Agent-based Macroeconomics Simulation via Object-Oriented Programming

Deniz Durmus
Sarajevo School of Science and Technology
Sarajevo, Bosnia-Herzegovina
International University of Sarajevo
Sarajevo, Bosnia-Herzegovina
deniz.durmus@stu.ssst.edu.ba
ddurmus@ius.edu.ba

ABSTRACT

Agent-based models rely on simulation to discover the properties of macroeconomics in the presence of heterogeneity and interaction. A robust and reliable simulation framework is a prerequisite for computational research in this field. In order to have the maximum freedom in constructing our future models we have chosen to develop our own agent-based simulation framework. In this paper, we give the details of code development along with the testing process. We have chosen to use C++ as our programming language so that we may achieve the best performance. In our case computational efficiency is of importance, since we plan to use our framework in many different scenarios and plan to make a large number of investigative runs.

CCS CONCEPTS

• **Computing methodologies** → **Agent / discrete models**; **Modeling and simulation**; • **Applied computing** → **Economics**; • **General and reference** → *Empirical studies.*

KEYWORDS

Agent-based Simulation, Macroeconomics, Object-Oriented Programming, Complexity

ACM Reference Format:
Deniz Durmus. 2023. Agent-based Macroeconomics Simulation via Object-Oriented Programming. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The development of agent-based modeling (ABM) and object-oriented programming (OOP) are contemporaneous, while each has benefited from the development of other (see [Wilensky and Rand 2015]). Specifically, the facilities chosen to be included in C++ were influenced by intended target applications. Simulation in general and ABM in particular were such application areas.

OOP is convenient for ABM. ABM agents are objects that contain (encapsulate) their individual data (their state) and their operations (their behavior or functionality). OOP develops classes, where instances of classes have their own data and functionality encapsulated within the class. ABM relies on agents, each with its own specific data and functionality. The collection of all data and functionality in an object (call it a package, or a bag) is referred to as **encapsulation** in the OOP paradigm. Similarly, instances of objects may have different functionality. This is supported in OOP, and is referred to as **polymorphism**. Polymorphism is once again a key feature of ABM, where agents may have differing choices, behaviors, and preferences. Among the OOP trilogy of determining characteristics, we also have **inheritance**. Inheritance means offspring objects could be derived from parent objects, making the content of the parent object available to the offspring. C++ has perhaps the most liberal approach to inheritance, supporting multiple inheritance (offspring objects receiving contents of multiple parent objects). While encapsulation and polymorphism play key roles in ABM, inheritance is mostly effective as a software tool to make the code better organized and more flexible and more extensible.

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 THE MODEL

In developing the simulation framework, we started with a "proof of concept" model that is rich enough to test its operation. The simulation given here is a simplified version of [Riccetti et al. 2015]. Once the testing and simulation phase of the software is successfully completed, we will extend the model within the established framework to address and accommodate the hypotheses to be pursued in our doctoral research.

The initial model contains three types of agents (banks, firms, and workers) in three markets (the credit market, the labor market, and the goods market). The simulation tracks all agents properties over time such as net worth, employment status, taxes, consumption, etc. The workers are also the consumers in the model.

Table 1: Interacting agents in each market

Market	Demand Side	Supply Side
Credit Market	Firms	Banks
Labor Market	Firms	Workers (Consumers)
Goods Market	Consumers (Workers)	Firms

3 SOFTWARE OVERVIEW

The software is developed as a multi-module project using the professional grade GNU C++ compiler and the GNU tool-chain. Each type of agent is implemented as a separate C++ class. The application takes full advantage of the "object-oriented" paradigm.

