POGAMUT 2 – A PLATFORM FOR FAST DEVELOPMENT OF VIRTUAL AGENTS' BEHAVIOR

Rudolf Kadlec, Jakub Gemrot, Ondřej Burkert, Michal Bída, Jan Havlíček, Cyril Brom
Charles University in Prague, Faculty of Mathematics and Physics
Dept. of Software and Computer Science Education, Prague, Czech Republic
rudolf.kadlec@gmail.com, jakub.gemrot@gmail.com, brom@ksvi.mff.cuni.cz
http://artemis.ms.mff.cuni.cz

ABSTRACT

Pogamut 2 is a freeware platform for rapid development of behavior of virtual agents embodied in a 3D environment of Unreal Tournament 2004. Pogamut 2 is specifically intended for research and educational purposes. A set of tutorials and videos makes it viable for newcomers.

In this paper, we describe unique features of Pogamut 2, which includes an "easy-to-cope with" integrated development environment, log management, and scripting language for running experiments. The platform architecture and intended workflow is also presented.

Keywords: Virtual agents, behavior, Unreal Tournament, 3D environment, agent development platform.

INTRODUCTION

Virtual agent is an embodied agent that is graphically represented by an avatar in the environment. The development of a complex behavior of a virtual agent acting in a human-like 3D world is in general very hard. There are two main reasons for this. Firstly, virtual agents act in a dynamic, unpredictable, interactive world. Secondly, virtual agent's goals can include many complex tasks – from movement in the environment to social interactions and emotional responses.

Nowadays, there are a lot of applications featuring virtual agents ranging from commercial computer games, through serious games (Aylett et al., 2005) to therapeutic tools (Hodges et al., 2001) and virtual storytelling (Cavazza et al. 2004). It is essential to have a high-quality agent development tool for fast prototyping of behavior of a virtual agent. Many different applications feature many advanced development tools, however, they are typically not donwloadable for public use. Moreover, those tools are domain-specific and cannot be used in general.

Therefore, newcoming students and researchers, who are concerned with virtual agents development are forced to use commercial (e.g. AI-Implant[1], SoftImage[2], Xaitment[3]), or freeware tools that facilitate agent development.

Commercial tools are inappropriate for newcomers in the field for several reasons. Firstly, they imply the knowledge of many advanced AI algorithms they implement. Secondly, they have to be connected to some virtual world and implementing this connection is not a trivial task. Moreover, they are quite expensive.

Examples of freeware tools include Gamebots (Adobbati et al. 2001) that is a plain interface to a game engine or F.E.A.R. (Champandard, 2003), which combines own interface and framework. Neither of them supply more advanced features like integrated development environment (IDE), introspection of agent's variables, log management, game control, etc.

Basically, a freeware platform that fits the requirements on fast prototyping of complex behavior of virtual agents is missing. In this paper, we present the toolkit Pogamut 2, which aims at filling this gap. It extends our previous work, Pogamut 1, providing these main features: (1) connection to a virtual environment, (2) auxiliary libraries – A*, sensory primitives, memory management, etc., (3) IDE with agent specific support for debugging, (4) built-in decision-making system (DMS) POSH (Bryson, 2001), (5) support for experiments defined by declarative rules.

Pogamut 2 is designed for research projects concerning investigating behavior of human-like virtual agents and the education of undergraduate students. Pogamut 2 uses the environment of the Unreal Tournament 2004 (UT04)[4]

1 Engenuity Technologies Inc.: AI-Implant.
URL: http://www.ai-implant.com [15. 6. 2007]
2 Softimage: XSI.
URL: http://www.softimage.com [15. 6. 2007]
3 X-AItment GmbH: X-Aitment.
URL: http://www.x-aitment.net [15. 6. 2007]
4 Epic Games: UnrealTournament 2004.
URL: http://www.unrealtournament.com [15. 6. 2007]

as a 3D simulator – a commercial[5] first-person shooter game. UT04 offers an adjustable 3D engine with a map editor and a lot of different locations, a library of predefined items, and modes with different rules of play. Use of UT04 as the 3D simulator can make the platform interesting also for the community of game players as we can find examples of the players experimenting with their favorite games[6].

