Historia Documentation

Kaelan Cooter

Game AI CS4150 - Final Project Documentation

Introduction

Historia is an AI simulation that aims to replicate the basic economic forces that drive an entire country. The end goal (outside of this class) is to have randomly generated and evolving countries and cultures interacting. The base layer of this simulation is what was done for this class project.

This “base layer” is the economic simulation underpinning each country which is composed of an agent-based market simulation.

This project used a pre-existing project of mine called *Hexgen* to generate random world maps. As this is not part of this project, most development took part on one randomly generated world map file which is provided with the project.

Progress so Far

- Two example Countries are generated next to each other. Every month they gain more territory
- Pops trade between other Pops in the same Province.
- If a Pop is not successful at trading, they are fined for being idle.
- Bankrupt Pops will switch to a new job.
- Successful Pops will increase in population, unsuccessful Pops will decrease the same amount.

Work Left to Be Done

After this class, I will continue to work on this project. Most notably, there is more work to be done on the market simulation. Currently, the Pops do not trade with neighboring Provinces. This means that a Province without Trees won't be able to produce Timber, which is a basic building block of the economy. This trading mechanic would be implemented by allowing Pops to trade with neighboring Provinces. I could have a special Merchant pop type be tasked with fulfilling orders between provinces.

The Country AI will be expanded. Settlement of new Provinces shouldn't create Pops out of thin air, it should rely upon Pops migrating to the new Province.

The Population number of each Pop should impact how much Goods they produce. Since I didn't implement inter-province trading, I didn't feel this was necessary just yet.

When Pops go bankrupt, they get $2 out of thin air. Eventually, I want to have special Banker pops lend money with interest to bankrupt Pops.

Currently, resources are infinite. I want to implement a form of land ownership system that would be traded just like a good (e.g. 1 hectare selling for $15.24).

Taxes could be implemented easily, but it wasn't the focus of this stage of the project since it's part of the Country AI. Similarly, tariffs could be implemented as a triggered event when the Country AI notices that trade with a foreign country is impacting its own Pops.

Problems Encountered

Implementing the market simulation was more time consuming than I originally thought. The original plan was also to implement the country AI, but as discussed with the professor I switched my focus to just the Pop simulation.

Additionally, trading between Provinces was deemed too buggy and difficult to get done within the time parameters. I wasted a lot of time going down one route involving trading routes that ended up not being ideal. Late in the project I determined that this feature would best be implemented as a *mesh network*.

Terminology

- **Hex** - The basic unit of the hexagon grid world map generated by Hexgen.
- **Country** - A grouping of Provinces a name.
- **Province** - Exist on a Hex if they're owned by a Country. Pops live in the Provinces. Each Province has a Market.
- **Market** - Where the Pops go to trade Goods
- **Pop** - A unit of population. Each Pop has a Job, saved cash, an inventory, and a population number.
- **Good** - A tradable commodity (e.g. Grain, Bread, Iron, Tools, etc)
- **Order** - A request to buy or sell goods at a market.
- **Price Belief** - The range of prices (high and low) that each Pop believes is true for each Good

- **Natural Resource** - If a Hex has a natural resource, then the Pop can produce it's good. (e.g. Trees allow Woodcutters to produce Timber)

Architecture

The project exists as two distinct parts. There is a Python command line project called *Historia* which generates a large JSON file containing the history of the simulation. This file contains all the data about each Country, Province, and Pop at every day of the simulation. Currently the simulation runs for 100 days, but that's just because anything over that will become too large. Eventually, I hope to switch to a server architecture that would generate the next day on the fly and then save it.

The next part is the user interface called *Explorer*. This was built especially for this project using Javascript and the React library. The runtime packager is Electron, which allows Explorer to (eventually) work on all three major operating systems. For now, it just runs on OSX.

Algorithms & Mechanics

The simulation revolves around days. Currently it runs for 100 days, but that's just because of file size and generation time restraints. The entire simulation runs in a loop over each of the days. The generated data is saved in a JSON file in your home directory, where it is read by Explorer.

