Introduction to Threadneedle

Jacky Mallett

Chapter 1

Introduction

Threadneedle is an economic simulation platform which has been designed to support agent based research on modern monetary systems using fractional reserve banking. It is centered on a full emulation of fractional reserve banking, implemented using double entry book keeping for all its banking operations.

Threadneedle provides a simulation environment in which individual economic agents interact with each other through monetary transactions. A base set of agents: individuals, companies, banks, markets and Governments is supplied with the framework, and these are supported within an infrastructure which provides access to banking services for all agents. The base set of agents provided can be used to build simple economies, and they can also be extended to provide additional behaviours as researchers require.

Agents in Threadneedle interact with each other solely through monetary signals provided by market pricing, taxation, lending and its associated interest rates, and the flow of money between individual agents as transactions occur. Within the constraints of fractional reserve banking, the regulatory structure surrounding the banking system can be configured to allow the exploration of different regulatory requirements. Central bank reserve and Basel capital reserve regulatory frameworks are fully supported, but full reserve and free banking systems systems can also be created by imposing restrictions on the operations of the banks within the simulation.

Threadneedle (version 1.0) is a research tool under active development, and is presented as such here. Some familiarity with Java and command line utilities is necessary in order to install the package. Simple economic experiments can be performed with the set of base agents included, but most researchers will want to enhance and extend these to provide richer simulations. Those who wish to create new agents for the model or change behaviours will need some programming experience with Java, and a programming guide is provided to assist with this. (See the Programmers Guide to Threadneedle for more information).

Threadneedle is provided free for non-commercial research purposes under the creative commons Attribution-NonCommercial 3.0 License. Questions about licensing, comments, bug reports, suggestions, feature requests and/or code contributions, should be sent to the author, Jacky Mallett(jacky@ru.is).

Chapter 2

Quick Start

2.1 Recommended Directory structure

Install the Threadneedle class and java files in a top level directory called Threadneedle. The CLASSPATH environment variable for Java will need to be set to include this directory and the classes directory. For example (linux, bash shell) if Threadneedle is in directory ~/src/Threadneedle:

```
CLASSPATH=\$CLASSPATH:~/src/Threadneedle:~/src/Threadneedle/classes:
export CLASSPATH
```

All the examples in this document are based on an installation using the following file structure:

```
src/
 |__Threadneedle/
 |   |__classes/
 |   |__Documentation/
 |   |__lib/
 |   |__src/
 |   |__ledgers/
 |__output/
 |__configs/
```

where the Threadneedle/ directory contains source and compiled classes, output/ contains results from batch runs of the program, and the configs directory contains the configuration and batch files for the researcher's experiments with Threadneedle.

2.2 Installation

Source code should be cloned from the Threadneedle github repository at:

https://github.com/jackymallett/Threadneedle

If you are not familiar with git and github, you can find a getting started guide at `https://git-scm.com/book/en/v2/Getting-Started-The-Command-Line`. The following instructions are for the command line interface.

From the command line interface, with git installed, the full command to install Threadneedle is:

```
git clone https://github.com/jackymallett/Threadneedle
```

Threadneedle is currently released under the Creative Commons Attributive Non-Commercial license.

Third Party Packages

The following third party packages are used by Threadneedle, and are provided in the Threadneedle/lib directory.

csvreader http://opencsv.sourceforge.net/

2.3 Compiling Threadneedle

To compile, change directory into the Threadneedle directory and use the *build* shell script:

Linux/OSX ~./build

Windows ~.\build

The build script will automatically launch Threadneedle if the build is successful.

2.4 Running Threadneedle

To run Threadneedle without compiling it, use the supplied *run* script from the command line.

Linux/OSX ./run

Windows .\run

Threadneedle can also be run directly from the command line using the following java incantation:

```
java -cp "classes:src/resources:lib/*:" gui.Threadneedle
```

Threadneedle provides the following command line options:

-cl	Run command line interface
-b	Specify batch file with commands to run
RNG seed	Replace the default random number seed

2.5 User Interface

The graphical interface[1] provides a simple ability to visualize some models and can be used to stop and start the simulation, single step, and to run it for a pre-determined number of steps.