We have already released the first stable version of Pogamut 2 which is available for download on our website[7] in the form of an installer. It offers all functionality discussed later plus examples and tutorials. The work on Pogamut 2 still continues. We are presently finishing some advanced features; specifically time-line debugging (review of agent's decision and surrounding environment during some interval of time), integrated map of the environment (3D map for visualization of multiple agents at once), and video tutorials.

GOALS

The Pogamut 2 attains to the following goals:

1. Extensibility and modularity – the code is modular and open-source thus allowing connection of different DMS (e.g. SOAR[8]) and extending the IDE.
2. User-friendly development environment – the IDE supports implementation, debugging and experiments.
3. Parallelization – client-server architecture separates UT04 and DMS, thus the load can be divided among multiple machines.
4. Steep learning curve – examples and video tutorials decrease starting time, this is vital for educational purposes.
5. Easy model validation – experiments allows for evaluation of the implemented model using rule based engine JBoss Rules [9]

We remark that the commercial tools (AI-Implant, SoftImage, Xaitment) fulfill these requirements but none of the freeware tools does (GameBots, F.E.A.R). Pogamut 2 is intended to fill this gap.

POGAMUT 2 - ARCHITECTURE

Pogamut 2 integrates six main modules: (1) UT04, (2) Gamebots2004 (GB04), (3) Parser, (4) Client, (5) IDE and (6) DMS.

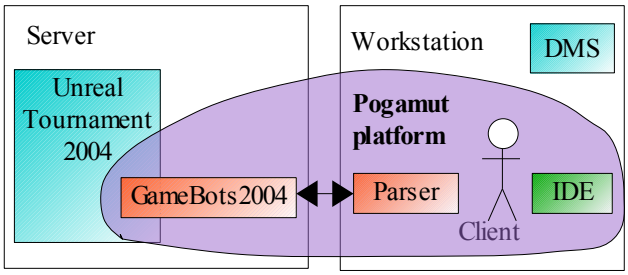

Fig. 1. Platform itself is composed of five main parts: UT2004, GB2004, Parser, Client and IDE. GB2004, Parser and Client cover the communication between DMS and an agent's avatar in the game. IDE serves for the development, debugging and experiments. It can also be used to observe multiple virtual agents in the simulation at once. Pogamut comes with DMS POSH, but various other DMSs can be easily connected.

Unreal Tournament 2004 is a commercial game, which is used as an environment for agents. It allows for connection of about 20 agents at once and contains an environment editor. UT04 is built on top of the Unreal Engine which provides open scripting language Unreal Script. This provides a way for extending the game with add-ons like the GB04, user-designed items, objects, animations etc.

The *Gamebots 2004* (GB) is a built-in server in the UT04, which takes care of the communication between a DMS and a virtual agent's avatar. Gamebots 2004 is our adaptation of the original GB, which was designed for the UT99. Additional functionality contains exporting of all items and navigation points at the beginning of the connection, automatic ray tracing, smooth movement along path, commands for replay recording, etc.

The *Parser* parses strings of the GB04 network protocol, creates a Java objects from them and sends them towards the Client. Its purpose is to optimize the communication between the GB04 and the Client; therefore it decreases the network load and allows for dozens of agents to be connected to the server at once.

The *Client* is a package of Java classes. It provides (a) a memory storing variety of sensory information, (b) commands, i.e. functional primitives for the control of virtual agent's body, (c) an inventory to manage items the agent picks up, (d) methods for movement around the map that are solving navigation issues, including A*.

IDE is made as a plug-in for NetBeans™ development environment[10]. It helps a developer during all important

