Jadex/JBdiEmo Emotional Agents in Games with Purpose: a Feasibility Demonstration

Štefan Korečko*, Branislav Sobota, Peter Zemianek

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, Košice, Slovakia; *stefan.korecko@tuke.sk

Simulation Notes Europe SNE 26(4), 2016, 195 - 204
DOI: 10.11128/sne.26.tn10351
Received: November 10, 2016
Accepted: December 5, 2016 (Special Issue Review)

Abstract. The jMonkeyEngine 3D game engine, combined with Jadex agent system and JBdiEmo emotional extension may offer a suitable toolset for effective creation of feature-rich virtual environments, provided that an appropriate interface, allowing to use the full potential of all included components, exists. Then, such environments may profit from the jMonkeyEngine ability to model and simulate the physical world and capability of Jadex and JBdiEmo to express both rational and emotional aspects of characters inhabiting it. One of the meaningful ways of utilization of such environments is to use them as virtual testing grounds for software controllers of various devices, embedded to them. To involve real humans in the testing, they may have a form of a game, where the testing occurs during an interaction between the devices and players. In this paper we present both the interface and the embedding on an emergency simulation game called JFireEmSim2. The primary goal of the player in the game is to rescue a family from a house under fire and the controller embedded into it is of a simple autonomous cleaning robot. The paper describes the architecture of the game, focusing on the interface, implementation of characters as Jadex and JBdiEmo agents and embedding of the controller. It also discusses suitability of the components for the given task.

Introduction

Jadex [5], [6] is a software framework, where applications are composed of active, service providing components. The components can be implemented in several forms, with cognitive BDI agents being historically the first and probably the most sophisticated ones.

BDI stands for belief–desire–intention, a model of human practical reasoning, introduced in [4]. A BDI agent has beliefs expressing what it knows about itself and its environment, desires that represent states it would like to achieve and intentions that provide means to achieve the states.

The simplicity of BDI is the source of both its popularity and criticism. The critics point out that BDI focuses on rational reasoning and ignores other aspects, such as emotions. To deal with this issue in the context of the Jadex framework we designed and implemented JBdiEmo [11], [12] emotional engine, which uses a modified version of the OCC model of human emotions. The OCC model [16] considers emotions to be results of cognitive processes and divides them into three classes: emotions that are reactions to events, reactions to agents and reactions to objects. The version used in JBdiEmo originates from [17] and has a form of an inheritance-based hierarchy of emotions.

A promising utilization of Jadex and JBdiEmo is in computer games, where they can simulate both rational and emotional behavioral aspects of non-playable characters (NPCs), modeled as agents. Such utilization is supported by Jadex via a visualization interface for the jMonkeyEngine (jME) 3D game engine. In [13] we used the interface to develop an emergency simulation game, where the player's goal is to rescue people from a flat under fire. The game proved that JBdiEmo can be used with the interface without any modifications, but also revealed that the interface provides only limited access to jME.

In [13] we intended to use the platform consisting of Jadex, JBdiEmo and jME for ordinary computer games and serious games for education and training. However, from the control system point of view, it is also interesting to examine the possibilities of its utilization for the so-called games with a purpose (GwP) [2], i.e. for games where players are helping to solve serious problems.

GwP try to hide their true purpose behind an interesting gameplay and players can be completely unaware of it. In our case, the GwP should provide a virtual environment for evaluation and testing of software controllers or their executable prototypes. In the game, the controller can be represented by an entity that resembles the device to be controlled by it in the real world. And the purpose of the player will be to evaluate the controller by interacting with the corresponding entity during the gameplay.

There is one prominent issue to deal with in order to use the Jadex/JBdiEmo/jME platform in such way: A new interface between Jadex and jME that will overcome the limitations of the visualization interface, provided by Jadex, should be developed. This is necessary to be able to use the full potential of jME for virtual environment creation. The new interface has been designed and experimentally implemented in a new emergency simulation game, named JFireEmSim2. JFireEmSim2, which is presented in this paper, shares the basic goal with the original one [13], but features more sophisticated gameplay and graphically-rich environment. To demonstrate the feasibility of the GwP idea outlined above, it also includes a formally verified control program supervising an autonomous robot.

