Simulation of an autonomous entity; or how to make an NPC seem alive.

Olivier Thill

A thesis submitted for the degree of Master of Science in Advanced Computer Science

Supervisor: Dr. Sam Steel
School of Computer Science and Electronic Engineering
University of Essex

August 2016

Student: Olivier Thill

Title: Simulation of an autonomous entity; or how to make an NPC seem alive.

A dissertation for the pursuit of a Master in Advanced Computer Science

Supervisor: Dr Sam Steel

Second Assessor: Dr Diego Perez

Date: 25/08/2016

Abstract

This dissertation first attempts to define concisely what a lifelike autonomous entity is, as there is no one authoritative definition for concepts such as life, autonomy or entity. What this dissertation finds, is that at least one way to define a lifelike autonomous entity is to describe them as self-contained things which continuously act on their own volition for the purpose of their survival, when survival is achieved by what can be described as acquiring resources both tangible and intangible.

Secondly, based on these insights, a model to simulate such lifelike autonomous entities is developed, designed and implemented. The main features of the model are simplicity and generality. In the model, states and resources are generalised into one single representation. Thus every part of the entity's state can be manipulated through the same mechanism, without the entity needing to know what the components of its state mean in the context of the domain of the simulation.

Lastly the results of simulations run in an implementation of the model are presented and are then used to discuss the apparent strengths and weaknesses of the model, as well as how practical this model is for simulating autonomous lifelike entities.

I would like to thank

My supervisor, Dr Sam Steel,
for his support to my work and insights provided.

My friends and family for their support in general.

And Marianne Jade Buffat in particular,
for accepting the doomed task of correcting my grammar.

Contents

This is a dissertation for a master in the computer sciences entitled:

“Simulation of an autonomous entity; or how to make an NPC seem alive”

In this dissertation, the notion of an autonomous lifelike entity is first discussed, and then based upon this a model for such an entity is developed.

Parallel to this dissertation an implementation of the developed model has been produced, and some results of simulations run using that implementation are also presented here.

Table of Contents

1. Introduction

There are two concepts which are important to this dissertation. The first concept is what a lifelike autonomous entity is. It will be shown that one definition for such an entity is a thing which exists on its own, makes its own decisions and uses intelligent reasoning to follow a purpose. The second important concept follows the first. What is the purpose of an autonomous lifelike entity, what does it want?

1.1. What is an entity

An entity is something which has a function and is complete on its own. Different contexts however often change what 'complete' means. Thus, while talking about a room then the furniture inside it might be considered entities, but the room itself is thought of as system rather than as an entity. If talking about a house however, the room becomes an entity. Consequently, while an entity theoretically cannot be subdivided into further entities, this clearly only holds in practice once we define what we consider the system or context. Therefore, entities represent the smallest parts of a system which have a distinct identity and function; which can fulfil this function without help from other parts of the system. The one exception to this would be data exchange between entities although data exchange might not be considered help anyway.

Without specifying what particular type of entity is of interest there are many examples which can be given. A few examples for entities would be stars, rocks, animals, plants, computers, objects in object oriented programing, pointers or citations. The meaning of entity is broad, yet much narrower than the generic thing, while this makes it a good starting point to think about, there is very little which can be concretely said about entities without further narrowing down what types of entities are of interest.

1.2. What is autonomy?

The meaning of the word 'autonomy' is roughly the ability or state of self-governance. Essentially, something which is autonomous is in full control of itself. Autonomy is also sometimes used to describe the amount of autonomy something has. For this dissertation, only the notion of full autonomy is of relevance.

An autonomous entity is then an entity which acts solely based on its volition, never on another entity's orders. Most often what an autonomous entity uses to decide its actions are its perception and its state. For example, a mouse whose state is hunger and who perceives cheese to the left will go left. However, neither the possession of state or of perception are needed to be an autonomous entity. For example, a rock in a zen garden is both complete and has a function, thus is an entity; and that rock can be considered autonomous as it is unlikely to roll over when ordered so. While it could be said

that a bulldozer could move the rock against its will, the same statement holds true for a human, which is a type of entity usually considered autonomous.

To give an example of an entity which is not autonomous, take a robot that is programmed to follow a specific set of orders. It is an entity, but it is not autonomous as it is in its nature to obey orders.

The robot example, through contrast, shows a way to refine how to describe autonomy in entities while avoiding the quandary found in the zen rock example. If an entity is not autonomous because obeying external forces is part of its nature, then an autonomous entity would be an entity whose nature is to not follow orders except if it wants to do so based on its own volition. This mostly intersects with the first definition developed for autonomous entities, without running into the trouble of either needing to question if anything hurled at approximately thirty kilometres per second around the sun due to no choice of its own is really autonomous, or needing to solve the problem through an exact definition of the entity's possessed degree of autonomy in relation to all possible actions.

1.3. What does lifelike mean?

Lifelike is, as the word implies, a quality used to describe something which appears alive. The reason 'lifelike' is used here rather than 'alive' is because when talking directly about something alive, it is easy to get side-tracked by life in the biological sense. This is fine if talking about entities in reality as what is lifelike and what is alive amount to essentially the same. In a simulation setting, however, entities can have a behaviour or mannerism similar to a living entity, but are not alive in the common biological sense.

A lifelike entity is thus an entity that appears to be alive to an observer who perceives only the entity's behaviour. Additionally, a lifelike entity needs to appear such infinitely. This constraint is added to exclude entities with purely scripted behaviour, which for a limited amount of time might appear to behave in a lifelike manner but will eventually cease to do so once a situation is encountered for which the script provides no guidance.

Deciding whether behaviour shown by an entity is lifelike or not is of course a highly subjective matter. To use it in any objective context it is needed to find some generally agreed upon common properties, which separate lifelike entities from those entities which are not.

For a simple thought experiment to this effect, take a rabbit and a toy rabbit of similar appearance. What will be the difference in behaviour of the two? What makes the one entity, which is known to be alive, lifelike and what prevents the other from appearing lifelike?

The first difference will be that the real rabbit moves while the toy does not. In particular, the real rabbit will be in constant motion as, even when he idles or sleeps, he will at the very least breathe and perform other small body motions. In contrast, the toy, which is not lifelike, may or may not do

anything, depending whether someone else wound up its clockwork or not, assuming the toy possesses such a mechanism. Otherwise formulated, the lifelike entity's state is constantly changing based on actions taken by this entity, whereas the not-lifelike entity's state is only ever changed by an external force.

This does show a very binary difference between that which is lifelike and that which is not. A lifelike entity always does something on its own while a not lifelike entity never does anything on its own. This difference has another property of note though. A lifelike entity like a rabbit can sometimes cease to act on its own but an entity which is not lifelike may never start to act on its own without external influence. In essence this describes death and (re)birth. Of course a lifelike entity might not be truly alive to start with so some of these words could seem weird when used for them, but for the sake of keeping the vocabulary noncomplex they will be used even if slightly incorrect.

The second difference that the real rabbit will show compared to the toy is purpose. There is a reason behind which actions the real rabbit chooses to take. The toy rabbit on the other hand, even if it has some clockwork mechanism to make it move, would only amble around aimlessly, presumably straight into the next wall.

What this shows is that to be considered lifelike an entity needs to possess some kind of intelligence, which determines what actions the entity will perform next based on its purpose. Importantly, that purpose cannot be nothing as that would pre-empt any reason for the entry to act in a reasonable manner. This is because the status quo would be as good as any other state, thus it may either do truly nothing, which is not lifelike as established earlier, or it would act at random which is as not any more lifelike.

In summary, by comparing a living rabbit and a toy rabbit, two distinctive differences have been found between lifelike entities and the entities which are not, which can be used to make a mostly objective statement on whether an entity belongs to the one category or the other.

The differences are that while a not-lifelike entity is a static being which never does anything on its own, a lifelike entity will always and constantly change its state by acting in an autonomous, intelligent and purposeful manner.

It is still not entirely possible without further thought to draw a line between lifelike and not lifelike entities. This is because two other vague concepts found their way in the established description of lifelike: intelligence and purpose. Intelligence is used to describe planning ahead and reasoning. Deliberating about intelligence in the most general context might very well fill an entire dissertation or more. Limiting it to the context of simulation, what is in fact described is artificial intelligence which is more formally presented in section 2.2. For now, intelligence will be used synonymous with reason, thus an entity is acting intelligently if its actions seem reasonable to the external observer.

Purpose however is not so much ill-defined as it raises the question of whether the purpose of lifelike entities can be everything, or if there are common features of the purpose found in living entities; and if so should that restriction be imposed upon lifelike entities in general, or should lifelike entities be allowed to have purposes of a kind not found in actual living entities?

1.4. The purpose behind what lifelike entities do.

It has been established that a lifelike entity will carry out actions and that these actions will have been chosen intelligently to further the entity's purpose. The question thus raised is if there are any particular commonalities between the purposes of different entity's?

Humans are a good example of lifelike entities for this as they show a broad range of purposes, are consistently considered as alive, and examples of actions taken by humans are reasonably easy to find.

A few examples of actions humans might undertake would be: Eat, buy clothes, sleep, find love, read a book, write a dissertation, learn, consume entertainment, procreate, earn money or play a game. Do these actions have anything in common? Some are done clearly for the purpose of survival or in the case of procreation the survival of the human genome. Some make survival easier and thus indirectly also have the purpose of survival. Some make life more pleasant or can be considered fun in other ways.