5 The license costs about 15$ (more information can be found at http://www.unrealtournament.com)

6 Bots United, the bot community. URL: http://www.bots-united.com

7 Pogamut homepage URL: http://artemis.ms.mff.cuni.cz/pogamut

8 University of Michigan: SOAR URL: http://sitemaker.umich.edu/soar/home [15. 6. 2007]

9 Jboss Rules, JBoss division of Red Hat. URL: http://www.jboss.com/products/rules [15. 6. 2007]

10 Sun Microsystems, Inc: Netbeans. URL: http://www.netbeans.org [15. 6. 2007]

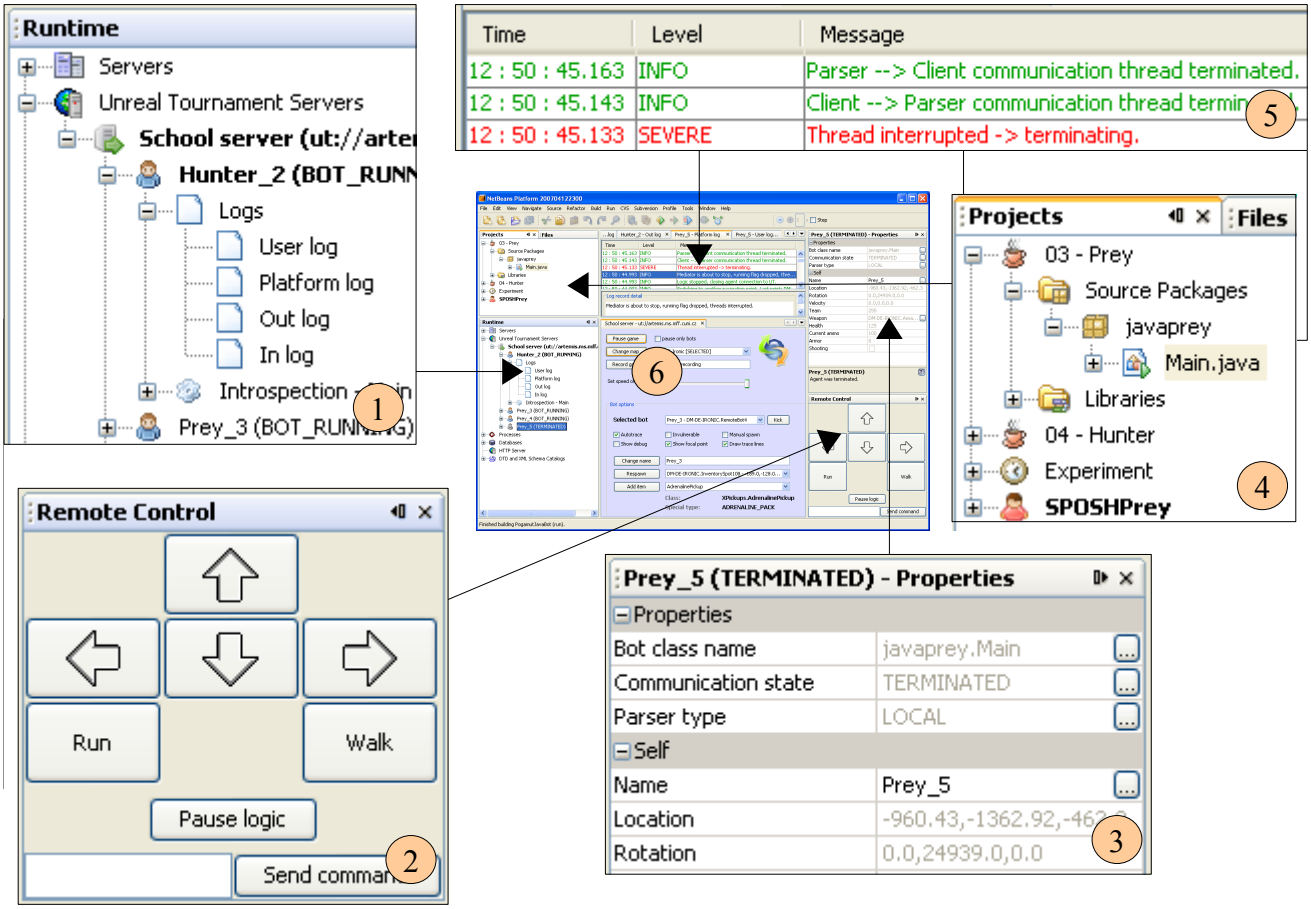

Fig. 2. A screenshot of the IDE during the debugging and the development

stages of work – development, debugging and experimenting. The IDE contains:

- Scripting of agent's behavior (Java, Python, POSH).
- Access to the library of virtual agents.
- Tools for debugging – an inspector of internal agent's variables, a viewer of agent's memory, viewers of logs of the DMS and communication, etc.
- Set of methods supporting scripted experiments, in which a user can define initial configuration, terminating conditions and interesting events to be logged.