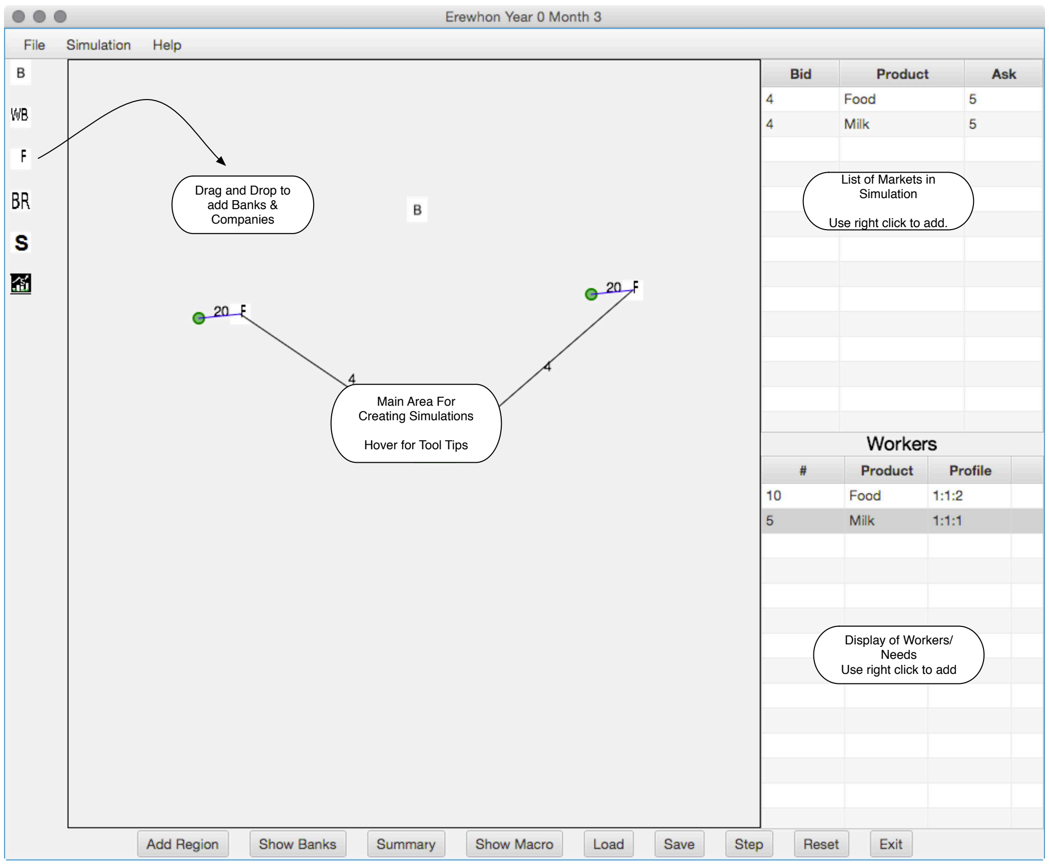
Threadneedle also provides a command line interface (see below for a list of commands) which can be used in conjunction with the graphical simulation, or as a series of commands in a batch file which can be executed without the GUI. Large models should be run without the GUI as it does slow down simulation considerably.

[1] At time of writing (2015), every expense has been spared on the results display for Threadneedle, in favour of developing a robust simulation engine. Patience is requested with some of the infelicities in the current UI.

Chapter 3

Configuration

3.1 Modelling



Bid	Product	Ask
4	Food	5
4	Milk	5

#	Product	Profile
10	Food	1:1:2
5	Milk	1:1:1

Figure 3.1: Threadneedle Main Window

A Threadneedle model is a description of an economic simulation to be by the Threadneedle framework. It is an entity based set of configuration descriptions which describes each individual accounting entity (agent) in the model. This description is then evaluated over time, according to the behaviours encapsulated by the java code of each 'agent' and the regulatory and tax framework set for the simulation. Economic parameters such as taxation, base interest rates, etc. can be adjusted at any point during a simulation, and agents can also be added if required.

Each step in Threadneedle corresponds to a single day in a 30 days per month, 360 day year - this is a simplification made primarily to ease simulation analysis and debugging. (Extension to properly support the calender year may be considered in the future.) The title bar for the main screen displays the notional number of years, months and days so far into the simulation. The Step button will run the simulation by one or more steps at at time (rotate the button using scroll to select more steps), and the Run and Stop buttons can be used to run the simulation continuously.

Each simulation run is controlled by a configuration file or files. Configuration files are stored as editable json format files. In practice, small modifications can be made manually to a configuration file, and slightly larger modifications can be made using tools such as sed, but this is not the recommended way of creating simulations. The graphical interface allows configurations to be created, modified and inspected. Configurations can be saved and loaded into the interface. Multiple configurations can be loaded into the simulation from different files, provided they are consistent with each other. This provides a method of building partial configurations that can be used as building blocks when creating multiple simulations.