The rest of the paper starts with a short overview of the Jadex/JBdiEmo/jME platform and limitations of the original visualization interface (section 1). Section 2 presents the new game, focusing on its overall architecture, new Jadex/jME interface and implementation of NPCs as Jadex/JBdiEmo agents. Section 3 elaborates the GwP idea by presenting a cost-effective version of corresponding development process and embedding of the control program into an already existing game, in this case the JFireEmSim2. The paper concludes with a summary of achieved results and plans for future research and development.

1 The Platform

The software platform, both games are built on, consists of three components, implemented in Java: the Jadex agent system, the emotional engine JBdiEmo and the game engine jMonkeyEngine (jME). While Jadex and jME are standalone components, JBdiEmo can be used only with Jadex.

1.1 Jadex

In Jadex agents are defined by beliefs, goals and plans. Goals stand for desires and plans for intentions. The plans are executed with respect to the current goals of the agent, messages the agent receives or events occurring in the system. On the other hand, an execution of a plan may result in new goals, messages or events.

Agents are specified as classes and actual agents are instances of these classes. Each class is described by an agent definition file (ADF), written in XML. An ADF defines all BDI elements (i.e. beliefs, goals and plans). It contains names, parameters and properties of the elements and links to Java classes that implement beliefs and plans. At runtime, an agent consists of a model, created from his ADF, and a set of instantiated objects, representing his plans and beliefs.

Beliefs are stored in a form of facts. These are accessed by goals and plans to acquire stored data values. A fact can be an arbitrary Java object.

Goals represent agent's specific motivations such as to reach a new state or to perform some activities. Jadex implements a full lifecycle for goals [5]: A goal can be created when its creation condition is met, or during an execution of some active plan. A newly created goal is adopted and enters the main part of its life cycle. On the basis of its context condition, an adopted goal can be in the state ‘option’ or ‘suspended’. A goal in the ‘option’ state eventually becomes active and executes corresponding plans. The active goal can be then suspended if the context condition is broken during the execution of the plans. If the plans of the goal achieve desired results, the goal is finished in the state of success. If they fail to achieve them, it is finished in the state of failure. A goal may also be dropped at any time if its results are no longer desired. Creation, context and other conditions are specified in ADF.

Plans provide means to achieve active goals. For every currently active goal, plans are executed until the goal is reached, suspended, failed or dropped. They are instantiated at runtime when corresponding events (e.g. a goal creation) are triggered. They are also capable of creating new goals.

Configuration of a whole Jadex application is specified in an application XML file, which defines how many agents of which type will populate the application. If 2D or 3D visualization is used, its settings are a part of this file, too.

1.2 JBdiEmo

JBdiEmo engine extends Jadex agents by emotions associated with their rational plans, beliefs and goals. The set of all emotions an emotional agent has, together with their intensity values, form his emotional state. The engine supports the whole modified OCC model and how the emotions are mapped to BDI elements is shown in Figure 1. JBdiEmo is implemented in such a way that agent's actions can influence his emotional state and the emotional state can, in turn, influence agent's further actions.

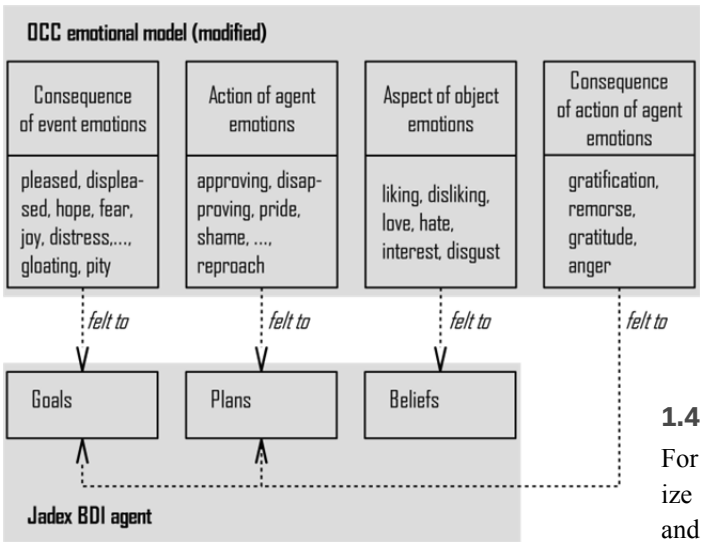

Figure 1: OCC to BDI mapping as implemented in JBdiEmo.

