Project Emohawk – programmer documentation

Table of Contents

1 Introduction

This document is a programmer documentation for project Emohawk. The purpose is to provide the developer an overview of the project, which will enable him to be better oriented in project JavaDoc (the main part of project programmer documentation). Some of the text here is based on the text present in the diploma thesis. The images here are present also in the diploma thesis.

Project Emohawk is a Java implementation of a simple story occurring in the 3D virtual environment of the action game Unreal Tournament 2004 (UT04). The story features four intelligent virtual agents (IVAs) with emotions (Fig. 1). Project Emohawk used platform Pogamut 3 as a base tool for connecting to UT04 and programming the IVAs decision making system. Emotion model ALMA was used to provide IVAs with emotions. The agents are able to interact with each other through actions and communication. Everything is handled symbolically with text messages.

Fig. 1. Two Emohawk IVAs interacting. On the figure we can two Emohawk IVAs interacting in the

environment of the game UT04 with text messages. Copyright Epic Games 2004-2009.

The implementation of project Emohawk can be divided into two parts. A minor part of implementation took place in UT04 – we have used native UT04 scripting language UrealScript to add and modify some of the game content. A major part took place in Java. Here we have programmed the IVAs, connected the emotion model ALMA and programmed the methods for Emohawk story analysis. From now on, we will be concerned with Java part of Emohawk implementation. The details about UT04 part will be provided in a chapter with the same name at the end of this document. This documentation is organized as follows: First we will describe the ways of IVAs interactions, then we will briefly mention IVAs memory, afterwards we will concern IVAs control architecture, next we will dig into IVAs environmental perception followed by a list of all emotion events in the project. We will end this document with brief packages and classes overview and brief overview of UT04 implementation.

2 IVAs interactions

Emohawk IVAs can communicate between each other through actions, proposals (special kinds of actions) and casual conversation. These three types of communications are all handled through text messages. For this purpose we have defined a simple keywords based protocol to parse the messages.

An example of interaction between agents may look like this:

Bruno: "To:Anne, making proposal kiss".

Anne: "To:Bruno, proposal kiss accepted".

Bruno: "To:Anne, ACTION KISS"

First Bruno made a proposal to kiss Anne, Anne accepted the proposal. After that Bruno performed action kiss toward Anne.

Keywords. The "To:" keyword followed by a set of names separated by commas determines the agents this message is for. If the "To:" keyword is missing the message is considered to be for everyone. If there is "To:" and no names are after, the message will not be parsed by anyone. Note that the agents need to see the one who is speaking to them in order to receive the message.

Now to determine if the message is an action, a proposal or simple text message another set of keywords is used. This set includes:

- "proposal" – followed by one of the proposal types. Marks that this message is a proposal of input proposal type. To further determine if the agent is responding to some proposal or is making a new proposal, one of following three keywords is added: "making", "accepted" or "rejected"
- "ACTION" – followed by one of action types. Marks that this message is an action of input action type.

If the message does not include any action or proposal keyword, it is treated as a casual conversation. For casual conversation agents parse a set of smilies to evaluate the conversation emotional value. Following smilies are parsed in the text messages:

- ":-)" and ":-(" – substitutes emotions joy and distress. More positive ":-)" smilies mean the emotion joy will be generated upon receiving this message. The intensity depends on the number of smilies. The highest joy intensity will occur if the result is that in the message there is five more positive smilies than negative ones.
- ":-*" and ">:@" – substitutes emotions liking and disliking. More positive ":-*" smilies mean the emotion liking will be generated upon receiving this message. The intensity depends on the number of smilies. The highest liking intensity will occur if the result is that in the message

there is five more positive smilies than negative ones.

- "#!" and ":-O" – substitutes emotions anger and fear. More "#!" smilies mean the emotion anger will be generated upon receiving this message. The intensity depends on the number of smilies. The highest anger intensity will occur if the result is that in the message there is five more "#!" smilies than ":-O" smilies.

Our agents are generating the smilies in the text messages according the actual emotions joy and distress toward the target agent (smilies ":-)" and ":-("), according the actual feeling value (smilies ":-*" and ">:@") and according the emotions anger and fear (smilies "#!" and ":-O").

3 IVAs memory

Our agent remembers interaction between him and other agents. He remembers last times of actions, proposals and communication he did toward target agent and last times of actions, proposals and communication other agent did toward our agent. Also our agent remembers last time he have seen certain player, the time when he lost sight of him and last times when he/she was at cinema, at park or at home with target agent. This information is used to generate emotion events properly and to pick actions and proposals toward another agents properly.

For storage of these information classes PlayerInfo and ConversationInfo are used. The lists of these objects are defined in EmotionalBot class.

4 IVAs architecture

Project Emohawk IVAs are controlled by finite state machine. They feature reactive behavior. In Pogamut agents reasoning is based on doLogic method, that is called periodically (usually each 250 ms – depends on settings). To get information about surrounding environment the agent can use either Pogamut modules (providing information like a list of all currently visible other agents) or Pogamut listeners, which are invoked each type certain event occurs in the environment (e.g. someone sent text message).

