Pogamut 3 – Virtual Humans Made Simple

Jakub Gemrot, Cyril Brom, Rudolf Kadlec, Michal Bída, Ondřej Burkert,
Michal Zemčák, Radek Píbil, Tomáš Plch

Charles University in Prague,
Faculty of Mathematics and Physics,
Department of Software and Computer Science Education.
Prague, Czech Republic.
jakub.gemrot@gmail.com, brom@ksvi.mff.cuni.cz
http://artemis.ms.mff.cuni.cz

Abstract

Virtual humans, or intelligent virtual agents, born from the mould of academic, industrial, and science-fiction communities, have been surrounding us for more then a decade. Theoretical knowledge allowing many practitioners and enthusiasts to build their own agents is now available. However, the initial learning curve imposed by this specialized knowledge is quite steep. Even more fundamentally, there is a lack of freely available software assisting in the actual building of these agents. In this chapter, we introduce the Pogamut 3 toolkit. Pogamut 3 is a freeware software platform for building the behaviour of virtual humans embodied in a 3D virtual world. It is designed for two purposes; first, to help newcomers to build their first virtual agents; second, to support them as they advance in their understanding of the topic. On the one hand, Pogamut 3 allows for rapid building of simple reactive agents, and on the other hand, it facilitates development of challenging agents exploiting advanced artificial intelligence techniques. Pogamut 3 is primarily tailored to the environment of the Unreal Tournament 2004 videogame, but it can be connected to other virtual world as well. It is suitable both for research and educational projects.

1. Introduction

People are fascinated by the idea of artificially created man. Who would not know Shelley's Frankenstein (1818), Jewish golems (e.g. Meyrink, 1915), or Čapek's Robots (1921)? Last century allowed some of these ideas to begin materializing. Undoubtedly, two most notable kinds of artifacts that can be connected with those fantasies are software programs with some, or mimicking some, human-level cognitive abilities, such as chess-playing programs, and robots. Recently, yet another kind of artifacts emerged**:** *intelligent virtual agents* inhabiting *virtual reality*, or, in other words, *virtual humans*.

Virtual humans are a kind of software agents (Wooldridge, 2002) inhabiting virtual worlds. Conceptually, they are typically conceived as consisting of two parts that are intervened with each other – a *virtual body* and a *virtual mind*. The virtual body is subject to the laws of the virtual world (e. g. gravity) and has predefined sensory-motoric capabilities. These capabilities are used by the virtual mind to control the body: the mind is embedded in full sensory-motor loop.

A virtual world is a relatively ecologically plausible model of the real world, it provides a semi-continuous stream of rich sensory data, e.g. at 15 Hz, as well as a complex environment for acting. At a first glance, this is similar to how robots are embodied. However, at a second look, several distinctions become apparent: going from the robotic world to the virtual one brings one closer to reality in one sense at the cost of losing some detail in another – this is a trade-off. In a robotic platform, one works with low-level, many would say relatively plausible, environmental input; however, the whole environment is typically small and artificial (e.g. a simple arena with a couple obstacles or a sloping pavement). Even though several attempts to use robots in real environments

have emerged recently (consider e.g. the Darpa challenge, Darpa 2009), robots in these experiments still act in relatively limited domains. In virtual worlds, the environmental structure - the walls, objects, and their positions - is typically represented explicitly, hence its extraction for the purpose of a virtual human is much simpler that in robotics. This is implausible, but consequently, the virtual worlds can be much larger and more complicated than environments of robots. Still, in a typical 3D world, it is possible to use sub-symbolic inputs, such us ray-casting, which is a mechanism through which a virtual human can see the underlying geometry of the virtual world similarly to how rats use their whiskers.

When it comes to the developmental cost, the virtual humans are relatively cheap comparing to robots. However, the development of a virtual world and its simulator is quite expensive - contrary to most robotic environments.

Virtual humans, similarly to robots, are now used in many industrial applications. Besides, researchers from various fields ranging from neurobiology to enactive artificial intelligence use robots as test-beds for investigating and refining their ideas concerning various cognitive and/or motor skills, producing detailed computational models. Virtual humans are not widely used for this purpose yet, except for artificial intelligence for videogames; however, this technology is now mature enough to be exploited in this way (Brom & Lukavský, 2008; Jilk et al., 2008). Arguably, it may be beneficial to investigate an abstract, system-level model, e.g. of localization and navigation, episodic and spatial memory, decision making or communication, using a virtual human rather than a robot. Some have also argued that virtual humans in videogames are a befitting test-bed for tackling the "grand goal of AI" to build general human-level AI system (Laird & van Lent, 2001).

However, what a newcomer to the field, be it a researcher, a teacher, a student, or just an enthusiast, should do? Where to start? The problem is that the initial learning curve is steep and, presently, there is a lack of systematic support of education of newcomers in the intelligent virtual agents community (Brom et al., 2008), probably because the discipline is younger than robotics. For example, one cannot buy a virtual human like an e-puck robot.

This chapter concerns itself with our work addressing this educational issue, aiming at helping to bring virtual humans to a larger audience. The topic of this chapter is the Pogamut 3 toolkit we have been developing for last three years, a freeware software platform for building behaviour of intelligent virtual agents embodied in a 3D virtual world. Pogamut 3 was intentionally created to facilitate beginning with IVAs, both for research and educational purposes. Additionally, Pogamut 3 can also support its users when they advance in their understanding of the topic. On the one hand, it allows for a rapid building of simple reactive virtual agents, on the other hand, it facilitates development of challenging agents exploiting advanced techniques such as neural networks, evolutionary programming, fuzzy logic, or planning.

Pogamut 3 is primarily tailored to the 3D virtual environment of the videogame Unreal Tournament 2004, but it can be connected to other virtual worlds as well. Capitalizing on its flexible architecture, Pogamut 3 integrates: 1) an integrated development environment (IDE) with a support for debugging; 2) a library of sensory-motoric primitives, path-finding algorithms, and reactive planning techniques for controlling virtual humans; and 3) tools for defining and running experiments. The platform documentations include several tutorials, videos and example projects. Support from an on-line forum is also provided.

Presently, the platform is most suitable for investigating control mechanisms of virtual humans for videogames; however, it can be used by anyone interested in setting up experiments utilising up to 10 virtual humans inhabiting a middle sized virtual world (e.g. 500 m^2). Pogamut 3 can be also

used as a tool, in which university and high-school students can practice their skills of building virtual agents (Brom et al., 2008).

Presently, the platform is not suitable for simulations of large crowds (e.g. Badler et al., 2008). Also the graphical part and natural language processing have been out of our scope (e.g. there is only limited support for facial expressions and conversations).

We are now extending the platform with features that will facilitate research and education in the domains of virtual storytelling and computational cognitive psychology. The most notable extension we are currently working on are:

1. seamless integration of the general cognitive architecture ACT-R (Anderson, 2007),
2. a tool allowing for the control of gestures,
3. a generic story manager,
4. a generic module enhancing characters with emotions,
5. a generic episodic memory module,
6. a connection of the Pogamut 3 to another 3D virtual environment.

The next part of the chapter will briefly overview intelligent virtual agents: their application domains, most notable issues already addressed by the community, and a few tools complementing Pogamut that are available. Pogamut 3 has two faces: it can be viewed from the perspective of those who want to build control mechanisms of virtual humans (for whatever purpose) but also from the standpoint of programmers willing to modify parts of the platform itself and/or extend it. After a brief introduction to Pogamut 3, this chapter will continue describing how to use the platform from the perspective of a virtual human developer. Then, the platform will be described from the point of view of a programmer. Finally, extensions of Pogamut 3, our current work in progress, will be reviewed. The most up-to-date information concerning this work can be found at the project's web page (Pogamut, 2009), where Pogamut 3 can be also downloaded. Teachers considering utilization of the platform in their classes are recommended to consult (Brom et al., 2008; Brom, 2009).

2. Intelligent virtual agents

The notion of *intelligent virtual agents* (IVAs) began to emerge about two decades ago. Today, most IVAs are capable of real-time perception and autonomous acting in a *graphical virtual environment* (VE). These environments are predominantly 3D, even though 2D or 2½D worlds[1] are sometimes used as well, predominantly for prototyping purposes. IVAs imitate a range of human-like or, less frequently, animal-like qualities. They often respond emotionally and possess various high-level cognitive abilities, such as communication skills, social awareness, learning, or episodic memory (Champandard, 2003; Brom & Lukavský, 2008; IVA, 2009). Frequently, VEs inhabited by IVAs are dynamic and unpredictable similarly to the real world. The VEs allow for some degree of interaction among agents and/or agents and humans.

During last years, IVAs expanded to many domains, including commercial and serious videogames (e.g. Aylett et al., 2005a), virtual tutoring systems (Woolf, 2008), therapeutic applications (Hodges et al., 2001), virtual storytelling (Cavazza et al., 2004), cultural heritage applications (Magnenat-Thalmann & Papagiannakis, 2006), the movie industry (SoftImage, 2009), the cognitive science research (Burgess, 2002), the military (Johnson et al., 2004), the medical training (Pulse!!, 2005-9), virtual companions applications (LIREC, 2008; Companions,

1 2½ worlds feature two full dimensions and one reduced dimension. For instance, depth planes of many arcade games represent such a reduced dimension.

2006), and industrial applications (Badler et al., 2002; see also Prendinger & Ishizuka, 2004; Badler et al., 2008; Magnenat-Thalmann & Thalmann, 2004). Building of IVAs is very hard. Besides graphical issues, which are a world of its own, there is the cognitive and behavioural side of the problem. Consider IVAs from FearNot! (eCircus, 2009; Fig. 1), which is one of the largest non-military serious game featuring IVAs delivered by the academic community to date. FearNot! is an anti-bullying game targeted at primary school pupils and its IVAs represent children of the same age. These IVAs posses many abilities. At the first place, they are able to perceive their surrounding and they can execute actions that change the environment. They are able to navigate in the environment and plan their actions with respect to goals they have (Aylett et al., 2006). They can interact with each other and they can communicate with a child using a text-based natural language interface. They have personalities, express emotions (Aylett et al., 2005b), and feature autobiographic memory (Dias et al., 2007).

Figure 1 - Screenshot from FearNot! – in the course of a bullying episode. Adopted from (eCircus, 2009).

The development of these agents took many man-years. However, the project (and many similar projects) accumulated a large amount of knowledge of how to build agents and with appropriate authoring tools, the development time can be substantially reduced. This would allow more people to start their own projects. Even though such tools are not readily available yet, at least not for public use, it seems that the research community started to reflect their necessity. For example, in virtual storytelling, a discipline neighbouring to the field of IVAs and overlapping with it partially, Pizzi and Cavazza (2008) put it: "Recent progress in Interactive Storytelling has been mostly based on the development of proof-of-concept prototypes, whilst the actual production process for interactive narratives largely remains to be invented. Central to this effort is the concept of authoring...". These authors also review activities focused on development of authoring tools for that domain. Authoring tools have been built also for development of videogame agents (reviewed in Kadlec et al., 2007). The discipline of cognitive science modelling is represented by the CAGE toolkit (CAGE, 2007). There is also Netlogo (Wilensky, 1999), an excellent entry-level tool for building simple agents and running social simulations, and tools for learning programming by means of virtual agents, such as Alice (Alice, 1995 – 2009). Also several toolkits for developing Machinimas are available, e.g. Movie Sand BOX (Kirchner, 2009)[2].

2 Machinimas are movies generated using real-time 3D, typically videogame, engines.

None of these applications presents a tool allowing for building agents of the FearNot! level of complexity in a reasonable time. Neither Pogamut 3 does. Nevertheless, many of them, including Pogamut 3, have made important steps towards this ideal.

While Pogamut 3 directly capitalises on the knowledge gained by many researchers during development of previous videogame tools and it outperforms these tools in many respects (Kadlec et al., 2007), it is complementary to other tools, not a rival. We depart from the other tools in several ways.

1. We explicitly focus on the support of newcomers, a feature only Alice and Netlogo and perhaps some Machinima tools share with Pogamut 3. However, these projects tend to support development of agents only by scripting; that is, the agents' behaviour need to be programmed step-by-step. Instead, in Pogamut 3, one can develop agents using programming paradigms befitting for controlling agents in dynamic environments better than scripting, such as reactive planning. Pogamut 3 even allows one to equip agents with cognitively plausible "minds." On the other hand, computer graphic issues are deeply addressed in many Machinima tools as opposed to Pogamut 3, Netlogo can run tens of simple agents, hence it can be exploited in agent-based ethological modelling (e.g. Kokko, 2007, ch. 8; Bryson et al., 2007), and Alice offers better support than Pogamut 3 for general teaching of programming.
2. We explicitly focus on simplifying the process of development of IVAs high-level behaviour. In this context, high-level behaviour is typically conceived as human-like behaviour requiring high-level cognitive abilities and lasting minutes or tens of seconds, as opposed to low-level behaviour being more akin to motor behaviour with a shorter time span. For example, high-level behaviour is finding an object in a house, which includes recalling where the object is and going for it while possibly avoiding obstacles. Low-level behaviour is picking the object up, which includes many animation issues such us computing the trajectory of the hand reaching for the object. Conversational characters are also out of the scope of Pogamut 3. In our focus on high-level behaviour as opposed to low-level behaviour and conversational behaviour, we complement many projects stemming from the fields of computer graphics, including Alice and Machinima tools, and virtual storytelling, such as Scenjo (Spierling et al., 2006).
3. Time spans of simulations that can be run in Pogamut 3 are not longer than hours or possibly days of simulation time. Essentially, Pogamut 3 employs abstraction of the human world. These two features – the time span and the ground level abstraction – are shared by many other toolkits. In fact, most of them enables only running even shorter simulations. However, due to its design philosophy, simulations in Netlogo can employ different levels of abstractions and consequently simulate much longer intervals. For example, one can run a population dynamic model in Netlogo.
4. Pogamut 3 is not a side product of a research prototype as is the case for many other authoring tools. We directly focus on the development of our toolkit, hence it is well build and tested. Actually, some published authoring tools have never been released for public.