The engine consists of seven components:

1. JBdiEmo core, responsible for representation of agent's emotional state, checking of eliciting conditions of each emotion, emotion intensity value calculation and messaging between emotional agents. Agents access it via a belief.
2. Emotional agent initialization plan, implementing an initial agent model mapping. It initiates processes that repeatedly monitor agent's events and the whole Jadex platform for a presence of other emotional agents.
3. Inter-agent emotional messaging plan, providing message delivery between emotional agents.
4. Language extension allowing to distinguish ordinary beliefs, goals and plans from the ones with associated emotions (i.e. from the emotional ones).
5. GUI for visualization of the actual emotional state of emotional agents. It also shows history of the events that influence emotional plans, beliefs or goals (emotional events).
6. Logger recording emotion intensity values to XML files for future processing.
7. Helpers, which provide mathematical calculations and other auxiliary functionality.

1.3 jMonkeyEngine

jMonkeyEngine (jME) is an open source 3D game engine, built on the OpenGL graphic library. It also provides an integrated development environment, called jMonkeyEngine SDK, which is based on the NetBeans Platform. Thanks to the features like material and terrain editors the comfort of game development in jME is comparable to commercial engines, such as UDK or Unity. In should also be noted that jME uses Bullet to simulate physical phenomena, which, according to [7], is one of the more accurate physics engines available in contemporary games.

1.4 Jadex 3D visualization interface

For Jadex, jME is only one of several options to visualize agent behavior. The other ones are textual output and 2D graphics. The application XML file is responsible for the visualization configuration. Here, 3D models and animations are associated with agents and models, textures and positions of other objects are defined. A direct access to jME is also provided, but only to a subset of its features.

The master-slave relationship between Jadex and jME prevents developers to use important jME features, such as more sophisticated visual effects, collision detection and physics engine. As [13] shows, this puts several constraints on the visual appearance of the game and forces developers to implement mechanisms like collision detection on the Jadex side. In consequence, the time spared by using Jadex to implement NPCs can be lost because of the additional implementation work.

2 JFireEmSim2 Game

The constrains the original visualization interface puts on developers, may render the whole idea of using Jadex and JBdiEmo for NPCs implementation inefficient. Fortunately, they can be overcome by designing a new interface that put Jadex and jME in more equal position.

The *JFireEmSim2* game (Figure 2), which implements such interface, is situated in a village, where the player has two goals. First, he has to rescue a family of four from a house under fire; and second, he has to save a depressed person on a nearby cemetery before he commits suicide. To successfully save the family, all its members have to be taken out of the house. A person may refuse to follow the player because of the fear of getting burned or due to the position in the family.

The actual appearance of the game can be seen in Figure 2, where the situation after saving the first member of the family is captured. There are health bars (red) for all the family members (father John, mother Marie, son Joe and daughter Jane) in the upper left corner. Player's health and extinguisher status are shown in the lower left corner. The player's character is the fireman on the right side, seen from behind. The house under fire is the wooden one on the left. There are two persons in front of the house; a neighbor in white T-shirt, observing the event, and the saved person (Joe, in blue shirt). In the background we can see another fire site and a fireman. Their role is described is Section 3.

Structurally the game can be divided into three components.

1. Game core with the entry point of the game and classes defining the basic gameplay, appearance of the game and user input processing. They have white background and names typed in normal font in the class diagram in Figure 3 and are explained in more detail in Section 2.1.
2. Jadex/jME interface allowing non-restricting communication between both frameworks. It consists of two classes, Communicator and AgentControl (white background and names in bold in Figure 3), described in Section 2.
3. Agents representing NPCs, which implement their behavior. The classes and ADF files belonging here have light gray background in Figure 3, and are treated in Section 2.3.

There is also a fourth part in Figure 3, consisting of 5 classes with dark gray background. These belong to the control program embedded into the game and are described in more detail in Section 3.

2.1 Game core

The core of the game is designed as a typical jME application with its main class, `App`, implementing the jME base class `SimpleApplication`.