Pseudo-code

```
For each Day:

    For each Country in the world:

        For each Market in Country:

            For each Pop in Market (in the Province):

                Perform Pop production (depending on Job)

                Create buy and sell orders for each Good

            Resolve all buy and sell orders for each Good

            For each Pop that has 0 money:

                Find them a new job and reset their money

        Settle new Province (sometimes)
```

Pop Jobs

Each Pop has a job. When Pops go bankrupt, the change jobs. All Pops except Bakers have 10% chance for consuming Tools each time they perform production.

- Farmer - Produces Grain, consumes Timber and Bread
- Miner - Produces Iron Ore, consumes Bread
- Miller - Produces Lumber, consumes Timber and Bread
- Woodcutter - Produces Timber, consumes Bread
- Blacksmith - Produces Tools, consumes Iron and Lumber
- Refiner - Produces Iron, consumes Iron Ore and Bread
- Baker - Produces Bread, consumes Grain

Buy and Sell Orders

After a Pop performs production, it will have surplus inventory that needs to be sold.

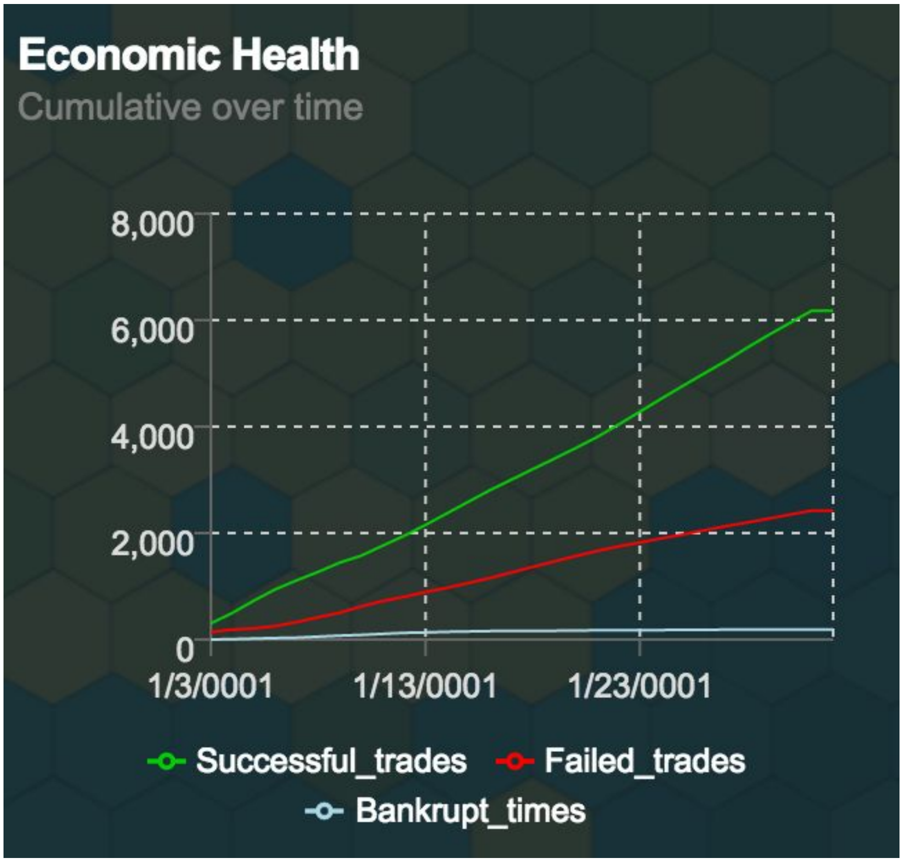
Additionally, depending on their job they need to buy more materials in order to continue producing. They create buy orders and sell orders accordingly.

Buy and Sell orders are matched in the resolve step. The highest price buy orders are matched with the lowest price sell orders. If quantity is left over, the order stays in the market. If there is no quantity left, then it is removed. Money and Goods are exchanged accordingly. All other orders are thrown out.