3. Pogamut 3 introduction

The Pogamut platform has already undergone a few major upgrades that completely changed the way a developer uses it and interacts with it. These updates reflected our major goals outlined above. Still, all versions of Pogamut employ the same underlying architecture. This architecture features four components: 1) a simulator of the virtual world, 2) a graphical user interface (GUI) for IVAs development, 3) a library of base classes for coding of IVAs behaviour, including IVAs'

senso-motoric primitives, 4) a middleware handling communication between (1) and (3). Roughly, the design process of developers building their IVAs is as follows. First, a developer will create an IVA, either using an external editor or in the GUI directly. In either case, he or she will employ the code from the Pogamut's library. Second, the developer will use the GUI as a launcher for running the IVA in the virtual world and for its debugging and testing.

The main scope of this chapter is Pogamut 3, the third major release of Pogamut. This section briefly reviews its predecessors, Pogamut 1 and 2, and then overviews the architecture of Pogamut 3.

3.1 History of the Pogamut platform

The first version of the Pogamut platform, Pogamut 1, worked with the Unreal Tournament 1999 videogame (Epic, 1999) and it was designed as a Python library with a simple agent launcher GUI, which had only limited debugging capabilities (Bída et al., 2006). Its successor, Pogamut 2 (Kadlec et al., 2007), is based on Java and its GUI is a full-fledged integrated development environment (IDE) developed as a NetBeans plugin. Pogamut 2 also moved to more advanced Unreal Tournament 2004 (UT2004) (Epic, 2004). As opposed to Pogamut 1, the new version has been extensively evaluated both by our team (Kadlec, 2008; Štolba, 2008; Brom et al., 2008) as well as by others (e.g. Small, 2008; Tence & Buche, 2008). It was also picked as a base for competition 2K BotPrize at Australia[3], where participants were challenged to develop an agent that can fool the panel of judges that it is a human.

It was found that the main drawbacks of Pogamut 2 were: 1) It was tightly bound to the world of the UT2004, which is a violent action game, thus limiting the field of the research to the gaming AI. Indeed, the users often brought up the topic whether the Pogamut platform could support more virtual environments (Brom et al., 2008). 2) The lack of documentation of the UnrealEngine, the engine of the UT2004, together with the multi-threaded design of the core of Pogamut 2 library brought an unnecessary burden for the developer. These two points led us to a decision to completely remake Pogamut 2 to simplify its core library and to make it a general platform for the development of virtual cognitive agents instead of just UT2004 agents.

3.2 Terminology

For historical reasons, we will use the terms *agent* and *bot* interchangeably for IVAs developed in Pogamut 3, as the bot is a term frequently used in the context of videogames. Importantly, IVAs community predominantly employs the mind-body dualism as a design metaphor. That is, IVAs are conceived as having two components: the mind, which is a set of data structures and algorithms responsible for controlling the IVA, and the body, which is a set of data structures and algorithms for graphical portrayal of the IVA's body. For the former, we will use the term a *bot's logic* or a *bot's mind*. For the latter, we will employ a frequently used term *avatar*. We will also use the following two terms interchangeably to refer to environments where IVAs reside: a *virtual world* and a *virtual environment*.

3.3 Pogamut 3 overview

The Pogamut 3 new heart lies in the *GaviaLib* (General Autonomous Virtual Intelligent Agents Library). The GaviaLib (see Fig. 2) is designed as a middleware that mediates communication between the virtual environment (left side of the figure) and the bot's logic (right side of the figure). The Pogamut 3 platform still features UT2004 integration that makes the reference implementation of the GaviaLib bindings to a specific virtual world (see Fig. 2). However,

3 http://botprize.org [27.1.2009], 2K BotPrize competition

Pogamut's integration with different environments is now possible, as described in Sec. 5 and 6 in more detail.

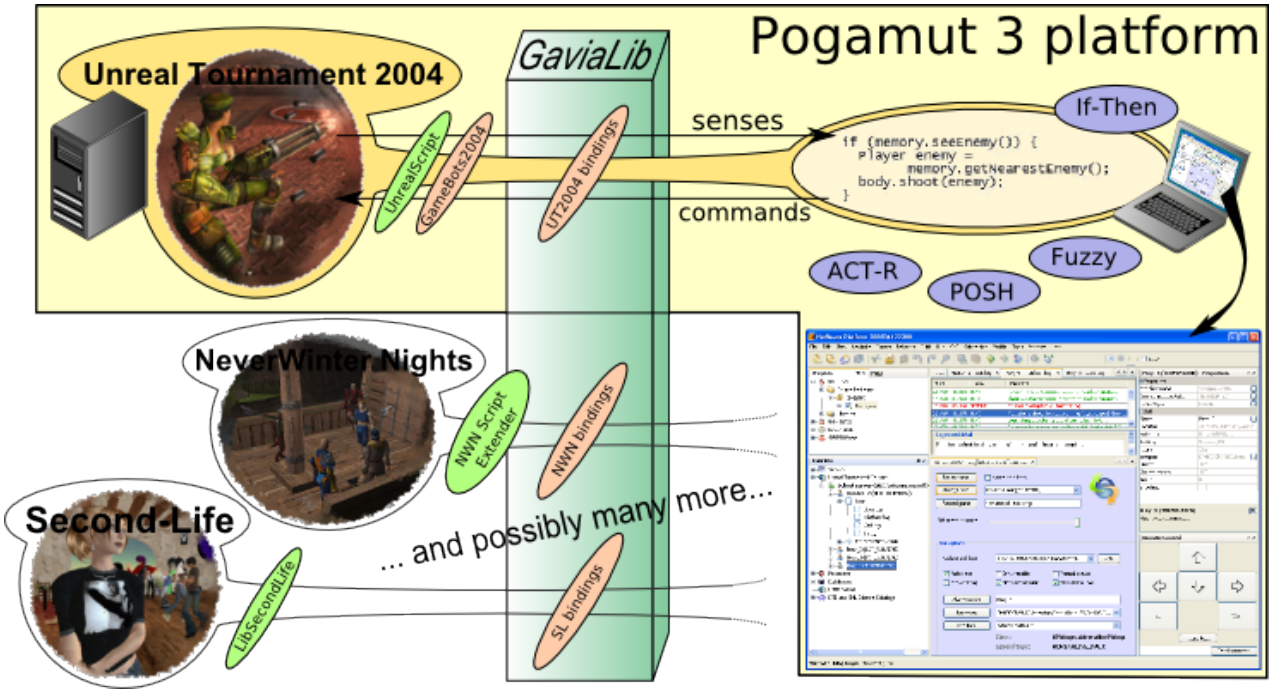

Figure 2[4] – Pogamut 3 platform overview picturing the generic GaviaLib library that acts as a mediator between the virtual world and the agent.

The communication pipeline from the virtual world to a bot's logic (transmitting virtual world events) and vice versa (transmitting bot's commands) can be easily reconfigured by a programmer willing to connect GaviaLib to another virtual world or to extend the communication protocol with UT2004. At the same time, this pipeline is disguised by a simple high-level API (application programming interface) from the eyes of regular IVA developers. On the one hand, this limits the developers in using only given virtual environments, on the other hand, the API allows them to access bot's senso-motoric primitives easily, so the developers only write what the bot should do and when and need not care about how the low-level communication really works. The GaviaLib interface also allows for controlling the bot's avatar remotely across the network. The GavialLib may be also utilised by various auxiliary modules such as the reactive planner POSH (Bryson, 2001a), which helps with development of a bot's logic, or an emotion module.

The next section will introduce Pogamut 3 from the perspective of a regular IVA developer, that is, refraining from low-level implementation details. After reading this part, the reader should have a basic idea about building of agents for UT2004 in Pogamut 3. Then, technical aspects of Pogamut 3 will be described for programmers willing to dive below the high-level GaviaLib API to exploit the full functionality of GaviaLib.

4 Using Pogamut 3 - Pogamut Netbeans Plugin

Present-day software developers expect not only a runtime support in a form of libraries of base classes with a code they can utilise to obtain the required functionality, they also expect a design-

4 The figure consists of screenshots from games Unreal Tournament 2004, NeverWinter Nights and Second Life copyrighted by Epic Games, Inc., BioWare, Inc. and Linden Lab respectively.

time support by means of an integrated development environment. This applies for a web building, database development etc. and there is no reason why development of IVAs should be any different. Pogamut 3 offers this functionality in a form of a plugin for NetBeans, a generic freeware Java IDE. The plugin provides (see Fig. 3):

- Empty bot template projects – make it easy for new users to start coding their first bot.
- Example bots – introduce various bot programming paradigms; one can derive own bots from these examples easily.
- List of UT04 servers – shows all servers registered in the IDE and their current status (e.g. RUNNING/PAUSED/DOWN etc.) (Fig. 3, Window 4).
- List of running bots – helps to keep a track of all running bots, shows their current status (e.g. RUNNIGN/PAUSED/EXCEPTION/FINISHED etc.) (Fig. 3, Window 4).
- Introspection of bot's variables – assists a developer in observing and changing parameters and variables of the agent during runtime (Fig. 7).
- Bot's properties – give a quick access to variables common for all agents, e.g. position, velocity, orientation, health (Fig. 3, Window 3).
- Log viewers – show incoming and outgoing messages together with important decisions made by the bot's logic (Fig. 3, Window 5).
- Bot remote control – allows a developer to move the bot manually to any desired location and to send it arbitrary commands using a text field (Fig. 3, Window 2).
- Server control – allows a developer to pause/resume the server, change the map remotely, save replays, change game speed etc. (Fig. 3, Window 6).

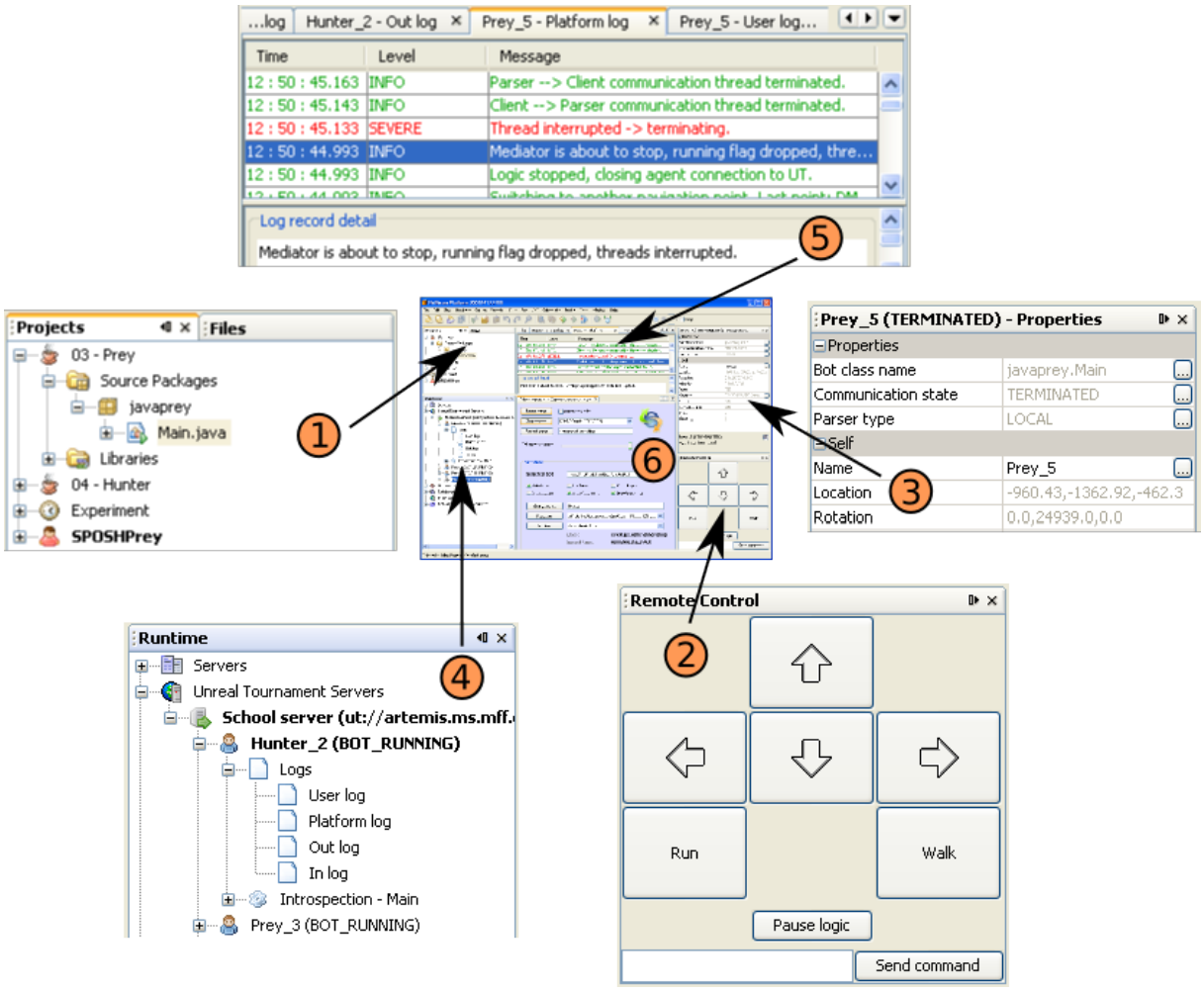

Figure 3 – Pogamut 3 NetBeans plugin (6) and its parts (1-5). The parts are described in the text.

