Capitalism 9.0

What is this?

Capitalism 9.0 is an app which shows how a capitalist economy functions, provides a testbed for different theories about how this happens, and allows users to learn about it and study it.

Where do I find out more?

The user guide

The file 'User Guide.pdf' walks you through the project, and at the same time teaches you how to use it, and what it's for.

Announcements

The file 'announcements.md' contains updates on what's been done recently and the current state of the project. It's worth checking, especially to see if there are any surprises, whether nasty or otherwise.

The Documents folder

The folder 'docs' contains material relating to the project - articles, spreadsheets, and a Word version of the user guide. It will help you understand the theory behind the app.

The TODO file

The file 'todo.md' in the docs folder is a more techie guide to what's going on, what the bugs are, what's been fixed, and so on. You don't have to read it unless you want to.

How do I run it?

The application runs from a single Windows executable called 'capitalism.exe'. See 'Installation' below.

What's it for?

Simulations have at least four purposes. They can be used to educate, to help us understand real life, and test hypotheses about the real world by verifying if they are at least possible and capable of yielding the results we expect. In principle they can also be used to attempt predictions about what will happen in real life, but this idea is much-abused in economics, whose predictions are generally speaking inaccurate in the most important sense of this word, namely, what actually happens is not what

was predicted. This is especially so for 'qualitative' predictions, of the type 'inequality will shrink' or 'there will be no financial crash'.

This simulation is designed mainly for the first three purposes. A real economy is more complex, not so much because of factors that cannot be taken into account by a simulation, but because there are a very large number of agents and a very large variety of commodities and ways of producing them.

This doesn't rule out using the simulation in the future to make predictions, though I suspect that to be usable, the number of entities would increase so much that a strategic redesign would be called for; the issue is however in this case quantity, not quality, because the methods would be the same as in this simple version. The main problem is thus to get the basic rules right. Most economic theories do not, because they rely on the concept of 'General Equilibrium' which, in a nutshell, means they suppose that the economy reproduces itself perfectly: it produces everything it needs, it consumes everything it produces, and so on. This leads to some important qualitative errors; not the least is that it assumes the market is perfect, so that crisis becomes impossible. Another critical weakness, related to the first, is that money plays no role in these theories.

A proper dynamic simulation, which this attempts to be, makes no such assumption so it is much more general, allowing is to incorporate money and finance, and study the possible mechanisms of crisis. In particular, it allows that goods can be in short supply, or that there can be an excess of them, that prices can depart quite far from values, and so on. It breaks in particular with what economists know as 'Say's Law' which supposes that whatever is produced must automatically be sold. This is tantamount to equilibrium, though the proof is beyond the scope of this document. The key point is that in a Say's Law economy, profits must always be invested in new production. Actually, this does not happen because, as both Marx and Keynes recognised, nay insisted, the capitalists can, and do, hoard profits in the form of money, or monetary instruments, instead of re-investing them. This is why the **qualitative** phenomena of financial crisis and depression cannot be 'simulated' in a Says's Law/Equilibrium model.

For this simulation, no assumption is made that profits are re-invested. To the contrary, the economy *reacts* to disparities between supply and demand. For example, if something is in short supply, its price will likely rise. In consequence higher profits can be made by producing it, and so capital tends to grow faster in the sector, or among the producers, that make it. This increases the supply, and the price falls.

The reaction may be more complicated; for example the state may intervene, or people may substitute, eating cake instead of bread.

An early (1991, see below) simulation proved that an economy governed by the simple rules above 'works' in the sense that it does not simply collapse, but runs for some considerable time, albeit with fluctuations - just like a real economy - and with longterm trends that undermine it - just like a real economy.

The simulation allows us to test the *reaction rules* that make this possible. It also, in principle, allows a community of game-players to experiment and play with different rules, just as with Sim-City. You can think of it as Sim-Capitalism.

At the time of this version, we have not yet provided reaction rules; we are at the stage of what I call 'flight-testing' the basic activities of allocation, trade, production and distribution, by which I mean ensuring that the basic **invariants of motion** of a dynamic economy are conformed to. When this is verified, the reaction rules will be introduced.

Nevertheless the simulation is free of Say's Law assumptions in the following, precise sense: we suppose that the capitalists attempt to maintain production at the existing level (that is, to carry on as before) but we make no assumption about what happens if they fail, or what happens to that part of the surplus-value that is not consumed by simple reproduction or capitalist consumption.