The class `App` contains the entry point of the game (method `main`) and methods required by `SimpleApplication`, such as `simpleInitApp` to initialize and `simpleUpdate` to update the game in each game loop cycle. Its properties, among others, provide access to the physics engine (`bulletAppState`), handle various objects in the environment, such as individual fire sites (properties `fire`, `firePositions` and `fireNodes`) and implement 2D user interface elements (e.g. `hudControl` to show health of the player and NPCs). `App` also includes an object called `start`, which initializes Jadex in a separate thread.

The player of the game is represented by the `Player` class, which defines its graphical appearance, including animations, properties (e.g. health, extinguisher charge level) and keyboard and mouse input processing.

NPCs are defined on three levels, first two of them belonging to the core:

1. `Character`, a class which provides basic functionality for all NPCs in the game, such as movement, collision resolution and animations. The class also defines an abstract method `act`, which should define NPC behavior and all its descendants must implement it.
2. Inherited classes, holding aspects specific to corresponding character category. These are SavingPerson for the family members, OtherPerson for a neighbor observing the situation and QuestPerson for the person about to commit a suicide on the cemetery. These classes are connected to Jadex agents (level 3) via corresponding AgentControl objects of the interface and their instances, one for each NPC, are properties of the App class.
3. Jadex agents, specifying their rational and emotional behavior.

All classes implementing entities that can be seen in the game are also connected to corresponding 3D models, textures and animations.

2.2 Jadex/jME interface

The new interface, implemented by classes `Communicator` and `AgentControl`, is designed as universal; it doesn't even require the part connected to Jadex to be implemented in jME. Its purpose is to keep the state of objects representing NPCs on the jME side synchronized with beliefs of corresponding Jadex agents.

Figure 2: JFireEmSim2 screenshot.

The class `Communicator` is implemented using singleton design pattern to ensure that only one instance of it is available in the game. Its property `agents` holds a list of `AgentControl` instances, one for each Jadex agent. The `Communicator` itself just allows to add and remove agents, so the whole synchronization is in the hands of the `AgentControl` objects, which write values to agents' beliefs, with their put methods and read belief values with `getBoolean`, `getInt` and `getFloat` methods.

While on the jME side the interface is accessed via the property `agent` of the classes inherited from `Character`, on the Jadex side it is done through a belief called `shared` and plan `UpdatePlan`. These are defined for each agent. The value of `shared` is an instance of the corresponding `AgentControl` object, obtained via the `Communicator`. The `UpdatePlan` automatically updates beliefs of the agent every time the values stored in the corresponding `AgentControl` object are changed.

2.3 NPCs behavior

The behavior of NPCs in the game is defined almost exclusively by Jadex agents and each NPC has its own ADF. In Figure 3 ADFs are depicted in the form similar to classes, i.e. divided into three blocks. The first one contains stereotype `<<ADF>>` to distinguish them from classes and name of the ADF. The second one lists agent's beliefs and the third one goals. Plans are shown as separate classes, connected to corresponding ADFs. The diagram doesn't show goals and beliefs common to all agents, i.e. the ones belonging to jBdiEmo and the belief `shared`. The goals and beliefs of `Marrie`, `Joe` and `Jane` are similar to those of `John`: The `family_saved` belief is specific to `John`. `Joe` and `Jane` don't have the `child_saved` belief, but have additional belief called `reproached`. `Joe` also has the belief `jane_saved` and `Jane` has `joe_saved`.

To illustrate how the agents define behavior of the NPCs, let us have a look on the family to be saved. When the game begins, an active goal of all family members is wander and `WanderPlan` is carried out. This means that the persons randomly wander around the house. If a person gets too close to a fire, the `run_from_fire` goal becomes active and `RunFromFirePlan` tries to get the person to safe distance. However, when the intensity of the `fear` emotion, felt to the goal `run_from_fire`, becomes too high, the `cry` goal becomes active. This initiates the `CryPlan`, which makes the person stop for few seconds and perform appropriate animation. Another aspect that can abort `RunFromFirePlan` execution is the value of `disapproving` emotion felt to the `wander` goal being higher than `0.7`. Then the `GiveUpPlan` is executed, which means that the person stays in place and only the player's presence can change it by making the `stay_calm` goal active and `StayCalmPlan` executed. When the player commands a person to follow him, the `follow_player` goal and `FollowPlan` are activated, provided the corresponding conditions are met: the mother refuses to follow until the children are saved, and the father is the last one to leave. Otherwise, the `reject_follow` goal and `RejectPlan` are activated.