These features were purposely designed to provide support during the agent development cycle, which can be conceived as having five stages: (1) inventing the agent, (2) implementing the agent, (3) debugging the implemented agent, (4) tuning the parameters of the agent, and (5) validating the agent through series of experiments (see Fig. 4). The rest of this section illustrates how exactly Pogamut 3 supports the latter four of these stages.

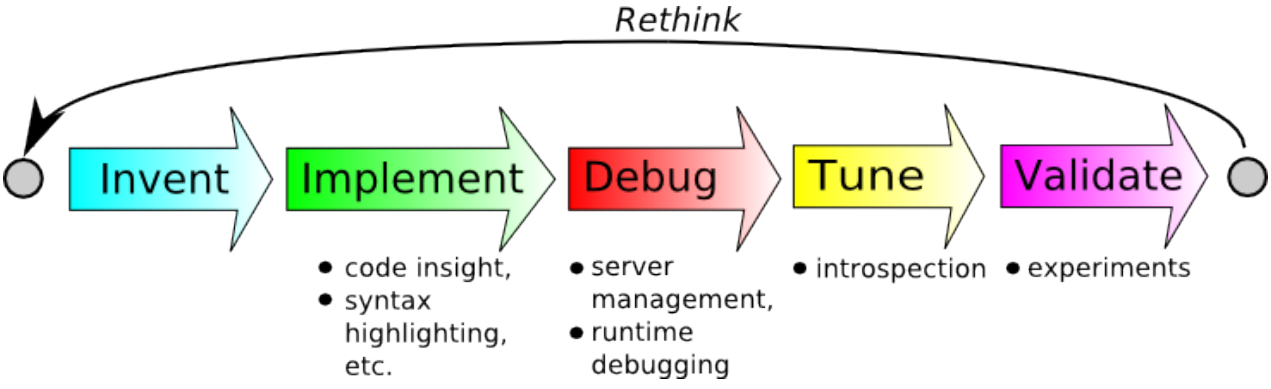

Figure 4 – The bot development cycle.

4.1 Implementing the model

Before a developer starts to implement an agent, he or she has to grasp two notions: the notions of *action selection mechanisms* (ASMs) and of the *communication* between GaviaLib and virtual world. The former is typically conceived as a formalism in which all what the agent can possibly do is represented *plus* algorithms deciding what the agent should actually do based on these representations and the state of the agent at a particular instant. In a sense, an ASM is the top of the agent's mind, albeit the mind possesses other capabilities as well, such as perception and memory. Note that sometimes, in the context of virtual agents, the representation of possible behaviour is referred to as *procedural memory*, which is, in a simplified way, analogical the notion of procedural memory in neuro-/psychology.

The second thing a developer should understand is the logic behind the connection between a bot's mind and the virtual world. Since we are in the realm of autonomous agents, everything what the agent knows about its surrounding has to come through its senses or be given *a priori*. Obviously, the agent has to have a kind of memory to retain this information. Similarly, all what the bot's mind can do in order to move the bot's body is to issue a motor command (recall the mind-body design metaphor). How exactly this communication works? Let us start with incoming messages. For explanatory reasons, we will employ the psychological distinction between sensations and perceptions. By analogy, the incoming messages from the virtual world can be conceived as sensations. The GaviaLib features a low-level API for handling these sensations. Most of the GaviaLib library can be then perceived as a periphery, which makes from these low-level sensations high-level percepts. These high-level percepts, or sensory primitives, are accessible via the high-level API by the ASM situated at the very top of GaviaLib (see Fig. 5). This high-level API disguises complexity of the periphery, and it is this API a typical designer has to work with. Similar mechanisms are employed in the opposite way. There is a high-level interaction API providing high-level motoric primitives, which are translated by GaviaLib to low-level outgoing messages. Importantly, the periphery also holds a kind of working memory available via the high-level API.[5]

5 The psychological distinction between sensation, a low level process of sampling the surrounding of an organism by its receptors, and perception, a high-level process interpreting sensations, is mirrored here only partially. An agent's ASM typically works with "percepts." (i.e. the high-level sensory primitives). Here, we may see the psychological analogy. However, virtual environments typically neither radiate physical energies to be detected by the agent's receptors nor model complex chemical processes to assist in modelling chemical senses, but mediate information more akin directly to outputs of receptors or some cells upstream in the processing pathways (e.g. a presence of an entity at a particular distance: think of robotic proximity sensors or a rat's whiskers; see also Footnote 7). In fact, virtual worlds also often send a high-level information akin to percepts to the agent, such as „there is a ball number 23 with features <X> in the direction *a*". The reasons are technical: to simplify the design and to speed up the computation. Still, not all percepts send by the virtual environment arrive to the agent's ASM, the core of its mind. Some may not pass through an "attention filter" for example. This brings us to he notion of a *possible* percept.

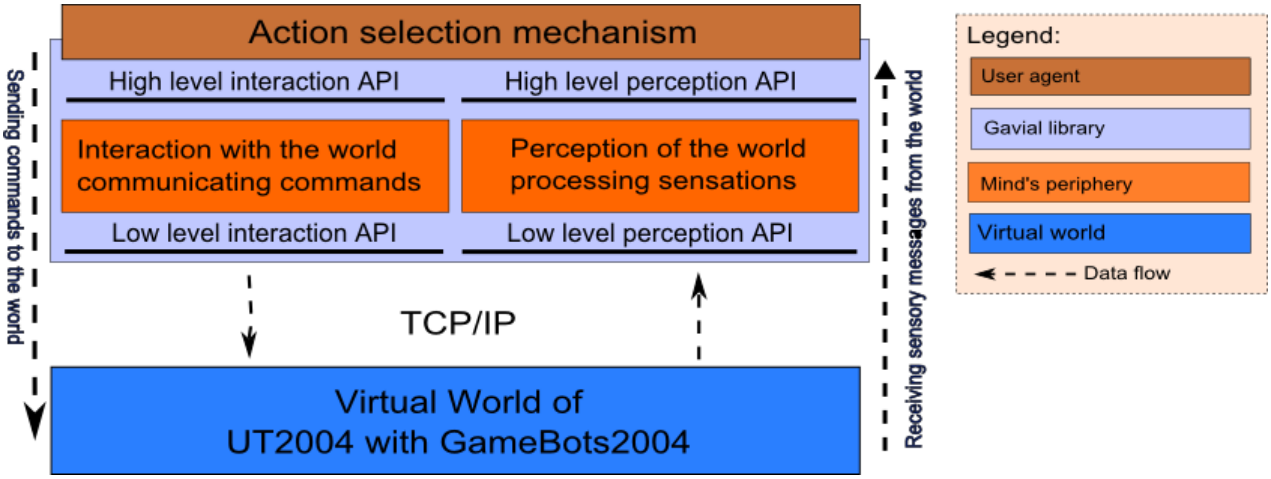

Figure 5 – Abstraction of the GaviaLib.

This section describes development of IVAs from the perspective of the high-level API. The "periphery" is detailed in Section 5.

Now, we have the notions of ASMs and communication, and we can return to our main issue: how to implement a bot using Pogamut 3 easily. At the first place, the designer has to choose an action-selection-mechanism (ASM) that will control his or her agent. The platform currently supports the development of the UT2004 bot's behaviour in Java, Python, behaviour-oriented reactive ASM POSH ("Parallel-rooted, Ordered Slip-stack Hierarchical") (Bryson, 2001a), and fuzzy rules (Štolba, 2008). We have been also adding the cognitive architecture ACT-R (Anderson, 2007), as detailed in Sec. 6.3.

Each type of ASM has its own advantages. Using the plain Java or Python code gives the user the complete power of the underlying GaviaLib library. POSH language is similar to Halo 2 behavioural trees (Isla, 2005) (or more correctly Halo 2 behavioural trees are similar to the POSH reactive plans) and gives the user a simple access to the high-level API and enables him or her to employ decision trees as the algorithm for selection of actions to execute. The ACT-R may be used to create psychologically plausible agents. Fuzzy rules are more suitable for research projects on action selection mechanisms.

The Pogamut 3 plugin together with the Netbeans IDE provides out of the box syntax highlighting and code completion for Java and newly also Python[6]. The Pogamut 3 plugin also offers analogous support for the POSH, namely syntax highlighting and alternative graphical editor (Fig. 6) for editing the POSH plans without any extra programming knowledge making the behaviour coding accessible for the high school students as well.

6 As of NetBeans v6.5.

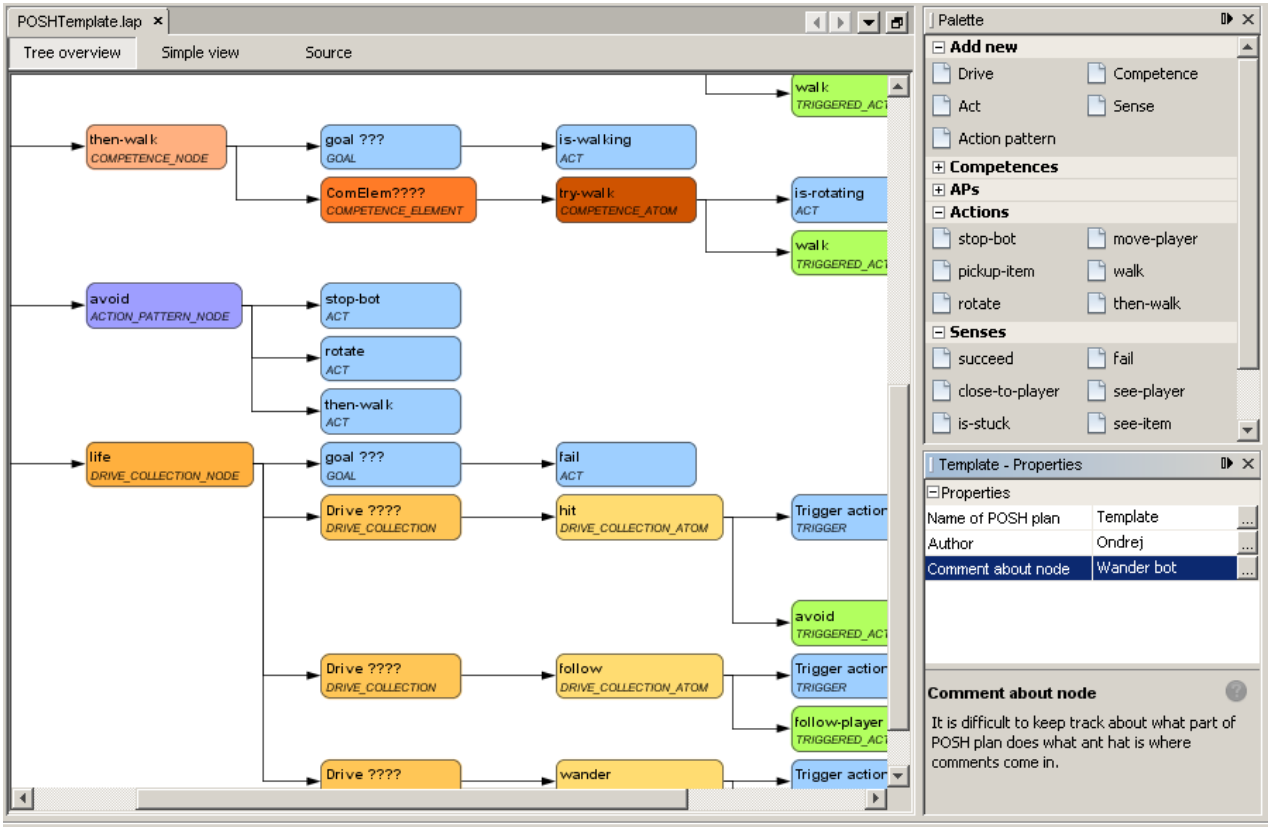

Figure 6 – The editor of POSH plans.

Regardless on the ASM the designer chooses, he or she will use some features of *UT04 integration*, which is a set of classes tailoring common GaviaLib classes to the virtual world of UT2004, namely so called *bot modules*. The bot modules are the classes that provide the high-level senso-motoric API. There are three kinds of bot modules: 1) command modules, 2) event modules, 3) advanced modules.

Command modules

The command modules are merely wrappers for low-level messages, so-called command objects (see Fig. 7), that are sent to the UT2004. The command modules spare the user of the necessity to instantiate a command object every time her or she wants the bot to do something in the world. They also aggregate similar commands into one command module, thereby providing an API for moving and shooting. Provided command modules are:

- *Simple commands* – basic locomotion and shooting commands suitable for beginners
- *Locomotion* – all possible movement commands including jumping and strafing
- *Shooting* – advanced shooting commands including shooting at a target or using alternative weapon modes

Event modules

Event modules represent the sensory primitives. Each module takes care about a different category of world events like information about the bot, the map or the bot's inventory (what items the bot has). Provided event modules:

- *Self* – provides internal sensors (level of health, armor, location in the world,…)
- *Map* – provides a list of navigation points and items and their positions
- *Players* – provides information about other avatars in the world; whether our agent can see them or not, their location, a weapon they are holding, etc.

Advanced modules
Advanced modules are usually built upon other command and/or event modules providing a specific functionality. At this moment, there exist two such modules – *Path planner* and *Path executor*. The *Path planner* module works over the *Map* module allowing the user to easily find the path in the world to a desired location by means of A* or Floyd-Warshall algorithm. The path is then followed by the *Path executor*, which is periodically issuing necessary commands so the bot may reach a target location.

