Habib Mubarak (6627597), Eraycan Gökcan (7000210), Basel Ali (7197202)

# Aufgabenblatt 2

### Aufgabe 1.1: Dokumentation für die Implementierung

Die objektorientierte Implementierung von Conways Game of Life, wird aus zwei Klassen bestehen:

1. **World-Klasse**

   Diese Klasse ist für die Logik des zellulären Automaten (bzw. Gitters) und dessen Aktualisierung verantwortlich. Sie beinhaltet:

   *Attribute:*
   - Die Höhe und Breite des Gitters
   - Die aktuelle Generation
   - Das Gitter selbst, implementiert als geschachtelter vector<vector<<int>>
   - Das Gitter wird sich toroidal verhalten. Dies wird durch Modulo-Operationen sichergestellt.

   *Methoden:*
   - Hilfsmethoden wie count_alive_neighbours(), is_equal()
   - Die Methoden gefragt in Exercise 1.2 wie evolve(), is_stable(), get-/set-Methoden, etc.
   - Methoden zum Hinzufügen von bestimmten Mustern (z.B. Glider)
   - Zu is_stable(): Da der Oszillationscheck auf eine Periode von 2 limitiert wurde, reicht es, wenn man für jede Generation folgendes überprüft: Ist der Zustand des Gitters der aktuellen Generation gleich der nächsten oder übernächsten Generation? Wenn ja, dann ist man in einem stabilen Zustand.

2. **cli-Klasse**

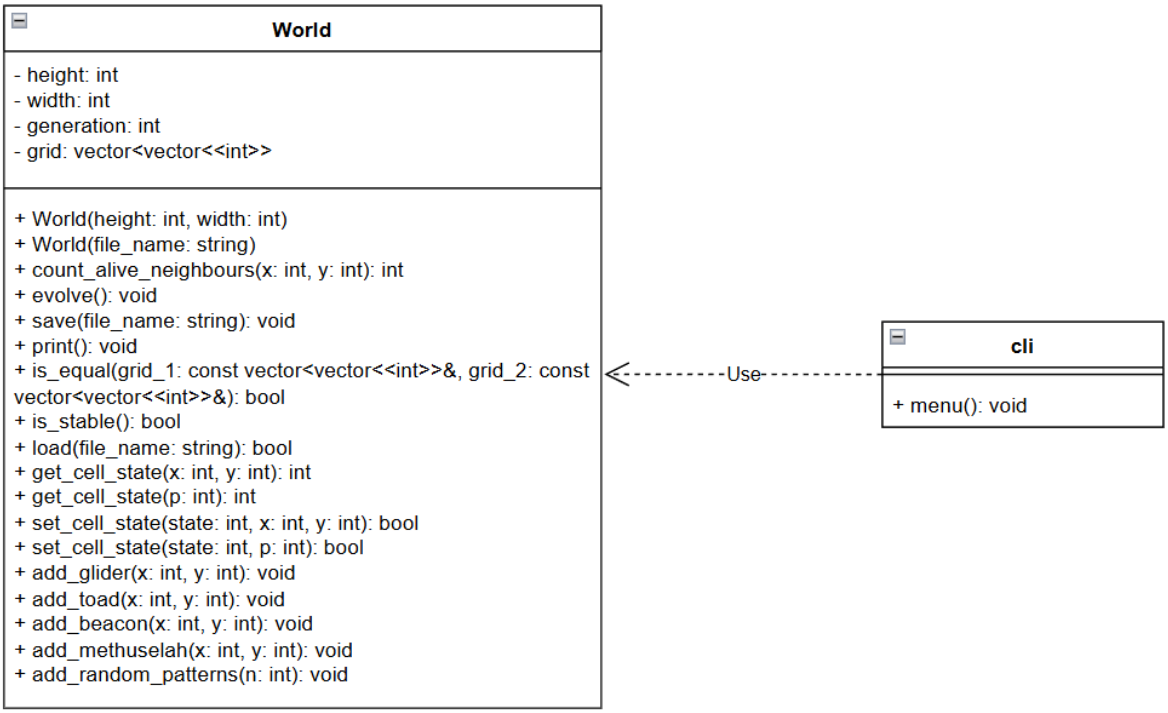
   Diese Klasse ist für den User Interface bzw. Command Line Interface verantwortlich. Sie ist für die User-Interaktion über ein textbasiertes Menü zuständig. Die Hauptmethode menu() bietet folgende Optionen:

   *Hauptmenü:*
   - Neue World erstellen (mit manueller Eingabe von Höhe und Breite)
   - World aus einer Textdatei laden

   *World-Management:*
   - Muster hinzufügen (Glider, Toad, zufällige Muster, etc.)
   - Simulation starten (für n viele Generationen oder kontinuierlich)
   - Zellen abfragen/ändern (get/set)

- Anzeige des Gitters ein-/ausschalten,
- Verzögerung (delay) anpassen
- Stabilitätscheck aktivieren/deaktivieren
- Aktuelle World als Textdatei speichern

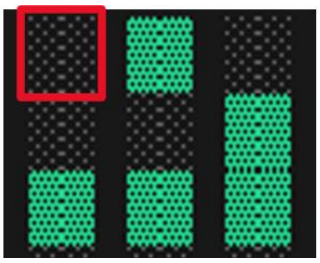
## Klassendiagram



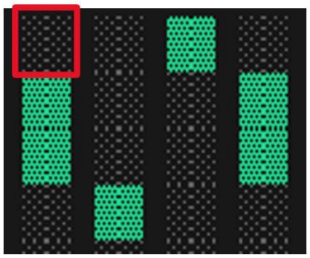

## Aufgabe 1.2 – Hinweis zum Code

Führe die Datei „build/GameOfLife“ aus oder direkt „GameOfLife“ im Hauptordner. CMakeLists.txt im Ordner „configurations“.

