TEACHING HOW TO ENGINEER BEHAVIORS FOR VIDEOGAME CHARACTERS

Jakub Gemrot
Faculty of Mathematics and Physics,
Charles University in Prague
gemrot@ksvi.mff.cuni.cz

Martin Černý
Faculty of Mathematics and Physics,
Charles University in Prague
cerny.m@gmail.com

Cyril Brom
Faculty of Mathematics and Physics,
Charles University in Prague
brom@ksvi.mff.cuni.cz

Abstract

In this paper we report on the practice lessons for the course on Modeling Behavior of Human and Animal-like Agents that teaches how to practically design and implement behaviors for videogame characters. We discuss education challenges that come from the inherent complexity of virtual behaviors as students need to learn how to interact with chosen virtual game environment before they can apply theoretical knowledge acquired during lectures. We present yaPOSH– a visual tool for design of behaviors that allows for separation of the behavior design from the actual behavior implementation. Students report that the tool helped them to separate their thoughts and development efforts between the design and implementation. Additionally we show that implementation details of the yaPOSH tool had positive impact on the effectiveness of learning.

Categories and Subject Descriptors D.2.2 [**Software engineering**]: Design Tools and Techniques, K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms Design, Languages

Keywords Education, Video Games, Virtual Behaviors

1. Introduction

Quality of videogame character behaviors is gaining more and more attention from players community and it is slowly becoming the tip on the scales that may bring the reception of the game down or glorify it. Therefore, the education of future video game developers should cover techniques how to model behaviors of videogame characters as well as provide them with an opportunity to practice these techniques within the boundaries of complex 3D virtual environments.

In this context, we run the Modeling Behavior of Human and Animal-like Agents course that focuses primarily on the virtual characters action-selection from the perspective of artificial intelligence, computer games and ethology. The course is running since 2005 and its theoretical part (lectures) is described in [1]. Here we report on practice lessons (classes) we have created for the course in 2008 and have been gradually improving since.

While the lectures are focused on the broader category of virtual characters (intelligent virtual agents) including topics on cognitive science research and computational ethologic simulations [2, 3], classes focuses directly on the development of video game character behaviors (referred to as bots, bots behaviors or simply behaviors for brevity) for 3D video game. Technically, the classes utilize the Pogamut platform [4].

The paper is structured as follows. First, we discuss educational objectives of the course classes. Second, we report on the classes structure, detailing how respective objectives are met. Third, we present the yaPOSH tool we use to exemplify, how behaviors can be designed in an implementation-agnostic way, teaching students a crucial behavior design point; how to think about and structure behaviors in a portable way. Last, we discuss students feedback we collect on the Pogamut platform and yaPOSH during final exam.

2. Related Work

As in other areas of programming, hands-on experience is the key to understanding behavior development. The choice of suitable environment that the students interact with (both as players and as programmers) is of high importance.

There are multiple software packages that aim to teach programming within the context of 3D virtual environments such as Alice [5] or CodeSpells [6]. While they provide 3D virtual environment for a programmer to work with, they are not aimed at the development of behaviors specifically; they are meant as environments for teaching *object first* approach in introductory computer science courses.

In the context of AI, educational scenarios based on Pac-Man and other simple game environments [7] have been proposed. Those are however not applicable to our case as they focus on classical AI techniques and do not involve environment comparable in complexity to 3D computer games.

Finally, it is possible to build the course around existing open-source (or at least not so expensive) 3D game engine that features well-developed game editor, such as Unity 3D [8], Unreal Engine 4 [9], Unreal Development Kit [10], CryEngine FreeSDK [11], etc. Even though these clearly are alternatives to the course tools described in this paper, they cannot be used out of the box for the behavior development education and the educator would need to invest non-trivial effort for their adaptation.

