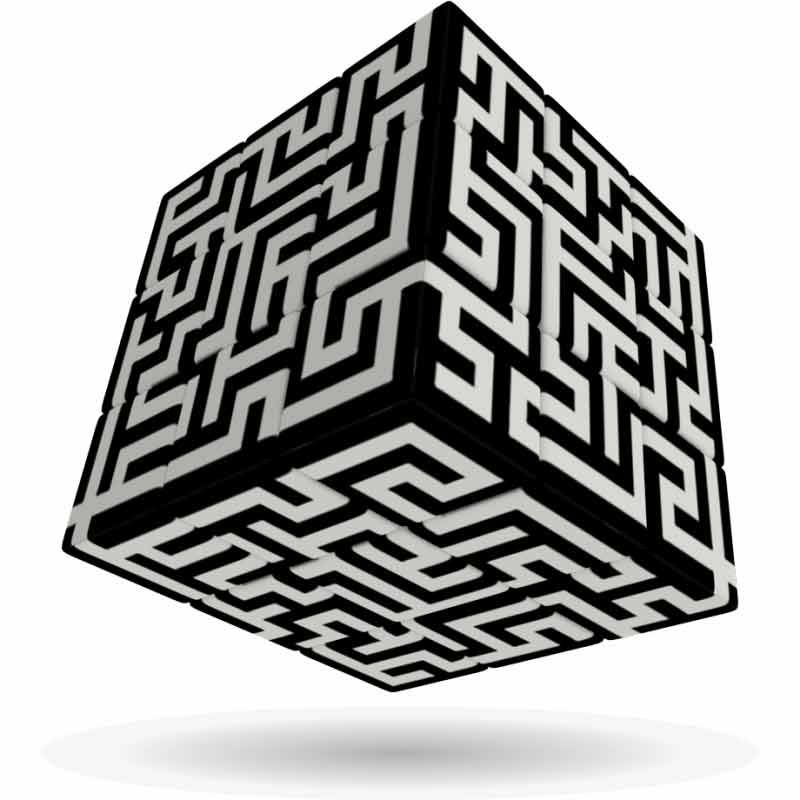
|  |
| --- |
| M226B und M120 Kombinationsprojekt |
| Mazerator |
| Ein interaktiver und visueller  Labyrinth-Generator und Löser |

|  |
| --- |
| Donato Wolfisberg, Jonas Koller  15. Juni 2018 |



Inhalt

[Einleitung 3](#_Toc516815286)

[Aufgabe 4](#_Toc516815287)

[Projektidee 4](#_Toc516815288)

[Anforderung 4](#_Toc516815289)

[Zeitplanung 5](#_Toc516815290)

[Paketzuteilung 5](#_Toc516815291)

[Projektplan 6](#_Toc516815292)

[Pattern 6](#_Toc516815293)

[Algorithmen 7](#_Toc516815294)

[Recursive backtracker 7](#_Toc516815295)

[Dijkstra-Algorithmus 7](#_Toc516815296)

[Klassendiagramm 8](#_Toc516815297)

[Mockup 8](#_Toc516815298)

[Anwendung 9](#_Toc516815299)

[Known Issues 10](#_Toc516815300)

[Programmcode 10](#_Toc516815301)

[Testprotokoll 10](#_Toc516815302)

[Rückblick 11](#_Toc516815303)

# Einleitung

In diesem Dokument wird unser Projekt «Mazerator» dokumentiert. Dieses ist in der Kombinationsarbeit der Module 226B und 120 entstanden. Bei dieser praktischen Arbeit haben wir die zuvor gelernte Theorie über GUI-Ergonomie und GUI-Programmierung, sowie den vermittelten Stoff zu Polymorphie, Vererbung und Entwurfsmustern in die Praxis umgesetzt. Somit konnten wir den Unterrichtsstoff optimal vertiefen und festigen. Für dieses Projekt hatten wir Zeit in der Schule während 18 Lektionen, sowie zuhause.

Entstanden ist dabei ein Programm, welches in Java-FX geschrieben ist mit dem MVC-Ansatz. Das Programm kann automatisch Labyrinthe generieren und diese lösen. Der Vorgang des Generierens, sowie des Lösens wird zudem grafisch dargestellt.