The overview of Emohawk IVA architecture can be seen on Fig. 2. The information from the environment is got through Pogamut sensory modules and Pogamut listeners. This information is processed by agent state logic (in classes EmotionalMaleBot for male IVAs and EmotionalFemaleBot for female IVAs) and may cause the state transitions. The text messages are processed by EventGenerator class, that provides more complex percepts for our agent (or more complex events), which is needed to provide emotion model ALMA with correct inputs. The information from EventGenerator are a) stored in agents memory (if they contain some information about other agents) and b) processed by other class (AEventGenerator), that is responsible for appraising the events with ALMA variables and providing the ALMA with input.

The agent decision making is then affected by a) Pogamut sensory information, b) agents history (history of interaction with the agent we are currently interacting with), c) ALMA affects – we count a feeling attitude (positive or negative) toward other agents based on ALMA affects and d) agents current state he is in.

Moreover in EmotionalBot class, there are many constants define that affects decision making and action generation – namely minimal delays between actions or behavior, feeling intensity needed to trigger certain actions etc. Some constants are as well in EventGenerator class.

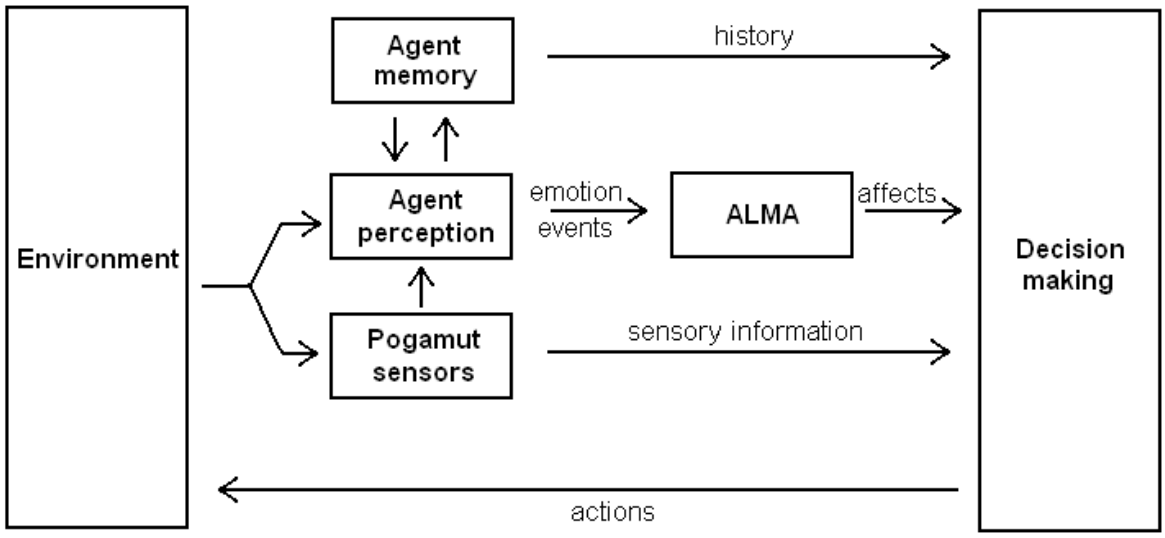

Fig. 2. Project Emohawk IVAs architecture. The information from environment are processed by a) Pogamut sensors providing our agent with basic sensory information and by b) agent perception module, that generates input for ALMA emotion model and stores relevant data into agent memory. The decision making is then affected by sensory information, agent history and agent affects got from ALMA.

Now we will provide more detailed overview of the agent's states – what do they monitor, what actions and proposals they can trigger etc.

4.1 Agent states

In our scenario we have two types of characters – boy and girls. For controlling these characters we use basically the same states with slightly different implementations for a boy and a girl. We will list the states of our agents here describing the behavior they produce and the mechanisms our agents use.

There are two important variables, that are used in our states. It is “agent with” and interrupters list. “Agent with” is some agent of opposite sex, that we generally like. To this agent we can make intimate proposals, with him we can go to cinema and to home. We can imagine this as someone of opposite sex, we are interested in. The interrupters on the other hand are the other agents we are not interested in, but who require our attention. For example they have approached us and speak to us, or they interrupted the agent we have currently selected as “agent with”. The interrupters can be agents of same sex, agent of opposite sex we do not like and animals.

The possible state to state transitions can be seen on Fig. 3.

