default title

default author
facculty

00/00/0000

Inhaltsverzeichnis

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 entstehen verblüffende Strukturen, welche einzigartige Verhaltensmuster, bis hin 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 Standard Spiel noch weitere Modi zu implementieren, welche eine Abwandlung des "Vanilla"GoL bieten. Zu Anfang 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
- PvP - lokaler Player versus Player Modus, welcher der Anwendung ihren eigentlichen Namen verleiht, da so aus dem Game of Life ein tatsächliches spiel für 2 Spieler entsteht

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, beginnend mit einer primitiven Grundversion des Game of Life. Die Meilensteine wurden dann 1:1 auf Sprints mit jeweils 1 oder 2 Wochen Länge aufgeteilt, je nach geschätzter workload. Hier raus ergaben sich zunächst die folgenden Meilensteine und Sprints:

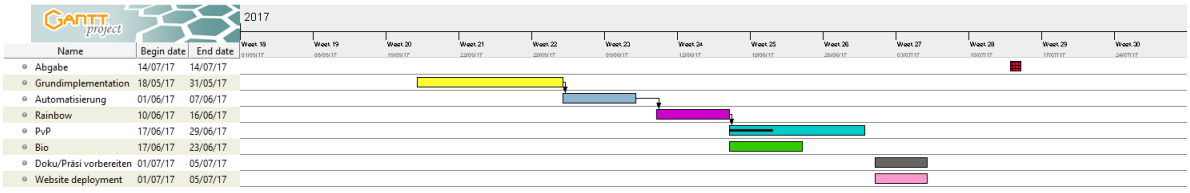
- grundimplementation: 1. primitive laufende version
- automatisierung: qol- features wie: play/stop, speedregler, load/save, preloaden von strukturen sog. species
- rainbow: custom rules, colormerge, colorwar
- bio: probability of life
- pvp: exterminate, populate
- dokumentation
- website: deployment einer einfachen webseite zum hosten der anwendung als web-applet oder bereitstellen zum download

Wir haben Grundfunktionalitäten priorisiert, da höhere Funktionalitäten sukzessiv auf niedrigere Features aufbauen, höhere Funktionen wäre ohne vorherige Implementation der nötigen Grundfunktionen nicht lauffähig.

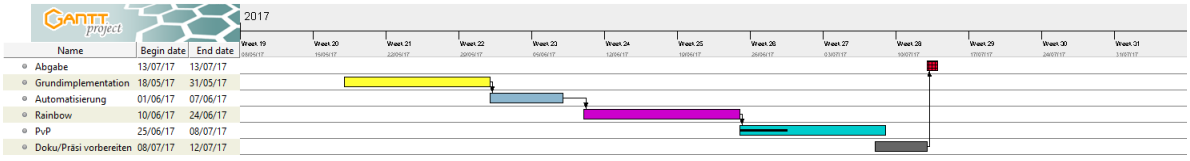
2.2 zeitplan

Die Sprints haben zu einander 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, zu sehen im ursprünglichem

Entwurf des gantt Diagramms.



Aufgrund von fehlerhaften Schätzungen wurden Subfeatures und einige Meilensteine komplett entfernt oder verschoben, wodurch sich auch der Projektverlauf veränderte (siehe auch: was wurde verworfen). Somit ergab sich letztendlich die Aktuelle Version der Planung.