Where did it come from?

Beginning with the world economic crash of 1974, economists became increasingly preoccupied with the evidence, today very stark, that something seriously wrong was affecting the world economy. Attention was drawn to the explanations offered by writers such as Marx but also Keynes in his original writings, to the effect that these problems arose from within the economy itself; they were the result of an inbuilt or structural instability which, the most radical critics concluded, could only be set right, if at all, if the state played a far more central role in the economy, especially as regards investment. Naturally, this was a controversial conclusion. Those who supported the free market did much to avoid drawing it; however, those who opposed the free market were also divided, because they could not agree on whether the state could in fact solve the problems of a free market or (to be more precise) capitalist economy, or whether this would need to be replaced by some entirely different system such as socialism.

These controversies came to a head in the 1970s but in a way that paralysed theoretical understanding. At their centre was an approach, both to the economy and to Marx's own ideas, known as 'simultaneism' which held that the economy should be understood as the outcome of a 'perfect market'; a system in which all produced goods were sold, and in which all savings were invested. The mathematics of these systems was not only wrong, but daunting, involving quite advanced linear (matrix) algebra. Mathematically, simultaneism is the formal expression of a Say's Law economy. It takes market perfection as its premise.

An alternative approach known as 'temporalism' emerged (see the http://www.copejournal.com). This rejected the assumption that the economy reproduced perfectly, which meant that economic magnitudes were understood as the solution to a set of simultaneous equations, having the same values at the beginning and the end of an economic period. Temporalists argued that economic magnitudes, especially values and prices, changed from the beginning of a period to the end of it. The economy should therefore be analysed as a **succession** of periods, and the requirement of a sound analytic method was to uncover and explain the laws of this succession by showing how the magnitudes at the end of a period arose from the *initial* magnitudes at the beginning of the period, and the processes of *production* (making things and services) and *social reproduction* (making or maintaining social classes, legal and property relations, political institutions and cultural relations).

Such an approach requires a different kind of mathematics, though one that is very familiar to engineers and physicists, namely the mathematics of difference and differential equations. Since society is very complicated, it is hard to write down a simple set of equations describing this motion, although it is possible to identify *invariants of motion* governed by quite simple laws, such as the law that value arises only in production.

Temporal scholars made much use of spreadsheets which are well-suited to this kind of analysis, but these are limited because they cannot display, or visualise, the more complex kinds of information needed to understand what is going on in a society with many different products and several classes. The author of this programme, and a couple of others, turned to the method of *simulation*; a computer programme that would both carry out the necessary calculations, and visualise them in a simple way so that ordinary, non-technically-specialised users could follow what was going on.

The original was written in 1992 in C++. It demonstrated, on the basis of some simple temporal assumptions, that a capitalist economy can reproduce itself through the movement of capital in response to price movements, relative to values, arising from the interaction of supply and demand. It thus accounts for the effect of price deviations within a value-theoretic framework. The simulation also exhibited cycles with a period approximately equal to the turnover time of fixed capital, as suggested by Marx.

It had theoretical limitations, the most important being that it used what scholars (see http://www.copejournal.com) call a 'dual-system' framework, the MELT being absent from the theorization of the price-value relation.

At various times, I attempted to correct this but at the same time updating the visual presentation of results to make use of the evolution of GUI(windows, browsers, etc). Meanwhile the underlying theory has progressed significantly (see, for example, the papers on https://geopoliticaleconomy.academia.edu/AlanFreeman

So the simulation hasn't caught up with the theory. This is an attempt to put that right.

The chief obstacle, for many years, was constructing a visual ('GUI') interface so that users could easily see the results. Technology was still primitive, and the work was very time-consuming. After several attempts I constructed a 'Vaadin'-based version which can be found in the https://github.com/axfreeman/capitalism.

Most of this involved a lot of learning, especially since the project was constructed from the get-go as a server-based web project, introducing many unnecessary complications. The learning experience can usually be forgotten (don't we all just want the prize at the end of the game) However, in this case, understanding the learning process is itself, I surmise, part of the prize.

