EmohawkVille: Towards Complex Dynamic Virtual Worlds

David Holaň, Jakub Gemrot, Martin Černý
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské náměstí 25, 118 00, Prague 1, Czech Republic
E-mail: david.p.holan@gmail.com, gemrot@ksvi.mff.cuni.cz, cerny.m@gmail.com

KEYWORDS
Interactive 3D Virtual Worlds, Intelligent Virtual Agents, Action Selection

ABSTRACT

Several recent successful computer games feature a large 3D open world where the player has a high degree of freedom and may roam freely through the environment. While graphical fidelity of contemporary open worlds is stunning, the behavior of non-player characters (NPCs) is kept rather simple. This is an opportunity for the academia to develop novel techniques and tools that would allow for easier creation of complex behaviors that are interactively believable — i. e. that remain believable despite unpredictable world changes stemming from player's actions. However, state-of-the-art techniques — both in academia and industry — are usually deployed only in 3D game environments with low complexity — i.e., the number of NPC actions is kept small. We think that if this complexity is raised, the state-of-the-art techniques will be challenged and will have to be improved if not transformed totally. However, no research-friendly complex 3D virtual world is publicly available. In this paper, we present EmohawkVille: an open-source first-person 3D virtual world that allows for simulation of a day-to-day life of NPCs and compare it with various environments already available. We discuss general issues arising during developing behaviors for the EmohawkVille domain. We also present a qualitative case study that confirms suitability of EmohawkVille for experiments concerning NPC behaviors.

INTRODUCTION

Many contemporary computer games take a great effort to achieve a high level of believability of their virtual worlds. This is especially true for games with large open world, where the user is free to discover the environment on his own and is relatively unconstrained by the game. Due to time and resource constraints and to the difficulties neccessarily involved in behavior design, non-player characters (NPCs) display complex behaviors only during crucial game events. In between, the NPC behaviors are rather schematic. The sole exception to this rule is the combat behavior, which is usually well designed and reasonably complex. But even recent and succesful open-world games such as Skyrim (Bethesda Game Studios 2011) or Red Dead Redemption (Rockstar Games 2010) have resorted to severely limited non-combat NPC behaviors.

The reason is that going beyond this simple behavior and still maintaining the suspension of disbelief introduces significant difficulties to the NPC behavior authoring. The designer cannot just script a peasant to go to a field and perform a harvesting animation — the player would expect the harvest to progress and eventually end with an empty field. The progress of the harvest cannot be scripted linearly because if a player action causes the peasant to be delayed on his way to the field or not to arrive at all, the harvest should not continue.

Even a very simple task, such as the harvest scenario, needs to take into an account many possible obstacles, if it is to become beliavable. What if the player steals the scythe from the peasant? What if a fight starts near the field? How about seasonal changes? Even navigating properly from a village to the field might be a challenge if there are dynamic obstacles, e.g. a chariot or band of villains. If such possibilites are not taken into account, the NPCs are easy for the player to "break" and may provide even worse illusion of a real world than rather static NPCs.

The harvesting scenario is just a single example. For a truly alive open world, dozens of different and often much more complex scenarios are needed, which implies that the world needs to be equipped with a rich ontology of items and actions NPCs (as well as the player) can perform.

As the world ontology grows, the number of meaningful NPC action sequences increases and the behavior complexity rises. Not only the means-ends analysis becomes more demanding, new problems emerge such as transitional behaviors, joint behaviors, behaviors ordering or behaviors interleaving (Plch 2009). At the same time, game studios usually cannot afford to let an expert programmer design such day-to-day behaviors, because that would be cost-prohibitive. Most of the NPC design is usually carried out with the aid of some visual tool by scripters with little programming experience.

At this place, academia could provide action selection mechanisms (ASM) and accompanying tools that would

help inexpert scripters to create complex behaviors that are interactively believable, that is, behaviors that sustain their believability under non-determinism brought by the player. However, most of the academic research is carried out in environments that either have simple ontologies or are static or discrete. Games on the other hand are dynamic, multi-agent environments that can be for all practical purposes considered continuous in both time and space. There are languages and techniques that can be applied to such worlds either from the multiagent community or the field of robotics or automated planning. However, to our knowledge, there is currently no 3D virtual world publicly available that would provide rich ontology for NPCs out of the box. This means that in this particular problem area, academia is one step behind the industry — we do not even have an environment to work with.

