Lernfortschritt M226a

# 01.09

Das Github-Projekt wurde erstellt. Ich habe mich mit Git auseinandergesetzt und den Vorteil von Versionskontrollsystemen in «Teamarbeiten» gelernt. So konnte ich Änderungen machen und meine Gruppenmitglieder (Alain und Seraphin) konnten diese Änderungen automatisiert übernehmen lassen.

# 02.09

Ich habe zum ersten Mal Ressourcen in einem Programm angewandt. Dabei ist mir auch aufgefallen, dass ich die Ressourcen zwar lokal funktionieren, aber nicht auf einem anderen Computer. Dies lag daran, dass ich die Pfade nicht relativ angegeben hatte. Auch musste ich die Texturen markieren, welche in das endgültige Programm einfliessen sollten.

## https://akrick.files.wordpress.com/2013/03/mvvm.pngMVVM

Ein Nebenpunkt hierbei ist das MVVM-Prinzip (**M**odel-**V**iew-**V**iew**M**odel). Dieses besagt, dass man die View möglichst von der Logik trennen sollte. Diese Lose Kopplung sorgt dafür, dass die View unabhängig der Logik ausgetauscht werden kann. So kann ich beispielsweise meine App an ein neues Bildformat / Design anpassen, ohne die Logik zu verändern.

In meinem Beispiel

# 08.09

https://akrick.files.wordpress.com/2013/03/mvvm.png

An diesem Tag habe ich mich mit der Problematik der Farben auseinandergesetzt. So gibt es in C# SolidColorBrush, ColorBrush, Color (System) und noch eine weitere Color. Diese können natürlich nicht untereinander verwendet werden.

## Factory-Pattern

Mit meinem Path (mein allgemeines DTO für die Bewegungslogik) habe ich mich mit dem Factory-Pattern auseinandergesetzt. Dies bedeutet, dass ich von einer externen Klasse aus nicht den Konstruktor des Path’s aufrufe. Ich rufe stattdessen die (normalerweise als Singleton umgesetzte) Instanz der Factory auf. Danach konfiguriere ich mein Endprodukt. Zum Beispiel sage ich, in welche Richtung der Weg geht und wie oft er in diese Richtung gehen kann.

## Singleton

Falls eine Instanz eines Objects global verwendet wird, kann man dies mit einem Singleton umsetzen. Dies kann in C# auf mehrere Weisen umgesetzt werden. Man kann ein Framework die Instanz erstellen lassen. In C# wäre das Prism -> Unity. Oder man verwendet ein statisches Property. Dieses heisst dann beispielsweise „Instance“ und man kann global über „Class.Instance“ den Singleton bekommen. Ich verwende dies beispielsweise für meine PathFactory.

# 10.09

Hier habe ich mein erstes User-Control erstellt. Dies kann danach im GUI als eigenes XAML-Element verwendet werden. Der Vorteil hierbei ist, dass ich den Code-Behind des XAML-Elements unter Kontrolle habe. So kann ich bei mir die Schachfigur Selektieren oder sogar bewegen.

# 28.09

An diesem Tag wurde die erste Methode überschrieben, nämlich die Equals-Methode. In C# können nur Methoden überschrieben werden, welche das Keyword „Virtual“ besitzen. Die Überklasse kann danach über das Keyword „base“ aufgerufen werden.

# 29.09

An diesem Tag habe ich auf Grund des grossen „Lags“ das C# spezifische Keyword Async und Await angewendet. Wenn ich eine Methode Async mache, muss diese beim Aufrufen mit dem Keyword Await aufgerufen werden. Dieses Refactoring hat sich trotz des Aufwandes gelohnt. Async Await sorgt dafür, dass die Async Methoden auf einem Hintergrund-thread laufen. Dadurch wird der Mainthread (GUI-thread) nicht überbelastet / blockiert. Wenn das passiert ergibt es das allbekannte „Programm reagiert nicht mehr“ Fenster.

## Race Condition

In meinem Programm kommt dies zum Glück nicht vor, und Await Async sorgen oft auch für Synchronisation. Trotzdem kann es passieren, dass zwei nebenläufige Methoden die gleichen Variablen verändern. Dies kann dazu führen, dass der Wert bei jedem Ausführen ein anderes Ergebnis beinhaltet. Es ist dann wie ein Rennen. Je nachdem wer schneller ist gibt es ein anderes Ergebnis.