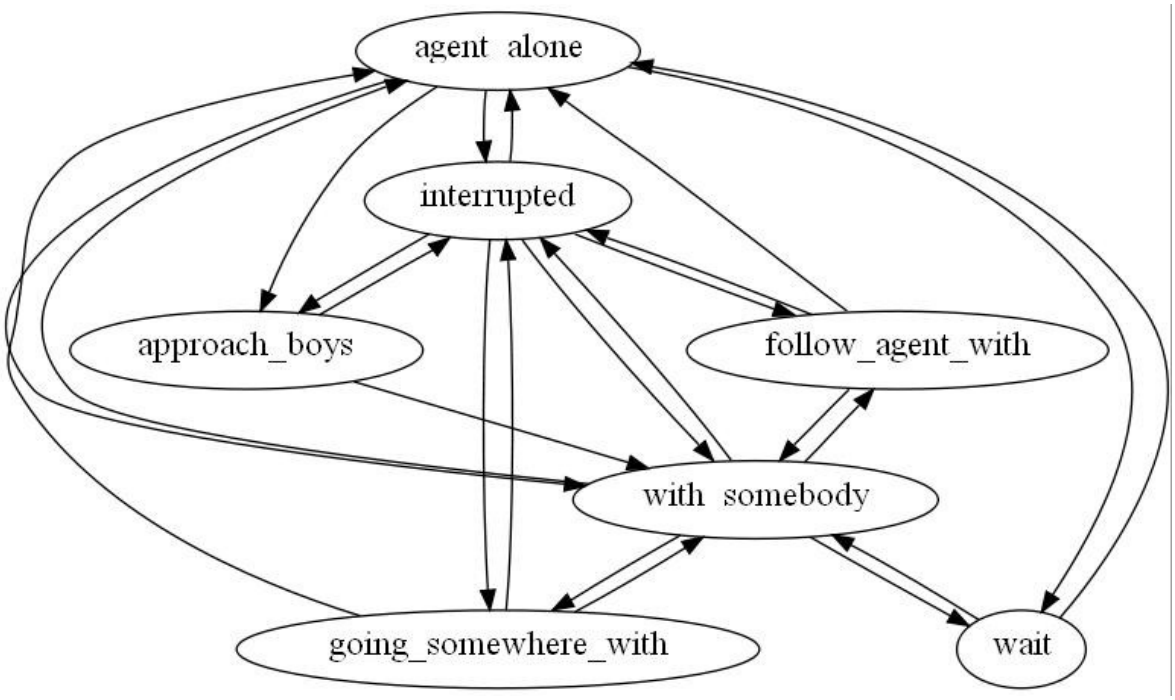

Fig. 3. Project Emohawk IVAs FSM transitions. On the figure we can see possible transitions between states for project Emohawk IVAs. State "approach boys" is reachable only for girls. Bruno and Anne begin the scenario in state "with somebody", Clementine in state "agent alone" (later she will go to near cinema location and switch to "wait" state waiting for Bruno). Emohawk is scripted and not controlled by states. Image created by GraphViz 2.0.

4.1.1 state agent alone

This state handles a situation when the agent is alone in the environment. Implementation for a boy and a girl differs. The boy will start exploring the environment with a goal of meeting someone, while the girl just goes to her home.

Both of the genders first check if there are any interrupters – if yes, they will shift to interrupted state. The girl in this state goes to home straightforward and if she reaches the home place she will start to wait until the timer expires.

Except the interruption check, boy is also checking if he sees some agent, he wants to start interaction with. This can be only girls he has feeling attitude above zero. If he sees the girl, he will start to walk to her. For this path finding will be used. When he reaches the interaction range, he shifts to agent with state and selects the girl as his "agent with".

4.1.2 state follow agent with

This state is active when our agent is leaded by "agent with" somewhere. A typical situation – "agent with" proposed our agent to go to the cinema together. Our agent accepted and switched to this state. Now "agent with" is leading the way to the cinema with our agent following him.

Our agent is simply following the "agent with", but he knows where they both are going – so he can check if they are already there. If yes, the state will be switched back to "with somebody" state. Also in this state again the condition if there are any interrupters is checked – if yes, we will switch to state interrupted.

4.1.3 state going somewhere with

This state is used when we are leading "agent with" somewhere. This time it was our agent who

proposed to “agent with” to go somewhere. When “agent with” accepted our proposal, we have taken the initiative and switched to this state, which will make us to go to desired location with the “agent with”. We will wait for “agent with” to catch up – if the distance between us will be above certain threshold, we will stop and turn toward the agent that should follow us. If the distance decreases we will continue to go to the goal place.

Again the interruption routine is at the beginning of this state. If someone speaks to us, we will switch to state interrupted.

4.1.4 state interrupted

This states implementation is the same for a boy and a girl. It is used when we are already with “agent with” or when we are doing some other activity in the environment and someone approaches us and speak to us. We will switch to this state, which handles the communication with multiple agents.

In this state we will first check if someone is selected as our “agent with”. If no, we will process the interrupters list if someone would be suitable to be set as our agent with. It can happen that no one is. In this case we will simply have our “agent With” set to null.

Secondly, we will check if all the interrupters are in our communication range – if some of them are not, we will perform action bye toward them and remove them from the list. After this check we will pick focus – this means the agent we will turn to and perform action and/or conversation to. Focus is picked according to the current feeling attitude and on the fact if other agent performed some action and/or conversation toward us, which we have not reacted to yet.

Thirdly we will a) check if our recent proposal we could make was accepted and b) respond to proposals someone has made to us (this is actually not synchronized with our focus). If no proposals occurred we will pick some action toward our focus (but we do not have to). After that if we are still in interrupted state we will send some conversation message to the agent – the conversation method will be invoked (if the time is right we will communicate with the agent).

We will immediately leave this state and switch to previous state if a) interruption timer has run out or b) the list of interrupters is empty.