In this paper, we present our extension of the Pogamut 3 platform (Gemrot et al. 2009) called EmohawkVille[1], the first step towards an open-sourced complex simulation of NPC everyday lifes in 3D virtual world. We believe that creating a fully working, accessible and polished environment fosters academic progress. The large amount of research work evaluated on our first environment — Pogamut for Unreal Tournament 2004 — supports this view. We have also exerted great effort to make Pogamut and EmohawkVille a mature tool and we have resolved a large part of the technical issues encountered during NPC behavior development (sensors and actuators interface, navigation and pathfinding, character animation support, etc.).

The structure of the paper is as follows. Firstly, we go through available virtual worlds. Secondly, we present our newly created virtual world EmohawkVille together with a library that allows to create NPC behaviors in Java. Thirdly, we present a concrete implementation of complex behavior in EmohawkVille and report on an user-case study of the library with four programmers. Finally we discuss the suitability of the current version of EmohawkVille for academical research.

RELATED WORK

To experiment practically with NPC behaviors means to find a suitable virtual world first. Firstly, the world should be extensible with new items, game mechanics etc. as NPC behavior experiments typically require unique settings. Secondly, the world should support human players which is especially important for interactive storytelling or immersive projects. Thirdly, it should be possible to code the NPC behavior in a standard programming languages (such as C++, C#, Java etc.) either directly or via a foreign function interface. It is impractical to create behaviors in a sandboxed scripting language that is native to the chosen virtual world (e.g. Papyrus of Elder Scrolls V: Skyrim) as the availability of 3rd party libraries (e.g. Prolog support) for such languages is severly limited. Fourthly, the world mechanics should be easy to understand even for people with little gaming experience. Fifthly, we are not concerned with virtual worlds that put an NPC into a role of some disembodied entity such as a commander of an army in the world of StarCtaft: Brood War — the ASMs for such worlds are fundamentally different from behaviors of emboddied NPCs. Sixthly, price of the software should be considered. Finally, we are omitting raw frameworks like Unity, Unreal Development Kit or CryEngine, or even more basic 3D game engines such as Ogre or Irrlicht, as it requires considerable amount of time to build a virtual world from scratch.

We continue with the list of virtual worlds of interest and conclude the section with comparison of their traits.

ADAPT. ADAPT (Shoulson et al. 2013) is a platform for designing and authoring human characters. It mainly focuses on animations so interactivity is not sufficiently detailed. Furthermore, the tool relies on Unity Pro (Unity Technologies 2005), which is not available for free.

CAROSA. CAROSA (Allbeck 2010) models behaviors of virtual agents and seems to be focused on crowd simulations. Actions are defined via Parametrized Action Representation (PAR). The agent behavior is specified by its schedule, groups and roles, i.e., a data-driven approach. Unfortunately the tool seems to be unavailable outside the parent institution.

Bethesda games. Recent role-playing games by Bethesda Softworks, such as The Elders Scrolls V: Skyrim, feature somewhat detailed interaction. However, the scripting language Papyrus does not provide foreign function interface and offers considerably less expressive power than modern programming languages.

MMOs. There is a long tradition of character control software targeting popular MMO worlds, such as SecondLife or World of Warcraft. Unfortunately, usage of such software is usually against the terms of service of the respective game. MMO games also predominantly use a client-server architecture with the server application being unavailable.

The Pogamut platform. The Pogamut platform aims at providing Java interfaces for various games. The most prominent is the Unreal Tournament 2004 (UT2004) module (Gemrot et al. 2009). UT2004 is a first-person shooter (FPS) and as a such only actions related to combar are available.

The Pogamut platform also provides bindings for a custom city-like environment for Unreal Engine 2: Runtime developed for StoryFactory (Bída et al. 2011) project. Our EmohawkVille uses many assets originally made for StoryFactory. While UE2 is available for free, its license effectively prohibits creation of virtual worlds supporting human players.

[1]Can be downloaded at `http://pogamut.cuni.cz/main/tiki-index.php?page=EmohawkVille`

Table 1: Table of interesting features of various virtual worlds. EV stands for EmohawkVille. Availability is listed with respect to academic and educational use.

