DOI 10.34658/jacs.2018.26.2.161-186

JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 26 No. 2 (2018), pp. 161-186

A Memory Model for Emotional Decision-Making Agent in a Game

Jakub Rogalski, Dominik Szajerman

Lodz University of Technology
Institute of Information Technology
Wólczańska 215, 90-924 Łódź, Poland
dominik.szajerman@p.lodz.pl

1. Introduction

Abstract. *Virtual characters are an important part of many modern computer games. This paper describes a graph-based memory system designed for artificial agents that also simulate simple emotions. The system was tested using virtual simulation environment and it showed many new and desirable AI behaviours. These behaviours include simple preferences, reactions based on bot's opinion of a stimuli or improvement of bot's ability to find objects to interact with.*

Keywords: *computer games, artificial intelligence.*

Since the start of the video games industry, the creators confronted the matter of creating smart and challenging non-player characters' intelligence. A few AI systems were designed solving various problems: the path-finding, navigation mesh generation, decision making or adapting enemies. Decision making developed into: decision trees, state machines or behavior trees. With the help of these systems, designers could easily create, manage and extend existing AI behaviors, what led to great improvement of game world for players. However, there is one significant issue with each of these algorithms. All they are based on rational decision making. This affects the believability of NPC – the player could find out how a character works and predict its choices. This led to a deeper reflection on the possibility of simulating the human psyche, for example by taking into account emotions [1, 2, 3, 4] and memory.

In this article we present a simple and easy to implement model for simulating rule-based agents with emotions and memory. There is also a description of a simulation environment along with a series of tests that were conducted. The simulation was created for the purpose of properly verifying many of the model's aspects and categorizing any unexpected behaviours that result from the essence of the model.

2. Related work

The creator of [5] has expanded decision trees AI by taking agent's emotions into account during the decision-making process. This enabled the agents to adjust their behaviour depending on their emotional state. That also made actors to behave in a less predictable way, which in term made the agents more realistic [6]. Has created a multi-layer emotion model composed, among others, of perception, emotions, motivations and behaviour.

Overall, modelling advanced memory or personality in games is not used as much as it could be due to the fact that in most cases it would not have much of an influence on the behaviour of agents: in most cases their behavior is based just on their role in the simulation. In cases where it is used, the memories are related to a very specific, feature focused thing, like bot's opinion of player or other agent.

Long-term memory model introduced in [7] has made it possible to create bots that were learning during the gameplay. They were successfully used in logic games, first-person shooters and real-time strategy games. In case of this model, the main focus was put on the maximizing the chance to win at the game, not on the believability of the AI.

Short term memory and limited perception that were simulated by [8] has greatly reduced the amount of information agent has access to. This in term has led to a situation the bots were making decisions based just on what they should know, which had a positive effect on the believability of the bots (and reduced the computational complexity of finding an action to perform). In this case though, the influence of emotions was not touched upon.

3. Data

Before going on with the descriptions on how exactly the presented model works, it is worthwhile to first introduce and describe the data types being used in the calculations. There are three main types of information we'll touch upon in this section:

1. Agent emotions.
2. Behaviour rules.
3. Memory graph.

3.1. Emotions

The proposed model is to a certain extent based on the emotional decision-making agent described in [2]. Its author has distinguished the three main types of stimuli that affect the actions performed by a bot: fear, pain and anger. This choice was dictated by the consistency of these stimuli between different cultures and social groups. Thanks to the fact that chosen stimuli were negative, the bot's responses could be more direct, which in term made the resulting agent behaviours much more emotionally responsive. Moreover, instead of using mathematical descriptions for the emotional state of the bot, the author has used a fuzzy logic system. This has greatly simplified the definition of behavior rules and the process of their selection. Overall, this made the model more readable. On the other hand, the choice of just the negative feelings only imposed the nature of the activities performed, which were mostly defensive responses to negative stimuli. In addition, the feeling of pain (although it worked out fine in this case) is not an emotion itself.

In order for bots to be able to express both positive and negative emotions we decided to use 4 different emotion scales that were first introduced in [9]. Each of the scales represents two opposing, conflicting emotions at its ends. The scales are:

1. sadness – happiness,
2. disgust – acceptance,
3. fear – anger,

4. surprise – awaitance.

Each of those emotions (E) is in the numerical range $[0, 1]$. E equal to $0,5$ is the neutral position, meaning that neither of emotions in the scale is manifesting. In case of scale of sadness – happiness, the greater E is, the happier a bot is. The opposite is true for sadness. Full emotional state (ES) can be then represented as a vector of four numbers. A neutral ES is equal to $(0.5, 0.5, 0.5, 0.5)$. ES of a bot that has suddenly found itself in a scary situation can be equal to: $(0.5, 0.5, 0.1, 0.1)$.

Every rule available to bot has a set of up to four emotion preferences: each for a different kind of emotion. These can be also represented as ES. A rule named "Run" may have a following description: $(-, -, 0.0, -)$. Unspecified emotion values are replaced with "–" and are not taken under consideration when picking a rule to perform. Rules also may define result ES and action perception ES similarly to rule emotion preferences.

Whenever there is a need to measure how two different emotional states are similar to each other, a state distance (SD) is calculated:

1. For each E defined in both states, their difference is added to total distance as if to calculate Manhattan distance.

2. Then, the total distance is divided by the number of additions made in step one to create average difference of ES components.

3. Finally, the value is raised to the power of two and subtracted from 1.

These steps produce a floating-point number ranging from zero to one that describes how two states are similar to each other (with value equal to one meaning states are equal).

