# A Model of Making – *an ODD Description*

*Bruce Edmonds, Centre for Policy Modelling, Manchester Metropolitan University*

This code of this model and this documentation are freely available [1]. Netlogo to run it is freely available at [2] and the necessary extensions at [3]. Some slides introducing the model [4].

## 1 Overview

### 1.1 Purpose

The purpose of this model is to provide the simulation infrastructure needed in order to model the activity of *making*. That is individuals using resources they can find in their environment plus other things that other individuals might sell or give them, to design, construct and deconstruct items, some of which will be of direct use to themselves, some of which they might sell or give to others and some of which might be used as a tool to help in these activities. It explicitly represents plans and complex objects as separate entities in the model – embedding the "Atoms – Bits" distinction highlighted within the DiDIY project. This allows plans to be shared between agents which give the steps of how to make objects of use – either on a commercial or a free basis.

The framework is intended as a basis upon which many, more specific, models could be constructed, allowing the exploration of a variety of "what if" or counterfactual possibilities and thus give a concrete but dynamic and complex instantiation of the issues and situations discussed within the DiDIY project. In a sense this model is a "bits" representation of the ideas discussed – hopefully these will converge!

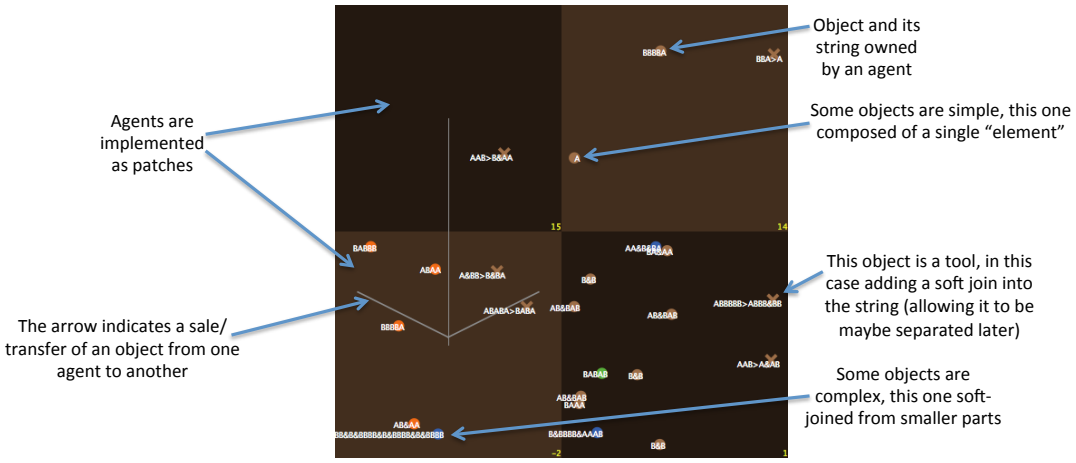


**Figure 1. A view of the model with agents, objects and a sale**

### 1.2 Entities, State Variables, Scales

There are three main kinds of entity in the model, agents, things and plans:

1. **Agents**. There are a fixed number of agents that do the making and decision making in the model. These are individually represented as "patches" in the model. They own and hold things. They can (depending on the nature of the things) act upon these things to make new things. At the moment their position is not important and they can swap/trade things with any other agent. They also hold in their mind a number of plans which they have either learnt themselves (through trial and error) or obtained form another agent.
2. **Things**. Things are individually represented and tracked within the model (from creation to destruction). They each have the nature of a 1D string of symbols, composed of a fixed number of "elements" (the letters "A", "B" etc. depending on the parameter which determines the number of elements) and the two symbols "&" and ">". "&" indicates a soft join, that is a join that an agent can make (or break) without a specific tool. ">" indicates that the item can also be used as a tool, that is it can be used to "transform" the string on the left of the ">" into the one on the right in another string.
3. **Plans**. Agents remember sequences of actions they used to construct/deconstruct strings and the cost/benefit of the result as explicit plans. These plans are a tree structure of actions and the strings that resulted. The ability to remember these plans allows agents to repeat successful plans and also allows the possibility of plans being shared/licensed between agents.