Virtual World	EV	UT2004	UE2	ADAPT	CAROSA	Skyrim	CTF
World Complexity	High	Average	Empty	Empty	High	High	Low
Dynamicity	High	Average	N/A	N/A	High	High	Average
Availability	Free	Paid	Free	Paid	Restricted	Paid	Free
Extensible	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Setting	Everyday Life	Armed Combat	N/A	N/A	Everyday Life	Fantasy	N/A
NPC's role	Civilian	Soldier	N/A	N/A	Civilian	NPC	N/A
NPC's Language	Java	Java	Java	C#	PAR, Tables	Sandboxed Script	Python
Human player	Yes	Yes	Yes	No	No	Yes	No

The relevant properties of the aforementioned worlds are shown in Table 1. Besides EmohawkVille, no virtual world possesses all the qualities we seek.

EMOHAWKVILLE

EmohawkVille is a FPS virtual world with detailed interactive elements of day-to-day life. For example, an agent can pick up a cucumber, put it on a chopping board and then start slicing it with a kitchen knife. EmohawkVille is based on Unreal Development Kit (UDK) (Epic Games Inc. 2009) and thus is capable of displaying the world in state-of-the-art graphics. UDK is free for educational and non-commercial use and EmohawkVille itself is available under GPLv3.

Architecture overview

Designing an environment for AI evaluation always (explicitly or implicitly) involves two parts: the actual mechanics that govern the virtual world and the interface the AI uses to perceive the state of the world and issue actions. In EmohawkVille the world mechanics are implemented in UnrealScript - a proprietary language deployed with the UDK toolkit. A low-level interface to the world is exposed through TCP/IP communication with a derivation of the GameBots protocol (Bída et al. 2012). The Pogamut platform provides a high-level Java interface for writing the actual AI and takes care of many common tasks (navigation, caching sensory data, etc.). Both the UDK and the Java part have been designed with possible further extensions in mind and the basic NPC support is separated from the model of the general EmohawkVille ontology which is in turn separated from the implementation of the specific mechanics for our cooking scenario (explained later). The UDK part also fully supports interaction with a human user through the UDK visual client.

Ontology

In this section we explain various aspect of the EmohawkVille ontology: it starts with general concepts and continues with a brief introduction to specific objects, actions and object-to-object interactions for our first EmohawkVille scenario.

We have decided to describe the ontology in great detail, as these "details" such as action durations or action interruption make an NPC behavior specification difficult in practice.

General Concepts

This section deals with generic concepts that govern EmohawkVille: avatar, action, simulation, process, item and container. A simplified class diagram is given in Figure 1.

Avatar. An avatar is a physical representation of an NPC or a human player in a virtual world.

Action. Actions performed by avatars either directly alter state of one or more objects or start a process that represents a durative action (see below). For example, a player or an NPC can set the power of a stove plate. EmohawkVille can be extended by providing custom character animation for any action.

Simulations and processes Simulations represent spontaneous changes of the world state such as a physical process taking place.

A process is a simulation of a long running action performed by an avatar. A process can be stopped by the interrupt action, e.g., an NPC may stop chopping the cucumber as soon as it chopped enough cucumber slices or because the water starts boiling in the pot.

Item. Items are lifeless objects that are meaningful for interaction. An item in EmohawkVille can either be directly present in the environment or it can be contained in an inventory of a possessor. Items can be dynamically created and destroyed. For example, the result of

