Kaffeeautomat

Programmier-Praktikum

von: Jesse Strathmann, Finn Geffken und
Maurice Beisel

Abgabe: 19.04.2020

Inhaltsverzeichnis

1. Projektbeschreibung

Das Projekt ist ein Unity-Programm, in dem man mit einer Kaffeemaschine interagieren kann und so ein Kaffeegetränk „kaufen“ kann. Die Kaffeemaschine soll nach dem Prinzip der Automaten aus der Informatik funktionieren. Diese Kaffeemaschine hat also zur jedem Zeitpunkt einen Zustand, welcher durch den Benutze verändert werden kann, was zu einem Zustandswechsel der Kaffeemaschine führt. Zusätzlich soll ein Zustandsdiagramm in Echtzeit angezeigt werden, wo man mögliche Zustandswechsel ablesen und den aktuellen Zustand sehen kann. Jedes Getränk hat eine Anzahl an benötigten Ressourcen, welche aus dem Speicher des Kaffeeautomaten verbrauch werden. Hinzu kommt, dass der Automat einen Vorrat an Wechselgeld hat, womit er dem Benutzer Geld zurückgeben kann, falls er nicht passend zahlen kann.

Die Mindestanforderung umfasst eine grafische Oberfläche mit einer Eingabemöglichkeit für den Benutzer und einer optischen Repräsentation eines Kaffeeautomaten mit einem Terminal und einer Geldanzeige mit zwei Signalleuchten. Angeboten werden sollten eine vorgefertigte Liste an Getränken mit einer definierten Anzahl an notwendigen Ressourcen und einem definierten Preis.

Als Erweiterung galt es, das Zustandsdiagramm in Echtzeit anzuzeigen und den derzeitigen Zustand zu markieren.

Brownie Points gibt es, wenn das Programm im Vollbildmodus ein voll bedienbarer Getränkeautomat ist, eigene Kaffeespezialitäten hinzugefügt wurden und auch für verschiedene Getränkegrößen.

2. Theoretische Überlegungen

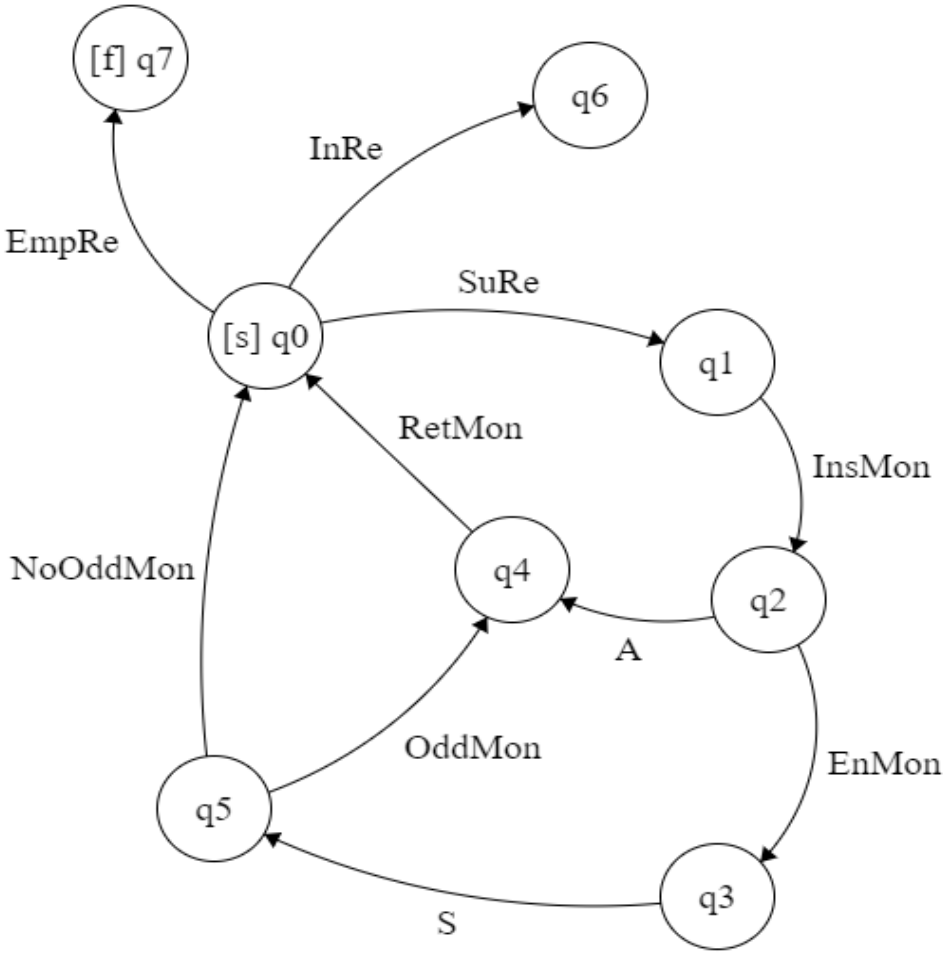
Wir bevorzugen die Umsetzung in Unity, da wir schon einmal im Mobile-GameLab an einem kleinen Projekt gearbeitet hatten und wir daran sehr viel Spaß hatten und viel Neues lernen konnten. Außerdem bietet Unity eine intuitive Art der Erstellung einer grafischen Oberfläche, was besonders bei diesem Projekt sehr angenehm ist.