After a successful or failed buy or sell order, each Pop involved will update their price belief for the Good involved in the order. Overall, the price belief will be shifted to the current market price of the Good. That price, of course, is dependent on the belief of every other Pop. Overall, successful Pops will have a better idea of the correct price.

Analysis

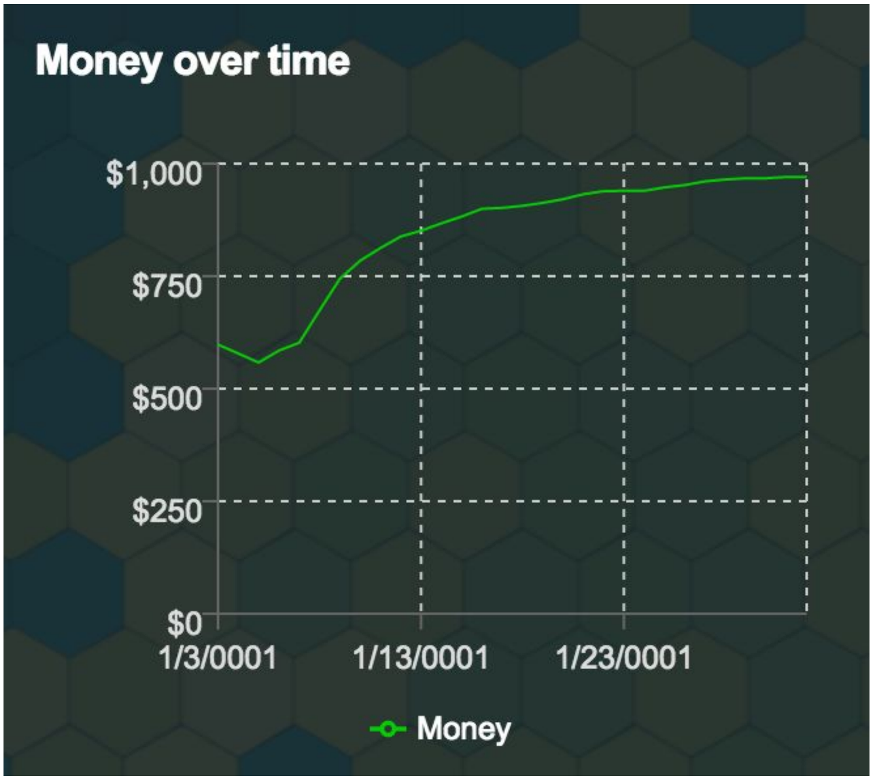
The Province screen shows numerous analytical data about the state of the trading simulation at the currently selected day.



Economic health of a capital Province February 1st, Year 1

In this graph, the green line represents the total successful trades between Pops in the Province for the given day. Successful trades happen when a buy and sell order of similar price are matched together. The red line represents failed trades. Observe how over time, the Pops get better at trading, and succeed at trading more often than failing.

The number of bankruptcies increase over time, but the number of pops going bankrupt decrease. This is because Pops don't move between Provinces, and the Market is a closed loop. If Pops were allowed to move between Provinces, the simulation would balance further.



Total capital savings of all Pops in the same Province

This graph is also important to illustrate how over time, Pops get more rich. This graph isn't as meaningful as the last one, since prices of Goods could tank in a Province and make it look like the total amount of money is decreasing. This is because this graph isn't adjusted for inflation.

Testing

On the Python side, the build-in unittest module was used to create unit tests.

No testing was done on the Javascript side as front-end Javascript testing is time consuming and I didn't see any benefit.

Tools

I used the Atom text editor. I didn’t use an engine or any library for the AI algorithms.

