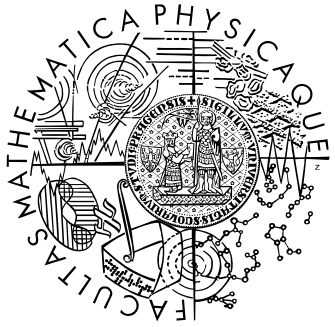
Charles University in Prague, Czech Republic
Faculty of Mathematics and Physics

BACHELOR THESIS



Radim Vansa

Výukové scénáře v projektu Pogamut
Educational scenarios in the Pogamut project

Department of Software and Computer Science Education

Supervisor: Mgr. Rudolf Kadlec

Study program: Computer science
Field of study: Programming

2009

I would like to give thanks to my supervisor, Mgr. Rudolf Kadlec, for his time and support. I would also like to express thanks to all other developers of the Pogamut project, namely to Mgr. Jakub Gemrot, Bc. Michal Bída, Jan Havlíček and Mgr. Ondřej Burkert. Special thanks belong to the team-leader of AMIS team, Mgr. Cyril Brom, Ph.D.

My thanks to my parents and family, whose support and guidance have helped me sail through difficult chapters of my life.

I hereby certify that I wrote the thesis by myself, using only referenced sources. I agree with lending the thesis.

In Prague Radim Vansa

Table of Contents

Název práce: Výukové scénáře v projektu Pogamut
Autor: Radim Vansa
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Rudolf Kadlec
e-mail vedoucího: rudolf.kadlec@gmail.com

Abstrakt: Tento projekt si klade za cíl rozšířit platformu Pogamut o nástroj pro neprogramátory, zvláště pedagogy, kteří chtějí využít 3D virtuální prostředí jako výukový prostředek. Měl by jim umožnit jednoduše navrhovat scénáře, ve kterých mohou testovat své žáky v různých situacích. Tyto scénáře by měly být zapisovány v přirozeně vyhlížejícím, lokalizovatelném jazyku interpretovaném v produkčním systému Drools. Jednou ze součástí této bakalářské práce je i implementace ukázkového scénáře, který zkouší schopnosti žáků orientovat se v městské zástavbě. Projekt Pogamut umožňuje prezentovat tyto scénáře ve špičkových virtuálních světech, například ve hře Unreal Tournament 2004.

Klíčová slova: výukové hry, virtuální realita

Title: Educational scenarios in the Pogamut project
Author: Radim Vansa
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Rudolf Kadlec
Supervisor's email address: rudolf.kadlec@gmail.com

Abstract: The goal of this project is to extend Pogamut platform with a tool for non-programmers, especially teachers, who want to use 3D virtual environment as a learning medium. They should be able to easily write scenarios, where they can test their students in different situations. One part of this bachelor thesis is the implementation of sample scenario proving student's abilities of orientation in urban environment. These scenarios can be scripted in naturally looking, localizable language interpreted in the Drools rule based engine. The Pogamut project allows to present these scenarios in first-class virtual reality worlds, for instance in Unreal Tournament 2004.

Keywords: serious games, virtual reality

Section 1
Introduction

In the last years we can notice a growing number of attempts to use computers at schools. They vary from basic applications for form filling (e.g. those used for english grammar tests as [1]), over interactive tutors and virtual laboratories [2] to very complex simulators requiring special hardware etc. Educational Scenarios (ES) can be classified as a tool for serious games [3], although they need not to be funny. These involve the use of games in education, training, health and public policy.

ES differ in the relationship between application's author and the person who creates the content of these applications. In hereinbefore mentioned applications, the content author (domain expert - teacher, psychologist, sociologist) usually needs to highly cooperate with the application's author because he himself cannot express his problem in computer terms. In ES the content author should be independent as much as possible.

ES is intended to be only a tool, not a complete application. Therefore it has to offer a way to be scripted by everyone, without any need for computer science knowledge or skills. To make computer-assisted training useful and serious part of the education, we have to produce sufficient amount of quality content. For this, we would need many specialists and we have to offer them an opportunity for standalone development. This is what ES does.

The example scenario that the submission contains should support training of orientation in urban environment which is part of the elementary schools' curriculum. Even some early-teenage children cannot properly navigate in unfamiliar city or town, although they have a map. This scenario tests basic navigation skills (using commands as turn left, go ahead), orientation using cardinal points as well as following path marked up in the map or even free movement between two locations within time limit. Tested player should also keep an eye on another character (e.g. his/her younger brother), evade losing him etc. Actions that are not linked with movement selection can be performed through dialogues.

Pogamut [4] platform provides platform-independent and environment-independent (game-independent) framework. The basic feature is Java API for agents control, but other projects on Pogamut platform also enable support for emotional model ALMA [5], gestures and face expressions or cognitively plausible architecture ACT-R [6][7]. Unreal Tournament 2004 (UT2004) [8] is used for ES as the virtual environment because this game is the only one supported in Pogamut so far.

UT2004 offers several game modes, but the basic principle is always the same: you (and your team) should shoot your opponents. There are many types of guns with different characteristics placed in the level[1]. You have to run around the level and collect better guns, ammunition for them and health packs in case that you are wounded. If you die, the opponent who killed you will increase his score and you will revive somewhere else with only the basic equipment.

It may seem not very suitable for educational purposes – it is a game full of violence, weapons and blood, but this is just one utilization of the UT2004 engine. With proper civil models and maps it can be used very well for serious games and education.

For scripting of scenarios was chosen the Drools [9] rules engine as it fulfilled all our needs.

1 The virtual area where the game is running.

Section 2
Related works

Thousands of computer-learning applications have been developed in last fifteen years. Here will be described a few of them which are using 3D virtual reality in an immersive way.

2.1 FearNot!

FearNot! [10] is a European framework V project that concentrates on improvised dramas addressing bullying problems. Children at the age of 8-12 were targeted. One of the main aims of this project was to create believable synthetic creatures allowing the user to build empatic relations with them.

FearNot! Provides various scenarios concerning bullying behaviour compiled by pedagogues but also automatically generates the story through emergent narrative techniques.

The interesting position of the player is not the major figure in the drama but his/her invisible friend, talking to the character through chat and viewing his actions and their consequences. This position was chosen because the situation of being the victim of bullying would appear too stressful for the child and would not satisfy the pedagogical objectives.

This project has been extensively tested in the UK, Germany and Portugal on more than 1000 children with satisfactory results.

2.2 Secure

Secure [11] is a virtual 3D online game for four players. It aims at epistemic problem-solving relative to work safety in the field of construction. The players should build a customer ordered hut within 60 to 90 minutes. They can move in the virtual world, dislocate objects, equip with them and use them. According to the website the environment and scenario are scripted.

There was a study [12] evaluating this project. The game was tested by 64 16-18 year old vocational students. Next to their score and speed was

observed the amount of communication between the players because the game is deeply aligned on players' cooperation. The authors of this study consider its results as positive.

2.3 ScriptEase

ScriptEase [13] project is the most similar to Educational Scenarios from all the projects that are mentioned here. It was introduced in [14] not as a final product but also a tool for non-programmers. ScriptEase utilizes Neverwinter Nights (NWN) [15] engine which already uses scripting but not in a very user-friendly way. On the contrary, ScriptEase provides a set of templates together with GUI tool.

The ScriptEase model is pattern template based, allowing designers to build up complex behaviors quickly without doing explicit programming. There is a library of event triggers and possible actions, which can be performed and the author only picks and joins them through the GUI.

When everything is set up, it generates the scripting code for the NWN.

ScriptEase is rather event-oriented. It allows to react very easily to opening a chest, entering some space etc. On the contrary, ES assumes that the scenario has some story line and the events just affect it. Although ScriptEase allows to change NPC's[1] dialog if the player had already talked to it, this is not actually a story line.

Somebody could see as a great advantage of ScriptEase no need to write code (even in user friendly DSL) but this is a matter of personal affection.

2.4 City Game

The City Game [16] does something very similar to the example scenario provided with ES. It creates a virtual city where the student can travel also using a map or compass. Additionally they can use photos and street signs. The program is proprietary but the city is defined in VRML [17].