Entwicklungsteam: Donato Wolfisberg

Jonas Koller

# Aufgabe

Es soll ein objektorientiertes Programm erstellt werden, welches zusätzlich über ein GUI verfügt und über dieses auch gesteuert werden kann. Wir haben uns mit Herrn Bucher darauf geeinigt, dass wir unseren Fokus auf die saubere Umsetzung des Programms nach MVC machen können. Zudem wird ein rekursiver Algorithmus verwendet, sowie ein weiterer Algorithmus, welcher allerdings nicht rekursiv ist.

## Projektidee

Der Mazerator soll ein modernes, interaktives und visuelles Programm sein, dass es ermöglicht Labyrinthe zu generieren und diese danach wieder lösen zu lassen. Das Programm verfolgt dabei das Ziel die Algorithmen zur Erstellung und Lösung eines Labyrinths zu visualisieren. Der Name kommt vom Englischen «Maze», was dort Labyrinth bedeutet.

Ein weiterer wichtiger Aspekt, welchen wir mit dem Projekt verfolgen wollen ist, dass wir uns tiefer mit Algorithmen und Rekursion beschäftigen möchten. Dazu ist ein solches Programm ideal geeignet, da es eine gute Grundlage bietet, um mit diesen Algorithmen zu experimentieren. Wir möchten zudem auch die GUI-Erstellung nach MVC festigen und diverse programmatische Entwurfsmuster verwenden.

# Anforderung

|  |  |  |  |
| --- | --- | --- | --- |
| Prio | Titel | Story | Grobaufwand |
| 1 | Labyrinth generieren | Als Benutzer will ich im GUI über einen Knopf ein Labyrinth generieren lassen können, damit ich dieses nachher im GUI ansehen kann. | 6h |
| 1 | Labyrinth lösen | Als Benutzer will ich im GUI über einen Knopf ein Labyrinth lösen lassen können, damit mir im GUI der Lösungsweg des Labyrinths angezeigt wird. | 5h |
| Akzeptanzkriterien: Labyrinth generieren und lösen | | | |
| 1 | Labyrinth generieren mit Animation | Als Benutzer will ich beim Generieren des Labyrinths sehen, welches der Stack ist, welche Zellen besucht wurde und welches die aktuelle Zelle ist, damit ich besser verstehe, wie der Algorithmus funktioniert. | 2h |
| 2 | Labyrinth lösen mit Animation | Als Benutzer will ich beim Lösen des Labyrinths sehen, welches die Queue ist und welches die aktuelle Zelle ist damit ich besser verstehe, wie der Algorithmus funktioniert. | 1h |
| Akzeptanzkriterien: Labyrinth generieren und lösen werden visuell dargestellt | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Geschwindigkeit | Als Benutzer will ich im GUI einstellen können, wie schnell die Animationen sein sollen beim Generieren und Lösen, damit ich mit diesem Wert herumspielen kann und sich mein Nutzererlebnis verbessert. | 1h | |
| 3 | Farben anpassen | Als Benutzer will ich die Farben, welche für die Animationen genutzt werden, anpassen können, damit ich die Applikation individuell anpassen kann. | 0.5h |
| Akzeptanzkriterien: Benutzer kann die Applikation anpassen | | | |

# Zeitplanung

|  |  |  |  |
| --- | --- | --- | --- |
| **Titel** | **Beschreibung** | **Soll** | **Ist** |
| Grundgerüst | Projekt aufsetzen und MVP bereitstellen | 20‘ | 20‘ |
| Wissen aneignen Generieren | Algorithmen Labyrinth Generierung einarbeiten, passenden Algorithmus finden | 180‘ | 190‘ |
| Labyrinth generieren | Zuvor erarbeitetes Wissen in Praxis einsetzen und den Labyrinth Generator implementieren | 120‘ | 150‘ |
| Wissen aneignen Lösen | Algorithmen Labyrinth Lösung einarbeiten, passenden Algorithmus finden | 180‘ | 200‘ |
| Labyrinth generieren | Zuvor erarbeitetes Wissen in Praxis einsetzen und den Labyrinth Löser implementieren | 120‘ | 140‘ |
| Animationen | Animationen für das Generieren und Lösen umsetzen | 180‘ | 220‘ |
| Parameter | Geschwindigkeit | 60‘ | 60‘ |
| Farben anpassen | Farben für Animationen einstellbar | 30‘ | 30‘ |

## Paketzuteilung

Das gesamte Projekt wurde gemeinsam als Partnerarbeit geplant, umgesetzt und dokumentiert, wodurch die Aufgaben zusammen erledigt wurden.