The new version has become a venture in its own right. The technical issues addressed go beyond constructing a visual interface and take us into the realm of logical analysis. The arduous task of writing a coherent, generalisable simulation confronts the writer with the need to scrutinise, in minute detail,

every single prejudice or assumption that can be skated over by writing down general formulas or citing texts. A simulation is the ultimate test of a theory; it forces the writer to discover whether the theory can even exist. Importantly, this task makes the assumptions explicit, in that they have to be embedded in computer code that is fully public and can be modified by anyone. The 'pieces' have to fit, and have to implement the rules of encapsulation, abstraction and inheritance that define the framework of every modern application and thereby form the basis of the social production of mental objects.

The visual interface itself takes advantage of the newly emerging (http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm) framework. This makes it possible to develop the simulation as a desktop application - which it was, originally. It can be later ported to the web/Android.

This version also uses a database standard known as JPA (Java Persistence). This allows the coder (me) abstract from the myriad database platforms in existence. All queries are written in JPQL, a standard query language that is part of the JPA standard. The original version kept all its variables in memory, which allowed it to run at a reasonable speed and made it easier to write. The new version keeps all its variables in a database, which means the results can be kept to study later and to share with others. Critically, this also means that when the application is eventually ported to the web, users can share their experiments and ideas by putting the database also on the web.

Finally the new version makes use of the learning I acquired in the first project which I hope will lead to a clean and portable project that can be developed by a community

I am continually learning, and both the standards and technology of the Java community are advancing at a formidable rate. I'll be writing about this under the heading of 'the production of mental objects' in a forthcoming book on the creative industries. Meantime, any comments on coding standards from experienced coders, that can lead to improvement and in particular, standardisation, are welcome.

What does the simulation do?

When it starts up, you see a screen with a number of tables in it, and various other controls. These are all described in the user guide.

The tables represent the following objects:

1. *use-values*, which are produced and consumed. Each use-value has a unit value and a unit price. The simulation also calculates, at various times, the total supply and demand for each type of use value along with the total quantity (which can be bigger than the supply, since it includes stocks that aren't available for sale, such as money, and the total value and total price of this total quantity.

2. *circuits* which are of two types, productive (capitalist) and social. A capital circuit is a producer, who consumes use-values including *labour-power* and produces other use-values. A social circuit is a class of people defined by what they own and what it entitles them to. Workers own labour-power

and are paid wages, capitalists own productive circuits and are paid profits. Bankers own money, which is not to say nobody else does; what is special about bankers is that they are paid interest. This is just a general guideline; simulations are perfectly possible in which workers receive interest, or run businesses on the side. They only real restriction is that labour-power is only supplied by workers.

3. *stocks* which are quantities of use-values. Each circuit has a stock of use-values for sale, which make up the supply of that use-value. Each circuit also has 'productive' stocks - supplies of the use-values it needs in order to produce. Both capitalists and workers have stocks of consumption goods.

As the simulation progresses, three processes take place

1. the capitalist circuits produce, adding to their sales stock and using up their productive stocks
2. the social circuits reproduce, using up their consumption goods and keeping their class alive
3. the various suppliers sell what they make and buy what they need

The simulation keeps track of: 1. the quantities of use value that are produced, consumed, and traded 2. the *values* of these use values which are calculated as the sum of the value consumed and the labour time used to create them (Marx's 'Constant capital' and living labour) 3. the *prices* of these use values, which are calculated by the rules of the simulation. These rules, in the intended final version, can be supplied by the user. In the original version, they were simply calculated as being higher or lower depending on the disparity between supply and demand. 4. the money used to buy and sell the use-values

The total price and the total value are related by a magnitude called the MELT (Monetary Equivalent of Labour Time) which at any given time is the same throughout the economy. That's why, when the capitalists buy stocks with which to produce, we can calculate their value in terms of labour time, even though money is used to buy them.

That's it! Nothing more complicated happens, unless rules are introduced to make it so. The real purpose of the simulation is to follow all of this through, and see what happens to the stocks, their values, their prices, and the growth or shrinkage of the circuits.

The user can make the simulation 'play' by pressing the action buttons that pass through an entire circuit of reproduction (Marx's C-M-C-P...C'-M'). As this proceeds, more and more records are added to the database, each defined by a 'timestamp' saying when it happened, and which stage the circuit had reached when the record was added. The user can then *visualise* what's going on by studying the tables. The user can also study the process by selecting a previous timeStamp, which allows them to go back and study the past to see how things have changed.

