

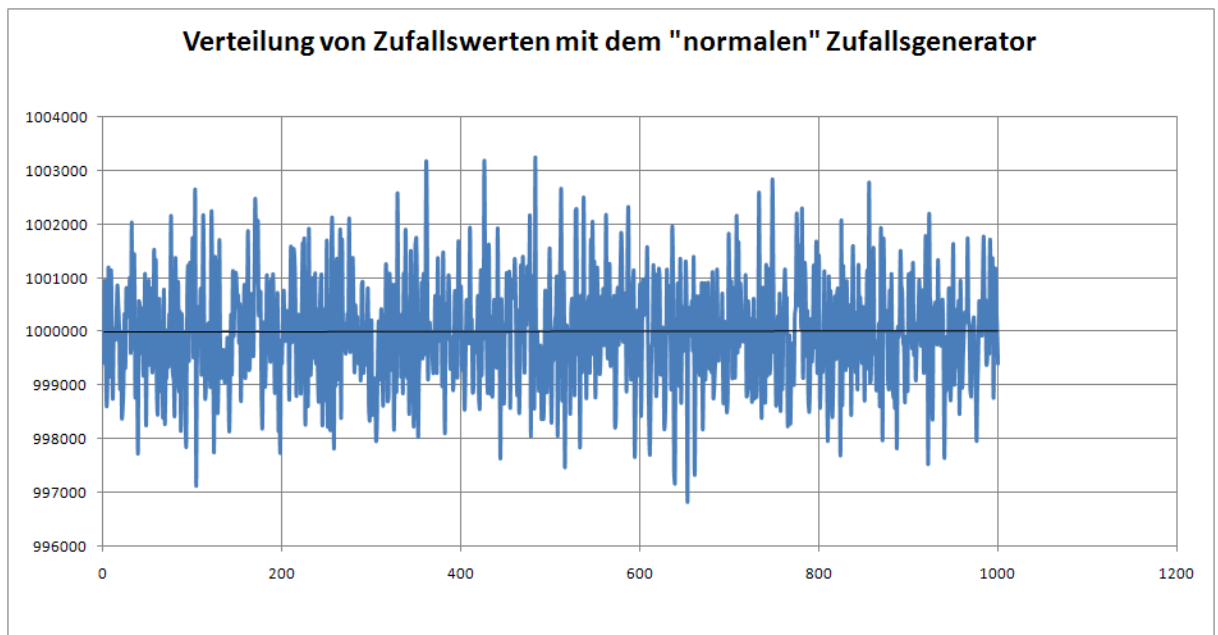
Tankomatik

1. Preisberechnung

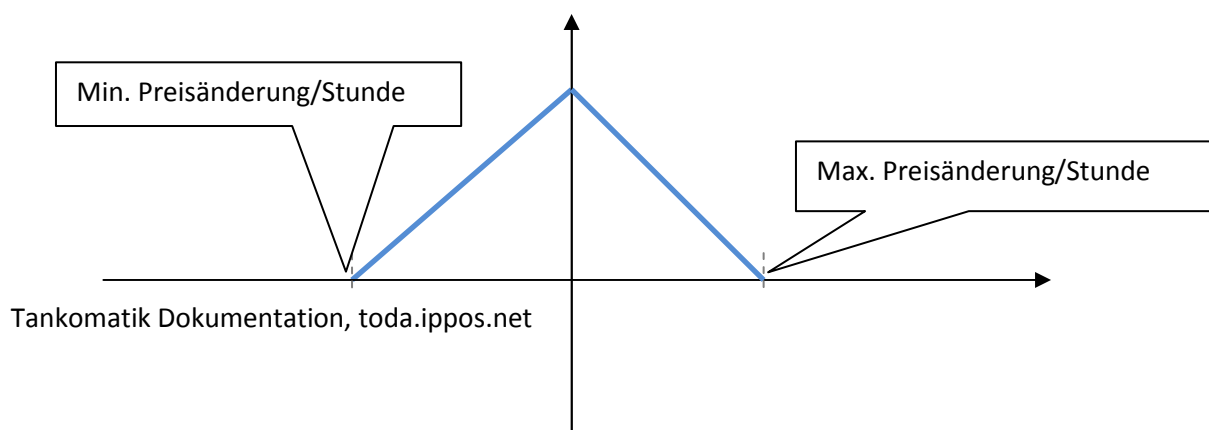
Ich habe mich entschieden innerhalb des Programms nicht in Euro oder Cent zu rechnen, da sich Preisänderungen auch im 100stel-Cent-Bereich abspielen und langfristig ausschlaggebend sein können. Darum wird im Programm mit Geldeinheiten gerechnet, wobei 10000 Geldeinheiten einem Euro entsprechen.

