transentis consulting

Simulating The Beer Distribution Game

Dr. Oliver Grasl
Berlin, 17.11.2021

Contents

1. Introducing the Beer Distribution Game 4
   1.1. What makes the Beer Distribution Game so interesting? 4
   1.2. Reminder: Structure of The Beer Distribution Game Supply Chain 4
2. Designing a Stock And Flow Model 5
   2.1. The Overall Module Structure 5
   2.2. The Order Decision 8
3. Summary 9

Simulating The Beer Distribution Game

A stock and flow model of the Beer Distribution Game that can be used to analyze game dynamics

The Beer Distribution Game was originally developed in the late 1950s by Jay Forrester at MIT to introduce the concepts of dynamical systems. This post develops a stock and flow model of the Beer Distribution Game that can be used to analyze game dynamics. The model developed here is also used as a basis for our online version of the Beer Distribution Game.

#simulation #business transformation #system dynamics #advanced analytics #BPTK #Beer Distribution Game

1. Introducing the Beer Distribution Game

The Beer Distribution Game was originally developed in the late 1950s by Jay Forrester at MIT to introduce the concepts of dynamical systems. I first learned about the Beer Distribution Game at the turn of the century, through Peter Senge's lovely exposition in "The Fifth Discipline" and I have found the game quite fascinating ever since.

1.1. What makes the Beer Distribution Game so interesting?

For me, the key point is that the game has quite simple rules, yet it still illustrates many aspects of managing a complex business ecosystem. Playing the game with multiple players, either online or using a board version of the game, adds a whole bunch of "soft factors", such as making decisions under (time) pressure and (lack of) communication between players add an extra dynamic.

Here is my list of favorite Beer Distribution Game features:

- The Beer Distribution Game neatly illustrates the *dynamics of supply chains*, in particular the bullwhip effect.
- In a very tangible way, the game demonstrates how difficult it is to *manage dynamic systems*, even if you think you understand them, along with the frustration that can arise when you don't have control of all variables in the system.
- It is a wonderful illustration of *model-driven scientific thinking* - reducing a complex reality to a conceptually much simpler model and then using that model to reason about reality.
- The Beer Distribution Game is also a great object of study to see how computational modeling can help you to understand complex systems. I've always been a fan of Raymond Queneau's brilliant Exercises In Style that retells the same story in 99 different styles – our open-source GitHub repository now illustrates three different ways of building a simulation model of the Beer Distribution Game, using System Dynamics XMILE, the System Dynamics DSL and Agent-based modeling.

I have already written about the Beer Distribution Game quite extensively, in particular, my Understanding The Beer Distribution Game analyzes playing strategies in detail using a simulation model built using System Dynamics.

That post does a good job of explaining the dynamics underlying the Beer Distribution Game, but it does not actually explain the structure of the stock and flow model itself. This post fills that gap.

Note: Our GitHub repository contains Jupyter notebooks and Python code that implements the stock and flow model illustrated here.

If you have never played the game, you might like to try it before you read this post. You can access the game at beergame.transentis.com.

1.2. Reminder: Structure of The Beer Distribution Game Supply Chain

The supply chain underlying the Beer Distribution Game leads from the brewery that produces the beer, via a distributor and wholesaler to the reseller, who sells beer to his customers, the end consumers.

The sketch below illustrates the supply chain and the flow of information along it:

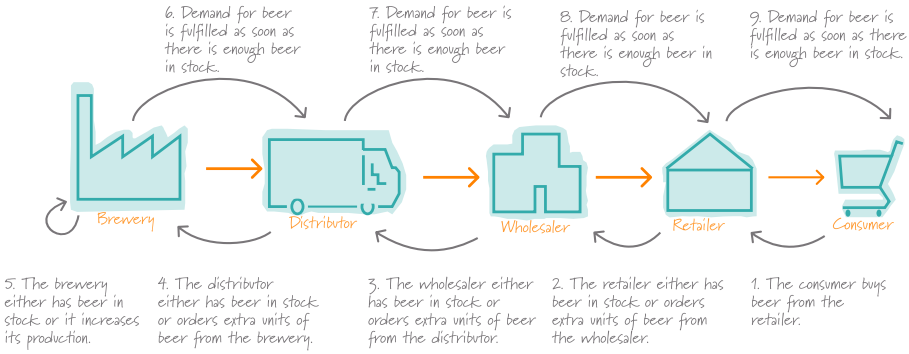

Figure 1: The Beer Supply Chain

2. Designing a Stock And Flow Model

Whenever you build a simulation model, the first thing you should try to figure out is the top-level structure of the model.