The authors of the project have performed a study on 10 elementary school children and 10 university students and compared the results. The tested

1 NPC – non player character

subjects were asked to find a way from various initial positions to different target positions. The study concerned in methods how the tested subjects recognized the right way. The results showed that the subjects have used the map and road signs in small scope, they used mainly significant objects as museum with dinosaur or petrol station.

The City Game includes larger city than the UnrealVille provided with ES and it is more fitted to spatial navigation testing. Nevertheless the graphics looks rather amateurish. The fact that it is 11 years old (6 years older than UT2004) could be an excuse. But the major difference is that it has fixed abilities because the scenarios themselves are hard-coded into the program.

2.5 ADMS

ADMS – the Advanced Disaster Management Simulator [18] is a commercial simulator used for training incident command and vehicle operation. The learning scenarios can be open-ended and the development of situation is driven by trainee's actions. The virtual environment uses realistic physics and effects in contrast to games where the playability goes at the expense of realism and believability.

This project has been examined because testing of children behavior in emergency situations as wildfire or car accident was considered during the early specification of ES example scenario. Although such scenarios are not implemented (mainly because of the need of special graphics which is not available now), ES could serve very well for this purpose.

Section 3
Problem analysis

Although the aim of this project is to create a general system for writing educational scenarios, in fact it has a particular goal to be accomplished – functional example scenario. Here should the player move through a virtual city. He is navigated through messages on screen but also sometimes he can use map of the city and compass. If he gets lost and is not able to return back, he should loose the control over his movements and be returned back. There is also another character representing player's younger sibling. He follows the player but when the player runs too far from the sibling, the sibling stops and the player has to return back to him. Actions that are not linked with movement selection can be performed through dialogues. The complete description of the example scenario is enclosed as Appendix B.

Such task forces following components to be implemented:

- detection of player's movement and transcription from cartesian coordinates to named areas
- external control of player movements
- interaction with player through messages, sounds and dialogues
- ability to place another character into the environment and control it
- user friendly scripting system to control the scenario

These modules are the major contribution of this thesis.

The classic structured programming would be very abstruse for authors with no previous programming experience. That is why some other way of expressing the algorithms has been looked for. Here came the idea of using business rules engine came. It is a production system [19] that consists of a set of rules and according to the world state, it executes one or more of them. Each rule has only single condition block and single block of commands. This simplifies the code.

A great advantage would be the possibility of writing the statements

(conditions and commands) in simple sentences in human language. This is called domain specific language (DSL) [20]. It is another method that brings the programming closer to the author.

Drools [9] rules engine was chosen because it fulfills the requisites mentioned above. It allows declaring domain specific language [21] (DSL) and create proprietary syntax understandable for any English speaker. The translation, for example into Czech or German, could be done with almost no additional effort. Drools have been also successfully tested in games with variable laws [22]. According to [23] or [24] the Drools rules engine offers satisfactory performance.

Section 4
Background overview

According to the thesis submission, this work extends the Pogamut project. This section briefly describes Pogamut architecture and the way how it is used. The last section is related to the Drools rules engine.

To clarify our terms here is a short definition list:

- *Agent* is any being "living" in the simulated virtual reality.
- *Bot* is an agent that is controlled by artificial intelligence (AI).
- *Game* is third-party virtual reality that Pogamut connects to. These are actually not limited to games - it can theoretically be also a pure virtual world like Second Life or military simulator. Nevertheless the support of such environment is not implemented at this moment.
- *Player* is an agent that is controlled by human using graphical interface to immerse into the agent. The word player can also be used for identification of the human itself. For exact denotation of the agent we can use the word *character*.
- *World* is the sum of all objects and their interactions in the virtual reality.

Pogamut 3 running on UT2004 consists of three parts: PogamutCore, PogamutUT2004 and GameBots2004 (GB).

The general architecture of Pogamut is shown on figure 4.1.

4.1 PogamutCore

The kernel of Pogamut with codename GAVIAL - General autonomous virtual intelligent agent library - provides platform-independent communication with the game. On the control program's side it offers *world view* where you can listen to any event that occurs in the world and the agent can sense it.[1] New agents can be instantiated in the world and the

1 You can also have *server* which can listen to any event in the world without limitation. The agents are connected through *bot connection* while this server uses *control connection*.

control program can send commands to be obeyed by their body. On the bottom side is a communication link with the game itself, concretely implemented by the game dependent part. This core library also provides basic algorithms as A* [25] or Floyd-Warshall [26] on the navigation points[1] as well as another useful general utilities.

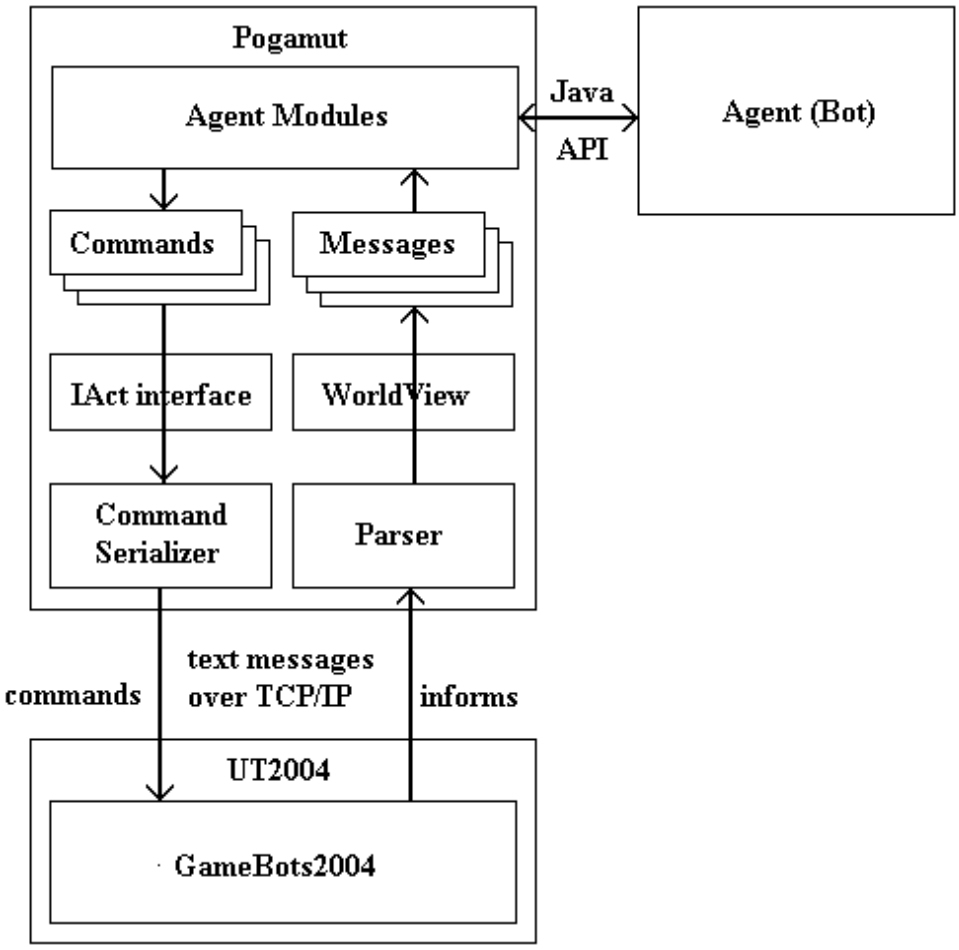


Figure 4.1 Pogamut architecture

4.2 PogamutUT2004

The PogamutUT2004 part contains the code dependent on the concrete virtual environment. The connection is implemented as TCP/IP socket connection between the game and the translator. Agent's request (*command object* executed on the agent's body) is serialized into single string message using the Command Serializer and sent over the network link. New events

1 Navigation grid is a graph with navigation points as vertices and links between them as edges. It is used to simplify the navigation in complex 3D enironment so the bots do not need to analyze its 3D shape. They are navigating between these points and they know that it is possible to safely travel between them without getting stuck.

in world, coming also as strings, are deserialized in Parser into *info objects* and passed through the world view.

PogamutUT2004 also provide more comfortable access to world view through *modules* that concentrate some information to be used any time, e.g. long-term memory. Nevertheless this is not the essential function.