Für die Preisberechnung hatte ich als erstes den Ansatz, dass sich der Benzinpreis zyklisch entwickelt: Am Nachmittag ist der Preis höher als in der Nacht, am Wochenende höher als unter der Woche, um die Ferienzeit höher als um die Nebensaison usw. Dazu sollte eine generelle Preisentwicklung zählen: Dass der Benzinpreis im Trend steigt: trotz Schwankungen stieg der Benzinpreis in den letzten 10 Jahren. Natürlich reicht es nicht, wenn man all diese Preisänderungen miteinander verrechnet. Man benötigt noch den Zufall: Krisen, Spekulationen, Lieferengpässe und Unvorhersehbares.

Nach längerem Überlegen bin ich dann zu dem Entschluss gekommen, dass sich der Benzinpreis nicht durch solche Funktionen vorhersagen lässt. Darum habe ich versucht, den Benzinpreis nur durch einen Zufallsgenerator zu erzeugen. Wenn man viele ganzzahlige Zufallswerte zwischen 1000 und -1000 berechnet und diese in einen Graphen zeichnet, erhält man folgende Verteilung

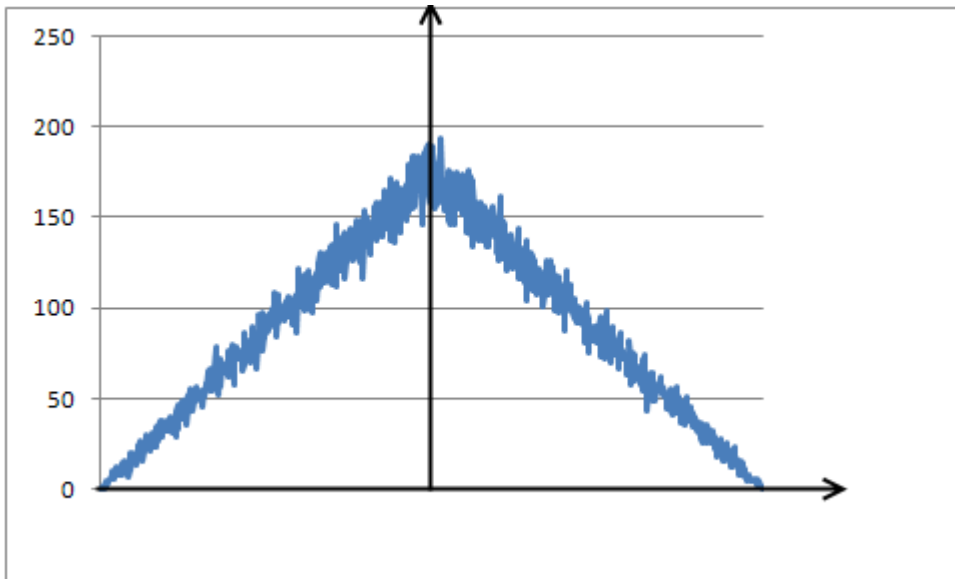


Zeichnet man eine Ausgleichsgerade so sieht man, dass diese konstant ist. Das war meiner Meinung nach nicht realistisch da es am Benzinmarkt sehr viel viel mehr kleine Preisänderungen gibt als große. Darum habe ich Funktion gesucht die in etwa so aussah:



Solch eine Funktion drückt aus, dass eine kleine Preisänderung die nahe an 0 liegt (also fast keine Veränderung) sehr viel häufiger vorkommt als eine Änderung nahe der maximalen oder minimalen Preisänderung. Die Funktion muss nur drei Bedingungen erfüllen: Sie ist zwischen der minimalen Preisänderung und der maximalen Preisänderung definiert, darf in keinem Punkt negativ sein und der Betrag der Fläche die vom Graphen eingeschlossen wird muss so groß sein wie die Simulationslaufzeit in Stunden.

