UNIVERSITATEA DIN PITEȘTI
FACULTATEA DE ȘTIINȚE, EDUCAȚIE FIZICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE MASTERAT: TEHNICI AVANSATE PENTRU PRELUCRAREA INFORMAȚIEI

LUCRARE DE DISERTAȚIE

Coordonator științific,
Conf.univ.dr. Costel BĂLCĂU

Absolvent,
Mădălin Grigore-Enescu

Pitești
2020

UNIVERSITATEA DIN PITEȘTI

FACULTATEA DE ȘTIINȚE, EDUCAȚIE FIZICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE MASTERAT: TEHNICI AVANSATE PENTRU PRELUCRAREA INFORMAȚIEI

Computational models in economics

Coordonator științific,
Conf.univ.dr. Costel BĂLCĂU

Absolvent,
Mădălin Grigore-Enescu

Pitești
2020

Contents

Introduction

As of April 2020, around 4.57 billion people were actively using the internet and this number roughly equals 59 percent of the Earth's current population (J. Clement, Jun 4, 2020).

Among these users, a significant percentage of people is involved in various leisure activities including but not limited to: mobile or web applications, online games and social networks. All of these currently support the acquisition of virtual goods or virtual rewards, using either virtual currencies or real currencies.

There is no denying that virtual economies are no longer a concept from Sci-Fi novels, but they are part of most of the current global population's day-to-day, normal life.

Perhaps some of the most revolutionary fields in the area of virtual economies are the newly emerging game economies. Compared to other applications that are somehow more linked to the real economy by delivering real services or products to the users, game virtual economies are independent from the real economy in several ways. The rules of the game economies, the products and the services involved are, at their core, pure virtual concepts with little relation to the real economies.

Having this freedom from the real world economies, game economies currently implement the most creative and complex systems that are able to simulate various virtual economies, that manipulate goods which do not even exist or are less possible to exist such as forcefields (in a space strategy game), or magic scrolls with an enchanting poetry that is supposed to render the user with immunity to vampires. All those goods are sold in abundance, generating virtual profit in virtual worlds that defy the limitations of real life and real economies.

Moreover, the degree of manipulation and automated control that an entity can enact on a virtual economy is endless. The entity that controls the virtual economy has endless power to collect accurate data about the economy state and to intervene almost instantaneously into the virtual economy processes, a god-like power that no government today possesses.

But, like with any form of freedom, all this power comes with great responsibility. The game companies today invest a lot of efforts into creating virtual economies that make their users happy and prevent them from running away to another game, much in the way emigrants are searching for better jobs across the borders when they are not happy with their country's economic state.

The happiness of the users is strongly related to the health of the virtual economy itself. Like in real-life situations where all the people want to live in an economy that offers them enough founds for buying fuel for cars, for example, the players like to have enough anti-matter for their ships to fly. On the other hand, just like with any good, the acquisition of that good should be difficult enough to offer the user the drive for working to acquire it, which translates in being active into the virtual economy and constantly interacting with other players.

Those fine lines are quite hard to maintain even when the game authors are having full control on the virtual economies they create. Also, there is an increasing demand into researching what are the best rules and measures you can implement into a virtual economy to make the players happy.

The aim of this paper is to research into such measures and try to understand how traditional economic systems (i.e. the real economies) could affect the general state of a virtual economy. For this purpose, we created an application that simulates virtual economies. The application is built upon three different types of economies, approximately resembling the socialist economic system, the capitalist economic system and something in between.

Please note that the simulation we created is not intended in any way to be used for determining which system is better for real economies; also, the outcome of our simulation is highly affected by the software implementation details and approximation of the real world economy actors and rules that we use into the simulation.

The simulation should only be used as a theoretical research direction into how those different types of economies can influence a full virtual economy. The outcome of the simulation can, in theory, help entities controlling virtual economies to switch between various systems in order to moderate the game economy when statistics are showing the current state of economy is going into the wrong direction for the players.

We believe that in a pure virtual economy all strategies studied here can be combined or even replaced for various periods of time depending on the state of the virtual economy.

The first chapter of this work aims at making a brief overview of the computational models in economics with accent on how they can be used with virtual game economies.

The second chapter will detail the principal mathematical concepts and formulas that are implemented into our virtual economy simulator.

The third chapter will detail the virtual simulator software architecture, the technology stack used for implementing the simulator and various information about the GUI of the simulator and how the simulator can be used.

The last chapter will contain the outcome of the simulator running with various parameters and a few conclusions we can extract from the data produced by the simulator.

I. Computational models in economics

I.1 Computational economics

Computational models in economics belong to a larger discipline called "computational economics". This discipline encompasses and interfaces many other fields of study including but not limited to: computer science, economics, management science, law, sociology and philosophy.

In the area of computer science, computational economics often involve the fields of the computer simulations, economic models, game theory and econometry.

It is worth mentioning that a debate seems to persist across all those disciplines, as each one has its own limitations but, in the end, these disciplines are in fact building upon one another to achieve the dream of any researcher on this subject: how to solve the complex problem of observing, optimizing and making predictions for an economic system.

From the debate around computer simulations and their lack of accuracy when used for complex systems (such as the human economy where the main subject is calibration techniques and their use in reducing the risk of creating inaccurate results based on inaccurate data), to the limitations of the economic models biased by the same lack of accuracy, to the approximations of the econometry (which gives us empirical quantitative analysis of the massive data produced by economies), every serious paper on those subjects has his own pitfalls chapter.

It is also worth mentioning that, when the subject of study for these fields become the virtual game economies, their true power unfolds and the fact that in a virtual game economy we have access to all pieces of information about the players

and the systems strongly reduces some pitfalls, when compared to the real economy applications.

For example, a computer game simulation is no longer the approximation of an economy, but it is the game economy in itself.

I.2 A brief history of economic models

The first attempts at creating an economic model come from Physiocracy, an economic theory developed by a group of 18th-century Age of Enlightenment French economists.

Among this group, François Quesnay is of particular interest for the development of the tables he called “Tableaux économiques”. These tables where quite similar with the modern Leontief model. (Almarin P. 1955).

The Leontief model is a model for the economics of a whole geographic entity such as a country. In the Leontief model there are two distinguishes models:

- The open model, in which some production is consumed internally by industries and the rest is delivered to external entities to consume;
- The closed model in which the entire production is consumed by industries;

During the 18th century many mathematicians contributed to the field of economics of insurance and to the theory of gambling, contributions that lead to the creation of probability theory and the actuarial science.

Once actuarial science was created, more and more economic models were developed by researchers in the field.

We will shortly present the most important economic models that marked the evolution of the modern economic theories.

The Cobb–Douglas model of production

The Cobb–Douglas production model was developed during 1927–1947 and it is often used as an utility function to represent the relationship between the amounts of two or more inputs including but not limited to physical capital and labor (Cobb, C. W.; Douglas, P. H. 1928).

In its most simple form, the Cobb–Douglas function is:

$$\boldsymbol{Y = AL^{\beta}K^{\alpha}}$$

where:

Y = the total production produced in a year;

L = the labor input, the total number of hours a person worked in a year;

K = the capital input, a measure of all capital which can be computed by the value of the capital itself divided by the cost of maintaining the capital;

A = the total factor productivity;

α = the output elasticities of capital;

β = the output elasticities of labor;

α and β values are constants.;

Keynesian Harrod–Domar model of economic growth

The Harrod Domar Model was developed independently by Roy F. Harrod in 1939 and Evsey Domar in 1946. This model suggests that the rate of economic growth depends on two things:

1. Level of Savings. A higher level of savings generates higher investment.
2. Capital-Output Ratio. A lower capital-output ratio is a clue that the investment is more efficient.

In its most simple form, Harrod–Domar model of economic growth is:

$$g = \frac{s}{k}$$

where:

g = rate of the economic growth;

s = level of savings;

k = capital output ratio;

Solow–Swan model of economic growth

The Solow–Swan model is another model of economic growth that superseded the Harrod–Domar model of economic growth.

This model attempts to explain the economic growth by taking in account capital accumulation, population growth and changes in productivity.

Solow–Swan model tries to distinguish increasing in capital output from technological progress by using capital output ratios that are not fixed like in the Harrod–Domar model.