The world of 1D strings is sufficiently complex to make the process of working out what sequences of actions would result in which valuable strings is a hard problem. Which strings are available in the environment and which strings have inherent "use" value are randomly determined at the start of the simulation. Which subset of strings are available to each agent and which subset can be redeemed by each agent can be varied, so as to be able explore the impact on the heterogeneity of resource availability and agents needs. This hardness is what makes plans valuable and so worth sharing.

## 1.3 Process Overview, Scheduling

The most important process is that of agents picking one of their plans (some of which are default, "*try something random"*, plans) and then doing its steps. The cost/value of the result of doing these plans are remembered (basically value – costs in making the object) so that later plans that are better can be preferentially chosen in this process. Thus this combines some trial-and-error with developing a focus on plans that worked better. The main process is outlined in Figure 2.

Each tick the following happens:

1. Some variables and links are cleared
2. **Main loop**. Each agent does the following in a random order of agents:
   - 2.1. Initialise variables, compile a choice list to speed up plan choice
   - 2.2. *Until* a plan succeeds or a given number of attempts have been exhausted:
     - 2.2.1. Choose a target string from plan memory, find a (given) number of plans from the memory that would result in this string
     - 2.2.2. Try these plans one at a time until one succeeds or they are exhausted:
       - 2.2.2.1. *If* plan seems to be possible to do, then do plan
   - 2.3. If are remembering too many plans, maybe forget one with a bias towards the least valuable
3. Update display, clear dead objects from can-buy lists, do stats and timing

**Figure 2. The main process**

# 2 Design Concepts

## 2.1 Basic principles

In this model, I aim to represent the process of making as directly as possible, given a universe of objects and tasks that allow for this to be meaningfully complex, so that communicating plans is worthwhile, and that there is motivation for trade and/or sharing objects.

Objects are represented individually and explicitly. Some objects can be extracted from the environment by some agents at a cost (depending on their composition). Some objects can be directly used by some agents so they gain value (depending on their composition). Objects can be soft-joined using an "&" (e.g. A and B to A&B) or split at a "&" (e.g. B&AA to B and AA). Some objects can be used to transform other objects, those with a ">" in them (e.g. AB>BA used once on the object ABB would result in BAB. Objects can be passed from one agent to another agent but can only be owned by a single agent at a time. Once used up they disappear from the simulation.

Plans are mental representations of the steps needed to construct/deconstruct strings. They reside in the agent's memory. They can be communicated with other agents, potentially either freely (following some license conditions) or on a commercial basis.

## 2.2 Emergence

Patterns of use and of buying and selling emerge as the simulation progresses.

## 2.3 Adaptation

Agents gradually adapt to what kinds of things are available from their environment, what they can buy and sell and with whom, and what they can directly utilise.

## 2.4 Objectives

At the moment, agents effectively learn to create the greatest value for the least cost, in terms of their own needs or sale values because plans with a higher value are preferentially tested for viability and used. The parameter, prop-targets-each, controls the heterogeneity of agent values, because if an agent cannot directly redeem a string its only value to them will be if they can sell it to another or, possibly, use the string in further construction/deconstruction.

## 2.5 Learning

Agents slowly build up more complex plans and learn which are the most valuable to perform.

## 2.6 Prediction

Agents assess possible plans and predict whether they think the plan is possible to complete as well as the resultant value/cost of doing so.

## 2.7 Sensing

Agents can sense what objects they have, as well as what objects they might be able to buy from others.

## 2.8 Interaction

At the moment, interaction is through the buying and selling of objects. In the future interaction will include the communication of plans.

## 2.9 Stochasticity

There are 4 main sources of stochasticity in the model.

1. The initialisation in terms of what kinds of object are available from the environment or usable by agents (see section 3.1).
2. Each tick each agent picks plans to do probabilistically, with a bias towards the more successful plans. If the plan is one of the default "random" plans this will include choosing random objects to do the actions with.
3. If an agent has too many plans in its memory it will probabilistically forget a plan with a bias towards those that are less successful.