Durch einen Algorithmus der den Zufallsgenerator benutzt, der, wie ich oben erwähnt habe bei einer Vielzahl von generierten Zahlen eine Gerade erzeugt, um die einzelne Funktionswerte zu verschieben (aber immer nur so, dass der Betrag der Fläche gleich der Simulationsdauer in Stunden ist) erhält man folgende Funktion:



Auf dieser Funktion basierend werden dann die Preise erzeugt. Wie genau die Preise erzeugt werden erfahren Sie hier:

1. Parameter wie *maximale Preisänderung* und *Laufzeit* (in Stunden) werden festgelegt.
2. Nun wird ein Array (*arrZufallsverteilung*) angelegt, das *maximale Preisänderung * 2* groß ist (da minimale Preisänderung = - *maximale Preisänderung* und der Index des Arrays ≥ 0 sein muss). Dieser Array wird nun mit den Werten aus der *Preisverteilungsfunktion* (die den „sauberen“ (also nicht durch Zufall verunreinigten Graphen) erzeugt) gefüllt.
3. Nun wird der Betrag der Fläche unter dem Graphen berechnet. Er kann trotz korrekter *Preisverteilungsfunktion* einen anderen Wert besitzen als *Laufzeit*, da die *Preisverteilungsfunktion* Fließkommazahlen berechnet die gerundet werden. Darum wird von *arrZufallsverteilung*[0 bis *maximale Preisänderung * 2*] solange 1 abgezogen bis das Integral mit der Laufzeit übereinstimmt. Im Prinzip ist es eine Verschiebung des Graphen in Richtung der X-Achse.
4. Nun werden *arrListZufallszahlen* (vom Typ `ArrayList<Integer>`) und *intGesamtSumme* deklariert. In zwei ineinander geschachtelten Schleifen (wobei der innere Schleifenkörper so oft ausgeführt wird wie der Betrag von *Laufzeit*) geschieht nun folgendes: Mit einer 50-50 Chance wird ein Wahrscheinlichkeitswert mit -1 multipliziert und in *arrListZufallszahlen* geschrieben. Wobei zu *intGesamtSumme* die so berechnete Zahl (also $\cdot(-1)$ oder nicht) addiert wird. Das dient dazu um festzustellen ob die Gesamtheit aller Preisänderungen eine Preissenkung über die Laufzeit darst oder einen Preisanstieg. Sollte der Preis in der Gesamtheit sinken (<0) oder einen Schwellenwert (in meinem Fall das 20-fache der maximalen Preisänderung) überschreiten wird der Punkt 4 wiederholt.

5. Nun wird immer per Zufall ein Wert von *arrListZufallszahlen* ausgewählt und in eine Zeile einer Textdatei geschrieben. Dabei wird zwischen dem Auswählen und dem Schreiben immer darauf geachtet, dass sich der Preis nicht unter *minPreis* und nicht über *maxPreis* befindet. Ist das nicht der Fall wird der Wert der in die Textdatei geschrieben wurde aus der ArrayList gelöscht.