Bei der Konzipierung des Automaten war es uns wichtig, dass die Zustände voneinander entkoppelt sind und dass ein Zustandswechsel von dem Automaten selbst getätigt wird, nicht von den Zuständen untereinander. Außerdem strebten wir an, dass der Automat leicht angepasst werden kann, wenn es um die Auswahl an Getränken und auch Ressourcen geht. Auch wollten wir, dass unser Automat leicht um neue Zustände erweiterbar ist, auch wenn das nicht notwendig war in diesem Projekt.

Da wir uns entschieden haben, das Projekt in Unity zu realisieren und Jesse bereits etwas Erfahrung mit der Entwicklung in Unity hatte, fanden wir es als Gruppe am sinnvollsten, wieder im Pair-Programming zu arbeiten, obwohl wir zu dritt sind. So konnten Finn und Maurice bei Verständnisfragen bezüglich Unity direkt Fragen stellen und wir konnten so produktiver arbeiten.

3. Softwareentwurf

3.1 Automatentheorie



(Bild in der .ZIP enthalten)

Legende

EmpRe → Empty Resources

InRe → Insufficent Resources

SuRe → SufficientResources

InsMon → Inserted Money

S → Start

[s] → Start State

EnMon → Enough Money

OddMon → Returns Odd Money

NoOddMon → Returns No Odd Money

RetMon → Returned Money

A → Abort

[f[→ Final State

q0 – Getränkauswahl

In diesem Zustand kann der Benutzer ein Getränk auswählen und die Anzahl an Milchportionen und Zuckerwürfel auswählen.

Gibt es genug Ressourcen für das Getränk, wird in den Zustand q1 gewechselt.

Gibt es nicht genug Ressourcen für **dieses** Getränk, wird in den Zustand q6 gewechselt.

Gibt es von vorne herein nicht genug Ressourcen für das Getränk mit den wenigsten Ressourcen, wird in den Zustand q7 gewechselt und der Automat ist im Endzustand.

q1 – Ressourcen vorhanden

In diesem Zustand weiß der Benutzer, dass das Getränk hergestellt werden kann und sieht den Preis des Getränks.

Möchte der Benutzer ein anderes Getränk, kann er „Abbrechen“ drücken und gelangt zurück in den Zustand q0.

Ist der Benutzer zufrieden mit seiner Wahl, kann er mit der Zahlung beginnen und es wird in den Zustand q2 gewechselt, sobald ein Geldstück eingeworfen wurde.

q2 – Zahlung begonnen

In diesem Zustand wird gewartet, bis der Benutzer genügend Geld eingeworfen hat um den Betrag zu begleichen.

Wurde genug Geld eingeworfen, wird in den Zustand q3 gewechselt.