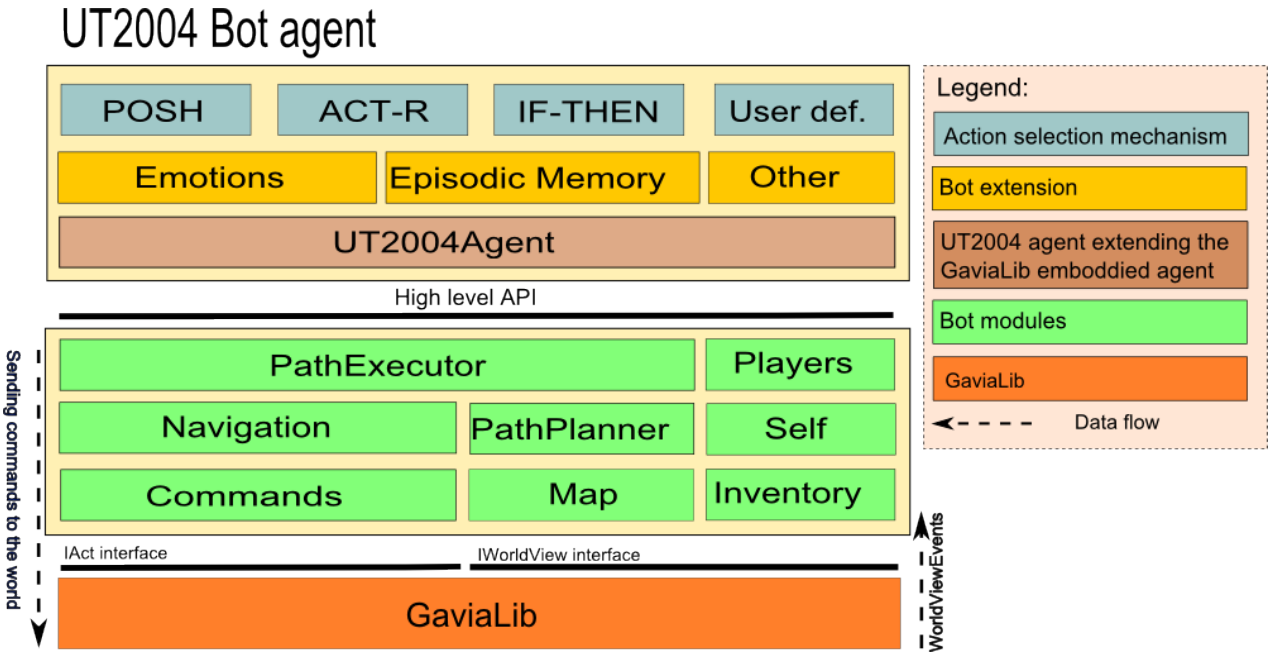

Figure 7 – Bot modules and agent extensions built on top of the GaviaLib. Compare this figure with Fig. 5.

These sensory-motoric primitives will be employed by a developer when developing an ASM. The ASM can also utilise various auxiliary advanced modules, such as episodic memory or an emotion module (see Fig. 7).

Description of how to develop the core of the agent's mind is out of scope of this chapter. The reader is recommended to consult our tutorials, which are available on-line (Pogamut, 2009), or an introductory paper (Bryson, 2001b).

4.2 Debugging

After an agent is created, it should be tested - a process that includes debugging. Actually, the processes of creation and testing are typically intertwined to some extent. Pogamut 3 supports this by enabling the user to use standard NetBeans Java debugger (step over, trace into etc.).
Pogamut 3 may pause the simulation inside UT2004 at any time, allowing the user to debug one

iteration of the agent's mind. Besides the classic debugging, the Pogamut 3 toolkit introduces other tools such as log management, which is useful for inspecting which events the agent has received and which commands has it issued recently, variable introspection, which assists in inspecting the internal state of the agent, and remote UT04 server management. The IDE also offers an option to change the speed of the simulation or to stop it. This feature helps the designer to consult logs on-line.

Typically, during debugging, a user will spend most time observing behaviour of his or her bot directly in the UT2004 environment. The UT2004 game provides a mode for observing the world from the spectator point of view, but this is not always convenient for debugging as many underlying information such as the navigation graph or the bot's intended path are not visible. Additionally, the spectator mode forbids the user to fly through walls. Thus we have extended this native mode with several visualization tools (see Fig. 8):

- Visualization of navigation points on the map together with the visualisation of the reachability grid on the map – gives a quick overview of a bot's navigation capabilities; places without navigation points are usually unreachable (Point 1)
- A list of all avatars in the world – serves as a quick overview of avatars present in the world, both computer driven as well as human driven (Point 4).
- Visualization of individual avatars on the map – allows a user to see avatars through walls, which assists in finding a desired bot in the environment quickly (Point 2)
- Visualization of a bot's trace lines – assists in development of steering algorithms and algorithms of low-level sensation[7]; green colour indicates that the ray does not collide with anything, red indicates a collision (Point 3)
- Visualization of the bot's field of view – helps to identify objects the bot can see; the yellow lines enclose the field of view while the white line indicates the bot's heading (Point 5)
- Visualization of the bot's intended path – helps with debugging of the bot's navigation (Point 6, blue lines)
- Visualization of the bot's health – helps with debugging of bots for the death-match game mode of UT2004 (Point 7, the health bar)

7 The trace lines, or ray casting lines (Fig. 8), serve to give a bot information about distances towards surrounding objects and walls; think of a rat's whiskers or proximity sensors of robots.

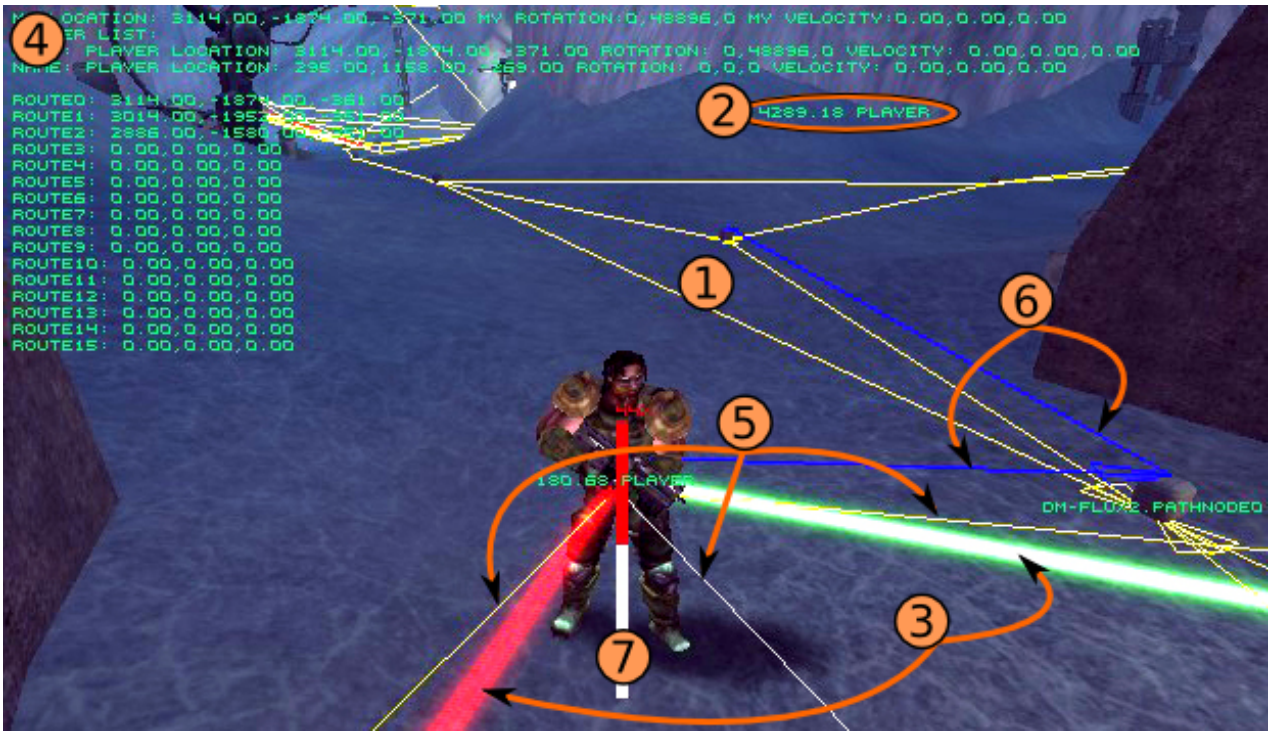

Figure 8 – Debugging features for the UT2004 environment. The individual features are described in the text. (Copyright Epic Games, Inc.)

4.3 Tuning the parameters

Behavioural models are typically quite sensitive to the settings of various parameters. It is convenient to have a support for changing these parameters in runtime, avoiding the need to recompile the agent's code each time a change is made. The IDE allows a developer to set those parameters through the Introspection window without the need for restarting the agent (see Fig. 9).

Presently, the introspection is available for agents with a Java-based, Python-based or POSH-based ASM**.** Java-based agents must have their parametrs annotated with `@JProp` annotation inside the code, while Python-based agents have all their class fields exported automatically. Their values are periodically updated at runtime and can be altered as well. The user may simply pause the simulation, set new values and resume the simulation.

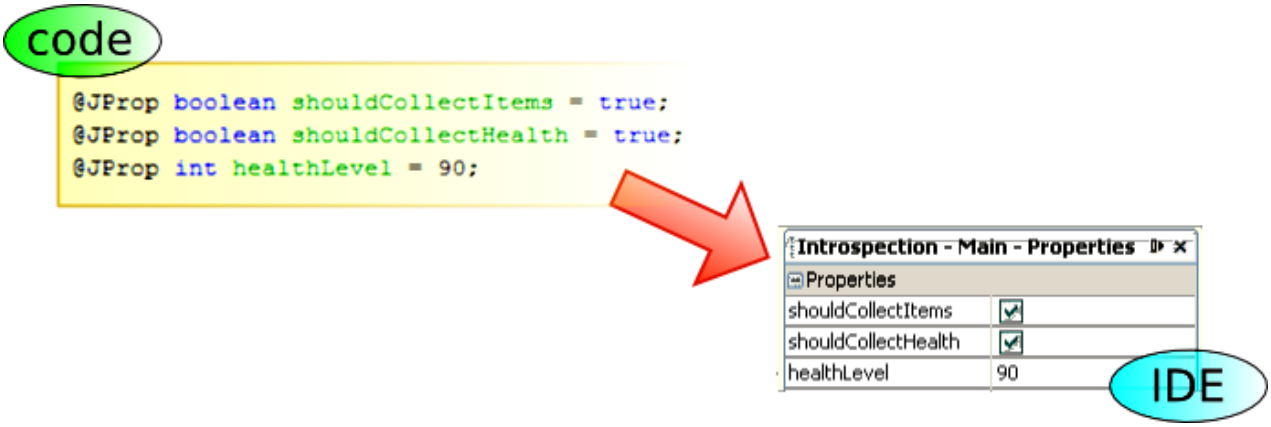

Figure 9 – An example of a bot's parameters introspection.

4.4 Experiments

A developer can define various experiments in the IDE to evaluate a finished bot. Each experiment can model a particular situation, in which the bot can be run multiple time. The main idea is to allow the developer to separate the bot's logic and various test cases, a reminiscence of so-called Unit Testing in Extreme Programming (XPprogramming.com).

The module for experiments enable the developer to define custom actions for various events that are based on the bot's internal variables and on the state of the environment. The API for experiments provides methods for the specification of an experimental scenario including a map of the virtual environment, a number of bots, their starting locations and their equipment. The IDE also allows users to configure breakpoints in the course of an experiment.

4.5 Beyond simple experiments – the Pogamut GRID

A solid statistical validation of a bot's behaviour requires many repetitions of an experiment. Running thousands of experiments on one computer can take days or even weeks, thus it is convenient to run the experiments in parallel on multiple computers. We have developed the Pogamut GRID for this purpose. The Pogamut GRID is based on the JPPF framework (JPPF, 2008). It supports management of many UT2004 servers running in pseudo real-time.

The grid architecture follows the architecture of any grid application built on top of the JPPF (see Fig. 10). The *Client* computer sends the definition of experiments to the *Driver*, which is the gateway to the GRID. The *Driver* resends the experiment to *Nodes*. The *Nodes* represent the computational units of the GRID – computers with UT2004. Each experiment is executed on the *Node* where it has been delivered. The result is sent back to the *Client* after the experiment had been finished. Powerful nodes can run multiple UT2004 servers simultaneously. The GRID also monitors system resources and pauses some of the nodes when their CPU load is close to the maximum sustaining the correctness of the experiment (otherwise the course of the experiment could differ from a run on a single computer).

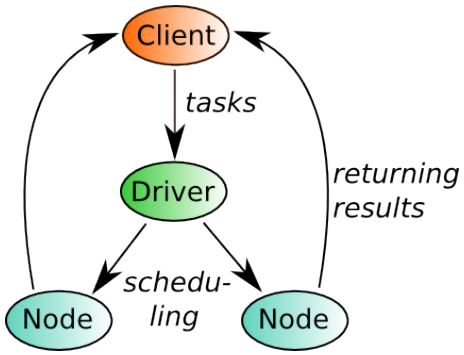

Figure 10 - The GRID architecture. The Client sends the definition of experiments to the Nodes through the Driver. Results of the experiments are returned directly to the Client.