## 2.10 Collectives

There are currently no collectives in the model, though there is the potential for collectives to form to collectively "manufacture" items or share tools and plans. Also various market structures and alliances would be simply implementable.

## 2.11 Observation

The statistics that the model currently generates is described in section 4.4.

# 3 Details

## 3.1 Initialization

The number of agents are fixed and are initialised as "patches" on a 2D grid. Each agent is initialised with the same "default" plans, which are given values. At the moment these plans and value pairs are: [-2 "get-random"] [-0.75 "apply-random"] [-0.5 "realise-random"] [-0.5 "sell-random"] [-2 "buy-random"] [-1.5 "split-random"] [-1 "join-random"], where [-2 "get-random"] means extract a possible string from the environment using the notional value for the plan of -2 when comparing it with other plans.

The main variation in terms of initialisation comes in what resources and redeemable target strings are available. The total "menu" of resources and targets are determined as follows.

***Environmental Resources***. This is a list of strings that can be got from the environment and the cost of doing so. This is determined as follows:

1. The basic "elements" (characters not "&" or ">") are given, the number set by parameter
2. Random strings using these elements are constructed with a length determined by a Poisson distribution whose mean is given by parameter
3. A given proportion of these strings have a soft-join (a "&") inserted with a given probability
4. An additional number of random tools are generated with the same length distribution
5. All of these are then attributed a random cost picked from a Poisson distribution with a given mean (if the item is available to an agent in terms of extraction from the environment, then it is with this cost it is extractable)
6. Each agent is then allocated access to a random selection of these, the number determined by a given proportion of the totality

***Target Affordances***. To ground the value of strings for agents some strings are allocated a redemption value – that is, an agent can get this value by "consuming" this string. Again, like resources, agents may have a different selection of such strings, representing different needs and goals, but always with the same value. The process for initialising these targets and their values are as follows.

1. The basic "elements" (characters not "&" or ">") are given, the number set by parameter
2. Random strings using these elements are constructed with a length determined by a Poisson distribution whose mean is given by parameter
3. All of these are then attributed a random value picked from a Poisson distribution with a given mean (if an agent is able to directly redeem this string, then this is the redemption value)
4. Each agent is then allocated access to a random selection of these, the number determined by a given proportion of the totality

Thus agents have a hard problem, how to make strings that they can redeem/sell given the strings that they have as a resource/buy.

For example the list of strings that might be extractable from the environment might be: A; A>; AA; AB; B>; BA; BB; A&A; A&B; AAA; AB>BA … and the list that is

redeemable in terms of value be: AB; A&B; AAA; AAB; ABA; B&A; BBA; BBB; A&AA …. Inspecting these one can see that A&B might simply be both extractable and consumable; A&AA could be made by joining A and AA, both of which might be available in the environment; to make B&A one would have to obtain A&B, split it apart and rejoin it as B&A; and to make AB one might have to apply the tool AB>BA on the item AB. Making items of high value might require a very complex series of steps, and some maybe not possible to make. This is more difficult because each agent will only be able extract a different subset of resources and redeem a different set, which provides a motivation for commerce and sharing of items.

The parameters that control this environmental framework of resource cost/distribution and target value/distribution determine how hard the making problem is for agents and whether it will be useful for them to trade. A very hard framework would make the discovery of plans that allow the construction of valuable items important, a great heterogeneity in terms of resources encourage trade, but the heterogeneity of targets might encourage the emergence of streamlined production (i.e. manufacturing).

## 3.2 Input Data

At the moment, there is no data input.

## 3.3 Submodels

The key global data structures are as follows:

| Name | Purpose | Fields | Notes |
|---|---|---|---|
| resources | A list of strings that could be available to agents | | Each agent might have only access to a subset of this |
| targets | A list of strings that can be directly realised for value | | Each agent might be only able to realise a subset of this |
| resource-costs | This is a table showing the costs of extracting kinds of resources | str → cost | |
| target-values | This is a global table showing the value of directly realising strings | str → value | |