User’s Manual

ZIP archive of json files needed to run / view the Project.

https://drive.google.com/open?id=0ByHYBDe_y_3ccDhmdmVzdTlIMHM

Production Installation (OSX)

1. Download the Historia file and place it in your home directory at `~/historia.json`.
2. Download the OSX (64bit) version of Explorer: (warning: large file)

   https://drive.google.com/file/d/0ByHYBDe_y_3cX1hIWG90ZW5RRkU/view?usp=sharing
3. Extract the archive
4. Run Explorer.app

Development Installation

For Historia

1. Download the repository:

   https://github.com/eranimo/historia
2. Install **PyPy3 2.4.0**

http://pypy.org/download.html

3. (optional) Create a virtualenv with this version of PyPy. Activate the virutalenv. https://virtualenv.pypa.io/en/latest/
4. Run `pip install -r requirements-r.txt`. This will install all the Python dependencies for the project.
5. Download the Hexgen file and place it at `~/hexgen.json`.
6. Run python bin/example.py to run Historia and export a JSON file to your home directory.

Running notes:

- If you encounter an error, run it again
- You can turn on debug mode in bin/example.py, but it will be much slower.

For Explorer

1. Download the repository: https://github.com/eranimo/explorer
2. Install NodeJS v4.4.3 or above https://nodejs.org/en/download/
3. In the repo folder, run npm install. This will install the nodejs dependencies.
4. Run npm run hot-server to run the development environment.
5. Run npm run start-hot to start Explorer in development mode.

To package a new version of Explorer:

1. Run `npm run build`

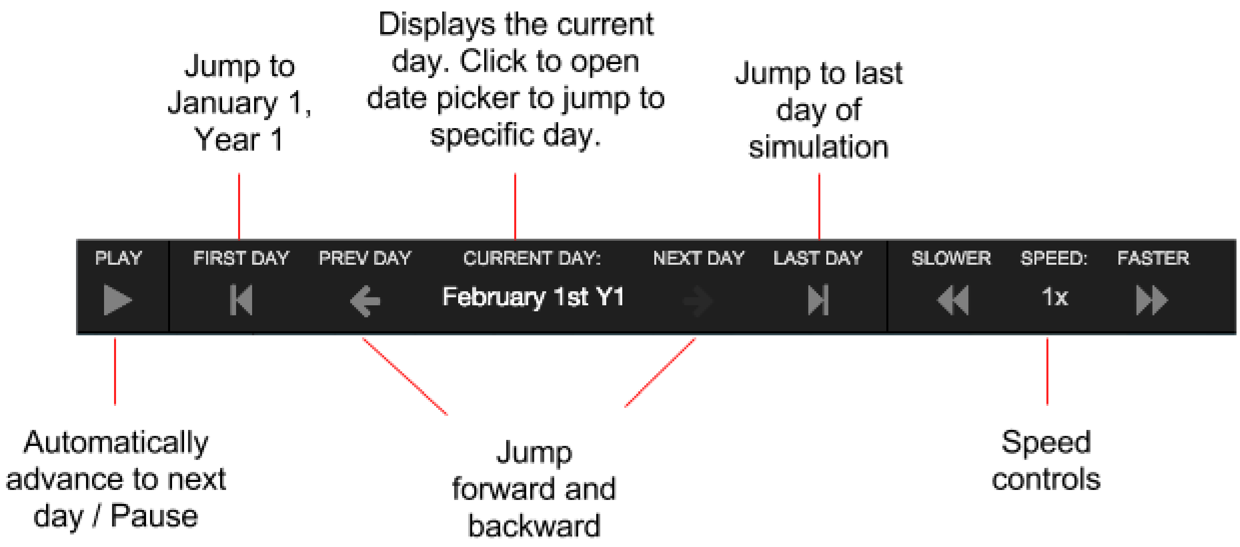
2. Run `npm run package` (to package for your current os)

   or run `npm run package-all` to package for all operating systems.