Default agents include banks, markets, producers, stock markets, workers and investors. Agents can also be added to the simulation by extending existing agents (See Programming Guide.). Several different financial instruments are supported including compound, simple and Icelandic indexed linked loans, Treasuries (government debt), securitized bank loans and company shares. Loans can be arbitrarily specified with respect to their duration, and the frequency with which payments are made.

Start

The main start screen is shown in Figure 3.1. Agents can be added to the simulation by dragging and dropping from the left hand bar, under agent specific menus on the right hand side, or directly from the command line interface (see below). Tooltips are available by hovering over items on the screen.

All simulations must first define at least one bank, as all agents in the simulation are required to have a bank account. Bank accounts will be automatically created for agents when they are added to the simulation, at the bank specified for the agent. When agents are being dragged and dropped onto the main screen, the agent's account will be created at the bank physically closest to the agent.

Once at least one bank is present, individual agents can be added to the simulation as required. There are some dependencies between agents, in particular producers need a market to sell their goods on, and producers and workers need markets to buy goods from. These can be modified later, but the recommended order to create agents for the simulation to avoid this is:

1. Banks (At least 1 is required)
2. Markets
3. Producers
4. Workers

Markets are either added automatically on drag and drop if an agent being added to the simulation requires one, or they can be added manually by right clicking on the top right hand side menu and using the Add Market dialog. Note, there are two places where workers can be added to the simulation, the bottom right

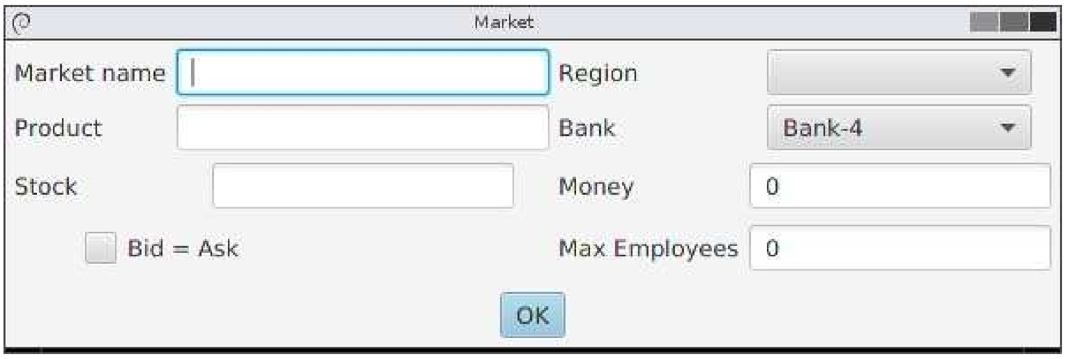


Figure 3.2: Add Market

hand "Workers" section which creates workers who can be hired from the labour market, and consume goods, and specialized financial workers through the bank handling screens (See Banking below).

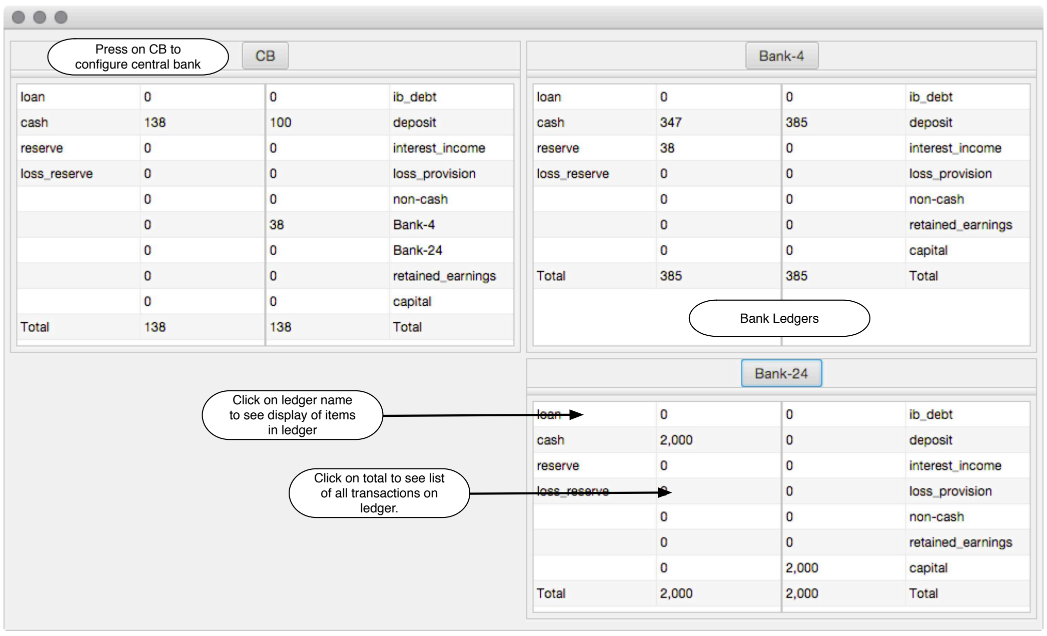
Clicking on agents in the main display will bring up agent specific menus that allow their individual attributes to be modified.