4.3 GameBots

A great deal of UT2004's code describing the game rules is not compiled into binaries but it is scripted in language known as Unreal Script [27] [28]. This is object oriented language with C-like syntax, which is precompiled into bytecode to be interpreted during the game session.

GameBots2004 manages the second side of network connection mentioned hereinbefore - it translates command messages from Pogamut to relevant method calls, serializes another method calls into info messages and sends them over the network.

GBScenario is GameBots2004's shell which concentrates the UT2004's code which does not only control the bots but offers additional support for scenario-like modules as ES. It enables camera movement, provides civil mode skins and such improvements which are not used in regular death-match games.

4.4 Drools

Each rule has following structure:

```
rule "Rule name"
   modifiers
   when
        condition1
        condition2      } LHS[1]
        ...
   then
        command1
        command2        } RHS[2]
        …
end
```

1 RHS – right hand side of the rule, the block of commands
2 LHS – left hand side of the rule, the block of conditions

All the objects we are working with are called *facts* and are stored in a *working memory*. When some object is inserted (sometimes also called as "assertion") or updated, the conditions in all rules the rules that can be affected by the new fact are tested in a top-down order. If all the conditions are satisfied, the rule is *activated*. After all rules have been tested the active rules *fire*. Their succession is determined according to their position in the rule file and mainly according to the modifiers and their commands are executed also in a top-down order.

The conditions have usualy the form:

```
Rectangle(width>50, height=30 || color="blue")
```

This condition tests if there is an object of Rectangle class in working memory which is wider than 50 and either has height of 30 or is blue.

Such object can be also stored in variable and used in following conditions

```
$rect: Rectangle(width>50, color="blue")
Circle(perimeter = $rect.width)
```

Such condition block (LHS) tests whether there is a blue rectangle and circle with the same perimeter as rectangle's width.

The command block (RHS) is piece of regular Java code but it can use the variables declared in LHS.

Drools cannot come to know if some fact is modified. The user is responsible for reporting object's change by calling the update method.

The DSL statements are nothing more complicated than abridged regular expressions with "slots". The first condition above could be written in DSL as:

```
there is a rectangle wider than {width} with the
height of {height} or coloured with "{color}"

Rectangle(width>{width}, height={height} ||
color = "{color}")
```

This is just a brief look on Drools, for details see Drools Documentation [9].

Section 5
Design and implementation

This section describes the work that have been done to implement easy scenarios scripting support into Pogamut. The first subsection refers to project's architecture generally, following sections analyze the detail of implementation of particular components.

Definitions of terms used in this section:

- *Scenario* is the story that we perform.
- *Act* is one part of scenario. Acts form a tree structure – any act can be a leaf in this structure or include several another acts.
- *Area* is a segment of space. Those areas represented by single geometrical shape are called *simple areas*, those which are composed of more areas are called *complex areas*.
- *Designer* is a person who writes the scenario, usually teacher or psychologist with user computer knowledge.
- *Rulebase* is an object where all rules are stored during run-time.
- *Working memory* is a system of references to all objects the rules can work with.

5.1 Architecture overview

The overall architecture of connection between Pogamut, GameBots and Educational Scenarios is summarized in the figure 5.1. ScenarioAddons and UT2004ScenarioAddons are separate modules that have been created for ES' needs but they are general and can be used also in other independent projects. They are described in section 5.7 in detail.

The root object is an instance of Scenario class. During start-up it loads the rules related to this scenario, the structure of acts, the map, connects to UT2004 server using control connection (see footnote 1 in section 4) and waits for required number of players to join.

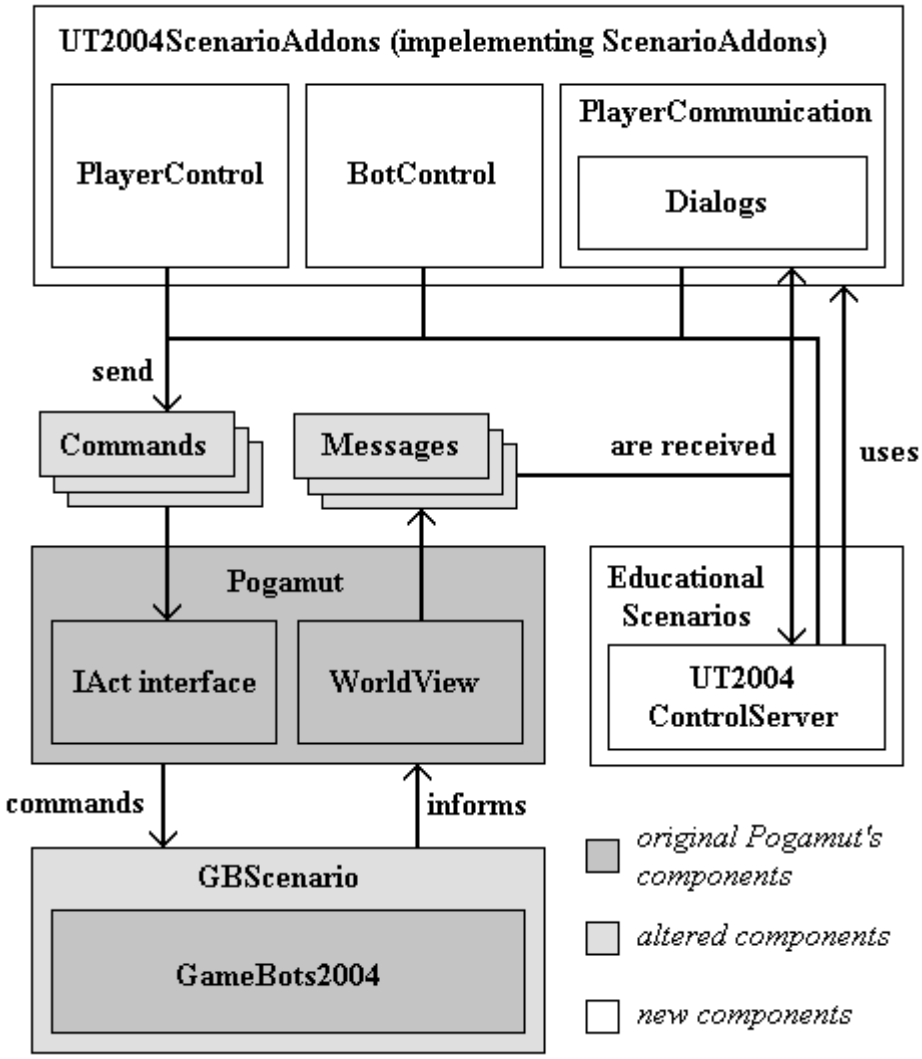


Figure 5.1: Connection between Pogamut, GameBots and ES

Localization was also taken into account – TranslatorFactory produces Translators which enable translation of words or phrases between different languages. The author's input data are in configuration file, map files and rule files. Because the rule files are written in pre-translated DSL the Translator is used only during the configuration file and map files parsing.

The internal structure of Scenario and the data flow is demonstrated in the figure 5.2. All the components are elaborated in following sections, this figure shows their connections.