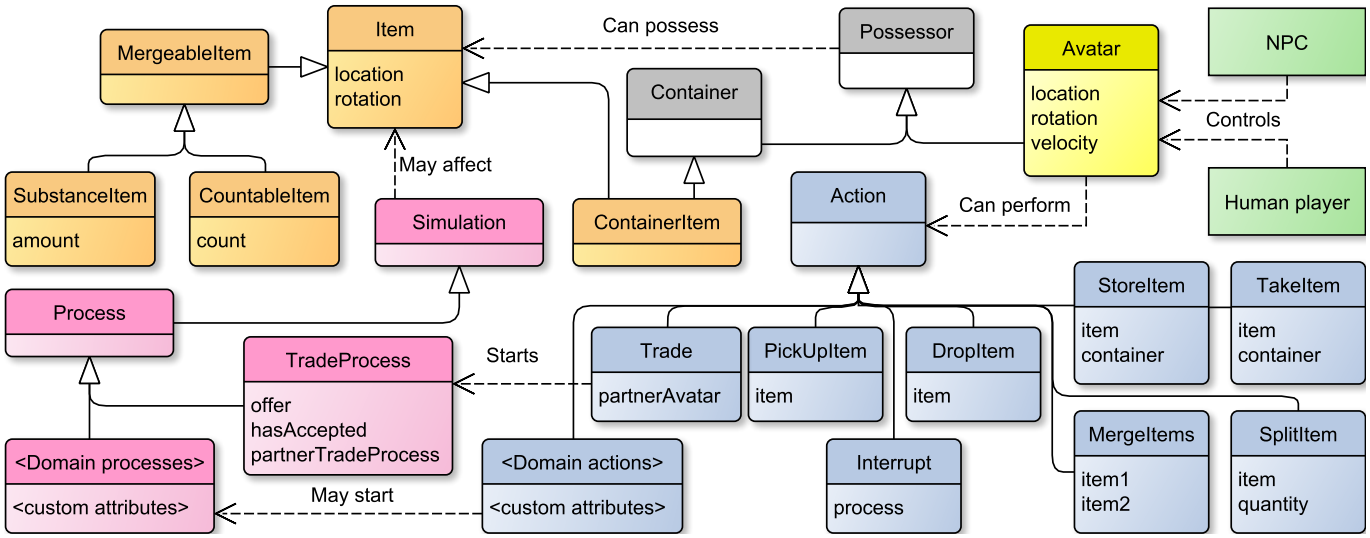

Figure 1: Class diagram of EmohawkVille's most important general concepts.

chopping a cucumber is the destruction of the cucumber and the creation of cucumber slices.

Possessor and container. A possessor is an entity that can own items. Possesors are either avatars or containers. In contrast to avatars that are active in the world, containers are passive objects (a pot, a wardrobe, etc.). Containers may constrain the amount/type of objects they store.

There are actions to pick up items, trade items with another avater, drop items in the environment or store them in a container.

Specific items and actions

The overall theme of EmohawkVille so far is cooking.

At this moment, EmohawkVille features 20 item types and the stove. Interaction is provided by 14 actions, of which nine are instant and four initiate a respective process, e.g. chopping a vegetable or stirring the broth. All actions can be performed both by an NPC and by a human player.

The central complexity of the NPC behavior stems from the simulation of cooking. Some ingredients can be boiled, some fried. Water evaporates from pots and ingredients may burn or char if not stirred or flipped. Concrete details are not relevant for the paper but can be found in (Holaň 2013).

CASE STUDY

To confirm that the EmohawkVille platform is suitable for NPC behavior experiments we performed a case study with four human subjects. The motivation of the evaluation was to 1) estimate overall usability of the platform for experiments with NPC behaviors and 2) to gain some insight on impacts of the increased complexity of the virtual world. Moreover, the evaluation helped us with identifying what parts of the virtual world and the Pogamut module are particularly hard to understand.

Empirical studies concerning usability of frameworks for NPC behavior development are scarce; we followed the methodology from our previous research (Gemrot et al. 2012). We designed 5 hours long controlled experiment with human subjects. Subjects were asked to solve two assignments — program NPC behavior for cooking tasks. Measurement of the subjects' performance during the task, questionnaires and structured interviews were used to gather both quantitative and qualitative data about the code subjects have written and their opinions about the EmohawkVille framework.

The crucial part of an experiment design are the tasks human subjects have to complete during the experiment. To design experiment tasks with an appropriate level of difficulty we firstly implemented a complex NPC for EmohawkVille ourselves called the Chef NPC. Insights gained from this implementation were then used to design tasks for the qualitative case study.

The Chef NPC

The Chef NPC was designed to use all items and actions the EmohawkVille world offers. It can prepare food consisting of multiple ingredients created by various cooking procedures.

Also the Chef can react to sabotage or help in an intelligent, albeit emotion-less, way, e.g., if the user puts water to the pot the Chef will not do it again. Additionally, if the Chef is unable to find required ingredients it may request them from other (human) players.

