POSH Tools for Game Agent Development by Students and Non-Programmers

Cyril Brom†, Jakub Gemrot†, Michal Bída†, Ondrej Burkert†, Sam J. Partington‡ and Joanna J. Bryson‡

***Abstract*— Agent based systems are becoming popular outside the agents research community, among biologist, artists and in the game industry. Yet tools are lacking that facilitate non-expert agent developers building complicated agents for modelling and systems. As a part of our own agent-based research programmes we have been developing such tools. In this paper, we review the progress made, highlighting issues of usability. Examples of agents developed in these tools are also given, with an emphasis on intelligent virtual combat agents situated in Unreal Tournament.**

***Index Terms*— AI, Dynamic Planning, Life-Like Characters, Programmability, Accessible AI**

I. Introduction

Building intelligent systems, like programming systems in general, is hard. While some principles such as modularity are widely agreed upon, even highly related questions such as how to integrate these modules back into a coherent system, are not. Standard multi-agent system techniques are formal and highly distributed but unnecessarily complicated for modular systems where the extent of the system is well bounded, as when it composes a single autonomous agent like a game character or robot. The problem of this complexity becomes apparent when one tries to hire ordinary or even exceptional programmers to create AI systems. Yet the ideal for the games industry would be if skilled story writers for interactive dramas could directly create — or even prototype or adjust — the characters they design into their narratives.

We have been working on the developing and extending the notion of reactive or *dynamic plans* as the key integration technique for an intelligent system. The advantage of these plans is that they can be developed from simple sequences of actions and prioritised lists of goals — skills accessible to most people as they are fundamental to everyday planning and scripting.

Based on this approach, we have developed several complex, apparently cognitive agents, as well as tools that facilitate such construction. In this paper, we review our progress in developing these tools, with a particular focus on their educational aspects. Examples of agents developed in the tools will be also given, primarily VR game-playing agents situated in the video game Unreal Tournament (UT) [32], although related systems have been and are being developed for applications from autonomous robots to scientific social simulations. In this paper, the review of each advance will necessarily be brief, however we reference more complete descriptions of the individual projects which have been published elsewhere.

Our primary objectives for this paper are to present an overview and to summarise our main lessons learned to date. We start with detailing the toolkits' requirements, continue with describing our methodological approach, and then describe the tools themselves, including example agents.

†Department of Software and Computer Science Education, Charles University, Prague 118 00, Czech Republic

‡Artificial models of natural Intelligence, University of Bath, Bath BA2 7AY, United Kingdom

II. Goals and Related Work

Currently, our main target audiences for our toolkits have been undergraduate computer science students (Prague) and graduate-level researchers in social and behavioural sciences (Bath). For both groups of users, we hope that by engaging them in directly exploring agency we will contribute to both their understanding of their discipline and at the same time provide them tools to be used in their future employment. Building agents situated in dynamic, potentially antagonistic environments that are capable of pursuing multiple, possibly conflicting goals not only teaches students about the fundamental nature and problems of agency but also encourage them to develop or enhance programming skills. Although academics are our most readily accessible testers, we expect our techniques to be of use to professionals from artists through games engineers. Many of our tools are available freely on line, and we have recently installed basic bug tracking facilities. In essence we want to expand access to AI as a tool for research, entertainment and education.

Toolkits to meet this aim must fulfill several requirements:

1) They must provide tools for facilitating agent development. These tools must allow for making *simple things simple, and complex things possible*. In other words, a non-programmer should be facilitated in building simple agents, while more experienced developers should be able to increase their agents' complexity or capabilities. The best way to meet these desiderata is with a modular architecture, iterative design practice and an easy-to-cope-with integration technique. Auxiliary tools such as a debugger or a graphical behavioural editor should also be available.
2) In a related point, toolkits should facilitate rapid prototyping, so that a writer or creator can try a number of different scenarios quickly, and so that a full character with a number of goals and/or scenes can be fleshed out in a reasonable amount of time.
3) They should provide a graphical virtual environment. Visualisation is vital to understanding agency. Most people have a great deal of difficulty reasoning about

the consequences of interacting parallel goals and behaviours in a single agent; how an agent will operate with a three dimensional, real-time world, where actions once committed cannot be recalled; and particularly with how interacting agents can influence and interfere with each other, even in simple ways such as attempting to take the same physical location. Visualisation at least assists with debugging, and sometimes established social reasoning makes recognition if not prediction of problems relatively straight forward. The visualisation environment can either be provided built-in, or as a stand-alone application with a provided API.

There are several commercial, relatively user-friendly toolkits which at least begin to fulfil these requirements available. For example, AI.Implant [3], a middleware for building complex computer game agents, or Behavior [29], which is a Softimage plug-in for controlling behaviour of virtual actors in movies. Such systems tend to be too expensive for academic and entry-level purposes.