Entscheidet sich der Benutzer jedoch gegen das Bezahlen, kann er auf „Abbrechen“ drücken und gelangt in den Zustand q4.

q3 – Zahlung abgeschlossen

In diesem Zustand kann der Benutzer entweder auf „Start“ oder auf „Abbrechen“ drücken.

Drück er auf „Start“ wird das Getränk hergestellt und es wird in den Zustand q5 gewechselt.

Drückt er auf „Abbrechen“ wird der Vorgang abgebrochen und es wird in den Zustand q4 gewechselt.

q4 – Geldrückgabe

In diesem Zustand erhält der Benutzer entweder sein eingeworfenes Geld zurück oder bekommt das Wechselgeld von seiner Bestellung.

Sobald das Geld entnommen wurde, wird in den Zustand q0 gewechselt.

q5 – Getränk bereit

In diesem Zustand ist das Getränk hergestellt und steht zur Entnahme bereit.

Wurde das Getränk entnommen und wurde passend bezahlt, wird direkt in den Zustand q0 gewechselt.

Erhält der Benutzer jedoch Wechselgeld, wird in den Zustand q4 gewechselt.

q6 – Fehlende Ressourcen

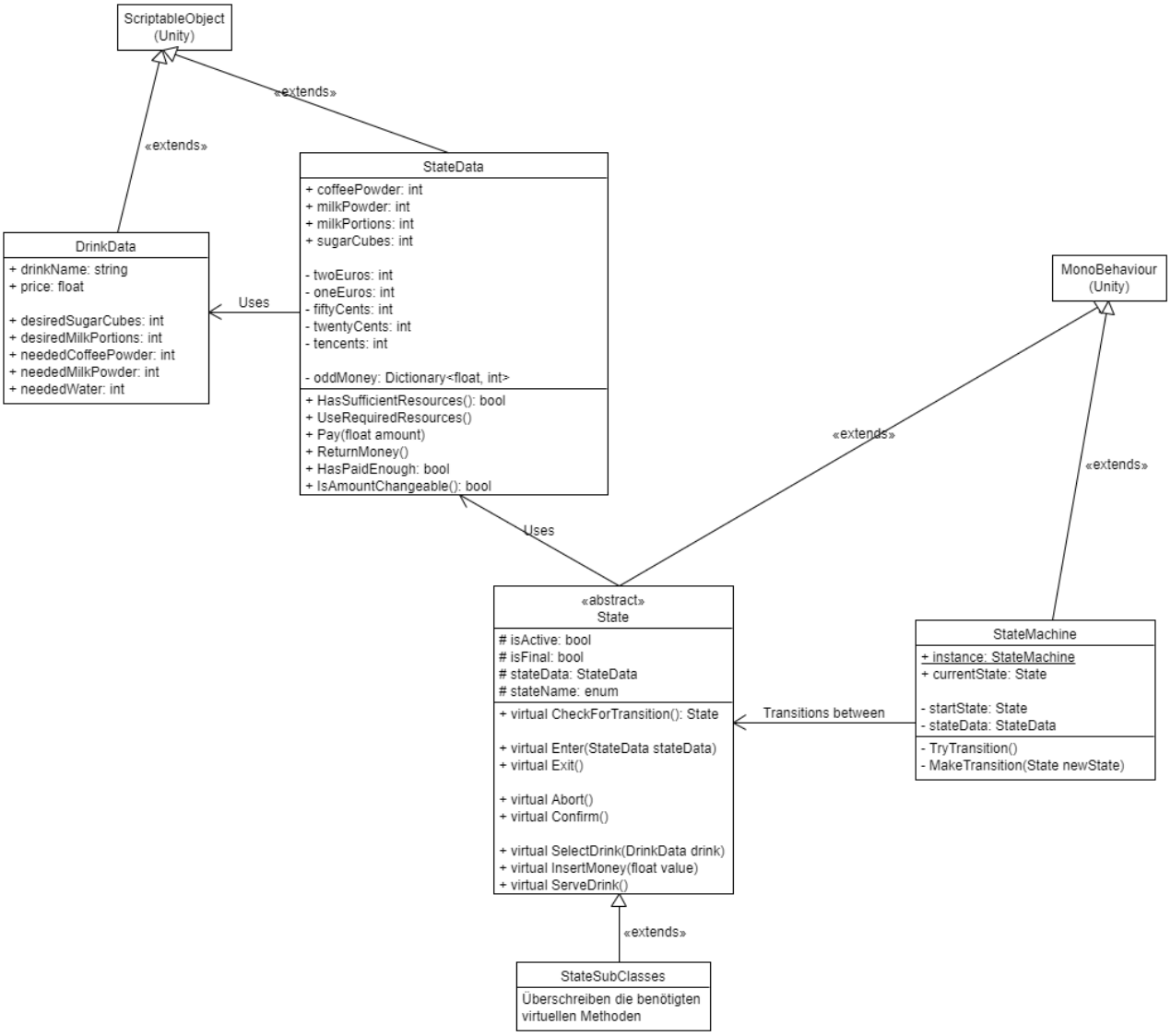
In diesem Zustand wird dem Benutzer gesagt, dass für das ausgewählte Getränk nicht mehr genügend Ressourcen vorhanden sind.