While the list of examples given is hardly exhaustive, all other examples which could have been given can be filed either as serving the purpose of survival in some form, or giving survival itself purpose by making it worthwhile to be alive. It can be concluded therefore that all actions of humans have the purpose of survival even if only indirectly. Does this hold true for other lifelike entities?

The actions of animals can certainly be classified as having survival as the overarching purpose behind their actions, whether considering actions where the purpose of survival is clear or actions providing enjoyment where the same argument as to how those pertain to survival holds as the one given for humans. Plants, if they are considered as lifelike entities, which they may or may not be depending on whether conscience is considered a necessity for intelligence, also only perform actions which further their survival.

It can thus be assumed that the purpose of any real world living entity is without exception survival. Does this need to apply to lifelike entities in a simulation? There is no correct answer to this, since this mostly depends on the aim of the simulation containing those entities. Most people will presumably expect that simulated lifelike entities mirror the behaviour of living entities, thus this dissertation will use survival as the purpose of all lifelike entities considered. That is not to say that entities which have a purpose other than survival are not an interesting topic for another time.

1.5. The requirements of survival

It has been decided that the purpose of all lifelike entities in the context of this dissertation have survival as their sole overarching purpose, possibly supported by more specific sub-purposes, which directly or indirectly further the goal of the entity's survival. What then does an entity need to do to survive?

Broadly speaking, survival of living entities depends on the availability of vital resources such as food, water or air, which are used to maintain particular states of the entity such as being rested, well fed or being warm. These two things can be summarised if the states such as rested and fed are seen as internal resources that must not fall below a certain threshold. Such resources can be said to be the entity's needs, and the resources strived for in supporting purposes such as the purpose of making survival worthwhile can be said to be an entity's wants.

In practice, unless there currently is a need which must be urgently attended to for the entity to survive, there is not much difference visible in behaviour between needs and wants. A lifelike entity such as a cat will split its actions appropriately between all of its needs and wants. While the precise order is likely to vary depending on each entities' preferences, entities in general will prioritise their needs and wants according to how well or how poorly they are defined, for example, a hungry cat will prioritise hunting over sleeping on their human's lap but a well fed one will most likely prefer the lap.

In conclusion then, what the autonomous lifelike entity pursues with each of its actions are a number of things or states which are either needed or wanted for the purpose of survival. At any given point an entity will give some of these needs and wants priority over others based on its current state. While needs are more likely to receive a high priority than wants, because needs directly affect survival which is the final purpose of all the entity's actions, needs and wants do otherwise not differ from each other.

There is however another factor beyond this simple priority that an intelligent entity will take into account when deciding what to pursue with its actions. Namely how its actions will affect the fulfilment of its needs and wants over a longer time span, as opposed to only the immediate effects. The incapability, of not being able to take this trade-off between long term planning and short term profit into account may, not make an entity not seem lifelike, but being able to plan ahead will most likely result in a much more believable behaviour to the outside observer.

1.6. Putting the pieces together.

An entity is something distinct with a function, and an autonomous entity is an entity which acts solely by its own volition. A lifelike autonomous entity thus is an autonomous entity which continuously changes its state by acting in an intelligent fashion. And which does so for the sole purpose of survival.

To survive an entity needs to fulfil its needs and wants. Both of these can be described as the acquisition of resources which may be of tangible or intangible nature. The entity's state generally dictates to which fulfilment of a need or want to work towards, however this may be influenced by biases intrinsic to the entity.

While not strictly required, the ability to plan ahead is found in many living beings, in some form and to different degrees. Consequently, it is likely to be of use to a simulated lifelike autonomous entity too, especially those entities from which a show of high intelligence is expected.

2. Literature

This section aims to give an overview of some of the concepts and actual scientific works, which can be found in scientific literature that are of particular interest in the context of this dissertation.

2.1. Agents

The word agent derives from the Latin verb ‘agere’ which simply means to do. There are many definitions of what an agent is, even when only looking within the context of the computer sciences. These definitions when ordered by complexity can be viewed as building upon each other. They are by no means a linear development, however there is enough exchange of ideas between them to make it seems so on a cursory glance. The simplest definitions describe agents as systems which take inputs and produce outputs. In more complex definitions, autonomy and persistency are added and in the most complex of definitions, which often but not always refer to agents as intelligent agents, the notion of reasoning and purpose is added to agents. [1]

When using the broadest definition for an agent as being an entity which takes a given input and reacts to it with some output, then much can be described as agent. Examples could be all but the simplest programs, but also many functions be they computational or of a mathematical nature. Or if applied to the real world, humans would be agents as would be thermostats and dishwashers and everything in-between. Because of this, such a definition can serve well to describe particular subsystems within a well-defined larger system, however within a less concrete context, this definition may be too broad to be of much use. [2]

When adding elements of autonomy and persistency to the definition of an agent, the amount of things which can be considered an agent shrinks significantly.
Persistency means that the agent has a state of its own which persists after each action and thus the agent’s actions are influenced by its previous actions. This excludes most programs and simple functions from being considered as agents as they lack a persistent state of their own: what they output is only ever dependent on the last input. [3]

Autonomy reinforces this by requiring that the output of an agent is only dependant on its own state and the input it perceives. Autonomy further implies that each actor has a concrete idea on how to act to fulfil its purpose, when presented with a specific input. While this does not explicitly make some form of intelligence mandatory, it implies that agents may benefit from being intelligent entities. [4]

Intelligent agents, sometimes referred to simply as agents, incorporate all the properties described before but also explicitly possess both the capacity to act intelligently and a purpose they aim to achieve. Such agents, as can also be expected from the prefix intelligent, need to possess some form of artificial intelligence which governs their actions based on their purpose, state, and current perception. [5]

In conclusion it can be said that an agent is an entity which is capable of perceiving its environment, whatever this may be, and which is able to autonomously act based upon that perception and upon its persistent internal state, such that its actions fulfil a purpose.

2.2. Artificial Intelligence

The field of artificial intelligence is a broad and diverse subfield of the computer sciences. There exist various distinct approaches within the domain of artificial intelligence, some of them so different from each other that their only real common point is that their aim is nominally the same, namely to create an intelligent entity or to give intelligence to a previous unintelligent one. [6]

One way to classify them is by ordering them along two axes. The first axis has the focus on behaviour at one end and the focus on thought on the other. The second axis opposes humanlike and rational with each other. This allows to classify the various approaches roughly into four categories. [2]

- The first, human behaviour, encompasses those approaches which seek to directly emulate humans. The maybe most know test for artificially intelligence, the Turing Test, tests this in particular. [7]
- The second, human thought, is also known as cognitive modelling. This type of approach does not attempt to emulate the behaviour of humans directly, rather it attempts to reverse engineer the actual process by which humans think by looking at the findings from neurobiology and psychology. [8]
- The third, rational thought, was developed by those subscribing to the logicist tradition of artificial intelligence. The roots of this approach can be found in logicians' work in the 19th century to develop ways to describe the world using only logical notation. Programs to solve problems presented in such notation exist since the seventies. However, this approach is not very common nowadays as transcribing reality into logical statements is more difficult than it seems at first glance, secondly programs to solve such problems are prone to require more resources than are available for all but the simplest of problems. While this last problem is also found in other approaches of artificial intelligence, it is most prominent in this approach. [9]
- The fourth, rational behaviour, more often referred to as the rational agent approach, aims to provide a way for an agent to act in such a way that that the outcome for the agent is the best or as good as possible if there is no clear best. What is particular about this approach is that it can incorporate elements of the other approaches without having to contend with most of their drawbacks such as the rigidity of the rational thought approach or the very domain bound approaches which emulate human thought or behaviour. [10]

While artificial intelligence and agent based systems are nominally two different fields, the rational agent based approach illustrates how the two disciplines can overlap. In the perspective of artificial intelligence this is about entities which act rationally and thus are intelligent. Observed from the other side, we have intelligent agents which fulfil all other criteria to be considered agents and possess sufficient intelligence to interact with their environment with a purpose.

2.3. Cognitive Modelling

At its core, computational psychology as cognitive modelling is also known as a highly interdisciplinary discipline, drawing heavily both upon psychology and computer science. In recent years both sociology and neurobiology have started to also have a visible influence. [11]

The aim of cognitive modelling is to create models which can model various behavioural phenomena observed from humans and thus can lead to increased understanding of these phenomena While there exist both pure mathematical models and conventional models, computational models, that is models which describe human behaviour in the form of algorithms, are the most prominent as, for the time being, they have shown themselves to be the most expressive type. [11]

One branch of cognitive modelling aims to create cognitive architectures. These cognitive architectures aim to be a concise system which can be used to model any phenomenon rather than a phenomenon specific model. Concretely this means that a unified theory is needed which can describe all effects of cognition such as memory, decision making, perception, interacting with the environment and so forth. Such a system should be composed of set of procedures which can model all these cognitive tasks while keeping the amount of distinct procedures as small as possible. Of course it is at present not possible to model all those things as both the computational resources and the understanding of human cognition is currently insufficient. Consequently, cognitive architectures need to attempt to be as good as possible rather than attempting to achieve a perfect representation of human cognition. [12]

Some of the more prominent cognitive architectures are SOAR, ACT-R and EPIC. These all fulfil the basic hallmark of being functional yet of reasonable simplicity, however they approach the problem of human cognition from different angles and thus are not often used in the same experimental context. More recently there have been attempts to merge these models, however only time will tell if this is the most promising approach to combine their functionality. [8]