The action selection mechanism of the Chef NPC exploits a hierarchical goal-driven architecture similar in principle to behavior trees (Champandard 2007). The tasks of the chef NPC have the following functionality,

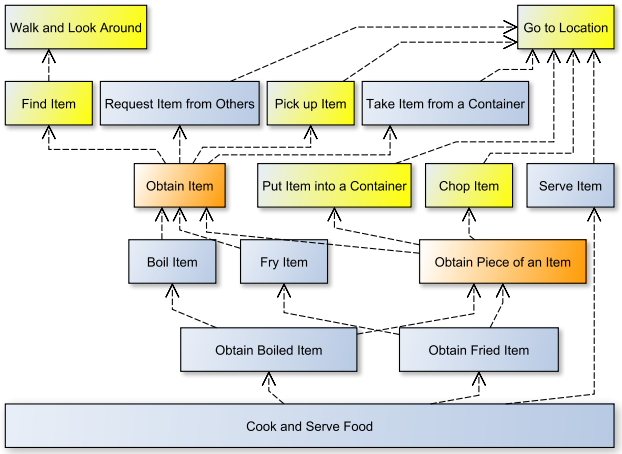

Figure 2: Dependency diagram of the implemented tasks of the Chef NPC. Yellow tasks were left in the NPC template for the case-study. Orange tasks were used as assignments for subjects of the case-study.

1) tell whether the task was finished with the distinction of success or failure to achieve the task goal, 2) execute logic for current frame that progresses the NPC towards fullfiling the task goal, 3) abort the task.

In every time step the agent executes its main task logic which in turn may execute one or more subtask logics. In the case of a static environment a task would simply execute a series of actions or subtasks in order. To achieve reactivity, the subtask that is closest to the end of the series and has its preconditions fulfilled is found in every logic iteration and executed. Thus the agent is both opportunistic and fault tolerant: it does not try to achieve subgoals that are already met and upon a hostile change in the environment it tracks back to the last unfulfilled subgoal. For example, if the NPC is to acquire cucumber slices and such slices already exist, reactive implementation can skip steps of collecting a whole cucumber and slicing it.

The examples of tasks used by our Chef are: acquire a certain item, go to a certain place, do a certain task (e.g. boil water or chop a cucumber), etc. See Figure 2 for the whole task hierarchy.

The implementation of all the tasks took about 10 man-days (80 man-hours) and was done by the creator of the Pogamut EmohawkVille module (the first author of the paper). This hints us that the desired behavior, although relatively simple at first glance is in fact fairly complex as even with high-level tools it tooks a non-trivial amount of time to implement. While the initial sketch of the behaviors was done quickly, the majority of the time was spent in determining possible points of failure, debugging and testing the behaviors. For example the relatively basic task of obtaining an item decomposes into four subtasks: 1) if the NPC can directly pickup the item, pick it up, 2) if the NPC knows a location of the item, move to that location, 3) if the NPC does not know of the item location search for the item 4) if the item cannot be found anywhere in the environment, find another avatar that possesses the item and try to trade it. Subtasks 2-4 cannot be directly executed in the environment and decompose further. Notice that this is a lower-level task (Figure 2). This shows that it is the complexity of the tasks and not user-unfriendliness of the platform that caused the 10 man-days long work and that better tools and techniques are needed for actual deployment of complex NPC behaviors in games. Note that we have just described a single scenario that is not backed, for instance, by buying ingredients or cleaning the kitchen up afterwards. We furthermore back this arguments with our experiences working on an AAA game project and that to our knowledge there is no larger scale commercial game that would involve NPCs with non-combat behaviors comparable in complexity to tasks on third level of our hierarchy not to mention the higher levels.

Experiment Setting

The experiment started with a 45 minutes long introductory presentation, which covered the mechanics of the EmohawkVille virtual world, the Pogamut module and the goal-driven architecture. The evaluation consisted of two assignments where the subjects were asked to implement a particular task in an NPC template based on the Chef NPC. Throughout the evaluation, the subjects answered appropriate questionnaires. All subjects had considerable previous experience with writing NPC behaviors for the Pogamut UT2004 module, and as a result, they could compare it with the Pogamut EmohawkVille module. The evaluation took 5 hours to complete (including the introductory presentation).

Assignments

As the implementation of the Chef NPC took days to complete, we had to scale it down for the subjects. We decided to create an NPC project template that was based on the Chef NPC code, but with empty implementation of "Obtain item" and "Obtain a piece of an item" tasks, which we asked subjects to implement. Tasks that were used in the original task implementation as sub-tasks were pointed out in the assignment.