Es wird gewartet bis der Benutzer auf „Abbrechen“ drück. Anschließend wird in den Zustand q0 gewechselt.

q7 – Ressourcen erschöpft

In diesem Zustand sind keine Ressourcen mehr für das Getränk mit den wenigsten Ressourcen vorhanden. Dies ist der Endzustand des Automaten und es wird gewartet, bis das Programm neugestartet wird.

3.2 Automatenimplementation



(Bild in der .ZIP enthalten)

Unsere Überlegungen bei der Erstellung des UML-Diagrammes waren, dass der aktuelle Zustand durch eine Klasse repräsentiert wird. Dieser Zustand nimmt dann Änderungen an einem Zustandsdatenobjekt vor, welches beim Zustandswechsel in den jeweiligen Zustand übergegeben wird. (State - *Enter(StateData stateData))*

Jeder Zustand soll entkoppelt von den anderen Zuständen sein. So bekommt jeder Zustand eine Liste an möglichen, nächsten Zuständen, welche beliebig änderbar sind.

Auch den Zustandswechsel soll jeder Zustand intern bestimmen und mitteilen, falls es einen gibt. So prüft der Automat, ob der Zustand in einen Zustandswechsel gehen möchte. (StateMachine – *TryTransition()*)

Um einen Zustandswechsel zu prüfen, wird die CheckForTransition-Methode aufgerufen, welche den neuen Zustand zurückgibt, falls ein Wechsel durchgeführt werden soll. Wurde

hier ein Zustand zurückgegeben, wechselt der Automat in den neuen Zustand und der aktuelle Zustand wird verlassen. (*State – Exit(), State - CheckForTransition(), StateMachine – MakeTransition(State newState)*)

Alle Unterklassen der State-Klasse können die jeweiligen Events wie „Abort()“, „Confirm()“ oder auch „InsertMoney(float value)“ überschreiben, falls diese Events in dem Zustand gebraucht werden. In der Basisklasse State sind diese als virtuell deklariert und beinhalten eine Basisimplementierung, welche eine Nachricht in die Konsole schreibt, dass diese Methode nicht implementiert ist. Möchte ein Zustand also beispielsweise den Zustand wechseln, wenn die „Start“-Taste gedrückt wird, kann es die „Confirm()“-Methode überschreiben, um über den Druck der Taste informiert zu werden.

Das Zustandsdatenobjekt wird durch die Klasse StateData abgebildet und enthält alle Daten, die relevant für einen Zustand im Automaten sind. Außerdem hat diese Klasse Methoden, welche die Daten manipulieren können und auch logische Zusammenhänge darstellen. (z.B. Entscheidet IsAmountChangeable(), ob der eingeworfene Betrag gewechselt werden kann.)

Außerdem hat das Zustandsdatenobjekt eine Eigenschaft, welche auf eine Instanz der Klasse DrinkData verweist. DrinkData bündelt alle Daten die ein jeweiliges Getränk beinhaltet. Diese Entkopplung ermöglicht das einfache Erstellen von neuen Getränken innerhalb von Unity und sorgt für eine bessere logische Abgrenzung der Daten.