The SOAR cognitive architecture is a production system, that is a system which operates on rules which govern its behaviour. SOAR operates by pursuing a specific goal by using these rules until it either fulfils the goal or cannot follow said rules anymore because it lacks knowledge. If the lack of knowledge causes it to stall in a so called impasse, SOAR temporarily creates a new goal which is to

resolve this impasse. Once the goal to solve an impasse has been reached, SOAR remembers the found solution for possible later use and then continues following its original goal. [12]

ACT-R is at its core also a production system, however it also incorporates some prominent sub-symbolic functionality. Historically the main focus of ACT-R was to model human memory. This is still an important focus but recent iterations of ACT-R also attempt to satisfactorily model perception and motor systems. ACT-R works using a central production system which trough buffers, which represent the current focus, interacts with various modules which can represent near anything but usually include goals, memory, current perception and current motor capacities. [13]

The EPIC cognitive architecture attempts to provide not only a model of cognition but also a model for the perception processes coming before cognition and for the motor processes coming after. One feature of EPIC is that the model takes into account various constrains on its components. For example, the different speeds of execution between its components is explicitly addressed, by making components communicate with each other and sometimes themselves by sending timed events to each other. Another example for limitations found in real humans and simulated in EPIC is that human vision deteriorates the further a perceived object is from the centre of a human's field of view. [14]

2.4. Finite State Machines

Finite state machines represent a conceptual way to think about complex systems in a complete yet comprehensive manner. A finite state machine can be understood as a unidirectional graph whose nodes represent the discrete states the system can be in and whose edges are possible transitions between these states. The state machine is at any time in a given state and changes state only when an external event occurs for which the current state possesses a matching transition. The new state is then determined by following this transition. [15]

It is possible to separated finite state machines further into deterministic finite state machines and nondeterministic state machines. In deterministic state machines there is only one possible transition in each state for each possible event and if there is no valid transition possible following an event then the state machine halts, thus if the sequence of all events observed by the state machine is known, then the precise ending state after processing these inputs is also known. In nondeterministic finite state machines however there may be multiple possible transactions after a given input of which one is chosen at random or there might be no possible transaction in which case the event is ignored. While this makes it impossible to determine the end state of the state machine in practical applications, this is not so much the case in a theoretical context as algorithms have been found which can for any nondeterminism finite state machine find an equivalent deterministic finite state machine, although the complexity of the deterministic state machine will usually be significantly greater than the complexity of the nondeterministic state machine. [16]

3. Aims

The aim of the project underlying this dissertation was to develop a model which can be used to simulate autonomous lifelike entities, and to prove that it is correct and usable in practice. This model should be as simple as possible, but still seem believable to an independent observer who is not familiar with the model or the simulated situation.

Firstly, an entity should exhibit at any given moment the following features:

- The entity possesses a distinct state.
- The entity's purpose is its own survival.
- The entity's survival is achieved by fulfilling needs and wants.
- The entity chooses all its actions after an intelligent fashion.
- The entity's actions always change its state in some sense.
- The entity never does nothing, that is its state always changes from one step to the next.
- The entity possesses properties which make it behave in a unique way.

Secondly, the model should be as general as possible while also being as simple as possible. This should be achieved by having as few distinct types of processes as can be used, such that it can still be used for any type of simulation by only changing the input parameters rather than changing and or extending the logic of the model.

Thirdly, the model should allow entities to produce intelligent reasoning without needing to take into account the nature of their surroundings or having any prior knowledge about the domain of the simulation. The judgment on how well the model succeeds at this shall be done by examining the behaviour of simulated entities as perceived by an outside observer who is not aware of the nature of the used model itself.

There is a significant overlap between these stated aims and the general aim of cognitive architectures, most notably when it comes to the notion of it being a general model based on as few mechanisms as possible. What does however strongly distinguish the aims presented here and the aims of cognitive architectures is that they aim to model human cognition based on real world knowledge, while this dissertation attempts to find a model for lifelike autonomous entities which is a concept rather than a tangible thing found in reality.

What is aimed for is in fact a model of an intelligent agent which can be used in any type of simulation. However, with a number of restrictions which are not part of the standard definition of an intelligent agent. In particular, those are that idling is an action like any other and that no action is an error, that the model must be as simple as possible and that the model must be completely domain agnostic, both in design and execution.

4. Model

The entity model which has been developed to meet the aims resembles a simple state machine, however the state of each model is represented as multidimensional continues state space rather than a set of discrete states as in a classical state machine.

One important overarching principle is that entities have no knowledge of the domain of a particular simulation and thus do not know what their own state means in the context of the simulation. Entities only know which states are beneficial for their survival and which states are detrimental.

The word ‘simulation’ will be referred to a number of time. In the context of this description of the model, the simulation refers to the information needed to simulate a particular situation or environment using this model. Also, use is made of the word ‘world’, which is to be understood as the mechanism inside the implementation which handles the execution of all entities based on this model.

4.1. State

Each entity has a state, that state can be understood as a vector whose elements are specific resources. A resource can represent an amount of something like food, time, money, hunger or happiness the entity possesses, or a resource can represent a particular situation the entity finds itself in such as its current location. In either case the resources value is represented using a real number.

If that number has an ordinal or cardinal meaning, then the more positive meanings are denoted by high values and negative meanings are denoted by low values. For example, if a resource represents food then the value of the resource could translate directly into units of food, however if the resource would represent hunger then the meaning would be reversed, a high value would mean that the entity is not hungry while a low value would mean that the entity is very hungry and a value of zero might mean that the entity is starving or has even died of starvation. If the resource represents a purely modal value, then the value of the resource is not linked to whether it is in a good or bad state.

How many or which resources are part of an entity’s state is dependent on what is being simulated. This works because, as stated earlier, the entities do not need to know what the resources in their state actually mean and thus can always threat all resources in the same manner irrespective of their number. There is one exception in that entities do know which resources represent needs. There is nothing special about such needs except that these are the resources the entities need to keep high to survive, and thus their values will be taken strongly into consideration when deciding what to do. While the general idea behind needs is that if they drop too low, bad things happen to the entity, the model itself does not define any particular effect that low needs have on the entity, this is left to be determined by both the implementation and the specific simulation being run in that implementation. An entity is assumed to be aware of all resources in a simulation and if it does not know what the value of that resource is for it, then it will assume the value to be zero.

4.2. Transactions

The actions taken by each entity are described in the simulation by transactions. A transaction is a set of changes to a subset of the entity's state which the entity can commit if its current state meets certain prerequisites.

There are two types of transactions. Deterministic transactions and non-deterministic transactions. When an entity commits a deterministic transaction, then the changes described in the transaction itself are applied to the entity's state and the entity continues on to select the next transaction to commit. If the committed transaction is non-deterministic however, then the world may modify the changes described in the transaction before they are applied to the entity's state. The reason non-deterministic transactions exist is that they allow to add both speculative transactions and transactions which need to be committed in parallel by two or more entities to a given simulation. In any case, the distinction between deterministic and non-deterministic transactions is opaque to the entities which only use the original values of transactions to determine which transaction they will commit next and any learning is done on the basis of the changes independent on how these changes where decided.

Because the entity has no knowledge of what its resources represent, an entity in this model also has no notion of time. As even if it might keep track of it as a resource in its state, it would not know what it represents. Therefore, in the eyes of the entity each committed transaction finishes instantly, while in practice the simulation would have in fact suspended the entity for the time it takes for the transaction to happen, if the transaction has a duration in the simulation that is.

After its last transaction finishes, each entity selects another transaction to commit based on reasoning which takes into account if the transaction will increase one of its needs now or help to do so in the future. The influence of the different needs is balanced against each other based on a priority which is assigned to each of them based on the entity's current state. The priority for each need is determined each time a new transaction needs to be committed. As described further down, the priority of each of the needs is determined as part of the intelligent thought process inside this model.

Normally, an entity always selects a transaction after the last transaction committed ends. However, as transactions require that an entity's state fulfils certain criteria, it is possible for the entity to arrive in a state where it cannot do any further transactions. If this happens then this model considers that entity to have died. An example for this would be a simulation of a robot where all transactions both require the battery charge resource to be non-zero and always reduce that resource. When the battery charge resource drops to low, the robot ceases to act lifelike. Which is in accordance with what an external observer would expect.

4.3. Memory and Perception

At any given point, each entity is aware of a subset of all transactions possible within the simulation. The collection of these know transactions, whether it is currently possible for the entity to commit them or not, represent the entities memory. The transactions in the memory are the only transactions from which the entity selects one to commit when choosing which transaction to commit next.

It is up to the simulation to determine which transactions an entity knows under which circumstances. The simplest way would be to give each entity knowledge of all transactions defined in the current simulation. While this might be suitable to many simulations, it may not be so for all.

To allow simulations to be closer to reality, the notion of perception is added to the model. Perception is an activity the entity does after a committed transactions finishes and before the entity selects the next transaction to commit. As there may not be a gap between transactions in this model, perception must be treated as being instantaneous and taking no time. A more realistic alternative would be to have perception happening in parallel to transactions, but as what an entity perceives only changes when a committed transaction finishes, this would add complexity for no functional gain. What the entity perceives after each committed transaction is a subset of all transactions in the simulation, whose composition depends on the current state of the entity. This always includes all transactions the entity can currently commit. If there are transactions in the entity's memory which are seemingly possible but are not perceived, then the entity removes them from its memory as they are evidently not possible anymore.