We typically refer to the high-level building blocks of a model as *module*s, and each module then contains other modules or - in the case of System Dynamics models – the actual stock and flow models.

2.1. The Overall Module Structure

The key insight when *building* a stock and flow model for the Beer Distribution Game is that the game is purposely designed to ensure that the brewery, the distributor, the wholesaler and the retailer are actually governed by the same dynamics - with one small exception: the brewery doesn't order beer from a supplier but produces the beer itself. But the beer production takes just as long as it takes to place and fulfill an order in the rest of the supply chain, so the dynamic behavior is the same.

Thanks to this insight, we can abstract this common behavior into a single structure, which we can then either reuse or replicate when building the actual simulation model.

Do we need any other modules? Yes, we do:

- The whole point of having a simulation model is that we want to monitor the performance of each of the players and the supply chain itself. Because we aren't just looking at individual players but also at the overall performance it makes sense to extract the monitoring of the supply chain into a separate part of the model.
- To ensure all parts of our supply chain use the same decision policies and settings for e.g the price of beer and the time it takes to order or produce beer, we extract the settings into a separate module.

This leaves us with the model structure shown in the diagram below.

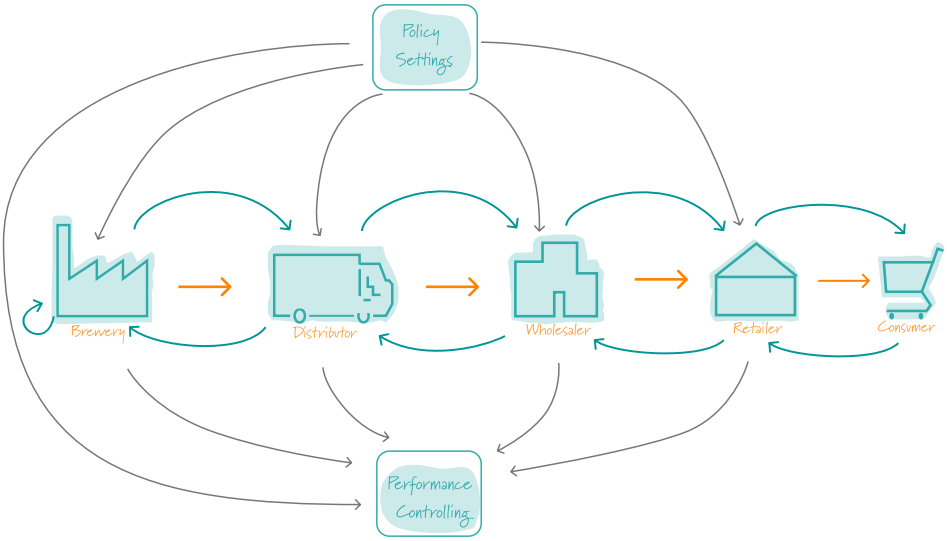

Figure 2: Beer Distribution Game Stock And Flow Blog Post Image 1 EN

Now that we know the overall model structure, let's take a look at the stocks and flows we need to model the supply chain modules.

Each module in the supply chain has a supplier and a customer. Let's pretend to be a player in the supply chain and let's walk through the supply chain management process from that perspective:

1. First of all, your customer calls you and makes an order. You note this down in a list called `Orders received`.
2. You then check your `Inventory` to see whether you can fulfill the customer's request.
3. You give your customer whatever you can and note that down in your `Deliveries made` list.
4. You make an order to your supplier, to cover the beer your customer ordered. To keep track of the orders you make you keep a list called `Open orders`.