Das UML-Diagramm wurde im Endprodukt etwas erweitert und abgeändert, es spiegelt jedoch die Grundüberlegung gut wider und ist lediglich in abgewandelter Form letztendlich im Code vorzufinden.

4. Projektplanung

4.1 persönliche Zielsetzungen

Jesse

Mein Ziel für dieses Projekt ist es, komplexere Code-Architekturen in Unity zu realisieren, welche robust, aber auch gut erweiterbar sind. Darüber hinaus möchte ich vertrauter mit Unity im Allgemeinen werden und einige Konzepte der Programmiersprache C# lernen.

Finn

Maurice

4.2 Arbeitsaufteilung

Zunächst half Jesse beim Einstieg in Unity und erklärte die Grundlagen der Entwicklung in Unity. Während der Eingewöhnungsphase in Unity, erstellte Jesse das Konzept für den Automaten und begann mit der Implementation der Zustandsmaschine hinter diesem

Automaten. So konnten Maurice und Finn sich mit Unity vertraut machen und zeitgleich hatten wir eine Arbeitsgrundlage, welche die spätere Arbeitsverteilung erleichterte.

Nachdem die Zustandsmaschine fertiggestellt war, galt es die verschiedene Zustände zu implementieren und die grafische Oberfläche zu entwickeln. Hier bot es sich an, die zu entwickelnden Zustände untereinander aufzuteilen, da wird ohnehin eine Entkopplung angestrebt haben und diese durch die Zustandsmaschine gegeben war.

Nachdem die Zustände implementiert waren, entwickelten wir ein Konzept für die grafische Oberfläche. Diese setzten hauptsächlich Maurice und Finn um. Zeitgleich begann Jesse mit dem Protokoll, an dem nach Abschluss des Programmes alle betrugen.

5 Projektverlauf

5.1 Überblick

Wir nutzten hauptsächlich die Zeit der PP-Stunden und der Mitbeschäftigung genutzt um an dem Projekt zu arbeiten. So konnten wir schnelle Rücksprache halten und wussten immer, wie weit wir aktuell sind.

Zu Beginn haben Finn und Maurice ein kleine Unity-Tutorial gemacht, während Jesse mit der Konzipierung des Automatens begann. Nach den Tutorials, hat Jesse den bis dahin vorhandenen Code des Automatens erklärt und stellte die Zustandsmaschine fertig.

Mit Fertigstellung der Zustandsmaschine teilten wir die zu implementieren Zustände mit der jeweiligen Funktionalität in kurzen Stichpunkten auf und arbeiteten alleine an diesen.

Als wir alle Zustände soweit implementiert hatten, begannen Maurice und Finn mit der grafischen Oberfläche des Automaten und Jesse testete den Zustandsautomaten, um die korrekte Implementierung der Zustände und die Zustandswechsel zu sichern.
Anschließend fing Jesse an, das Protokoll zu schreiben, während Finn und Maurice noch die grafische Oberfläche entwickelten. Nach Abschluss des Programms, beendeten wir gemeinsam das Protokoll.

5.2 Testprotokolle

Getestet haben wir zunächst die Zustandsmaschine, ob die Lifecycle-Methoden der Zustände korrekt aufgerufen werden und ob die Zustandswechsel wahrgenommen werden. Dies geschah mit Dummy-Zuständen, da bis dahin noch keine Zustände implementiert waren.

Als wir die Zustände implementiert hatten, testeten wir jeden Zustandswechsel aus dem Zustandsdiagramm auf Funktionalität. Hier gab es zunächst Probleme mit dem Erkennen, ob ein Betrag gewechselt werden kann, da das eingeworfene Geld erst nicht als Wechselgeld berücksichtigt wurde. Auch die Zustandswechsel waren zunächst durch fehlerhafte Wechselbedingungen nicht immer möglich. Dies wurde jedoch schnell behoben, nachdem wir gemeinsam einmal den Ablauf durchgegangen sind.

