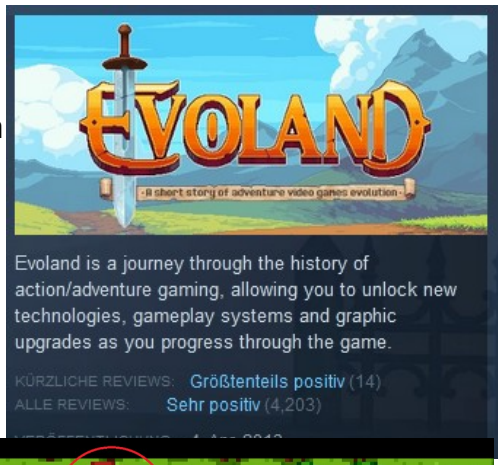
4.1.2 Modell View Controller (MVC)



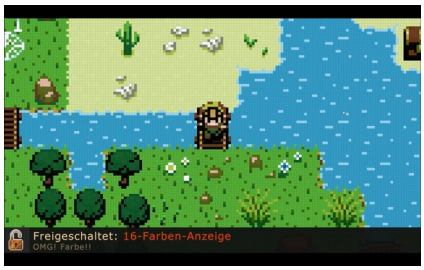
Wir haben in unserer Karriere als Informatiker an der Schule bereits einige Software geschrieben. Das Pattern MVC stellt eine mögliche Standardlösung für die Konzeption jedes Programms und gliedert ein Programm in verschiedene Aufgabenbereiche. Schauen wir uns die ganze Sache an einem Beispiel an.

Wir betrachten das Spiel Evoland, an dem wir uns alle drei Bestandteile einer Software anschauen können.

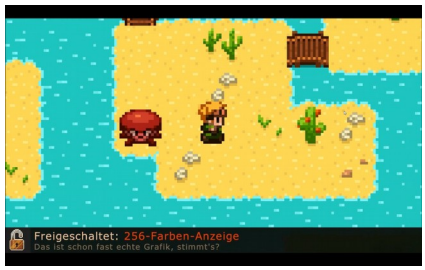
Beim Erstellen eines Spieles wie z.B. Evoland, überlegen wir normalerweise, welche Objekte auf dem Spielfeld zu sehen sind. In unserem Spielscreenshot sehen wir, dass es z.B. einen Spieler und Monster gibt. Zu allen diesen Spielobjekten würden wir ein Objekt einer Klasse anlegen und so unseren „Spielstatus“ speichern. Diesen Teil nennen wir das **Modell** des Programms.

Das Spiel Evoland handelt von der Entwicklung der Videospiele, also kommen dabei auch Grafikupdates zum Einsatz. Dabei erkennen wir den zweiten Bestandteil eines Programms, die sogenannte **View**. Das Spiel verändert sich in den Screenshots unten nicht wirklich, was beispielsweise das Spielprinzip oder die Monster angeht. Das sollte auch so sein, da die Anzeige getrennt von der Logik und dem Modell sein sollte, um jederzeit ein „Grafikupdate“ durchführen zu können. Große Free-To-Play Titel wie League of Legends oder Fortnite zeigen dieses Prinzip perfekt. Das Spielprinzip ändert sich nicht, egal welchen „Skin“ (welche Charakterdarstellung) man auswählt.