User Interface

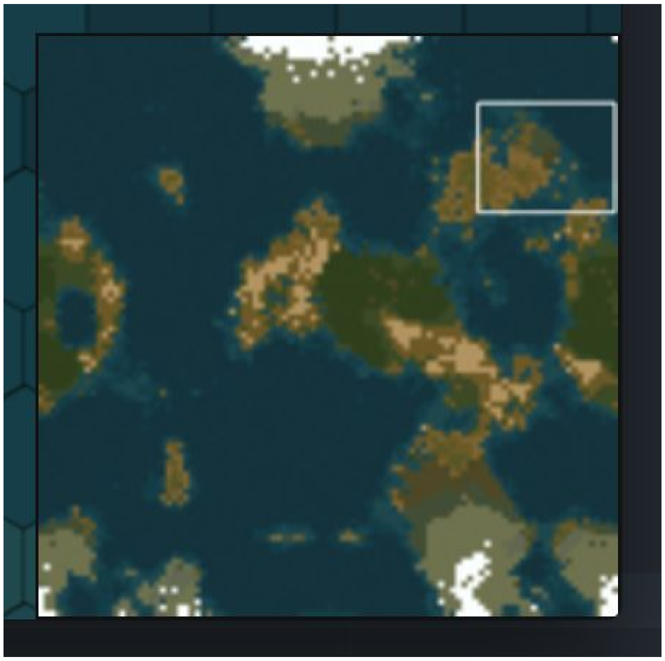
Time Controls

You can change the current day of the simulation as well as control how fast time progresses from the time control section on the top part of the app.


Jump to January 1, Year 1
Displays the current day. Click to open date picker to jump to specific day.
Jump to last day of simulation
PLAY
FIRST DAY
PREV DAY
CURRENT DAY:
February 1st Y1
NEXT DAY
LAST DAY
SLOWER
SPEED:
1x
FASTER
Automatically advance to next day / Pause
Jump forward and backward
Speed controls

Minimap

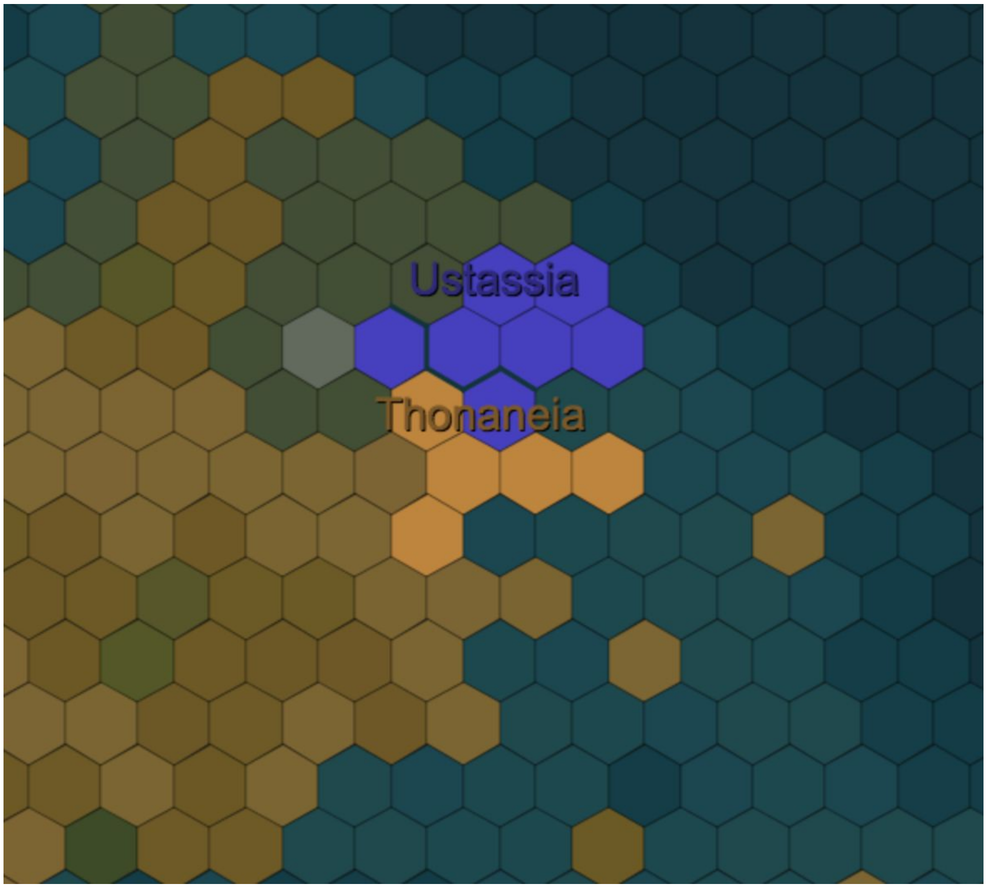
The minimap allows the user to quickly jump to a part of the world map. Currently, the only two countries in the simulation are in the richest part (climatewise) of the world in the area enclosed by the white box in the following image.