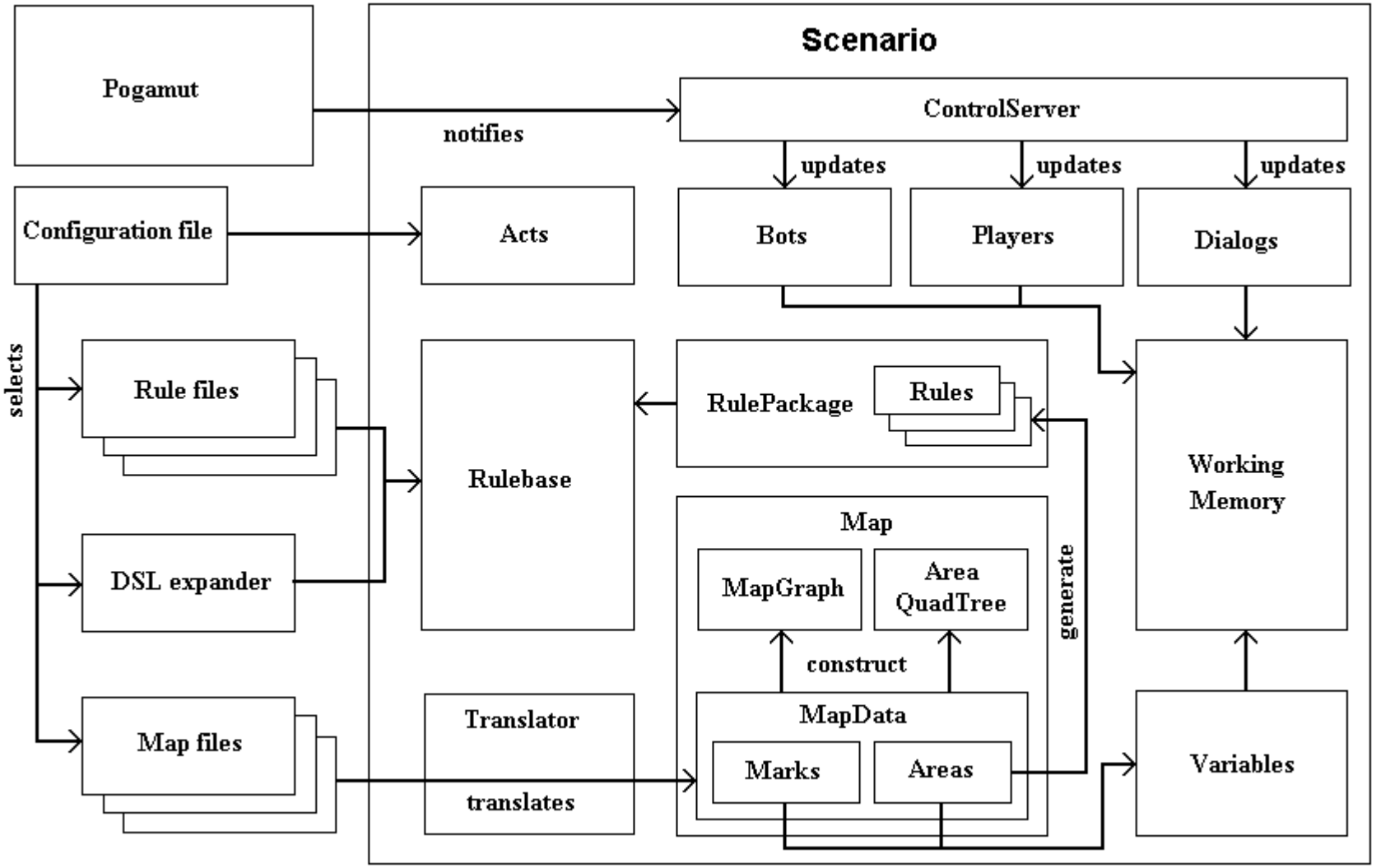


Figure 5.2: EduScenariosCore architecture

5.2 Acts

Our experience reflects that the scenario has often a hierarchical finite state machine structure. Sketching of the scenario before the actual implementation eases thinking the scenario over and also simplifies referring to parts of the story. For example a scenario “workday” should include acts “morning”, “work-shift” and “evening”, where “morning” includes “waking up”, “breakfast” etc. The workday is in one of these states in each moment.

Nevertheless such straight division is frequently not enough. When you see a car rushing at you, you'll try to get out from the road regardless of whether it's morning or evening and then continue with previous tasks. Therefore in ES the designer can sketch several such trees of acts, where the scenario is in one node of each tree at every moment.

The structure of these acts should be written down in the configuration file.

5.3 Map

The Map object is not absolutely necessary for every scenario but in most cases, including the example scenario, it is essential. The hierarchical structure of map should be described in standalone XML files based on the UT2004 map. All the information can be stored in a single file but is is recommended to use two files: first file describing the static layout of the city (streets, squares etc.) and second one with scenario-specific objects such as paths and points of interest.

These files can be created and edited using any text editor but such approach would be very uncomfortable. Therefore a WYSIWYG[1] editor is provided as an Eclipse[2] plug-in. More detailed description of this plug-in will be provided later in this document.

The structure of map should follow the way how we think about the parts – city consists of districts, districts consist of streets, streets of houses and

1 What You See Is What You Get – common abbreviation for editing tools with user friendly graphical interface.

2 Eclipse IDE is the recommended IDE for ES. This IDE was chosen because Drools already provide an Eclipse plug-in.

houses have floors and rooms. You can use any language you need for labeling of the levels of abstraction but it is strongly recommended to be consistent. If you label all streets with the same label (e.g. only “street”, not “avenue”, “street”, “road” etc. together), you can enumerate them very easily.

The lowest level of map are simple geometric shapes – squares, rectangles, circles and polygons. These don't care about the depth and height (z-coordinates), simply setting them from negative infinity to positive infinity. For 3-dimensional description you can use cuboids or cylinders.

All shapes are internally represented as polygons (with z-coordinate limits), because it simplifies union and intersection operations. An open-source library General Polygon Clipping Library [29] offered implementation for most of our needs. However it lacked some functionality and therefore it was modified and few minor improvements have been added. For example the original version of GPCL has not exposed the intersection interface and it has not contained the polygons collision testing or queries if polygon contains specified point.

Although it would be technically possible, the unions and intersections of basic shapes are not instantiated for the complex areas – these would have a lots of vertices causing performance problems. Complex areas generate few rules in run-time and insert them into the rules engine instead. These rules differ according to type of the complex area and subareas the area contains. If the agent enters some of these subareas these rules propagate the event and cause that the agent is also noted in the superior areas.

You can also test whether is the agent between two areas (e.g. whether he is on some street between two crossings with another streets). After all the map data are loaded the map generates a graph with all areas as vertices and their collisions as edges. There Dijkstra's algorithm can be used to test whether the agent is on shortest path between these two locations. Unfortunately, this forces us to instantiate some higher-level areas. That's not very fast[1] but almost all the work can be done in the preprocessing

1 Intersection of two general polygons with m and n vertices can take up to $O(m*n)$ time.

phase, causing no additional performance looses in run-time.

When we get an information from UT2004 that an agent is located on some position, we would like to determine rapidly, which areas contain specified position. This also requires some preprocessing – a *quad tree*[1] is constructed after loading of data where every node holds all areas that could be in this node's range. The calculation is based on the bounding rectangle of the area.

The node splits in case that it contains more areas than the depth of this node in the quad tree. This guarantees optimal behavior of the data structure regardless of used map scale.

Because Pogamut could operate with multiple games with different sizes of maps, the quad tree has no maximal size – it starts as minimal square for first inserted area, splits if it is appropriate and grows if it cannot contain actually inserted area. It also enables the queries for all areas within specified rectangle.

5.4 Drools support classes

As mentioned in previous section it was necessary to find a way to inject some run-time-generated rules into the rulebase. The Drools API does not support that directly in any simple way, but it can be done. Drools can load the rules from a file or stream anytime, so we only need to generate the stream. The *RulesPackage* class provides methods for comfortable generation of it and subsequent loading into the rulebase.

Comfortable assertion of facts into the working memory and retraction of them also needs attention. Any object that could be in working memory should implement *IFact* interface. The object can either subclass *SimpleFact* class, declare delegate methods with *SimpleFactDelegate* class or just be enveloped with *SimpleFactHolder* object if it is not possible to modify it. All these helper classes implement IFact interface for objects that can be in only one working memory at one moment. Their implementation is synchronized for usage in multi-threaded environment, too.

1 Quad tree is common technique for storage of information about space with different density. If the square contains too much information, we divide it into four half-size squares and split the information into smaller packs.

5.5 Variables

There is a different syntax in the rules when they appear in conditions and in commands. We cannot directly ask the working memory for object satisfying some criteria in the RHS as we could do in the LHS. In plain Drools rules this wouldn't be a problem – we would add one condition putting the object into local variable but in DSL we cannot burden the user to do that – as it does not know the implementation of the rules, this would confuse him.

That is why the Scenario uses a map called *variables* with all the objects in the working memory. The key is either their name (in case it is an instance of the INamed interface) or another dedicated name when we want to store the object as a variable. We cannot change the object's name, of course but there still needs to be a bijection between the object's name and the key under which the object is stored. Therefore it must be proxied by another object with the specific name. However these proxies are not implemented as general because the proxy could require a specific features (for example areas do).