The key agent data structures within agents, are currently as follows:

| Name | Purpose | Fields | Notes |
|---|---|---|---|
| my-resources | The subset of resources that the agent can access | | |
| my-targets | The subset of targets that the agent can | | |

| | | | |
|---|---|---|---|
| | realise | | |
| my-plans | An agents plan library (a factbase) | str, id, value, plan, plan-hash | and plan is one of the following (recursive) lists<br>* get itm<br>* buy itm agent<br>* realise itm-tree<br>* sell itm-tree agent<br>* apply tl-tree itm-tree<br>* split itm-tree<br>* join itm1-tree itm2-tree<br>* plus the random plans: "" val plan<br>– -1 [buy-random]<br>– -1 [get-random]<br>– 0 [split-random]<br>– 0 [join-random]<br>– 0 [apply-random]<br>– 1 [realise-random]<br>– 1 [sell-random] |
| plan-table | A table to speed up plan location | thing → plan id | |
| for-sale | A factbase of items offered and available for sale, agents must check this before buying | str, itm, price | Is kept up-to-date by selling agent |
| can-buy | A factbase of information about items agent could buy | str, itm, price, agent | This might be fallible, as the item might be sold, but is used by a buying agent to see what might be available |

In a sense, the algorithm of agent learning and decision making is a submodel, but that is described above. Obviously, at this stage this is pretty basic.

In terms of the market in the model, this is pretty basic. Each agent has a "for sale" list of things it would like to sell, with their price. They also keep a (fallible) list of things they might buy derived from others' for sale lists.

### 3.3.1 Parameters

Apart from some debugging options, the model has the following parameters:

- num-agents – sets the number of agents in the simulations

- num-elements – the number of different characters ("A", "B", …) that can be used to construct strings
- num-resources – the number of strings that can be extracted from the environment
- cost-resources – the average cost of obtaining resources from the environment (the distribution is a Poisson with this as the mean)
- len-resources – the average length of resource strings (the distribution is a Poisson with this as the mean)
- prop-resources-each – the proportion of all resources that each agent can access (each agent will get a random selection)
- prop-breaks – the proportion of resources strings with "soft joins" inserted into them ("&")
- prop-nat-tools – the proportion of resource strings that are tools (i.e. with ">" in them)
- num-targets – the number of target (directly redeemable) strings there are
- value-targets – the average value of the targets (the distribution is a Poisson with this as the mean)
- len-targets – the average length of the targets (the distribution is a Poisson with this as the mean)
- prop-targets-each – the proportion of the total list of targets that is randomly made accessible to each agent
- num-plans-remembered – the maximum number of plans that an agent remembers, if this is exceeded then probably some will be discarded
- max-tries – the number of different goal strings (derived from its plans) an agent will consider each tick
- num-alternatives – how many different plans for each goal stings are considered and compared
- tool-use-cost – the cost of using a tool
- choice-bias – the bias towards trying plans that are remembered with a higher value (0 means choice is random, and higher numbers mean that there is less exploration of lower value or random plans)
- dup-discount – the drop in value for a plan that produces an item the agent already has (0 means it does not care about producing duplicates, the higher this number the less likely a duplicate might be made)
- action-cost – the cost of doing any action, thus doing complicated plans costs more than a simple one with the same result (0 means no bias towards simpler plans)
- max-time – the max number of ticks the simulation will run for (0 means no maximum)

There are also a number of debugging options which do not affect the results of the model (see below in section 4.2 for some of these).

# 4 Simulation Outcomes

## 4.1 Model View

The model view is shown in Figure 1. above. Each patch is an agent (patches that are not agents are coloured black). Things are represented on the patches, with their label being the string they represent. The colour of things indicates the last kind of action that was used in making it, according to the following associations:

- Apply – red
- Split-left and split-right – green
- Join – blue
- Get – brown
- Buy – orange
- Sell – magenta (in practice this will not appear, since everything sold is also bought by the recipient)
- Realise – violet (in practice this will not appear, since once it is realised it disappears)

Items that are tools are put to the upper right hand side of patches and displayed as crosses, non-tools to the left hand side as circles. The yellow number in the bottom right hand corner of patches is the (cash) value they have. When an item is sold or transferred an arrow is shown from patch to patch.