4.6 Example bots

The Pogamut 3 NetBeans plugin also features a set of example bots that support first steps with the platform. These examples are tailored to be used during the class courses utilizing Pogamut. Reading example bot's code is perhaps the most convenient way how to begin with the Pogamut platform. The bots range in complexity from simple ones, e.g. a path-follower, to the complex one, the Hunter bot that features a full set of behaviours that are needed for the death match game

mode (collecting items, combat behaviour, etc.). Examples build upon each other – every example bots' code introduces the user a few (one to three) GaviaLib features. These features include:

- motoric library of the bot (running, jumping, shooting, etc.)
- bot's sensors and the event handling model (seeing the enemy, querying the game state, etc.)
- path-retrieving and path-following, obstacle avoidance with ray casting
- bot's customizations (its shooting skill level, appearance in the game, etc.)
- the POSH reactive planner

4.7 Summary

Pogamut 3 brings together all basic features necessary for newcomers to start their first projects with IVAs, as well as for advanced developers seeking for a platform that would assist them in their development and/or research: a virtual world simulator, the GaviaLib library containing the code that solves many tedious technical issues and providing clear high-level API, and NetBeans integrated development environment together with the set of example bots. Beginners can found useful a workflow example in Appendix A, on-line tutorials, and the web forum (Pogamut, 2009).

Form the educational perspective, it is important that the virtual environment presently integrated with Pogamut 3, the UT2004 game, appeals to many (typically male) students who like to play computer games. Pogamut 3 introduces them IVAs from an unknown perspective; from inside, so to speak. It allows them to *build* agents which they could only interact with by that time, thus giving them a gentle knock out of the door of the gaming world into the world of behavioural modelling. Importantly, Pogamut 3 can be used as a tool for high-school computer science courses (Brom et al., 2008).

From the educational perspective, it is no less important that UT2004 is a violent action game. This clearly limits possible applications of Pogamut 3, as also showed by the users feedback (Brom et al., 2008), and questions whether it is applicable for girls at all. The issue of connecting Pogamut 3 to another environment is detailed in Section 6. That section also reviews our current work in progress aiming to make Pogamut 3 more suitable for the domains of virtual storytelling, serious games, and cognitive science research.

Now, we turn our attention towards technicalities of Pogamut 3, that is, we introduce it from the perspective of a programmer willing to know more about how the GaviaLib library and the communication with UT2004 work.

5 Developing Pogamut 3

This section concerns itself with the things under the roof of the high-level API of the GaviaLib library. It describes individual components of GaviaLib, and GameBots2004. After reading this section, a programmer should have a basic idea how to modify either of these two parts of Pogamut 3.

5.1 GameBots2004

The UT2004 videogame, developed by Epic Games and Digital Extremes, is one of a few videogames with so-called "partly opened" code. The core of the game comprises UnrealEngine, providing basic functionality for running the virtual world, such as rendering 3D graphical

models. To execute various functions of this core, the authors of the game built UnrealScript, a native scripting language of the game. The game mechanics, including bots' logics, is programmed in UnrealScript. While the code of UnrealEngine is not opened, the code written in Unreal is. This fact and additional support[8] allowing for creation of new maps and other models easily enables users to create new extensions and blend them with the original game content. This feature makes UT2004 very appealing for the community of advanced players with programming and/or computer graphics skills, and, not surprisingly, also for researchers. The game extensions are called *mods*.

GameBots2004 (GB2004) is a special mod for UT2004. Its main purpose is to make the rich virtual world of UT2004 available for virtual agents developed *outside* the UnrealScript. GB2004 achieves this by providing a network TCP/IP text protocol for controlling in-game avatars by external programs.

The original GameBots project (Adobbati et al., 2001) was started by Andrew N. Marshal and Gal Kaminka at the Information Sciences Institute of University of Southern California. Their GameBots used older version of the videogame – Unreal Tournament 1999. The goal of the project was to enable the use of the environment of the Unreal Tournament game for research in artificial intelligence.

The GameBots project was continued by Joe Manojlovich, Tim Garwood and Jessica Bayliss from RIT university (RIT, 2005). They ported original GameBots to Unreal Tournament 2004.

In the meantime, first Pogamut GameBots branch emerged due to the debugging of the old Marshal & Kaminka GameBots version. This branch was used in the Pogamut 1. Later, this branch went through major code refactorisation, implementation of new features, correcting bugs and extending the original communication protocol. Finally, it was ported by our team to UT2004 for Pogamut 2 and it is also used for Pogamut 3 (with minor extensions). This branch has been named Gamebots2004. As far as we know, GameBots2004 is the only branch still developed at the time of writing this chapter.

The GB2004 mod runs a server, which provides a means for controlling avatars inside UT2004. Typically, every agent features one instantiation of the GaviaLib library and connects to this server as a client. Whenever a connection to GB2004 server is made, GB2004 spawns a new avatar into the UT2004 world and the server becomes the eyes and the ears of the agent, periodically sending sensory information to the agent while accepting its commands (see Fig. 11).

The GB2004 communication protocol is detailed in the on-line supplementary material (Pogamut, 2009).

8 Namely UnrealEd, which is the UT2004 virtual world editor.

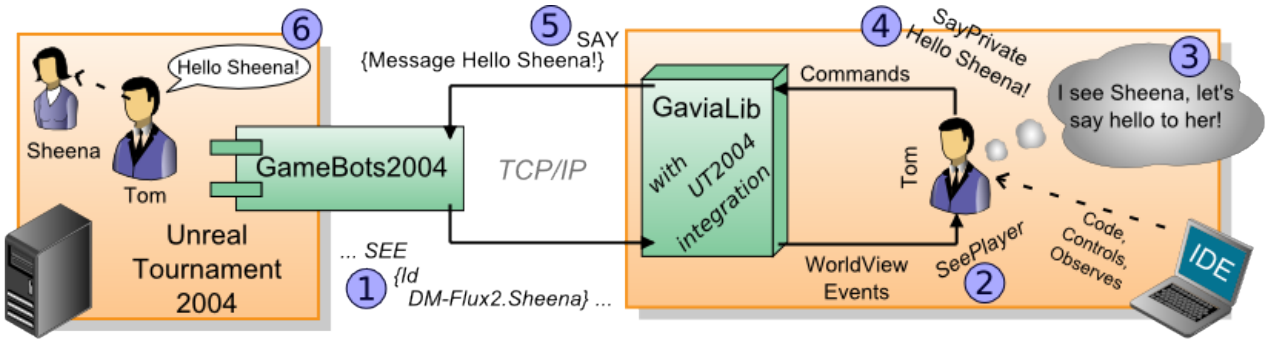

Figure 11 – The high-level architecture of the Pogamut 3 platform with GB2004 and UT2004 integration picturing one iteration of sense-reason-act cycle. On the left side, there are the UT2004 server with GB2004 mod and two bots (Tom and Sheena) inhabiting the virtual world. The GB2004 notices that Tom can see Sheena, therefore it exports this information with *SEE* message (1). *SEE* message is then translated into a Java object inside GaviaLib thanks to the UT2004 integration and presented to the Tom's action selection mechanism via the high-level API in the form of so-called WorldView event *SeePlayer* (2). The Tom's ASM decides that he should greet Sheena (3) and issues the *SayPrivate* command (4). The command is translated into the low-level *SAY* message (5) that is understood by the GB2004, which orders Tom's avatar to do it (6).

5.2 GaviaLib

The real core of the Pogamut 3 framework lies in the GaviaLib, which acts as a middleware between a virtual world and the core of agents' minds. The important point is that GaviaLib is defined as a generic library for different virtual worlds and agents utilising different action selection mechanisms.

However, Pogamut 3 also features a set of classes extending GaviaLib to enable development of UT2004 agents. This extension is called UT2004 integration.

GaviaLib itself is composed of the following three parts (see Fig. 12):

1. Perception of the world – this part serves to process information messages from the virtual world that Pogamut 3 is connected to.
2. Interaction with the world – this part sends commands to the virtual world.
3. Agent abstractions – this part comprises stubs of agents, defining high-level interface for agent's minds.

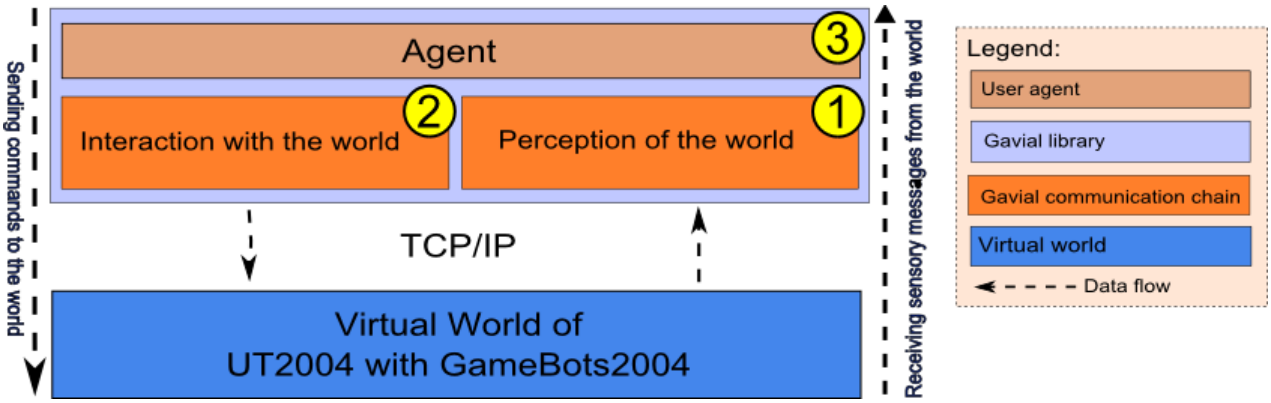

Figure 12 – GaviaLib architecture concept – the library consists of three parts described in the text. Compare this figure with Fig. 5.

UT2004 integration customises each of these parts for the purposes of UT2004 agents.

1. Perception of the world – this part tailors the handling of incoming communication to the protocol employed by GB2004.
2. Interaction with the world – this part tailors the outgoing commands of GaviaLib to the communication protocol of GB2004.
3. Agent abstractions – it provides abstract classes for three types of UT2004 agents together with the high-level API for bot developers; essentially, this API presents the bot modules described in previous section.

Note that agents from diverse worlds can differ not only in classes for perception and interaction, but also in classes for agent abstractions. The reason is that high-level information about what an agent knows about its virtual world is represented in these classes and these representations can be different for different virtual worlds. As an example, consider high-level terrain representation. While UT2004 uses navigation points, the world of Quake III (Id software, 1999), another action game which is sometimes utilised as a virtual environment, describes the terrain by means of rectangles (van Waveren, 2001).

A developer can either use the high-level API (3) to communicate with a virtual world, or dive bellow this API and use perception (1) and interaction (2) classes to control a bot's avatar more directly. While the former option is suitable for beginners, the latter is advisable only for advanced developers. A programmer willing to connect Pogamut 3 to a new virtual environment will need to customise (1 – 3) for the purposes of this environment.

Generally, GaviaLib utilises two Java technologies:

1. Java Management Extension (JMX) (JMX, 2008) for remote agent instrumentation
2. Guice[9] (Guice, 2008) for dependency injection.

The rest of this section overviews the three parts of GaviaLib and then details the whole communication pipeline from an agent's mind to a virtual world and other way round.

5.2.1 Perception of the world

This part of the library presents an abstraction of an agent's world view, its sensory periphery. The important assumption we made is that the virtual world is event–driven, which means that its state changes can be described by instances from a finite set of possible events. GaviaLib distinguishes between two types of events: object events and general events. The former are events describing a state of an object that resides in the world as perceived by the agent ("the agent sees a green ball"), or the change of this state ("the green ball has changed its position"). The latter are other events that the agent can possibly be aware of, e.g. "agent hears a clap of hands", "agent hits a wall".

9 Guice is the Google lightweighted dependency injection framework based on the Java annotations.

The events are processed by the main library component called the *EventDrivenWorldView* (*EDWV*). This component also represents the agent's current known state of the world, offering the following functionality:

- maintaining a list of known world objects and updating their states via object events,
- allowing the agent to set up various listeners for incoming both general and object events,
- a locking mechanism, which enables the agent to freeze its current state of the *EDWV* for reasoning purposes.

5.2.2 Interacting with the world

The second part of the library is quite simple, it merely translates high-level motor commands of an agent's ASM to text messages GB2004 understands.

5.2.3 Agent abstraction

This part of GaviaLib presents stubs for defining bots' logics. These stubs provide basic start/pause/resume/stop methods for controlling the bot's avatar and describe a high-level state of the bot (e.g. running, terminated, etc.). The library distinguishes between three types of agents:

1. an observer agent,
2. a ghost agent,
3. an embodied agent.

Observer agent
The observer agent is an abstraction of a bodiless agent that can not perform any physical action in the virtual world – it can only receive world events. This kind of an agent can be utilised as an observer storing statistics about the world and agents behaviours, for instance counting the number of conversations between agents.

Ghost agent
The ghost agent is an abstraction of a bodiless agent that can not only perceive the world but also act. It may be utilised to create a world controller agent. For instance, this type of agent can be used in storytelling as a story director (Magerko, 2006) that is not present in the world but may observe actors and guide them (e.g. alternating their goals).

Embodied agent
The embodied agent is an abstraction of classical embodied agent. Technically, it differs from the ghost agent in only one feature: it has the body and hence an avatar in the virtual world.