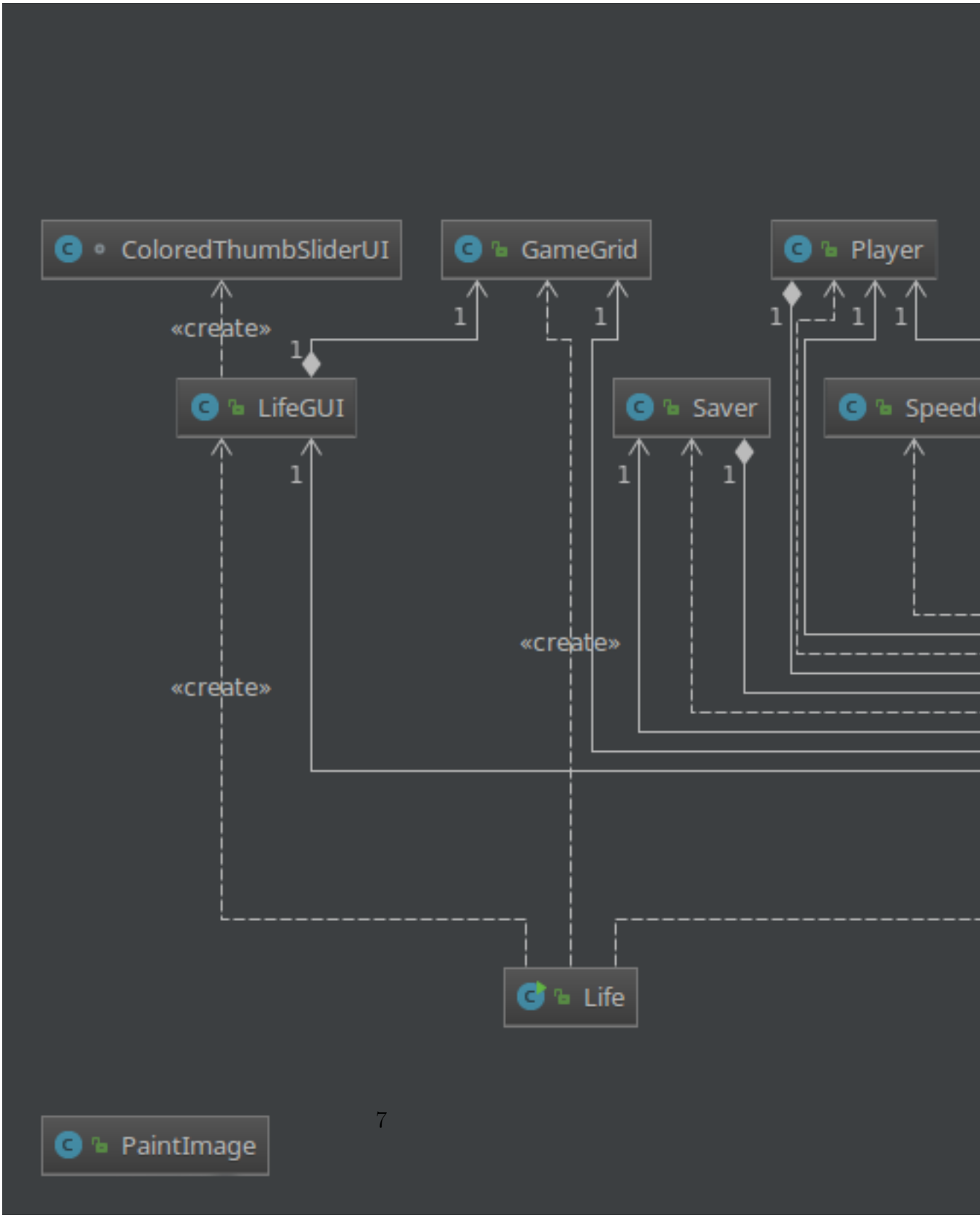
2.3 testplan

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

Der Systemarchitektur liegt der 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 speciesslibrary 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.


ColoredThumbSliderUI
GameGrid
Player
«create»
1
1
1
1
1
1
LifeGUI
Saver
Speed
1
1
1
«create»
«create»
Life
PaintImage

