default title

default author
facculty

00/00/0000

---

1 Konzept

-GOGOL (kurz für Game of Game of Life) ist eine Desktop Applikation für das berühmte nullspieler-spiel Game of Life, welches vom britischen mathematiker John Horton Conway im jahr 1970 entwickelt wurde. - das grundkonzept des spiels besteht aus einem zweidimensionalen, rechteckigen gitter, wobei jedes feld in diesem gitter einer zelle entspricht - eine zelle kann pro generationsschritt einen von zwei zuständen haben: lebend, tot - der zustand einer zelle ist abhängig von den 8 nachbarn die, die zelle umgeben - bei einem generationsübergang wird nun der zustand einer zelle bestimmt: - Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren. -Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an isolation. - Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration am Leben. - Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.

Aus diesen einfach regeln enstehen verblüffende strukturen, welche einzigartige verhaltensmuster, bishin zur turing-completeness aufweisen.

Die Grundidee war das entwickeln einer soliden desktop anwendung die weitere nützliche bedienfunktionen hat. Darüber hinaus soll das programm, neben dem standart spiel noch weitere modi zu implementieren, welche eine abwandlung des "vanilla"GoL bieten. Zu anfangs geplant waren: standard conway colormerge - das verschmelzen von zellen mit farbeigenschaften colorwar

- kampf von zellen mit "teamfarben"probability of life - random warscheinlichkeit bei der geburt von zellen

zuletzt sollte noch ein lokaler player versus player modus implementiert werden, welcher der anwendung seinen eigentlichen namen verleiht, da so auf dem game of life ein tatsächliches spiel ensteht

Sinn der applikation besteht darin, aus dem game of life einen unterhaltungswert zu gewinnen und somit eine beschäftigung für zwischendurch zu schaffen

2 Planung

2.1 meilensteine

idee war es die jeweiligen features in meilensteine zu unterteilen sobald eine primitive version des spiel besteht, die meilensteine wurden dann 1:1 auf sprints aufgeteilt die alle 1 oder 2 wochen länge hatten, je nach vermutung der workload, so enstanden zu nächst folgende meilensteine und sprints: -grundimplementation: 1. primitive laufende version -automatisierung: qol-features wie: play/stop, speedregler, load/save, preloaden von strukturen sog. species -rainbow: custom rules, colormerge, colorwar -bio: probability of life -pvp: exterminate, populate -dokumentation -website: deployment einer einfachen webseite zum hosten der anwendung als web-applet oder bereitstellen zum download

wir haben gfrundfunktionalitäten priorisiert da höhere funktionalitäten suksessiv auf niedrigere features aufbauen, höhere funktionen wäre ohne vorherige implementation der vorherigen grundfunktionen nicht lauffähig

2.2 zeitplan

die sprints haben zueinander einen critical path, jedoch war theoretisch die abgabe schon nach dem abschließen der automatisierung möglich für die abgabe war der plan, eine woche vor deadline fertig zu sein um bei auftretenden problemen einen puffer zu haben

LINK OLD GANT:PNG der erste entwurf des gant diagrams

wie hat sich die planung im laufe des projekt verändert? aufgrund von fehlerhaften schätzungen wurden subfeatures und einige meilensteine komplett entfernt oder verschoben

demnach veränderte sich der projektverlauf siehe auch: was wurde verworfen

LINK NEW GANTT letzte aktuelle version der planung

2.3 testplan

tests wurden dem jeweiligen sprint zugeteilt und nach fertigstellung der features implementiert mit ausnahme von rainbow, dort wurde ein test first ansatz verwendet

2.4 technische beschreibung des systems

die systemarchitektur liegt dem MVC (Model, View, Controller)-Ansatz zugrunde: Der View-teil besteht aus einem GUI, sowie einem Gamegrid, welches an das GUI übergeben wird und dieses darstellt. Das GUI hält das gamegrid lediglich als container und führt keine sondierenden methoden auf dem gamegrid aus. Als Model-Teil wurden zunächst die Zellen implementiert, diese halten keine logik sondern geben nur ihren status nach außen oder kriegen ihre eigenschaften von außen gesetzt. Die Zellen werden dann in einem 2 dimensionalen array: der survivalmatrix, gesetzt. Desweiteren gibt es ein Regelwerk hier: Ruler, dieser hält alle methoden der spielregeln und gibt das cellverhalten vor, welche vom controller dann verwendet werden. Für den PvP modus ist ein Referee zuständig der die 2spielerreglen verwaltet. Für das preloaden der species wurde ein species model entwickelt, welches die eigenschaften für die SVM hält und in einer specieslibrary verwaltet wird. der Preloader wendet diese auf die SVM an. Hauptkomponente des Controller-teils stellt der Controller dar, dieser hält alle logikteile, die wiederum doe modelle halten. er hält außerdem als einziger die SVM. des weiteren kriegt er bei seiner erstelluing das gamegrid und das gui überreicht. daher ist er für die verwaltung zuständig, er stellt verbindung zwischen logik und gui her in dem er alle listener erstellt und an die buttons bindet.