4.4. Intelligence

To select a transaction to commit, the entity firsts calculates what the priorities of its needs are, then it looks at all transactions it can currently commit and calculates a utility value for each of these based on the priorities of its needs and the changes made to its state by the transaction if it would commit it. If the simulation specifies that an entity should do long term planning, then the entity also internally simulates which transactions it presumably could commit after committing a specific transactions and which transactions it could commit after committing one of these and so forth up to a depth specified in the simulation. To calculate the utility of such transaction sequences, the summation of changes to the entities by all transactions in the sequence is treated as if it where one single transaction.

The model does not specify which algorithms to use to determine the priority of each needs or the utility of each transaction or transaction sequence. One conceivable option to determine the priorities of needs would be a simple function which calculates the priorities based directly on the values of the needs. Such a function could however not make good use of the rest of the entity's states as the entity does not know the relationships between its resources. A more complex option would be to use a machine learning algorithm to find a good function which can utilise the entire state to determine the

needs' priorities. Such a learning system could be trained while the simulation is running using so called reinforcement learning, which uses the values of the entities' needs as reward. To calculate the utility of transactions, a simple comparison of cost, negative change, and benefit, positive change is most likely sufficient, but more advanced approaches may be worth investigation for more complex simulations.

4.5. Personality

The model so far is expected to be capable of simulating autonomous lifelike entities at this point. However, all entities simulated with it would act identical when their states are equivalent. This may be an acceptable limitation depending on what is simulated. If what is simulated is for example a beehive, then observing such consensus between entities would not necessarily be a surprise or could even be expected by an outside observer. However, if the simulation is about entities more associated with individuality like humans or other mammals, then this would most likely not give satisfactory results.

To add individuality to the model, each entity is given a personality coefficient for each resource which affects the selection of transactions to commit by modifying the perceived utility of that resource. These coefficients should be real valued and high values should increase the perceived utility and low values should decrease it. If used appropriately this mechanism will make each entity with a unique personality behave in a unique manner while still maintaining the other qualities of the model.

It is worth noting, though, that in most simulations personality should not be determined completely at random, as very high values could make transactions containing that resource always attractive independent of the priorities of the needs, and very low values could make entities care very little or not at all about the changes of that resource when determining the utility of transactions. Of course for some simulations or specific resources within simulations this might be the desired behaviour. In particular, forcing a personality to zero is a way of making an entity ignore the actual value of a resource, whose value is modal, and only make it consider that resource when determining which transactions it can do or could do after committing a transaction which changes this resource. An example for such a resource would be location whose value would not be of relevance to the entity except when determining what it can currently do. It could be said that this is giving entities knowledge of the domain of the simulation, however a simulation needs to either avoid modal resources or tell entities that they don't care about the actual value of these resource, and using the entities personality to do this is presumably the most intuitive and subtle way to do so.

4.6. Summary

In the presented model, entities possess a homogenous state described by a feature vector of real values. Each of these values is called a resource and represents some possession or state of being an entity can have in the current simulation. Each entity constantly commits transactions which change a particular subset of its state in a manner described in the transaction or determined by the world if the transaction is not deterministic.

To select the next transaction to commit after the previous transaction is finished, the entity computes the current priorities of its needs and determines the utility of all currently possible transactions and transaction sequences based on the changes to the entities state the transaction is expected to have, the priorities of the needs and the entities personality. The transaction with the highest utility or the first transaction of the sequence with the highest priority is then committed by the entity.

In general, the entities in this model do not need to know anything about what the resources in their states represent. All that entities need to know about the simulation is which transactions are available to them, and which transactions they currently see, and entities need to be able to determine if a change of a resource in a transaction is a cost or a gain. The latter is facilitated by the model requiring the simulation to define low values of transactions as bad and high values as good, and in the case of modal resources to set the relevant personality value such that the entity does not care about the change in value of that resource, but only about what the current value of this resource is.

5. Terms

The following terms are commonly used in this dissertation and have mostly been introduced in the model section above. For clarity each of them is given a concise definition here which unless otherwise noted is how these terms will be used for the rest of this dissertation:

Transaction: A set of changes to a subset of the entity's state, which may only be committed if the entity's state meets certain prerequisites as defined in the transaction.

Commit: Committing to a transaction means that the entity passed a point of no return and will not act further until that transaction has ended. For the outside observer, time may pass before the committed transaction ends, but for the entity committing the transaction this is an instantaneous process.

Entity: A lifelike entity with a private state which it continually modifies by committing transactions.

Simulation: The simulation consists of descriptions of the available resources, of which some are needs, all potentially possible transactions, and the conditions under which entities perceive those transactions. It further specifies how entities are created and, if applicable, how the world handles each non-deterministic transaction.

World: The world is less a place than a process which validates the entities' commitments and which keeps track of what entities perceive based on the simulation which is currently running.

State: The set of all resources an entity possesses.

Resources: The building blocks of an entity's state. A single real value.

6. Design

The system consists of the world and any number of entities which communicate directly with the world and indirectly with each other using transactions.

6.1. Notation

All data structures described in the following are, unless explicitly specified otherwise, either sets or maps. Maps are defined as sets whose elements are ordered pairs of type $\langle key \in \mathbb{N}_0, value \in \mathbb{R} \rangle$.

Standard notation will be used for the sets. However, for maps S the following extensions to set notation shall be used:

S^i shall refer to the pair element in the set whose element key equals i.

S_i shall refer to the element $value$ of the pair element in the set whose element key equals i.

Furthermore, arithmetic operators $+$ and $-$ on such sets A B shall give a new set C and be defined as operating on the element $value$ of the pairs $p \in A$ and $q \in A$ whose element key matches directly and if there is no such match by using zero as the second operand.

$$C = A + B = \bigcup_{i \in I_A, i \notin I_B} \langle i, A_i + 0 \rangle \cup \bigcup_{i \in I_A, i \in I_B} \langle i, A_i + B_i \rangle \cup \bigcup_{i \notin I_A, i \in I_B} \langle i, 0 + B_i \rangle$$

$$C = A - B = \bigcup_{i \in I_A, i \notin I_B} \langle i, A_i - 0 \rangle \cup \bigcup_{i \in I_A, i \in I_B} \langle i, A_i - B_i \rangle \cup \bigcup_{i \notin I_A, i \in I_B} \langle i, 0 - B_i \rangle$$

6.2. Transactions

Transactions contain a map detailing how an entity's resources change when it commits this transaction and three maps detailing conditions as to when the transaction can be committed. Concretely these maps describe which resources must be at least a certain value, which resources must equal a certain value, and which resources must be at most a certain value.

There are two types of transactions, those whose outcome is fully deterministic and those whose outcome depends also on the transactions committed by other entities during the same time span. Both types do not need to be present in a given simulation.

It must be stressed, however, that whether a transaction is deterministic or non-deterministic is only of relevance to the world and to the simulated environment. From the perspective of the entity, there is no distinction between the two types of transaction. Both may have some conditions and describe how they might change the entity's resources. Whether the actual change after committing the transaction is equivalent to the listed change is not taken into account by the entity.

Deterministic

Deterministic transactions are transactions which modify the entity's state exactly as described in the transactions at the time the entity chooses to commit it.

Non-Deterministic

For non-deterministic transactions, the world may modify the changes made to the entity's state after committing the transaction based on what else is, invisible to the committing entity, going on. An example for the use of non-deterministic transactions would be that when an entity commits such a transaction, one or more transactions become possible to other entities to commit. The outcome is then determined by which of these transactions have been committed by other entities.

6.3. Entities

The entities are autonomous systems in their own right, whose single interface with the world lies in the transactions presented to them by the world as possible, and in their choice of which of these transactions the entity actually commits.

Entity Data

Each entity contains a memory M, which is a set containing all transactions which it has perceived up to this point in time. The exception is for transactions which the entity previously perceived but later found to be not possible anymore based on its perception, not based on whether it personally can or cannot commit that transaction.

The entity contains a map R whose values represent its current resources.

Each entity contains a map N and a map P whose contents are static but unique for each entity.

Each value N_i represent the minimum amount of resource R_i the entity needs to be content.

Each value in P_i represent how valuable the entity considers each resource R_i.

Furthermore, the following subsets from the sets above will be used to simplify certain equations:

M^* shall be the subset of M containing only those transactions the entity can commit now.

M_t^* shall be what M^* would be after the entity commits transaction t.

M_i^S shall be all currently possible transaction sequences of length i.

R_t is what R would be after the entity commits transaction n t.

R^N shall be the subset of R with only those elements with an associated element in N.

$$R^{*N} = \bigcup_{i \in I_N} R^i$$

Q, V and W_i are maps whose elements are recalculated each cycle. Q contains the current priorities of the entities' needs found in R^N. V the current value of each resource. W_i the current worth of each transaction in M_i^S.

Note that: $I_R = I_P = I_Q \wedge I_N = I_{R^*}$

Additionally, the entity stores a set of parameters ψ which is used in decision making and which can be improved using reinforcement learning.

Entity Cycle

Each entity goes through seven steps each cycle.

In the first step, the entity queries the world as to which transactions it currently perceives and adds any previously unknown transactions to M, but also removes those transactions it would have expected to perceive but did not from M.

In the second step, each resource associated with a need is assigned a priority value between 0 and 1 based on the entity's current resources according to a function f and the set of parameters ψ. If the resource is not associated with a need it is given a value of x which is a constant parameter.

$$Q = f(R, \psi) \cup \bigcup_{i \notin I_N} (i, x)$$

A high value in Q signifies that it is important for the entity to acquire the associated resource in R to keep it above the associated threshold in N while a low value indicates that the entity can focus safely on other resources.