# Projektplan

Unser Projekt werden wir mit JavaFX erstellen, da wir im Geschäft Java verwenden und damit besser vertraut sind als mit C#. Die Bedienung des Programms soll über ein GUI (Graphical User Interface) möglich sein. Bei JavaFX werden wir den MVP (Model, View, Controller)-Ansatz verwenden. Somit ist sichergestellt, dass wir unsere Daten sauber von der Darstellung trennen können. Über das GUI sollen Einstellungen gemacht werden können.

Wir werden für das Generieren der Labyrinthe den Recursive-Backtracking-Algorithmus verwenden. Das ist ein rekursiver Algorithmus, welcher garantiert, dass jedes Feld einen Weg zum Start hat. Somit können wir den Endpunkt leichter bestimmen. Der Recursive-Backtracking-Algorithmus wird später noch genau beschrieben.

Zum Lösen des Labyrinths benutzen wir den Dijkstra-Algorithmus. Das ist ein Algorithmus der «Greedy»-Klasse. Somit ist dieser sehr schnell und gut optimiert. Alternativ hätten wir auch den Breitensuche-Algorithmus (BFS) oder den Tiefensuche-Algorithmus (DFS) verwenden können. Da der Dijkstra aber der optimierteste für diese Aufgabe ist, haben wir uns für diesen entschieden.

Um die Daten aus dem Algorithmen auf dem GUI darzustellen und diese sauber vom GUI zu trennen verwenden wir Events. Die Algorithmen rufen also «update»-Events auf. Das GUI implementiert ein Interface und kann dann auf die Updates «subscriben». So können die Informationen effizient weitergeleitet werden und der Code strukturiert gehalten werden.

Bei dem Speichern der Labyrinth-Daten haben wir uns für einen Objektorientierte Variante entschieden. Wir haben die Einzelnen Zellen als Klasse abgebildet und das gesamte Labyrinth ist auch eine Klasse. So können wir Informationen in der Zelle selbst speichern (z.B. ob sie schon besucht ist) und solche für das ganze Feld (z.B. wie gross es ist).

Zum zeichnen des Labyrinths verwenden wir das JavaFX-Canvas. Dieses funktionier ähnlich wie ein HTML-Canvas. Wir können sagen von wo bis wo eine Linie gemalt werden soll, oder wo eine Fläche ausgemalt werden soll. So können wir das Labyrinth darstellen.

## Pattern

Wie oben bereits angetönt verwenden wir ein Observer-Muster. Dieses kommt an drei Stellen zum Einsatz und ermöglicht es uns, dass die Daten an das GUI weitergegeben werden können. Dies zum einen, da wir ansonsten gegen das MVC-Muster verstossen und zum anderen, da JavaFX nur einen Thread zulässt, welcher Änderungen am GUI machen kann. Damit wir die «Listeners» auslagern konnten, mussten wir noch ein drittes und ein viertes Interface machen, welches die Information, ob das generieren des Labyrinths fertig ist, weiter an das GUI gibt, um dort Buttons zu steuern.

## Algorithmen

Da wir immer wieder ändernde Labyrinthe wollten und nicht ein Labyrinth immer wieder lösen, mussten wir einen Generations- sowie einen Lösungsalgorithmus verwenden. Diese werden hier genauer beschrieben.

### Recursive backtracker

Der Recursive backtracker wurde zum generieren verwendet. In Worte gefasst macht der Algorithmus folgendens:

1. Startzelle setzen und diese als besucht markieren
2. Solange es unbesuchte Zellen hat
   1. Wenn die aktuelle Zelle unbesuchte Nachbarn hat
      1. Einen zufälligen Nachbarn auswählen
      2. Die aktuelle Zelle dem Stack hinzufügen
      3. Die Wand zwischen der aktuellen Zelle und dem ausgewählten Nachbarn entfernen
      4. Den ausgewählten Nachbarn als aktuelle Zelle setzen und als besucht markieren
   2. Sonst, wenn der Stack nicht leer ist
      1. Eine Zelle aus dem Stack holen via «pop»-Methode
      2. Diese Zelle zur aktuellen Zelle machen

Der Algorithmus ist beendet, wenn der Stack leer ist.