Several purely educational toolkits exist for learning programming by means of agents, such as Alice [2]. Unfortunately, these allow for building only simple agents with entirely scripted behaviour. NetLogo is a popular tool for agent-based modelling, partly because it meets the rapid-prototyping desiderata above. It has also recently become extendible due to a Java API [33]. However, this does not facilitate creating engaging single VR agents, which we see as a powerful and vital mechanism both for creating truly animal-like systems and holding the interest of the average (rather than the exceptional) student. Similar problems hold for agent development toolkits specially intended for artists, such as Movie Sand BOX [19].

Robust and formally well-founded tools for agent development do exist, such as the general-purpose Jack [17], or the powerful cognitive modelling languages like Soar [28] and ACT-R [4]. However, it is not easy for entry-level programmers to create engaging human-like agents in these architectures. Further, even for professional programmers, building intelligence in such 'heavy' systems takes a good deal of time. Such systems also tend to take too much CPU for complex societies or game play.

Thus there is still a need for systems which provide accessible development of virtual-reality animal-like and humanoid characters, but also allow extensions into full programming languages. What we propose as a basic starting point is a system built on the high-level yet powerful dynamic programming language python. Python is a scripting language which allows for rapid prototyping, yet it has access to extensive libraries which allow for powerful and detailed computation. Beyond this though, agent building requires special idioms or design patterns, and a character-based AI development platform should provide for these.

III. Approach: Behavior Oriented Design

We have taken as a starting point Behavior Oriented Design (BOD) [10, 11]. This is a modular technique that draws both from object-oriented design (OOD) and behavior-based AI (BBAI), with additional features for integrating an intelligent system. From BBAI, BOD takes the principle that intelligence is decomposed around expressed capabilities such as walking or eating, rather than around theoretical mental entities such as knowledge and thought. Each module supports a related set of expressed behaviours called *acts*, whatever sensing is necessary to control such acts, and whatever memory and learning is necessary to inform and disambiguate such sensing and acting. For example, the act of going home requires being able to sense and recognise the current location, which requires remembering previous routes or some other form of map. A diversity of standard machine learning techniques can be included inside a single agent: BOD supports efficient learning by allowing per-module specialization of techniques and representations.

From OOD, BOD takes both the object metaphor (BOD modules are built as objects in an object-oriented language such as Java, C++, CLOS or in the present case python) and an agile, iterative development process [c.f. 5]. BOD consists of two sets of heuristics. The first are for the initial design of an agent, and the second are for recognising — after a period of development — optimisation opportunities for simplifying the agent's code. In other words, BOD encourages regular refactoring so that the agent remains as easy to expand and maintain as possible. Details of these heuristics can be found elsewhere [10, 12].

The core of this simplification process is a good mechanism for integrating the behaviour modules that compose an agent. Modular decomposition has no benefit if the process of making certain the modules can execute without interfering with each other is more complicated than building a homogeneous architecture would have been in the first place. Unfortunately, this problem plagued early BBAI approaches such as subsumption architecture [9] and spreading activation networks [22], making them difficult to scale. BOD provides a relatively simple action-selection mechanism for providing behaviour arbitration, which we describe next.

A. POSH Action Selection

BOD uses Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) dynamic plans for action selection. These allow the specification of an agent's goals and priorities, or in other words the contexts in which an agent acts. The primitives of these plans are the acts supported by the library of behavior modules just described (referred to as *the behavior library*), as well as set of sense primitives provided by the same library. These sense primitives inform the plans at decision points about the current context. *Context* here is both environmental (e.g. visible food, heard enemies) and internal (e.g. remembering the way home, feeling hungry or happy.)

Besides these two primitive types, there are three types of POSH planning aggregates: simple sequences, competences and drive collections. The *sequence* functions as expected; the drive collection is a special form of competence that serves as the root of the plan hierarchy, we return to this below.

The *competence* is the core of a POSH plan. It is an instance of a fundamental AI design pattern, which we refer to as a

basic reactive plan [10]. This is essentially a small, prioritised set of productions. A *production* is a condition-action pair which forms the core of most expert systems and cognitive modelling architectures. The idea is that AI can be described as a look-up of an action based on the current context[1]. The problem is that specifying such context in sufficient detail to be unambiguous for an agent with multiple goals in a complicated, dynamic environment is either tedious and cumbersome (for a human programmer) or computationally intractable (for machine learning or planning.) By using a hierarchical structure, we can assume we know a large amount of the context by the fact we are even considering this particular plan fragment. For example, the agent has already realized that it is hungry and there are bananas around, now it just needs to peel one. Thus a competence only needs to express how to consummate one particular subgoal. By assuming the rules are prioritised, we can further assume that for any particular step of the plan, no better action closer to consummating goal is a available, or it would already have been executed. Thus for each potential action, the only thing that the context needs to describe is not whether it *should* be performed, but rather only whether it *can* be.