3.2. Rules

A behaviour rule in the proposed model can be described by a set of the following elements:

$rule_{action}$, which is an action related to the rule, the actor will perform it whenever the associated rule is picked. The fact of performing this action itself is called the event. For example, in a typical game, the rule can be defined as follows:

- **Condition**: life points below twenty percent of their maximum value.

- **Action**: drink a healing potion.
- **Event**: the potion was drunk.

$rule_{preference} := ES$ defines the emotional state in which the bot should be, so that the chance that a given rule is selected $p(ES)$ is the largest.

$$rule_{preference} \rightarrow \quad \nexists S := ES \quad p(S) > p(rule_{preference}). \tag{1}$$

$rule_{result}$ contains information about the emotional state to which the agent's state will get closer after the event. $rule_{result}$ is represented in form of an ES vector, as well as for each of its defined elements: an effect. The effect is in the numeric interval of $[0, 1]$ and describes the maximum change of a given emotion after the event.

$rule_{perception}$ contains information about the emotional state to which the status of the bot will get closer after observing another bot performing the given action. It is represented the in same way as the $rule_{result}$.

$rule_{participants}$ describes all participants of the action. For each of them, a preferred opinion is defined in a similar form to $rule_{result}$. Also, it contains required object type as well as a list of required tags for each of the participants.

$rule_{importance}$ is a importance factor that acts as a multiplier for the probability of selection of the rule.

3.3. Memory

Many of the rules available for bots may require some kind of interaction with the simulation environment. In many cases there will not be any matching rule participants close to the bot. For these kind if situations we extended the base system by introducing a memory graph that stores all of the information a bot currently remembers.

Two main kinds of elements of the graph: vertices and edges represent different things. Each vertex represents a memory of an unique in-game-object or a past event. It contains info about the object's type, its unique characteristics and bot's own opinion of a memory (in form of an ES). An edge contains info about bot's association of two different memories, its importance and is mainly just used in the process of browsing bot's memory.

There are three main types of memory objects:

- location,

- person or item,
- event.

During the simulation, a memory graph is constantly undergoing changes because of bot's perception and actions he performs. There are five cases when bot's memory is modified:

1. Bot enters location *A* for a first time: add a vertex for each location *A* is connected to and connect them with *A* node.
2. Bot sees a person or an item *A* for the first time: add a vertex that represents a given object and connect it by an edge with bots current location.
3. Bot sees a person performing an action: add a vertex representing a memory of that action and connect it with actions participants and persons current location. Slightly modify opinion of a person perceived depending of action perceived.
4. Bot finishes performing an action: add a vertex representing a memory of that action. Connect it with current bots location and all other objects that participated in it. Modify opinion of actions participants depending on a rule finished.
5. Bot is told about something, discovers information when reading or other means: a set of new vertices and edges is added to the graph.

Memory graph is regularly browsed for potential participants. First step of this process is to pick a small group of starting memory nodes (hook nodes). This group consists of some randomly picked nodes. Some of them are currently perceived by the bot, recently added to the graph or just are the ones that were modified recently. Then, each hook node picked is analyzed separately from others in a following way:

1. The node is analyzed for whether it is a good participant for a rule in question: its type, characteristics and bot's opinion of it are tested.
2. If node is indeed a fitting participant, the node is added to a potential participants group.

3. Another unvisited node connected to the current one is picked to replace current node. The node is picked using using weighted random selection based on connection importance.

4. After that, process returns to step 1 and repeats 5-10 times or until there are no more possible nodes to go to.

This whole process can be repeated multiple times per node, thus producing multiple paths through memory graph. After All potential rule participants are picked, five best ones are selected and from among these, then one is selected as final participant using weighted random selection (the better the participant fits, the greater his chance of being picked). Memory edges leading to final action participant from the original hook node also have their importance slightly raised and thus will be more likely to be visited in future. In case there are no potential participants available, another rule is considered.

After all requirements for a rule are met, the bot can begin to perform an action associated with the rule. Actions can be easily implemented and added to simulation by designers and define precisely how a bot behaves after a rule is picked.

4. Decission process

The one of the main goals of the model being described in this paper, is that agents have to make decisions based on their emotional state and memory. The general scheme of the model is presented in figure 1.

4.1. Picking a rule

The first step in the decision making process is the initial selection of the few best actions that will be considered further. Each rule that a bot can perform includes a $rule_{preference}$ value which describes a state in which the bot is most likely to pick that rule. By using it, you can easily calculate the difference between it and the current mood of the actor $actor_{state} := ES$. It is defined as the average of squared differences between successive corresponding elements of two vectors. Let us assume the following:

$$\begin{aligned} 1 \leq k \leq 4, \qquad k \in \mathbb{N} \\ a_1, a_2, ..., a_k \\ i_1, i_2, ..., i_k. \end{aligned} \tag{2}$$