Map

The map is the main user experience of Explorer. It allows you to navigate around the world in the current day. All Countries and Provinces will be visible on the map. Country names will appear over its territory.

A large part of the map code was reused from a previous project, but I did a lot of work to improve its performance.



Controls

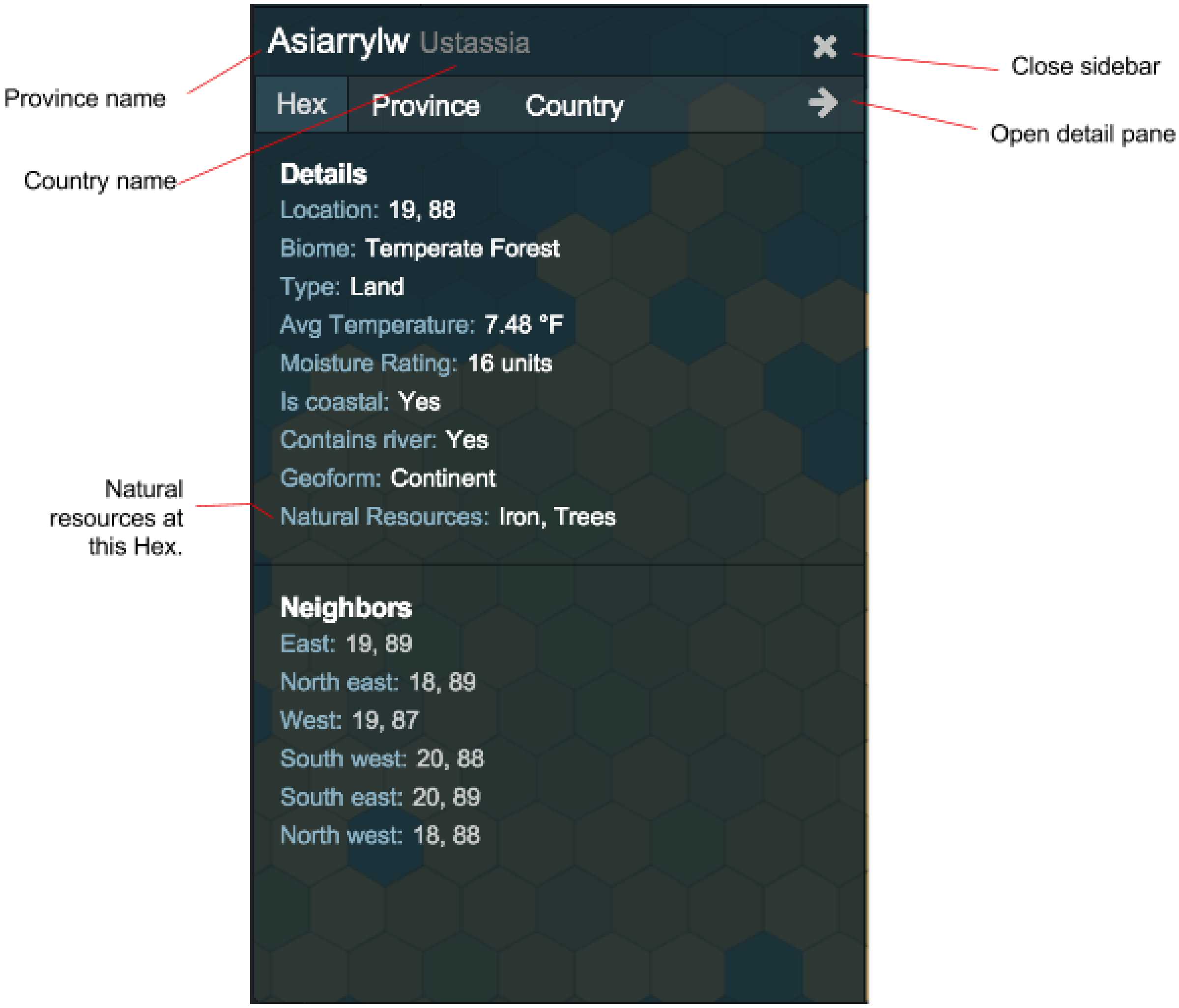
- **Panning** - Click and hold anywhere in the map. Drag the mouse to drag the map with the mouse.
- **Zooming** - Use the mousewheel. Up is to zoom in, down is to zoom out. Press spacebar to return to the default zoom level.