Banking

A Central Bank will be automatically created with the Government, which will also hold its account there. Banks will use their reserve accounts at the central bank for clearing operations.[1]

Show Banks

Once Banks have been added to the simulation, their ledgers can be displayed using the "Show Banks" button at the bottom of the screen. Ledgers are shown for each bank. Clicking on the bank's name will


Press on CB to configure central bank
CB
Bank-4
Bank Ledgers
Bank-24
Click on ledger name to see display of items in ledger
Click on total to see list of all transactions on ledger.

loan	0	0	ib_debt
cash	138	100	deposit
reserve	0	0	interest_income
loss_reserve	0	0	loss_provision
	0	0	non-cash
	0	38	Bank-4
	0	0	Bank-24
	0	0	retained_earnings
	0	0	capital
Total	138	138	Total

loan	0	0	ib_debt
cash	347	385	deposit
reserve	38	0	interest_income
loss_reserve	0	0	loss_provision
	0	0	non-cash
	0	0	retained_earnings
	0	0	capital
Total	385	385	Total

loan	0	0	ib_debt
cash	2,000	0	deposit
reserve	0	0	interest_income
loss_reserve	0	0	loss_provision
	0	0	non-cash
	0	0	retained_earnings
	0	2,000	capital
Total	2,000	2,000	Total

Figure 3.3: Show Banks

bring up a dialog specific to that bank allowing interest rates, capital amounts etc. to be controlled, and account holders to be added.

Clicking on the name of an individual bank's ledger will bring up a summary menu for the accounts or loans held by that ledger where this is appropriate. Selecting the amount shown for the ledger will bring up the list of double entry book keeping transactions that have been made to the ledger.

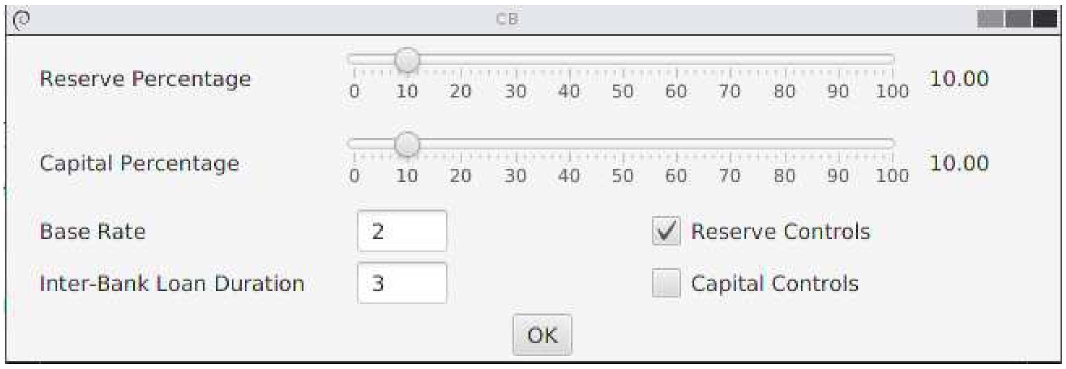
All changes take effect in the step they are made. Changes can be made during the simulation, for example to view the impact of interest rate changes on a given simulation banking system.

Central Bank Configuration

Selecting the central bank will show the dialog shown in Figure 3.4. This dialog allows the reserve percentage and capital percentages for the banks controlled by that central bank to be modified, and a base interest

[1]The exact mechanisms by which banks perform clearing is country and region specific, and has also varied considerably over time. Today computerized Real Time Gross Settlement(RTGS) systems exist in most countries which typically use domestic banks accounts at their central bank for clearing. Separate systems can exist, major European inter-bank clearing operations are performed using the TARGET2 system, and clearing for small US banks can be done via local federal reserve banks. There are potential simulation effecting liquidity considerations depending on exactly how this is performed - for example historically it depended on physical exchanges, requiring several days to process, and required higher reserves limits than typical today, simply to handle the implicit buffering.