### Dijkstra-Algorithmus

Der Dijkstra wird wurde verwendet, um die generierten Labyrinthe wieder zu lösen. Auch hier eine in Worte gefasste Anleitung, wie dieser funktioniert:

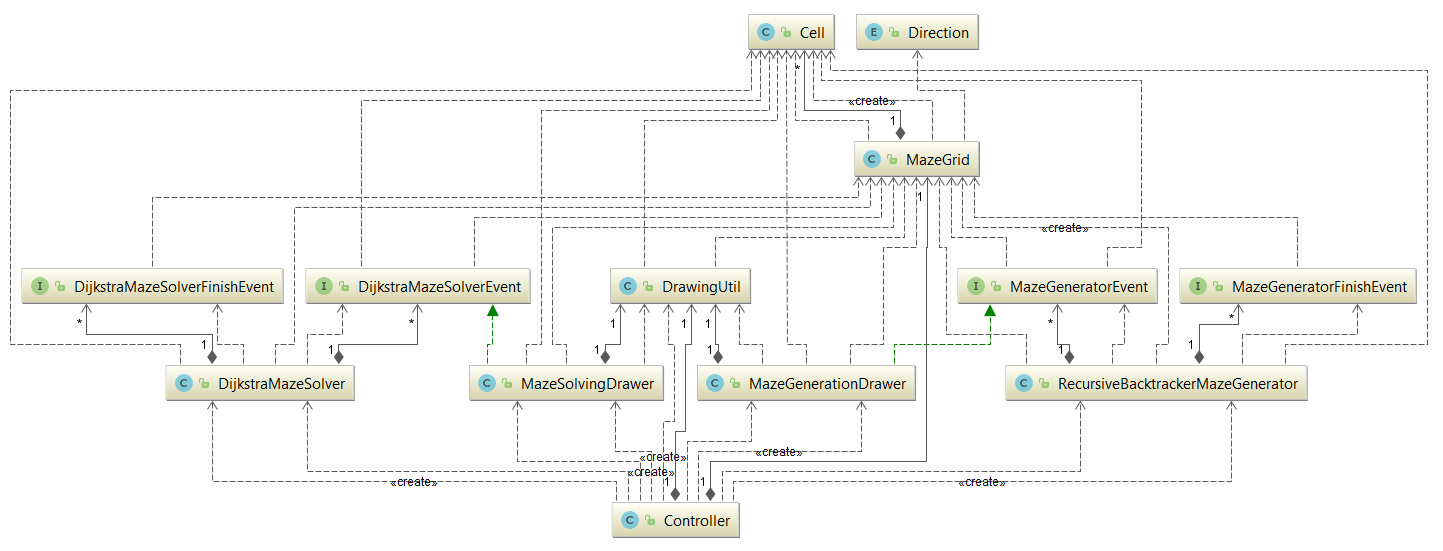
1. Weise allen Zellen die beiden Eigenschaften „Distanz“ und „Vorgänger“ zu. Initialisiere die Distanz im Startknoten mit 0 und in allen anderen Knoten mit ∞ (bei uns Integer.MAX\_VALUE).
2. Solange es noch unbesuchte Zellen gibt, wähle darunter denjenigen mit minimaler Distanz aus und
   1. speichere, dass diese Zelle schon besucht wurde.
   2. Berechne für alle noch unbesuchten Nachbarzellen die Summe des jeweiligen Kantengewichtes und der Distanz in der aktuellen Zelle.
   3. Ist dieser Wert für eine Zelle kleiner als die dort gespeicherte Distanz, aktualisiere sie und setze die aktuelle Zelle als Vorgänger.

Dieser Schritt wird auch als *Update* bezeichnet.

Der Algorithmus ist beendet, wenn alle Zellen besucht wurden.