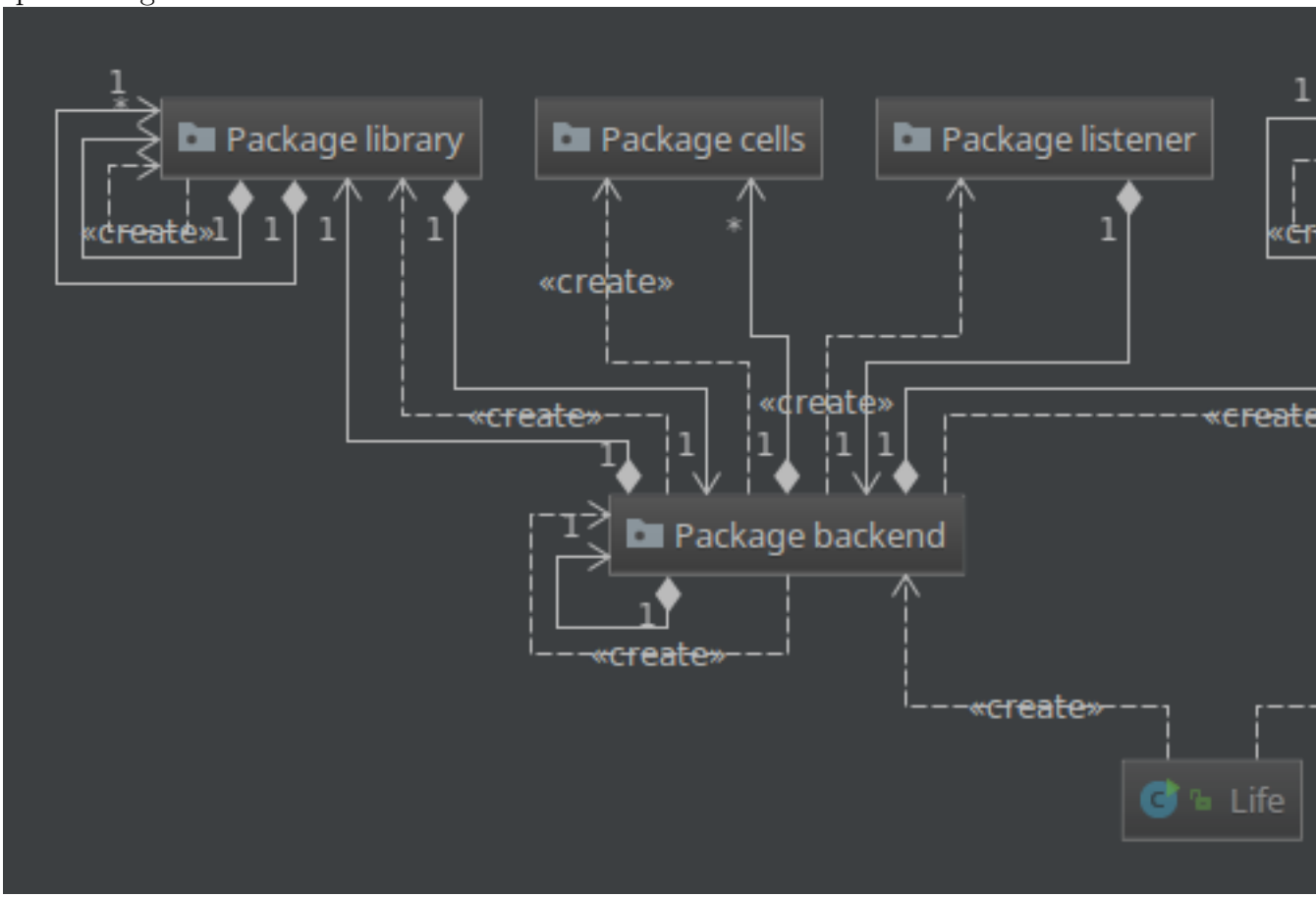
als kapselung wurden die klassen in mehrere packages unterteilt und somit ihrer funktionalität unterteile: frontend: SLiderUI EndgameDialog Gamegrid LifeGui PaintImage

backend: command controller player referee ruler saver

cells: cell coloredcell conwaycell pvpcell

library: preloader species specieslibrary

listener: buttonlistener celltogglelistener gamemodelistener preloadlistener speedchangerlistener



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 implementiert. 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.