LINK CLASS DIAGRAM

als kapselung wurden die klassen in mehrere packages unterteilt und somit ihrer funktionalität unterteile: frontend: SLiderUI EndgameDialog Gamegrid LifeGui PaintImage

backend: command controller player referee ruler saver

cells: cell coloredcell conwaycell pvpcell

library: preloader species specieslibrary

listener: buttonlistener celltogglelistener gamemodelistener preloadlistener speedchangerlistener

LINK PACKAGE DIAGRAMM

3 stand des projekts

3.1 Was ist fertig?

Das klassische Game of Life nach Conway ist vollständig, sowie die alternativen Spielmodi ColorMerge, ColorWar und PvP. Für jeden dieser Modi sind die Funktionalitäten Speichern/Laden des aktuellen Spielstandes, laden bestimmter Presets, sowie (ausgenommen für den Modus PvP) eine zufällige Initialisierung des Spielfeldes.

3.2 Verworfen

Der geplante Modus Propability of Life wurde verworfen. Dieser hätte beinhaltet, dass Zellen abhängig von ihrer Anzahl Nachbarn eine bestimmte Wahrscheinlichkeit haben in der nächsten Generation lebend oder tot zu sein. Das Zellverhalten wäre somit jedoch nicht mehr deterministisch. Eine Nutzung der Presets oder das eigene finden bzw. erstellen solcher wäre in diesem Modus somit unmöglich, womit der entscheidende Aspekt des Game of Life verloren gegangen wäre. Auch wenn die Idee interessant ist, hat sie keinen Bezug zu dem Game of Life und den restlichen Spielmodi.
Der geplante Modus PvP - Exterminate wurde verworfen. In diesem PvP Modus wäre is das Ziel gewesen möglichst viele Zellen des Gegners zu vernichten. Schon die Definition wann eine Zelle vernichtet oder einfach nur abgestorben ist gestaltet sich als schwierig und wäre für die Spieler schwer verständlich und während des Spiels nicht in realistischer Zeit nachvollziehbar.
Das geplante Feature Custom Ruleset wurde verworfen. Dieses hätte dem Spieler ermöglicht die Bedingungen, bei wievielen Nachbarn eine Zelle lebend oder tot sein wird anzupassen. Dadurch werden jedoch Presets nicht mehr nutzbar, da diese auf dem Verhalten nach den Conway regeln beruhen, womit ein wichtiges Feature in diesem Modus vom User nicht erwartete Ergebnisse erzeugt hätte.
Das Feature Change Gridsize ist momentan nicht nutzbar. Die Funktionalität ist bereits implementiert, jedoch ist kein Button auf der Öberfläche imple-

mentiert. Grund hierfür ist mangelnder Platz und nicht ausreichend Zeit um die GUI zu refactorn und welchen zu schaffen. In zukünftigen Patches wird dieses Feature vermutlich aufgenommen.
Das geplante großflächige Bereitstellen von Presets ist momentan noch nicht nutzbar. Geplant war aus http://conwaylife.appspot.com/library die bekannten Presets per Webscraping auszulesen. Dies ist erfolgreich implementiert, redoch wird der code wir einzelne Presets nicht korrekt interpretiert, wodurch diese falsch dargestellt werden. Des Weiteren gestaltet es sich schwer die große Menge an Presets für den User übersichtlich darzustellen. In zukünftigen Patches wird dieses Feature vermutlich aufgenommen.
Überlegungen zur Verbesserung der Effizienz bei der Zustandsberechnung der Zellen wurden noch nicht implementiert. Grund hierfür sind mangelnde Zeit und bereits ausreichend gute Performance der Anwendung. In zukünftigen Patches wird die Effizienz vermutlich verbessert.

3.3 Bekannte Bugs und Probleme

Bisher sind keine Bugs bekannt.
Ein bekanntes Problem ist, dass der SpeedSlider eine Exponentielle Funktion für die Geschwindigkeit verwendet. Diese liefert zwar die gewünschte Funktionalität, ist jedoch weniger intuitiv als eine Lineare Funktion. Aufgrund geringer Priorität wird diese Anpassung in zukünftigen Patches durchgeführt.

3.4 Workload

Die Workload des Meilensteins "Rainbow"wurde deutlich unterschätzt. Grund hierfür waren unterschiedliche Interpretationen der Methodendefinition. Hier raus resultierten Fehler im Verständnis der vom jeweils anderen geschriebenen Methoden und falsche Anwendung dieser. Anstatt einer Woche waren hier zwei Wochen nötig.
Die Workload des Meilensteins "PvP"wurde überschätzt. In diesem Meilenstein konnte sein sehr großer Teil der Funktionalität auf dem Modus ColorWar aus dem vorherigem Meilenstein "Rainbowäufgebaut werden. Anstelle der geplanten zwei Wochen wurde weniger als eine Woche benötigt.

3.5 Arbeitsaufteilung

Die folgenden Aufgaben wurden von Kolja Hopfmann und Jonas Sander gemeinsam durchgeführt:
Projektplanung, Implementierung der Klassen Controller und Referee
Die folgenden Aufgaben wurden von Kolja Hopfmann übernommen:
Projektmanagement, Implementierung der Packages Frontend und Listener, Implementierung der Klasse Player und des Commandhandling im Backend
Die folgenden Aufgaben wurden von Jonas Sander übernommen:
Implementierung der Packages Testing, Cells und Library, sowie die Implementierung der Klassen Saver und Ruler im Backend.

- readme

- verwendete Programme - Eclipse Neon - IntelliJ IDEA Ultimate 2017.1.2 - gimp - texmaker - GanttProject 2.7.1 - Funktionsweise Java.AWT.Graphics war komisch - Es wurde kein Framework verwendet - alle relevanten Klassen und Methoden wurden selber entwickelt und implementiert - wir haben uns die Definition von Game of Life angelesen - Implementierung, Datentypen und Darstellung wurden selber erarbeitet