The most interesting Solow–Swan model implication is that less developed countries are expected to grow faster compared to the rich ones, an implication

resulting from the assumption that economic systems under the Sollow-Swan model tend to achieve an equilibrium steady state. Since the developed countries already achieved this steady state, it is more probable that the poor countries will grow faster, trying to achieve that equilibrium state too. (Solow, Robert M., February 1956)

Lucas islands model of money supply

The Lucas islands model was developed by Robert Lucas and published for the first time in 1970. This is an economic model designed using rational expectations that establish the relation between money supply and price. In economics, "rational expectations" are expectations that may be wrong but are true on average on a longer period of time.

The theory of modelling expectations was put forth by John F. Muth in 1961. The main hypothesis was phrased by John F. Muth as following:

“expectations of firms (or, more generally, the subjective probability distribution of outcomes) tend to be distributed, for the same information set, about the prediction of the theory (or the "objective" probability distributions of outcomes).”

The most important implication of the Lucas island model is that it is important to distinguish between all predicted and unpredicted changes in the monetary policy of the economy and anticipate changes affecting the prices due to the inflation.

Heckscher–Ohlin model of international trade

The Heckscher–Ohlin model is an economic model of international trade. The model was first theorized by Eli Heckscher at the Stockholm School of Economics in 1919 and was improved by Bertil Ohlin, a student of Eli Heckscher in 1933.

The most important postulate of Heckscher–Ohlin model is that countries export resources and products that are abundant for the country, either by being available at a low cost or that can be produced cheaper with the production means available in the country. The other way around, the country will import more products that are more expensive to produce or resources that are not cheap/available in the country. The cost and availability are, of course, dependent on the raw materials available to the country as well the availability of the work force necessary for extracting resources or producing the goods.

Black–Scholes model of option pricing

The Black Scholes model of option pricing is a mathematical model for estimating the variation over time of financial instruments such as stocks. The model assumes that these instruments will have a lognormal distribution of prices.

The main idea behind the model is to assume that the price of traded assets follows a geometric Brownian motion with constant volatility and attempts to establish the proper moments to buy and sell the assets the right way in order to eliminate financial risks.

The model formula was developed by three economists, Fischer Black, Myron Scholes and Robert Merton and published in the Journal of Political Economy as a paper titled “The Pricing of Options and Corporate Liabilities”. For their work,

Robert Merton and Myron Scholes received the 1997 Nobel Memorial Prize in Economics.

The Black–Scholes model is considered one of the most well-known pricing option models in the stock trading domain.

The formula is calculated by multiplying the current stock or other underlaying price by the cumulative standard normal probability distribution function N as shown below:

$$C = S_t N(d_1) - Ke^{-rt} N(d_2)$$

where:

$$\mathbf{d_1} = \frac{\ln \frac{S_t}{K} + \left(r + \frac{\sigma_v^2}{2}\right) t}{\sigma_s \sqrt{t}}$$

and:

$$d_2 = d_1 - \sigma_s \sqrt{t}$$

where:

C = Call option price;

S = Current stock price;

K = Strike price;

R = Risk-free interest rate;

t = Time to maturity of the stock;

N = A normal distribution;

AD–AS model

The AD–AS - aggregate demand–aggregate supply model is a mathematical model that links price level and economic output through the relationship of demand and supply.

The AD-AS model is useful to illustrate the national income and changes in the price level as well as the business cycle. The business cycle is the economic cycle that marks the downward and upward variations of gross domestic product. The AD-AS shows that the price level and the output level of the economy are in equilibrium.

AD–AS aggregate demand–aggregate supply model formula for the AD-AS curve is:

$$\boldsymbol{Y = Y^{d}(\frac{M}{P}, G, T, Z_1)}$$

Where:

Y = real GDP (gross domestic product);

M = the nominal money supply;

P = the price level;

G = real government spending;

T = real taxes level;

Z1 = other variables that affect the location of the IS curve;

I.3 A brief history of computer simulations

Compared to the history of economic models, the history of both computer simulations and virtual game economies starts at a much later time in connection with the evolution of modern computers.

Like with many other scientific fields, the evolution was driven by war efforts during Manhattan Project in World War II. The need for creating a nuclear bomb conducted to the creation of the first computer simulation. John von Neumann used the ENIAC computer to perform calculations for studying neutron diffusion in radioactive material, using a Monte Carlo algorithm created with the purpose of finding out all the possible interactions between 12 hard spheres, which were simulating particles.

The computer simulation field evolved, with many applications to this day, when simulations are used in many fields of science, like for example, the computer simulation of the COVID-19 spreading pattern, CovidSim microsimulation model developed by the MRC Centre for Global Infectious Disease Analysis hosted at Imperial College, London. This simulation was taken in account for political decisions at the highest level in United Kingdom regarding the measures to implement based on the outcome predicted by the simulations. (Bostock, B)

Some people expressed their concern on the subject of this computer simulation with voices debating the accuracy of the simulation predictions. New Scientist reported in March 2020 that a group from the New England Complex Systems Institute suggested that it contained "systematic errors". (Hamzelou Jessica).

We took those two different stories as the perfect example to show how a simulation can prove to be the best undisputed tool for the job or a controversial approach in the second case.

We found the subject is discussed in deeper details into a paper published by Aki Lehtinen and Jaakko Kuorikoski. They claim that “the dearth of simulation models is most conspicuous in the most widely respected journals that publish papers on economic theory”.

I.4 A brief history of virtual game economies

The birth date of virtual game economies is hard to point out. Most common examples we found by studying various publications on the subject, tend to mention only the first multiplayer games that made a huge economic success by building virtual economies.

The researcher’s tendency is to focus on virtual economies from games that drove massive profits to the real economy as well, ignoring older games that actually created the first virtual economies we see in later games.

We find this attitude biased and we believe that is more appropriate to link the birth date of virtual economies to the birth date of the first strategy game. We did a lot of extensive research and we found out that the exact game that started the revolution was harder to point out than we initially expected.

Some may argue that early Risk-like games from 1972 are the parents of virtual game economies.

We dare to point to a game named Utopia. Utopia was never published and most detailed information we know about this game comes from an article published by Barry Tim on May 11, 1981 in InfoWorld magazine called "In Search of the Ultimate Computer Game".

Utopia was a game running on an IBM System/370 Model 168 at a large San Francisco Bay area company and it was never published due to the obvious fact

that nobody had a mainframe computer in their home. The game was known to the company employees only and it was such a success that the employees where mesmerized by it, stopping all the work when the computer was up and running at a scheduled daily hour. According to the description gave by Barry Tim, we think that this was the first game implementing what we know now as a virtual game economy and the parent of all his children.

The virtual economies in more later games are easier to point out. We will not present all information, which can be easily found in the bibliography included, but we will mention a few games that evolved into both massive virtual economies and real economic success.

A title worth mentioning is the Second Life game, an online virtual world which at its peak, had around 1 million regular users with over 1.2 million transactions each day for virtual goods. It is worth mentioning the Second Life game implemented a virtual currency called Linden Dollars.

Another title worth mentioning is World of Warcraft which was one of the world's most popular massively multiplayer online role-playing game (MMORPG) by player count of 10 million players in 2009. By 2020 the World of Warcraft game generated more than $10 billion in revenue.

It is also worth mentioning that many titles still exist and the virtual economies they deployed many years ago are still available today.

A good example is EVE Online, a space-based, persistent world MMORPG that was released in May 2003. The Eve Online economy is still up and running, receiving new updates from the developers and is populated with a lot of corporations. Measures similar to the ones took by states are taken from time to time such as the modification of the game economy rules or virtual currencies exchange rates.

Virtual economies are, of course, not specific only to games. Many mobile and web applications today implement virtual currencies and ways to reward users with those virtual currencies for their activities across the application. Slashdot and Reddit are only a couple to mention. Social networks like Facebook are known to express interest for implementing cryptocurrencies for their platforms in the future.

Which lead us to a subject worth mentioning when talking about virtual economies: the cryptocurrencies. Cryptocurrencies are the perfect digital asset designed to work virtual economies.

I.5 Game theory elements

We believe that we cannot go further in studying virtual game economies without at least mentioning the game theory field of study.

According to Roger B Myerson, game theory is “the study of mathematical models of strategic interaction among rational decision-makers”.

We will not analyze all the game theory concepts but rather try to mention only the ones that are used by our virtual economy simulator.

Regarding the game type, our simulator will implement a combination of the following types of games from the game theory:

Non-cooperative.

The players will not be able, in our simulation, to form alliances, as opposed to cooperative game theory, which focuses on predicting which coalitions will form between the players.