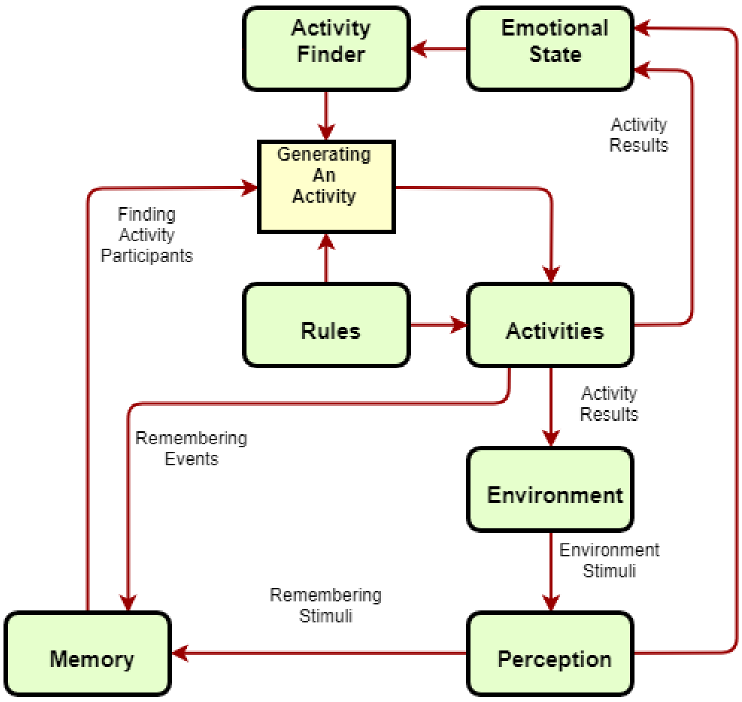

Figure 1: A scheme of the model.

k represents the number of emotions compared. The $a_1, a_2, ..., a_k$ are the distinguished E values of agent status, located, respectively, at the coordinates $i_1, i_2, ..., i_k$ of ES vector.

We can then distinguish T, which contains all possible sets of emotion values that are compared in a test:

$$T = \{p \in X \times Y \ : \ pr_{i_1}(p) = a_1 \wedge ... \wedge pr_{i_k}(p) = a_k\} \tag{3}$$

T is then a set consisting of only those vectors which perfectly meet our expectations as to the values of the agent's condition being tested. We can choose any vector of features p, which will serve as a specific set of required ES. Then, with the use of p, we can calculate the final result of the $score(p)$:

$$score(p) = \frac{\sum_{j=1}^{k}(pr_j(p) - a_j)^2}{k} \tag{4}$$

Whenever a rule does not define certain emotion in $rule_{preference}$ (for example anger), this emotion is not taken into account when calculating the distance. Thanks to that, regardless of the number of defined emotions, the calculated $score(p)$ will always be in the range $[0, 1]$.

Out of all rules, 5 are selected with the profiles that are closest to the current state of the actor, i.e. those whose calculated $score(p)$ is the smallest.

Then, with the calculated distances one rule is selected with the use of weighted randomization. It is done in a following way:

1. For each distance o_i the similarity of p_i is defined as: $p_i = 1 - o_i$.
2. Then the sum of all similarities is calculated $P = \sum_{i=1}^{n} p_i$
3. One number $X = [0, P]$ is picked randomly.
4. Finally, all selected rules $rule_i$ are considered one after another. For each of them, p_i is subtracted from the X value.
5. When the value of X after subtracting the any p_i will become lesser than or equal to 0, the $rule_i$ is chosen.

In case the chosen rule turns out to be impossible to perform, it is possible to easily return to this stage from further parts of the algorithm.

4.2. Choosing Participants

The next step is to select participants for the rule picked. Each $rule_{participant}$ defined in the rule must be matched with an appropriately similar object in the simulation. Otherwise, the rule will not be possible to perform.

The method of browsing the memory graph used in this model is largely based on the ant colony graph search optimization algorithm proposed by Marco Dorigo in [10]. It is a heuristic algorithm for optimizing the search for the most efficient path in a graph. In his algorithm descriptions, he compares its functionality to the the colonies of ants which seek food. At first, the ants move around the area (along edges of the graph) in chaotic manner and look for food (in this case for the appropriate vertex). If any of them finds a valid path, it will return to the colony using the same way it came from and leaving behind a trail of pheromones. This trace informs other ants that it can lead them to food, and this in turn increases the chance that they will at least partially follow that trail. This does not guarantee that future ants will use exactly the same route, but instead they could arrive at the destination using a different, slightly modified and possibly better path, and then leave behind even stronger trail with pheromones (because their path was better). After some time, pheromones gradually evaporate. This in turn leaves only the best paths clearly marked in the graph.

Having how ant algorithm works in mind, we can begin describing the participant selection process. It is initiated by selecting a certain subgroup of the starting vertices from the memory graph. Each of these corresponds to ant colony (but in this case there are more than one colonies). The size of this group usually does not exceed ten elements. This group consists of:

- some randomly selected objects within the bot's perception range,
- recently memorized vertices,
- vertices used recently in past actions,
- objects already found and used in the action,
- randomly selected vertices of the graph.

Then, from each of those vertices, five to ten randomly selected paths are lead. The chance of path crossing any edge of the graph adjacent to the current end is proportional to its weight $edge_{importance}$ (which corresponds to the concentration of pheromones in the ant colony algorithm). Each of paths created has a length of up to five edges. The path can not contain the same vertex twice. In some cases there are situations in which the path cannot go anywhere, because all possible edges coming out of the vertex have already been visited. In this case, the final path simply ends at the problematic tip and ends up being shorter.