3 Controller in Game

Now, let us assume that we are developing a safety-critical part of a control program for an autonomous cleaning robot, which works as follows: The robot stores a list of locations to clean in its memory. After it's turned on, it goes to the first location from the list and cleans it. Then it proceeds to the next one. After cleaning all locations, it goes to a parking position and switches to a standby mode.

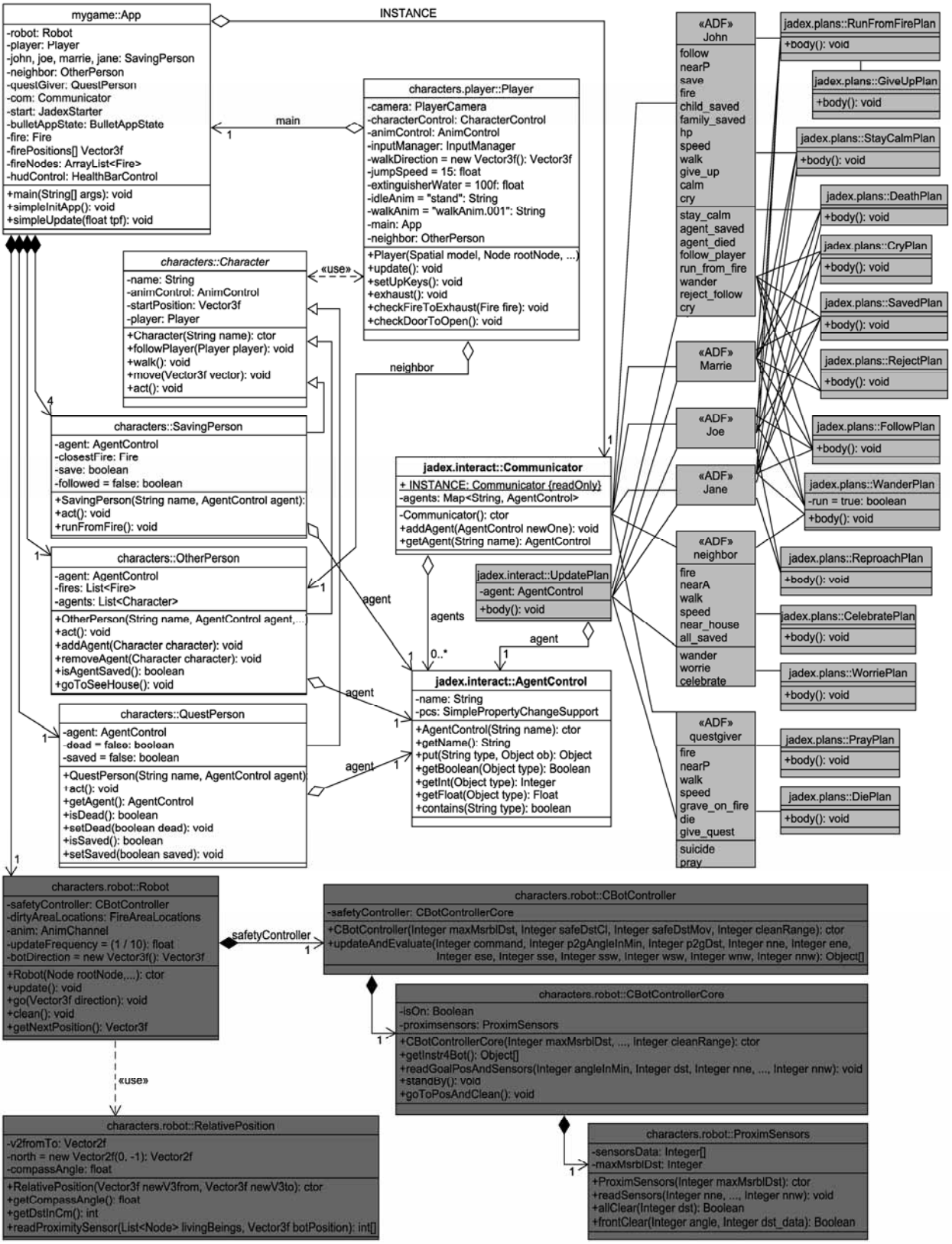

Figure 3: UML class diagram showing essential part of the JFireEmSim2 game structure.

