zufälligen Werten zwischen 1 und 0 auszuwerten, müssen wir nun eine passende Konversion von double zu Color machen. Wir können das recht Simpel machen, indem wir jedem Wert zwischen 1 und 0 eine Color zuweisen. Die Color entnehmen wir von einer bereits existierenden Map mittels eines Color Pickers. Nun kann in den weiteren Schritten der Vektor ausgelesen und dargestellt werden.

Die daraus erstellte Map speichern wir in Word und verwenden sie jedes Mal wenn die Funktion drawWorld() ausgeführt wird.

7.4.2 DrawCreatures()

Für die Kreaturen haben wir bereits einen Vektor erstellt, welcher nun noch in eine Grafik umgewandelt werden muss. Hierzu benutzen wir die Funktion von vorher, um das Grid zu zeichnen. Man könnte auch Texturen oder ähnliches verwenden, dies ist jedoch in dieser Arbeit noch nicht beinhaltet.

Ebenfalls wird die Funktion noch im gleichen Geschehen verwendet, um Vegetables zu zeichnen.

8 Kontrolle

Da das Programm für den Nutzer kein weites Umfeld hat können die Testfälle sehr kurz gehalten werden. Es werden vor allem Interne Dinge gewertet welche nur durch beobachten getestet werden können.

8.1 Vorlage

Entwickelt von: Patrick Zürcher

Getestet von:

Getestete Version des Projektes:

Verwendete Version:

Betriebssystem: Windows 10

Test-Datum:

Eingabe	Ergebnis	Bestanden
Kartenverwaltung		
Drückt der Exe.	Programm startet.	❑
Drücken von X (Fenster schliessen).	Das Fenster schliesst.	❑

8.2 Test 1

Entwickelt von: Patrick Zürcher

Getestet von: Patrick

Getestete Version des Projektes: V1.0

Verwendete Version: V1.0

Betriebssystem: Windows 10

Test-Datum: 24.02.2022

Eingabe	Ergebnis	Bestanden
Kartenverwaltung		
Drückt der Exe.	Programm startet.	Ja
Drücken von X (Fenster schliessen).	Das Fenster schliesst.	Ja

9 Auswertung

9.1 Auftrag

Der Auftrag war grundsätzlich sehr weit gewählt und hat somit nicht die nötigsten Rahmenbedingungen angeboten um stark darauf auszubauen. Die Grundidee, war es herauszufinden wie man mit C++ programmiert und wie man eine Simulation gestalten. Beides sind sehr weite Gebiete und vor allem auch unterschiedlich, wie tief man in die Themenkomplexe eingehen kann. Am Ende wäre es wahrscheinlich besser ein einfaches Spektrum zu wählen.

9.2 Umsetzung

Die Umsetzung ging sehr gut voran jedoch weis man nie wie weit man in die Materie eingehen soll und man sollte sich erstmal an der Oberfläche aufhalten und auf ein einzelnes Thema fokussieren.

9.3 Fazit

Im Endeffekt war das Projekt erfolgreich, zumindest in dem, was selbst erlernt wurde. Ich konnte mich auf viele Ideen einlassen und C++ kennenlernen.

10 Quellenverzeichnis

Tutorial für Rauschfunktionen: https://www.redblobgames.com/maps/terrain-from-noise/

Github für Rauschfunktion: https://github.com/deerel/OpenSimplexNoise

Visual Studio SFML Install: https://www.sfml-dev.org/tutorials/2.5/start-vc.php.

11 Anhang

11.1 Github

https://github.com/Plantoloz/GitGameOfEvolution