There are then various gizmos that help this visualization, such as a log window that records the changes as they happen, and sub-windows that let you study particular circuits in more detail.

Installation

1. The easiest way is to download the windows executable file 'capitalism.exe' from the same site as this document. Download it, and run it as you would any windows programme. Your virus checker may take a peek at it, will declare it OK, then it will let you run it.
2. The programme will create a sub-folder in your 'Documents' folder called 'Capsim', short for Capitalism Simulation. It will create two subfolders. One contains the data files that the simulation uses;the other contains a set of 'log' files which record the progress of the simulation. You don't have to study either of these but they are designed so that they can be understood, relatively easily, without requiring extensive technical knowledge. You can also view the most important log file, which is a kind of commentary on the simulation, from the application itself by pressing the button marked 'log'.

3. If this works and you are satisfied, stop here. If it doesn't work, tell me. If it works but you are not satisfied, volunteer.

4. If you want more control, or if option (1) didn't work, and/or you are a glutton for punishment, download 'capitalism.jar' which is in the '\bin' folder, and put it somewhere. Then (the difficult bit)

5. download and install the Java Runtime Environment (JRE) from https://java.com/en/download/. Follow the instructions on that page. It's easiest if you create an environment variable so that your computer knows where to find the files you just downloaded. I only know how to do this in Windows.

6. Press 'start' and type 'environment'; you get an option to 'edit the system environment variables'. You should see a dialog box with a button labelled 'environment variables'.

7. Press it. You should get a new dialog box with two panes. In the pane labelled 'system...' choose the variable 'PATH'. Edit this, adding the path to the place you installed JRE.

8. Open a 'Command' (DOS) window in the directory where you put 'capitalism.jar'.

9. Type 'java -jar capitalism.jar'

10. If you want to try and add your own data: you can't yet but this feature is a high priority. Suggestions, accompanied by cheques, bankers' drafts, cryptocurrency equivalents, or EFT for $1000 or more, are always welcome. But actually, labour is more important than money so offers to do it yourself will attract an automatic waiver. See (4)

11. Finally if you really want to get involved, compile the application, for which you need to know the language Java and the programming environment.

Technical notes

Don't read any further unless you are comfortable with steps 3 and 4 above.Most technical details are

detailed in the 'todo.md' file but occasionally I'll report big changes here.

Tech installation notes

1. You need the Java Runtime environment (8 or later) and the latest JDK
2. If you want to edit or compile the project, it works with the Eclipse IDE, the Eclipselink Plugins, and the JavaFX plugin. I haven't tried the project in other IDEs and I haven't tried a command-line build. So I'm not saying this is the only way to build it, but it's the only one I have tested. I haven't constructed a Maven or Ant build or any of the other standard things one is supposed to do as a Java programmer. I just ploughed ahead until it worked. One day I may go back and fix all this, but there's no real reason to, if someone else can do it.

How does it work?

The circuits, stocks and usevalues are all in databases. I use a database called H2 which is 'embedded' in the application, that is it requires no attention from the user. At present the database is re-initialised every time the application is run, but eventually, the idea is that it will be preserved from one use to the next, so that you can start where you left off. I also envisage that users can create 'branches' called project versions, and switch between them at any time. Finally, if the application becomes web-based, users will be able to share simulations with each other.

Every table has a special column called 'timeStamp'. Every time the simulation does something, a new timestamp gets created describing what was done, the data is modified, and a new copy of every item in the simulation is written back to the database, with the timeStamp moved on by one step. A description of what was done is written to the 'TimeStamp' table. For example 'register supply' changes the column 'total supply' associated with each use value, by totting up what is available for sale from everyone who makes that use value. So we register a new record for each of these use values. But we also register a new record for everything in the system - every stock, every circuit. Thus, with each action that is undertaken, the entire state of the system is recorded with a new timeStamp.

This may seem like overkill, but it's way less complicated than anything else and, with modern database technology and a small number of entities, it runs at an acceptable speed. It could become more difficult if there are a larger number of entities, but it doesn't matter that we are creating many records (of a small number of entities) because these days, storage is cheap.

legal and license stuff

Copyright (C) Alan Freeman 2017-2019

flaticons license