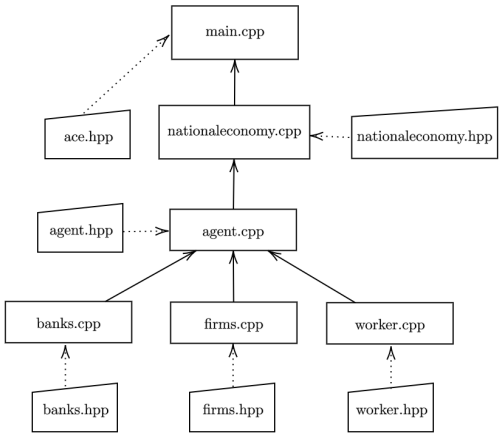

Figure 1: Software hierarchy

Figure 1 shows the general overview of the software. This is not a software specification that gives an Application Programming Interface (API), but rather a general understanding of the structure, the motivation, and design choices of the software. Files with extension ".cpp" along with their corresponding ".hpp" header files are the modules of our software. While header files are used to store the attributes of each module and list function prototypes (the syntax of the functions), ".cpp" files give function implementations.

All classes share some common design aspects. Each class has private data members, encapsulating and shielding the details. For data members that must interact with external classes, there are "get" and "set" member functions. External classes ask for member data using enumerated-type identifiers. All requests may be routed through the container class "NationalEconomy". For example, agents (e.g. banks) may issue a "get" call to NationalEconomy using the enumerated-type identifier CENTRAL_BANK_INTEREST_RATE. This practice is strictly followed so that as the software becomes more complicated, a strict inter-agent interaction discipline is established. This follows good software engineering practices to facilitate maintainability, and more importantly, extensibility. Moreover, it allows each attribute to be stored in a single unique place, with no duplications. In terms of notation, we use capitalized names for member functions other than the generic "get" and "set" prefixes. Data members are indicated by a preceding "m_" to make it clear that the variable is a member of a class.

In Figure 1, at the top, we have main.cpp where all the support functions (e.g. random number generation), initializations, and interactions in the three markets are carried out. An extensive flow of main.cpp can be viewed from this link. The top header file ace.hpp defines some structures, gives prototypes of utility functions, and includes application constants, such as the default number of banks. For example, we use a structure to keep (encapsulate) information regarding loans.

```
struct Loan{
double fAmount;
double fInterestRate;
};
```

The names of the member elements of structures and classes are rather explicit to aid in the development of "self-documenting code", a software practice that is strictly followed throughout the application. A good example of a utility function, globally defined and ubiquitously used is the uniform random variable generator. Its prototype is included in ace.hpp as below.

```
// global utility functions (prototypes)
double UniformRV(void);
```

The global function UniformRV() returns a pseudo random variable in the range of $[0, 1]$.

4 SOFTWARE TESTING

Code with 1024 firms, 1024 banks, 1024 workers, and 1024 time periods was taken to be a logical upper bound of parameters. Although our code runs with larger numbers, the 1024 (1 Kilobyte) limit was deemed practical, since this is the maximum number of columns common spreadsheet programs support. The code with these maximum numbers runs in 63 seconds, about the same time it takes interpreted languages to run small-sized models. The 63 seconds includes the time it takes to write the output to a 1.2GB file. It takes considerably longer to open the file in a spreadsheet program.

The code for 400 periods runs in 116 milliseconds, that is about a tenth of a second. It should be noted that a similar size code takes about a minute to run on interpreted languages such as Java, Python, or on interpreted software such as MatLab and R.

Table 2: Code Modules

Module	Line Count
ace.hpp	65
agent.cpp	36
agent.hpp	27
bank.cpp	241
bank.hpp	62
firm.cpp	482
firm.hpp	79
main.cpp	507
nationaleconomy.cpp	479
nationaleconomy.hpp	93
worker.cpp	198
worker.hpp	57

Table 3: Execution Times

Number of Firms	Number of Banks	Number of Workers	Run Time	Execution Time (seconds)	Output Log File Size
10	10	10	10	0	111.4 KB
100	100	100	100	1.41	11.1 MB
40	10	200	100	1.74	8.2 MB
40	10	400	100	36.63	13.8 MB

REFERENCES

Luca Riccetti, Alberto Russo, and Mauro Gallegati. 2015. "An agent based decentralized matching macroeconomic model." *Journal of Economic Interaction and Coordination*, 10, 2, 305–332. Publisher: Springer.

Uri Wilensky and William Rand. 2015. "Why Agent-Based Modeling." en. In: *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. The MIT Press, Cambridge, Massachusetts. ISBN: 978-0-262-73189-8.