Selecting a Hex

Click on a hex to select a hex. This opens the sidebar detail for that Hex.

Sidebar

The sidebar is where most of the information about the world happens. If the currently selected hex contains a Province, a tab will appear with information about it.

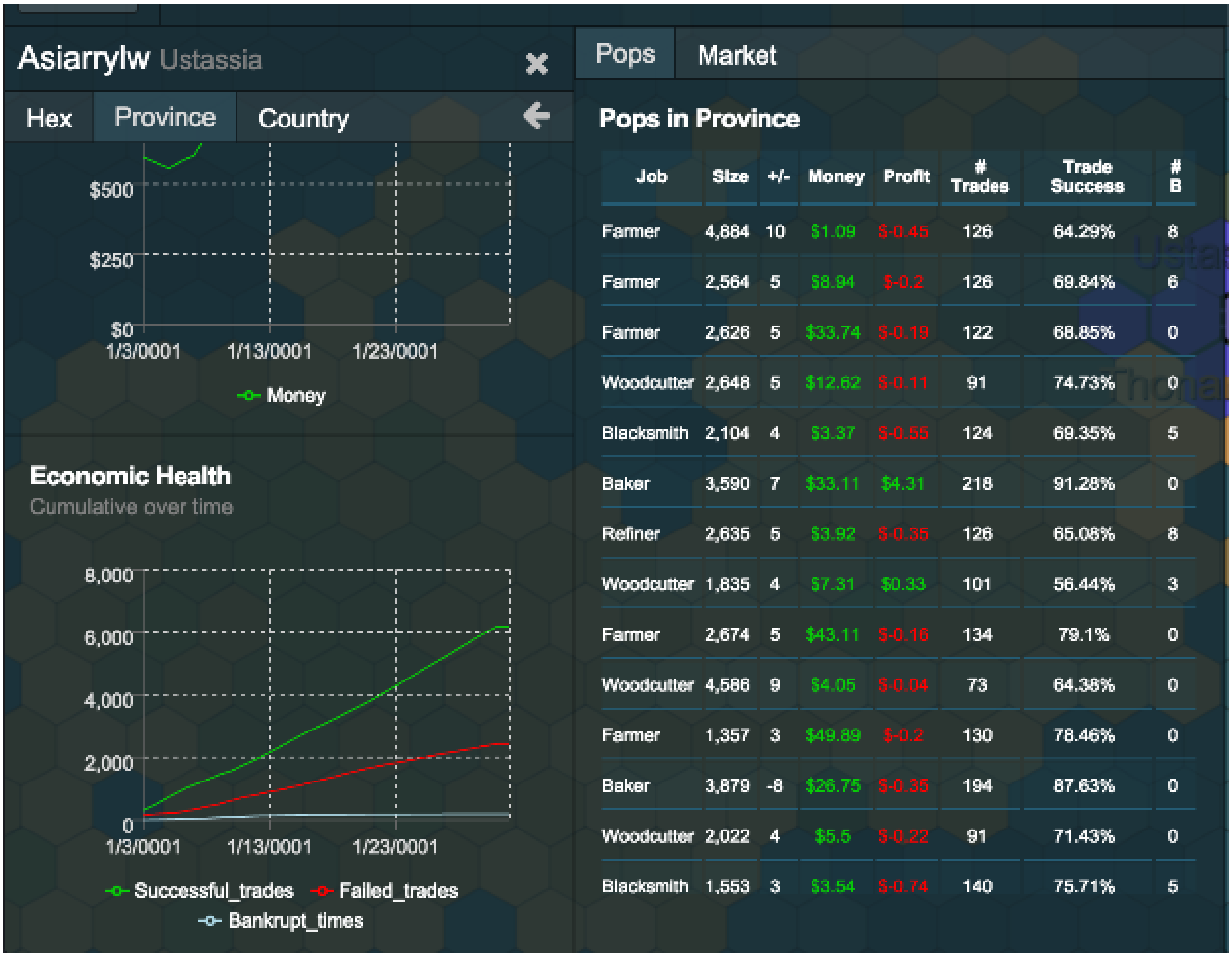


Details
Location: 19, 88
Biome: Temperate Forest
Type: Land
Avg Temperature: 7.48 °F
Moisture Rating: 16 units
Is coastal: Yes
Contains river: Yes
Geoform: Continent
Natural Resources: Iron, Trees

Neighbors
East: 19, 89
North east: 18, 89
West: 19, 87
South west: 20, 88
South east: 20, 89
North west: 18, 88

Hex tab showing details about the currently selected Hex

On this tab you can jump to neighboring hexes easily, and see climate information from the Hexgen project that impacts how Historia works.



Pops in Province

Job	Size	+/-	Money	Profit	# Trades	Trade Success	# B
Farmer	4,884	10	$1.09	$-0.45	126	64.29%	8
Farmer	2,564	5	$8.94	$-0.2	126	69.84%	6
Farmer	2,626	5	$33.74	$-0.19	122	68.85%	0
Woodcutter	2,648	5	$12.62	$-0.11	91	74.73%	0
Blacksmith	2,104	4	$3.37	$-0.55	124	69.35%	5
Baker	3,590	7	$33.11	$4.31	218	91.28%	0
Refiner	2,635	5	$3.92	$-0.35	126	65.08%	8
Woodcutter	1,835	4	$7.31	$0.33	101	56.44%	3
Farmer	2,674	5	$43.11	$-0.16	134	79.1%	0
Woodcutter	4,586	9	$4.05	$-0.04	73	64.38%	0
Farmer	1,357	3	$49.89	$-0.2	130	78.46%	0
Baker	3,879	-8	$26.75	$-0.35	194	87.63%	0
Woodcutter	2,022	4	$5.5	$-0.22	91	71.43%	0
Blacksmith	1,553	3	$3.54	$-0.74	140	75.71%	5