Each vertex on the path is tested whether it can be used in an action related to the currently selected rule. First, types of vertex and the searched object are compared. Then the emotional profile is tested. The distance between participant profile and object opinion is calculated. It is done slightly differently way than when calculating the rule distance: the average of the distance of select ES elements is calculated. If the distance exceeds a certain limit, the object is discarded. After analyzing all the paths created, one of the vertices found is selected randomly. Those vertices that match the profile more closely have a better chance of being chosen. The last condition is to make sure that tags of the vertex contain all of the $rule_{participant_{tags}}$, i.e.:

$$rule_{participant_{tags}} \in v_{tags} \tag{5}$$

In some cases, it is required that a certain tag must not be present in a given vertex. In such situations, a '!' sign is added at the beginning of the unwanted tag. If a rule required that the action objective was a clean place to sit, the participant tags could look like this: *(SEAT, !DIRTY)*.

The last stage of the selection process is to select just one participant for each participant required. It is simply done by randomly selecting one of all potential participants. An alternative to this solution can also be to use the participant's calculated score and then either selecting the best one or using a weighted random selection.

If no proper candidate for any of the participants is found, the whole rule is rejected and then the whole process goes back to rule selection phase. The other newly selected rule one is added to the pool of the five best rules and one of them is picked randomly again.

The method of searching for rule participants is to an extent inspired by the process of thinking about subsequent object associations. One can often think about a few seemingly unrelated objects through a series of associations and then come up with another one that suits the activity currently being considered. It is also worth noting that the computational complexity of single search in memory is not dependent on its size: this approach allows to achieve a large improvement compared to when all the objects remembered would be considered for a candidate.

5. Mood Adjustment

After the action associated with rule picked is completed and the modifications are made to the memory graph, emotional state of the agent is adjusted. This process is divided into three smaller stages:

1. Soothing phase.
2. Participant impact phase.
3. Action impact phase.

However, they are described in a modified order, because the stage of participants' influence is easier to explain when basing on the action impact description.

5.1. Soothing phase

The initial step in the mood adjustment process is to modify the emotional state so that it becomes closer to its neutral value, i.e. $ES = (0.5, 0.5, 0.5, 5.5)$. It imitates the effect of agent calming down after recent events.

Let's assume a certain value s:

$$s = [0, 1], \tag{6}$$

and call it a **soothe factor**. Its value will tell us how much the emotional state of the agent will approach the neutral state after the stage is completed. With s it is possible to express the values of successive elements of the agent's emotional state $\alpha_0, ..., \alpha_n$:

$$\forall_{0 \le k \le n} \qquad \alpha_k = s * 0.5 + (1 - s) * pr_k(agent_{state}). \tag{7}$$

Based on the expression above, it is easy to deduce that for $s = 1$ the resulting value will always be equal to 0.5. Also, for $s = 0$ the value will not change. By adjusting the value of s you can control the rate of soothing agents' emotions in a simple way.

The emotional state of an agent after this phase can be described as:

$$agent_{state} := (\alpha_0, ..., \alpha_n). \tag{8}$$

5.2. Action impact phase

Each rule defines a certain result $rule_{result}$ and features one effect weight per each element defined of the result: $w_1, ..., w_k$. These weights are located on coordinates $i_1, ..., i_k$ of the $rule_{result}$ vector. Let's take the following assumptions:

quantity of $agent_{state}$ elements affected by the rule: λ.

vector containing coordinates of $agent_{state}$ elements affected:

$$I = \{i_0, ..., i_\lambda\}. \tag{9}$$

vector containing effect weights associated with rule result:

$$W = (w_{i0}, ..., w_i). \tag{10}$$

Let's define a MoveTo function:

$$MoveTo : [0,1] \times [0,1] \times \mathbb{R} \to [0,1], \tag{11}$$

where MoveTo is described by a following formula:

$$MoveTo(x, y, t) := \begin{cases} \frac{y-x}{\|y-x\|} \cdot t & if \;\; t < \|y - x\|, \\ y - x, & otherwise. \end{cases} \tag{12}$$

Therefore the MoveTo function accepts three values (x, y, t), which will be called: **initial value**, **target value** and **step**. From each trio of values results a single value called simply the **final value**. The final value resulting from this function is the initial value changed at most by step in such that it is as close as possible to the target value.

In order to to calculate the value of new emotion resulting from the end of the action, let's define another function we'll call Adjust. The result of it is the new value of a single emotion after the action is completed:

$$Adjust : [0,1] \times [0,1] \times \mathbb{N} \to [0,1], \tag{13}$$

and let's describe it with the following formula:

$$Adjust(x, y, j) := pr_j(agent_{state}) + MoveTo(x, y, pr_k(W)) \tag{14}$$

Adjust therefore accepts a set of three values: (x, y, j), which are called: **initial emotion**, **target emotion** and **emotional index**. The value resulting from this function is called **final emotion**. The Adjust function can be described as follows: If the index of current emotion is in the vector of coordinates I, then the result of the function is equal to the initial emotion shifted towards the desired result emotion value of appropriate index. This translation though is scaled by the weight value w_i corresponding to the given emotion index. If the emotion index does not appear in I the result is simply equal to the initial emotion and Adjust is simplified to:

$$Adjust(x, y, j) := pr_j(agent_{state}) \tag{15}$$

The function defined in this way allows us to describe, in a simple way, the values of subsequent components of the emotional state of the agent $\alpha_0, ..., \alpha_n$:

$$\forall_{0 \le k \le n} \quad \alpha_k = Adjust(pr_k(agent_{state}),\ pr_k(rule_{result}),\ k), \tag{16}$$

and this, in turn allows as to finally define the resulting agent emotional state as:

$$agentstate := (\alpha_0, ..., \alpha_n). \tag{17}$$

5.3. Participant impact phase

Each participant profile $rule_{participant}$ defined in behavior rule defines a desired, perfectly fitting opinion $participant_{profile} := ES$ and, just as in the action impact phase, it distinguishes a set of effects weights on individual emotions $w_1, ..., w_k$ located, respectively, on the coordinates $i_1, ..., i_k$ of the vector $member_{profile}$. Each of the participants selected also has a corresponding vertex in the memory graph containing the agent's opinion about the participant $v_{opinion} := ES$.