**Aufgabe 1.3 - Startzelle für alle Muster**



*Glider*



*Toad*

*Methuselah*

*Beacon*

**Aufgabe 1.4 – Laufzeitmessungen**

<u>Im DEBUG Modus:</u>
O0 → 2060ms
O1 → 573ms
O2 → 559ms
O3 → 261ms

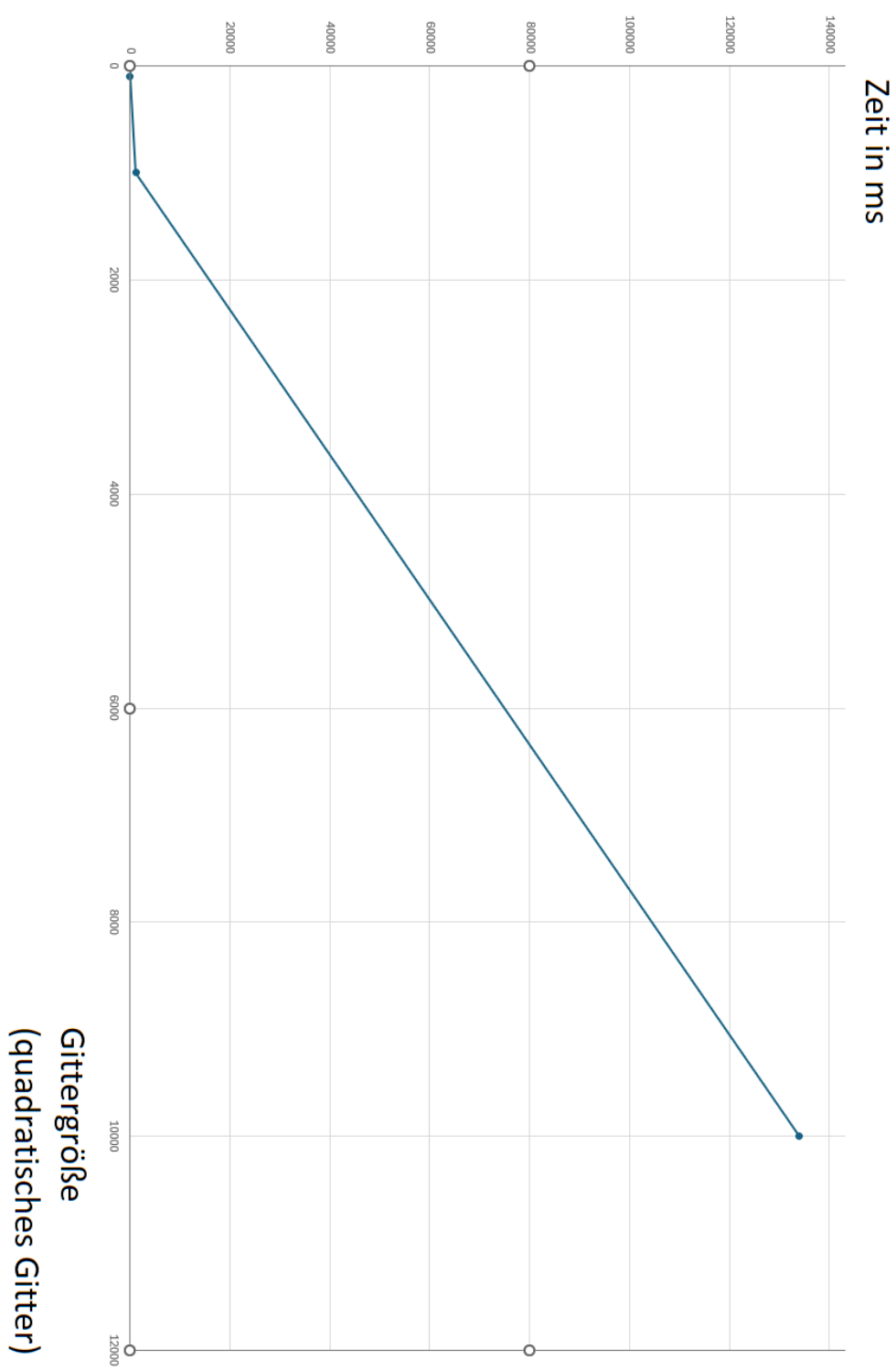
- Je höher das Optimierungslevel, desto kürzer ist die Laufzeit.

<u>Im RELEASE Modus:</u>
O0 → 2068ms
O3 → 2072ms

- Höheres Optimierungslevel scheint für keine weitere Optimierung oder Beschleunigung zu sorgen. Keine Erklärung dafür gefunden.

**Aufgabe 1.5 – Laufzeitmessungen**

Da p67_snark_loop.txt eine feste Gittergröße hat (100x100) und nicht skaliert werden kann, haben wir stattdessen ein Glider in der Koordinate (0,0) auf unterschiedlich großen Gittern (10x10 bis 10.000x10.000) platziert, um die Skalierbarkeit zu testen. Jedoch sind die Ergebnisse nicht direkt mit p67_snark_loop.txt vergleichbar, da das Snark-Loop-Muster viel komplexer ist. Hier sind die Laufzeiten für eine Simulation von 100 Generationen jeweils:




10x10: 174 Mikrosekunden
20x20: 533 Mikrosekunden
100x100: 16 Millisekunden
1000x1000: 1186 Millisekunden
10000x10000: 134072 Millisekunden