4.1.5 state wait

This state is active when we are waiting for someone. We will leave this state when the agent we are waiting for returns or when the waiting timer runs out. The agent we are waiting for is marked as our current “agent with” – even when we do not see him (this is an exception to the rule that we will reset our “agent with” when we lost sight of him). If there are some interrupters we will handle them in this state except of switching to interrupted state. We are not moving when we are at this state. Our agent can only turn to some interrupter.

4.1.6 state with somebody

This is the main state handling “agent with” interaction. In this state we pick proposals to the other agent. Which proposal will be picked depends on where we currently are (for example the sex proposal can be made just at home). Also we look on our proposal history. We will not propose to go to the cinema if we were recently there with the agent.

First we will check here if any interrupters are present – if yes, we switch to interrupted state. Second we will check if our recent proposal was accepted and change our state accordingly or if “agent with” made proposal to us – if yes, we will respond. Then we will try to pick some proposal to “agent with” – just if the time is right. If no proposal occurred, we will try to pick some

action. After that we can make a conversation with the other agent by sending text message containing causal conversation.

The proposals that we can make in this state and that can affect our behavior are (note that girl is limited just to make a kiss proposal):

- Cinema – character one will propose character two to go to the cinema with him. If this proposal is accepted, the character one will lead the way to cinema followed by character two. Reaching the cinema location, the characters will start to watch the film – this lasts for 15 seconds, where the characters will not speak interact with anyone. After this the characters resume the interaction between them.
- Park – character one will propose character two to go to the park with him. If this proposal is accepted, the character one will lead the way to park followed by character two. Nothing special happens when they reach the park place.
- Home – character one will propose character two to go to home of character two. If this proposal is accepted, the character one will lead the way to character two home followed by character two. Nothing special happens when they reach the home place. However the home place allows for extended set of proposals to be selected.
- Kiss – character one asks character two if he/she can kiss him/her. When accepted character one will perform action kiss toward character two. This proposal can be made at cinema or at home.
- Sex – character one asks character two if he/she wants to make love with him/her (possible just between opposite gender). When accepted character one and two will make love with each other. This is visualized by text messages in the environment. This proposal can be made just if our agent is at home of the other agent.

4.1.7 state approach boys

This state is used just by the girls. Since normally the girls just go home when they are alone, we have added this state that can be triggered after some time the girl is waiting at home. The girl will explore the environment when in this state searching for opposite gender characters approaching them and starting the interaction. Also the interruption routine is present in this state (switching to interrupted state when there are some interrupters present).

5 IVAs environmental perception

Here we will consider the mechanism of agent perception of the environment as well as technique we use for mapping environment events to ALMA inputs.

The agent perception is based on Pogamut sensory methods and Pogamut listeners. Sensory methods allow are agent to ask for example: “Do I see any player right now? And if yes, who is it?” The listeners allow our agents to listen to certain message types, that brings information about certain environmental events. For example we can “listen” to all of the text messages our agent receives.

These methods and listeners provides us with basic information about environment – what do we see or what we recently heard. However, since we wanted our agents to be able to react to various types of situations emotionally, we needed to built upon these basic methods to provide our agent with more complex percepts. For example we are not interested just if we see someone, but we would like to know how long we are seeing him, when was the last time we saw him, if we actually like him and so on. We solved this by post-processing some of the environment events in EventGenerator class. This provided our agents with more complex percepts. Using these our agents are able among other things to recognize pre-defined set of events affecting their emotion state. These events are appraised by a set of ALMA emotion variables (in AEventGenerator class)

and processed by ALMA. This can result in a change of agents emotions and mood. In Chapter below we will list a set of all emotional events we are able to generate as well with emotions they trigger.

Emotion model ALMA is configured by two XML files (AffectComputation.aml and CharacterDefinition.aml). We have used slightly modified default XML configuration files to configure project Emohawk IVAs. Especially we have modified CharacterDefinition.aml so emotion computation and decaying is the same for all the three characters.

6 List of all emotion events in the scenario

List of all generated emotional events will follow. The name of the events is taken from the name of methods that generate the events. One event can generate more emotions. Every emotion is associated with some elicitor or eliciting condition – this will usually be the agent who have caused the event. For simplicity there will be three categories of emotion intensities low (0.01 – 0.3), medium (0.3 – 0.6) and high (0.6 to 1). To elicit the events we have used three ALMA EEC variables – desirability, praiseworthiness and appealingness.

The names of the methods here follow certain logic. If the keyword Action is present it means the event is result of some action. If the proposal keyword is present, it means that the event is associated with proposals somehow etc. “By” keyword means that the cause of the action was not our agent (someone else has done the action). Keyword “Other” means that our agent was not involved at all in the event.

6.1 generateActionKissEvent

Result of Action KISS – we have kissed the agent.

Emotions generated depends on the feeling toward other agent.

Emotions:

- joy and love – high intensities – associated with other agent
- distress and hate – high intensities – associated with other agent

6.2 generateActionKissOtherEvent

Result of Action KISS – someone has kissed some other agent in the scenario (our agent wasn't involved). Emotions are dependent on the feeling – if the feeling is positive we will generate emotions towards kisser.