Bei der grafischen Oberfläche war es unser Ziel, alle Eingabeelemente, die nicht verfügbar sind in diesem Zustand zu deaktivieren. Dies haben wir erreicht, in dem wir zu Beginn alle Eingabeelemente deaktivierten und jeder Zustand die gültigen Eingabeelemente zu Beginn aktivierte und beim Austritt aus diesem Zustand wieder deaktivierte. Hier war zunächst das Problem, alle Elemente zu Beginn des Programmes zu deaktivieren, da die verwendete Lifecycle-Methode aus Unity eine Referenz auf die Eingabeelemente vor der Erstellung Dieser lieferte. Durch Lesen der Dokumentation und Antworten auf Fragen in Foren, konnten wir die richtige Lifecycle-Methode für unser Vorhaben finden und so das Problem beheben.

Als das Programm fertig war, testeten wir diverse Eingaben in der GUI und prüften die Zustandswechsel erneut über die GUI. Hier stellte sich raus, dass wir eine Variable im Zustandsdatenobjekt nicht richtig zurückgesetzt hatten, nachdem es einen erfolgreichen „Getränkdurchlauf“ gab, was dazu führte, dass der Automat ungewollt in einen anderen Zustand gesprungen ist.

6 Projektergebnisse

6.1 Ergebnisse

Unser Ergebnis ist ein funktionstüchtiger Kaffeeautomat in Unity, welchem man sehr einfach verschiedene Startzustände zuweisen kann und dessen Getränkeliste sehr schnell erweiterbar ist. Die Bedienung ist sehr intuitiv, da nicht verwendbare GUI-Elemente ausgegraut werden und im Terminal klare Anweisungen stehen und dem Benutzer immer ein Feedback gegeben wird.

Auch das Wechseln von Geld und das Verbrauchen von Ressourcen funktioniert wie erwartet.

Gibt es Geld was der Benutzer entnehmen soll, wird ein Geldstapel auf dem Automaten angezeigt. Kann der Benutzer sein Getränk entnehmen, ist dort ein Kaffeebecher abgebildet, welcher klickbar ist.

Durch das Drücken der H-Taste, kann das Zustandsdiagramm ein- und ausgeblendet werden.

6.2 Mängel

Aus unserer Sicht enthält das Programm keine Mängel, die die Nutzbarkeit beeinträchtigen.

Ein potenzieller Mangel ist jedoch, die fehlende Beschriftung der Übergangsfunktionen und des Eingabealphabets im Zustandsdiagramm des Programmes, gegen welche wir uns aber bewusst entschieden haben. Wir finden, dass das Zustandsdiagramm so besser lesbar ist und auch bei Interaktion nachvollzogen werden kann, welche Aktion in welchen Zustand geführt hat.

6.3 Fazit

Jesse

Ich bin sehr zufrieden mit dem Projekt. Ich konnte zum einen viel über die Implementierung eines Zustandsautomaten lernen, aber auch die Entwicklung von grafischen Oberflächen mit Unity mit entkoppelter Logik war sehr interessant. Auch die Arbeitweise im Pair-Programming bzw. das gemeinsame Arbeiten in Anwesenheit des ganzen Teams fand ich sehr angenehm und trug essenziell zur Produktivität bei, da es direkte Rücksprachen und Hilfe gab.
Auch neue Konzepte von Unity konnte ich mitnehmen, wie zum Beispiel die Entwicklung im Stile von Komponenten. Darüber hinaus lernte ich die Verwendung von „virtual“ in C#, welche sich hier als sehr nützlich erwies.

Finn

Ich bin zufrieden mit diesem Projekt. Trotz Jesse’s Hilfe, habe ich trotzdem das Gefühl mich gut eingebracht zu haben. Ich konnte mich gut in Unity und die Entwicklung von Automaten mit entkoppelten Zuständen einarbeiten, dank Video-Serien auf YouTube und Jesse’s Hilfe. Das Pair-Programming nach meiner persönlichen Einarbeitung hat gut funktioniert und ich kann mich Jesse bezüglich der Produktivität in jedem Fall anschließen. Nach anfänglichen Problemen mit C# konnte ich mich dann zusammen mit Maurice, mit der Entwicklung der grafischen Oberfläche ein wenig besser ausleben, und ich denke das ist uns auch gut gelungen.

Maurice