Non-zero-sum

In non-zero-sum games, a gain by one player does not necessarily involve a loss by another player. Our simulation is not a zero-sum type of game because it

implements a mechanism similar to the real economy extractive industries that take raw materials from earth and stock the extracted resource for their own usage.

We call the processes for extracting virtual resources “mining”. In order to simplify things, the players will only mine one resource type called “energy”. In our simulator we randomly assign a mining skill to the random generated player population. The players with higher random mining skill will mine more “energy” they can sell to other players or stock for themselves for later use.

Evolutionary game theory

Our simulation relies heavily on the evolutionary game theory. Despite the fact that other game theory elements can be recognized in our simulation, at the core, we created an evolutionary game.

We implemented some aspects of the evolutionary game theory with a randomly generated population that behaves differently in our simulation and deploys different strategies for achieving profit. The members of the population are changed from time to time using a genetic algorithm that we will describe later in this chapter.

II: Computational models used in our virtual economy simulator

II.1 The genetic algorithm used for population

For generating and handling population during the lifetime of a simulation we use a genetic algorithm. The algorithm is inspired from natural selection and implements all the common stages of a common genetic algorithm.

First of all, an initial population is generated, each population individual having a genome. The individual behavior in the simulation is determined by each individual genes so, in order to introduce a more variate behavior among our virtual economy players, we took all the measures necessary to generate random behavior across our players.

After the initial population is created, the simulation will advance step by step. With each step, the simulation will increase the energy resource of each player through mining, according to the player genome, exchanging energy between players, paying taxes to the simulated state entity and receiving energy back from the state through redistribution of taxes. This is what we call a simulation cycle.

At the end of every simulation cycle, the genetic algorithm will update all the simulation population using a combination of a fitness function for selecting individuals to remove or keep and we will also perform crossover and create new individuals through combinations and mutations of the genes of existing population individuals.

The detailed implementation is described in Chapter III.6 (Simulation process).

The individual genome and the initial population

We call one individual from the population the “player”. As written above, each player behavior in the simulation is established by its; genome.

Our simulation defines a set composed of 6 types of genes. Those genes act for each player as skills. The default set of genes pool (or skills) contains the following:

0 - archaic gene (has no effect on player behavior);

1 - miner gene (increase the mining skill);

2 - trader gene (increase the trading skill);

3 - lazy gene (increase the laziness of the player);

4 - thief gene (increase the thief skill);

6 - hoarder gene (increase the predisposition of the player to store resources).

Each player genome is a multiset of genes from the genes pool. In our simulation the same gene can occur many times in the same player genome. The amount a particular gene occurs, increases the effect of that gene on the player behavior.

The gene pool is:

$$G = \{x \mid x \in N, x < n\}$$

where n is a constant (in our case 6). However, please notice that we can add as many gene types (skills) as we want to the gene pool, simply by changing the s constant.

All possible genomes multisets are: $\binom{n}{k}$

where k is a constant, the cardinality of the genome. We use the k constant to restrict the genome of each player to a predefined number of genes. We could, in

theory, create multisets with different cardinality but, for the simplicity of the simulation, we decided to use a fixed cardinality.

For simplicity, when creating the initial population, we may not actually generate all possible genome multiset combinations, we can rather use a random number generator to generate random multisets till we reach the population limits.

However, it may be an interesting approach to generate all possible combinations and determine the population limit from the number of possible combinations for the specified k. In this case, the maximum population limit can be determined with the binomial coefficient (n; k). (The binomial coefficient is the number of ways of picking k unordered outcomes from n).

The value of the population (for $n \geq 0$ and $k \geq 0$ is given by the formula:

$$\boldsymbol{nC_k} = \binom{\boldsymbol{n}}{\boldsymbol{k}} = \frac{\boldsymbol{n}!}{(\boldsymbol{n}-\boldsymbol{k})!\,\boldsymbol{k}!}$$

Fitness function

We implemented three different fitness functions the user of the simulator can choose for a simulation. The choices of fitness functions for the current simulation are parametrizable from the simulation web GUI.

The fitness function will return a real number $T \in [0,1]$. Our convention is that a bigger T value represents a better fitness compared to a lower T value.

The first fitness function will favor players that contribute more to the economy in the current simulation step. We call this fitness function the “profit-oriented” one.

For T = 0 the fitness of the player is 0:

$$\boldsymbol{T = 0 \leftarrow T = 0}$$

For M = 0 the fitness of the player is 0:

$$\boldsymbol{T = 0 \leftarrow M = 0}$$

For M > 0 the fitness of the player is:

$$\boldsymbol{F = \frac{T}{M} \leftarrow M > 0}$$

where:

T = the total amount of the taxes payed by the player for the current simulation step;

M = the maximum taxes amount payed by a single player for the current simulation step.

The second fitness function will favor players that contribute less to the economy. We call this function the “unfair” one.

For T = 0 the fitness of the player is 1:

$$\boldsymbol{T = 1 \leftarrow T = 0}$$

For M = 0 the fitness of the player is 1:

$$\boldsymbol{T = 1 \leftarrow M = 0}$$

For M > 0 the fitness of the player is:

$$\boldsymbol{F = \frac{M - T}{M} \leftarrow M > 0}$$

where:

T = the total amount of the taxes payed by the player for the current simulation step

M = the maximum taxes amount payed by a single player for the current simulation step.

The third fitness function will not favor any players but rather randomly return a fitness value no matter what the profit generated by the player was. We call this function the “neutral” one. The formula for the neutral fitness function is simply a random real number in between 0 and 1 interval:

$$\boldsymbol{F = x \in R, x \in [0, 1]}$$

Selection

For the selecting stage of the genetic algorithm, we simply order the population players using descending order by the value returned by the fitness function and we pick the first X players. X is a constant that can be customized from the simulation web GUI.

Due to the complexity of the sort operation, this method of selecting is, of course, computationally demanding. We could implement a faster method of selecting that uses the so-called stochastic acceptance, either by implementing a truncation selection algorithm based on taking a portion of the population and selecting the best individuals from that segment or other means. However, for the purpose of our simulator we used the simplest solution.

Crossover

In genetic algorithms, crossover is a genetic operation that combines the genomes of two members of the population called parents, in order to create a new child. In our simulator, the crossover function will combine two different player’s genomes.

There are many possible ways to combine the genomes of the parents such as the single-point crossover, the k-point crossover or uniform crossover.

We don’t use either one of them in our simulator, but a modified version of the uniform crossover method. Our method resembles the uniform crossover due to the fact that we iterate all the genome gene-by-gene and decide which one to keep for the child, but the decision is not taken using an equal probability algorithm like most common uniform crossover algorithms do, instead we simply use a random generator function that is not guaranteed to create equal an distributed output. As a result, our implementation is not guaranteed to be uniform distributed and strongly depends on the random generator algorithm output.

Mutation

In genetic algorithms, mutation is a genetic operation that imitates the mutation that occurs in the process of life evolution. The mutation is useful to maintain a genetic diversity, otherwise our population gene pool will deplete faster by recombining the same genomes again and again. For the mutation, we used the uniform type of mutation. The uniform mutation is performed by choosing a number of genes from each genome and replacing it with a random value selected between the upper and lower bounds for our gene pool $G = \{x \mid x \in N, x < n\}$

III: Virtual economy simulator

III.1 Simulator goals and specifications

The main simulator goal is to emulate the economic interaction between different players. The secondary goal of the emulation is to study the best actions in order to favor the economic growth of the various virtual economies through taxation.

In order to achieve the secondary goal, the simulator implements mechanisms for continuously changing the interaction rules between the players, with the purpose of moderating the virtual economy in order to encourage value added to the economy by players, favor players interconnectivity and increase the virtual economy profit by virtual taxation.

The simulator will continuously try to find new ways to increase the value of the economy through two different categories of measures:

1. Redistribution of the economic growth, measured in simulation currency acquired through taxation in between the players.
2. A genetic algorithm used to handle the replication of the players as well as to emulate the scientific progress of the virtual society that, in the end, increases the mining rate of the energy by the simulated players.

For the first category of measures, redistribution of the economic growth, as presented in detail in the previous chapter II, three different algorithms are implemented:

- one that favors players that contribute more to the economy,

- one that favors more the players that contribute less, in the hope that they will generate more value to the economy in the future and

- one that will attempt an equal distribution of the economic growth across most players.

For the second category, the genetic algorithm, at least three methods for establishing the fitness of each individual must be implemented and studied into the simulation. A formula that favors more the replication of the players that are found to add more profit to the economy, a random formula that does not favor either the best or the weakest players and a formula that favors the weak players and encourages their replication instead of the replication of the best players.