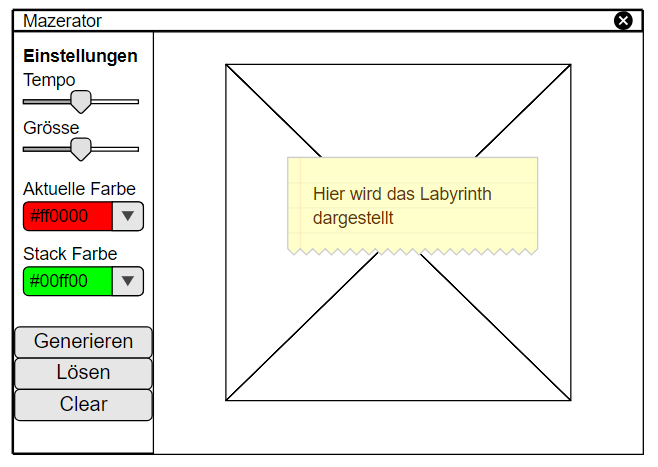
## Klassendiagramm

Folgend kommt das Klassendiagramm von unserer Applikation. Die Interfaces sind grün und mit einem I markiert, die Klassen blau und mit einem C markiert.

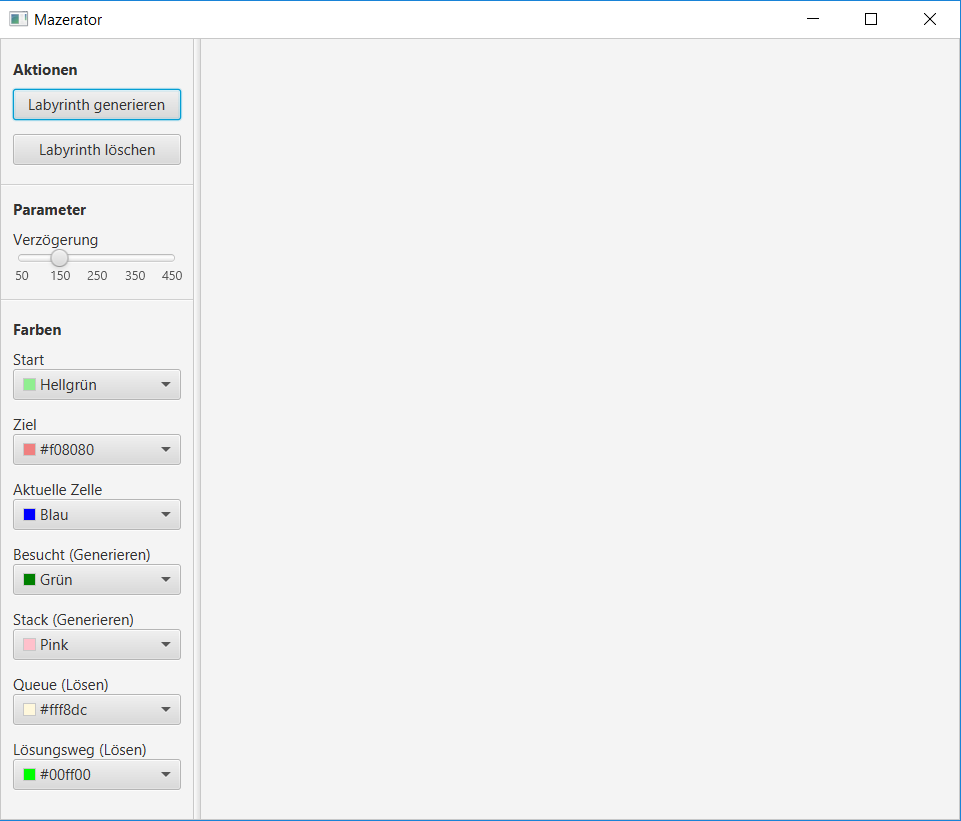


## Mockup

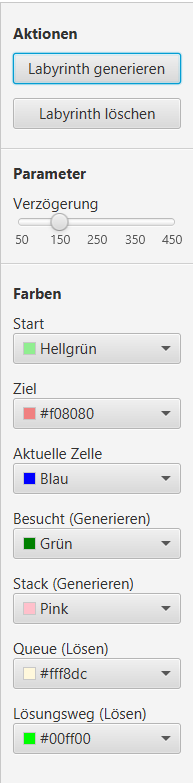
Wir stellen uns unser Endprodukt in etwa so vor:



# Anwendung

Das GUI ist in 2 Teile gespaltet. Es wurde ein Split Screen verwendet, damit diese Grössen angepasst werden können.

Auf der linken Seite kann Parameter einstellen für sein Labyrinth, auf der rechten wird das Labyrinth angezeigt.

Mit dem Knopf «Labyrinth generieren» kann ein neues Labyrinth generiert werden. Falls breites eines vorhanden ist, wird der Knopf zu «Labyrinth lösen» umbenennt. Wenn man diesen drückt, wird das zuvor generierte Labyrinth gelöst.