Because POSH plans are hierarchical, each 'action' as described above in a competence may be primitive acts supported by the behaviour library, or they may in fact be another competence or a sequence. At the root of this hierarchy is a special competence called a *drive collection* which is executed on every program cycle to ensure that there is no more important goal the agent should be attending to than the one it is currently attempting to complete. The drive collection also supports enough state such that goals can be pursued in coarse-grain parallel. Thus an agent could for example stand up every few seconds and look around, while spending most of its time concentrating on building a fire.

To small extent, POSH plans resemble languages built upon BDI architecture, e.g. JACK Agent Language [17]. However, POSH is especially designed for non-agent experts, which means that it does not have some advanced BDI features (e.g. pre-defined meta-level planning), but on the other hand it is easier for the intended audience to cope with. What makes it easy to design is that all of its aggregates can be designed initially as sequences. Competences are however capable of executing actions out of sequence (skipping or repeating elements) as needed to respond appropriately to the uncertain consequences of behaviour in a dynamic environment.

IV. Platforms

Based on the BOD approach and the POSH action selection mechanism, we have created three development systems fulfilling the requirements outlined in Section II. The first two each consist of two applications — first a pyPOSH action selection engine that implements a POSH mechanism and allows for BOD design [20], and second an environment for running experiments. The first of these pairings is BOD/MASON, which allows artificial life simulations. The second, Pogamut, is a platform integrating pyPOSH with the Unreal Tournament 3D environment and providing several development tools. The third system, IVE (for "intelligent virtual environment"), is a stand-alone application which is a complete simulator of large virtual worlds inhabited by tens of virtual humans [7]. We describe these three systems below.

A. PyPOSH and BOD/MASON

PyPOSH[2] is a POSH action selection mechanism built in the python language. It can be edited with the standard POSH plan editor, ABODE, and can be connected to any arbitrary system or virtual environment, e.g. a robot or an agent-based modelling tool. We have recently integrated pyPOSH with the agent-based modelling toolkit MASON [21], producing BOD/MASON [13]. This has two advantages: for novice agent programmers, it provides a platform and basic behavior libraries for simulating animal-like behaviour — BOD/MASON comes with a sheep/dog demo illustrating how different agents can be created with different POSH plans and the same behavior library. For agent-based modellers, BOD/MASON supports making more elaborate or individuated agent intelligence than most platforms, such as MASON on its own or NetLogo.

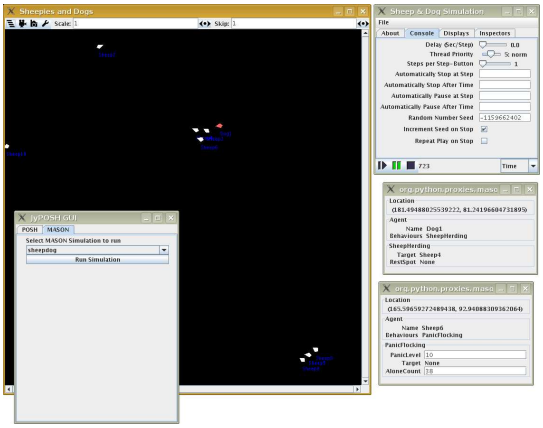

Fig. 1. A screenshot of BOD/MASON running the sheep/dog demo.

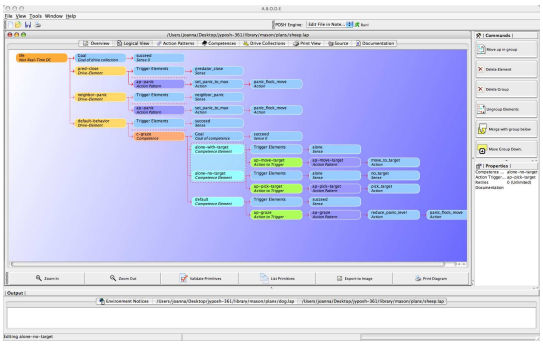

Fig. 2. A screenshot of ABODE editing a sheep's POSH plan.

[1]Many machine-learning approaches to AI make the same assumption and call the lookup structure a *policy*, which they attempt to learn.

[2]pyPOSH and ABODE can be downloaded from http://www.bath.ac.uk/comp-sci/ai/AmonI-sw.html. The pyPOSH distribution includes BOD/MASON and also another set of UT libraries (not Pogamut), which are described further below.