8-Bit Grafik



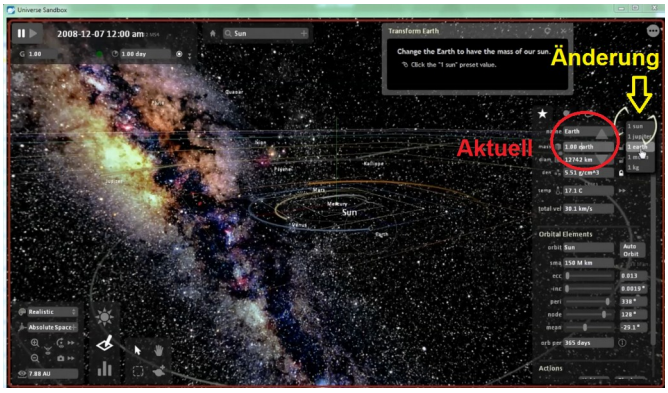
16 Bit Grafik



256 Farben

Free-To-Play Titel verdienen oft ihr Geld mit dem Verkauf von kosmetischen Veränderungen des Spieles. Hier links der inzwischen antike Screenshot aus dem Jahre 2015, indem alle optischen Veränderungsmöglichkeiten eines Charakters angezeigt werden. Spielerisch verändert sich nichts. Fähigkeiten etc. haben mit allen „Skins“ den gleichen Effekt.

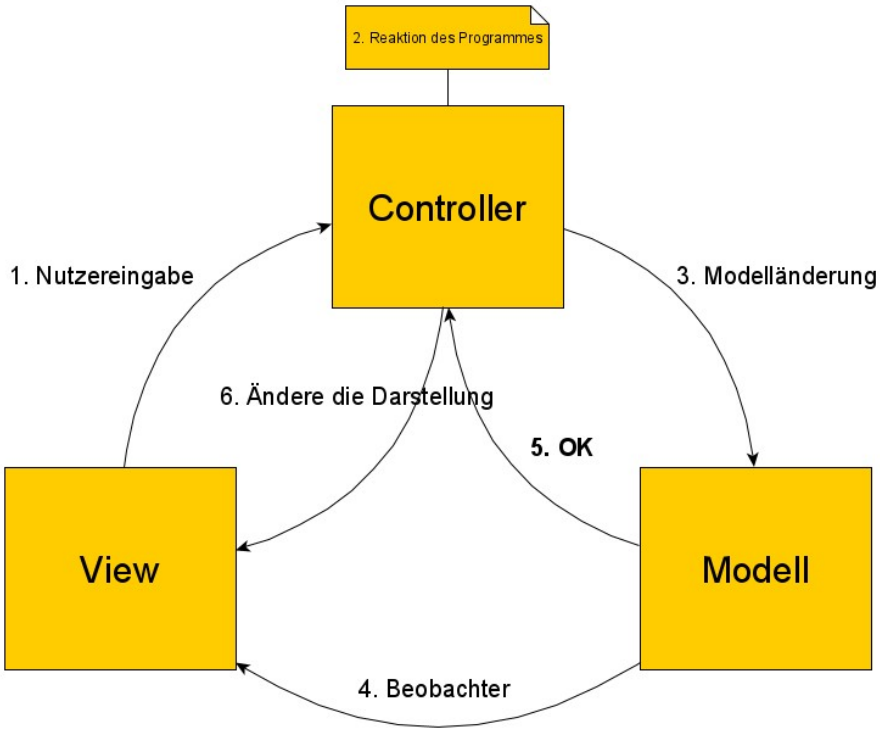
Der letzte Bestandteil ist der sogenannte Controller. Der Controller hat die Spiellogik und überprüft beispielsweise, welche Spielzüge gültig sind. In unserem Beispiel verliert man beim Berühren eines Gegners Lebenenergie in Form von Herzen. Hat man keines mehr, heißt es Game Over. So einfach das klingt: hätte nicht der Controller diese Regeln in sich, könnte jeder „Hacker“ das Spiel überlisten und einfach weiterspielen, ohne sich an die Regel halten zu müssen. Besonders in Online Games sieht man immer wieder, wenn Controller und Modell nicht genug getrennt sind, in dem Hacker modifizierte Clients Programmieren, die z.B. Wallhacks etc. ermöglichen.[1] Als Anschauungsbeispiel sehen wir unten einen Screenshot aus dem Spiel „Universe Sandbox“, in dem man den „Controller“ eines Universums zur Laufzeit manipulieren kann. Man kann sich das so vorstellen, dass wir im Spiel die „Spielregeln“ verändern können, was typischerweise der Controller erledigen würde.



