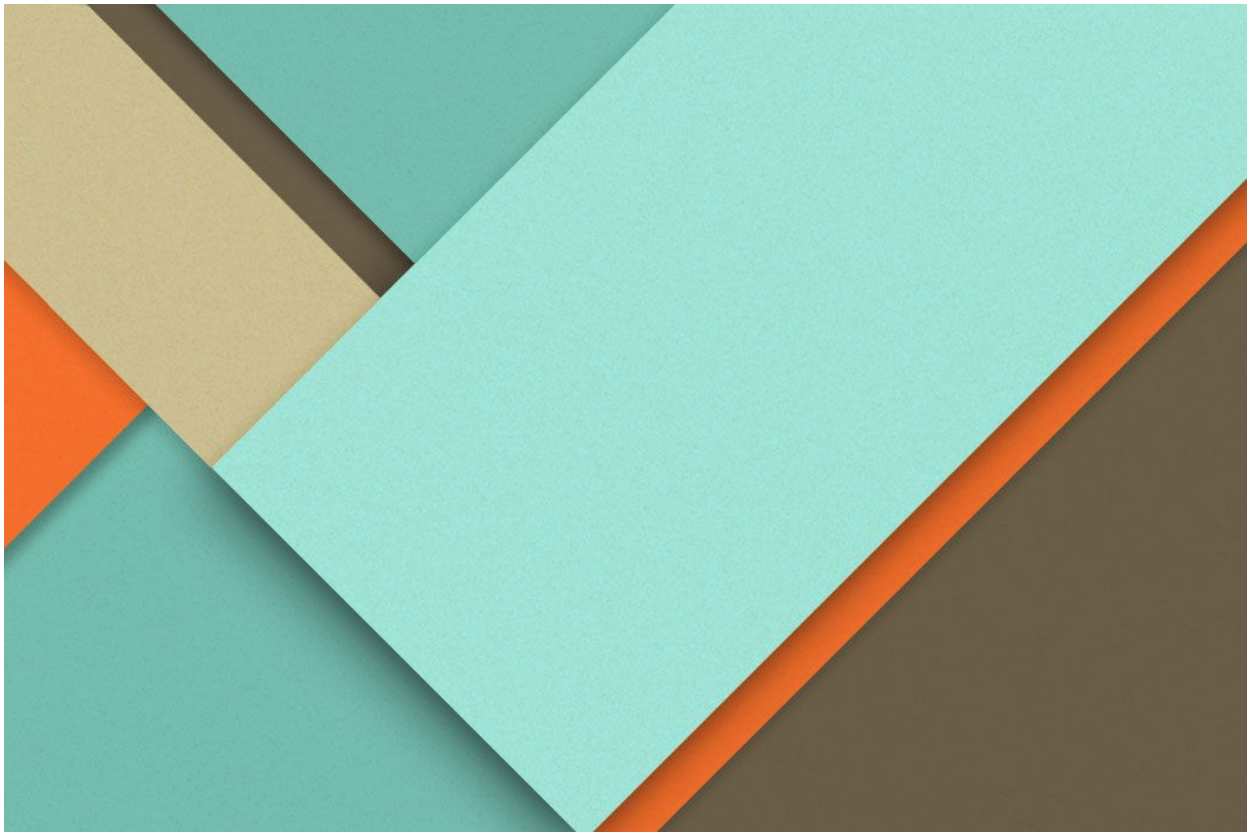

Schiffe versenken

17.06.2018

Alexander Oberländer
FA75
https://curtisy.visualstudio.com/Battleships (Privates Repository)

Übersicht

Im Rahmen der AS Halbjahresaufgabe entwickelte ich mit Hilfe der Windows Presentation Foundation (WPF) ein mehr oder weniger generisches Schiffe versenken, dass mit wenig Arbeit modular erweiterbar ist, dabei aber seine Einfachheit beibehält

Ziele

1. Einen Überblick über WPF verschaffen und die Grundstruktur des Spielbretts bauen
2. Die Schiffplatzierung anhand der Spezifikation im Skript nachstellen und einen Algorithmus entwerfen, der diese Aufgabe übernimmt und prüft
3. Einstellungen für Spiel gegen den Computer bereitstellen, sowie andere “Goodies”
4. Algorithmen finden oder entwerfen, die verschiedene Schwierigkeitsgrade abbilden

Spezifikation

Der Umgang mit WPF war mir neu und schien, vor allem ohne den XAML-Designer recht schwer, deshalb sollte das Spiel von der Oberfläche einfach gehalten werden, dafür aber mehrere sinnvolle Funktionen abbilden können. So kann man einen Rundenzähler und eine Stopwatch einstellen und gegen eine AI spielen. Das Board ist aufgebaut aus 10x10 Rechtecken in einem WrapPanel, da dieses sich automatisch um die Anordnung kümmert.