The challenge of assignments lay in 1) learning how to use sub-tasks, 2) designing task stages, 3) implementing stage decision logic, 4) implementing the code executing each stage.

The "Obtain item" assignment was chosen for its relative simplicity. The task was to collect an item matching a given predicate.

The "Obtain a piece of an item" assignment was chosen as a representative of a complex task. The task required an NPC not only to produce slices obtained by chopping appropriate vegetable but also to satisfy chopping pre-

conditions (obtaining knife, gathering vegetable, finding a chopping board) and chop as many vegetables according to the number of slices required by the task parameter. For reliable execution, the task decomposes into 9 stages, none of which can be left out.

Hypothesis
We had two hypothesis. 1) The EmohawkVille with its Pogamut module is not hard to understand and use. 2) It is particularly hard to write robust NPC behaviors in a complex environment.
The first task was meant to produce data for the former hypothesis, the second task was meant to produce data for the latter hypothesis.

Subjects
We recruited four subjects for the evaluation. The first two subjects were students who finished the course on human-like agents (S1 and S2) and therefore had a considerable experience with development of NPC behaviors for the Pogamut UT2004 module. The other two subjects were researchers who developed parts of the Pogamut platform itself (R1 and R2).

Questionnaires
Every subject received six questionnaires throughout the evaluation. Questionnaires contained both questions with 5-point Likert scale (1 - "best/most", 5 - "worst,least") as well as open-ended questions. Questions were designed to 1) assess programming experience and skills of the subject, 2) check whether subjects understood the assignment and determine quality of their solution, 3) investigate usability of EmohawkVille and its Pogamut module, 4) report on the difficulty of the assignment.

Results and Discussion

All subjects understood the assignments well; The question "Have you understood the assignment?" was always answered by the option 1 or 2.
Researchers required considerably less time to complete the first assignment (Table 2). This can be explained by greater experience with Java, libraries and associated design patterns. The times to complete second assignment were similar for all subjects. One possible explanation is that first two subjects learnt the relevant skills during the first assignment, which would support Hypothesis 1.
Let us take a look at subjective difficulties of the first assignment (Table 3). The responses showed that the EmohawkVille Pogamut module is not sufficiently intuitive yet. Subjects reported on the confusing concept of observations and memorizations. This does not support our Hypothesis 1. On the other hand, the mechanics of the virtual world were found to be intuitive by all subjects, which in turn supports it.

Table 2: Times subjects estimated and actual times required by subjects to complete the first two assignments.

Assignment	**1**		**2**	
	est.	**actual**	**est.**	**actual**
R1	30m	47m	60m	91m
R2	25m	35m	45m	96m
S1	45m	62m	45m	95m
S2	20m	95m	60m	99m

Table 3: Reported difficulties of the first assignment, 1 - very easy, 5 - very hard. Subjects were asked what they think about difficulty of 1) the assignment (Assgn), 2) understanding the EmohawkVille Pogamut module (Module), 3) understanding the NPC template (Tmpl), 4) understanding the virtual world mechanics (Mechs), 5) formulating the NPC behavior (Form).

	Difficulty of				
	Assgn	**Module**	**Tmpl**	**Mechs**	**Form**
R1	2	4	2	1	3
R2	3	3	2	2	2
S1	2	4	2	2	2
S2	4	3	4	2	4

For the second assignment (Table 4), the first question was changed to be relative: "What do you think about the difficulty of the assignment in comparison with the first assignment?" (1 - much easier, 5 - much harder). Researchers reported increase in difficulty while students considered the assignment to be of the same difficulty or even less difficult. It became clear from open-ended questions, that this was because students had not realized the full extent of the task. Researchers were more aware about weaknesses in the code they have produced while students did not notice that some of their behaviors could fail. This partially supports our Hypothesis 2. It also provides ground for the research of tools suitability, as less-experienced user were not aware of problems with behaviors they produced; such as is the situation within computer game industry, where NPC behaviors are typically scripted by junior programmers as senior programmers are working on the game engine core.

CONCLUSION

In this paper we have presented the EmohawkVille virtual world. The case-study has found the platform to be suitable for experimenting with complex NPC behaviors.