Emotions:

- distress and disliking – from medium to high intensities (higher when the feeling is bigger) – associated with the kisser

6.3 generateActionKissByEvent

Result of Action KISS. Someone has kissed us. Emotions depend on the feeling toward the agent – the higher the feeling, the bigger positive emotions and vice versa.

Emotions

- joy and love – low to high intensities – toward the other agent
- distress and hate – low to high intensities – toward the other agent

6.4 generateActionSlapByEvent

Result of Action SLAP – we have been slapped by some other agent. Emotions are not dependent on the feeling towards agent.

Emotions:

- distress and disliking and anger – high intensities – associated with other agent

6.5 generateActionSlapOtherEvent

Result of Action SLAP. Someone has slapped some other agent. We will generate emotions if the feeling toward the agent slapped exceeds a certain threshold.

Emotions:

- distress and anger – high intensities – associated with the slapping agent

6.6 generateActionKickOtherEvent

Result of Action KICK. Someone has kicked some other agent. We will generate emotions if the feeling toward the agent kicked exceeds a certain threshold.

Emotions:

- distress and anger – high intensities – associated with the kicking agent

6.7 generateActionKickByEvent

Not implemented yet. Just animals should be kicked, not agents. Called when someone kicks us.

6.8 generateActionByeByEvent

Result of Action BYE. Some other agent has said bye to us. Emotions depends on the feeling toward agent (if positive or negative).

Emotions:

- joy – low intensity – associated with the other agent (if the feeling is negative)
- distress – low intensity – associated with the other agent (if the feeling is positive)

6.9 generateActionByeOtherEvent

Result of Action BYE. Someone has left other agent. Emotions depend on the feelings toward both of the agents.

Emotions:

- joy – low to medium intensity – toward both of the agents – if someone we don't like said bye to someone we like or else.
- distress – low intensity – toward both of the agents – if someone we like said bye to someone we like

6.10 generateActionComplimentByEvent

Result of Action COMPLIMENT. Someone has said compliment to us. Emotions intensities depend on the feeling toward other agent.

Emotions:

- joy and liking – low to medium intensities (higher the feeling, higher the emotions)

6.11 generateActionComplimentOtherEvent

Result of Action COMPLIMENT. Someone has said compliment to other agent. Emotions intensities depend on the feeling toward the agent who said the compliment.

Emotions:

- distress and anger – low intensities – toward the agent who said the compliment

6.12 generateActionCuddleByEvent

Not implemented yet – just animals should be cuddled, not real agents.

6.13 generateActionInsultByEvent

Result of Action INSULT. Other agent has insulted us. Emotions don't depend on the feeling.

Emotions:

- disliking and anger – high intensities – toward the other agent.

6.14 generateActionInsultOtherEvent

Result of Action INSULT. Someone has insulted other agent. Depends on the feeling toward the agent who was insulted.

Emotions:

- joy and pride – medium intensities – toward the agent who made the insult (feeling toward the insulted agent is negative)
- distress and anger – medium intensities – toward the agent who made the insult (feeling toward the insulted agent is positive)

6.15 generateActionSexByEvent

Result of Action SEX. We are making love with other agent or vice versa. This is the result of accepted proposal sex. The emotions depend on the feeling (for positive emotions the feeling should be higher than sexFeelingConst – this should be true, otherwise we wouldn't accept the proposal anyway).

Emotions:

- liking and love – high intensities – toward the other agent – if the feeling exceeds threshold
- disliking and hate – high intensities – toward the other agent – if the feeling does not exceed threshold

6.16 generateActionSexOtherEvent

Result of Action SEX. Someone is making love with some other agent. Emotions depend on the

feeling toward the agent of opposite sex in the couple. The higher the feeling the higher the intensities of emotions. The emotions here are associated toward to both of the agents.

Emotions:

- distress and hate – medium to high intensities – toward both of the agents

6.17 generateActionLeaveByEvent

Result of Action LEAVE. Someone has left us – said bye in a bad way. Emotions depend on the feeling toward the agent.

Emotions:

- joy and gratitude – low intensities – toward the other agent (feeling negative)
- distress and anger – high intensities – toward the other agent (feeling positive)

6.18 generateActionLeaveOtherEvent

Result of Action LEAVE. Someone has left other agent. Emotions depend on the feelings toward both of the agents.

Emotions:

- joy – low to medium intensity – toward both of the agents – if someone we don't like left someone we like or else.
- distress – low intensity – toward both of the agents – if someone we like left someone we like

6.19 generateAloneEvent

Generated every 90 seconds when we are alone (agent doesn't see anyone).

Emotions

- distress – medium intensity – towards no one

6.20 generatePickupEvent

We have picked up some item – can be flower, condom or gun. Emotions do not depend on the type of the item.

Emotions:

- joy – low intensity – towards nothing

6.21 generatePlayerLostEvent

Generated when we have lost sight of other agent (we have stopped seeing him and this state lasts for some time now). Emotions depend on the feeling toward the agent are more intense if the agent is of the opposite sex.

Emotions:

- joy – low to medium intensity – toward the other agent (if the feeling is negative)
- distress – low to medium intensity – toward the other agent (if the feeling is positive)

6.22 generatePlayerTogetherEvent

Generated every 60 seconds we are together with some other agent. Emotions depend on the feeling toward the other agent. Emotions are more intense if the agent is of opposite sex.