HOW TO WORK WITH POGAMUT 2

Creating a virtual agent has these stages: (1) inventing the model, (2) implementing the model, (3) debugging the implementation, (4) tuning the parameters of the model and (5) experiments.

The IDE of Pogamut 2 was purposely designed to support the latter four of these stages. In this section we will illustrate how exactly Pogamut 2 achieves this.

Implementing the model

The platform currently supports the development of DMS in Java, Python and behavior-oriented language POSH (Bryson, 2001). The developer is using a high level API and doesn't need to care about the GB network protocol. A snapshot of a Java code demonstrating how API is used follows. The Java object *memory* provides access to the sensory primitives, while the Java object body accesses action primitives.

```
// do you see enemy? -> start shooting // hunt the enemy
if (memory.getSeeAnyEnemy() &&
    memory.hasAnyLoadedWeapon())
        { statePursue(); return; }
// are you shooting? -> stop shooting, you've lost your target
```

```
if (memory.getIsShooting())
        { body.stopShoot(); return; }
// are you being shot? -> turn around - try to find
your enemy
if (memory.getIsBeingDamaged())
        { body.turnHorizontal(355);
          return; }
```

The next example illustrates the control logic of a particular agent written in POSH. Basically, the agent is driven by three if-then rules: if see_enemy & hasBetterEapon then doRearm; if see_enemy & armed then engageEnemy; if stuck then jump.

```
(drives
  ((rearm
            (trigger (
                  (seeEnemy)
                  (hasBetterWeapon)
            ))
            doRearm
  ))
  ((engage
            (trigger (
                  (seeEnemy)
                  (armed)
            ))
            engageEnemy
  ))
  ((stuck
            (trigger (
                  (stuck)
            ))
            jump
  ))
)
```

The IDE also offers an editor with a project management (Fig. 2, window 4), syntax highlighting and code completion.

Debugging

Debugging using the IDE is composed of several parts.

- A list of running servers and agents (2.1) helps with a management of multiple agents.
- Agent s properties (2.3) give a quick access to variables common for all agents, e.g. position, velocity, orientation, health.
- Logs (2.5) display logged messages from communication between GB and Parser, logs from the platform and agent s logs (dedicated to the agent s DMS).
- The UT04 game server can be remotely administrated from (2.6).

The IDE also offers an option to change the speed of the simulation or even stop it. This feature helps the designer to consult logs on-line.

Tuning the parameters

Behavior models are typically quite sensitive to the settings of various parameters. The IDE allows the developer to set those parameters through the Introspection window without the need of restarting the agent (Fig 3.).

```
@PogProp boolean shouldCollectItems = true;
@PogProp boolean shouldCollectHealth = true;
@PogProp int healthLevel = 90;
```

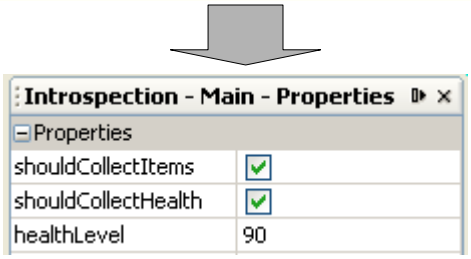

Fig. 3: Example of introspection of agent's parameters. All variables annotated with the *@PogProp* annotation are shown in the Introspection window and their values are periodically updated at runtime.