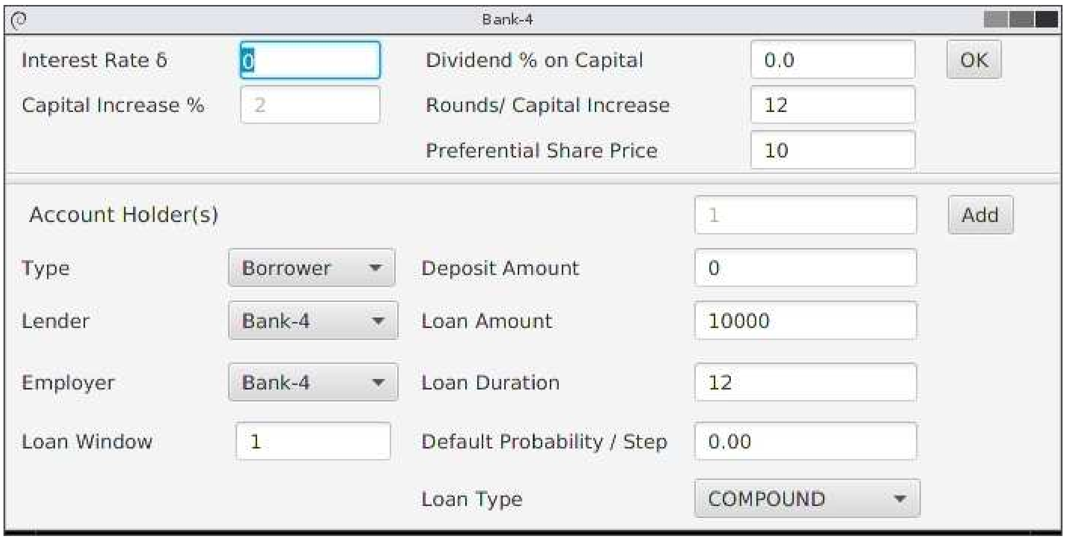
rate to be set for the simulation. Reserve and capital controls can be enabled and disabled at will to see the impact on the simulation, or to explore simulations without reserve controls, etc. The duration of an inter-bank loan (no. of steps) can also be controlled. Changes to central bank parameters take immediate effect within the model currently being simulated.



Reserve Percentage 10.00
Capital Percentage 10.00
Base Rate 2
Inter-Bank Loan Duration 3
[✓] Reserve Controls
[ ] Capital Controls

Figure 3.4: Central Bank Configuration

Comercial Bank Configuration


Bank-4

Interest Rate δ: 0
Capital Increase %: 2
Dividend % on Capital: 0.0
Rounds/ Capital Increase: 12
Preferential Share Price: 10
OK

Account Holder(s): 1
Add
Type: Borrower
Deposit Amount: 0
Lender: Bank-4
Loan Amount: 10000
Employer: Bank-4
Loan Duration: 12
Loan Window: 1
Default Probability / Step: 0.00
Loan Type: COMPOUND

Figure 3.5: Individual Bank Configuration

Figure 3.3 shows the bank configuration dialog which can be obtained by clicking on the individual bank's name. This allows different types of financial agent to be added to the simulation, and provides control over loan default rates, capital growth, and dividend payments for each bank individually. The following types of account holder are currently supported:

Borrower Borrowers are a special type of financial agent that provide a way to explore the banking system, decoupled from the rest of the economy. Borrowers attempt to take out loans for the amounts/periods/types specified by the dialog, and then repay them. Borrowers are treated as bank employees, in order to short circuit monetary flows from interest payments, and receive a salary from the bank that is sufficient to allow them to meet their capital and interest payments, *if, and only if* the bank can afford to make these payments from its income.

Which Bank employs the borrower can be set using the configuration dialog. Note that if borrowers are defined as receiving employment from one bank, and a loan from a different one, then an asymmetric flow of asset money will occur between the two banks. This can be used to exercise the interbank lending mechanisms, and explore related long term liquidity issues.