Emotions:

- joy – medium intensity – toward the other agent (if the feeling is positive)
- distress – medium intensity – toward the other agent (if the feeling is negative)

6.23 generatePlayerAppearedEvent

This event is generated when we have spotted the agent for the first time or when we haven't seen the agent for some time and we see him again (emotions are lower in the second case). Emotions depend on the feeling toward the agent. Emotions are more intense if the agent is of opposite sex.

Emotions:

- joy – low to high intensity – toward the other agent (if the feeling is positive)
- distress – low to high intensity – toward the other agent (if the feeling is negative)

6.24 generateAngerFearMessageEvent

Someone has sent us message that contains anger and/or fear smilies. We will generate anger or fear emotion toward the agent based on the number of the smilies (more anger smilies = anger and vice versa). The emotion intensity is affected by the number of messages we got from the agent in last 30 seconds. The more messages we got the lower the intensity of the emotion. The intensity of the emotions decrease rapidly.

Emotions:

- fear – low to medium intensity – toward the other agent
- anger – low to medium intensity – toward the other agent

6.25 generateHappySadMessageEvent

Someone has sent us message that contains joy and/or distress smilies. We will generate joy or distress emotion toward the agent based on the number of the smilies (more joy smilies = joy and vice versa). The emotion intensity is affected by the number of messages we got from the agent in last 30 seconds. The more messages we got the lower the intensity of the emotion. The intensity of the emotions decrease rapidly.

Emotions:

- joy – low to medium intensity – toward the other agent
- distress – low to medium intensity – toward the other agent

6.26 generateLikeDislikeMessageEvent

Someone has sent us message that contains like and/or dislike smilies. We will generate like or dislike emotion toward the agent based on the number of the smilies (more like smilies = like and vice versa). The emotion intensity is affected by the number of messages we got from the agent in last 30 seconds. The more messages we got the lower the intensity of the emotion. The intensity of

the emotions decrease rapidly.

Emotions:

- like – low to medium intensity – toward the other agent
- dislike – low to medium intensity – toward the other agent

6.27 generateMessageToOtherEvent

Someone has sent a message to some other agent that contains like and/or dislike smilies. We will generate emotions based on the smilies. We will generate the emotions just in case the feeling toward the agent who said the message is positive. There is big reduction in emotion intensity based on the number of the messages that the agent said to other agent.

Emotions:

- distress and anger – low to medium intensity – toward the speaker (if the message contains more liking smilies)
- joy and gratitude – low to medium intensity – toward the speaker (if the message contains more disliking smilies)

6.28 generateConversationIgnoreByEvent

This event is generated after some time we have spent together with some other agent who is not responding to our conversation. When the conditions are met the event will be generated every 35 seconds.

Emotions:

- anger – low intensity – toward the other agent

6.29 generateInterruptedEvent

We are with some other agent and we are doing something with him – speaking or going somewhere – and we are interrupted by other agent or agents – someone is speaking to us, polymorph approached us or else and we want to react to these other agents (we are not ignoring him, he is close enough etc.). Emotions depend on the feeling toward the agents.

Emotions:

- joy and gratitude – medium intensity – toward interrupting agents (feeling is positive)
- distress and anger – medium intensity – toward interrupting agents (feeling is negative)

6.30 generateReceivedItemEvent

We have received item from someone. We will react based on the feeling toward the giver and on the type of the item. Generally we will be pleased, just if the item is condom, then we will be offended unless the feeling toward the agent is high.

Emotions:

- joy and liking – low to medium intensities – toward the giver (medium for flower or condom and feeling high)
- distress and disliking – medium intensities – toward the giver (condom and feeling low)

6.31 generateItemJelausyEvent

Some agent gave item to other agent. We will generate emotions if we have positive feeling to the giver and the item is a flower – we will be jelaous.

Emotions:

- distress and anger – medium intensity – toward the giver (if the feeling is positive and item is flower)

6.32 generateProposalToOtherEvent

Someone is making proposal to other agent (and none of them is currently with us). We will generate emotions just in case we have positive feelings toward the agent who is making the proposal. The emotion intensities are dependent on the type of proposal – kiss and sex proposals generate high negative emotions while other proposals just low negative emotions.

Emotions:

- distress and disliking – low or high intensity – toward the agent who is proposing something

6.33 generateProposalToOtherByAgentWithEvent

The agent we are currently with is making proposal to other agent. This is the same as above with generateProposalOtherMaking, except the emotions generated are slightly higher.

Emotions:

- distress and disliking – low or high intensity – toward the agent who is proposing something

6.34 generateProposalOtherResponseEvent

Someone is responding to proposal to other agent (and none of them is currently with us). We will generate emotions just in case we have positive feelings toward the agent who is responding to the proposal. If he accepts the proposal we will generate negative emotions and vice versa. The intensity is also dependent on the type of proposal – kiss and sex proposals generate medium positive/negative emotions while other proposals just low positive/negative emotions.