Experiments

Evaluation of the model is allowed by definition of the experiments in the IDE. This part of the IDE enables to model the intended situations and run it multiple times. The idea behind experiments is to allow a programmer to separate agent's logic and various test cases, similarly to so-called unit testing in Extreme Programming.
Every experiment is defined by a set of if-then rules. Therefore it is possible to define custom actions for various events that are based on agent's internal variables. For processing of if-then rules the JBoss Rules rule engine is used. It is possible to specify the kind of environment, the number of agents, their starting locations and their equipment. The IDE also features configuring breakpoints during an experiment.
An example of a rule follows. The rule fires whenever the agent Hunter sees the agent with ID rabbitID and reports whether Hunter is shooting at this agent

```
rule "Hunter spots Rabbit"
  when
    hunterMem : AgentMemory( name == "Hunter" )
```

```
  eval(
    hunterMem.seeEnemy(globals.get("rabbitID")
    )
  then
    if (hunterMem.isShooting())
      log.info("Hunter is shooting at the rabbit.")
    else
      log.info("Hunter can see the rabbit but is NOT
                shooting.")
end
```

DISCUSSION AND WORK IN PROGRESS

As said above, Pogamut 2 is intended for research on virtual agents and education of undergraduates. We are going to use it 1) for a research on emotions in action games, 2) as an educational tool for a course on modeling behavior of virtual agents taught at our faculty, 3) to develop specific scenarios for research on spatial navigation of humans using virtual reality environments. It is also possible to use the platform for modeling social interactions, cooperation of multiple agents in 3D environment, and for design of educational games and scenarios.

Pogamut's 2 main benefits are in integrated development environment, rich library of predefined methods for agent design and possibility to create agents using reactive planner POSH. Nevertheless there are some limitations. Maximum of twenty agents can be connected to the server at once, so it is not designed for mass simulations. Another issue is the flow of the time; the speed of the simulation can not be adjusted, which makes the platform impractical for evolutionary computing.

There are some additional features we are going to add soon. Those include more scripting languages and different DMSs (e.g. SOAR). Another feature will be a timeline. It will enable recording the simulation and then replaying what happened in the game simultaneously with listing the logs of the DMS.

CONCLUSION

We introduced a freeware platform for the development of virtual agents in a complex virtual world. The platform uses UT04 as the 3D simulator. The agent is provided with sensors, effectors, memory, inventory and the notion of the environment. The integrated development environment (IDE) features a variable inspector, viewers for logs, an agent's inspector, an agent's sensory memory, a script editor and others. Integrated together, Pogamut 2 forms a working platform suitable for rapid development of virtual agent's behavior.

The workflow with the tool is intended to be 5-staged: invention of the model (specification), implementation of the model, debugging the model, tuning it, and experimenting with it. The later four are supported by the IDE. Hence a developer is spared of tedious background work behind the communication, memory organization, representation of the objects in the game, etc. and can focus directly on his or her main problem.

Brought all together the main advantages of our platform are (1) freeware license, (2) complex tools that facilitate the development of virtual agents (IDE, debugging, etc.), and (3) easy work with the platform, which is supported by a set of tutorials and videos. Moreover, the platform is well documented and the platform architecture is designed to support further extensions. We believe that with all these features our platform fulfills the requirements on fast prototyping of complex behavior of virtual agents and fills the gap left by other freeware platforms.

Acknowledgements

This work is supported by the grant GA UK 1053/2007/A-INF/MFF and partially supported by grants GA UK 351/2006/A-INF/MFF and "Information Society" 1ET100300517.

REFERENCES

Adobbati, R., Marshall, A. N., Scholer, A., and Tejada, S. 2001 "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), URL: http://www.planetunreal.com/gamebots [15. 6. 2007]

Aylett R.S., Louchart S., Dias J., Paiva A., Vala M. 2005. "FearNot! – An Experiment in Emergent Narrative." In *Proceedings of Intelligent Virtual Agents*, LNAI 3661, Springer, 305–316

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

Cavazza, M., Charles, F., Mead, S.J. 2004. "Developing Re-usable Interactive Storytelling Technologies." In *Proceedings of the 2004 of IFIP World Computer Congress* (Toulouse, France).

Champandard, A.J. 2003: *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors.* New Riders. URL: http://fear.sourceforge.net [15. 6. 2007]

Hodges, L.F., Anderson, P., Burdea, G.C., Hoffman, H.G., Rothbaum, B. O. 2001. "Treating Psychological and Physical Disorders with VR." In *Proceedings of the 2001 of Computer Graphics and Applications*, IEEE 2001, 25-33.

Mateas, M., Stern, A. 2003. "Façade: An Experiment in Building a Fully-Realized Interactive Drama." In *Proceedings of the 2003 of Game Developers Conference*.