Province tab showing economic information.

The Market tab (not shown) has a table and charts for the price and quantity of each Good in the Market. These two panes are scrollable.

Resetting Explorer

If you encounter a problem. From the Explorer OSX menu item you can select “Reload”. This will reload Explorer.

References

Doran, Jonathon, and Ian Parberry. "Emergent Economies for Role Playing Games." Laboratory for Recreational Computing Department of Computer Science & Engineering University of North Texas, June 2010. Web. 30 Mar. 2016. <http://larc.unt.edu/techreports/LARC-2010-03.pdf>.

Kendall, Graham, and Yan Su. "The Co-evolution of Trading Strategies in A Multi-agent Based Simulated Stock Market Through the Integration of Individual Learning and Social Learning." University of Nottingham. Web. 30 Mar. 2016. <http://www.eecs.harvard.edu/~parkes/cs286r/spring08/reading6/icmla03.pdf>.

Dinther, Clemens Van. "Agent-based Simulation for Research in Economics." Research Center for Information Technologies at Universität Karlsruhe, 2008. Web. 30 Mar. 2016. <http://www.im.uni-karlsruhe.de/Upload/Publications/7f5a13a7-72ee-4fad-93aa-11f7b98d5227.pdf>.

Acknowledgement

An implementation of the algorithms introduced in “Emergent Economies for Role Playing Games” -- which form the backbone of this project -- called BazaarBot was helpful in filling in some of the gaps from the paper. Specifically, how the price belief system should work was an area that was lacking in detail in the paper.