1 Anmerkung: Bei manchen Spielen, die online Multiplayer haben, sieht man sehr gut, dass man selber auf seinem Rechner nur ein Modell mit einem abgespeckten Controller hat. Der Controller im Server sagt letztendlich, welche Spielzüge passieren. Dadurch kann es bei Lags/hoher Latenz passieren, dass Spiele über die Map „springen“ anstatt durchgehend zu laufen. In den meisten modernen Spielen bemerkt man dies allerdings kaum mehr, da hier „smoothing“ Algorithmen benutzt werden, um die Clientposition „smooth“ an die Serverposition anzupassen.

Kommunikationsablauf in MVC

Die Kommunikation im Pattern MVC läuft etwa folgendermaßen:



1. Die View erhält eine Nutzereingabe (oder das spielt läuft einfach weiter).
2. Der Controller entscheidet, welche Reaktion das Programm/Spiel zeigen sollte.
3. Er passt die Werte (z.B. Leben) im Modell an.
4. Die View wird mittels Beobachter Pattern über den neuen Stand informiert (z.B. 1 Herz weniger)
5. Der Controller erhält die Nachricht, dass das Modell angepasst wurde (return)
6. Controller gibt ggf. an die View weitere Änderungen weiter (z.B. ein neues Fester wird geöffnet)

Rollen im MVC im ausführlichen Überblick

Modell (Hintergrunddaten)

Das Modell beinhaltet alle Daten des Programms. Meist besteht es aus vielen zusammengesetzten Datentypen (z.B. die Klasse SCHLANGE in Snake) und einer „Modell-Oberklasse“, die alle Manipulationsmöglichkeiten des Modells zur Verfügung stellt. Somit müssen alle anderen Klassen nur mit dieser einen Klasse kommunizieren, wenn Sie etwas am Modell ändern wollen (siehe Controller). Das Modell darf dabei auch schon gewisse Programmlogik beinhalten, so kann das Modell schon „falsche“ Werte verhindern (z.B. negative Werte bei Attributen, die nicht negativ werden dürfen etc.).

View (GUI = Graphical User Interface)

Die einzige Aufgabe der View ist es, dem Nutzer die Möglichkeit zu geben, mit dem Programm zu kommunizieren und dem Nutzer die Daten des Modells in irgendeiner Art und Weise anzuzeigen. Je nach Anwendung (z.B. Browser vs. Computerspiel) sieht der Nutzer manche Daten nur umgewandelt (wieder z.B. die SCHLANGE bei Snake). In den meisten Anwendungen bekommt die View die Daten des Modells über das Beobachter Entwurfsmuster mitgeteilt. Dies ist allerdings nur in zeitkritischen Anwendungen zwingend notwendig, da ein Umweg über den Controller langsam sein kann. Die View darf in keiner Weise direkte Kontrolle über die Daten des Modells bekommen!

Controller

Der Controller beinhaltet alles, was für die Programmlogik wichtig ist. Er entscheidet, wie das Programm auf Nutzereingaben, die ihm von der View mitgeteilt werden reagiert und verändert dementsprechend das Modell. Je nach Anwendung teilt der Controller nach erfolgreicher Änderung des Modells der View mit, dass sich das Modell geändert hat (oft in den Fällen, wo die View dies nicht so wie so über „Beobachter“ vom Modell erfahren hat, oder z.B eine neue View = zweites Fenster etc. gestartet werden soll).

Aufgaben:
1) Kopieren Sie die Vorlage MVC aus dem Klassenordner.

2) Schauen Sie sich die Klassen MODELLOBERKLASSE, VIEW und CONTROLLER an und machen Sie sich einen Überblick über deren Methoden und Attribute.

3) Betrachten Sie die Klasse START. Erstellen Sie eine neue Instanz der Klasse und Testen Sie das Programm. Wie wird sichergestellt, dass die Kommunikationswege von oben vorhanden sind?

4) Schrieben Sie eine zweite View, die die vorhandenen Strukturen anders darstellt. Nutzen Sie die vorhandenen Klassen zur Darstellung.