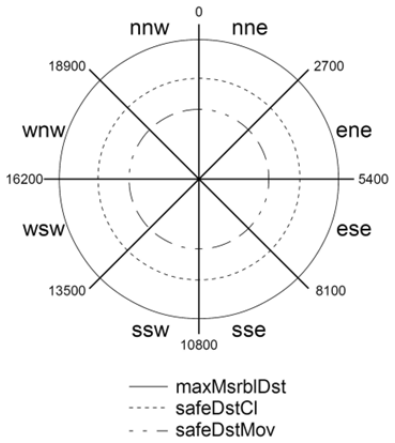

Figure 4: Cleaning bot sensors arrangement.

The robot should be able to perform its job in public areas, so from the safety point of view it is critical to prevent it from hurting people. To detect them, it possesses a circularly arranged sensor array (Figure 4). The sensor array returns 8 values, `nne`, `ene`, `ese`, `sse`, `ssw`, `wsw`, `wnw` and `nnw`. The value `nne` (north north east) is the distance to a nearest person, detected in the region from north (compass angle `0` minutes) to north east (`2700` minutes), `ene` (east north east) the distance in the region from `2700`` to `5400`` and so on. If no person is detected in a region, then the corresponding value is equal to the maximum distance, measurable by the array (`maxMsrblDst`). The robot may hurt someone when cleaning as the cleaning process is harmful for anyone close enough or when a collision occurs during movement. To prevent this, the control program should obey safety critical properties, which can be formulated as follows:

1. The cleaning cannot start or continue if anyone gets as close or closer to the robot as the distance safeDstCl.
2. The robot cannot move if anyone gets as close or closer to its front as the distance safeDstMov.

According to this specification, we design and implement the control program part with a method `updateAndEvaluate` as its interface (Figure 5). The first three parameters of the method specify what the robot should do in the given situation, and the rest (`nne` to `nnw`) are readings from the sensor array. The parameter command defines whether the robot should:

- switch to standby mode immediately (value 0)
- go to a specified position and then clean (1) or
- go to a specified position and stand by (3).

The specified position is given as a compass angle (`p2gAngleInMin`) and distance from the current position (`p2gDst`). The method evaluates the situation and issues instructions for the robot to follow. These include commands to turn the robot on or switch it to standby (the output parameter `botOn`), to start or to stop the cleaning process (`botCleaning`) and the destination where the robot should go (`angleInMin` and `dst`).

Now, assume that the control program part has been developed using formal methods such as B-Method [1], so we are sure that the safety critical properties hold in its implementation. What we are not sure is whether the distances `safeDstCl` and `safeDstMov` are set optimally. They should be large enough to prevent the robot from hurting people, but not too large, as it will cause the robot to interrupt its operation too often. To establish the distances, simulation experiments can be used.

3.1 Jadex/JBdiEmo/jME as simulation platform

We consider Jadex/JBdiEmo/jME to be a suitable platform for such simulation because of the following reasons:

1. Support for quick construction of a virtual environment where the robot will operate, thanks to the editors of jMonkeyEngine SDK and ability to import models already available online or created in different applications. For example, most of the buildings in JFireEmSim2 are freely available models and the house under fire has been created in SketchUp (ww.sketchup.com). NPC models were created in MakeHuman (http://www.makehuman.org/) and their animations in Blender (https://www.blender.org/).
2. Built-in physics simulation in jME, provided by jBullet, a Java port of the Bullet engine. Bullet is used in several simulation platforms, such as Gazebo (http://gazebosim.org/) and V-REP (http://www.coppeliarobotics.com/). The jBullet port has been used for simulation purposes as well, for example in [3] for cells and surrounding fluid.
3. Possibility to populate the virtual environment with characters with complex personality, thanks to the integration of Jadex and JBdiEmo. In addition, the characters can be developed separately and integrated to the environment afterwards.

Both jME and Jadex have already been used for simulation purposes. The Jadex case is described in [6] and jME is the basis of several robotics simulation environments, such as jmeSim [8] and MARS [18].