Die Koordinaten werden über die Uid gesetzt, da es sonst keinen sinnvollen Weg gibt, außer das Rectangle zu erweitern. Darauf baut auch die grundlegende Logik der Algorithmen auf.

Algorithmen

Die Algorithmen sind eine Auswahl der BattleShip AI Competition, angepasst für die Nutzung auf WPF. Im Grunde musste ich diese komplett umschreiben. Ursprünglich war die Idee drei Algorithmen zu nutzen, doch Dreadnought stieß auf grundlegende Implementierungshindernisse, die es unmöglich machten im Zeitplan zu bleiben, weshalb ich diesen entfernen musste. Farnsworth und BP7 sind aber mehr als ausreichend.

Meilensteine

I. Automatische Schiffplatzierung

Es war gar nicht so leicht über eine vorgegebene Anzahl an Schiffen eine zufällige Platzierung zu gewährleisten, am Ende habe ich eine Art Brute-Force Variante gefunden, die diesen Fall meistens zuverlässig abbildet.

II. Algorithmen

Die Algorithmen zu finden war kein Problem, jedoch waren diese gänzlich unbrauchbar, weshalb ich mich an deren Logik herantasten musste und diese auf deren Grundlage geradezu neu implementieren musste. Dabei war es möglich alles so zu vereinheitlichen, dass beide nahezu identische Methodenaufrufe benötigen.

Hindernisse

III. WPF

Ich glaube nicht, dass diese Plattform irgendeinen Vorteil gegenüber Winforms hat, sie ist zwar Event-basiert, aber das macht es nur noch schwerer richtig mit Aktionen umzugehen. Das Setup ist mühsam und man kommt schnell an den Punkt, bei dem man 3 mal das gleiche für eine etwas andere Checkbox implementiert, ganz zu schweigen davon, dass man ohne XAML-Designer kaum eine wirklich schöne und intuitive Nutzeroberfläche hinbekommt. Glücklicherweise lag mein Fokus nicht darauf

IV. DependencyObjects

Habe ich zwar vorher schon gemacht, jedoch ist auch das durch WPF gänzlich unintuitiv. Wenn ich einen PArent habe, erwarte ich mehr oder weniger auch, dass ich damit ohne einen Cast etwas machen kann. Genau das ist leider nicht der Fall, weshalb ich mir eigene Extensions schreiben musste, die diese Aufgabe für mich übernehmen. Eigentlich ist das auch ein Meilenstein, da es das Projekt deutlich vorangebracht hat.

Fazit

Das Spiel funktioniert mehr oder weniger ohne Fehler. Manchmal kommt es bei der Platzierung noch zu angrenzenden Feldern, aber diesen Bug konnte ich nicht fixen. Dafür funktionieren die Algorithmen ohne Fehler und das ganze ist meiner Meinung nach leicht verständlich geschrieben. Ich weiß auf jeden Fall, dass das mein letztes WPF Projekt war, die Erfahrung war es wert, aber Winforms schafft das gleiche in weniger Zeit, da man sich nicht um die Events kümmern muss. Ich bin etwas unzufrieden mit der Schiffplatzierung vom User, aber WPF bietet leider auch keine andere Möglichkeit. Hat man einmal das Prinzip verstanden, ist es auch logisch.

Am meisten gelungen ist mir die Modularität. Es sollte nicht lange dauern in den bestehenden Code weiter Funktionen zu integrieren, wie zum Beispiel Multiplayer. Alle Funktionen, die dazu nötig sind, haben Parameter um dies abzubilden. Der Code an sich ist etwas unordentlich strukturiert, aber zum refactoren blieb mir am Ende leider keine Zeit mehr. Ansonsten hat das Projekt großes Potenzial weiterentwickelt zu werden, was ich eventuell auch tun werde, da es doch Spaß gemacht hat daran zu arbeiten.

Eine weitere Alternative wäre, das Projekt unter .NET Core laufen zu lassen, zum Beispiel per AvaloniaUI oder libUI. Dazu habe ich auch schon erste Versuche unternommen, nur gibt es die meisten Kontrollstrukturen wie das WrapPanel dort nicht, weshalb der Weg dort noch einige Zeit mehr in Anspruch nehmen würde.