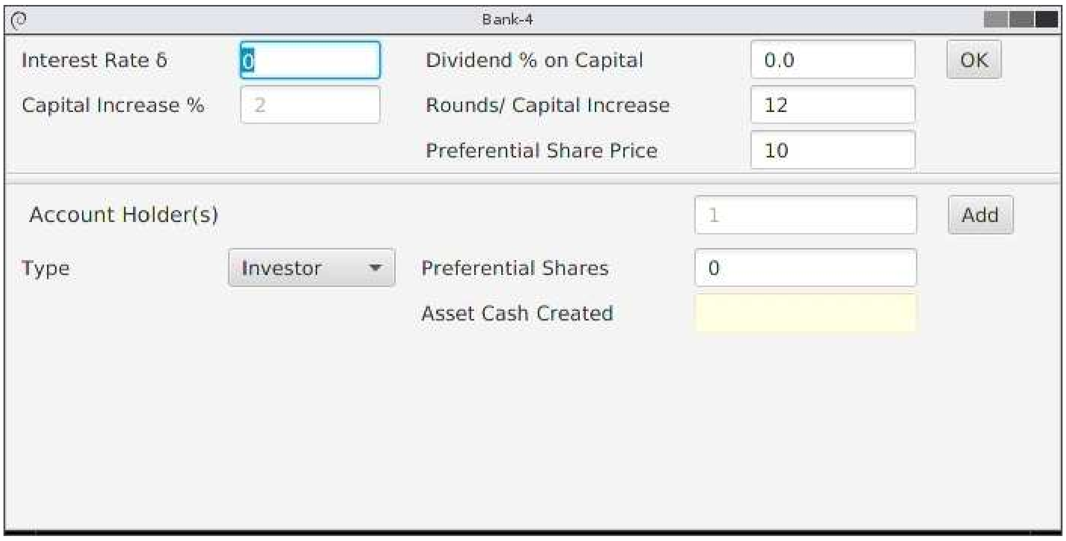
Investor Investors provide an initial amount of asset cash, and receive preferential shares which pay the dividend on capital specified in the dialog for the bank.

Saver Savers provide asset cash, which is matched by a deposit at the bank where they hold their account. They perform no other actions under simulation - their main purpose is to provide initial liquidity, and to facilitate experiments with 'stagnant' deposit money accounts in the banking system. No interest is paid on these accounts.

Loan Types

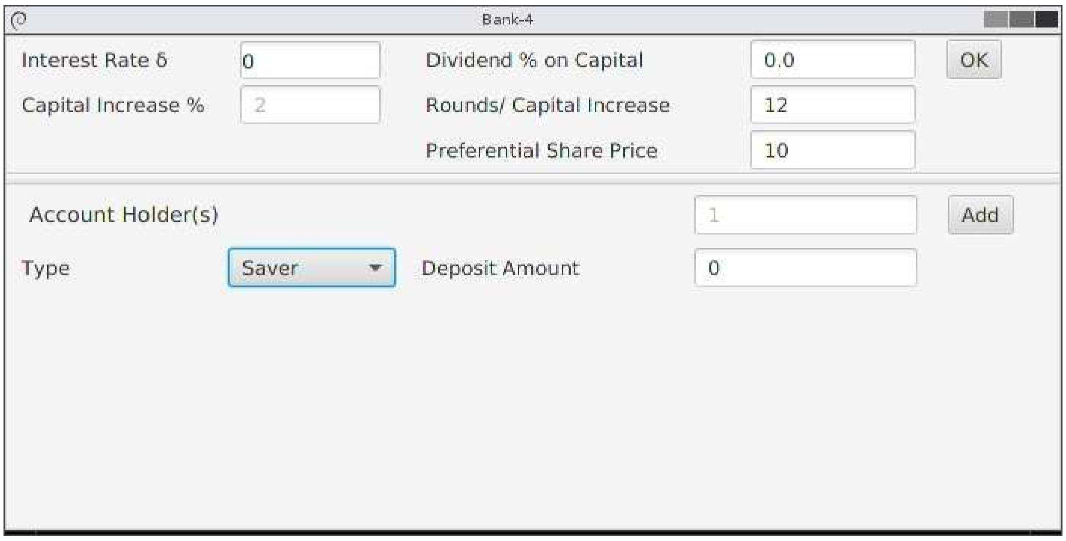
Individual Banks are currently restricted to issuing a single type of loan, which is specified on configuration. The following loan types are supported:

SIMPLE	Fixed rate, simple interest loan
COMPOUND	Fixed rate, compound interest loan
ICELANDIC	Icelandic indexed linked loan



Interest Rate δ	0	Dividend % on Capital	0.0
Capital Increase %	2	Rounds/ Capital Increase	12
		Preferential Share Price	10
Account Holder(s)			1
Type	Investor	Preferential Shares	0
		Asset Cash Created	

Figure 3.6: Account Holder:Investor



Interest Rate δ	0	Dividend % on Capital	0.0
Capital Increase %	2	Rounds/ Capital Increase	12
		Preferential Share Price	10
Account Holder(s)			1
Type	Saver	Deposit Amount	0

Figure 3.7: Account Holder:Saver

Charts

A variety of charts are available to show the state of the simulation. The charts being shown can be configured for each simulation by clicking on the Charts display and selecting as appropriate. Additional charts and statistics for display here can be added to the simulation if desired (See Programming Guide).