Bei der Verzögerung lässt sich der Abstand in Millisekunden einstellen, zwischen den Aktionen beim Generieren und Lösen. Das heisst, das zwischen den Animationen jeweils soviel gewartet wird.

Hier können die Farben, welche bei den Animationen verwendet werden sollen konfiguriert werden. Alle Farben sind mit einem Label beschrieben und sollten selbsterklärend sein.

Ganz oben gibt es noch den «Labyrinth löschen»-Knopf. Wenn dieser gedrückt wird, wird das Labyrinth geleert und man kann ein neues erstellen lassen.

Auf der rechten Seite wird ein 800x800px Labyrinth dargestellt. Dieses ist nicht interaktiv.

# Known Issues

Leider ist das Programm nicht ganz Bugfrei. Es kann vorkommen, dass es beim Generieren oder Lösen plötzlich abbricht. Das ist, da JavaFX Probleme mit mehreren Threads hat. Wir brauchen diese aber, da wir das Generieren und Lösen auslagern wollen. Leider konnten wir noch keine vernünftige Lösung dafür finden.

# Programmcode

Im ZIP-File «materator.zip» befindet sich nur der Sourcecode des Projekts.

Im ZIP-File «mazerator-mit-intellij-einstellungen.zip» befinden sich auch noch meine Intellij-Einstellungen.

Die Datei «Mazerator.jar» ist das fertige Programm, welches via Doppelklick ausgeführt werden kann, wenn Java 8 installiert ist.

Der Sourcecode ist ebenfalls auf Github unter <https://github.com/JonasKoller/m226-120-mazerator> zu finden.



# Testprotokoll

Wir konnten alle Anforderungen umsetzen. Wir haben diese umgesetzten Anforderungen praktisch in unserer Anwendung auf ihre Funktionalität getestet.

**Labyrinth erstellen** und **Labyrinth generieren mit Animation**: Beim Klick auf den «Labyrinth generieren»-Knopf wird auf der rechten Seite ein Labyrinth generiert. Dies passiert mit einer Animation. Es ist visuell zu erkennen, welche Zellen noch nicht besucht wurden, welche auf dem Stack sind und welche die aktuelle Zelle ist.

**Labyrinth lösen** und **Labyrinth lösen mit Animation:** Beim Klick auf den «Labyrinth lösen»-Knopf wird auf der rechten Seite das bestehende Labyrinth gelöst. Dies passiert mit einer Animation. Es ist visuell zu erkennen, welche Zellen noch in der Queue sind und welches die aktuelle Zelle ist. Zudem wird am Ende des Lösungsvorgangs der Lösungsweg markiert.

**Geschwindigkeit**: Ich kann, bevor oder während ich ein Labyrinth generiere oder löse, die Geschwindigkeit verstellen am Slider. Wenn ich dies mache, während die Animation bereits läuft, passt sich die Geschwindigkeit direkt an.

**Farben anpassen**: Wenn ich auf der linken Seite bei einem der ColorPicker eine neue Farbe einstelle, wird diese bei der Animation so dargestellt. Wenn die Animation bereits läuft, kann ich trotzdem Farben einstellen.

# Rückblick

Rückblickend sind wir sehr zufrieden mit unserer Arbeit und der Anwendung. Da beide von uns bereits einige Kenntnisse in Java haben, war es uns wichtig, ein eher schwierigeres Projekt umzusetzen, indem wir einen rekursiven Algorithmus und einen Greedy Algorithmus verwendeten. Des Weiteren haben wir uns an den MVC Ansatz gehalten und die Daten vom GUI strikt getrennt. Die Zusammenarbeit funktionierte hervorragend, da vieles gemeinsam erledigt wurde und man sich gegenseitig stark unterstützen konnte. Es war einem jederzeit klar was der Andere für Vorstellungen und Ziele hatte und die Aufgaben wurden fair geteilt. Durch unsere Vorkenntnisse konnten wir sehr schnell und effizient arbeiten und wir kamen gut voran, trotzdem achteten wir stark auf einen sauberen und gut strukturierten Programmcode.

Donato Wolfisberg, Jonas Koller  
Donnerstag, 14.06.2018  
Tribschenstrasse, Luzern