The simulation will advance in steps, each step creating a new generation containing both old and newly added players. At each simulation step, a random number of players will be eliminated according to the chosen fitness function.

The simulator defines a standard simulation currency called “Energon” that can be earned by the players and used to pay taxes or to spend on simulation items. The items bought by the players will reward the player with more productive ways to create the standard simulation currency and, in return, increase the simulation economy through taxation.

The application must offer the possibility to run the simulation for a specific number of steps or indefinitely until the user of the simulation decides to stop the simulation manually.

All running simulations must log the simulated virtual economy state at each cycle and the information will be displayed using the Web service GUI through various graphs including a final report with the final state of the economy at the end of the simulation.

Results of running various simulations with various measures in place will be published.

III.2 Technology stack

We used the following technology stack to implement our virtual economy simulator:

- Erlang functional programming language
- Cowboy - a small, fast and modern HTTP server for Erlang/OTP - https://github.com/ninenines/cowboy
- SGTE - a simple Erlang Template Engine for rendering the HTML - https://github.com/filippo/sgte
- Mnesia – a database using erlang native mnesia module - http://erlang.org/doc/man/mnesia.html
- Rebar3 – a tool for easing the compilation of Erlang applications - https://www.rebar3.org/

All graphics are taken from various open source projects and are free to use. Icons from the Web service GUI are from the Linux Gnome project.

From the beginning, we wanted to keep dependencies usage as low as possible in order to ease the future update process of the virtual economy simulator application. The short technology stack should also make the application portable and available on a large number of operating systems.

Our goal was to create an application that anybody can run on any modern operating system that supports the programming language choice and implements everything with a minimal set of tools, including developing a web framework from scratch.

Why choose Erlang?

First of all, Erlang is a general purpose functional programming language. This characteristic recommends Erlang for solving complex mathematical formulas with ease, being strongly related to mathematics. Erlang treats most functions as deterministic mathematical functions as opposed to the imperative programming approach that, in the effort to describe the exact steps used to solve a problem, furthers away from the original mathematical formula for solving the problem itself.

Another argument for choosing erlang is the fact that the Erlang runtime system was designed by Open Telecom Platform product unit at Ericsson with the following features in mind:

- Distributed
- Fault-tolerant
- Highly available
- Soft real-time
- Implements hot code swapping feature

Erlang is a highly scalable runtime system that is quite capable to simulate a huge virtual game economy with millions of players involved. Erlang capabilities were demonstrated in our real economy systems, being used to implement massively scalable soft real-time systems from complex global banking software systems at Mastercard, to huge computer telephony network at Ericsson and instant messaging at WhatsApp. Erlang is one of the software components behind the global real economy, being used in many interbank communication servers and protocols.

The Erlang runtime system can be easily scaled up by adding more nodes to the Erlang distributed network. The fact that Erlang allows hot code swapping also makes Erlang a good candidate to use for building a real-time game economy with

no need to stop the servers or the players from playing just because we want to deliver updates or fixes to the virtual economy simulator.

III.4 Application architecture

We built the system in such ways that there are no limitations whatsoever regarding the number of the simulations you can run concurrently.

We tested the application with more than 50 simulations running in the same time and displaying the status on web service webpages, and it's working fine! There are no limitations whatsoever, but please notice the simulation display page can become slow the more simulation processes we display (it's a display optimization problem). We don't advise triggering more than 100 simulations processes if you want to observe the simulations evolution in real-time using the webservice GUI because this could result in some form of DDOS attack of the HTTP server. This is usually only a web service page/displaying simulation problem not a simulation server problem.

Each simulation runs on its own process, and the simulation loops are implemented using triggers – messages that are sent from time to time to the simulation process to trigger the next simulation loop, allowing short time processes hibernation in opposite to common loops that use a big amount of CPU power. Internally we tested more than 1000 simulation per node, and they worked flawlessly. Many more simulations can be run by deploying more erlang nodes on different machines.

III.4.a Component overview

The virtual economy simulator is composed of the following components:

- **Webserver** delivering the webservice GUI

- **The simulation manager** handling many concurrent simulations
- **The simulation process** itself
- **The database** used to log the results of the simulations

Next, we will guide the reader through the application starting sequence for a better understanding of the relations between application components.

Since erlang is not intended to run as a single isolated application but in a distributed cloud environment, we need to setup and start the simplest distributed erlang environment possible containing at least one nod.

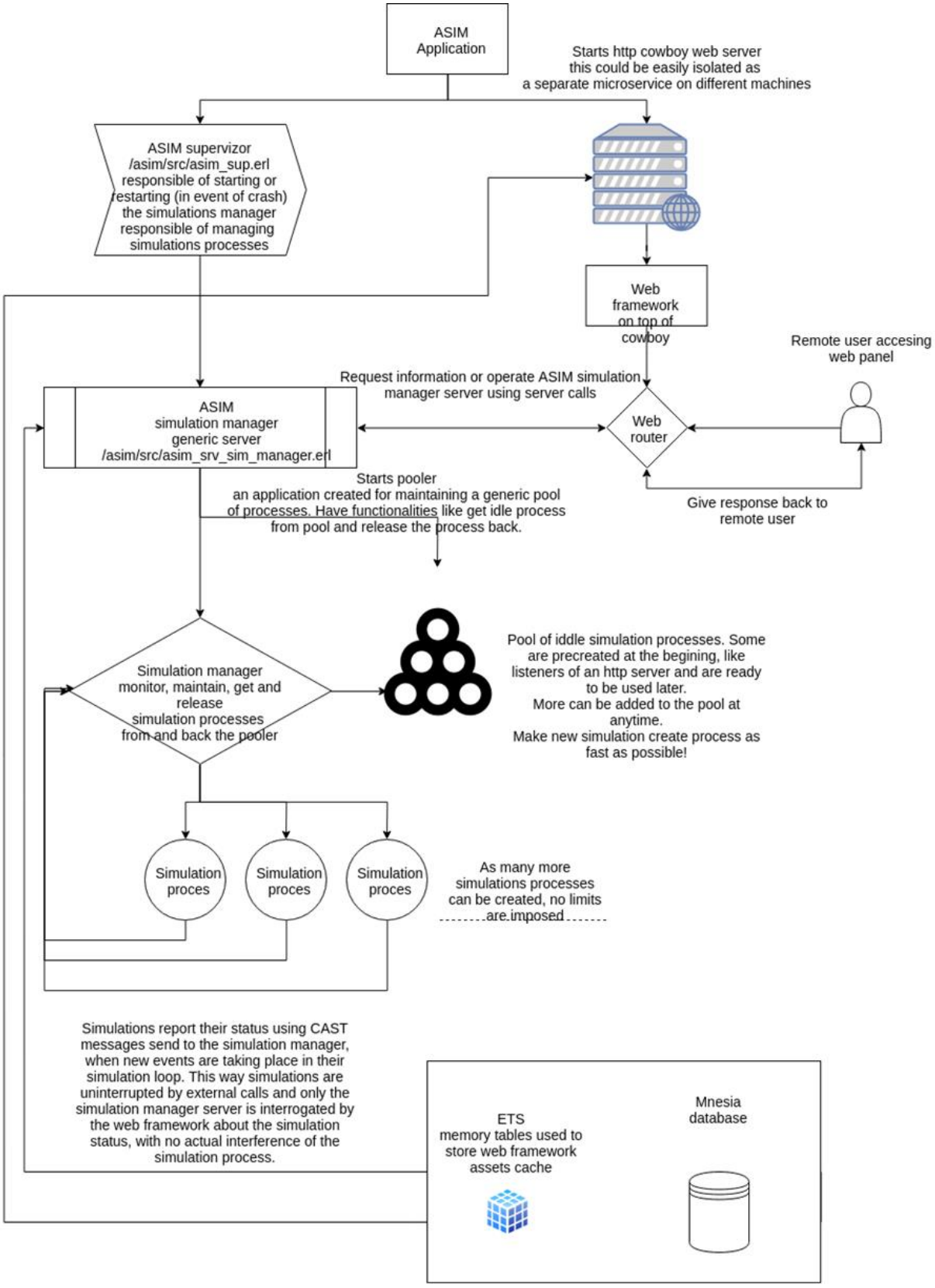
The application startup script configures and starts this erlang node. At this stage, the startup script already instructed the erlang virtual machine which applications to run to run at this node. As a result, the asim application and all erlang OTP application asim depends on are now started on the node.