With these assumptions we can use the formula (14), which will allow to describe subsequent elements of the agent emotional state $alpha_0, ..., alpha_n$ after a particular participant is taken into account:

$$\forall_{0 \le k \le n} \qquad \alpha_k = Adjust(pr_k(agent_{state}), pr_k(v_{opinion}), k). \tag{18}$$

It is worth noting that the the result is not dictated by the participant profile $participant_{profile}$, but by the opinion about actual participant found: $v_{opinion}$.

The resulting emotional state of the agent can be then simply defined as:

$$agent_{state} := (\alpha_0, ..., \alpha_n). \tag{19}$$

All of the participants are analyzed one by one, in a specific order, each of them changing the emotional state of the agent in a similar way.

6. Memory modification

The system created has to create new memories and connect them with relations, that in time should evolve. In this section the process memory graph creation is described.

6.1. Memory importance

At this stage, all the initial requirements needed for the actor to perform the rule chosen are met. However, before this happens, an additional step is performed.

The goal of this step is to make easier future searching of the graph. In relation to the ant colony optimization algorithm this step corresponds to the process of ants leaving the pheromones after their return to the colony and also their gradual evaporation.

For each participant of the action, a path through the memory graph that led to it is remembered. The weight of each of the edges along the path is increased by a certain fixed value of the W_{plus} multiplied by the $rule_{importance}$ of the performed action.

In order for an agent to be able to forget, The total weight of all edges in the memory graph is limited to 1. If, after increasing the weight of the edges recently used, this sum exceeds 1, it is necessary to reduce the weights of graph edges accordingly. A similar situation can also occur when a new association (edge) between two memories is added. The act of lowering weights of the edges may lead to a situation where a weight of an edge will become smaller than the W_{min}. Any edge below that value is then removed from the graph. If that happens, both vertices that were connected by the edge removed are checked for whether they have any other edges associated with them. If not, the bot is unable to associate the memory with anything else it remembers: this vertex is thus removed from memory and completely forgotten. This mechanism simplifies the graph of memories and imitates the process of gradually forgetting. The steps to normalize the sum of weights are as follows:

1. Calculate the sum of weights of all edges of the memory graph W_{sum}.
2. For each edge v_i divide its weight by the sum of all weights: $w_i = w_i/W_{sum}$.
3. Delete each edge v_i for which its w_i is smaller than defined in expression (2) W_{sum}.
4. For each vertex that is the end of the deleted edge, check if it has any other edges. If not, remove this vertex from the graph.

The minimum value of the W_{min} association weight defined in the expression (2) and the fact that the sum of all association weights does not exceed the number 1 limit the maximum number of edges that can exist in the agent's memory graph at once. The maximum number of associations in the graph is $\frac{1}{W_{min}}$. By modifying the W_{min} and W_{plus} values as well as the initial weight of the newly created edges, it is easily to control how much information the agent can store at the one time.

Actions described in this section promote the selection of edges that most often lead to a successful search for a participant. An additional effect that resulted from this mechanism is the creation of preferences in bots: two objects of the game world may be identical, but because one of them is connected to the edge with a high $edge_{importance}$ value, it will be chosen more often than the other one. In addition, the introduction of the process of removing unused edges and memories simplifies the graph and creates an easy to adjust limit on the amount of agent memories.

The memory graph of each agent is constantly undergoing changes during as a result of the actions a he performs. One of the basic mechanisms contributing to the creation of new vertices and edges is the perception. In addition, the agent also memorizes any actions taken and other kinds sources. Below are possible situations in which the graph is changed:

- A new location is noticed by the agent. For each newly discovered site, a vertex will be added to the graph, and then it will be associated by new edges with any already known neighboring sites.

- If an agent gets close to an object, he will notice it and then create a new vertex. It will be connected by an edge to the memory corresponding to location where the object is currently located.

- Agent noticing another bot performing an action related to the a certain rule will change its opinion of that agent $v_{opinion}$ slightly towards $rule_{perception}$ of that rule.

- Each rule rule completed by a bot will mean two types of changes in the graph. The first one of these is that the opinions about the participants will be changed so that they'll resemble more closely opinions about appropriate participant profiles desired in the action. The second of the changes is that a new vertex of the graph is created: it represents the event itself and is connected to vertices representing every other participant of the action performed. The vertex created is also connected with the current location of the agent.

- There may also be situations in which the bot's memory will be modified in a more abstract way. This can happen, for example, as a result of reading something on a piece of paper or talking with another agent about something. In these situations, the agent memory graph G is extended by a different $G2$ graph. $G1 = G2 \cup G0$.

6.2. Evolving opinion

As described, each object remembered by the agent has a certain $v_{opinion} := ES$ value which represents his opinion about the object. After the completion of an action with a participant, the opinion of the agent about an appropriate memory will change depending on the result of the action. As a result, the value of $v_{opinion}$ will get closer to the $rule_{result}$ of the rule performed.