## 4.2 Model Output

The main detail of what is happening is shown in the text displayed with the output window to the right. What is shown here depends on the debugging options above this. If debug? is true then the calling of every high-level procedure is shown along with values it is called with. If trace? is true then only the actions of agents are logged. Setting aft-go? to true means both of these only have effect after setup is finished. If any string is entered into the filter-string box then only those lines which include this string are shown, allowing logs that only record what happens to one agent etc. If the pause? option is enabled then the code will pause every time it reaches a "_p" statement in the code, to facilitate step-by-step debugging.

## 4.3 Monitors

There are currently 4 monitors:

- resources – which shows the start of the global list of possible resource strings
- targets – which shows the start of the global list of possible targets for redemption
- s/tick – which shows the average seconds each tick takes
- Num.Th. – which shows the total number of things in existence

## 4.4 Output Statistics

If the stats? option is set, various statistics are generated, for use with the behaviour space. This generates a number of lists of numbers, one number for each agent in the list at any tick. These are lists of:

- num-things-list – the number of things that each agent has
- num-tools-list – the number of tools that each agent has
- num-distinct-things-list – the number of different things that each agent has
- av-length-list – the average length of thing strings that each agent has
- num-get-list – the number of get actions each agent has done
- num-buy-list – the number of buy actions each agent has done
- num-sell-list – the number of sell actions each agent has done
- num-join-list – the number of join actions each agent has done
- num-split-left-list – the number of split-left actions each agent has done
- num-split-right-list – the number of split-right actions each agent has done
- num-apply-list – the number of apply actions each agent has done
- num-realise-list– the number of realise actions each agent has done
- num-plans-list – the number of plans that each agent has
- num-for-sale-list – the humber of sales each agent has achieved
- max-plan-value-list – the value of the most valuable plan that each agent has
- income-list – the income in the last tick that each agent has made/lost
- money-list – the money that each agent has

These can be used in the "Measure runs using these reporters:" box of the BehaviourSpace dialogue, in conjunction with "av", "sd", etc. e.g. av num-things-list, max num-things-list or sd num-plans-list.

## 4.5 Illustrative Results

Figure 1 is a graph of the average number of items realised for different average lengthed targets showing the increasing difficulty of making/finding longer strings. Note the gap between Len=5 and Len=6 since the average length of resources available in all these runs was 2, meaning that finding/constructing strings of a length of 6 and above was hard for agents.

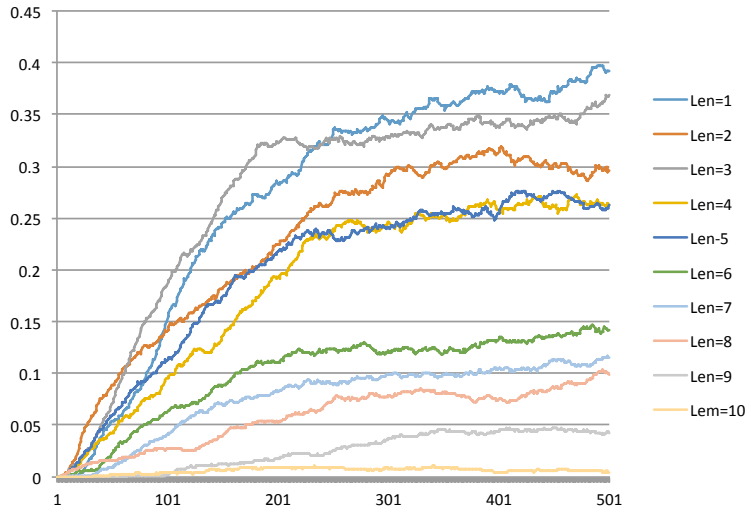


**Figure 3. Average number of items realised for different settings for the average length of target strings (over 10 runs per setting)**

Drilling down we just look at the runs above with Len=2. Below are graphs of the average wealth of agents in these 10 runs (Figure 4) and their spread, measured by standard deviation (Figure 5). These show a general increase in wealth over

time as the agents work out how to make/get valuable strings, but these differ a lot in terms of the inequality in wealth that results.