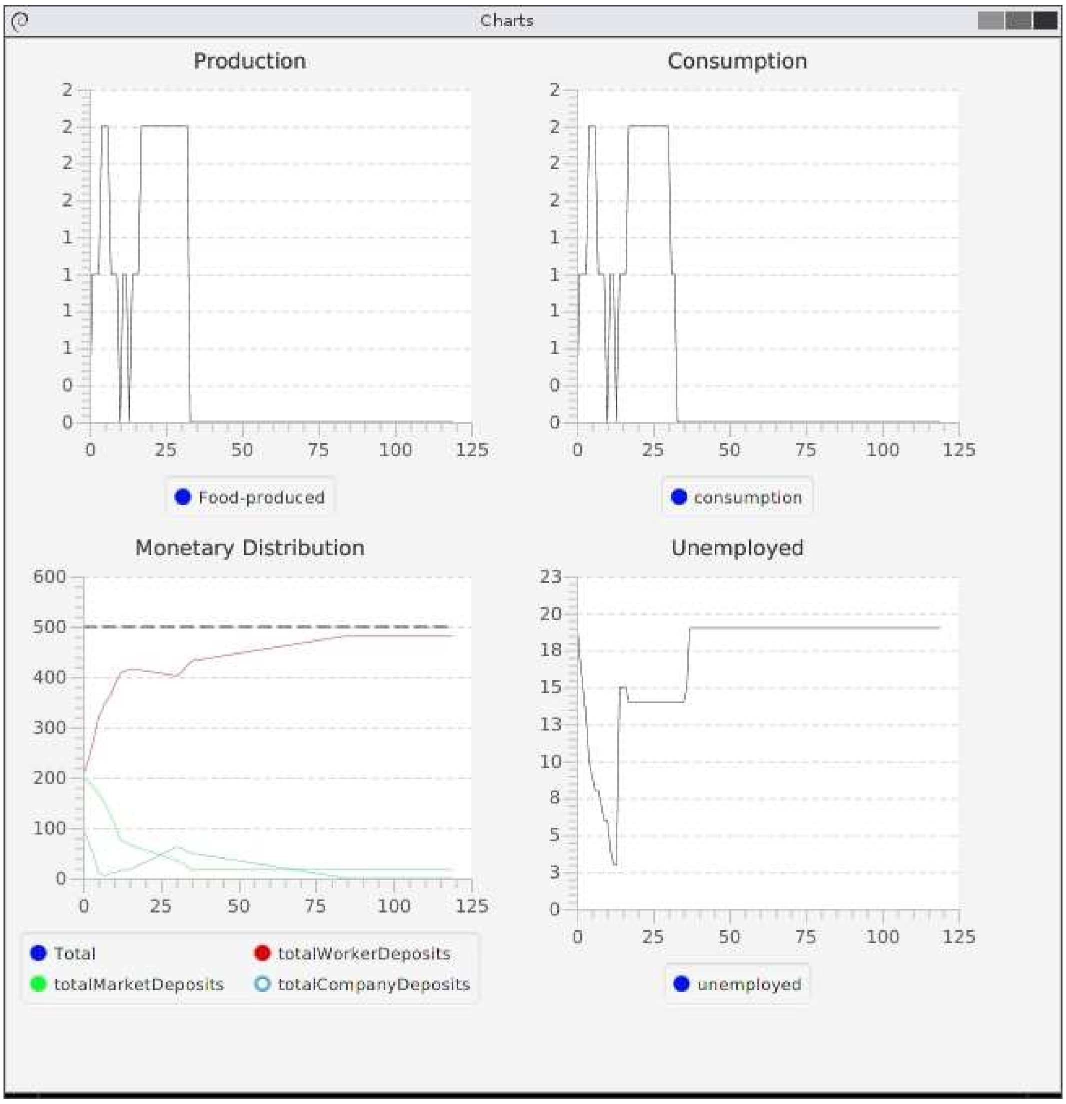


Figure 3.8: Simulation Charts

Chapter 4

Command Line Interface

Threadneedle also provides a command line interface which can be used to directly run and modify the active simulation, as well as interact with batch mode. To run with a command line, use the option "--cl" on startup, i.e.

```
./build -cl

./run -cl
```