2. Beispiele für Preisänderungsverteilungen

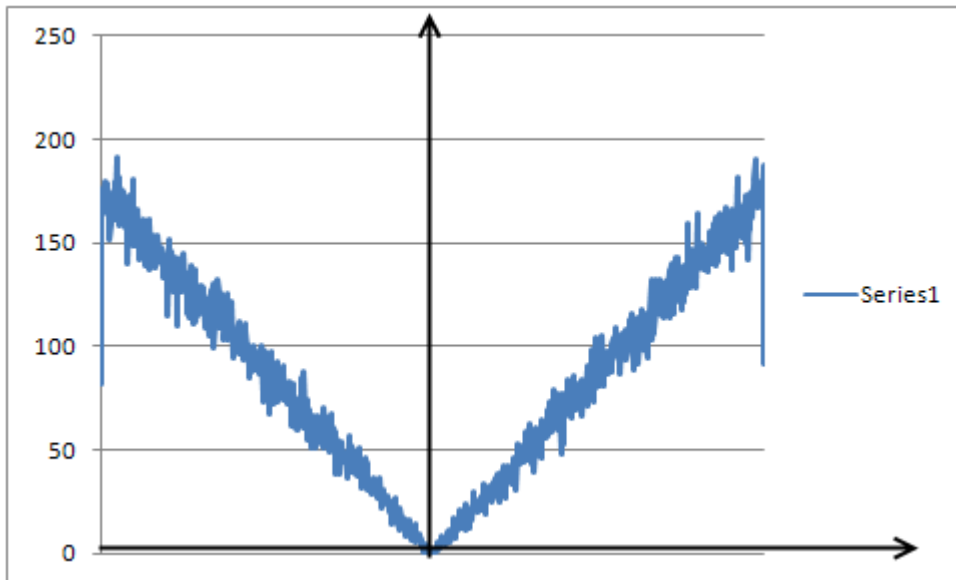


Abbildung 1 - extrem unstabiler Markt

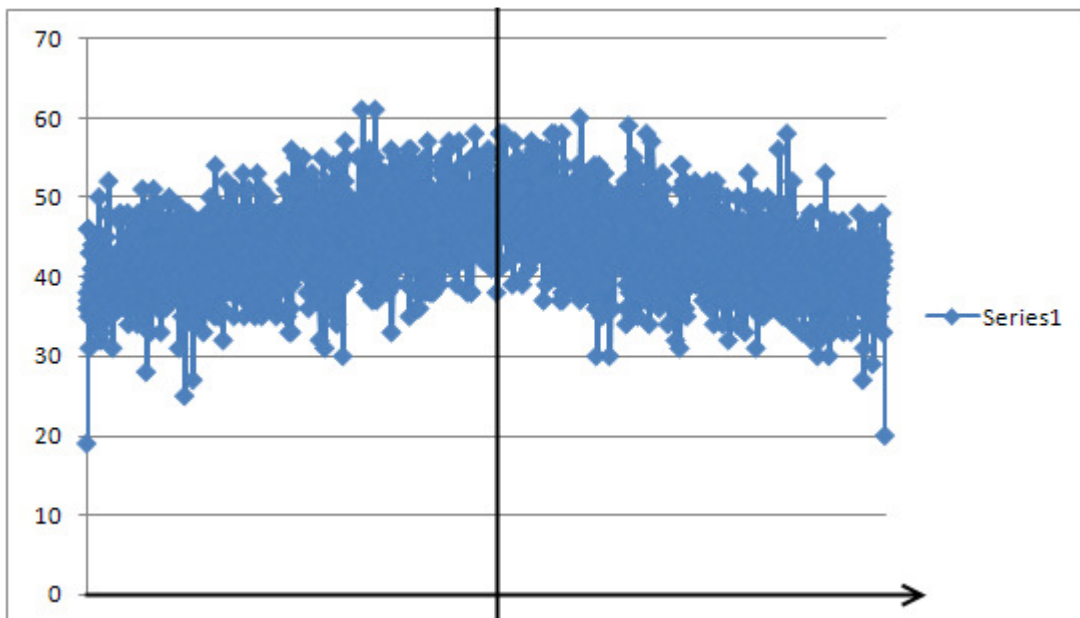


Abbildung 2 - Markt der unberechenbar und nicht zu stabil ist

3. Die eigentliche Simulation

Der Erfolg der Tankstellen wird in dem Geld gemessen das sie verdienen. Um das zu berechnen habe ich mich entschieden, im Stundentakt einen Simulationsschritt zu berechnen. Dazu gibt es mehrere Klassen:

Tankstelle – eine abstrakte Klasse die die Eigenschaften aktueller Verkaufspreis, aktueller Einkaufspreis und verdientes Geld implementiert. Dazu die get- und set-Funktionen (getAktuellenVerkaufspreis, getVerdientesGeld, setAktuellenEinkaufspreis, verkauf). Die Funktion „aktualisiereVerkaufspreis“ ist abstrakt, da die Tankstellen Asso und Scholl (so heißen auch deren Klassen) hier verschieden agieren. Die Funktion „abrechnen“ berechnet das Geld das in der letzten Stunde verdient wurde und addiert es zu *intVerdientesGeld*. Die Asso-Klasse hat noch die Möglichkeit über „setReaktionsWerte“ die Centbeträge um die der Verkaufspreis erhöht bzw. erniedrigt wird zu verändern.

Tankomat::Tankstelle
#intAktuellerVerkaufspreis #intAktuellerEinkaufspreis #intVerdientesGeld #kundenLetzteStunde
+Tankstelle(intStartpreis : int) +getAktuellenPreis : int +getVerdientesGeld() : int +setEinkaufspreis(intNeuerPreis:int) +abrechnen() +setKundenLetzteStunde(intKunden : int) +aktualisiereVerkaufspreis()

Spotmarkt stellt der Simulation einen Einkaufspreis pro Stunde zur Verfügung. Der Verlauf des Einkaufspreises kann (wie oben beschrieben) berechnet werden, abgespeichert oder aus einer Datei geladen werden.