Emotions:

- joy and liking – low to medium intensity – toward the responding agent (in case the proposal was rejected)
- distress and disliking – low to medium intensity – toward the responding agent (in case the proposal was accepted)

6.35 generateProposalResponseToOtherByAgentWithEvent

The agent we are currently with is responding to proposal made by some other agent. This is the same as with generateProposalOtherResponse event above, except the emotion intensities are medium to high.

Emotions:

- joy and liking – medium to high intensity – toward the responding agent (in case the proposal was rejected)
- distress and disliking – medium to high intensity – toward the responding agent (in case the proposal was accepted)

6.36 generateProposalByOtherToAgentWithEvent

Someone is making a proposal toward agent we are currently with! We will generate negative emotions with intensities depending on the type of the proposal. Sex and kiss proposals will trigger emotions with higher intensities.

Emotions:

- distress and disliking – medium to high intensities – toward the agent who made the proposal

6.37 generateProposalResponseEvent

Someone has responded to our proposal. If accepted we will generate positive emotions and vice versa. The feeling toward agent is not important here.

Emotions:

- joy and liking – medium intensity – toward the other agent (if the proposal was accepted)
- distress and disliking – medium intensity – toward the other agent (if the proposal was rejected)

6.38 generateProposalIgnoreEvent

Someone we have made proposal to is not responding to it (he ignored our proposal). We will be a bit unhappy by this.

Emotions:

- anger – low intensity – toward other agent

6.39 generateReceivedProposalEvent

Someone has made proposal to us. We will react based on the feelings toward the other agent. If the proposal will be accepted we are happy, if we will reject it we will be offended by kiss or sex proposals. If we don't like the agent (feeling negative) we will be offended a little by any proposal and vice versa (except the kiss and sex proposals, that are handled separately).

Emotions:

- joy and gratitude – low to medium intensities – toward proposing agent
- distress and anger – low to medium intensities – toward proposing agent

6.40 generatePolymorphActionBiteEvent

The polymorph has bitten us! We will generate negative emotions.

Emotions:

- fears confirmed and fear – high intensities – toward polymorph. Fears confirmed emotion will lead probably to other negative emotions such as distress. Otherwise it is not monitored by the agents.

6.41 generatePolymorphDislikeFearEvent

We are close to hostile polymorph who is making bad noises. We are afraid of him and do not like him. There is ongoing reduction of intensity based on the number of the negative messages the polymorph has made to us.

Emotions:

- dislike and fear – low to high intensity – towards polymorph

6.42 generatePolymorphBiteEvent

The polymorph has damaged us – caused our health to be lower. Not generated right now, will be probably erased and just PolymorphActionBiteEvent will be used.

Emotions:

- fear confirmed and fear – high intensity – toward polymorph

6.43 generatePolymorphLikeEvent

We have encountered friendly polymorph and he is making friendly noises toward us. There is ongoing reduction of intensity based on the number of messages the polymorph has made to us.

Emotions:

- liking – low to medium intensity – toward polymorph

6.44 generateWaitAgentNotReturnedEvent

We were waiting for the agent and the agent didn't come to us in time.

Emotions:

- distress and disliking – medium intensity – toward the other agent

6.45 generateWaitAgentReturnedEvent

We were waiting for the agent and the agent returned in time.

Emotions:

- joy and liking – medium intesity – toward other agent

6.46 generateWaitInProgressEvent

We are currently waiting on the agent. Every few seconds (30) this event is generated while waiting.

Emotions:

- distress and anger – low intensities – toward the agent we are waiting for

6.47 generateWeHitPolymorphEvent

We have hit the polymorph with the gun – we are proud on ourselves.

Emotions:

- joy and pride – high intensities – toward us

7 Java packages and classes overview

Here we will present a brief project Emohawk Java packages overview. We will list what is the package responsible for and what classes it contains.

7.1 Package AlmaBasedModel (almabasedmodel)

This package contains classes that provides the connection to ALMA emotion model. We need to make a new instance of emotion model and provide it with our inputs. ALMA is configured through XML configuration files. Also this package is used to provide classes, where we can store the ALMA affect outside ALMA (currently emotions and mood), and contains our affect computation methods.

7.1.1 Class Aemotion

Class used to store ALMA emotion outside ALMA.

7.1.2 Class AEmotionState

Here we compute our own affects based on the affects got from ALMA. (the feeling attitude)

7.1.3 Class AEventGenerator

Here all the emotion events for ALMA are appraised and sent as input to ALMA.

7.1.4 Class AMood

Class used to store ALMA mood outside ALMA.

7.1.5 Class PogamutALMA

Here we create an instance of ALMA emotion model. Main class for Pogamut -> ALMA connection.

7.2 Package bot

This is the main behavior package. The IVAs finite state machines are defined in classes here. Also the classes needed to make a new instance of our IVAs in Pogamut are here. This classes also provide the connection with the UT04 environment.

7.2.1 Class EmotionalBot

The parent class of our agent decision making class. Some methods providing decision making are defined here (the methods that are the same for both the girl and the boy). The only state preprapareScenario is defined here.

7.2.2 Class EmotionalBotModule

This module provide information for Guice, so our agent can be properly instantiated.

7.2.3 Class EmotionalBotTestCase

This is the main class we use for running our scenario. It will instantiate all the four IVAs we use and as well quit the scenario after 10 minutes and invoke methods for storing the scenario run.