From this point forward, the asim application takes control of the remaining initialization process. Asim will start the main application supervisor process which is responsible in cascade for starting the rest of the processes. One important child process of the main supervisor is the simulation manager which does exactly what its name implies – manage the simulations. In the same time the http server based on cowboy is started on a different process and it’s ready to server the web GUI. A router is populated with routing rules for the incoming requests to the HTTP servers to execute the proper erlang modules from the web GUI.

The pooler application is also started and configured to spawn simulation processes in advance for getting them faster later when simulations are actually triggered by the user of the web GUI. The main responsibility for using the pooler and respond to commands from the web GUI belongs to the simulations manager.

The way those components interact with each other is also presented in the software architecture diagram below.

III.4.b Software architecture diagram



III.4.c Project file structure

All project files are placed into "asim" (A SIMulator) directory. The following sub-directories exists:

- **assets** - contains assets used by the web service, including HTML templates. Because of the mandatory structure of an OTP Erlang application, views were placed outside src directory.
- **assets/templates** - contains HTML templates used like views by web service controllers.
- **assets/files** - contains other assets files
- **assets/files/css** - contains CSS files
- **assets/files/img** - contains HTML images
- **assets/files/js** - contains java script files
- **assets/files/http** - contains HTTP server necessary files like database of IMT, used to build the proper content header when serving images, CSS, JS files.
- **assets/files/vendors** – contains various files used by the web framework HTML
- **bin** - contains necessary scripts for controlling the simulation server and other shell scripts provided for easily compiling the server.
- **config** - contains the simulation server configuration files including erlang cloud configuration, ip the server is listening on, port and other options.
- **mnesia** – contains Mnesia database persistent files
- include - contains the simulation server include files
- **src** - contains simulation Erlang source code
- **src/db** – contains some functions for creating memory tables and mnesia tables
- **src/ext** – contains the web framework (implemented from scratch) controllers. We call them extensions. Each web page has module implemented in this directory.
- **src/http** – contains various functions for dealing with an http request

- **src/web** – contains the web framework source code
- **src/utils** – contains various utils modules
- **_build** - contains files related to building the application with Rebar
- **.idea** - contains IntelliJ Idea project files

III.4.d Database

As presented into the dependencies chapter, we decided to use the Mnesia distributed NoSQL database. The database can be easily configured to replicate itself on a cloud. The performance of the database should be quite amassing since Mnesia can be configured to use memory copies of the tables and act like Redis or other in-memory data structure stores.

Please note that Mnesia does not really implement standard database data types, but, for simplifying the understanding of the database structure for any software engineer which will analyze this simulation, we created a diagram that uses data types closer to the SQL equivalent.

Columns have no type limitation in a Mnesia database and can hold any erlang term, including complicated records or infinite integer numbers. Some types have no equivalent in SQL like atoms so we converted them (into the diagram) to integer types, most like the way we insert some enum values in any traditional databases.

Because Mnesia supports complicated structures embed into columns, that are quite hard to represent into standard database diagram, we simply noted them with blob. For example, in the asim_simulation table we saved the final state of simulations, that contains complete final states of the game economy from the simulation.

This way, we were able not only to display the simulation game result in web service, but the various graphs representing the simulations outcomes and many other information like the last state of the economy.

Please notice we didn’t use auto-increment integer ID’s for database records primary keys, like in traditional SQL databases. Integer unique insert ids have many limitations and cannot be used in big distributed databases. We used UUID instead that are generated at insert time and guaranteed to be unique across a cloud without using any global counter facility or requiring any database table locking mechanisms.

The source code for generating UUID is in /asim/src/utils/asim_lib_utils_timeuuid.erl file.

The time UUID was designed by the author of this paper as a creative exercise, it is 64 bytes long and it's made up of 4 components:

- 2 bytes - UUID version, always 00 for current implementation version, allows us to change the UUID implementation in the future, gives us to design 99 more versions of the UUID and still keeping compatibility with older UUID including the proper order (any new hash generated with a new algorithm being bigger than a previous version. This could be very valuable for time series.)
- 13 bytes - current Unix Timestamp in milliseconds limited at maximum value of 9999999999999 =11/20/2286 date
- 10 bytes - local cloud node int32 monotonic increasing counter limited at a maximum value of
- 4294967295 (allows us to generate a maximum of 4.294.967.295 unique UUIDs per millisecond per node)

- 39 bytes - current node name hash (128-bit hash) for ensuring the uniqueness across the cloud (limited at a maximum 340282366920938463463374607431768211456 combinations)

It is quite safe to assume we will never reach those limitations into the near future.

III.5 Web service GUI

The web service is implemented using Cowboy HTTP library. We used no web framework and we actually implemented a basic web framework from scratch, that is able to handle various HTTP requests, using a router that determines and executes the current request “extension”, “module” and module “operation”.

The architecture of the web service resembles a very basic MVC web framework, where erlang modules are the controllers, the views are some HTML templates that are used by the current requested module (controller) to render various data.

We implemented a view caching mechanism, the views are read from view files and kept in RAM, in ETS tables to be reused the next time the views (templates) are requested. Views are placed under the /asim/assets/templates/v1/html directory. I used v1 to allow easily management/change of various versions of the views and avoid caching problems.

Views are compiled and views parameters are replaced with proper values using a template library. The rules are quite simple, any variable that the controller wants to replace is written into views using $variable_name$ notation. Compiled views are kept in RAM so the views parameters can be replaced very fast. The same mechanism is implemented for other assets in assets directory, like images, CSS

files or java script files. Everything is loaded into RAM only when it's first requested by a controller. The mechanism is the same as memcache modules implemented for an ordinary apache server. As a result, the web service should be able to serve any assets file very fast with no disk access.

The reader can find the implementations caching views mechanism in asim/src/utils/asim_lib_utils_assets.erl

Caching of assets can be enabled in production or disabled in testing stage, allowing the developer to see the changes instantly.

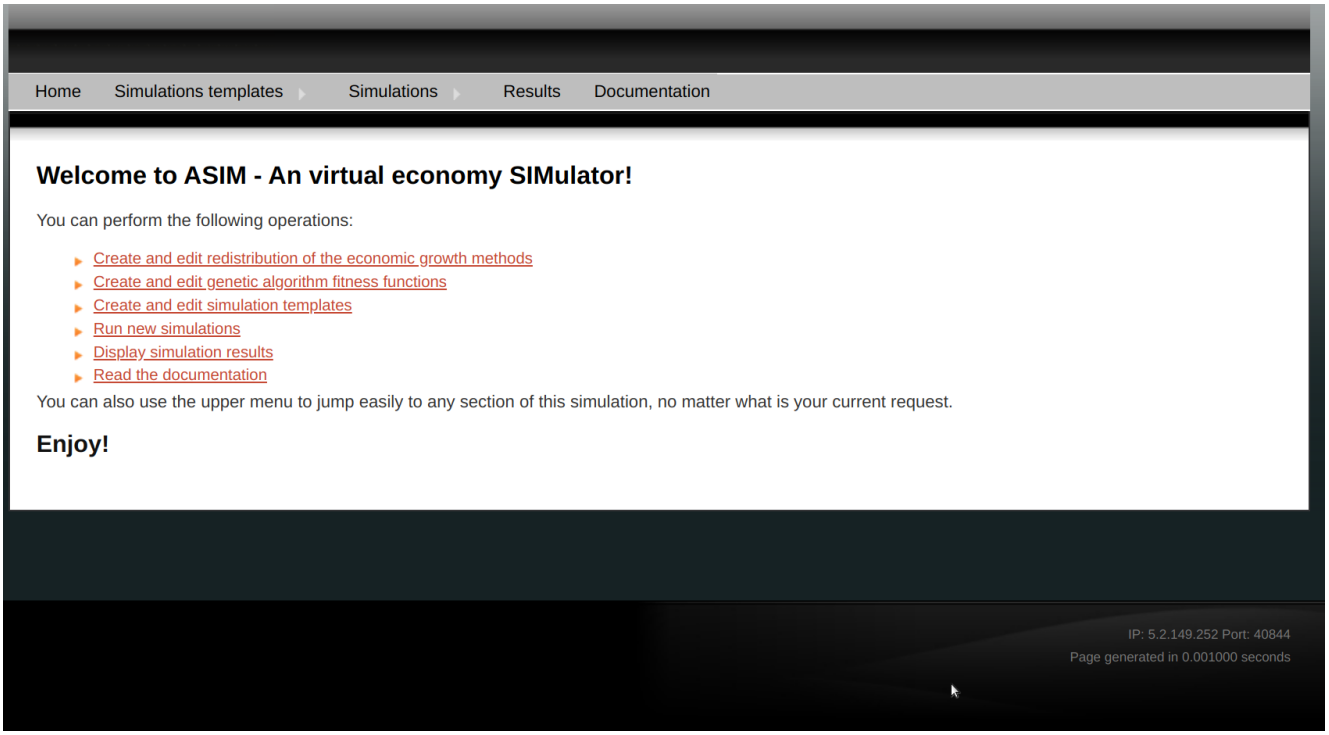
We also implemented some very basic modules that can be used to generate html forms, create, edit, delete one particularly row from database, or a list of rows from the specified table. We didn’t create a standard MODEL implementation from the usual MVC frameworks, we only implemented the functionalities necessary for the simulation forms.