B. Pogamut

To provide additional tools facilitating games development, we have developed Pogamut[3] middleware and integrated it with pyPOSH and UT via the Gamebots interface [1]. The system architecture is depicted in Fig. 3. Each agent is treated as a triple (*avatar*, *behaviors*, *plans*), where *avatar* is merely a body driven in UT, *behaviors* are the set of behavioral modules in python maintained by Pogamut and *plans* is the set of plans by which pyPOSH controls behavioral modules.

The following tools are included in Pogamut:

- a simple agent management system,
- a debugger and a development environment for controlling agent's properties and communication with the environment (Fig. 4),
- a graphical editor for behaviors and dynamic plans called ABODE (Fig. 4), and
- a set of auxiliary class and basic behaviors, e.g. the navigation module.

The system can manage multiple agents simultaneously and can be easily plugged into another virtual environment, provided only with a gamebots-like interface (API) to that environment.

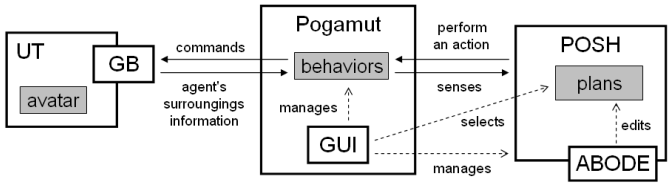

Fig. 3. Pogamut system architecture.

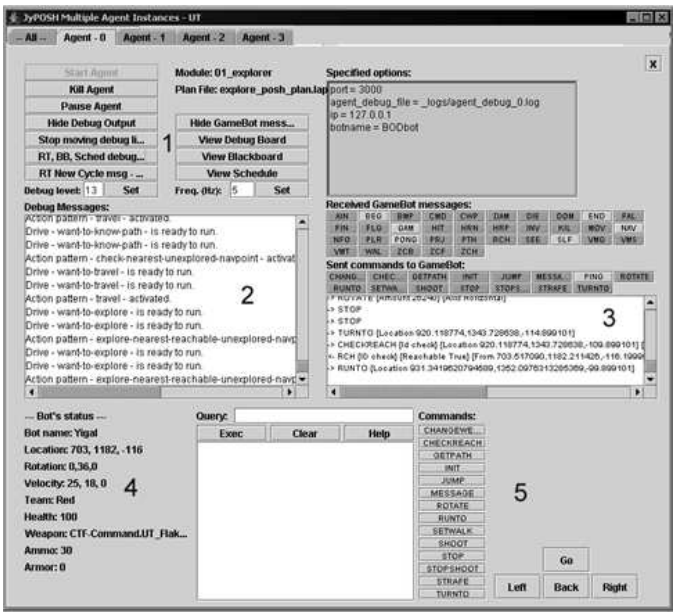

Fig. 4. Pogamut GUI: 1—the control pane. 2—the POSH log. 3—the Gamebots communication log. 4—the agent's properties. 5—the manual command pane.

[3]Pogamut can be downloaded from http://carolina.mff.cuni.cz/∼gib/. UT99 should be bought; it costs about 10 Euro.

In academia, UT with Gamebots have been used in several research projects already [18, 23], which makes it valuable for comparing AI approaches from different laboratories. The goal of Pogamut is to extend it to create a platform that can be used extensively by students and new programmers.

C. IVE

IVE[4] is a stand-alone Java framework and middleware supporting development of virtual human-like agents [7] (Fig. 5). IVE itself already includes development tools, i.e. a debugger, a neat GUI, and the action selection engine, which is based on a POSH extension. Virtual environments as well as behaviour of agents is specified in external XML and Java files.

IVE is specifically intended for simulations of large environments, which is its most notable distinction from pyPOSH / Pogamut (one can control hundred of actors in IVE, for example). There are several non-trivial issues stemming from large environments [detailed in 7] and the most features of IVE are designed to cope with these. These include:

- IVE uses the level-of-detail technique for automatic simplification of the simulation in unimportant places. Contrary to its typical use in the domain of computer graphics, we exploit this for simplifying the space and actors' behaviour [30].
- IVE exploits a knowledge representation that allows for adding new objects and actions into the environment in the runtime and for controlling actors both as autonomous agents or from a centralised director.

IVE is not the only recent work using LOD for behaviour [see e.g. 24]. Unlike other approaches, our technique is applied directly to the planning mechanism, which allows for gradual and robust simplification of the simulation that to our knowledge has not been addressed previously. For example, in our bar-drinking scenario, we allow for 4 degrees of detail of bar behaviour for each bar actor.