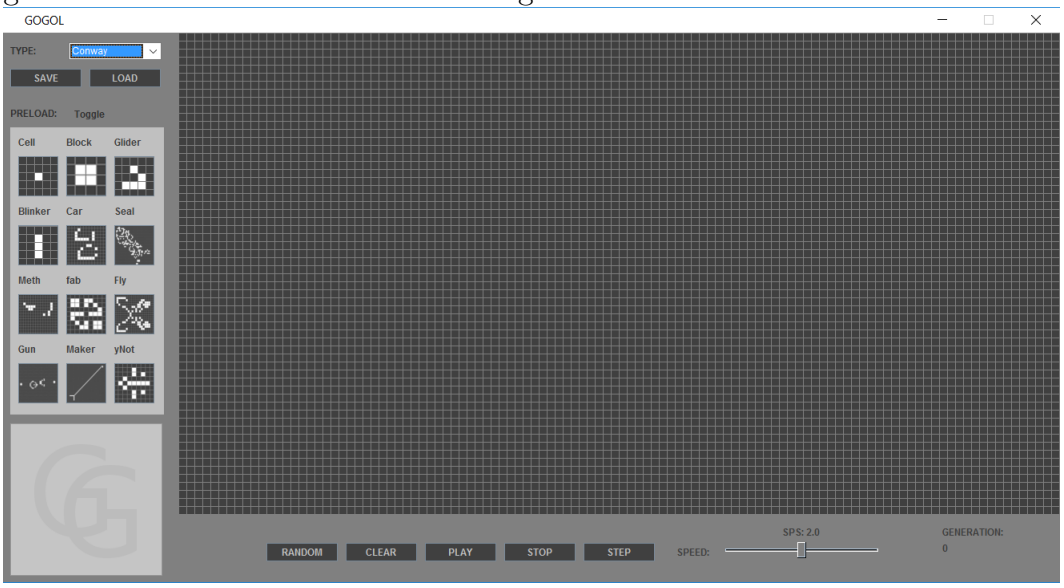
4 Funktionsbeschreibung

4.1 Installation und Systemvorraussetzungen

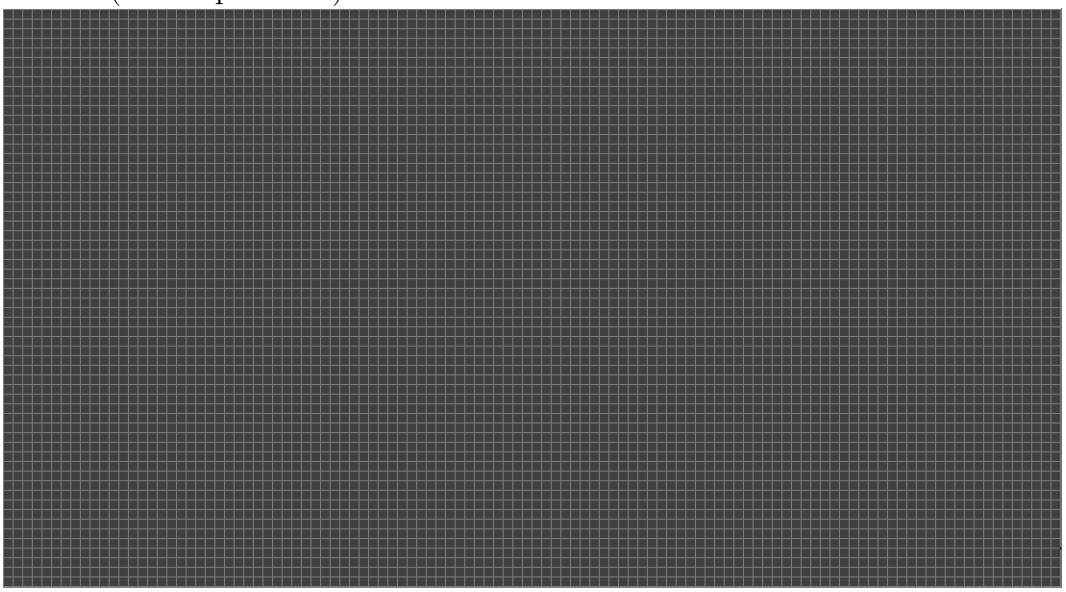
be stuff here

4.2 Bedienungsanleitung

Die Anwendung wird durch Ausführen der EXE (Windows) bzw. JAR (Linux) gestartet. Zu sehen ist das Anwendungsfenster.