5.6 Changes to GameBots

Several messages between Pogamut's Java side and GameBots with their implementation on both sides have been added. All of them are messages for control connection, because ES controls the scenario from all-knowing position; bots are not supposed to act autonomously.

First set of messages is used to take control over the agents, e.g. when the player gets lost and we want to return him to location, where he's supposed to be:

- SETPLRCTRL sets whether the player can control his movements with keyboard and mouse (making him only passive observer of his character's actions if set to false).
- COMPLR commands the player's character to do some action depending on the message's attributes, e.g. move to some location, turn to specified direction, displace him etc.

- COMBOT is similar to COMPLR but it should command a bot instead of player. It is also more limited, because we can command bots in different ways (as the original Pogamut does).

Another group of messages enhances communication with the player:

- SHOWTEXT displays a short text for specified amount of time.
- PLSND plays some sound.
- SETSENDKEYS sets whether GameBots should send information about every key press to Pogamut.
- KEYPRESS sends the key player actually pressed.
- DLGBEGIN, DLGITEM, DLGEND are used for construction of a dialogue.
- DLGCMD shows, hides or cancels a dialogue.
- DLGREPLY returns information inserted into the dialogue.

These commands are utilizable generally, not only in ES. That's why they have been integrated into GBScenario package and aren't provided separately. Especially the dialogues system provides functionality useful in more projects independent on ES.

On the Java part would be uncomfortable to use these messages directly. Therefore a two modules were designed:

ScenarioAddons project contains platform-independent interfaces for modules using the added messages. These are three modules (as seen in figure 5.1):

- *IPlayerControl* covers the SETPLRCTRL and COMPLR commands
- *IBotControl* covers the COMBOT command
- *IPlayerCommunication* uses SHOWTEXT, PLSND and the key and dialog messages. Key listeners can be registered here automatically switching the SETSENDKEYS to true and false. The dialog system is far more complicated and it is elaborated in following section.

UT2004ScenarioAddons contains the implementation of ScenarioAddons for UT2004.

5.7 Dialogues system

UT2004 has its own GUI system but it is not suitable for our needs. UT2004's GUI assumes, that all the dialogues and menus are known at the time of compilation. It can be expressed in special syntax tempered to its purpose, but it lacks the ability to create the dialogues on the fly.

For Pogamut is such ability essential. The new system was inspired with Java's Swing or SWT where we create objects, define relationships between them, define their properties and register listeners on events.

Here the class names taken from the Java part are used. Each one has its counterpart in the GBScenario (written in UnrealScript). Their functionality is bound so closely, that there is no need to think about them as about two different object. The object is set in Java and after synchronization update call the change reflects in UT2004. User interaction is performed in US and after pressing some active component (see *DialogButton* below) the listener in the Java part is notified. See the figure 5.3 for draft of the principle.

The dialogues system consists of a couple of widgets called components. Objects from the *Dialog* class keep the state information about dialog. The dialog can be shown, hidden and updated (changing components properties, adding new components or removing another ones) several times. It also distributes backward DLGCMD messages.

Every Dialog has one *DialogPanel*. User can define it's size, position on screen and background color. Because we cannot assume specific resolution of screen, all coordinates and sizes are defined as relative with respect to superior component (or whole screen in case it has not any superior component). DialogPanel can have multiple children. It does not arrange them, user is responsible for that, but they compute their absolute coordinates with respect to the parent.

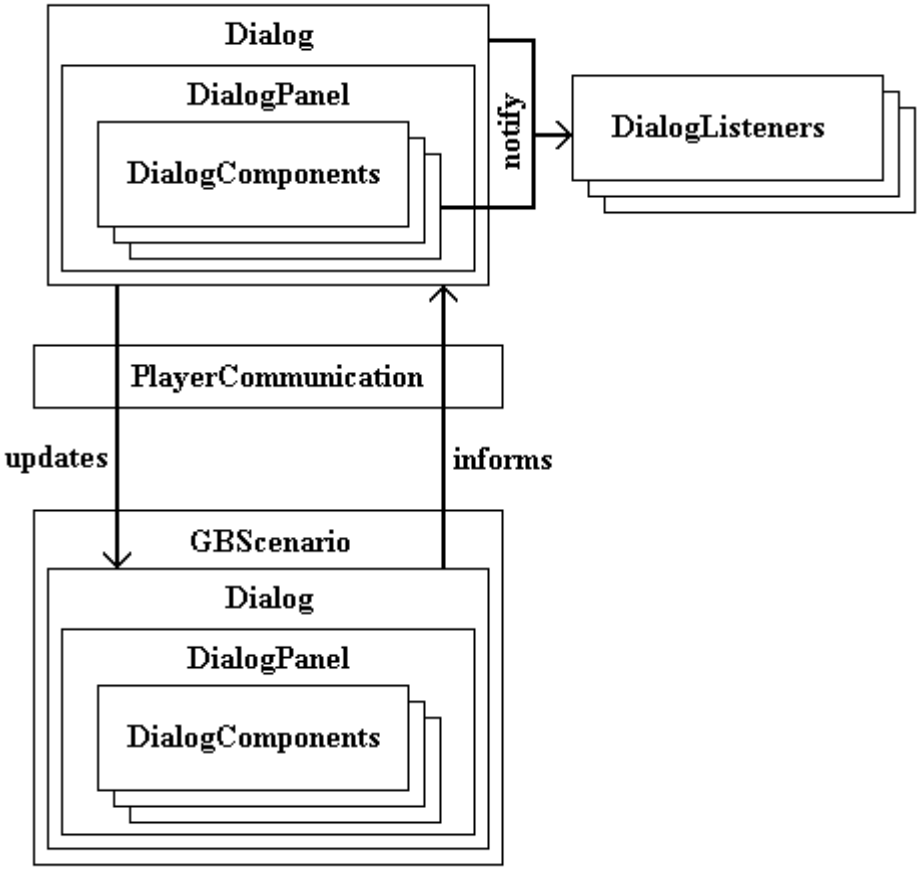


Figure 5.3: The dialogues system

Very important component is *DialogButton*. The button can contain some image or text. It reacts on mouse click – collects information selected in the dialogue and sends it through the DLGREPLY message to the Java side. Here is the message automatically directed to the proper Dialog, where can be registered a dialog-wide listener or listener specific to some active component (DialogButton is not the only one which generates the DLGREPLY message).

In the dialogue we often want to offer a choice among multiple options. *DialogOptionList* is a simple component arranging several *DialogOption*s and determining their behavior – it can allow only single choice (radiobuttons) or multiple ones (checkboxes).

The most complicated component is *DialogLayeredContainer* (DLC). This arranges several images (*DialogImageLayer*), texts (*DialogTextLayer*) or broken lines (*DialogPathLayer*) upon themselves, hiding some of them and

shifting them. These layers can be also displayed only partially – the DLC has a viewport that sets the visible portion of it. This can be used e.g to scroll over a larger image. It is to be noted that the shifting of layers and viewport is specified in pixels, not as relative coordinates. It's necessary cooperate correctly with images displayed in their native resolution.

The DLC is an active component, it sends exact coordinates of the mouse click, displayed size, maximum size and other information. The *DialogText* and *DialogImage* are in fact also DLCs, but they are forced to have only one layer and delegate methods to this layer.