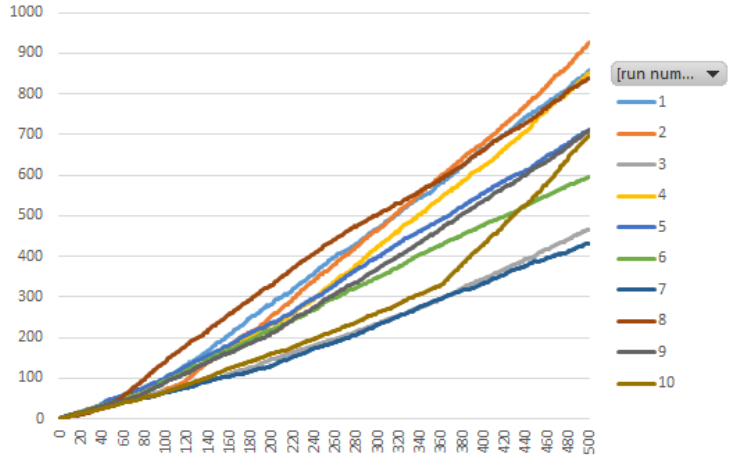


**Figure 4. Average wealth of agents for 10 runs (for av. Len=2 for targets)**

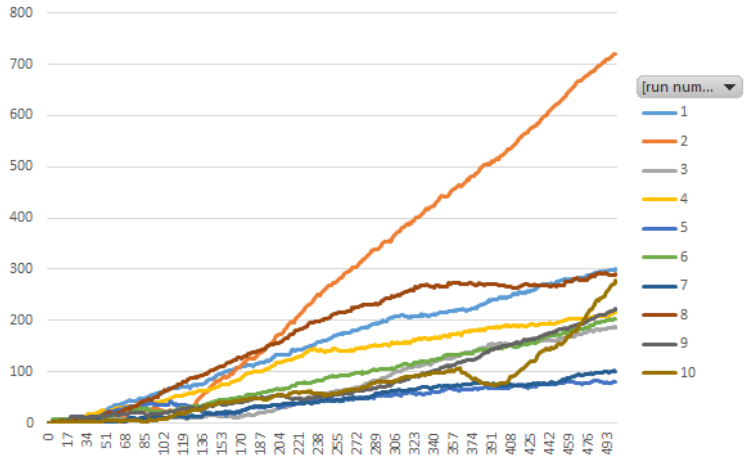


**Figure 5. Standard Deviation of the same runs as immediately above.**

We now drill down to a single run, in terms of run 10 in the graphs immediately above. The following figures show a summary of a number of measures for this run, over time, all scaled so their maximums are 1 (so they all fit on the same graph), and some are smoothed to make the trends more visible.