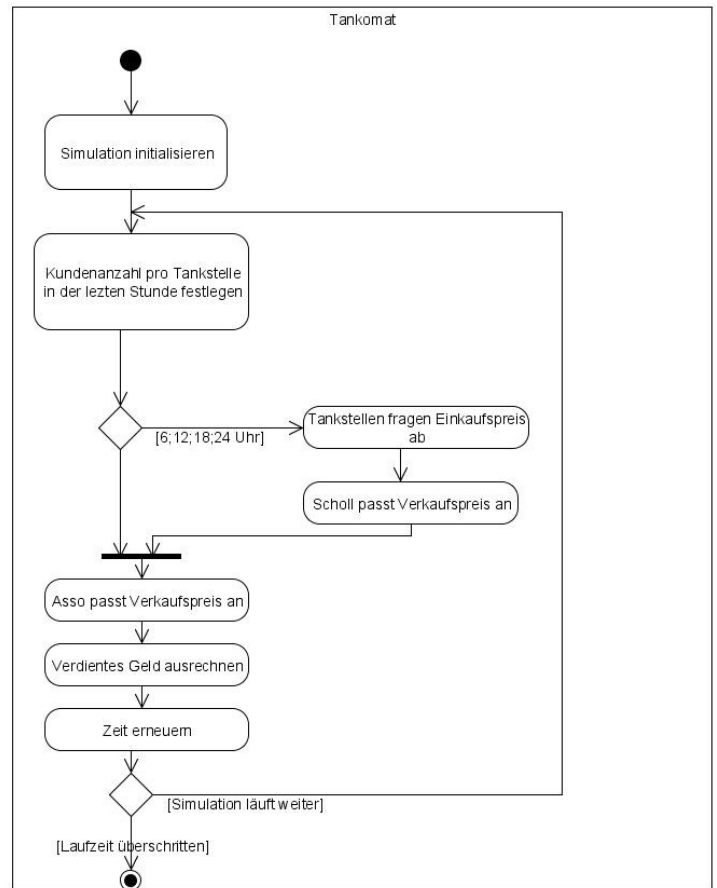
Tankomat::Spotmarkt
-myPreisentwicklung[] : int -myReader : BufferedReader
+Spotmarkt(int intAnzahlStunden, int intStartpreis) +Spotmarkt(strFile : String) +getSimulationsdauer() : int +getSpotmarktpreis(intGesamtSimulationsStunden : int) +close() +savePreise(strFile : String) -generatePreise(intAnzahlStunden : int, intStartpreis : int)

Die Funktion Simulation steuert den Ablauf der Simulation und schreibt zu jeder simulierten Stunde Ergebnisse in eine Log-Datei. Die Simulationsklasse erstellt auch Instanzen von Scholl, Asso und Spotmarkt und verwaltet außerdem die simulierte Zeit.

Tankomat::Simulation
-simulationsTagesZeit : int -simulationsTag : int -PrintWriter : logger -mySpotmarkt : Spotmarkt -myAsso : Asso -myScholl : Scholl -intSimulationsDauer : int
+TankomatSimulation(strBezeichnung : String, strSpotmarktpreisEntwicklungsDatei : String) -openStream(strFilename : String) +run() +getTagesZeit() +getTag() +getGesamtsimulationsStunden() -incZeit()

Der Ablauf der Simulation wie er durch die Aufgabe gegeben ist:

Die Simulation wird initialisiert (indem eine Instanz von Simulationsklasse erzeugt wird). Dabei wird die Quelle der Preisentwicklung bzw. die Parameter für deren Berechnung übergeben. Als nächstes wird jeder Tankstelle zugewiesen, wie viele Kunden in der letzten Stunde bei ihr getankt hatten. Durch die Aufgabenstellung ist nicht gegeben, wie sich „Wechseltanker“ (die immer zur billigeren Tankstelle gehen) verhalten, wenn die Verkaufspreise beider Tankstellen gleich sind. Da die beiden Tankstellen gegenüber liegen nehme ich an, dass von den 60 Autos die pro Stunde tanken die Hälfte auf der einen Straßenseite und die andere Hälfte auf der anderen Straßenseite tankt. Alle 6 Stunden wird durch mySpotmarkt den Tankstellen ein neuer Einkaufspreis geliefert und myScholl passt ihre Verkaufspreise an (wie in der Aufgabe beschrieben). Asso passt seine Verkaufspreise allerdings stündlich an. Anschließend wird ausgerechnet, wieviel Geld die beiden Tankstellen in der letzten Stunde verdient haben. Da es keine Informationen darüber gibt, wieviel Liter die Kunden tanken gehe ich einfachheitshalber von einem Liter aus. Die Simulationszeit wird um eine simulierte Stunde erhöht, und wenn so die Simulationslaufzeit noch nicht überschritten wurde beginnt der Prozess von vorne (nach dem Initialisieren natürlich).