5.2.4 GaviaLib in detail

This section describes GaviaLib in detail. This part is quite technical and the reader willing to be spared these subtleties is suggested to skip to the next section. For explanatory reasons, the library is described together with the UT2004 integration. Figure 13 provides detailed overview of the library architecture. At the top of the figure, there is an agent that uses high-level GaviaLib API. The UT2004 world together with GameBots2004 is pictured at the bottom.

We will start the description in the point when a message is generated by GameBots2004 and sent via TCP/IP to the agent client. This messages utilises the GameBots2004 communication protocol,

which has a text format. The generated massage is, first, accepted by the class *SocketConnection* and, second, passed over to the *Parser* that transforms the text content of the message into a Java object called *InfoObject* (which is a wrapper for parsed textual message). *InfoObject* is forwarded to the *WorldMessageTranslator*, whose main function is to translate *InfoObects* into *WorldEvent* objects and categorise them either as object or global events (see above). While semantic of *InfoObject* is based on the needs of UT2004, the semantic of *WorldEvent* objects is tailored to the needs of agents' ASMs. This translation may include aggregation of more *InfoObjects* into one *WorldEvent* object, or generation of more *WorldEvent* objects based on one *InfoObject*, for instance more PathNode *InfoMessages* may form one Path *WorldEvent.*.

The whole process described in the preceeding paragraph is controlled by one thread. This thread is wrapped in the *Mediator* object. Actually, the *Mediator* reads world events periodically and sends them up to the *WorldAdapter*.

The *WorldAdapter* is optional component of the diagram. It serves for three purposes. First, it can provide a place for altering information sent by the world to the agent, if needed. For example, it can implement an attention filter. Second, a virtual world simulator may actually lack some functionality and the *WorldAdapter* can provide it. For instance, UT2004 does not compute many drives that may a developer need for his or her agents, such as stress or hunger. These drives can be implemented by the *WorldAdapter*. Finally, the *WorldAdapter* passes the *WorldEvent* objects up to the *EventDrivenWorldView* (*EDWV*).

The *EDWV* is the most complex object of the GaviaLib, which has been already introduced in Sec. 5.2.1. As stated, it implements three mechanisms, which function together as a kind of working memory: 1) It maintains a list of known world objects and updates their states via object events. 2) It allows the agent to set up various listeners for incoming events. 3) It implements a locking mechanism, thus the agent may freeze the state of the *EDWV* for reasoning purposes – one usually wants to reason over the fixed world state (but this is not mandatory behaviour).

The output of the EDWV is used by bot modules introduced in Sec. 4.1, which provides the high-level API for ASMs.

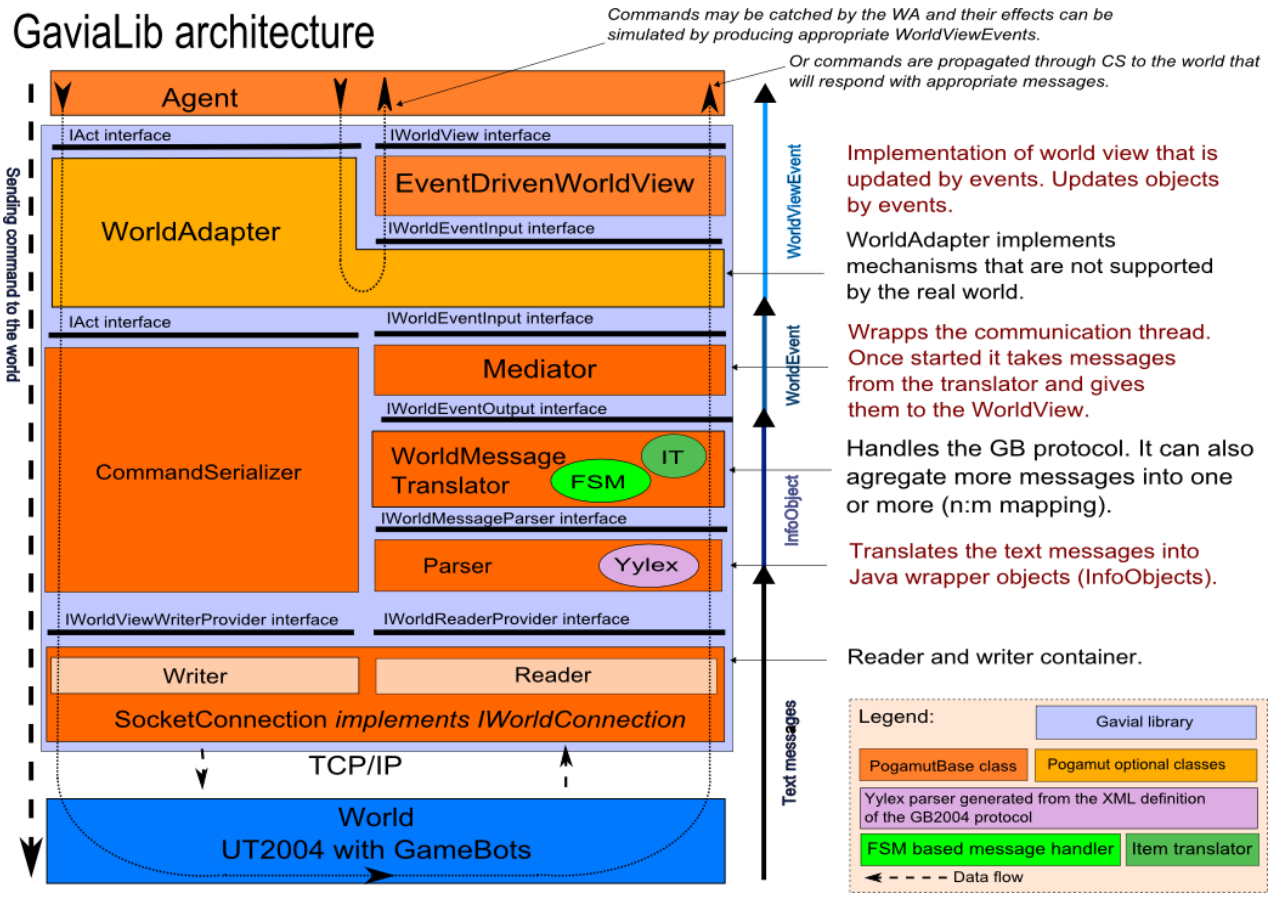

Figure 13 – The architecture of the GaviaLib together with the UT04 integration. This figure presents a detailed view on the architecture depicted on Figure 5.

The communication from the agent towards the UT2004 is much simpler. The agent's ASM can control the agent's avatar by means of motor primitives. There is a fixed set of them as defined by GB2004. Technically, the ASM needs to send a so-called *CommandObject* wrapping a desired motor primitive to the *WorldAdapter*. This functionality is provided by command modules (Sec. 4.1).

Every *CommandObject* can have one of the two fates. First, it may be handled solely inside the *WorldAdapter*, which may consequently react by generating a new *WorldEvent* object back for the *EDWV*. This possibility applies for situations in which the EDWV implements functionality missing in the virtual world, such as simulating the missing hunger drive. Second, the *CommandObject* may be passed through the *CommandSerializer* to the *SocketConnection* that will send it down to the UT2004 where the command is handled by GB2004.

Object technologies

All objects instantiated from classes of the GaviaLib are strictly interacting via interfaces and all objects are created using dependency injection[10]. Every interface defines a contract[11] for a class implementing it. As the GaviaLib is utilising Guice technology[12] for instantiation of objects, it is a

10 Dependency injection means that objects are constructed with all needed objects specified within the constructor call.

11 Contract is an expected behaviour of the object implementing a specific interface. The semantic of a contract is typically specified in the interface documentation.

12 See Footnote 9.

matter of one line of code to swap an object of a default class implementing a chosen interface with a user-created object of a class implementing this interface. For instance, if a developer wish to implement new "eat apple" *CommandObject* that is not supported by the UT2004 world, he or she may create own implementation of the *WorldAdapter* that will catch every "eat apple" *CommandObject* and generate appropriate *WorldEvent* objects for *EDWV* and/or low-level commands for UT2004 avatar removing the apple from the agent's inventory and decreasing the agent's hunger drive.

Flexibility of GaviaLib
The object technologies employed and the design of GaviaLib makes the architecture of the library quite flexible. For instance, GaviaLib does not require the virtual world to communicate via TCP/IP. GaviaLib works with standard Java objects *Reader* and *Writer*, which are provided by the GaviaLib's interfaces *IWorldReaderProvider* and *IWorldWriterProvider*. We may also totally omit the *Parser* and the *WorldMessageTranslator* classes in cases the virtual world is run inside the same Java Virtual Machine running the GaviaLib and connect the virtual world directly to the *Mediator* via *IWorldEventOutput* interface.

Remote control of GaviaLib agents
The GaviaLib utilises the JMX framework that may be used for the remote management of the agents. Thus the agents may be controlled from different Java Virtual Machines even across the network. This feature is used by the IDE but it also allows for building resource-consuming agents, for instance agents with planners, and running them on multiple machines while managing all of them from one machine.

6 Limitations and prospect

The integration of Pogamut 3 with UT2004 and the support provided by GaviaLib classes make Pogamut 3 most suitable for gaming AI projects. This suits some, but is unsuitable for others. Apart from other things, it is disputable whether a gaming world is the right environment for a general platform of teaching building of the behaviour of IVAs.

We are now extending Pogamut 3 with features that simplify the platform's usage also in the fields of virtual storytelling and computational cognitive psychology. This section reviews the most notable extensions.

6.1 Support for artificial emotions and gestures

When a virtual agent expresses emotions, the user can typically better understand its internal state, thus the whole scenario in which the agent acts. Not all present-day applications actually need emotions to help to elucidate an agent's internal state, such as action games like Unreal Tournament, but many do. For instance, emotions are important for serious games such as FearNot! (eCircus, 2009). Present version of Pogamut 3 supports neither emotion modelling nor expressing. However, we have been working on this issue along three lines. First, we have been integrating Pogamut 3 with a generic emotion model. Second, we have been developing new graphical models for agents' avatars expressing emotions. Third, in order to offer higher flexibility of avatar animation, we decided to give users a way of influencing avatars' body animations. The first and the third points will be now detailed.

6.1.1 Emotion modelling

We have been working on integration of the ALMA model with the Pogamut 3 platform. ALMA (Gebhard, 2005) is an emotion model based on OCC (Ortony et al., 1988) cognitive theory of emotions, a theory on which many virtual agents projects capitalise. ALMA models a character's affect with OCC emotion categories and 3-dimensional mood; the dimensions are pleasure, arousal, and dominance. For defining an agent's unique emotion reactions, ALMA exploits "Big five" personality model (McCrae & John, 1992). This personality model affects both the emotions as well as the mood of the agent. The ALMA model also features a powerful GUI where the user can inspect an agent's current emotions and mood and specify the agent's parameters and personality.

Figure 14 – Example of the gesture.

6.1.2 Body animation

Besides equipping agents with emotions, the challenge is to allow users to control UT2004 avatars directly from the IDE of Pogamut 3 at the level of atomic animations, parametric animations and their sequences; this is not possible presently. Examples of atomic animations include "arm flexing," "arm twisting," or "arm abduction." A parametric animation is "pointing with a finger towards a target." We have been extending Pogamut 3 with this feature. To allow the user maximum flexibility, the graphical libraries and the parser will be situated on the side of the IDE, not the virtual world.

Technically, the graphical libraries will exploit behaviour Markup Language (BML), which was developed by (Mindmakers, 2005-8) and is intended to become "a framework and an XML description language for controlling the communicative human behaviour of embodied conversational agents" (Mindmakers, 2008). It provides a convenient way to create and parse sequences of behaviours, including gestures, facial animations and speech as well.

Communication with the virtual world will be carried in slightly modified FBA bitstream format specified in MPEG4 (Capin et al., 2000). MPEG4 describes a protocol, which allows definition and transportation of parametric animations using very little bandwidth. A user will be able to fabricate his own frames, that is, to control animations directly. FBA bitstream is quite modest itself, but in order to achieve even less traffic, we decided to include a variable frame rate. Most common human movements such as waving or clapping hands can be easily interpolated without no considerable impact on believability.

6.2 Interactive storytelling

Interactive storytelling is a modern discipline that joins researchers from various fields attempting to challenge the paradigm of conventional linear stories by creating dynamically changing narratives in which users can interact. These attempts have many facets. For example, one goal is to develop a text-based system for automatic narrative generation, in which a user can make important decisions about the course of the story on-line. Another goal is to build an application unfolding stories in virtual environments inhabited by IVAs (e.g. Cavazza et al., 2007). The latter is also our scope. The starting point is a tool that would assist in rapid development of short scenarios in which several virtual characters and possibly human avatars interact. An example of

such a scenario is buying a cinema ticket or asking another agent for an information. Not many people realise that specifying such scenarios and their unfolding typically requires different techniques than those that are utilised by action selection mechanisms of typical videogame agents. While there already exists a couple of tools for building some storytelling applications (reviewed in Pizzi and Cavazza, 2008), we are not aware of any free tool allowing for building such scenarios.

6.2.1 Technical details

One useful way of conceiving control architectures of virtual storytelling applications is the two-layered "individual characters – story manager" conceptual framework (e.g. Szilas, 2007; Magerko, 2006). Basically, besides embodied agents, there is a bodiless agent in the virtual environment that coordinate behaviour of the embodied agents to unfold the desired scenario correctly. The embodied agents can be autonomous to some extent, but the story manager is typically able to alter their goals and needs.