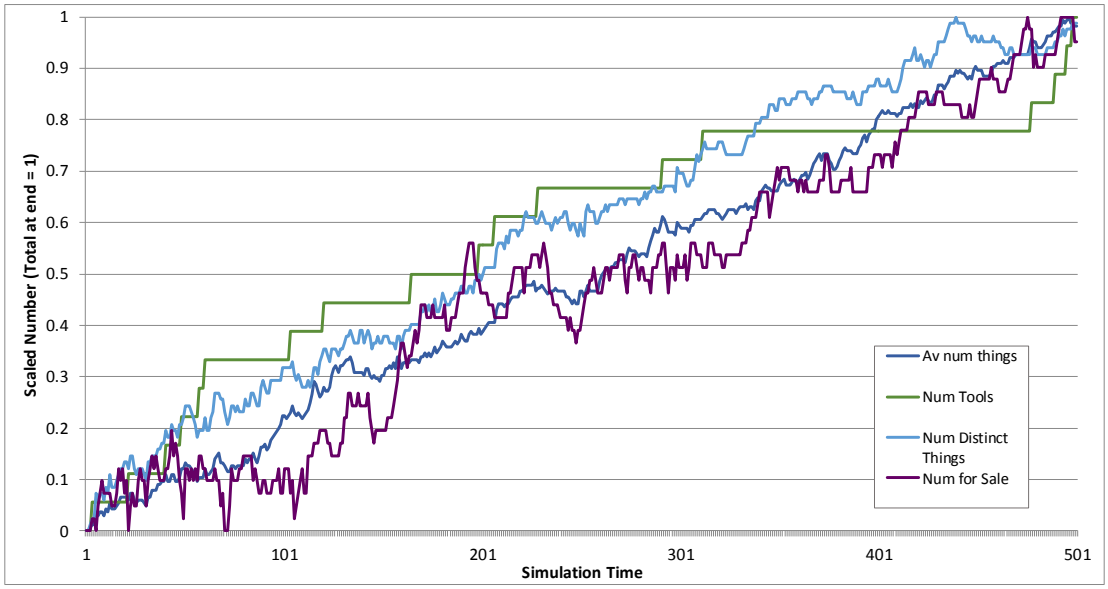


**Figure 6. Number of things, distinct things, tools and items for sale in a single run of the model (Run 10 in the graphs immediately above), scaled so the maximum of all measures is 1. (Some of these lines are exponentially smoothed to ease viewing)**

The number of things, the number of distinct things, and the number of tools (Figure 7). This indicates that agents are learning to make new things and gradually discover new tools. The number of things for sale also increases, albeit not smoothly. This shows the development of an informal market in items as the simulation progresses.

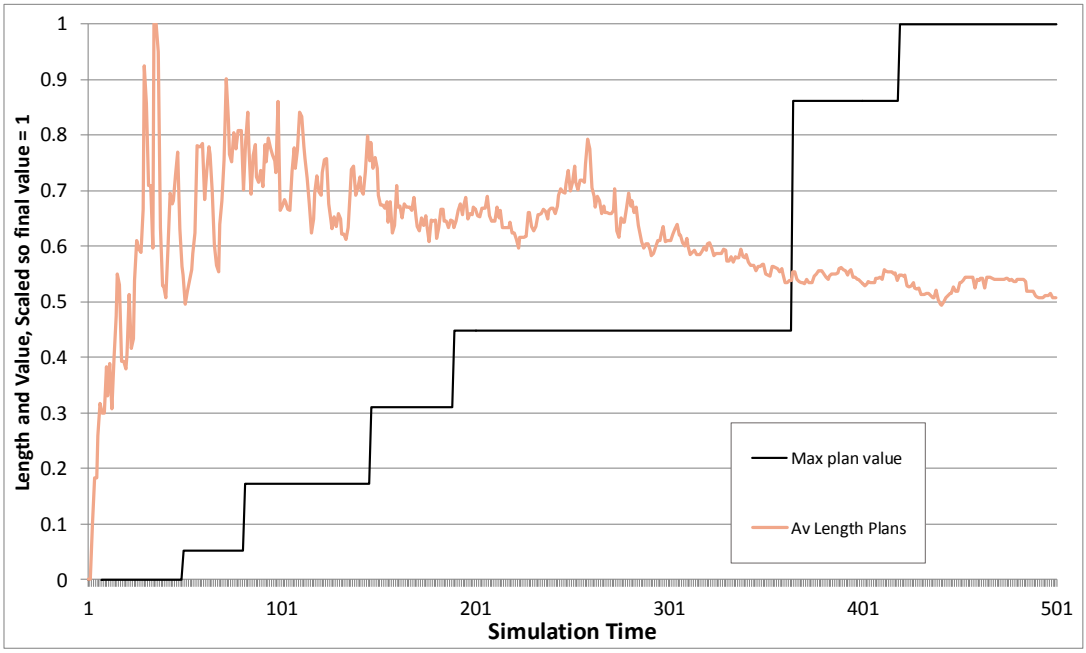


**Figure 7. The average length of plan and the greatest plan value in the same single run of the model (Run 10 in the graphs immediately above), scaled so the maximum of all measures is 1.**

The average length of agent plans increases rapidly at the start as random exploration of plans seeks for useful strings, but after this is a gradual reduction

as more efficient plans (with lower costs) are discovered Figure 7. It also shows the value of the most valuable plan. Around the tick 360 an agent discovers a much more valuable plan than previously existed – a step change in its ability to create value.