That defines the stocks that you need to keep track of what is going on in your part of the supply chain.

Note that the beer itself can be in one of three states: it is either on order (ie. still with your supplier or on the way), it can be in your inventory, or you have made the delivery and it is on the way to your customer (or has already been received by the customer).

We can model this as a chain of stocks and flows, as shown in the diagram below.

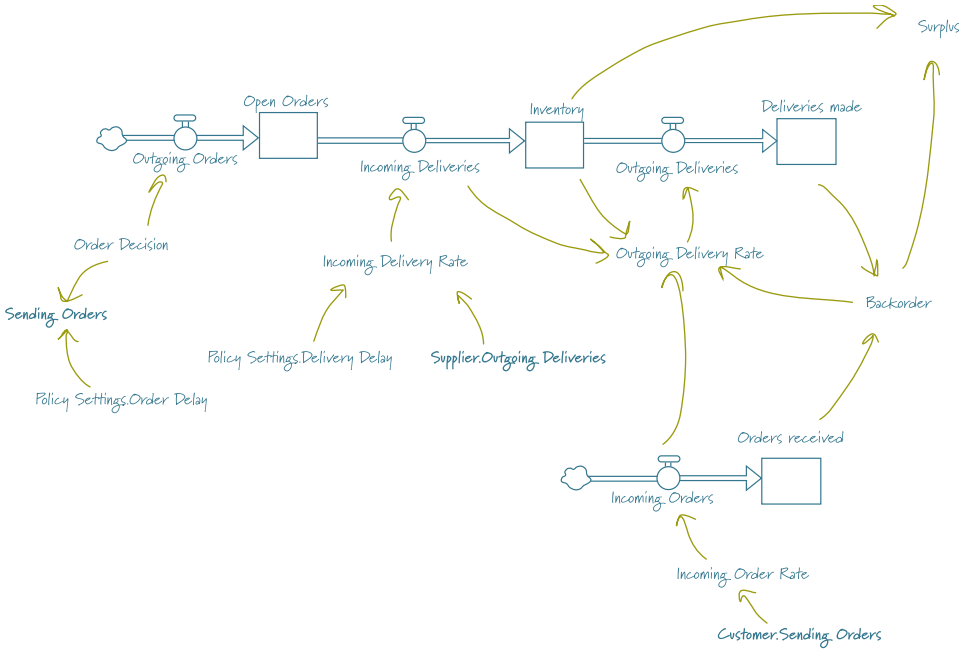

Figure 3: Beer Distribution Game Stock And Flow Blog Post Image 2 EN

The diagram also shows two indicators that we can use to measure the performance of our supply chain: the *backorder* and the *surplus*.

The *backorder* is simply the difference between orders received and deliveries made. In the best case the backorder is equal to 0, i.e. you can always fulfill your customer's wishes:

```
Backorder = Orders_received - Deliveries_made
```

The *surplus* is the difference between the inventory and the backorder. The surplus measures how much beer you have to cover incoming customer requests. It should always be a positive number, but not too big, to keep inventory costs down.

```
Surplus = Inventory - Backorder
```

Now that we have the stocks and the indicators, we can take a look at the flows and associated rates.

Let's start with the *incoming orders*. This is equal to the *incoming order rate*, which itself is just equal to the order being sent by your customer:

```
Incoming_Orders = Incoming_Order_Rate = Customer.Sending_Orders
```

And now let's walk through the beer supply chain:

The number of outgoing orders is equal to the orders you make.

```
Outgoing_Orders = Order_Decision
```

Of course, the key question in the entire Beer Distribution Game is how you make that order decision - because it is so important we'll defer the detailed discussion of the order decision to the next section.

The incoming deliveries depend on the incoming delivery rate, and this in turn is equal to the outgoing deliveries being sent by your supplier, delayed by the amount of time it takes for the goods to reach you.