Table 4: Reported difficulties of the second assignment, 1 - very easy, 5 - very hard. Subjects were asked what they think about difficulty of 1) the second assignment relatively to the first assignment (Assgn), 2) understanding the EmohawkVille Pogamut module (Module), 3) understanding the NPC template (Tmpl), 4) understanding the virtual world mechanics (Mechs), 5) formulating the NPC behavior (Form).

	Difficulty of				
	Assgn	**Module**	**Tmpl**	**Mechs**	**Form**
R1	5	3	1	1	4
R2	4	3	2	2	2
S1	3	4	3	2	2
S2	2	3	3	3	2

We have argued that the academia should focus on designing novel tools and techniques that would ease the burden of designing complex NPCs behavior, but such research is infeasible due to the unavailability of open-sourced 3D virtual environment. We have shown that EmohawkVille overcomes this obstacle.

As a future work we plan to gamify the environment by introducing a challenge inspired by the MasterChef TV show. Subjects will be challenged to create a team of cooperating NPCs that cook the maximum number of specified dishes within a given timeframe. During this experiment we aim to evaluate multiple behavior specification tools and guidelines to determine, which are the best suited for a practical application. We believe that testing tools on realistic tasks is a vital yet often underestimated part of the academic research concerning development of computer games.

ACKNOWLEDGEMENTS

Human data were collected with APA principles in mind. This research is supported by the Czech Science Foundation under the contract P103/10/1287 (GAČR), by student grants GA UK No. 655012/2012/A-INF/MFF (70%) and 559813/2013/A-INF/MFF (30%). This research is partially supported by SVV project number 267 314.

REFERENCES

Allbeck J.M., 2010. "CAROSA: a tool for authoring NPCs". In *Proceedings of the Third international conference on Motion in games*. Springer-Verlag, Berlin, Heidelberg, MIG'10, 182–193.

Bethesda Game Studios, 2011. "Elder Scrolls V: Skyrim". `http://www.elderscrolls.com/`. Last checked: 2013-09-13.

Bída M.; Brom C.; Popelová M.; and Kadlec R., 2011. "StoryFactory — a tool for scripting machinimas in unreal engine 2 and UDK". In *Proceedings of the 4th international conference on Interactive Digital Storytelling*. Springer-Verlag, Berlin, Heidelberg, ICIDS'11, 334–337.

Bída M.; Černý M.; Gemrot J.; and Brom C., 2012. "Evolution of gamebots project". In *Proceedings of the 11th international conference on Entertainment Computing*. Springer-Verlag, Berlin, Heidelberg, ICEC'12, 397–400.

Champandard A., 2007. "Understanding behavior trees". *AIGameDevcom*. URL `http://aigamedev.com/open/article/bt-overview/`.

Epic Games Inc., 2009. "Unreal Development Kit documentation". `http://www.unrealengine.com/udk/documentation/`. Last checked: 2013-07-01.

Gemrot J.; Brom C.; Bryson J.; and Bída M., 2012. "How to compare usability of techniques for the specification of virtual agents' behavior? an experimental pilot study with human subjects". In *Proceedings of the 2011 international conference on Agents for Educational Games and Simulations*. Springer-Verlag, Berlin, Heidelberg, AEGS'11, 38–62.

Gemrot J.; Kadlec R.; Bída M.; Burkert O.; Píbil R.; Havlíček J.; Zemčák L.; Šimlovič J.; Vansa R.; Štolba M.; Plch T.; and Brom C., 2009. "Agents for Games and Simulations". Springer-Verlag, Berlin, Heidelberg, chap. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. 1–15.

Holaň D., 2013. "Items and actions in 3D virtual environment Emohawkville". Faculty of Mathematics and Physics, Charles University in Prague. Master Thesis.

Plch T., 2009. *Action selection for an animat*. Master's thesis, Charles University in Prague.

Rockstar Games, 2010. "Read Dead Redemption". `http://www.rockstargames.com/reddeadredemption/`. Last checked: 2013-09-13.

Shoulson A.; Marshak N.; Kapadia M.; and Badler N.I., 2013. "ADAPT: the agent development and prototyping testbed". In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, New York, NY, USA, I3D '13, 9–18.

Unity Technologies, 2005. "Unity documentation". `http://docs.unity3d.com/Documentation/Manual/index.html`. Last checked: 2013-07-02.