IVE can be used (and is being used) *as is*, both as a research and an educational platform. Current research includes:

- investigating level-of-detail AI techniques,
- simulating a virtual company
- augmenting IVE with a *drama manager* for the purpose of an educational virtual game in civics. Drama manager is a component for controlling agents' top-level goals according to a given story-plot (specified by Petri Nets as detailed in [8].)

V. Example projects

We have been using BOD-based systems on variety of projects, from controlling mobile robots to simulating primate task learning. In this section, we illustrate the systems' potential on three projects concerning gaming agents' AI. We start with a description of a gaming agent capable of juggling multiple goals that plays capture the flag. Next we describe an agent with an emotional module, which extends POSH action selection with an additional control layer. Finally we illustrate that our approach can be scaled for synchronising

[4]IVE can be downloaded at http://urtax.ms.mff.cuni.cz/ive.

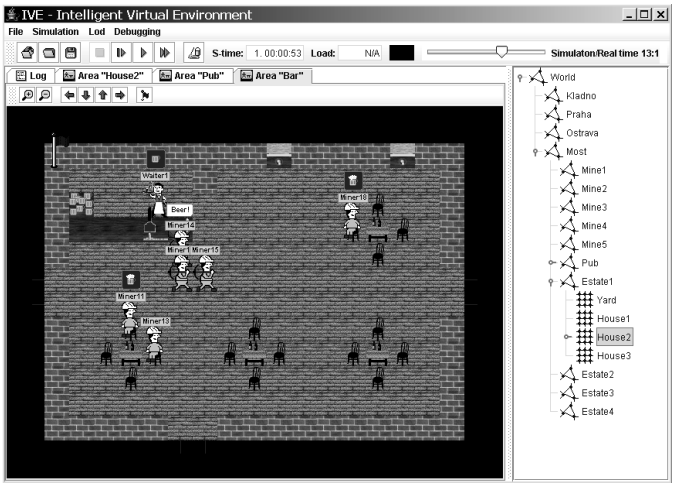

Fig. 5. A screenshot from IVE restaurant scenario. Miners and a waiter are depicted. Notice, there are tens of such actors in the scenario and yet the simulation runs on a single PC in a timely fashion. The pane on the right depicts the world structure.

two gaming agents. All these projects have been conducted by undergraduate students, which illustrates the tools' accessibility and educational potential.

A. Capture the Flag

The goal of this project was to test whether BOD scaled well for the agents acting in complex gaming environments, as well as ensuring that undergraduates can learn to use BOD. The full iterative development of this capture-the-flag gaming agent for Unreal Tournament is documented elsewhere [26]. The agent was coded using python and pyPOSH [20] directly, not with Pogamut. A slightly modified version of this agent (which can play in two-person teams) is currently distributed with pyPOSH.

Partington wrote a two-layer behaviour library, consisting first of four modules for expressed behaviour: *movement* and *combat* (behave as per their names), *status* (contains state regarding health level, weapons held etc.) and the class containing some *primitives* for UT communication. Additionally, there are three modules dedicated to maintaining internal state useful for more than one of the expressed behavior modules (e.g. information about position). He also developed one of the most intricate POSH plans to date (see Fig. 6) which allowed the agent to both attack and defend as necessary, sometimes at the same time (e.g. strafing an attacker while returning a flag to base). However, some of the complexity of the final bot plan was unnecessary — dedicated to timing out outdated memories. This indicates we need to clarify idioms dealing with memory and time to improve the BOD framework.

B. Emotional gaming agent.

In modern games, the key feature is agents *believability*, which simply stated means *the extent to which the players think the agents look and behave how the players expect*. It includes whether the agent acts like a human (if it is human-like), perceives only things a human could perceive, etc. Note,