The opinion changing process itself is to an extent based on performing operations similar to those performed when modifying the emotional state of an agent after performing an action. To describe the modification of the opinion, we will use described in the expression (14) $Adjust$ function. With its use, we can describe subsequent values of the opinion vector $0, ..., n$ after the modification:

$$\forall_{0 \leq k \leq n} \qquad \alpha_k = \text{Adjust}(pr_k(agent_{state}), pr_k(v_{opinion}), k). \tag{20}$$

For simplicity, it is assumed that when using the function (12) for calculating adjusted opinion, weights of effects $W = (w_{i_0}, ..., w_{i_n})$ are divided by ten.

With such assumptions we can define a new opinion about the object used in an action as follows:

$$v_{opinion} := (\alpha_0, ..., \alpha_n). \tag{21}$$

7. Simulation

In order to properly verify the model, a dedicated simulation environment has been created. Its individual parts were then analyzed for the occurrence of specific mechanisms and measured. There were also specific requirements that needed to be fulfilled:

- the simulation has to define a set of behavior rules designed to verify selected elements of the model,

- the simulation must contain all of the necessary objects to perform every action,
- the simulation must also allow multiple agents to interact simultaneously and to interact with each other.

The finally selected real environment, that ended up being reproduced by the simulation, is the accounting and sales office. This choice is dictated by a number of justifications. First and foremost, the office environment fills the requirements described above nicely. It provides a wide set of behaviour rules related to everyday tasks, variety of objects in the office as well as the presence of many agents who work at the office and interact with each other.

This kind of environment makes it possible to measure how much work each of the agents performs. By making a change in simulation and observing any changes in their productivity, we were able to tell to a greater extent what kind of influence it had on agents. This allowed us to draw more conclusions about any of the aspects of model being tested. There are 3 kinds of work-related measures we used: current productivity of an agent, his willingness to perform productive activities, as well as his willingness to perform activities unrelated to work.

Thanks to offices being the places where employees spend several hours a day, they are, to a cartain extent, isolated from outside environment. This makes longer simulations retain some reflection in reality.

7.1. Layout

The designed office consists of several rooms designed to serve specific purposes. A diagram displaying office layout is presented on the figure 2.

Basing on the letter marks from fig 2, we can describe all parts of the office:
A – Workspace: The place where the agent workstations are located. It also contains some other objects such as a water machine and a copy machine.
B – Kitchen: contains the equipment needed to prepare a meal, a couple of seats and a coffee machine.
C – Facilities: a room that contains a vending machine and is connected to the kitchen.
D – Manager's room: Room for a special agent with which several additional mechanics are associated.
E – Meeting room: A room filled with chairs and containing a whiteboard. Meetings can be organized here (by the office manager).

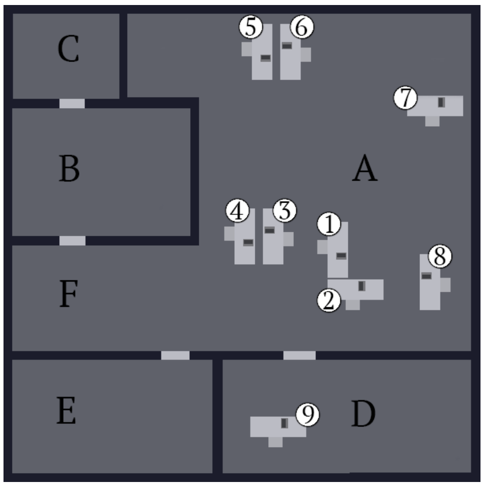

Figure 2: A general layout of the office. Letters A-F mark important parts of the office. Desks belonging to agents 1-9 are numbered.

F – Hall: a small corridor connecting the work space, kitchen and meeting room. It also contains a second copy machine.

The office manager has been marked with number 9. His desk is located in a separate room.

Each time the simulation begins, agents do not have any memories or opinions – their memory is created entirely during the simulation.

7.2. Available rules

In total, twenty-one behavior rules are present in the simulation. Each of them has its own unique configuration. Four of these rules are related to the exploration of the environment. Another 11 rules are related to the activities done in the office. The next five rules require some kind of interaction between agents. One of those is available only to the office manager and related to his position (the rule called "GiveAScarySpeech"). Two of the five interaction rules are related to the interaction of the employee with the supervisor.

7.3. Observed bahaviours

Before we go on to the detailed description of the tests, it is worth to describe a few fragments of simulations and them.

Printers: There are two copy machines that are annoying to use. They are used in the "UsePrinter" rule. The emotional result of this rule is a feeling of boredom and anger. One of the printers is in a frequently visited F location. This location itself is not often used in rules, but as a result of being visited frequently when searching the memory, it usually is connected with edges with high values of weight. This fact in turn means that the printer in location F much more likely to be used by agents than the other printer, located at the back of the office.

Two agents owning their desks at the back of the office, close to the second copy machine, are an exception to this rule. They use both printers interchangeably.

As a result of an agent noticing an object, his emotional state changes slightly (it gets closer to the opinion about the object). This means that agents who are walking through the F location are often irritated at just the sight of the printer.

A similar situation took place in the case of the manager. Occasionally, he performs a "GiveAScarySpeech" rule. While he is doing that, other agents can attend to the meeting room and listen to the speech. The emotional result of hearing the speech is mainly related to fear. This however means that over time, the opinion employees about the manager may become more and more fearful. This in turn causes the manager to be cease being a participant in certain actions performed by other agents (like arguing). Even in cases where the manager himself decides to perform a positive "TellJoke" action, the receiving agent can become even more scared than amused after hearing the joke.