We also implemented an output buffer caching mechanism (asim/src/web/asim_lib_web_ob.erl), an exceptions handler that output the proper HTTP error codes to the browser (asim/src/web/asim_lib_web_exceptions_handler.erl), an url builder and some other necessary functionalities for building this little web framework from scratch.

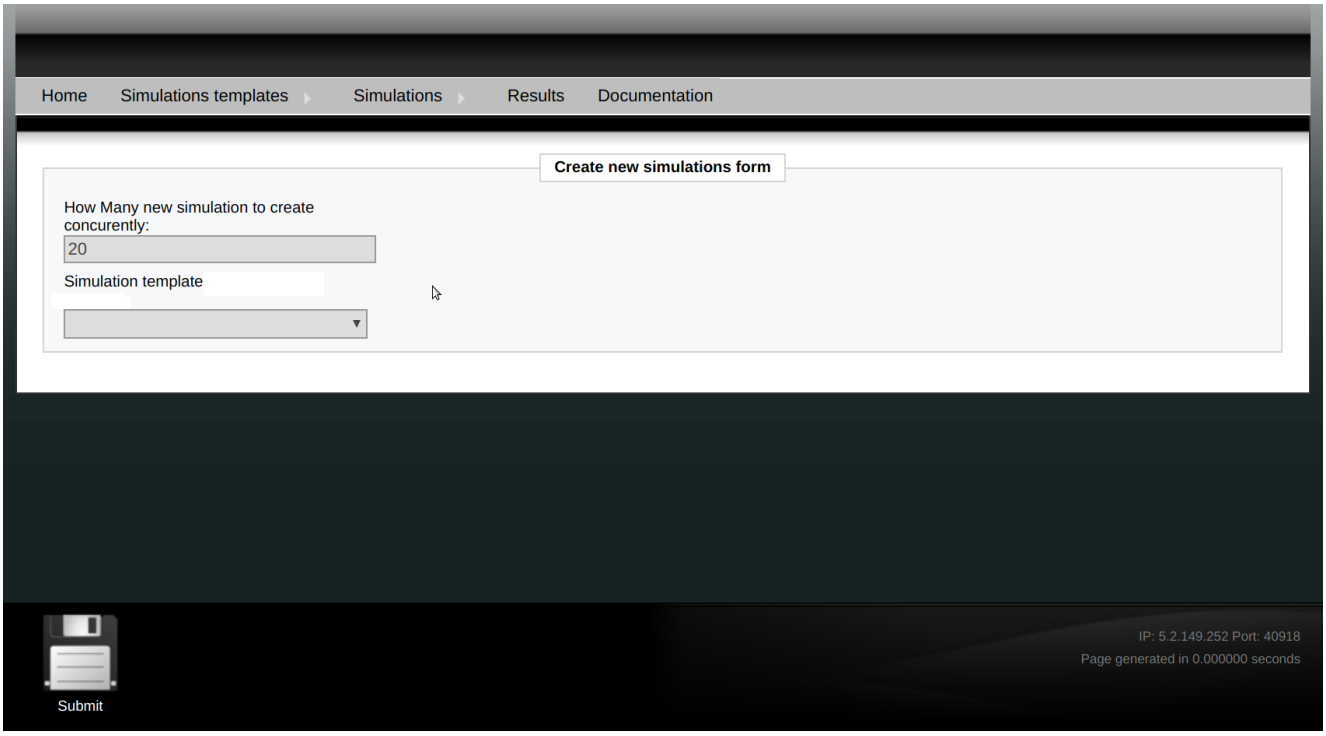
For serving files we had to create a mechanism similar with the one implemented in nginx or apache for detecting the proper file MIME types and output the proper headers to the browser for serving images or other files. This is implemented into asim/src/web/asim_lib_web_imt.erl The module use a database of common Internet media types (IMT) that is loaded from assets directory into RAM ets tables (asim/assets/files/http).