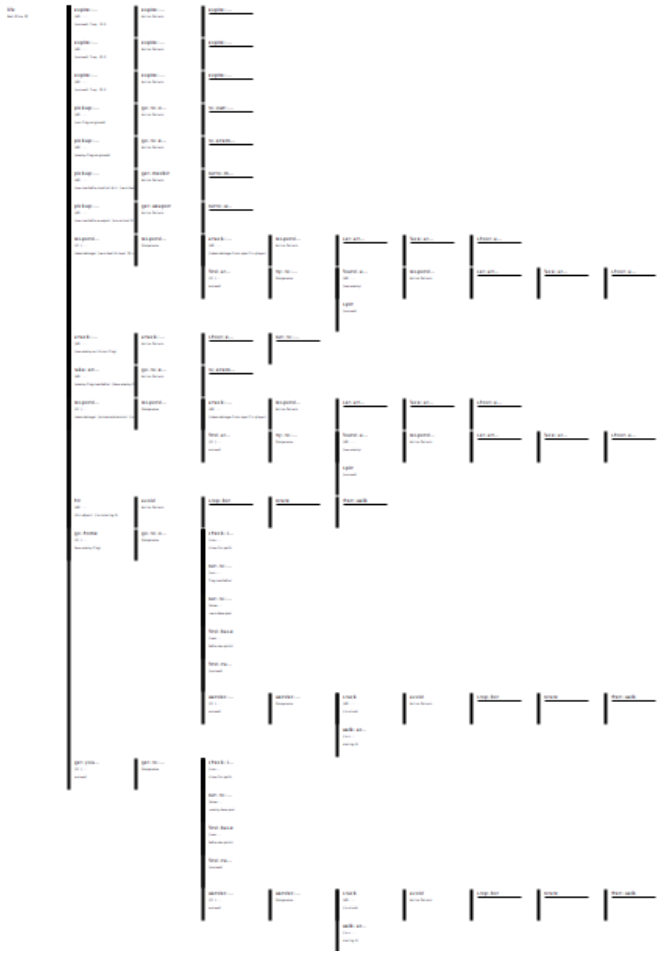

Fig. 6. A high-level view of Partington's capture-the-flag POSH plan showing the extent and limits of plan depth necessary for a complete game agent. Plan details, while not legible here, are available elsewhere [26]; the complete plan is also included in the (free) pyPOSH download.

however, that believability is more related to imitation than to rational reasoning or psychologically plausibility.

The hypothesis behind this project was that expressing emotions at the behavioural layer (i.e. not only by facial changes) may increase believability of a first-person shooter (FPS) game agent. We developed the emotional UT agent using the Pogamut platform. An emotional module is added in the BOD architecture. The module is based on Champandard's model [16], which is intended for such game agents. Champandard's model is partially based on Plutchik's psychologically grounded model [27]. Champandard also prototyped an emotional game agent, however ours is more elaborate.

The emotional model uses eight emotions in complementary pairs: *pride – shame*, *fear – anger*, *joy – sorrow*, *amusement – weariness*. It also exploits moods, feelings and sensations, which are other affective states of "minds". The outcome of the emotional module is threefold. First, different emotions enforce different dynamic plans (and thus behaviors). Second, emotions influence agents' properties (i.e., shooting accuracy). Third, the agent gesticulates and comments on the situation according its emotional state.

Preliminary tests reveal first that the model is too compli-

cated and overly psychologically plausible for the purpose of a UT agent. Since a typical UT agent lives for tens of seconds, the agent does not have five minutes for grief, even if that is more human. Second, we have realized that it is hard to parameterise the model without habituation, which is a form of adaptation to repetitive stimulation by increasing probability of ignoring it. Although POSH supports primitive habituation, neither our nor Champandard's model works with it. Appropriate dynamics are vital even for a simple, non-plausible emotional model [31], thus BOD should also probably be extended with an pattern for supporting such state. Third, it was extremely simple to layer emotions upon pyPOSH as well as implement different plans for expressing different emotions, which demonstrated scalability of the basic BOD architecture. We now plan to simplify the model and to conduct a psychological study of players playing against agents with and without emotions.

C. Agent Twins

Several FPS platforms include a *team deathmatch* mode, where agents and players fight in teams against each other. Although agents in one team might cooperate, this is not always the case. In recent FPS games, cooperation often occurs accidentally as an 'emergent' phenomenon only. The goal of this project was to develop agent twins that cooperate in a decentralised manner (i.e. each twin is self controlled; there is no leader) and test whether the cooperation is fruitful. Additionally, we wanted to verify POSH/BOD approach in a multi-agent scenario[5]. The twins have been developed on the Pogamut platform. Generally, they cooperate in two ways: (a) they perform some tactical operations, and (b) they inform each other about positions of objects and other agents. Finally, we have tested the twins of type (a), and (b), and the twins that do not cooperate at all in the deathmatch against each other and against original UT bots (who do not cooperate).

The tests surprisingly showed that cooperation by information passing (b) was fruitful, but that of tactical operations (a) was not. We think the reason is that our cooperation was intended to be plausible. However, the deathmatch in UT is not plausible at all; UT is not a realistic combat simulator. We have stumbled here on another "plausibility — believability" tension. We have also demonstrated that tests themselves can be easily managed in the Pogamut system, since pyPOSH is flexible enough to allow for simply switching on/off different behaviours and types of cooperation.

We have also shown that POSH does not cope well with expressing of certain more complicated behaviours, particularly the human adult capacity for finishing (or at least putting into order) a current task when motivation has switched so that another task is now the primary goal. This is a basic problem for dynamic planning. See [6] for details; some of these are being addressed in IVE.

We plan to conduct a study for more than two agents and to augment this work with some aspects of centralised reasoning and classical planning, which has recently attracted gaming industry attention (e.g. [25]). The project is detailed in [15].