Each agent owns a telephone located on his desk. The phone can occasionally start ringing and the agent then has to pick it up in time and talk for a moment while responding to a call. However, there is a risk that the agent will not be able to answer the phone in time and the action of responding to a phone call will fail. During the tests, it turned out that most often this happened to agents who had their desks located the farthest from other popular places in the office (like the kitchen). This was the most visible in the case of two bots owning desks at the back of the office, not far from the previously mentioned, unpopular printer. This in turn made them on average to be in a worse mood.

The table in the kitchen with several chairs was used both for the "GrabASnack" and "GrabAMeal" actions, and each of its chairs can be reserved separately by the agents. These two actions are a good example of how agents can temporarily

Table 1: Examples of emotional states and the corresponding willingness to work and to take a break

ES				WorkMood	BreakMood
0.5	0.5	0.5	0.5	0.3988	0.1614
0.0	0.0	0.0	0.0	0.1908	0.1614
1.0	1.0	1.0	1.0	0.2209	0.3151
0.6	0.5	0.5	0.7	0.4198	0.2297
0.6	0.5	0.4	0.2	0.4070	0.0964
0.2	0.5	0.5	0.5	0.3614	0.1453

reserve some objects to perform an action, so many agents can perform the same (or similar) action at the same time.

8. Tests

8.1. Emotion tests

Both the verification of the emotional state and of other aspects of the model were based on measurements of the aforementioned productivity of the bots. Each rule available in the simulation has been assigned two numerical values, which in turn tell us about how much a given rule is related to work and how closely it is related to avoiding work. On the basis of these, we can then define 3 measures associated with it: the amount of work currently being performed, the willingness of the agent to work and the willingness to take a break from work. They are defined as follows:

Productivity is defined as the average value of work done in recent actions. Each successive value of productivity is defined as the previous productivity halved and increased by the work value associated with the action just performed.

Willingness to work is obtained by calculating the average score of actions multiplied by their respective degrees of work relation.

The willingness to take a break is obtained by calculating the average score of actions multiplied by heir degree of relation to avoiding work.

In the Table 1 there are several example emotional states and the corresponding willingness to work / take a break

The smallest willingness to work was achieved for the most extreme *ES* values. For $ES = (0.6, 0.5, 0.6, 0.7)$ the mood for work was the greatest (because the agent was in a generally good mood), but the result of willingness to take a break was also big (because willingness to take a break largely depends on the awaitance of the agent). For $ES = (0.6, 0.5, 0.4, 0.5)$ the desire to break turned out to be the smallest.

The chart 3 shows the productivity of an agent along with the subsequent activities he performs. This relationship was presented in conjunction with his for awaitance and anger emotions, which had a big impact on the willingness to work and rest of the agent and thus on his productivity.

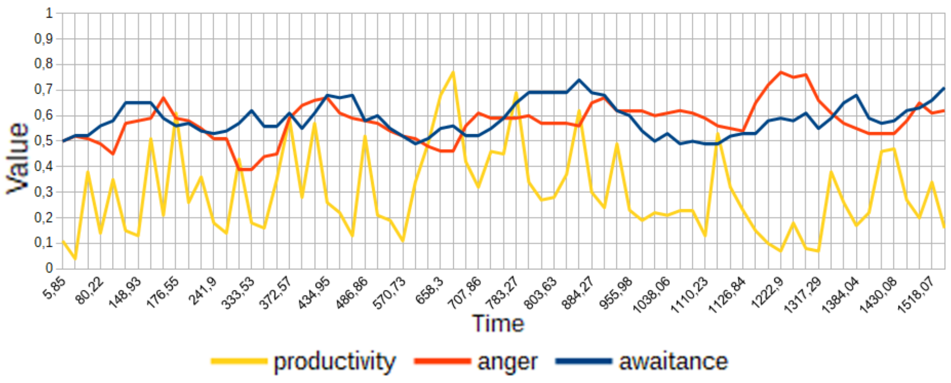

Figure 3: Agent's productivity depending on his value of awaitance and anger emotions.

8.2. Memory tests

Let's begin with the amount of memories created during the simulation and total agent memory capacity.

The minimum value of the edge weight W_{min} used in the simulation is equal to 0.001. Bearing in mind that the maximum combined weight of all edges in the graph should not exceed 1, the graph is able to store a maximum of $\frac{1}{W_{min}} = 1000$ associations. In practice however, the importance of associations are constantly changing so this is very unlikely. The 4 figure shows how the average number of memories and associations of all bots evolves during the simulation.

As visible in the figure 4, the total number of associations does not get even close to the maximum number of associations which is equal to 1000. It is also

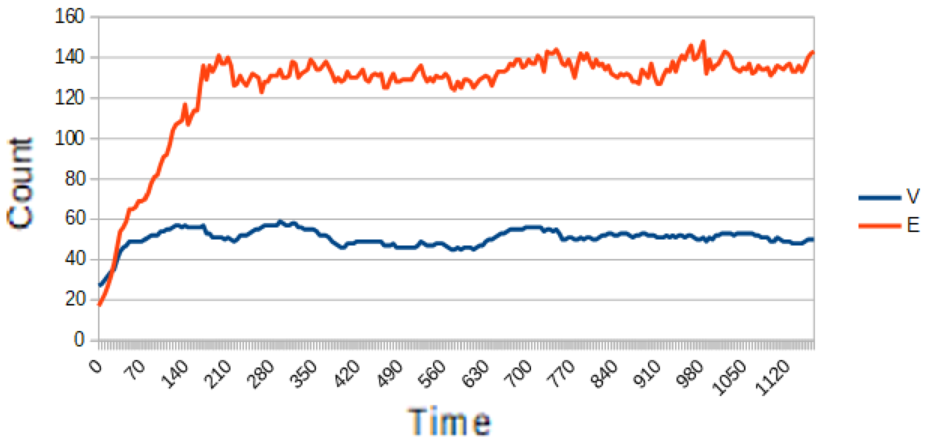

Figure 4: Average number of memories and associations among the agents during the simulation.

clearly visible that the graph's growth starts to stabilize at a certain point. This gives us a good perspective on the amount of computer memory the agent requires to operate. Based on that we can draw some conclusions on how to adjust the agent's memory based on how much we want him to remember at once. By reducing W_{min}, we are able to easily increase the memory capacity of agents. In addition, two more factors influence how the memory is constructed. These are the initial importance of memory association and the base value of associacion importance increase when an edge is used.