In the third step the entity calculates a coefficient of worth for each resource based on its personality and the priority value found in the first step.

$$\mathrm{V} = \bigcup_{I_R} (i, 1 \times Q_{\mathrm{i}} \times \mathrm{P_i})$$

The fourth step consists of the entity determining all sequences of transactions M_x^S it believes it can currently commit, given the assumption that all transactions are deterministic and that all transactions which are available now will still be so in the intermediate future, up to a certain depth x. M_x^{*S} is found by iterating over all transactions $t \in M^*$ and simulating what would happen if transaction t would be committed, and what M_1^* would thus be. This is repeated until the desired depth of look ahead is reached.

In the fifth step, the entity first determines what it deems to be the cost and benefit of each transaction sequence it found to be possible in the third step, and then it calculates the worth of each transaction sequence by dividing the presumed benefit by the presumed cost.

$$W = \bigcup_{t \in M_x^*} \sum_{i \in I_t} (t_i \times W_i) \div \sum_{i \in I_t} (-t_i \times W_i)$$

Division by zero can be avoided by treating all cost below a certain threshold to be equal to that threshold. Similarly, the benefit can also be given a non-zero minimum value to ensure that different costs give different results if there is no benefit to the entity in the transaction.

The sixth step consists in committing the first transaction in the sequence with the highest worth as determined in the fourth step. The entity observes committing a transaction and the resulting changes in R as a single atomic step. However, depending on the nature of what that transaction represents, the world might suspend the entity for a given amount of time to simulate the time taken to finish said transaction.

The seventh step consists in improving the values in ψ, used in step one, using a function q which is given ψ, how well the entity's needs are fulfilled after committing the selected transaction and the priorities Q calculated for each need in the first step.

$$\psi = q(\psi, \bigcup_{i \epsilon I_{R^*}} \langle i, R_i^* - N_i \rangle, Q)$$

6.4. World

The role of the world is both to keep track of which transactions exist and which of these are currently possible, and to determine the precise outcome of each transaction committed by entities. One main difference between the world and the entities within it, is that the world has some understanding of the domain of the simulation while the entity has none. For example, the world is aware of which resources represent time or location and can use that knowledge to guide its actions.

World Data

The world contains the set of all entities E, a set C of all transactions committed but not yet finished and a map of locations L. Each location l contains a set of transactions T^l which represent those transactions potentially visible to an entity visiting this location and a map of resources R^l which is used to determine which transactions in T^l can currently be committed by entities and which cannot. The values in T^l can change each time an entity commits a transaction while the entity is considered to be at location l.

The world also keeps track of time τ which is a monotonously increasing number.

World Cycle

The world goes through three steps each cycle.

In the first step, the world determines which transactions in C are coming to fruition in this cycle based on when that transaction was committed, its duration and τ.

In the second step, the world determines which of the transactions in C are deterministic and which are not. Those transactions which are deterministic a simply passed back to the committing entity to signal that they have finished. For those transactions which are non-deterministic however, the world must first determine how they should change the committing entities state based on the current state of the world. Once the world has determined the outcome of a non-deterministic transaction, it is passed in its modified form back to the committing entity like a deterministic transaction.

In the second step the world determines first if and how the entities' transactions, which ended in the first step, changed the available transactions and resources in each location l and updates T^l and R^l accordingly.

7. Implementation

The implementation created alongside this dissertation based upon the design presented above is written in C++ using visual studio. C++ was chosen as it provides high level features such as a comprehensive standard library and full support for object oriented programing, yet is low level enough to compile down to native code which makes the resulting executable run reasonably fast.

Heavy use has been made of the Microsoft visual studio ide both for debugging and for code formatting, code generation and code improvements. Most notably through the C++ version of the ReSharper plugin developed by JetBrains.

Care has been taken that all settings can be provided either through the command line or configuration files, such that all simulation parameters can be changed without needing to recompile the executable.

There are three main classes, Server, World and Entity. The server class provides utility functions such as logging and information about the configuration parameters. The world class does the bookkeeping of what happens in the simulation, such as which transactions are known to each particular entity. The entity class contains the actual implementation of the model of lifelike autonomous entities described previously.

7.1. Server

The server class is the single object directly instantiated by the main function (the entry function in C++). Its public methods provide firstly an interface to load, modify and save both the configuration file and the file describing the simulation to be run. Secondly they provide controls to start and stop the simulation. Additionally, this class this class provides some public static functions whose purpose is to allow other objects to look up configuration values on the fly and to provide output to be written into one log file in an ordered atomic manner.

7.2. World

The world contains hashmaps containing pointers to all entities, transactions and locations in the current simulation, as well as a priority queue containing all currently committed transactions ordered by their time of completion. It should be noted that, for the sake of memory efficiency, all memory intensive resources are centralised in the World object, and that the entities refer to them by their map key and only access the actual resource when needed by reference via methods of the world object.

After being instantiated, the world object, of which there only will be one instance of the class at a time, will loop over a simple cycle of actions. During each cycle, the world will first check if all entities are currently committing a transaction. If this is the case, then the world will be increasing the simulation time counter to the finish time of the top element of the priority queue containing all

committed transactions. Then the world will handle any committed transactions in the queue whose finish time is less or equal than the simulation time.

For each such transaction, the world will check if it is currently possible, and, if so, the world will notify the committing entity that its last commit has finished and instruct the entity on how to change its state. If the transaction is not currently possible anymore then the world will order the committing entity to select a new transaction to commit. However, the affected entities state remains unchanged if a transaction cannot be validated.

As non-deterministic transactions are not implemented at this time, the changes to the entities' states are always exactly what is described in each transaction. Once all outstanding transactions are processed, the world informs all entities about which transactions are currently possible at their current location and which transactions at their location can currently not be committed.

Note that a location here refers to an object which holds information about what an entity perceives in terms of possible transactions when the entities' resources, which represents location, is equal to a certain location's id. As usual the entity itself is not aware that that particular resource has any particular relevance compared to any other resource in its state.

One thing which may influence which transactions are available at a given location is the availability of certain resources at these locations. Locations where transactions which care about this are available keep track of resources in a similar manner as an entities' state keeps track of its resources, with the main difference that the condition for the availability of a transaction always is that a resource may not fall below a value of zero; and further, the transactions committed at the location cause the opposite effect on the location's pseudo state than they have on the committing entity's state.

7.3. Entity

The entity class encapsulates all functionality needed to implement the model for an autonomous lifelike entity as described earlier in this dissertation. For ease of programming that functionality is split between a number of objects stored as fields in each entity object, even though there is no design reason to do so.

In terms of data structures, each Entity object holds a set containing all currently know transactions and three different objects, which contain its need, state, and personality.

The inventory object holding the entity's state is capable of testing if a given transaction can be committed given the entity's current state, and presents methods to actually commit the changes to the entity's state described in a transaction to be invoked by the world after the committed transaction finishes.

The actor object is the brain of the entity. When the previously committed transaction ends, the actor selects the next transaction to commit based on which transaction or transaction sequence promise to have the highest utility. As the number of possible transaction sequences the entity has to decide between grows exponentially with the depth of the look ahead, it is not possible with current machines to look at all possible sequences at once to determine the transaction sequence with the highest utility if there is a desire to look more than a few steps ahead. To avoid this limitation, each possible transaction sequence is looked at and then discarded unless it had the best utility so far.

To determine the utility of a transaction sequence, the changes described in all transactions inside the sequence are first summed up. Then the utility value of each resource is determined by multiplying the amount the resource change by the resources priority and personality coefficient. Finally, the utility values of all positive changes are summed up and divided by the absolute sum of all negative utility values. If the sum of positive changes is smaller than 0.1 then it is considered to be 0.1. Similarly, the absolute sum of negative changes is always considered to be at least 1.

To determine the priorities for needs, the actor uses a neural net, which takes the entity's current state as input, and outputs a number between 0 and 1 for each resource which is a need. The code for the neural net was however not written in the course of this project, but was instead copied over with only a few modifications from work undertaken at a previous point.

The critic object looks at the entity's state each time a committed transaction finishes, and informs the actor about what it thinks the priorities should have been in the last step by comparing the resources which are considered needs to a threshold in the needs object, and reporting the difference after passing it through a logistic function. This signal provided by the critic is used by the actor to improve the weights in the neural network.

8. Results

A number of simulations of varying complexity were run. They are listed here in order of increasing complexity.

Each simulation is presented with a short overview of which resources are available and what they are meant to represent. Of course, what the resources represent is for the benefit of forming an opinion of how well the simulation performs, as their meaning does not impact the actual simulation. Afterwards the transactions possible in the system are listed. Each with a small description of what action it is supposed to represent.

The notation for transactions used is a list of properties. The number after the transaction header is the id of the transaction, by which it may be referred to. Anything after a # is a comment, usually used to

give the transaction a description of what it represents. All other lines are in the form of LETTER pace resource id space value. Where LETTER can be:

- R which means the resource must be equal to the value to be possible.
- L which means the resource must be equal or less than the value to be possible.
- M which means the resource must be equal or greater than the value to be possible.
- D which means the resource is changed by the value when committed.

All simulations presented here share that resource 0 represents time and resource 1 represents location and is flagged as a nominal resource, and thus is not taken into account when calculating the value of a transaction because the associated personality coefficient is forced to zero.

Each simulation is defined in one configuration file, and one simulation file, and the output can be found in a log and in a summary. The names of these files submitted together with the code will be x.cfg, x.ses, x.log and x_summary.log where x is the name of the simulation.