7.2.4 Class EmotionalFemaleBot

Here the girl control logic is defined (all states and gender based decision making methods).

7.2.5 Class EmotionalFemaleBotModule

This module provide information for Guice, so our female agent can be properly instantiated.

7.2.6 Class EmotionalMaleBot

Here the boy control logic is defined (all states and gender based decision making methods).

7.2.7 Class EmotionalMaleBotModule

This module provide information for Guice, so our male agent can be properly instantiated.

7.2.8 Class EventGenerator

This class is responsible for processing of the text messages in order to generate emotion events. Also the information about last time of actions and proposals someone did to us are got here and stored in agent memory.

7.2.9 Class PolymorphBot

Here the polymorphs reactive behavior based on doLogic method is defined.

7.2.10 Class PolymorphBotModule

This module provide information for Guice, so our polymorph agent can be properly instantiated.

7.3 Package info

In this package we have classes that we use to a) store the IVAs history (e.g. What actions or proposals or etc. our IVA made to other IVAs and/or what actions etc. some other IVAs made to our IVA). Also the enumeration constants we use in Emohawk are defined in classes here.

7.3.1 Class ActionType

Here we have all scenario action identifiers defined.

7.3.2 Class ConversationInfo

This class stores information about a casual conversation with some other agent. When was the last time we have sent some message to target agent and vice versa...

7.3.3 Class ConversationType

Here we have all scenario conversation identifiers defined.

7.3.4 Class EventId

Here we have all scenario emotion events identifiers defined.

7.3.5 Class ItemRequest

This class stores information about item request we have received from some agent.

7.3.6 Class PlaceType

Here we have all scenario places identifiers defined.

7.3.7 Class PlayerInfo

This class is used to store information about agent to agent interaction. Here we will have the information about last times we have issued some action toward the agent and as well about last time the agent issued some action toward us. Moreover information about the last time we were at cinema with the agent or at park are stored here as well.

7.3.8 Class PolymorphEventInfo

This class stores information about an event the polymorph may triggered for our agent. This will help us to habituate toward polymorph events that are triggered regularly.

7.3.9 Class ProposalInfo

This class stores information about one proposal. E.g. if this proposal was accepted or rejected. Who is the agent this proposal if for/from etc.

7.3.10 Class ProposalType

Here we have all scenario proposals identifiers defined.

7.3.11 Class ScenarioItemType

Here we have all scenario item identifiers defined.

7.3.12 Class StateType

Here we have all state identifiers defined.

7.3.13 Class ScenarioType

Here we have all scenario identifiers defined.

7.3.14 Class TimerType

Here we have all timer identifiers defined.

7.4 Package logging

In this package all the objects needed to store and process the scenario run are present. In other words the automatic part of the analysis of our scenario is defined in class here.

7.4.1 Class ActionLog

Class used for storing of action issued in the scenario to a binary file.

7.4.2 Class AgentLogProcessor

This class is used to create a graph and text files outputs for all the experiments. Also simple analysis is conducted (output again in text files and graphs).

7.4.3 Class AgentLogging

This class provides a storage of all events, action, proposals etc. occurred for one agent during the scenario. This information will be stored to Java binary file for future analysis.

7.4.4 Class AgentStateLog

Class containing a snapshot of some of the agent internal variables at certain time in the scenario.

7.4.5 Class EmotionEventLog

Class holding one emotion event that occurred at certain time for our agent in the scenario.

7.4.6 Class EmotionsLog

Class used to store a complete list of agents emotions at certain time in the scenario.

7.4.7 Class FeelingLog

Class used for storing a feeling of agent to other agent in the scenario to a binary file.

7.4.8 Class FeelingSceneResult

Used for scenario analysis – holds the feeling development result for certain sub-scene in the scenario.

7.4.9 Class ItemLog

Here we log information about item we have given to someone or item we have received from someone.

7.4.10 Class LocationLog

Class used for storing a location of agent in the scenario to a binary file.

7.4.11 Class MoodLog

Class used for storing a mood of agent in the scenario to a binary file.

7.4.12 Class RotationLog

Class used for storing a rotation of agent in the scenario to a binary file.

7.5 Package utils

This helper package contains two classes with static methods we use in project Emohawk. The classes here are never instantiated.

7.5.1 Class Algeb

This class contains methods for our simple obstacle avoidance code implementation.

7.5.2 Class Time

Through this class we retrieve information about current time.

8 UT04 part

In UT04, we have modified package GBScenario. This is UT04 mod based on GameBots2004 classes. We have performed following modifications:

1. Our agents are able to see other agents when they are less than 300 UT units (~3 meters) away from them – even if our agent is turned with his back to them.
2. We have created a command that triggers sending all of the players in the environment through synchronous PLR messages (even of those that are not visible). This is used at the beginning of the scenario to get Ids of all the agents for all the agents.
3. We have added civil non-violent models for our IVAs – we needed to set up classes that holds information about meshes and textures.
4. We have added command and message for handling the item exchanges between our agents.
5. We have implemented an emitter that is capable of emitting flares we use for emotions visualization. Also we have created a command that can set up and control this emitter.