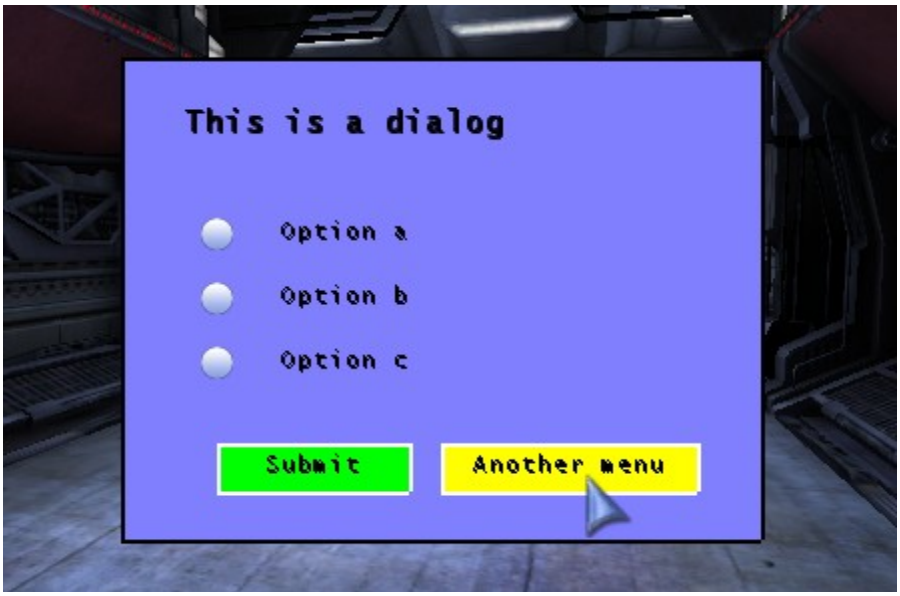


Figure 5.4: Example of a dialog

5.8 UT2004 dependent part of ES

The *IControlServer* in Scenario is implemented by *UT2004ControlServer*. It starts the actual server from PogamutUT2004 and repeatedly asks for the list of agents in the world.

After the message with actual agents and their coordinates comes it creates new players or bots and registers them to the Scenario or removes those who are not mentioned in the message. If there are no agent additions or removals it just updates their position and notifies the Map about the change.

More accurately the UT2004ControlServer does not create players as

objects of the *Player* class but its derivation, *UT2004Player*. The UT2004Player is equipped with two dialogs: *MapDialog* and *CardinalDirectionsDialog*. Both of these are derived from the Dialog class from UT2004ScenarioAddons.

The MapDialog is an image with scroll buttons in the margins which shows enabled marks (paths and points of interest) and agents whose position is known to the player. The CardinalDirectionsDialog is a compass rose with eight directions. User can select the direction and each of them can be highlighted.

The *UT2004Scenario* has very simple function – it just creates the UT2004ControlServer. It also defines the distance that should be considered as “nearby”.

5.9 Map editor

Although XML file format is human readable, it is rather uncomfortable for common user. Especially figuring out the coordinates for objects on the map and their translation into UT map coordinates would be almost insane. The map editor does this in an user friendly way.

Two options were considered. It could either be a standalone application or an Eclipse[1] plug-in. The standalone application has the advantage of being completely independent on user's choice of editing tools but it could be seen as disadvantage. Integration of the editor into the IDE makes the development of ES scenario look more compact, user can edit everything in single program.

The IDE also provides additional support for creating editors in a standardized and well designed way. Therefore the final decision was made for the plug-in.

EduMapEditor, how is the project called, intensively uses classes from the main ES project, *EduScenariosCore*. However, Eclipse's plug-ins do not use traditional Java's class loading but they use the OSGi [30] model. Therefore

1 As stated in footnote 1 in section 4.3 the Eclipse IDE was chosen because Drools already provide a rules development plug-in for this program.

all the projects including original Pogamut were converted into Eclipse plug-ins, too. This meant no change of source code but only adjustments of their configuration[1].

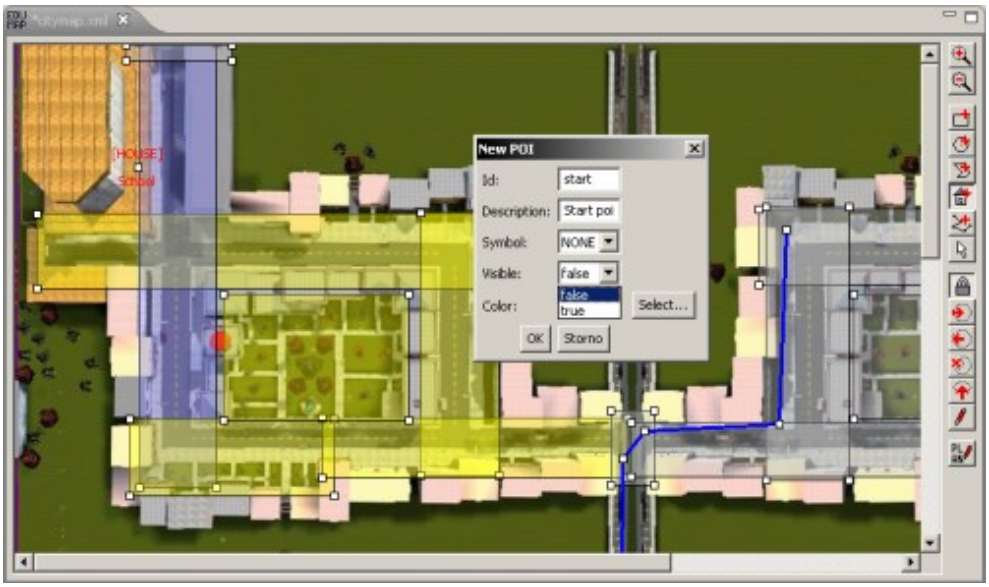


Figure 5.5: Map editor screenshot

ES map editor has these functions:

- generating of the XML file through wizard
- zooming the map to any level of detail
- creation, resizing and shifting of rectangular, circular and polygonal areas
- grouping of simple areas into larger complex areas and their removal or excluding
- insertion, removal and shifting of POIs[2] as well as selection of their

1 There was a problem: Drools plug-in org.drools.eclipse for some reason does not expose its core, compiler and other classes necessary for rules compilation and execution and therefore it cannot be simply included in project's dependencies. Although EduMapEditor does not use Drools it is dependent on the EduScenariosCore project and this depends on Drools. The solution required conversion of the Drools binaries into OSGi bundles. There had a problem arisen – Drools use some Eclipse's internal packages. The standard Java class loading does not care about packages privacy but OSGi does and these internal packages are declared as private (accessible only .from several another packages). Few approaches have been tried as alternation of the Eclipse's org.eclipse.jdt.core plug-in where the problematic packages were located or export of the plug-in into a new one with different privacy policy but none of them worked. The final solution was taken from another program, DA-Launcher [31] which was able to generate all the necessary bundles automatically. So the bundles used in ES are taken from its cache.

2 POI – point of interest, common abbreviation for abstract representation of shops,

symbols and legends

- insertion, removal and shifting of paths (broken lines)
- complete rescaling of the plan

Its detailed manual is provided in the enclosed documentation of ES.

5.10 PuppetBot

In the example scenario the user comes to an another character which represents his/her younger brother. The scenario can spawn arbitrary[1] number of such characters called *puppets*. These bots are not controlled through control connection, their internal logic is adapted to plain execution of commands available through publicly exposed interface. Therefore the scenario can assign orders to the PuppetBot through conventional method calls.

restaurants, cash dispensers, bus station, drug stores and other objects on maps

1 The number is limited only by computational power of the UT2004 server. There can be approximately 6-8 bots spawned at one moment. This could be a major limitation to some scenarios. Unfortunately the only solution for those scenarios is switching to some mass multiplayer virtual environment.

Section 6
Additional discussion

This section discusses the decisions made during the development, their expediences and disadvantages.

6.1 Choice of Drools

There was no previous experience with Drools and the only information about them was from JBoss website and few other articles on the web. From the then point of view it has appeared to be a very good candidate for our purposes. Their DSL and Eclipse-integrated editor tools looked very promising.

But next to the problems mentioned in section 7.1, the overall data-driven system is not the best choice. The event-driven model as used for example in [13] might have been more fitting. Although the event may be simulated by insertion of some new object in the working memory (as it is done with key events) and therefore the data-driven model is more general it is not always better. The conversion of data update to events must be done in DSL what is not the best place for more code. Also the event must be sometimes manually discarded (when there are multiple rules that can react to it).

The event-driven model is even more suitable in larger simulations where the events can be locally specific and they could be automatically discarded if not in player's proximity, saving the computational performance.

All this does not mean that Drools were poorly chosen. Drools can do the job but there could still be a better suited system.

6.2 Process of development

The first step was writing the example scenario in use cases document in plain text exactly describing the scenario. This document is enclosed as Appendix B. The original proposed scenario was not changed.

In the second step was the scenario rewritten into separate rules. Here the

scenario's FSM[1] nature was found and after the rule-groups[2] were found insufficient for this purpose, the Acts were designed. These are described closely in section 5.2.