Simulation 1

The first simulation is very simple in that it contains only three locations and four resources of which two are needs and all transactions are either used to move location or to fulfil a need. The "sane" files can be used to run this simulation.

Setup

Entities perceive transactions 1 to 5 when at location 0, 6 to 7 when at location 1 and 8 to 9 when at location 2. The state of each entity is initialised to 0=0 1=0 2=5 3=5.

The needs, resource 2 representing food and 3 representing drink, need to be 1 or greater or the entity can't commit a transaction and dies. They are capped at 10 to represent the inability to stock them up indefinitely. A quirk in the simulation is that, as time is a cost in this simulation, time would seem to run backwards for the entities if the entities would have a notion of time.

```
Transaction 1
#Idle
M 2 1
M 3 1
D 0 -1
D 2 -1
D 3 -1

Transaction 2
#Eat
R 1 1
M 3 1
L 2 10
D 0 -1
D 2 5
D 3 -1
```

```
Transaction 3
#Drink
R 1 2
M 2 1
L 3 10
D 0 -1
D 2 -1
D 3 5
```

```
Transaction 4
#Go to Food
R 1 0
M 2 2
M 3 2
D 0 -1
D 1 1
D 2 -2
D 3 -2
```

```
Transaction 5
#Go to Drink
R 1 0
M 2 2
M 3 2
D 0 -1
D 1 2
D 2 -2
D 3 -2
```

```
Transaction 6
#Go to Home
R 1 1
M 2 2
M 3 2
D 0 -1
D 1 -1
D 2 -2
D 3 -2
```

```
Transaction 7          Transaction 8          Transaction 9
#Go to Drink           #Go to Home            #Go to Food
R 1 1                  R 1 2                  R 1 2
M 2 1                  M 2 2                  M 2 1
M 3 1                  M 3 2                  M 3 1
D 0 -1                 D 0 -1                 D 0 -1
D 1 1                  D 1 -2                 D 1 -1
D 2 -1                 D 2 -2                 D 2 -1
D 3 -1                 D 3 -2                 D 3 -1
```

Results

After running the simulation for roughly ten thousand steps with sixteen entities, less than a minute, it is visible that all entities follow a clear pattern after committing either transaction 4 or 5. That transaction pattern is 4 2 5 3 and repeat. There are a few exceptions, first of all transactions 2 and 3 may be done multiple time in a row and transaction 1 can be committed at any point in the pattern, as the entity idles while its needs are near completely fulfilled.

One other phenomenon can be rarely observed, and that is that an entity sometimes takes the long way from location 1 to 2 using transactions 6 and 5. This most likely happens because moving that way might be perceived as less costly than moving directly followed by some idling in this simulation.

While all entities keep their needs fulfilled by moving back and forth between location 1 and 2, they don't commit their transactions in sync. This shows that the personality as implemented can make entities behave in a unique fashion without interfering with their ability to act reasonably.

The results of this simulation can also be taken as proof that the model succeeds in making entities pursue their survival without fail, as after thousands of steps no entity ended up in a state where it cannot commit any further transactions anymore, which would happen if it would not keep its needs high, the stated goal for survival in this model.

Simulation 2

The second simulation focused heavily on entities trading tools around, which can be used to fulfil their needs without consuming them. This simulation only has one location and can be run by using the "trade" files.

Setup

The state of each entity is initialised to 0=0 1=0 2=10 3=10 4=10 5=0 6=0 7=0 8=2.

As there is only one location, entities always know about all transactions which are currently possible in the entire simulation. There are three needs, resource 2 representing food, 3 representing drink, 4 representing energy. As with the previous simulation they should not fall below 1. For each need, there is a resource representing an item which is used to increase the value of the need. However, each such item must be exchanged with another resource representing coin, and there are ten of each of the

three types of item in the simulation. As there are three needs there are three relevant items however the entity only starts with two coins, thus it needs to constantly trade coins for items, and items for coins.

```
Transaction 1
#Idle
M 2 1
M 3 1
M 4 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1

Transaction 2
#Eat
M 3 1
M 4 1
M 5 1
L 2 20
D 0 -1
D 2 9
D 3 -1
D 4 -1
```

```
Transaction 3
#Drink
M 2 1
M 4 1
M 6 1
L 3 20
D 0 -1
D 2 -1
D 3 9
D 4 -1

Transaction 4
#Sleep
M 2 1
M 3 1
M 7 1
L 4 20
D 0 -1
D 2 -1
D 3 -1
D 4 9
```

```
Transaction 5
#Get Plate
M 2 1
M 3 1
M 4 1
M 8 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 5 1
D 8 -1

Transaction 6
#Get Chalice
M 2 1
M 3 1
M 4 1
M 8 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 6 1
D 8 -1
```

```
Transaction 7
#Get Bed
M 2 1
M 3 1M 4 1
M 8 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 7 1
D 8 -1

Transaction 8
#Give Plate
M 2 1
M 3 1
M 4 1
M 5 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 5 -1
D 8 1
```

```
Transaction 9
#Give Chalice
M 2 1
M 3 1
M 4 1
M 6 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 6 -1
D 8 1

Transaction 10
#Give Bed
M 2 1
M 3 1
M 4 1
M 7 1
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 7 -1
D 8 1
```

Results

There are two types of behaviour apparent in the entities when running this simulation. Some entities keep one type of object and only swap between the two other ones, other entities seem to be content to constantly swap and reacquire all three types of object.

The most likely explanation for this behaviour is that the randomly generated personality of some of the entities makes it assign a much higher value to one type of item than to any of the other two, and thus the entity is unlikely to ever part with it.

The success of this simulation demonstrates that entities can think ahead, as acquiring or selling an item does not directly fulfil any need. This simulation also shows that the implementation is able to properly handle a transactions becoming unavailable after it was committed it, but before it finished.

Simulation 3

The last simulation aims to emulate a rural village. It is moderately complex, featuring 4 locations, 13 resources and 31 transactions. The “rural” files can be used to run this simulation.

Setup

The thirteen resources are 0 Time, 1 Location, 2 Energy, 3 Hunger, 4 Thirst, 5 Warmth, 6 Water, 7 Wood, 8 Grain, 9 Milk, 10 Flour, 11 Bread and 12 Currency. The needs are resources 2 to 5. While the implementation attempts to keep all needs above zero, in this simulation a need can fall as low as -9, as the entity cannot commit any further transactions if a need becomes -10. The entities state is initialised to 0=10000000 1=0 2=12 3=12 4=8 5=12 6=5 7=5 8=5 9=5 10=5 11=5 12=10.

Additionally, all transactions cost time, and if an entity's time becomes 0, the entity cannot act any further and dies. Here, this is added to demonstrate that a concept of finite lifespan can be simulated in the present model, as it is actually unlikely that any entity will ever run out of time while running this simulation. A very rough estimate is that it would take about a decade for the simulation to reach a time in the simulation where entities start dying.

Location 0 represents the entities' home, and here the entities can commit transactions such as sleeping, warming themselves at the stove, or baking bread. Location 1 is the market place, where entities can buy or sell most tangible resources for currency. This can be considered a form of entity interaction as the stock of the market is dependent on what transactions have previously been committed, and thus the actions of the entities directly influence if a sell or buy transaction is currently possible at the market. Location 2 represents a well, where entities can acquire water which can only be increased here, as it is not sellable or buyable. Location 3 stands for the outskirts of the village, where entities can acquire raw resources such as wood, milk or grain, and refine grain into flour.

The transactions with an id lower than 100 are perceived in location 0, while all other transactions are perceived in location 1. Transaction 1 represents idling. Transactions 2 to 5 are transactions used to fulfil need directly. Transactions 10 to 15 represent the gathering of various resource. The transactions whose id is between 70 and 99 represent movement between different locations. Finally, the transactions with ids greater than 100 are trading transactions.

```
Transaction 1
#Idle
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 5 -9
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 5 -1
```

```
Transaction 2
#Sleep
R 1 0
M 0 5
M 2 -3
M 3 -3
M 4 -3
M 5 -3
L 2 24
D 0 -4
D 3 -2
D 4 -2
D 5 -2
D 2 12
```

```
Transaction 3
#Eating Bread
M 0 0.1
M 11 3
L 3 12
D 0 -0.1
D 3 24
D 11 -3
```

```
Transaction 4
#Drinking Water
M 0 0.1
M 6 1
L 4 8
D 0 -0.1
D 4 8
D 6 -1
```

```
Transaction 5
#Feeding the Fire
R 1 0
M 0 0.1
M 7 1
L 5 12
D 0 -0.1
D 5 24
D 7 -1
```

Transaction 10
#Backing Bread
R 1 0
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 6 1
M 7 1
M 9 1
M 10 1
L 11 13
D 0 -1
D 2 -1
D 3 -1
D 4 -1
D 6 -1
D 7 -1
D 9 -1
D 10 -1
D 11 3

Transaction 11
#Draw from well
R 1 2
M 0 0.1
M 2 -9
L 6 13
D 0 -0.1
D 2 -1
D 6 3

Transaction 12
#Milk a Cow
R 1 3
M 0 1
M 2 -8
M 3 -9
M 4 -9
M 5 -9
L 9 14
D 0 -1
D 2 -2
D 3 -1
D 4 -1
D 5 -1
D 9 2

Transaction 13
#Grind Flour
R 1 3
M 0 1
M 2 -8
M 3 -9
M 4 -9
M 5 -9
M 8 1
L 10 14
D 0 -1
D 2 -2
D 3 -1
D 4 -1
D 5 -1
D 8 -1
D 10 2