The million dollar issue here is to unfold a correct story, or to generate one automatically, despite a user's interactions, so-called "narrative-interactive" tension. We do not address this question directly. Instead, our goal is to build an infrastructure for developing story directors in Pogamut 3, including a language, called StorySpeak, for the description of how to unfold particular scenarios (Gemrot, 2009). Using this infrastructure, a developer will be able to challenge the issue of "narrative-interactive" tension.

The language alone extends AgentSpeak(L) (Rao, 1996) and gets inspiration from its Java implementation Jason (Bordini et al., 2007). Basically, StorySpeak will allow for specifying joint agents plans. The language is built without any assumptions about the underlying virtual world, meaning its core will not contain any sensory-motor primitives directly. However, we will provide also UT2004 implementation, which will define the sensory-motor primitives for this world and which will also feature a set of predefined small joint plans such as greeting.

6.3 ACT-R extension (PojACT-R)

Recently, it has been proposed that computational cognitive science modelling can benefit from implementing models using IVAs (Brom & Lukavsky, 2008; see also CAGE, 2007; Jilk et al., 2008; see also Laird & van Lent, 2001 for similar arguments concerning building human-level AI systems). That is, a neuro-/psychologically plausible model can be integrated within an IVAs "mind" and then tested in scenarios in virtual worlds. Advantages of this approach has been discussed in (Brom & Lukavsky, 2008). One potential way how to do this is to adopt a general cognitive architecture, such as Soar (Newell, 1990), ACT-R (Anderson, 2007) or LIDA (Franklin et al., 2005), as the main architecture of an IVA's mind. These architectures are somewhat plausible and tend to be modular, hence one can embed his or her model in one of the modules without loosing much of its original plausibility, as Jilk and colleagues (2008) did for a model of visual cortex.

To allow the cognitive science modeling community to use Pogamut 3 in such a way, we have started to work on integration of ACT-R with Pogamut 3.

6.3.1 Technical details

ACT-R is originally designed for LISP but few years ago Anthony Harrison from Naval Research Laboratory started the jACT-R project (Harrison, 2008), a Java port of ACT-R theory. jACT-R supports running of models described by the old-style LISP syntax as well as a new XML syntax. We have been integrating jACT-R with GaviaLib, developing a generic template for PojACT-R agents (that is, Pogamut – jACT-R). This template features classes translating world events into PojACT-R messages, which can be interpreted by jACT-R as sensations of agents. In a similar fashion, the template supports translation of jACT-R motor commands to GaviaLib motor primitives allowing jACT-R to control the agent's avatar.

6.4 Episodic memory

From the psychological point of view, episodic memory (Tulving & Donaldson, 1972) represents personal history of an entity. Episodic memories are related to particular places and moments, and connected to subjective feelings and current goals[13]. It was argued that episodic memory is an important component for many kinds of virtual characters, ranging from role-playing games agents (Brom et al., 2007) to pedagogical agents and virtual companions (Ho & Watson, 2006; Castellano et al., 2008). All of these agents are supposed to "live" for a long period of time, hence it is important that they remember what has happened. Moreover, experimenting with IVAs enhanced by episodic memory can provide us with an interesting feedback on neuro-/psychological theories underpinning our understanding of episodic memory. Importantly, this understanding advanced immensely in last 30 years, but is still limited (see e.g. Baddley et al., 2001).

We have been working for a while on a generic model of episodic memory for virtual characters (Brom et al., 2007; Brom & Lukavsky, 2009). Originally, the project was independent on the Pogamut platform. Now, we are integrating it with GaviaLib to allow Pogamut's users to develop agents with episodic memory easily. The model possesses five parts: 1) a memory for events, 2) a component reconstructing the time when an event happened, 3) a topographical memory, 4) and an allocentric and 5) egocentric representations of locations of objects. Its main functional features include: detail representation of complex events (e.g. cooking a dinner) over long intervals (days) in large environments (house), forgetting, and development of search strategies for objects in the environment. The model is partly grounded in behavioural data, increasing the agent's plausibility.

6.4.1 Dating of episodic memories

We will briefly illustrate the intricacies of episodic memory modelling on the example of memory for time of events (the component (2)). Every agent with episodic memory should have the notion of time, otherwise it is not able to tell when an event happened. The notion of time has many facets, for instance the agent should be able to remember dating of events and to compare recency of two events. Importantly, a believable agent can not just remember absolute time for it is known that humans are relatively poor in dating (Friedman, 1993). The notion also includes the agent's ability to adapt to a new way of life, e.g. noticing a change of a time-zone. The agent should be also able to speak using relative time concepts such as "after a lunch", or "morning". However, in terms of absolute time units, "morning" is context dependent – Monday morning is typically sooner than Sunday morning. All of these issues, and many others, should be addressed, and

13 In the context of virtual agents, some prefer to refer to autobiographic memory instead of episodic memory (e.g. Ho & Watson, 2006). From the psychological point of view, some distinctions between these memories can be drawn (see e.g. Conway, 2001); however, we will disregard these subtleties here for the terminology is not settled among psychologists anyway (see Baddley et al., 2001).

preferably validated on real users (that is, users should be asked whether it seems to them that an IVA in question behave believably). We have been addressing these issues exploiting a set of special neural networks (Burkert, 2009).

6.5 Genetic behaviour optimization

Fine tuning of parameters of a bot can be a tedious task with a slim chance of success. We have made several experiments applying genetic algorithms to this problem to prove that it can be, at least to some extent, automatized using the Pogamut platform. We have conducted two kinds of experiments: evolving high-level behavioural structures for DeathMatch and Capture the Flag Unreal Tournament game modes using genetic programming (Koza, 1992), and evolving low level movement tasks using the NEAT algorithm (Stanley & Miikkulainen, 2002). Detailed discussion of these experiments can be found in (Kadlec, 2008). What is important for present purposes is that this work demonstrated that at least some evolution computation problems can be addressed in Pogamut 3. This work employed the Pogamut GRID.

6.6 Other virtual environments

As already said, the environment of Unreal Tournament 2004 limits possible use of Pogamut 3 to gaming AI. To overcome this limitation, we are adding a new environment, specifically the Virtual Battle Space (BI, 2008a) developed by (BI, 2008b). Even though this project originated as a gaming virtual environment, recently, it has been also used for military purposes, and because it features large civil environments and many models of avatars, it can be used for the purposes of serious games and storytelling applications. We are also considering other virtual environments, for example Second Life or Defcon[14].

7. Conclusion

We end where we started, with the realisation of how little has been done so far for newcomers willing to enter the field of virtual humans. Our contribution lies in the development of the Pogamut 3 toolkit addressing this issue directly. Pogamut 3 is not a universal authoring platform, but it allows beginners as well as those who are already advanced rapid development of high-level behaviour of virtual characters. Presently, the toolkit is most suitable for gaming AI projects, but several on-going subprojects make it suitable also for general development of human-level AI systems, for computational modelling for cognitive neuro-/psychology and for virtual storytelling.

Acknowledgement

This work was partially supported by the project CZ.2.17/3.1.00/31162 that is financed by the European Social Fund, the Budget of the Czech Republic and the Municipality of Prague. It was also partially supported by the Program “Information Society” under project 1ET100300517, and by the research project MSM0021620838 of the Ministry of Education of the Czech Republic.

14 Defcon is a real-time strategy game where player controls one of the nuclear mocnost and scores points by destroying opponents strategic targets and cities.

References:

(Adobbati et al., 2001)
Adobbati, R., Marshall, A. N., Scholer, A., Tejada, S.: Gamebots: A 3d virtual world test-bed for multi-agent research. In: *Proceedings of the 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal, Canada, 2001. URL: http://www.planetunreal.com/gamebots [27.1. 2009]

(Alice, 1995 – 2009)
Pausch, R. (head), Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., White, J.: Alice: Rapid Prototyping System for Virtual Reality, IEEE Computer Graphics and Applications. (1995-2009) URL: http://www.alice.org/ [27.1. 2009]

(Anderson, 2007)
Anderson, J.R.: How can the human mind occur in the physical universe? Oxford University Press. (2007)

(Aylett et al., 2005a)
Aylett, R.S., Louchart, S., Dias, J., Paiva, A., Vala, M.: Fearnot! - an experiment in emergent narrative. In: *Proceedings of IVA 2005*, Springer Verlag LNAI 3661. (2005) 305 – 316

(Aylett et al., 2005b)
Aylett, R. S., Paiva, A., Woods, S., Hall, L., Zoll, C.: Expressive Characters in Anti-Bullying Education. In: *Animating Expressive Characters for Social Interaction*, L. Canamero and R. Aylett, Eds.: John Benjamins. (2005)

(Aylett et al., 2006)
Aylett, R.S., Dias, J., Paiva, A.: An affectively-driven planner for synthetic characters. In: *Proceedings of ICAPS 2006*, AAAI Press. (2006)

(Baddley et al., 2001)
Baddley, A., Conway, M.A., Aggleton, J. (eds): Episodic memory: New directions in research. Oxford University Press. (2001)

(Badler et al., 2002)
Badler, N., Erignac, C., Liu, Y.: Virtual humans for validating maintenance procedures. In: *Comm. of the ACM.* (2002) 57 – 63

(Badler et al., 2008)
Badler, N., Allbeck, J., Pelechano, N.: Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation). Morgan and Claypool Publishers. (2008)

(BI, 2008a)
Bohemia Interactive Australia: Virtual Battlespace 2. (2008) URL: http://virtualbattlespace.vbs2.com [27.1.2009]

(BI, 2008b)
BI: Bohemia Interactive. (2008) URL: http://www.bistudio.com/ [27.1.2009]

(Bída et al., 2006)
Bída, M., Burket, O., Brom, C., Gemrot, J.: Pogamut – Platform for prototyping bots in Unreal Tournamentu. In: *Sborník příspěvků z konference Kognice a umělý život,* Jozef Kelemen, Vladimír Kvasnička (eds.), Slezská Universita v Opavě, Czech republic, Třešť. (in Czech) (2006) 67-74.

(Bordini et al., 2007)
Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, Ltd. (2007)

(Brom, 2009)
Brom, C.: Curricula of the Course on Modelling Behaviour of Human and Animal-like Agents. In: *Proceedings of the Frontiers in Science Education Research Conference.* Famagusta, North Cyprus (2009) 71 – 79

(Brom & Lukavský, 2008)
Brom, C., Lukavský, J.: Episodic Memory for Human-like Agents and Human-like Agents for Episodic Memory. Technical Reports FS-08-04, Papers from the AAAI Fall Symposium Biologically Inspired Cognitive Architectures. Westin Arlington, VA, USA, AAAI Press. (2008) 42 – 47

(Brom & Lukavský, 2009)
Brom, C., Lukavsky, J.: Towards Virtual Characters with a Full Episodic Memory II: The Episodic Memory Strikes Back. In: *Proc. Empathic Agents*, AAMAS workshop (2009) 1--9

(Brom et al., 2007)
Brom, C., Pešková, K., Lukavský, J.: Where Did I Put My Glasses? Determining Trustfulness of Records in Episodic Memory by Means of an Associative Network. In: *Proceedings of ECAL 2007*, LNCS 4648, Lisbon, Portugal. Springer-Verlag, Berlin. (2007) 243 – 252

(Brom et al., 2008)
Brom, C., Gemrot, J., Burkert, O., Kadlec, R., Bída, M.: 3D Immersion in Virtual Agents Education. In: *Proceedings of First Joint International Conference on Interactive Digital Storytelling ICIDS 2008*, LNCS 5334, Erfurt, Germany, Berlin: Springer-Verlag. (2008) 59 – 70

(Bryson, 2001a)
Bryson, J.J.: Inteligence by design: Principles of Modularity and Coordination for Engineering Complex AdaptiveAgent. PhD Thesis, MIT, Department of EECS, Cambridge, MA (2001)

(Bryson, 2001b)
Bryson, J.J.: How to Make a Monkey Do Something Smart. A brief document about behaviour Oriented Design (BOD). (2001) URL: http://www.cs.bath.ac.uk/~jjb/web/how-to-monkey.pdf [27.1.2009]

(Bryson et al., 2007)
Bryson, J.J., Ando, Y., Lehmann, H.: Agent-based modelling as scientific method: a case study analysing primate social behaviour. Philos. Trans. R. Soc. London B 362(1485). (2007) 1685 – 1698

(Burgess, 2002)

Burgess, N.: The hippocampus, space, and viewpoints in episodic memory. The Quart. Jn. of Exp. Psych, 55A(4). (2002) 1057 – 1080

(Burkert, 2009)
Burkert, O.: Connectionist Model of Episodic Memory for Virtual Humans. Master thesis. Dept. Software & Comp. Sci. Education. Charles University in Prague (2009)

(CAGE, 2007)
CAGE: Cognitive Agent Generative Environment. (2007) URL: http://arlington.setassociates.com/cage/ [27.1. 2009]

(Čapek, 1920)
Čapek, K.: R. U. R. Rossum´s Univesal Robots. Aventinum (1920)

(Capin et al., 2000)
Capin, T.K., Petajan, E., Ostermann, J.: Very low bit rate coding of virtual human animation in MPEG-4. In: *IEEE International Conference on Multimedia and Expo*, Volume 2 (2000) 1107 – 1110

(Castellano et al., 2008)
Castellano, G., Aylett, R., Dautenhahn, K., Paiva, A., McOwan, P.W., Ho, S.: Long-term affect sensitive and socially interactive companions, Fourth International Workshop on Human-Computer Conversation, Bellagio, Italy, 6-7 October 2008. (2008)

(Cavazza et al., 2004)
Cavazza, M., Charles, F., Mead, S.J.: Developing Re-usable Interactive Storytelling Technologies. In: *Proceedings of the 2004 of IFIP World Computer Congress*. (2004) 39 – 44