The figure 5 contains the plot of evolution of agents' average opinion of office manager.

As can be seen in the chart, at some point in the simulation, the employee's opinion has changed significantly. This change resulted from the manager has performed "OrganizeAScaryMeeting" action, which has significantly changed the employee's opinion about his supervisor. For the rest of the simulation, this opinion was slowly returning to normal, because the manager did not perform more unliked activities and behaved otherwise normally.

Due to the nature of the algorithm, the computing cost associated with finding the action participant has remained at approximately the same level. However, there in some cases, a rare situation occurred when a participant was not found.

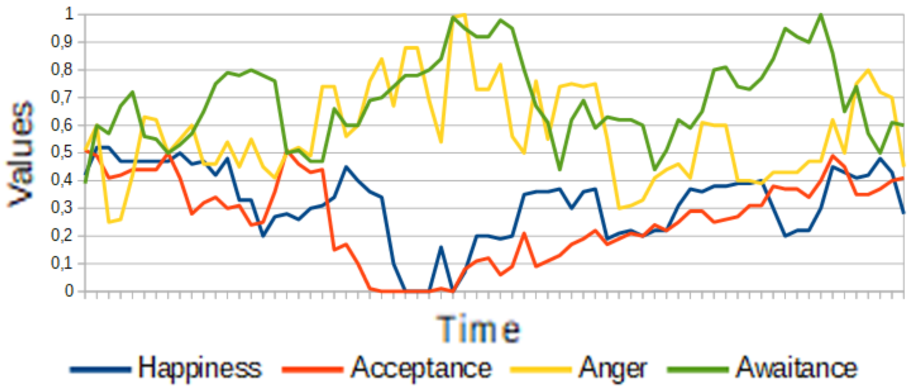

Figure 5: Evolution average employee opinion of the office manager as the simulation progresses.

The rarity of this occurrence may have been partly due to the small scale of the simulation and it is necessary to conduct larger-scale tests in the future to test whether the algorithm will remain efficient.

9. Conclusions

The following activities have been recognized as the main goals of the work:

- Proposing a model of virtual agents that simulate memory and personality.
- Implementation of the proposed model in a selected game engine.
- Creating an efficient way of choosing actions to perform.
- Implementation of a simulation in which the model described in this paper is used, further examination and categorization of agent behavior resulting from the nature of the designed system.
- The ability to customize the implementation for varying kinds of games.

These goals have been to a large extent achieved. A goal that has not been fully achieved though is the proper verification of the model. Although the most important elements of it have been presented in form of examples, no large-scale have been carried out to statistically prove its operation.

Above described results show clear potential of the system in form of many lifelike behaviour mechanisms that come with the addition of more complex memory to emotion based agent systems. An interesting direction of exploration may be to extend the model by integrating an agent personality system.

References

[1] Ochs, M., Sabouret, N., and Corruble, V., *Simulation of the Dynamics of Non-player Characters' Emotions and Social Relations in Games*, IEEE Transactions on Computational Intelligence and AI in Games, Vol. 1, No. 4, dec 2009, pp. 281–297.

[2] El-Nasr, M. S. and Skubic, M., *A fuzzy emotional agent for decision-making in a mobile robot*, In: 1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228), IEEE, 1998.

[3] Daszuta, M., Szajerman, D., and Napieralski, P., *Emotional Intelligence in Mobile Games*, PTI, Polish Information Processing Society, 2017.

[4] Wróbel, F., Daszuta, M., Szajerman, D., and Wojciechowski, A., *Adaptation of WASABI emotion engine for use in video games*, Lodz University of Technology Monograph, 2017.

[5] Johansson, A. and Dell'Acqua, P., *Emotional behavior trees*, In: 2012 IEEE Conference on Computational Intelligence and Games (CIG), IEEE, sep 2012.

[6] Liu, Z., *An autonomous emotion model for virtual human*, In: 2008 IEEE International Conference on Automation and Logistics, IEEE, sep 2008.

[7] Wang, W., Tan, A.-H., and Teow, L.-N., *Semantic Memory Modeling and Memory Interaction in Learning Agents*, IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 47, No. 11, nov 2017, pp. 2882–2895.

[8] Mahmoud, I. M., Li, L., Wloka, D., and Ali, M. Z., *Believable NPCs in serious games: HTN planning approach based on visual perception*, In: 2014

IEEE Conference on Computational Intelligence and Games, IEEE, aug 2014.

[9] Arnold, M. B. and Plutchik, R., *The Emotions: Facts, Theories and a New Model*, The American Journal of Psychology, Vol. 77, No. 3, sep 1964, pp. 518.

[10] Bonabeau, E., *Swarm Intelligence*, OUP USA, 1999.