Screenshots

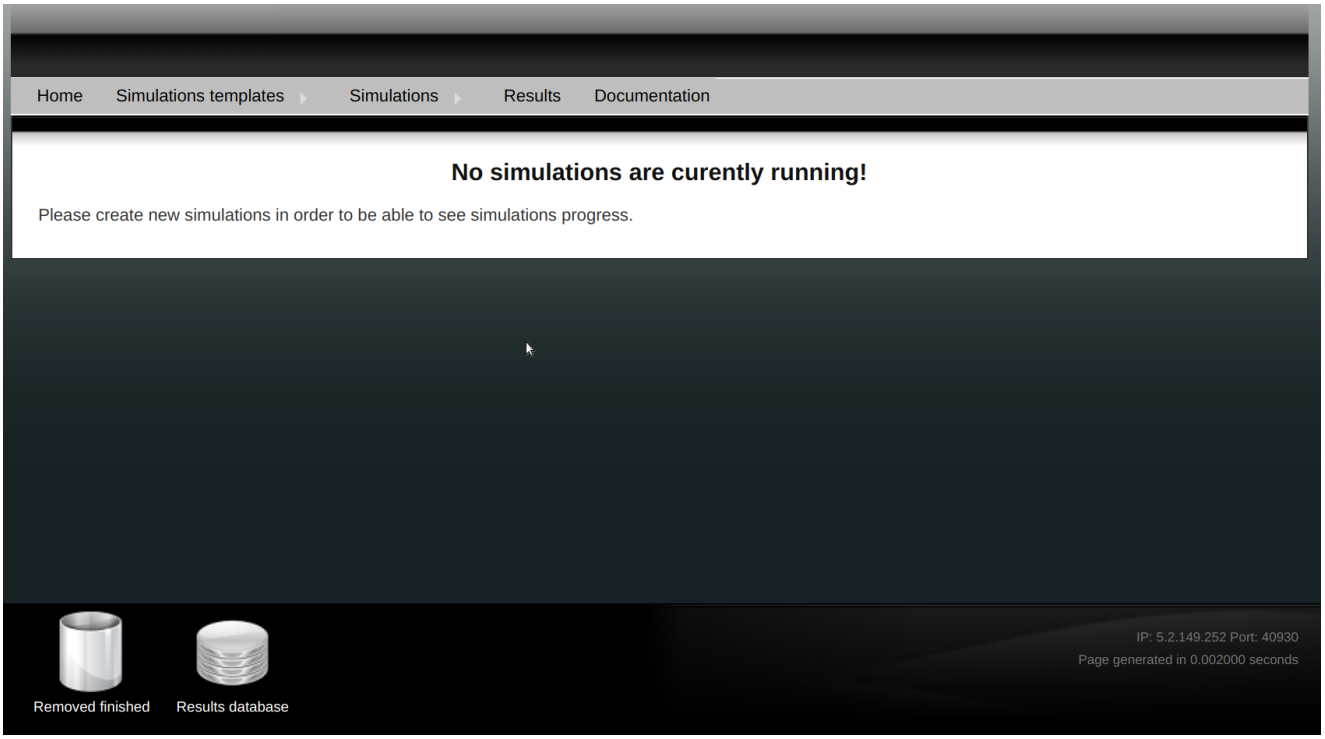
Webservice home page



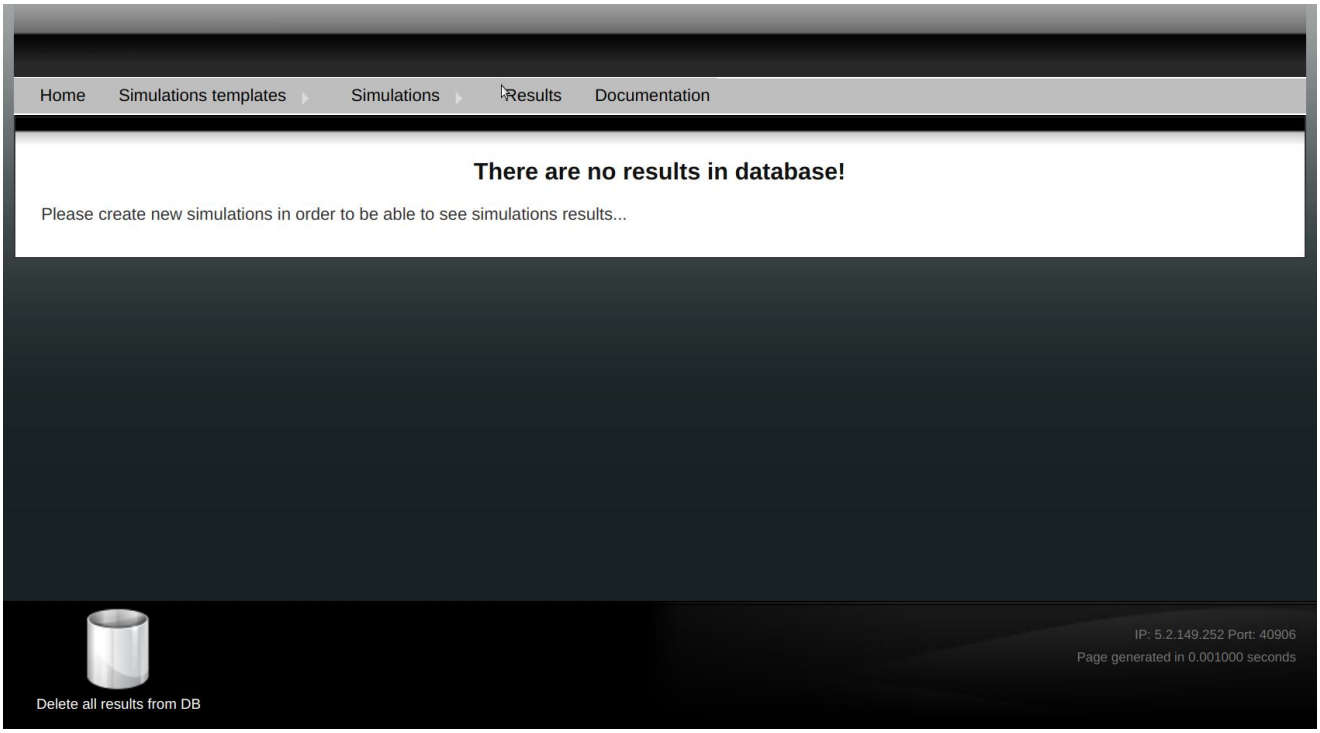
Create new simulations form



Display running simulations (no running simulations)



Simulations empty results page



III.6 Simulation process

The simulation process is started using a module that implements gen_server Erlang behavior. Traditionally, a server is part of a server client model where the server is used by many clients for operating on a resource managed by the server.

Probably the simplest way to implement the simulation was by spawning a process that simply loops through the simulation loop until the simulation is finished. Unfortunately, if the simulation were to be designed this way, the simulation process could become quite irresponsive until finishing the simulation task.

That's why we implemented the simulation process using a gen_server. The simulation server s responsible both for handling the simulation and responding to various calls from clients including calls that ask for the simulation status or calls to stop the simulation.

By using a gen_server the possibilities are endless. We can easily add new features like pause a simulation and resume it later from a web GUI triggered client for example.

Also, the gen_server behavior can be easily used to implement timers that can be setup to run the simulation steps. Instead of running the simulation steps in an uninterrupted loop, the gen server can cast a message to itself with a timeout parameter. The timeout interval allows us to pause in between simulation steps and this way tune the simulation running speed.

Further, at each step, the simulation server can actually spawn a separate process for the simulation loop and continue immediately responding to calls from web GUI clients as well as calls from the simulation process itself that can report back

when the current step of the simulation ended so the server can spawn the next simulation step process.

Another advantage of having the simulation server and the simulation process is that the server will be unaffected by any crash in a simulation step and can take necessary measures to either retry the step or update information in database about the failed status of the simulation.

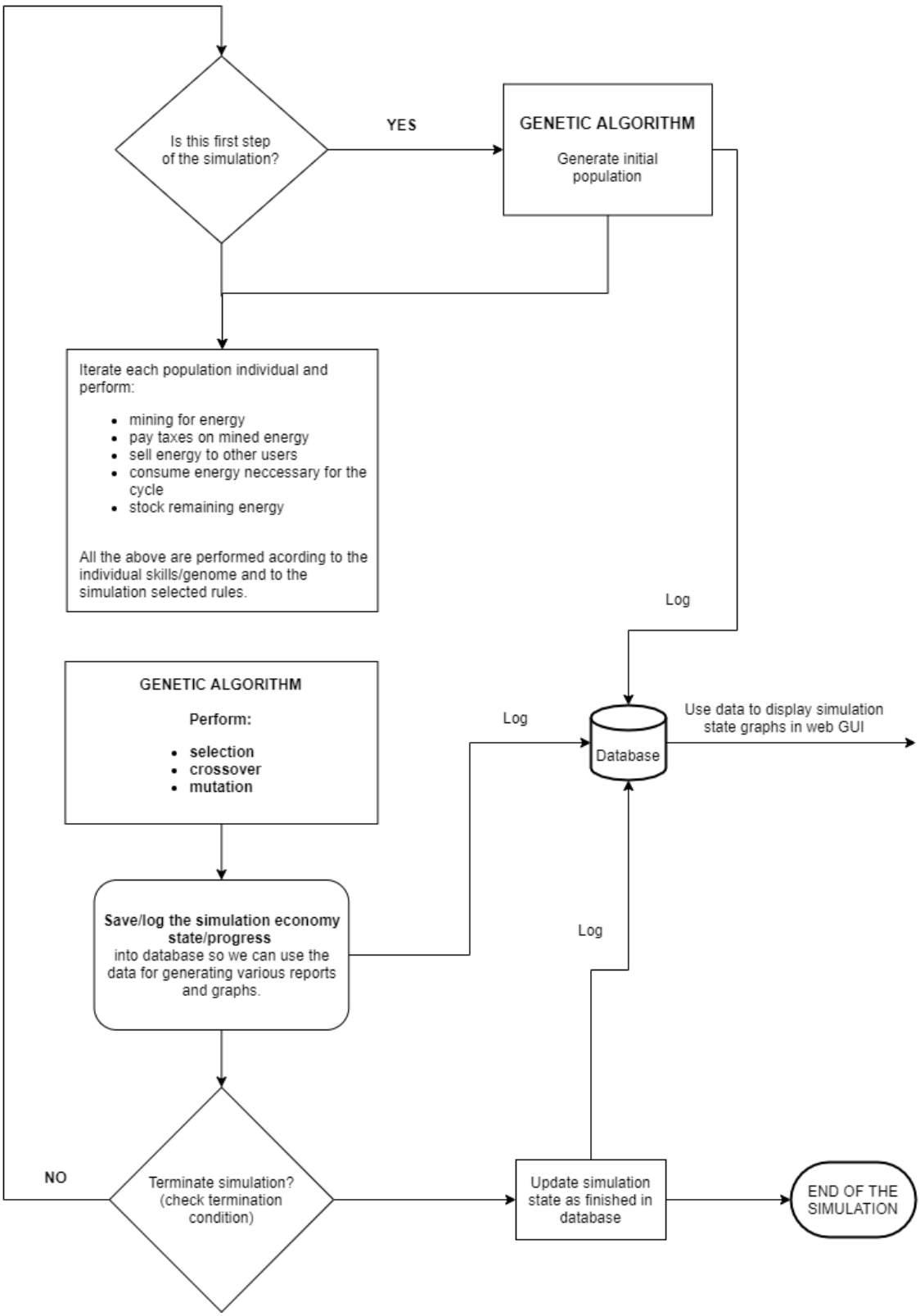
Now let's present some details about the flow we implemented into each simulation process. When a simulation is started, the simulation process will prepare some initial simulation state and execute the first simulation step. The simulation step function will take care to generate the initial population if necessary and update the database with some information about the newly started simulation that can be used by the web GUI if necessary.

Next, the simulation will iterate each generated population individual and perform individuals' actions according to their genome which is composed by a set of genes that acts like skills. According to the genes each individual possesses, they will either mine for energy, sell energy to other layers, stock energy or perform a combination of those tasks. At the end, the individual will pay taxes for the mined energy according to taxation level established by the simulation rules in web GUI when started.