Transaction 14
#Cut Trees
R 1 3
M 0 4
M 2 -4
M 3 -6
M 4 -6
M 5 -6
L 7 9
D 0 -4
D 2 -6
D 3 -4
D 4 -4
D 5 -4
D 7 8

Transaction 15
#Gather Grain
R 1 3
M 0 2
M 2 -7
M 3 -8
M 4 -8
M 5 -8
L 8 12
D 0 -2
D 2 -3
D 3 -2
D 4 -2
D 5 -2
D 8 4

Transaction 91
#Home to Market
R 1 0
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 5 -9
D 0 -1
D 1 1
D 2 -1
D 3 -1
D 4 -1
D 5 -1

Transaction 81
#Market to Home
R 1 1
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 5 -9
D 0 -1
D 1 -1
D 2 -1
D 3 -1
D 4 -1
D 5 -1

Transaction 82
#Market to Well
R 1 1
M 0 0.5
M 2 -9.5
M 3 -9.5
M 4 -9.5
M 5 -9.5
D 0 -0.5
D 2 -0.5
D 3 -0.5
D 4 -0.5
D 5 -0.5
D 1 1

Transaction 83
#Market to
#Resources
R 1 1
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 5 -9
D 0 -1
D 1 2
D 2 -1
D 3 -1
D 4 -1
D 5 -1

Transaction 72
#Well to Market
R 1 2
M 0 0.5
M 2 -9.5
M 3 -9.5
M 4 -9.5
M 5 -9.5
D 0 -0.5
D 2 -0.5
D 3 -0.5
D 4 -0.5
D 5 -0.5
D 1 -1

Transaction 73
#Resources to
#Market
R 1 3
M 0 1
M 2 -9
M 3 -9
M 4 -9
M 5 -9
D 0 -1
D 1 -2
D 2 -1
D 3 -1
D 4 -1
D 5 -1

Transaction 101
#Sell Grain
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 8 1
L 12 23
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 8 -1
D 12 2

Transaction 102
#Sell Flour
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 10 1
L 12 21
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 10 -1
D 12 4

Transaction 103
#Sell Wood
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 7 1
L 12 23
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 7 -1
D 12 2

Transaction 104
#Sell Milk
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 9 1
L 12 23
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 9 -1
D 12 2

```
Transaction 105
#Sell Bread
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 11 1
L 12 16
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 11 -1
D 12 9
```

```
Transaction 201
#Buy Grain
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 12 3
L 8 15
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 8 1
D 12 -3
```

```
Transaction 202
#Buy Flour
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 12 5
L 10 15
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 10 1
D 12 -5
```

```
Transaction 203
#Buy Wood
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 12 3
L 7 15
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 7 1
D 12 -3
```

```
Transaction 204
#Buy Milk
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 12 3
L 9 15
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 9 1
D 12 -3
```

```
Transaction 205
#Buy Bread
R 1 1
M 0 0.1
M 2 -9.9
M 3 -9.9
M 4 -9.9
M 5 -9.9
M 12 10
L 11 15
D 0 -0.1
D 2 -0.1
D 3 -0.1
D 4 -0.1
D 5 -0.1
D 11 1
D 12 -10
```

Results

After running this simulation for about two hours, each entity performed about 200 transactions. The difference between the number of transactions performed stems from the fact that transactions in this simulation take varying amounts of time.

One entity died once, presumably of thirst, as its last action was to go to the well. While the ideal result would be that no entity ends up in a state where it cannot commit any further transactions, that only one entity died can still be considered in a positive light given the complexity of the simulation and the lack of fine tuning of the learning algorithm employed.

Overall, entities committed a broad range of the transactions available to them. Although it can be observed that each entity has different priorities. This can be seen in that some entities committed much more trade transactions than others, and the entities also differed between which trade transactions they mostly committed.

Generally speaking, the entities appear lifelike as far as can be determined from the log of such a crude simulation of reality.

9. Conclusions

After testing the model with the simulations described above it can be concluded that the model itself is fully capable of simulating lifelike autonomous entities according to the requirements listed in the aims of this dissertation. It does however suffer from the practical weaknesses of requiring significant computational resources for complex simulations, which will need to be addressed to use this model for practical simulations and not solely to prove that it is sound and works.

As the current implementation shows, the entity itself does not use any knowledge of the domain of the simulation except when randomly initialising personality as resources with nominal values are given zero personality coefficient to represent that changes in them have neither cost nor benefit. This both allows it to simulate anything, and allows to postulate what its general performance for a specific simulation would be based on the performance of another simulation, as in principle the single difference between simulations is that the used numbers differ.

What also can also be seen from the results is that computational resources currently limit the maximum complexity of simulations which can be run at reasonable speeds. For example, the third simulation was run for approximately two hours but only resulted in around 200 commits for each of the sixteen simulated entities.

The limitations on performance in this model can be particularly felt if attempting to create a simulation where there exist many not directly beneficial transactions between the beneficial transactions. For example, a simulation with many locations, as this would increase the number of steps the entity needs to look ahead which causes an exponential grow of required computational resources.

While this limitation will be felt less over time as computers become faster, and could already be somewhat alleviated by implementing it specifically for performance using parallel computing frameworks such as OpenCL or Vulkan to fully utilise the capabilities of current machines, it will always represent a ceiling of performance for this model as it does for many other models which attempt to simulate intelligence in some form.

10. Further Work

While the work described in this dissertation proves that the model developed is usable as is, there is still room for much further work left. Both in the sense that the implementation can still be much improved and expanded, and in the sense that much theoretical work is still undone, as this dissertation only provides the barebones proving that the presented ideas live up to their expectations.

Priority Learning

The model itself does not define how an entity computes the priorities of its needs during each step, or if and how this is subject to a form of machine learning. The current implementation uses a neural network, which computes these priorities based on the entities entire current state. While direct observation of the neural net shows that it is indeed learning, there is no easy way to tell if it really does give better results than ordering the needs only by how much they are currently fulfilled. In simple simulations such as the ones presented here, including the slightly more complex third, it is not expected that there will be much difference. However, crafting and running a simulation complex enough to either prove or disprove the utility of the neural network, or any machine learning algorithm in general over a plain algorithm, is a time consuming activity which has to be done another time.

Transaction Selection

The best transaction is currently determined by dividing their positive changes by their negative changes. To look ahead a sequence of transactions is first summed up then treated as one single transaction.

While this method has been carefully chosen to remove as much bias from differing magnitudes and such, it is at the end the result of a number of, while informed, nevertheless arbitrary choices, which are validated only through the apparent functionality of the implementation. Even without changing the core idea of this ranking scheme there are many possible variations which could be tested to see if they give a better result than the current method.

Beyond that there may well be more complex methods of ranking and look ahead which are worth consideration, although the rule that a simple solution is a better solution may well be applicable here.

Non-deterministic transactions

Non-deterministic transactions are discussed in this dissertation; however, they are not found in the implementation. The reason for this lies in the fact that while they require no work on the entities side, they do require an entire mechanism on the world's side, both to tag them as being non deterministic and in the algorithms needed to determine their outcome. Consequently, due to time constraints this feature was dropped for this submission.

There are however many uses which can be conjectured for non-deterministic transactions. The two most prominent are that they allow for much more organic entity to entity interaction, and that they can be used as placeholder transactions representing general knowledge the entities might have without actually being privy to the exact details. For example, an entity might know that it can acquire food somewhere but not which food. Another use of placeholder transactions might be for transactions which the world, for some reason, wants to make seem either more or less attractive than they in reality are.

11. Summary

In this dissertation some of the most important features of lifelike entities have been found to be autonomy, purpose, reason and constant change. Autonomy means that the entities have an internal state and act only based on that state, their perception, and their own volition, and never on the behest of another; although they may, of course, follow suggestions of others if they choose to do so. Purpose means that their actions are guided by a defined goal or goals. For lifelike entities, this goal is, put in its simplest form, survival with means to simplify that goal existing as secondary goals. The notion of reason extends upon the notion of purpose, in that entities not only act on purpose but also act in an intelligent or reasonable manner. Constant change is to be understood as the fact that, as long an entity is alive, meaning lifelike, its inner state is undergoing constant change.

An overview has been given of some related concepts found in the scientific literature, such as agents, artificial intelligence, state machines and cognitive modelling. The model proposed in this dissertation understands entities in much the same way as agents are understood, and models the internal state of entities and their actions akin to state machines while taking some inspiration from cognitive modelling.

A model has been proposed to simulate such lifelike entities in a broad range of contexts. The model's main points are that it is fully domain independent, as entities can reason without needing domain specific knowledge, and that it encodes all interaction between entities and their environment using only one single type of information. Each entity possesses a state which can be visualised as a state space with a number of dimensions equal to the number of distinct resources the entity is aware of. The purpose of each entity is to remain in those areas of that space in which the resources considered needs are high. To move around this state space, entries commit transactions which represent discrete atomic changes to the entities' state. Depending on its current state, an entity might perceive additional transactions it can commit, or perceive information causing it to forget previously learned transactions. Any transaction which is defined as causing no changes to an entity's state is illegal in this model as the state of an entity may never cease to change in some way.

This model has been implemented in the C++ language. Test simulations run using this implementation show that the model works in a practical setting. As is often the case, practical considerations mean that not every detail found in the model is strictly implemented as written. For example, the implementation uses a transaction which doesn't change an entity's state at all, which is not allowed in the model but in the implementation is used as a convenient way of signalling that the committing entity has died as it cannot find a legal transaction to commit. Nevertheless, the author believes that the differences are insignificant enough to not prevent the implementation from being used to draw conclusions about the model itself.