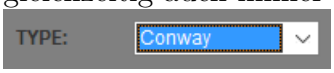


Der größte teil ist das Spielfeld. Jedes Quadrat auf dem Spielfeld ist eine Zelle. Zellen können durch anklicken mit dem linken Mausknopf verändert werden (siehe Spielmodi).



Über das Dropdown Menü "TYPE"links oben kann der Spielmodus gewechselt

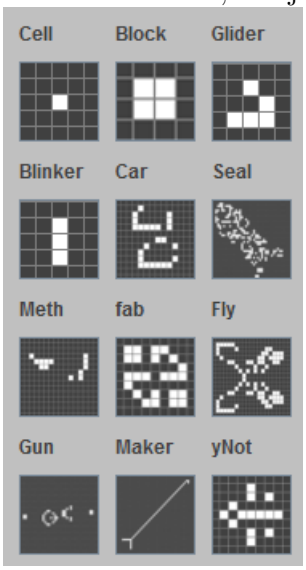
werden (siehe Spielmodi). Beim Wechsel des Spielmodus wird das Spielfeld gleichzeitig auch immer geleert.



Mit dem Knopf SSAVE"kann der aktuelle Spielstand gespeichert werden. Über den Knopf "LOAD"kann ein früherer Spielstand geladen werden. Hierbei wird automatisch auf den entsprechenden Spielmodus gewechselt und ggf. die Spielfeldgröße angepasst.

Das feld "PRELOAD: "gibt an auf welche Art Zellen momentan gesetzt werden. Toggle steht hierbei das Setzen einer einzelnen Zelle. Ist ein Preset ausgewählt, wird dessen Name angezeigt.

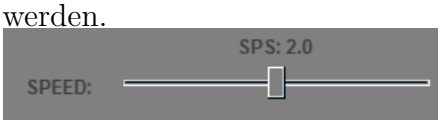
Auf den verschiedenen Preset-Knöpfen ist die Zellbelegung des Jeweiligen Presets angezeigt, sowie dessen Name. Durch klicken eines Preset-Knopfes wird dieses Preset ausgewählt. Wird nun eine Zelle auf dem Spielfeld geklickt, wird dort das entsprechende Preset automatisch geladen. Presets sind besondere Strukturen, die jeweils ein bestimmtes Verhalten haben.



Über die Kontrollknöpfe kann das Spiel gesteuert werden. "RANDOM"belegt alle Zellen des Spielfeldes mit zufälligen Werten (siehe Spielmodi). "CLEAR"leert das Spielfeld. SSTEP"bringt das Spiel um eine Generation voran. "PLAY"lässt das Spiel automatisch eine bestimmte Anzahl Schritte pro Se-

kunde voranschreiten. SSTOP"hält ein automatisch laufendes Spiel an. Im Spielmodus "PvPïst das Verhalten der Knöpfe leicht verändert (siehe Spielmodi).

SPS gibt die Anzahl Schritte an, die pro Sekunde bei einem Automatisch laufendem Spiel durchgeführt werden. Über den Slider kann die SPS angepasst werden.



Generation gibt die Aktuelle Zellgeneration des Spiels an. Bei jedem leeren des Spielfeldes, einer Zufallsbelegung und beim Wechseln des Spielmodus wird die Generation auf 0 zurückgesetzt.

4.3 Spielmodi

4.3.1 Conway

Dieser Spielmodus bietet das originale Game of Life nach Conway mit einigen Features zur vereinfachten Benutzung. Conways Game of Life ist ein Null-Spieler-Spiel. Der Spieler kann hier das Spielfeld nur bei Beginn einmalig modifizieren und danach das Zellverhalten beobachten. In unserer Version von Game of Life ermöglichen wir auch den späteren eingriff in das Spiel nach jeder Generation. Jede Zelle ist entweder tot oder lebendig. In jeder Generation wird für jede Zelle neu berechnet ob sie tot oder lebendig ist.