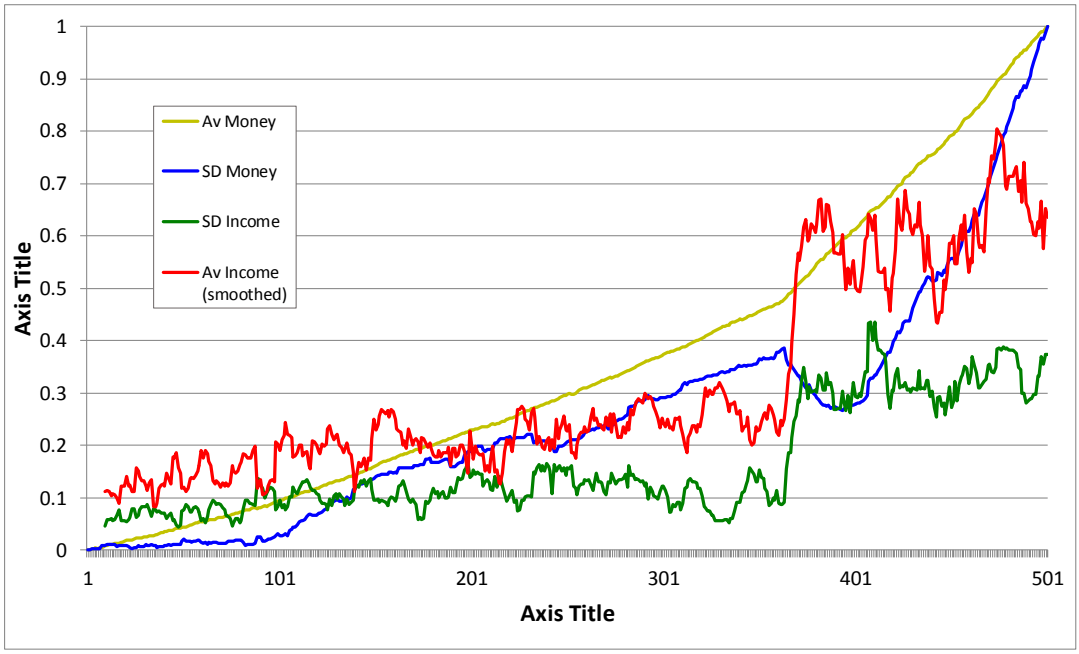


**Figure 8. The average money, the standard deviation of the money, the average income and the standard deviation of income of agents in the same single run of the model (Run 10 in the graphs immediately above), scaled so the maximum of all measures is 1. (income lines are exponentially smoothed to ease viewing)**

This discovery around tick 360 has an impact upon incomes and accumulated value of agents (Figure 8). It seems to cause a sudden increase in both income and spread of income but a temporary drop in the spread of wealth before these then increases sharply – it seems that the discovery was made by a previously (relatively) unsuccessful agent that then overtook the old wealth leaders. After this event, the rate of wealth accumulation increased.

In this model, the potential for technological advancement is implicit in the affordances built into the making possibilities – this is built in to the model. The ability to try actions and learn what seems to work, according to the motivations provided to the agents is also built in. However the discovery of what works by the agents, and what they discover is largely a matter of chance. The inequality in terms of value accumulated is something that emerged in the runs that were done. Similarly the kind of market that emerges in terms of buying and selling is a macro-level outcome form the model – the micro-level actions and learning of all the agents combines to produce it.

## 4.6 Sensitivity Analysis

TBC

## 5 Acknowledgements

Paper developed under the DiDIY Project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644344. The views expressed in this paper do not necessarily reflect the views of the EC. More information about this project can be found at http://didiy.eu

## 6 References

[1] Edmonds, Bruce (2016). "A Model of Making". CoMSES Computational Model Library. Retrieved from: http://www.openabm.org/model/4871

[2] Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[3] Netlogo Extensions, http://github.com/NetLogo/NetLogo/wiki/Extensions, accessed 5th Feb 2016.

[4] Edmonds, B. (2016) A Model of Making, Slides from the DiDIY Project meeting, Thessaloniki, Greece. http://slideshare.net/BruceEdmonds/a-model-of-making