The strengths of the model lie majorly in the domain of flexibility and simplicity. There is only one core mechanic, which is the transaction, and this can be used to describe anything. The drawbacks of the model are that the computational resources needed to use it rise exponentially with the complexity of the simulated environment, although this is hardly the only model of this sort which suffers from needing a large of computational resources to do simulation complex enough to be of practical use.

This dissertation contains the proof that the core ideas of the model are sound and usable in practice, but also shows that there are many facets and concepts related or pertaining to this model which have not been addressed or fully considered yet, as to do so would have been outside of the scope of this dissertation. All of these are worth to be looked into at a future time.

Appendices

References

[1] S. Franklin and A. Graesser, "Is It an agent, or just a program?: A taxonomy for autonomous agents," in *Third International Workshop on Agent Theories, Architectures, and Languages*, Budapest, 1996.

[2] S. Russell and P. Norvig, Artificial Intelligence A Modern Approach, Third ed., Prentice Hall, 2009.

[3] D. Smith, A. Cypher and J. Spohrer, "KidSim: Programming Agents Without a Programming Language," *Communications of the ACM,* vol. 37, no. 7, pp. 55-67, 1994.

[4] J. Brustoloni, "Autonomous Agents: Characterization and Requirements," Carnegie Mellon University, Pittsburgh, 1991.

[5] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artificial Intelligence,* vol. 72, no. 1-2, pp. 329-365, 1995.

[6] S. Cheng and W. Bin, "An Overview of Publications on Artificial Intelligence Research: A Quantitative Analysis on Recent Papers," in *Fifth International Joint Conference on Computational Sciences and Optimization (CSO)*, Harbin, 2012.

[7] A. Turing, "Computing Machinery and Intelligence," *Mind,* vol. LIX, no. 236, pp. 433-460, 1950.

[8] R. Chong and R. Wray, "Unified Theories of Cognition," in *Encyclopedia of Cognitive Science*,

John Wiley & Sons, Ltd, 2006.

[9] M. Genesereth and N. Nilsson, Logical Foundations of Artificial Intelligence, San Francisco: Morgan Kaufmann Publishers Inc., 1987.

[10] M. Wooldridge, Reasoning about Rational Agents, Cambridge: The MIT Press, 2000.

[11] R. Sun, The Cambridge Handbook of Computational Psychology, Cambridge University Press, 2008.

[12] A. Newell, Unified Theories of Cognition, The William James Lectures, Harvard University Press, 1990.

[13] J. Anderson, D. Bothell, M. Byrne, S. Douglass, C. Lebiere and Y. Qin, "An integrated theory of the mind," *Psychological Review,* vol. 111, no. 4, pp. 1036-1060, 2004.

[14] D. Kieras, "EPIC Architecture Principles of Operation," 19 07 2004. [Online]. Available: http://ix.cs.uoregon.edu/~hornof/downloads/EPIC_Tutorial/EPICPrinOp.pdf. [Accessed 17 08 2016].

[15] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming,* vol. 8, no. 3, pp. 231-274, 1987.

[16] J. G. Brookshear, Theory of Computation: Formal Languages, Automata, and Complexity, Redwood City: Pearson, 1989.

Implementation Readme

The implementation and the files to run the described simulations are separately provided in a zip file. Both the files needed to run the simulations and outputs of sample runs can be found in the "Simulations" folder. The c++ code and Microsoft visual studio project files to build it are provided in the "Code" folder.

There are two projects in the solution. The server project contains the code for the actual simulation and the analysis project contains code for a small utility program which can be used to parse the log file generated when running a simulation, to either summarise what happened in the simulation or to split the actions of one particular entity into a separate log file.

Both programs are to be run from a terminal.

The main program will if given two arguments interpret the first argument as the path for the log to be created and the second argument as the path of the configuration file which describes the simulation to be run. If a different number of arguments is provided, then the supplied arguments are ignored and the default paths of "./log.log" and "./sim.cfg" are used.

The analysis program similarly takes either two or three arguments. If two arguments are provided, then the first is the path where a summary of the events in the simulation will be written to and the second argument will be understood as the path to the log file produced by the simulation to be summarised. If three arguments are given, then the utility will write all actions pertaining to an entity with an id given in the third argument found in a log file located at the path defined in the second argument to a file who is written to the location pointed at in the first argument. If a different number of arguments is given, the program defaults to writing a summary to "./summary.log" based on a simulation log expected to be at "./log.log".

Custom Simulations

To run custom simulations a configuration file and a simulation file need to be created.

Configuration File

The configuration file contains parameters pertaining to the simulation. Each parameter is defined on its own line by writing the name at the start of the line and adding a space separated list of values to it.

The parameters are:

- "SimPath" whose value is the path to the simulation file.
- "NumberNeeds" whose value is an integer equal to the number of needs in the simulation.
- "NumberRessources" whose value is an integer equal to the number of resources in the simulation.

- “SequenceDepth” whose value is an integer equal to the number of steps of look ahead the entities in the simulation perform beyond the current transaction.
- “EntityNumber” whose value is an integer equal to the number of entities in the simulation.
- “InventoryIds” whose value is a list of integer denoting all resource ids used in the simulation.
- “InitialiseInventory” whose value is a list of real values used to initialise entities’ state. The ordering is the same as in “InventoryIds”.
- “NeedIds” whose value is a list of integers of the ids of all need resources used in the simulation.
- “TimeId” whose value is the integer equal to id of the resource representing time.
- “StateRessourceIds” whose value is a list of integers of the ids of all modal resources used in the simulation.
- “StockTransactions” whose value is a list of integers of the ids of all transactions, which can change the resources available at a location, used in the simulation.

Simulation File

The simulation file contains all locations and entities. The simulation files are written using a custom markup language. The formal syntax for this is provided on the next page. It is suggested that the example simulation files supplied with the code are studied before attempting to create such a file, “example.ses” in particular as it describes all elements in detail using comments.

The lowest elements in the hierarchy, those prefixed with a single ‘?’, describe maps found in data structures found in the implementation. The meaning of those is described earlier in this dissertation.

The lines inside these are always in form of key space value where key is an integer and value is a real number. The exception are the inner lines of transaction which add a letter in front of the key value pair. The meanings of those letters are:

- R means that the transaction is possible if the value of the resource in the entities state is equal to the value listed here.
- L means that the transaction is possible if the value of the resource in the entities state is equal or less than the value listed here.
- M means that the transaction is possible if the value of the resource in the entities state is equal or greater than the value listed here.
- D means that the transaction changes the resource in the entity’s state by this value when the transaction is committed by the entity.

Simulation file syntax

END ::= nothing
EMPTY ::= empty string
COMMENT ::= '#' arbitrary text
ID ::= integer
SEED ::= integer
VALUE ::= real number
PAIR ::= ID VALUE
FILE ::= ELEMENTS
ELEMENTS ::= ELEMENT '\n' ELEMENTS
ELEMENT ::= EMPTY | COMMENT | SIMULATION | END
SIMULATION ::= SIM_HEADER SIM_ELEMENTS "!!!"
SIM_HEADER ::= "??? Simulation" SEED
SIM_ELEMENTS ::= SIM_ELEMENT '\n' SIM_ELEMENTS
SIM_ELEMENT ::= COMMENT | EMPTY | LOCATION | ENTITY | END
LOCATION ::= LOC_HEADER LOC_ELEMENTS "!!"
LOC_HEADER ::= "?? Location" ID
LOC_ELEMENTS ::= LOC_ELEMENT '\n' LOC_ELEMENTS
LOC_ELEMENT ::= COMMENT | EMPTY | TRANSACTION | STOCK | END
TRANSACTON ::= ACT_HEADER ACT _ELEMENTS "!"
ACT _HEADER ::= "? Transaction" ID
ACT _ELEMENTS ::= ACT _ELEMENT '\n' ACT _ELEMENTS
ACT _ELEMENT ::= COMMENT | EMPTY | LESS | MORE | EXACT | CHANGE | END
LESS ::= 'L' PAIR
MORE ::= 'M' PAIR
EXACT ::= 'R' PAIR
CHANGE ::= 'D' PAIR
STOCK ::= STK_HEADER STK _ELEMENTS "!"
STK _HEADER ::= "? Stock" ID
STK _ELEMENTS ::= STK _ELEMENT '\n' STK _ELEMENTS
STK _ELEMENT ::= COMMENT | EMPTY | PAIR | END
ENTITY ::= ENT_HEADER ENT _ELEMENTS "!!"
ENT_HEADER ::= "?? Entity" ID
ENT _ELEMENTS ::= ENT _ELEMENT '\n' ENT _ELEMENTS
ENT _ELEMENT ::= COMMENT | EMPTY | TRANSACTION | NEED |
INVENTORY | PERSONALITY | END
NEED ::= NED_HEADER NED _ELEMENTS "!"
NED _HEADER ::= "? Need" ID
NED _ELEMENTS ::= NED _ELEMENT '\n' NED _ELEMENTS
NED _ELEMENT ::= COMMENT | EMPTY | PAIR | END
INVENTORY ::= INV_HEADER STK _ELEMENTS "!"
INV_HEADER ::= "? Inventory" ID
PERSONALITY ::= PER_HEADER PER _ELEMENTS "!"
PER _HEADER ::= "? Personality" ID
PER _ELEMENTS ::= PER _ELEMENT '\n' PER _ELEMENTS
PER _ELEMENT ::= COMMENT | EMPTY | PAIR | END