- 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.

Ein klick auf eine Zelle änderte den Status der Zelle zwischen lebendig und tot. Tote Zellen sind dunkel grau, lebendige Zellen sind weiß.

4.3.2 ColorMerge

In diesem Spielmodus erhält jede Zelle zusätzlich eine Farbe. Ob eine Zelle lebendig oder tot ist wird nach den selben Regeln wie im Spielmodus "Conway"bestimmt. Die Farbe einer neu geborenen Zelle berechnet sich durch den Mittelwert der Farben ihrer Nachbarzellen. Durch klick auf eine Zelle wird ihr Status wie Folgt geändert:
tot => lebendig, rot => lebendig, grün => lebendig, blau => tot => ...
Ein klick auf eine Zelle mit einer gemischten Farbe ändert diese zu lebendig, rot. Der Knopf "RANDOM"weist jeder Zelle zufällig einen Status lebendig oder tot zu, sowie jeder lebendigen Zelle eine Zufällige Farbe Rot, Grün oder Blau.

4.3.3 ColorWar

In diesem Spielmodus erhält wie in "ColorMerge"jede Zelle eine Farbe. Ob eine Zelle tot oder lebendig ist, wird weiterhin nach den selben Regeln wie im Spielmodus "Conway"bestimmt, jedoch mit einem Zusatz für die Geburt neuer Zellen. Wird eine Zelle neu geboren, muss es unter den Nachbarn eine Farbe mit eindeutiger Zellmehrheit geben, damit eine Zelle geboren werden kann. Die Farbe der neu geborenen Zelle ist immer identisch mit der mehrheitlichen Farbe der Nachbarzellen.
Durch klick auf eine Zelle wird ihr Status wie Folgt geändert:
tot => lebendig, rot => lebendig, grün => lebendig, blau => tot => ...
Der Knopf "RANDOM"weist jeder Zelle zufällig einen Status lebendig oder tot zu, sowie jeder lebendigen Zelle eine Zufällige Farbe Rot, Grün oder Blau. Werden in geladene Presets Zellen mit anderer Farbe gesetzt, verliert die Struktur möglicherweise ihre Funktionalität.

4.3.4 PvP

Dieser Spielmodus konvertiert Conways Game of Life in ein tatsächliches 2-Spieler-Spiel und ist Namensgeber für die Applikation GOGOL (Game of Game of Life). "PvP"beruht auf dem Spielmodus "ColorWar". Hier gibt es nun jedoch einen roten Spieler und einen blauen Spieler. Jeder Spieler hat einen in seiner Farbe markierten Bereich. Jeder Spieler kann Zellen nur innerhalb

seines Bereiches und in seiner eigenen Farbe setzen.Das Laden von Presets ist möglich, jedoch werden diese abgeschnitten, wenn sie über den Bereich des Spielers hinaus gehen. Jeder Spieler verfügt bei Spielstart über eine bestimmte Anzahl Zellen. Nach einem Spielzyklus erhält jeder Spieler neue Zellen, die er vor beginn des nächsten Spielzyklus setzen darf. Der Spieler, der bei Spielende mehr lebendige Zellen in seiner Farbe auf dem Spielfeld hat gewinnt.

- Zellen bei Spielstart: 100 Zellen
- Neue Zellen pro Spielzyklus: 50 Zellen
- Dauer eines Spielzyklus: 100 Generationen
- Dauer eines gesamten Spiels: 1000 Generationen

Ungenutzte Zellen können in den nächsten Spielzyklus mitgenommen werden. Zellen einer Farbe können sich aus dem Bereich des Spielers heraus und in den des Gegenspielers hinein verbreiten. Lediglich das Setzen neuer Zellen ist auf den Spielerbereich beschränkt. Jedes setzen einer Zelle, egal ob von tot zu lebendig oder von lebendig zu tot, kostet den Spieler eine seine verfügbaren Zellen. Das setzen von Presets kostet den Spieler so viele Zellen, wie das Preset lebendige Zellen beinhaltet. Hat ein Spieler nicht genug verfügbare Zellen für ein Preset wird dieses nur zum Teil geladen.