In our game, this delay is equal to one week, in the model the delay is governed by the delivery delay setting in the policy settings. This leads to the following equation:

```
Incoming_Deliveries = Incoming_Delivery_Rate =
Delay(Supplier.outgoing_deliveries,Policy_Settings.Delivery_Delay,100.0)
```

The only flow remaining is the *outgoing deliveries*, which is equal to the `outgoing_delivery_rate`. Here the policy is quite simple: we always want to deliver as much as we can, i.e. the sum of the backorder and the currently incoming order. But we can't deliver more than what is currently the inventory and what is just arriving. This leads to the following equations:

```
Outgoing_Deliveries= Outgoing_Delivery_Rate =
Min(Backorder+Incoming_orders,Inventory+Incoming_Deliveries)
```

2.2. The Order Decision

Now let's come back to the order decision. In our model, we have implemented two order decisions, a "naive" one that ignores backorders and the supply line and a "sophisticated" one that takes them into account. The different effects of this decision are discussed at length in my blog post - so here I will just give you a brief summary.

In the naive order decision, you essentially order everything the customer is ordering right now, plus the entire backorder, plus the difference between your current inventory and your target inventory:

```
Naive_Order_Decision =  Backorder + Incoming_Order_Rate +
Policy_Settings.Target_Inventory - Inventory
```

A more sophisticated order decision ignores the backorder but includes the supply line (i.e. the open orders). It also avoids sending shocks up the supply line and adjusts inventory slowly. This leads to the following equation:

```
Sophisticated_Order_Decision = Incoming_Order_Rate +
(Policy_Settings.Target_Inventory - Inventory + Target_Supply_Line - Open_Orders)/
Policy_Settings.Inventory_Adjustment_Time
```

The target supply line needs to account both for the expected order and for the delivery delays:

```
Target_Supply_Line = Incoming_Order_Rate *(Policy_Settings.Delivery_Delay +
Policy_Settings.Order_Delay)
```

The diagram sums up the factors that are relevant for the order decision:

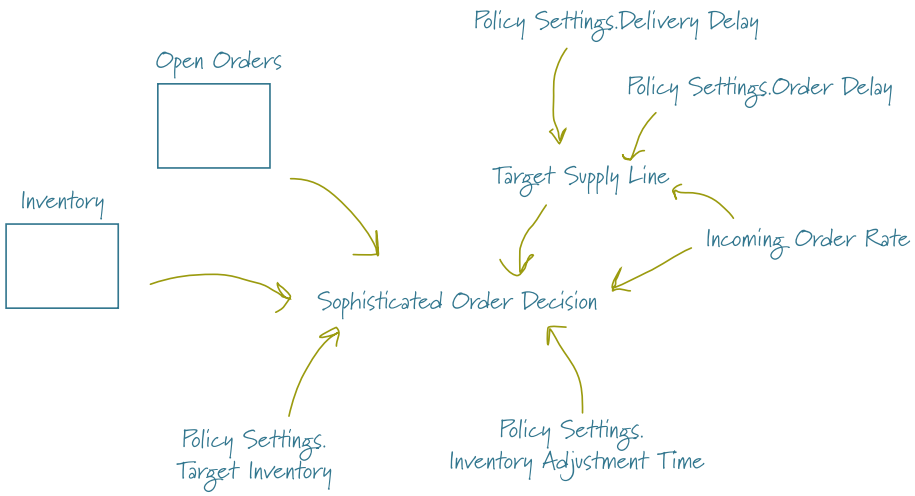

Figure 4: Beer Distribution Game Stock And Flow Structure image 3 EN

Here is a little dashboard for you to test the effect of the order decision on the supply chain.

Embedded content: https://app.metapad.ai/embed/16cf74a7-e642-4277-b22a-8b5576bca422/eaadcfee-480b-4d66-a961-42965de61205

3. Summary

You can find two implementations of the stock and flow model illustrated here in ourGitHub repository- one implementation in Python using the BPTK SD DSL and one implementation as a Stella model.

The SD DSL version is discussed in detail in the companion notebook to this blog post.

© www.transentis.com, 1997-2026 transentis consulting GmbH & Co. KG