help

Print a summary of available commands.

```
> help
Commands: help
step [n]                    : Step simulation n steps (default 1200)
steps [n]                   : Step simulation n steps while updating charts
repeat <times> <command> : repeat a command multiple times
forall <regex pattern> <command> : provide simple for loop capability
wait [time]                 : wait a while before continuing reading commands
reset                       : reset simulation
set                         : set parameters in simulation
load <file>                 : load new config file
preferences <filename>      : load Threadneedle parameters from file
config                      : show current parameters for simulation
statistics                  : show statistics registered with simulation

printmoney agent x      : increase agent's deposit by x
addagent <type> <bankname> [options] : add an agent with [options] as properties key=valu
agentinfo <agent>       : invoke toString() on <agent>
debug [file]            : toggle debugging information on and off
agentinfo <agent>       : invoke toString() on <agent>
htmlcharts [dir]        : write charts out in html/png format
report [file]           : print report on simulation optionally to file
setbaserate             : change the base interest rate
shareholders <stockmarket> : print the shareholders of the given stockmarket
savechartdata [dir]         : write chart's series data out to a file for that chart as pn

savechartcsvdata [dir] : write chart's series data out to a file for that chart as csv
printorders <agent>     : print the orders listed on stockmarket <agent> or put up by <age
quit                : exit simulation
```

report [file]

Print a report of the simulation results, optionally to a named file.

htmlcharts directory

Write charts specified for the simulation to the specified directory in png format. Directory will be created if it does not already exist, and existing charts files will be overwritten.

htmlsetup directory

Write a simplified report of the simulations configuration to a file called $test_config$ in the specified directory. This will be automatically displayed by the Threadneedle.php page if this has been setup.

addagent

Add an agent to the model. An initial deposit must be specified, other parameters for the agent can be passed in as strings. For example:

```
addagent <type> <bankname> initialDeposit=X [options]
eg.
addagent Person Bank-4 initialDeposit=50
addagent Farm   Bank-4 initialDeposit=100 labourInput=3
```

addneed/addwant

```
addneed agent-id name purchase consume store consumable useLoan
eg.
addneed Person-60 Food 1 1 1 true false
```

Associates a need or a want with a specified agent.

set

Set parameters in the active simulation's configuration. Note, these will not affect the simulation's behaviour until the simulation is reset using the reset command and run from the beginning. The format of the set command is:

```
set <agent name> <parameter> <value>
eg.
set A0 initialDeposit 20
```

forall

The forall command provides a limited looping capability that can be used in conjunction with other commands. The format of the command is:

```
forall <regex pattern> <command>
e.g.
forall ^A set initialDeposit 15
```

will set all the initial deposit for all agents whose name begin with A to 15.

printmoney

The printmoney command can be used while the simulation is running to increase the money supply by increasing an agent's deposit at the bank. At present this only works in constant money supply simulations (those using the simple Bank class), since double entry book keeping prevents agent's deposits being arbitrarily increased within a fractional reserve framework without side effects. The format command is:

```
printmoney <agent name> <amount>
e.g.
printmoney A0 300
```

reset

Reset the simulation state to that of the configured parameters. This will cause any changes made by the set command to take effect.

step

Step the simulation a specified number of steps.

quit

Exit the simulation.

4.1 Command line interface and Batch Mode

Threadneedle provides a batch mode which allows experiments to be run from either the command line, batch command files or a combination of both. Graphs and results showing the results of these experiments can be saved to a user controlled directory for later viewing, as well as being displayed during the batch run.

Connection between batch and config files

A batch file provides a series of commands that are executed sequentially on the simulation described by the configuration file Threadneedle is invoked with. There is no connection between the batch file and the configuration file beyond the requirement that the commands in the batch file can be applied to the agents specified in the configuration file.

The commands available from a batch file are the same as those available from the Threadneedle command line, and can be seen by using the "help" command on the Threadneedle command line.

Comments

In both the configuration file and the batch file, a # symbol at the beginning of the line can be used to add comments to the file

Running in batch mode

The command is:

```
java -classpath "classes:src/resources:lib/*:" gui.Threadneedle –b=<filename>
```

Optionally the "–charts" option can be supplied to view the simulation's chart information without displaying the main simulation screen. An RNG seed can also be specified.

4.2 Web Integration

To facilitate viewing the results from multiple experiments a *node.js* server is provided which allows a web browser to easily view the results of experiments.

The chartserver.js file can be found in the ../Threadneedle directory. By default the server uses the contents of the *output* directory in the directory it is run from, and the results can viewed using a webbrowser from the url `127.0.0.1:1234`. To run the node.js server use the command:

```
node chartserver.js
```

Reloading the webpage pointing at `127.0.0.1:1234` will then cause the server to automatically cycle between the charts in the market1 and market2 output directories.

Chart images can be created in batch (or from the command line) by using the htmlsetup and htmlcharts commands. For example, the following batch file will load the market1.json configuration file, run it for 100 steps, save the charts to output/market1 and then run for another 100 steps before saving the charts to output/market2.

```
load configs/market1.json
debug off
step 100
htmlsetup output/market1/
htmlcharts output/market1/
step 100
htmlsetup output/market2/
htmlcharts output/market2/
```

Chapter 5

Known Problems