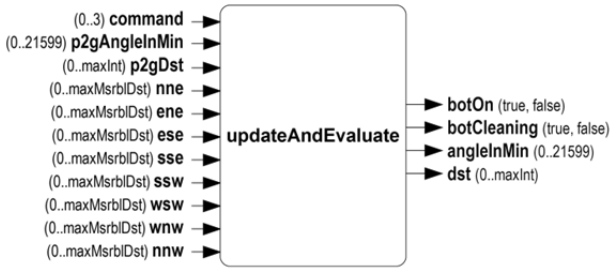

Figure 5: Cleaning bot interface.

3.2 Simulation vs. GwP

In principal, an environment created with Jadex/JBdiEmo/jME can be used in two ways:

1. For simulation experiments.
2. As a game with purpose.

The first case is de facto a game without player, with properties and behavior of entities inhabiting the environment given by their code and models only. So, different simulation experiments under equal conditions can easily be performed.

In the second case, a human player is involved and may interact with an entity representing the control program. The GwP element is that the interaction provides data essential for evaluation or adjustment of the control program, e.g. whether certain values of `safeDstCl` and `safeDstMov` may endanger persons that come close to the cleaning bot. This interaction can be completely natural, for example the cleaning bot can be situated in an area, which the player will visit either way.

3.3 Cleaning bot in JFireEmSim2

While the Jadex/JBdiEmo/jME platform allows to build a virtual environment relatively quickly, the more economical option is to embed the control system to be evaluated to an already existing game. This requires two tasks to be performed:

1. Create an entity that will represent the controlled system in the game.
2. Adjust the code of the game to integrate the controller.

To minimize the effort put into the first task we should consider reusing the assets already available in the game. For example, provided that the basic parameters like dimensions, weight and speed of the cleaning bot are similar to an average human, it can have a form of an additional fireman in JFireEmSim2 and the positions to be cleaned can be represented as fire sites to be extinguished (Figure 6).

Figure 6: The second fireman representing the robot approaching the fire sites, i.e. places to clean.

The controller can be integrated as it is shown in Figure 3 (classes with dark gray background). The controller itself consists of classes `CBotController`, `CBotControllerCore` and `ProximSensors`, which have been generated from formal specification, created and verified using B-Method. The `updateAndEvaluate` method can be found in `CBotController`.

The connection to the game is implemented via the class `Robot`. It serves the similar purpose as the class `Character` for NPCs, i.e. it provides visual representation of the robot (as a fireman). It is also responsible for executing the `updateAndEvaluate` method and performing actions according to the values it returns. The input parameters of `updateAndEvaluate` are computed from actual positions of the robot, place to clean, player and NPCs by the class `RelativePosition`.

4 Related Work

Several aspects presented here can be found in other sources, but the combination of using a game engine for the basic gameplay and simulation of the physical world, emotional agents for characters, behavior and the resulting game for evaluation of the control systems via seemingly ordinary gameplay remains unique to this work. Regarding jBdiEmo, to our knowledge, it is the only existing emotion-implementing extension for Jadex.

The idea of turning a computer game into a testing ground for control systems is in great extent realized with the real-time strategy game Starcraft. According to the survey [15], most of these works are implemented via the Brood War Application Programming Interface (http://bwapi.github.io/), which allows replacing a human player with a computer program and competitions are organized where bots play against other bots or human players.

A toolchain similar to ours has been used in [9], to implement a serious game that teaches players about energy consumption. To create the game the authors of [9] used jME, MakeHuman and Blender, too. The game doesn't include an agent system, but uses co-simulation via the Functional Mock-up Interface (FMI) to integrate thermal and physical models of a building and appliances. FMI should also be considered for a future version of the jME/Jadex interface or an interface between control programs and games.

The GwP idea has been formulated in [2] and GwP usually contain gameplay focused on solving specific problems, such as protein folding in Foldit [10] or finding program loop invariants in Xylem [14].

Other implementations of non-emotional or emotional artificial agents in computer games exist as well and a short overview and comparison to Jadex and JBdiEmo can be found in [13].

5 Conclusion

The new Jadex/jME interface, presented here as a part of JFireEmSim2 game, allows to use both Jadex and jME to their full potential when building virtual environments for games or simulation experiments. The experimental integration of the cleaning bot controller also proved feasibility of the idea of computer games utilization as testing environments with agent-based NPCs and active participation of players. While JFireEmSim2 in the state presented here provides only basic realization of the idea, it can be developed and experimented with in several different ways. One of them is to use Jadex and jBdiEmo to implement more complex gameplay and personalities of NPCs. The gameplay should include active interaction between the player and the robot, for example a task to adjust robot parameters for maximum performance.