[5]The version of pyPOSH currently distributed includes a two-agent (offence and defence) team version of Partington's UT code, developed at Bath by Steve Couzins.

VI. Summary; Present and Future Work

In this paper, we briefly introduced several agent-based development tools — BOD/MASON, Pogamut and IVE. These are systems which provide entry-level development humanoid and animal-like characters, which can be used by students and other non-professional programmers. The systems are extendible, grounded in standard programming languages. We have demonstrated this in the described research. This scalability is a notable distinction from similar toolkits.

We have at least a dozen finished students' projects using these platforms, and more in progress. These projects not only demonstrate that the systems can be used by the students, but also verify that BOD and POSH, which the projects and platforms are built upon, are accessible, flexible and scalable. We have also used the platforms for education of non-AI experts (including artists).

In addition to the observations made earlier on improving BOD and/or POSH, current limitations of our systems include that we have not tested the platforms extensively on non-AI experts yet. In IVE, we do not have a neat editor yet, which makes specifying new virtual worlds and behaviour of agents slightly complicated for non computer scientists. The editor is current work. In Pogamut, perhaps the main problem is that it relies on Gamebots interface [1], which limits the amount of information passed from UT. Additionally, Gamebots code is not well optimised and has several bugs. Rewriting Gamebots is another work in-progress; we have already fixed some of the bugs. We also plan to incorporate Gamebots UT 2007.

Acknowledgement.

This work is partially supported by several grants. For Cyril Brom and Prague GA UK 351/2006/A-INF/MFF and "Information Society" 1ET100300517. For Joanna Bryson and Bath the EPSRC GR/S79299/01 (AIBACS). Jan Drugowitsch and Tristan Caulfield have assisted with BOD/MASON as developers and Hagen Lehmann as a naive tester.

References

[1] Adobbati, R., Marshall, A. N., Scholer, A., and 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)

[2] Alice 3D Authoring system. Alice v2.0. Carnegie Mellon University. Project homepage: http://www.alice.org/ [4th Aug 2006]

[3] AI.implant. EngenuityTechnologies, Inc. Product homepage: http://www.biographictech.com/ [4th Aug 2006]

[4] ACT-R. A cognitive architecture. ACT-R Research Group. Carnegie Mellon University Project homepage: http://act-r.psy.cmu.edu/ [11th Aug 2006]

[5] Extreme Programming Explained: Embrace Change. Addison Wesley (1999)