Tankstellen ein neuer Einkaufspreis geliefert und myScholl passt ihre Verkaufspreise an (wie in der Aufgabe beschrieben). Asso passt seine Verkaufspreise allerdings stündlich an. Anschließend wird ausgerechnet, wieviel Geld die beiden Tankstellen in der letzten Stunde verdient haben. Da es keine Informationen darüber gibt, wieviel Liter die Kunden tanken gehe ich einfachheitshalber von einem Liter aus. Die Simulationszeit wird um eine simulierte Stunde erhöht, und wenn so die Simulationslaufzeit noch nicht überschritten wurde beginnt der Prozess von vorne (nach dem Initialisieren natürlich).

Stabil1.txt – Stabiler Markt mit Preisen zwischen 0,80€ und 1,60€

Stabil2.txt – Stabiler Markt mit Preisen zwischen 0,90€ und 1,40€

unstabil1.txt – unstabiler Markt mit Preisen zwischen 0,80€ und 1,60€

4. Bedienung des Programms und Voraussetzungen

Das Programm wurde mit JAVA6 programmiert und benötigt die Packages *java.io.** und *java.util.ArrayList<E>*. Um den Programmstart zu vereinfachen, liegt eine .bat-Datei bei („starte Tankomatik.bat“) aber die .jar-Datei kann auch mit dem Befehl „java -jar Tankomatik.jar“ in der Konsole gestartet werden. Es handelt sich um eine reine Konsolenanwendung. Startet man das Programm, so hat man die Wahl, ob man eine Preisentwicklungsdatei erstellen will oder eine Simulation starten will. Ersteres erfordert die Eingabe des Startpreises und eines Maximal- bzw. Minimalpreis, der nicht über- oder unterschritten werden darf. Dazu kommt noch eine Output-Datei in der alle Preise (87600 = für ein Jahr) gespeichert werden. Das Programm legt diese Dateien immer relativ zum

eigenen Pfad an. Ich empfehle, dass man den Dateinamen die Endung .txt anhängt. Auf der CD finden Sie (im selben Verzeichnis in dem die .jar-Datei liegt) Dateien, die die folgenden Preisentwicklungen enthalten:

Stabil1.txt – Stabiler Markt mit Preisen zwischen 0,80€ und 1,60€

Stabil2.txt – Stabiler Markt mit Preisen zwischen 0,90€ und 1,40€

unstabil1.txt – unstabiler Markt mit Preisen zwischen 0,80€ und 1,60€

Alle Preisentwicklungen haben den Startpreis von einem Euro.

Um eine Simulation zu starten braucht man einen Namen für die Simulations-Log-Datei und eine Preisentwicklungsdatei (z.B. stabil1.txt). Die Simulations-Log-Datei schreibt mit Komma getrennte Werte in eine Textdatei die man anschließend z.B. mit Excel öffnen kann. Anschließend fordert das Programm den Benutzer noch auf, die Reaktionswerte von Asso einzugeben. D.h. um wieviel Cent der Verkaufspreis rauf- oder runtergesetzt wird, wenn Asso seinen Preis anpasst (also wie aggressiv Asso auf Preisänderungen reagiert). In der Konsole sieht man aber auch welche Tankstelle am Ende der Simulation mehr Geld verdient hat.

Bei der Preisentwicklung kann es in seltenen Fällen zu Endlosschleifen kommen geben Sie dem Programm aber trotzdem mindestens 5 Minuten. Je enger ein Preisbereich gesetzt wird, desto häufiger kann es zu einer Endlosschleife kommen.

5. *Wie kann Asso erfolgreicher sein?*

Asso verdient mehr Geld (bei einem stabilen und unstabilen Markt) wenn sie stärker auf Preisänderungen reagiert: Anstelle den Preis um einen Cent nach unten zu setzen wenn weniger als 25 Kunden die letzte Stunde da waren sollte sie zwei oder mehr Cent nach unten gehen mit dem Verkaufspreis. Das hat zur Folge, dass sie pro Kunde zwar weniger verdient, dafür aber mehr Kunden hat, da sie öfters die billigere Tankstelle ist.