Another way is to enhance the possibilities of player movements or perform experiments with selected groups of human players. We also would like to return to the importance of the idea with respect to formal methods, which is touched only lightly here, in a separate work. The JFireEmSim2 game is available by request from the authors.

References

[1] Abrial J. R. *The B-Book: Assigning Programs to Meanings*, 1st ed. Cambridge: CUP; 1996. 816 p., 1996.

[2] Ahn L. von . Games with a Purpose. *Computer*. 2006; 39(6): 92–94.

[3] Applewhite-Grosso T et al. A multi-scale, physics engine-based simulation of cellular migration, *2015 Winter Simulation Conference*; 2015 Dec; Huntington Beach, CA, 1230-1239. doi: 10.1109/WSC.2015.7408248

[4] Bratman, M.E. *Intentions, Plans, and practical reason*.1st ed. Cambridge, MA.: HUP; 1987. 200 p.

[5] Braubach L, Pokahr A, Jander K.. The Jadex Project: Programming Model. In: Ganzha M, Jain C L, editors. *Multiagent Systems and Applications: Volume 1:Practice and Experience*. Berlin, Heidelberg. Springer; 2013. p 33.

[6] Braubach L, Pokahr A. The Jadex Project: Simulation. In: Ganzha M, Jain C L, editors. *Multiagent Systems and Applications: Volume 1:Practice and Experience*. Berlin, Heidelberg. Springer; 2013. p 22.

[7] Erez T, Tassa Y, Todorov E. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx, *ICRA*, 2015, IEEE. pp. 4397–4404.

[8] Haber A, McGill M, Sammut C. Jmesim: An open source, multi platform robotics simulator. *Australasian Conference on Robotics and Automation*; 2012 Dec; Wellington. 270-276.

[9] Kashif A et al. Virtual Simulation with Real Occupants Using Serious Games. *14th International Conference of the International Building Performance Simulation Association*; 2015 Dec; Hyderabad. 2712-2719.

[10] Khatib F et al. Crystal structure of a monomeric retroviral protease solved by protein folding game players. *Nat Struct Mol Biol*. 2011;18(10): 1175–1177. doi:10.1038/nsmb.2119.

[11] Korečko Š, Herich T. On Some Concepts of Emotional Engine for BDI Agent System, *14th IEEE International Symposium on Computational Intelligence and Informatics*; 2013 Nov; Budapest. 527-532. doi: 10.1109/CINTI.2013.6705254.

[12] Korečko Š, Herich T., Sobota B. JBdiEmo – OCC Model Based Emotional Engine for Jadex BDI Agent System, *12th International Symposium on Applied Machine Intelligence and Informatics*; 2014 Jan; Herľany. 299-304. doi: 10.1109/SAMI.2014.6822426.

[13] Korečko Š, Sobota B, Čurilla P. Emotional Agents as Non-playable Characters in Games: Experience with Jadex and JBdiEmo, *15th IEEE International Symposium on Computational Intelligence and Informatics*; 2014 Nov; Budapest. 471-476. doi: 10.1109/CINTI.2014.7028721.

[14] Logas H. et al. Software Verification Games: Designing Xylem, The Code of Plants. *9th Int. Conf. Foundations of Digital Games*; 2014 April; Ft. Lauderdale.

[15] Ontanon S. et al. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Trans. on Computational Intell. and AI in Games*. 2013; 5(4):1-19.

[16] Orthony A, Clore G, Collins A. *The cognitive structure of emotions*. Cambridge: CUP; 1988. 207 p.

[17] Steunebrink B.R, Dastani M, Meyer J.J.Ch. The OCC model revisited. In: Reichardt D, editor. *Proc. of the 4th Workshop on Emotion and Computing - Current Research and Future Impact*, 2009.

[18] Tosik T, Maehle E. MARS: A simulation environment for marine robotics, *OCEANS 14 MTS/IEEE*; 2014 Sept; St. John's. doi: 10.1109/OCEANS.2014.7003008