5 Sonstiges

5.1 Verwendete Programme

- IDEs:
  - Eclipse Neon
  - IntelliJ IDEA Ultimate 2017.1.2
- Gimp 2.0
  - Design der GUI-Komponenten
- Texmaker
  - Erstellung der Dokumentation

- GanttProject 2.7.1
  - PM: Erstellung des Gantt-Diagramms

5.2 VCS

Entgegen der Empfehlung für Subversion haben wir uns für die Nutzung von Git in Verbindung mit Github als Open-Source Prinzip entschieden. Gründe hierfür war die Priorität eines dezentralen Cloud-Systems, welches jederzeit über jedes Endgerät erreichbar ist. Mit Git war es uns Möglich mehrere Lokale Versionen und eine Remote-Version zu haben. Da wir mit mehreren Geräten gearbeitet haben war dies ein enormer Vorteil, da wir im Falle eines Merge-Errors oder einer korrupten Remote-Repository eine der verfügbaren Local-Repositories pushen und auf einen früheren Commit backstagen konnten. Desweiteren entschieden wir uns für Git da alle Projektmitglieder fundierte Erfahrung mit Git hatten.

5.3 Statistiken

- Anzahl Java-Klassen: 35
- Java-Codezeilen: 3861
- Commits in der Remote-Repository: 229
- Anzahl Branches: 3

6 Reflektion

6.1 Was lief gut?

Zwar hat die anfängliche Planung Zeit gekostet, jedoch hat diese im weiteren Projektverlauf erhebliche Vorteile eingebracht, da einem stets bewusst war in welchem Abschnitt des Projekts man sich befand und wie gut man im Zeitplan lag. Genaue Definition der Systemarchitektur machte es einfach neue Komponenten hinzuzufügen und diese in den Programmkontext sauber einzubinden. Tägliche SScrum-meetings"haben zusätzlich geholfen einen genauen Überblick über die Arbeitsverteilung zu haben. Desweiteren wurden keine

sekundären Frameworks verwendet. Alle relevanten Klassen und Methoden wurden selber entwickelt und implementiert

6.2 Was lief suboptimal?

Zu Anfangs gab es einige Probleme mit der Verwendung des java.awt.Graphics-Framework, da genaue Funktionsweisen nicht ersichtlich waren. Gelöst wurde das Problem indem wir das Konzept der paintComponents-Methode aus dem DrawProject auf unser Projekt übertrugen.

6.3 Was musste erarbeitet werden?

Das Arbeiten mit ImageIcons in bezug mit lokalen Classpaths führte anfangs zu unerwarteten Ergebnissen, da die jeweilige IDE korrekt kompilierte, dies jedoch als eigenständiger Prozess zu fehlern führte, da sich die Classpaths zur eigenständigen Laufzeit änderten.

6.4 Was haben wir gelernt?

Projektmanagement in der Theorie macht wenig sinn, da die Notwendigkeit nicht greifbar ist. Die Gestaltung dieses Projekts gab uns die Möglichkeit nachvollziehbares Verständnis über Projektplanung zu sammeln und die Vorteile dieser zu erkennen. Für die Projektmitglieder war es zudem, das erste fertiggestellte programmiertechnische Projekt, was dazu führte das sehr viel Programmiererfahrung gesammelt wurde.

6.5 Fazit

Schlussendlich ist zu sagen, dass es sehr viel Spaß gemacht hat, ein eigenes vollständiges Produkt zu entwickeln. Dadurch stieg bei uns die Motivation weitere eigene Projekte zu starten, da wir nun die Erfahrung und Zuversicht für einen erfolgreichen Projektabschluss besitzen.