According to the scenario, the plot should be highly dependent on player's position. It was necessary to develop a system representing the virtual environment in higher abstraction level than the plain coordinates and therefore the map system analyzed in section 5.3 has been implemented.

Before the map could be debugged and tested, some data had been required. The need of map editor was revealed and because it would be a waste of time to create the data manually for this moment the editor was created.

The support of player control the dialogues system has been implemented in the next step. The dialogues system supersedes UT2004 GUI which was found unsuitable There is a plenty of code also both on the Java side and in UnrealScript so this part took a great amount of time. The *MapDialog* and *CardinalDirectionsDialog* were implemented as complicated dialogs requiring special needs.

The scenario uses some simple non-player character. This is called the PuppetBot and its description is in section 5.10.

The direction of the process from concrete needs has led to implementation of only the required parts for the example scenario. Although the code was designed rather general it is possible that for some scenarios it is not completely suitable. Some matters are too complicated to be written in the rules. For example it would be very difficult to create the map dialog as several rules[3] but it is still possible[4].

1 FSM – finite state machine

2 Rule-groups are Drools way how to separate the rules into several parts. At every moment is only one rule-group active and only the rules from this active group can be fired. By default are the rules in group MAIN and this is the one active after startup. The absence of hierarichy in groups made it unusable for our purposes.

3 There are multiple actions that could be performed on the map dialog and each one would require at least one rule to react on this action. The dialog is also automatically updated with agents' positions, that would require another set of rules.

4 This is not a verified fact but a just opinion.

Section 7
Future work

This section is describing additional work that should be done to unfold the full concept of Educational Scenarios. The need of new DSL statements for further scenarios is obvious and so it is not mentioned in the following paragraphs.

7.1 DSL enhancements

Although Drools come with some implementation of DSL it still has some disadvantages. ES use the version 4.0.7 of Drools but even in the new released version 5 the DSL parser does not offer anything stronger than regular language. Such limitation has the advantage of really simple method of writing the conditions and commands where you can only place some kind of slots into the sentences and then use them in the translated code but this is all that you can. The experience had shown that the ability to use nested expressions and recursion would be sometimes very helpful and would reduce the number of very similar rules.

The timer can be used as a simple example: we can set it in seconds or minutes. The version with minutes differs only by multiplication by 60, if we could nest it, there could be only one "timer {name} is over {time}" where the timer could expand. Other example could be the arithmetic – using it in DSL is very uncomfortable, we have to write Java code.

Another difficulty was found when trying to use two identical conditions (with different parameters) in one rule. If the condition defines some Drools variables, they would be defined twice with the same identifier. The compiler will not allow this, of course. The solution is either to use as the variable identifier some parameter or force the user to provide additional unique identification mark for the variables. The first solution is not general and leads to problems when there is an illegal character in the parameter and the second one is not user friendly. This solution had to be used for example in the condition "distance n{x} between {var1} and {var2} is less than {number}" where the Drools variables are $a{x} and $b{x} – the x is

substituted also in the variable name producing variables $a1, $a2 and so on.

Both the problems could be solved by developing a special parser or better, by modification of some existing parser. This could produce the Drools code without the above mentioned obstructions. Nevertheless beyond the new parser the introduction of a new DSL language would raise the need of a new editor supporting this language.

Despite the fact that it is a non-trivial task to write such piece of software, it would be a very useful tool worth of the effort.

7.2 Porting to other virtual environments

Although there are some civil models for UT2004, there are not a much of them and sometimes they have to be developed from users' resources. Therefore a virtual environment with greater supplies of civil models and maps would be appreciated and would help the spreading of ES. Using 3D editor is not a simple task and we cannot expect our target audience to learn how to work with such software. The usage of ES with current amount of civil models is therefore really limited.

Although the 3D modeling is easy nowhere some environments aimed on user-generated content have tools more suitable for beginners. For example SecondLife [32] offers the possibilities of direct in-world editing and moreover loads of already created and scripted objects.

The port of the current version of Educational Scenarios would involve mostly changes in DSL and in the system of dialogues which is platform-dependent. Generally all the packages starting with cz.cuni.amis.pogamut.edu.ut2004 would require reimplementation. These are approximately 15-20% of the Java part of this project. The system of user player control and dialogs implemented in GBScenario (which consumed about 30% time spent on this project) is platform-dependent and it would also need to be reimplemented on the target platform but the design is already made up.

Section 8
Conclusion

Although no non-programmers have tried to realize their ideas about computer-assisted learning in Educational Scenarios, it has been proved that it is possible to use Pogamut with UT2004 as a tool for education in 3D virtual reality environments. The general architecture of the scripting tool was designed and implemented and the example scenario is the evidence of that. There is not command or condition for everything, so the holy grail of content author absolutely independent on programmer is not reached. The usefulness as the number of possible combinations would grow rapidly with increasing number of DSL statements. This allows to claim that the goals of this project as proposed in the submission were successfully accomplished.

References

[1] English tests. 31 October 2006. Lingo4you GbR. 2 August 2009. <http://www.ego4u.com/en/cram-up/tests>

[2] Virtual Labs. 22 May 2003. Howard Huges Medical Institute. 27 June 2009. <http://www.hhmi.org/biointeractive/vlabs/index.html>

[3] Serious Games Initiative. 19 October 2008. Woodrow Wilson International Center for Scholars. 26 March 2009 <http://www.seriousgames.org>

[4] Pogamut project documentation. 20 June 2009. Charles University in Prague. 11 July 2009. <http://artemis.ms.mff.cuni.cz/pogamut>

[5] Emotional AI in UT project website. 8 October 2008. Charles University in Prague. 11 June 2009. <http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Emotional+AI+in+UT>

[6] ACT-R: Theory and Architecture of Cognition. 26 March 2009. Carnegie Mellon University. 26 March 2009. <http://act-r.psy.cmu.edu/>

[7] PojACT-R project website. 29 May 2009. Charles University in Prague. 11 July 2009. <http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=PojACT-R>

[8] Unreal Tournament.1 August 2006. Epic Games, Inc. 26 March 2009. <http://www.unrealtournament2003.com/ut2004/index.html>

[9] Drools. 15 January 2008. JBoss Community. 26 March 2009. <http://downloads.jboss.com/drools/docs/4.0.4.17825.GA/html_single/index.html>

[10] FearNot! software homepage. 26 April 2006. eCIRCUS. 28 June 2009. <http://www.e-circus.org>

[11] Secure Game. 6 July 2009. PEDAGAMES project. 6 July 2009. <http://www.snap.fi/services/pedagames/www/pelit.php?peli=secure>

[12] Hämäläinen R., Mannila B., Oksanen K., Koutaniemi L. (2006): Secure: Scripted 3D-Game Environment in Vocational Learning, *ECER 2006*, Geneva, Switzerland, September

[13] ScriptEase - A Scripting Language for Computer Role-Playing Games. 1 November 2008. University of Alberta. 6 July 2009. <http://www.cs.ualberta.ca/~script/>

[14] Carbonaroa M., Cutumisub M., Duff H., Gillis S., Onuczkob C., Schaeffer J., Schumacher A., Siegel J., Szafronb D., Waughb K. (2006): Adapting a Commercial Role-Playing Game for Educational Computer Game Production, *GameOn North America*, Monterey, USA, September 2006

[15] Neverwinter Nights. 6 July 2006. Bioware Corp. 6 July 2006. <http://nwn.bioware.com/>

[16] Volbracht S., Dimik G., Backe-Neuwald D., Rinkens H-D. (1998): The 'City Game' An Example of a Virtual Environment for Teaching Spatial Navigation, *Journal of Universal Computer Science*, 1998, **4**, 461-465

[17] Virtual Reality Modeling Language. 11 July 2009. Wikimedia Foundation, Inc. 13 July 2009. <http://en.wikipedia.org/wiki/VRML>

[18] ADMS webpage. 14 February 2008. Environmental Tectonics Corporation. 28 June 2009. <http://www.admstraining.com>

[19] Production system. 4 April 2009. Wikimedia Foundation, Inc. 2 August 2009. <http://en.wikipedia.org/wiki/Production_system>