(Cavazza et al., 2007)
Cavazza, M., Lugrin J-L., Pizzi, D., Charles, F.: Madame Bovary on the Holodeck: Immersive Interactive Storytelling. ACM Multimedia 2007, Augsburg, Germany. (2007)

(Champandard, 2003)
Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive behaviours. New Riders. (2003) URL: http://fear.sourceforge.net [27.1. 2009]

(Companions, 2006)
Companions: Intelligent, Persistent, Personalised Multimodal Interfaces to the Internet. The European Community's Seventh Framework Programme. (2006) URL: http://www.companions-project.org/ [27.1.2009]

(Conway, 2001)
Conway, M.A.: Sensory-perceptual memory and its context: autobiographical memory. In: *Episodic memory: New directions in research*. Oxford University Press (2001) 53 – 70

(Dias et al., 2007)
Dias, J., Ho, W.C., Vogt, T., Beeckman N., Paiva, A., Andre, E.: I Know What I Did Last Summer: Autobiographic Memory in Synthetic Characters. In: *Proceedings of Affective Computing and Intelligent Interaction 2007*, Springer Berlin / Heidelberg. (2007) 606 – 617

(DARPA, 2009)
Defense Advanced Research Projects Agency, http://www.darpa.mil/grandchallenge/index.asp

(eCircus, 2009)
eCircus: FearNot! an anti-bullying application. (2009) URL:
http://www.macs.hw.ac.uk/EcircusWeb/ [27.1.2009]

(Epic, 1999)
Epic Games: Unreal Tournament 1999. (1999) URL: http://www.unreal.com [27.1.2009]

(Epic, 2004)
Epic Games: Unreal Tournament 2004. (2004) URL: http://www.unreal.com [27.1.2009]

(Franklin et al., 2005)
Franklin, S., Baars, B.J., Ramamurthy, U., Ventura, M.: The role of consciousness in memory. Brains, mind, media 1. (2005) 1 – 38

(de Freitas, 2008)
de Freitas, S.: Serious Virtual Worlds report. 3 November, 2008. (2008) URL:
http://www.jisc.ac.uk/publications/publications/seriousvirtualworldsreport.aspx [27.1.2009]

(Friedman, 1993)
Friedman, W.J.: Memory for the Time of Past Events. In: *Psychological Bulletin 113(1)*. (1993) 44 – 66

(Gebhard, 2005)
Gebhard, P.: ALMA - A Layered Model of Affect. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, Utrecht. (2005) 29 – 36

(Gemrot, 2009)
Gemrot, J.: Behaviour coordination of the virtual characters. Master thesis. Dept. Software & Comp. Sci. Education. Charles University in Prague (2009)

(Guice, 2008)
Guice. (2008) URL: http://code.google.com/p/google-guice/ [27.1.2009]

(Harrison, 2008)
Harrison, A.: jACT-R. Project homepage: http://www.jactr.org (2008) [27.1.2009]

(Ho & Watson, 2006)
Ho, W.C., Watson, S.: Autobiographic knowledge for believable virtual characters, In: *Proceedings of Intelligent Virtual Agents 2006 (IVA 2006)*, Springer LNAI. (2006) 383 – 394

(Hodges et al., 2001)
Hodges, L.F., Anderson, P., Burdea, G.C., Hoffman, H.G., Rothbaum, B.O.: Treating Psychological and Physical Disorders with VR. In: *Proceedings of the 2001 of Computer Graphics and Applications*. IEEE press. (2001) 25 – 33

(Id software, 1999)

Id software: Quake III Arena. (1999) URL: http://www.quake3arena.com/ [27.1.2009]

(Isla, 2005)
Isla, D.: Handling complexity in Halo 2. Gamasutra Online, November 3, 2005. (2005)

(IVA, 2009)
IVA: 9th International Conference on Intelligent Virtual Agents. (2009) URL: http://iva09.dfki.de/ [27.1.2009]

(Jilk et al., 2008)
Jilk, D.J., Lebiere, C., O'Reilly, R.C., Anderson, J.R.: SAL: An explicitly pluralistic cognitive architecture. Journal of Experimental and Theoretical Artificial Intelligence, 20. (2008) 197 – 218

(JMX, 2008)
JMX: Java Management Extensions Technology. (2008) URL: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ [27.1.2009]

(Johnson et al., 2004)
Johnson, W.L. et al.: Tactical Iraqi. (2004) URL: http://www.tacticallanguage.com. [27.1.2009]

(JPPF, 2008)
JPPF framework. (2008) URL: http://www.jppf.org [27.1.2009]

(Kadlec, 2008)
Kadlec, R.: Evolution of intelligent agent behaviour in computer games. Masters thesis. Charles University in Prague, Czech Republic. (2008) URL: http://artemis.ms.mff.cuni.cz/main/papers/GeneticBots_MSc_Kadlec.pdf [27.1.2009]

(Kadlec et al., 2007)
Kadlec, R., Gemrot, J, Burkert, O., Bída, M., Havlíček, J., Brom, C.: Pogamut 2 – a platform for fast development of virtual agents' behaviour. In: *Proceedings of CGAMES 07.* La Rochelle, France. (2007)

(Kirschner, 2009)
Kirschner, F.: MovieSandBox. (2009) URL: http://www.moviesandbox.net [27.1.2009]

(Kokko, 2007)
Kokko, H.: Modelling for Field Biologist ad Other Interesting People. Cambridge University Press. (2007)

(Koza, 1992)
Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA. (1992)

(Laird & van Lent, 2001)
Laird, J.E., van Lent, M.: Human-level AI's Killer Application: Interactive Computer Games. In: *AI Magazine 22(2)* (2001) 15 – 26

(LIREC, 2008)

LIREC: LIving with Robots and InteractivE Companions. The European Community's Seventh Framework Programme. (2008) URL: http://www.lirec.org [27.1.2009]

(Magerko, 2006)
Magerko, B.: Intelligent Story Direction in the Interactive Drama Architecture. In: *AI Game Programming Wisdom III*, Charles River Media. (2006) 583 – 596

(Magnenat-Thalmann & Papagiannakis, 2006)
Magnenat-Thalmann, N., Papagiannakis, G.: Virtual Worlds and Augmented Reality in Cultural Heritage Applications. In: *Recording, Modeling and Visualization of Cultural Heritage.* Taylor and Francis Group. (2006) 419 – 430

(Magnenat-Thalmann & Thalmann, 2004)
Magnenat-Thalmann, N., Thalmann, D. (eds.): Handbook of Virtual Humans. Wiley. (2004)

(McCrae & John, 1992)
McCrae, R.R., John, O.P.: An introduction to the five factor model and its implications. Journal of Personality, vol. 60. (1992)

(Meyrink, 1915)
Meyrink, G.: Der Golem. Verlag Kurt Wolf (1915)

(Mindmakers, 2005-8)
Mindmakers forum. (2005-8) URL: http://www.mindmakers.org/index.jsp [27.1.2009]

(Mindmakers, 2008)
Mindmakers: BML draft 1.0. (2008) URL: http://wiki.mindmakers.org/projects:bml:draft1.0 [27.1.2009]

(Newell, 1990)
Newell, A.: Unified Theories of Cognition. Harward University Press, Cambridge, Masachusetts. (1990)

(Ortony et al., 1988)
Ortony, A., Clore, G., L., Collins, A.: The Cognitive Structure of Emotions. Cambridge University Press, Cambridge, MA. (1988)

(Pizzi & Cavazza, 2008)
Pizzi, D., Cavazza, M.: From Debugging to Authoring: Adapting Productivity Tools to Narrative Content Description. First Joint Conference on Interactive Digital Storytelling (ICIDS), Erfurt, Germany, November 2008. (2008) 285 – 296

(Pogamut, 2009)
AMIS: Pogamut. (2009) URL: http://artemis.ms.mff.cuni.cz/pogamut [27.1.2009]

(Prendinger & Ishizuka, 2004)
Prendinger, H., Ishizuka, M.: Life-Like Characters: Tools, Affective Functions, and Applications (Cognitive Technologies), Springer. (2004)

(Pulse!!, 2005-9)

Texas A&M-Corpus Christi: Pulse!! The Virtual Clinical Learning Lab. (2005-2009) URL: http://www.sp.tamucc.edu/pulse/home.asp [27.1.2009]

(Rao, 1996)
Rao, A.S.: AgentSpeak(L): BDI Agents speak out in a logical computable language. Springer Berlin/Heidelberg. (1996)

(RIT, 2005)
RIT: GameBots branch. (2005) URL: http://www.cs.rit.edu/~jdb/gamebots/ [27.1.2009]

(Shelley, 1818)
Shelley, M.: Frankenstein; or, The Modern Prometheus. Harding, Mavor & Jones (1818)

(Small, 2008)
Small, R.K.: Agent Smith: a real-time game-playing agent for interactive dynamic games. In: *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, Atlanta, GA, USA. (2008) 1839 – 1842

(Softimage, 2009)
Softimage Co. Avid Technology: Softimage. (2001–2009) URL: http://www.softimage.com [27.1.2009]

(Spierling et al., 2006)
Spierling, U., Weiß, S., Müller, W.: Towards Accessible Authoring Tools for Interactive Storytelling. In: Göbel, S., Malkewitz, R., Iurgel, I. (eds.) *TIDSE 2006.* LNCS, vol. 4326, Springer, Heidelberg. (2006) 169 – 180

(Stanley & Miikkulainen, 2002)
Stanley, K.O. & Miikkulainen, R.: Evolving neural networks through augmenting topologies. volume 10, Cambridge, MA, USA, MIT Press. (2002) 99 – 127

(Szilas, 2007)
Szilas, N.: BEcoll: Towards an Author Friendly Behaviour Narrative. In: *Proceedings of 4th ICVS*, LNCS 4871. Springer-Verlag. (2007) 102 – 113

(Štolba, 2008)
Štolba, M.: Rozhodovaci pravidla pro projekt Pogamut 2. Bachelor thesis. Charles University in Prague, Czech Republic, in Czech. (2008)

(Tence & Buche, 2008)
Tence, F., Buche, C.: Automatable evaluation method oriented toward behaviour believability for video games. In: *International Conference on Intelligent Games ans Simulation (GAME-ON'08)*. (2008) 39 – 43

(Tulving & Donaldson, 1972)
Tulving, A., Donaldson, W.: Organization of memory. New York: Academic Press (1972)

(van Waveren, 2001)

van Waveren, J.P.: The Quake III Arena Bot. Master of Science thesis, Delft University of Technology. (2001) URL: http://www.kbs.twi.tudelft.nl/Publications/MSc/2001-VanWaveren-MSc.html [27.1.2009]

(Wilensky, 1999)
Wilensky, U.: NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. (1999) URL: http://ccl.northwestern.edu/netlogo/ [27.1.2009]

(Wooldridge, 2002)
Wooldridge, M.: An Introduction to Multiagent Systems, John Wiley & Sons, Chichester, England. (2002)

(Woolf, 2008)
Woolf, B.P.: Building Intelligent Interactive Tutors: Student-centered strategies for revolutionizing e-learning. Morgan Kaufmann. (2008)

(XPprogramming.com)
XPprogramming.com: Extreme programming. URL: http://www.xprogramming.com/xpmag/whatisxp.htm [27.1.2009]

A. Appendix – A workflow example

This appendix shows how to create the first bot for UT2004 using Pogamut 3 in several steps. This bot features a Java-based ASM. Individual steps are demonstrated in the on-line tutorials (Pogamut, 2009).

1. Run Pogamut 3 (consult the on-line videotutorial)
2. Run UT2004
3. Add a new UT server instance to the IDE. Later, you will find your running bot in the corresponding *treeview*.
4. Create a new empty bot project. The IDE contains empty project templates with all the necessary files properly configured. After creating the empty project, you can instantly run the bot without any action selection mechanism and see it standing in the virtual world.

The newly created bot project will contain a new agent class that extends the base *UT2004Agent* class of the Pogamut 3 platform. This class contains several methods that needs to be implemented, most importantly:

- `createInitializeCommand()` – this method is used to set the agent's name, skin, skills, etc.
- `botInitialized( ConfigChange config, InitedMessage init )` – this method is called after the agent connects to the game
- `doLogic()` – this is the main method called periodically by GaviaLib; the bot's action selection mechanism is implemented here
- `botKilled( BotKilled event )` – this method is called each time when the bot is killed; the method is usually used for resetting internal variables

One of the simplest possible example bots available in Pogamut 3 is the *Follow bot*. This bot is programmed to follow any player it sees. The `doLogic()` method of the *Follow bot* is:

```
@JProp
boolean shouldTurn = true;

protected void doLogic() throws AgentException {
      // find visible friendly player
            Player player =
                  players.getVisibleFriends().values().iterator().next();
      // if the player exists
            if (player != null) {
                  // then follow him
                  locomotion.moveTo(player);
            } else {
                  // if none player is visible then turn around to find any
                  if(shouldTurn) locomotion.turnHorizontal(10);
            } // of else
      } // of doLogic
```

The `players` variable references the sensory module, which manages information about players in the game. The `locomotion` variable references motor module with methods for basic movement. These two modules are a part of the UT2004 integration, which extends the general GaviaLib library. Now, you can:

1. Execute the bot, check the bot's logs, and observe its behaviour directly in the game.
2. Return to the IDE and adjust values of parameters that have been previously marked with the `@JProp` annotation. Observe how this affects the bot's behaviour.
3. If need, pause the execution of the bot and use the arrows to get it to a desired location.
4. Decrease the game speed in Server control window and resume the bot. The game will run slower. Observe details of the bot's decisions in the logs.