[6] Brom, C.: Hierarchical Reactive Planning: Where is its limit? In: Proceedings of MNAS: Modelling Natural Action Selection. Edinburgh, Scotland (2005)
[7] Brom, C., Lukavský, J., Serý, O., Poch, T., Safrata, P.: Affordances and level-of-detail AI for virtual humans. In: Proceedings of Game Set and Match 2, Delft (2006)
[8] Brom, C., Abonyi A.: Petri-Nets for Game Plot. In: Proceedings of AISB: Artificial Intelligence and Simulation Behaviour Convention, Bristol (2006) III, 6–13
[9] Brooks, R.: Intelligence without reason. In: Proceedings of the 1991 International Joint Conference on Artificial Intelligence, Sydney (1991) 569595
[10] Bryson, J.: Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD thesis, Massachusetts Institute of Technology (2001)
[11] Bryson, J., Stein, A.L.: Modularity and Design in Reactive Intelligence. In: Proceedings of IJCAI'01 (2001)
[12] Bryson, J.: The Behavior-Oriented Design of Modular Agent Intelligence. In: Mueller, J. P. (eds.): Proceedings of Agent Technologies, Infrastructures, Tools, and Applications for E-Services, Springer LNCS 2592 (2003) 61-76
[13] J. J. Bryson, T. J. Caulfield, and J. Drugowitsch, "Integrating life-like action selection into cycle-based agent simulation environments," in *Proceedings of Agent 2005: Generative Social Processes, Models, and Mechanisms*, M. North, D. L. Sallach, and C. Macal, Eds. Chicago: Argonne National Laboratory, October 2005.
[14] Bryson, J.J., Prescott, T.J, Seth, A.K. (edited conference proceedings): Modelling Natural Action Selection: Proceedings of an International Workshop. AISB, UK (2005)
[15] Burket, O.: Unreal Tournament Twins. Bachelor thesis. Charles University in Prague, Czech Republic (2006) (in Czech)
[16] Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders (2003)
[17] JACK toolkit. Agent Oriented Software Group. Project homepage: http://www.agent-software.com/shared/home/ [4th Aug 2006]
[18] Kaminka, G. A., Veloso, M. M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A. N., Scholer, A. and Tejada, S., "GameBots: A flexible test bed for multiagent team research," *Communications of the ACM*, 45(1):43–45 (2002)
[19] Kirschner, F.: Movie Sand BOX. Project homepage: http://www.moviesandbox.com/ [4th Aug 2006] (2006)
[20] Kwong, A.: A Framework for Reactive Inteligence through Agile Component-Based Behaviors. Master's thesis, Department of Computer Science, University of Bath (2003)
[21] S. Luke, G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus, "MASON: A Java multi-agent simulation library," in *Proceedings of Agent 2003: Challenges in Social Simulation*, D. L. Sallach and C. Macal, Eds. Argonne, IL: Argonne National Laboratory, 2003, pp. 49–64.
[22] Maes, P. The agent network architecture (ANA). In: SIGART Bulletin, 2 (4) (1991) 115-120
[23] Muñoz-Avila H., Hoang H.: "Coordinating Teams of Bots with Hierarchical Task Network Planning," in *AI Game Programming Wisdom I*, S. Rabin, Ed. Charles River Media, Inc., Hingham, Massachusetts (2006)
[24] O'Sullivan C., Cassell J., Vilhjálmsson H., Dingliana J., Dobbyn S., McNamee B., Peters C., Giang T. "Level of Detail for Crowds and Groups," in *Computer Graphics Forum*, 21(4):733–742 (2002)
[25] Orkin, J.: 3 States and a Plan: The AI of F.E.A.R. Game Developer's Conference Proceedings. San Francisco, CA (2006)
[26] Partington, S.J., Bryson, J.J.: The Behavior Oriented Design of an Unreal Tournament Character. In: Proceedings of IVA'05, LNAI 3661, Springer (2005)
[27] Pluchick, R.: Emotion: A Psychoevolutionary Synthesis. Harper and Row, New York (1980)
[28] Soar architecture. University of Michigan, USA. Project homepage: http://sitemaker.umich.edu/soar [4th Aug 2006] (2006)
[29] Softimage/Behavior. Softimage Co. Avid Technology. Homepage: http://www.softimage.com/ [4th Aug 2006]
[30] Serý, O., Poch, T., Safrata, P., Brom, C.: Level-Of-Detail in Behaviour of Virtual Humans. In: Proceedings of SOFSEM 2006: Theory and Practice of Computer Science, LNCS 3831, Czech Republic (2006) 565–574
[31] E. A. R. Tanguy, P. J. Willis, and J. J. Bryson, "A dynamic emotion representation model within a facial animation system," *The International Journal of Humanoid Robotics*, 2006.
[32] Unreal Tournament. Epic Games, Inc. Product homepage http://unrealtournament.com [4th Aug 2006]
[33] Wilensky, U. NetLogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL. Project homepage: http://ccl.northwestern.edu/netlogo/ [4th Aug 2006] (1999)

Cyril Brom is a PhD candidate situated at Charles University, Prague. His research interest is in modelling artificial environments and behaviour of human-like artificial agents. Additionally, he teaches courses on modelling and computer games development and supervises about a dozen undergraduate students, whose theses concern gaming AI. He holds Magister (Master equivalent) in computer science and optimisation from the Faculty of Mathematics-Physics, Charles University in Prague.

Jakub Gemrot is a Magister (Masters equivalent) candidate situated at Charles University, Prague. His interests are artificial environments and software engineering. He holds a Bachelors in computer science from the Faculty of Mathematics-Physics, Charles University in Prague.

Michal Bída is a Magister (Masters equivalent) candidate. His interests are artificial intelligence in computer games, especially those featuring 3D virtual environment, artificial emotions and psychology. He holds Bachelor in computer science from Faculty of Mathematics-Physics, Charles University in Prague.

Sam Partington undertook a major evaluation of Behavior Oriented Design as part of studies at the University of Bath. His interests include both the practical and theoretical aspects of this methodology and of AI in general. Having obtained a BSc from Bath in 2005, he currently works in Software Development for RM Consultants, a company in Abingdon, Oxfordshire, UK.

Ondrej Burket is a student of the masters program at Charles University, Prague. His main interest is in gaming artificial intelligence, his last work concerned a couple of cooperating agents in UT. He holds Bachelor degree in general computer science from the Faculty of Mathematics-Physics, Charles University in Prague.

Joanna Bryson lectures computer science at the University of Bath; she founded her research group there in 2002 following a postdoc in the Primate Congitive Neuroscience Laboratory at Harvard. Her research interests include increasing access to building intelligent systems and using these techniques to model intelligence as found in nature. She holds a BA in Psychology from Chicago, Masters in AI and Psychology from Edinburgh, and a PhD from the MIT AI Laboratory.