[20] Wile D. S. (2001): Supporting the DSL Spectrum, *Journal of Computing and Information Technology - CIT 9*, 2001, **4**, 263–287

[21] Domain specific languages in Drools. 15 January 2008. JBoss Community. 26 March 2009. <http://downloads.jboss.com/drools/docs/4.0.4.17825.GA/html_single/index.html#d0e4177>

[22] Zhu L., Morgan G. (2008): Runtime Evolution for Online Gaming: A Case Study using JBoss and Drools, *GDTW 2008, Sixth International Conference in Game Design and Technology*, Liverpool, UK, November 2008

[23] ILOG JRules 6.5 brings rules to SOA. 2 August 2007. Núñez S. 26 March 2009. <http://www.infoworld.com/article/07/08/02/31TCjrules_1.html>

[24] Microsoft's Rule Engine Scalability Results - A comparison with Jess and Drools. 16 September 2005. Young, Ch. 26 March 2009. <http://geekswithblogs.net/cyoung/articles/54022.aspx>

[25] A* search algorithm. 25 March 2009. Wikimedia Foundation, Inc. 27 March 2009. <http://en.wikipedia.org/wiki/A*_search_algorithm>

[26] Floyd-Warshall algorithm. 25 March 2009. Wikimedia Foundation, Inc. 27 March 2009. <http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm>

[27] UnrealScript Language Reference. 21 December 1998. Tim Sweeney, Epic MegaGames, Inc. 28 March 2009. <http://unreal.epicgames.com/UnrealScript.htm>

[28] Unreal wiki. 26 March 2009. community project 28 March 2009. <http://www.unrealwiki.com>

[29] General Polygon Clipping library Java port. 16 January 2004. Solution Engineering, Inc. 29 March 2009. <http://www.seisw.com/GPCJ/GPCJ.html>

[30] OSGi Alliance webpage. 28 June 2009. OSGi Alliance. 28 June 2009.

[31] DA-Launcher website. 6 July 2009. DynamicJava.org. 6 July 2009. <http://www.dynamicjava.org/projects/da-launcher>

[32] SecondLife website. 11 July 2009. Linden Research, Inc. 11 July 2009 <http://www.secondlife.com>

Appendices

Appendix A: Simple scenario

Here is an example of very simple scenario as plain text and its transcription into DSL rules and extract from configuration and map file.

Mission: Go from point A to point B

Event:

Fired by: Scenario start

Message: "Go from point A to point B. These points are shown in the map. Press M key to hide the map."

Effect: The map with current position and point A and B shows.

Event:

Fired by: Player's arrival to the proximity of point B

Message: "Great, you are here."

Effect: The scenario ends.

Such simple scenario has in fact two acts: the first is the briefing when the player is looking into the map and in the second he is moving towards point B. This is why the *scenes* section of configuration file would look like

```
<scenes>
    <act id="part">
        <act id="briefing"></act>
        <act id="movement"></act>
    </act>
</scenes>
```

The contents of the map file could be very simple in this case, it could look for example as:

```
<POI x="100" y="200" z="0" id="A" symbol="CIRCLE"
    description="Point A" color="green"
    visible="true"></POI>
<POI y="300" y="400" z="0" id="B" symbol="STAR"
    description="Point B" color="red"
    visible="true"></POI>
```

The most important part are the rules:

```
rule "Scenario start"
    when
        scenario is ready
    then
        start scenario
        transport player "Player1" to "A"
        perform act "briefing"
end

rule "Briefing"
    when
        performing act "briefing"
        map is hidden
    then
        show message "Go from point A to point B.
These points are shown in the map. Press M key to
hide the map." for 15 seconds
        show map
end

rule "Map hiding"
    when
        performing act "briefing"
        user pressed "M" key
    then
        hide map
        set keypress "M" processed
        perform act "movement"
end

rule "At B"
    when
        performing act "movement"
        player is nearby "B"
    then
        show message "Great, you are here."
        finish scenario
end
```

The acts are not absolutely necessary here because the map dialog itself could keep the state information but the approach presented here is more general.

Appendix B: Example scenario description

Mission: Return home from school and pick your younger sibling up from kindergarten hereat.

Event:

Fired by: Scenario start

Message: "Go from your school to the kindergarten. Mind the advices for direction, where you should go. Press M key when ready."

Effect: The map with school and kindergarten shows.

In the first part is the player navigated by basic commands - turn left, go ahead, turn right...

Possibilities:

- the player goes correctly
- the player disobeys the navigation and avoids localy allowed area

Effect: Warning with message, which way to return. One point penalty. Timer is started.

Possibilities:

- The player returns to allowed area within 30 seconds
- The player doesn't return within 30 seconds

Effect: Warning "You have got lost". Three points penalty. System takes over the control and returns the actor to last correct location.

Event:

Fired by: Second part of route from school to kindergarten.

Message: "Now you are going to orient yourself according to cardinal directions. Look on the map and try to orient yourself - which way do you think you look currently? Click on the arrow pointing thitherward."

Effect: Map with 8 arrows from current position, the player should select one of them.

Possibilities:

- The player selects the correct arrow

Message: "Great, you have selected the right direction!"

- The player selects an adjacent arrow

  Message: "You have made a little mistake, in fact you are looking to this direction."

  Effect: One point penalty, highlighting of the correct arrow

- The player selects incorrect arrow

  Message: "You haven't oriented yourself correctly, in fact you are looking to this direction."

  Effect: Two points penalty, highlighting of the correct arrow

Message 2: "From now you can press the M key and look on the map anytime. Another keypress hides the map again."

In the second part the player is navigated by cardinal directions - north, south, east, west and their combinations.

Possibilities: The same as in first part

Event:

Fired by: The player arrives to the kindergarten.

Message: "Great, your brother already awaits you. Now you can return home together. Pay attention to him!"

Event: Map with school, kindergarten, home and highlighted path between kindergarten and home.

Note: The brother will follow the player. He will never run and when the player goes far than approx. 5 meters, he will stand and start to cry (warning will be displayed), until the player returns. If the player doesn't return to the bot within 15 seconds, he gets a one point penalty. Each another 15 seconds delay will result in one point penalty.

Possibilities:

- The player goes correctly
- The player swings out from the path - warning as in first part.

Event:

Fired by: In one third of way home.

Message: "Your mum calls that you should bring some bread. Go to the shop and buy it."

Effect: Map with kindergarten, home, shop and path from current position to the shop.

Event:

Fired by: somewhere on the way to the shop

Message: "Here is a icecream stall. Your brother would like chocolate ice cream. What will you do?"

Dialog:

- "You'll buy one for him and one for yourself"
- "You'll buy him the icecream."
- "You'll buy nothing."

Effect: The brother will cry all the way to the shop (ambient sound)

Event:

Fired by: Arrival to the shop

Possibilities:

- If he bought some icecream

  Message: "You have arrived to the shop. Unfortunately you haven't enought money for the bread. what will you do?"

  Dialog:

  - "You'll buy as much rolls as you can."
  - "You'll buy candy, when you're already here."
  - "You won't buy anything."

- If he didn't buy anything before

  Message: "You have arrived to the shop. What would you like to buy?"

  Dialog:

- "You'll buy only the bread."
- "You'll buy the bread and some other candies."
- "You'll not buy anything."

Message 2: "You can go home now. I'll not advise you anymore, you have to find your own way."

Event:

Fired by: After two minutes going home

Message: "Do you know, where you are now? Try to mark the point on the map."

Effect: Map with possibility of selection of the place. Penalty points according to mistake.

Message 2: "In fact you're here. Keep on your way home."

Event:

Fired by: Arrival home (with brother)

Message: "Great, you have returned home!"

Effect: The scenario is completed.

Appendix C: Contents of the enclosed CD

The enclosed CD contains these folders and files:

- **documentation/**: End-user documentation of the Educational Scenarios.
- **source/**: Source code of all projects.
- **eclipse-rcp-europa-winter-win32.zip**: the archive with Eclipse IDE
- **es-setup.exe**: the installer of Educational Scenarios
- **readme.txt**: installation instructions
- **thesis.pdf**: the electronic version of this document.
- **examplescenario.avi**: a video with example scenario's record

A licensed version of Unreal Tournament 2004 is necessary for running of the program (not enclosed).