In the next step, the evolution part of the genetic algorithm will take place. The population will be updated according to the genetic algorithm rules described in chapter II by performing selection, crossover and mutations. After this, the current state of the simulation economy will be saved into the database (for each step) so web GUI can easily display various graphs representing the simulation evolution.

When the simulation termination condition is true, the simulation will end, otherwise the process will continue again from the beginning with a new step.

Simulation process diagram:



III.8 Deploying and testing the application

III.6.a Application configuration

Like stated before into the Project file structure chapter /asim/config directory contains all simulation configuration files as follows:

- **/asim/config/vm.args.src** - By default asim will give a basic vm.args.src file that sets a node name and cookie. For a complete list of options and their use check the Erlang documentation.
- **/asim/config/sys.config.src** - For passing application environment variables there is sys.config.src file.

The simulation can be run under an unprivileged user or under the root user. That's why the web service port can be configured to use a privileged standard port like http port 80 or a higher one that does not require any root privileges or kernel tuning. We chose as default port the 5012 port.

The ip web services are listening on must be configured according to current machine configuration.

The erlang node full name must be configured according to the current network interfaces available.

Other configuration options are not necessary to be changed and some defaults where provided that should work well on any machine.

III.6.b Compiling and running the simulation

Simulation server must be compiled on the platform you are intending to run/deploy. In order to be able to compile the simulation server you need to:

1. Install Erlang OTP version 21 (preferred) or later on your platform.
2. Install “development tools” on your platform including make and a C/C++ compiller.

Configure the server by modifying the /asim/config/sys.config.src (most parameters here are self-explanatory). Most importantly, you must change the node_public_ip, host and location of the mnesia database. The database will be setup automatically on runtime if missing when running asim server.

However, please make sure the server process has the proper access rights to the database directory.

```
[ {asim, [
{node_public_ip, <<"5.2.149.252">>},
{protocol, <<"http">>},
{host, <<"5.2.149.252">>},
{http_port, 5012}
]},
{mnesia, [
{dir,"/home/madalin/asim/mnesia"}
]}].
```

Change directory to /asim directory and run # make release

A new release will be created into _build directory.

Change directory to asim/_build/default/rel/asim/bin and execute #./asim console to run the simulation server in console mode. That’s all.

Please notice we tested the server on latest Fedora, RHEL and CentOS distribution. It should work well on any Linux flavor as time as you install the proper erlang version and build tools (make & GNU C/C++ compiler). Please also notice that you may need to open the 5012 port on your firewall and allow the asim application proper access to bind ports if you use some sort of Selinux enabled distro.

IV. Conclusion

It was clear for us, from the beginning, by design, that our virtual economy simulator has little applications in the field of real-world economies. Apparently, the results of running the simulation have little applications and the simulation can not be used in any way to predict or study economies of countries, for example.

At a first glance, the simulator seems to be just a game. We come to the conclusion it may not be.

When building and testing the simulator, we realized there are many ways for improvement that can transform this simulator into a tool that may prove very useful when analyzing exactly the type of virtual economy was build for: a virtual game economy with no relation to reality.

The outcome of the simulations tells us little about what's the best approach to increase the income into a virtual economy. It is more probable that there isn't the best approach and if the apparent outcome of one simulations shows a strategy that produces better results, it could be just the result of a unintended mistake.

However, the simulation inspired in us an idea. Maybe the best approach for maintaining the best economy state in a game, and the players happy is to deploy an adaptive approach that changes in relation with the population changes, a combination and a continuously switch of the different strategies deployed according to the evolution of the population – our players.

In a real game, our genetic algorithm used for emulating players is replaced by real players that form a population very similar with the generated one but so different and more complex at many levels. In a virtual game, where the population is no longer simulated by our genetic algorithm, the population changes because

players come and players go. And their happiness can be easily measured by the total amount of the active players population.

The most important conclusion of our experiment is that our simulation can be improved and evolve in a framework that can detect those population changes and try various strategies to increase the population happiness.

More, the player population can be easily segmented into different sets, and different virtual economy rules can be applied to those particular sets like our application is able to run concurrent simulation.

The application can be improved and tested before throwing it into a game populated by human players. It has the potential to become more like a framework for testing different algorithms before releasing them in an environment populated by human players. And the possibilities in this direction are endless, including in the area of adding more and more different combinations of strategies.

We believe that our simulator has the potential to evolve to such a tool mainly by offering more and more freedom to the simulation user through web GUI to define and test its own measures for increasing the wellness of the virtual economy is running on.

Bibliography

1. Almarin P. (1955), *The Tableau Économique as a Simple Leontief Model*, Quarterly Journal of Economics, The MIT Press
2. Armstrong, J. (11 July 2007), *Programming Erlang: Software for a Concurrent World* (1st ed.), Pragmatic Bookshelf
3. Bostock B., (2020), *How 'Professor Lockdown' helped save tens of thousands of lives worldwide and carried COVID-19 into Downing Street*, Business Insider, https://www.businessinsider.com/neil-ferguson-transformed-uk-covid-response-oxford-challenge-imperial-model-2020-4
4. Clement J., (Jun 4, 2020), *Worldwide digital population as of April 2020*, https://www.statista.com/statistics/617136/digital-population-worldwide/
5. Cobb, C. W.; Douglas, P. H. (1928), *A Theory of Production,* American Economic Review
6. Domar, E., (1946), *Capital Expansion, Rate of Growth, and Employment*, Econometrica, Vol. 14, No. 2 (Apr., 1946), pp. 137-147
7. Fischer B.,; Myron S., (1973), *The Pricing of Options and Corporate Liabilities*, *Journal of Political Economy*.
8. Harrod Roy F., (1939), *An Essay in Dynamic Theory*, The Economic Journal, Vol. 49, No. 193 (March 1939), pp. 14-33
9. Hebert F., (2013), *Learn You Some Erlang for Great Good!: A Beginner's Guide*, No Starch Press
10. Muth, John F., (1961), *Rational Expectations and the Theory of Price Movements*, Econometrica
11. Solow, Robert M., (February 1956), *A contribution to the theory of economic growth*, Quarterly Journal of Economics.

12. Thompson, Simon J.; Cesarini, Francesco (19 June 2009), *Erlang Programming: A Concurrent Approach to Software Development*. O'Reilly Media
13. Nazir M.; Siu Man Lui (2016), *A Brief History of Virtual Economy, Journal of Virtual Worlds Research*, Volume 9, Number 1, April, 2016
14. Lehtinen L.; Kuorikoski J., (2007), *Computing the Perfect Model: Why Do Economists Shun Simulation?,* Philosophy of Science, The University of Chicago Press, Vol. 74, No. 3, July 2007
15. Myerson, Roger B. (1991) - *Game Theory: Analysis of Conflict*, Harvard University Press
16. Hamzelou J. (March 23, 2020), *UK's scientific advice on coronavirus is a cause for concern*, New Scientist, https://www.newscientist.com/article/2238186-uks-scientific-advice-on-coronavirus-is-a-cause-for-concern/
17. Metropolis, N. (1987). *The beginning of the Monte Carlo method*, Los Alamos Science.
18. Neumann J., (1928), *Zur Theorie der Gesellschaftsspiele*, Mathematische Annalen
19. Neumann J, (1959), *On the Theory of Games of Strategy*.
20. Friedman, D., (1998), *On economic applications of evolutionary game theory*, Journal of Evolutionary Economics.
21. Logan, Martin; Merritt, Eric; Carlsson, Richard, (28 May 2010), *Erlang and OTP in Action*, CT: Manning Publications
22. Castronova, Edward, (2005) *Synthetic Worlds: The Business and Culture of Online Games*, University Of Chicago Press.
23. Tomić, Nenad Zoran, (January 2019), *Economic model of microtransactions in video games*, Journal of Economic Science Research.

24. Gilbert, David (May 7, 2014), *Eve Online: Meet the Man Controlling the $18 Million Space Economy*, International Business Times.
25. Gilbert, Dan; Whitehead, James; Whitehead, James II (2007), *Hacking World of Warcraft*, John Wiley & Sons
26. Moore, Christopher (February 2011), *Hats of Affect: A Study of Affect, Achievements and Hats in Team Fortress 2*, Game Studies.
27. Myerson, Roger B. (1991), *Game Theory: Analysis of Conflict*, Harvard University Press