3. Course Classes

3.1 Background

The course is tailored to computer science students at least in their fourth term of bachelor studies after they attended several courses on programming (10.5[1]), mathematics (17), general IT skills (8.5) and algorithms (5.5) [1]. Every year, the course is attended by about 20 students. All students are typically familiar with Java language used throughout the classes.

3.2 Classes Objectives

Classes objectives are (1) to allow students to exercise behavior development practically, (2) to exemplify a layered architecture of behaviors, (3) teach students how to separate action-selection from its implementation in order to produce readable and maintainable code.

3.3 Virtual Environment

The choice of virtual environment for the behavior development classes is the most crucial point. It may (a) strongly affect the learning curve of the behavior development basics, (b) determine a range of behaviors students can practice on, (c) constrain the structuring of the classes and (d) impose high requirements on the teacher.

The challenge here is to have an environment (and related tools) that would "sell" the course to students by offering huge possibilities, but at the same, it should allow for teaching the environment API in smaller steps, mixing the details of API with concrete tasks (behaviors to implement) the students can practice on, while being relatively easy to use.

We base our classes on an environment of Unreal Tournament 2004 (UT2004), an older 3D first-person shooter video game. The environment is of commercial quality and complexity and still appeals to student graphically. Most importantly, the game mechanics of UT2004 are very similar to mechanics of contemporary games.UT2004 provides navigation graph [12] – standard interface for navigation within the environment – and both individual and team-oriented game modes. Individual game modes such as death-match (DM) are easier to understand and allow for quick implementation of bots that play the game well, but offer only limited complexity. Team oriented modes, e.g. capture-the-flag (CTF), require to mix both tactical and strategic decisions and offer endless possibilities for improvements as well as for applications of advanced AI techniques.

The game in its raw state is quite a hostile programming environment – it features proprietary scripting language that has no standard debugging support (watches, breakpoints, etc.). Although more friendly programming environments are very rare in the game industry, it would not be a very good starting point for novice programmers.

To make the game suitable for education, we are using the Pogamut platform [4] – a toolkit we have originally developed for behavior development research. The platform lets the programmer to control bots inside the game via Java API (stable, bug-free, well documented)[2]. The platform features in-game developer tools such as bot state visualization, navigation visualization and logging (detailed in [13]). It can also be used for creation of custom game modes, which we use for fine structuring of the classes. Finally, the Pogamut platform features yaPOSH – a visual tool for design of behaviors that supports both editing and debugging of yaPOSH plans. It is an incarnation of behavior trees [14], which are considered to be an industry standard for the development of behaviors.

The drawback of the choice is the absence of smart objects within the environment [15], which is a useful technique for behavior structuring.

[1] Normalized number of courses on given topic; 1 course equals to 13 lessons unit (13 x 90 minutes). The course presented in this paper amounts to 1.6 (classes included).

[2] The use of Java language does not pose unnecessary obstacles for our students as they learn it early during the studies (c.f. possible use of proprietary language such as UnrealScript [16] for UT2004, Papyrus [17] for Skyrim [18] or custom action-selection mechanisms such as UDK's Kismet [19] or CryEngine behavior trees [20]).

Table 1. Classes, their topics and the tasks students have to solve.

No.	Topic	Task	Bot Behavior Description
1	Low-level movement	Dog Bot	Follow the player around the environment.
2	Tactical movement	Tag! Bot	Play children Tag! game, both roles (catcher, runner).
3	Path finding, Stuck detection and resolution, Navigation	NavBot	Navigate randomly around the environment, solves tucks.
4	Visibility, A* extensions	Hide&Seek Bot	Play childrenHide&Seek game, both roles (seeker, runner).
5	Items management	ItemPicker Bot	Navigate around environment and pick items, prioritize the item order according to their distance and relevance.
6	Weapons manag., Shooting	DM Bot	Play DM mode of UT2004.
7	CTF rules	CTF Bot	Single bot that can play CTF mode of UT2004.
8	Team communication	CTF Team	Team of bots playing CTF mode of UT2004.

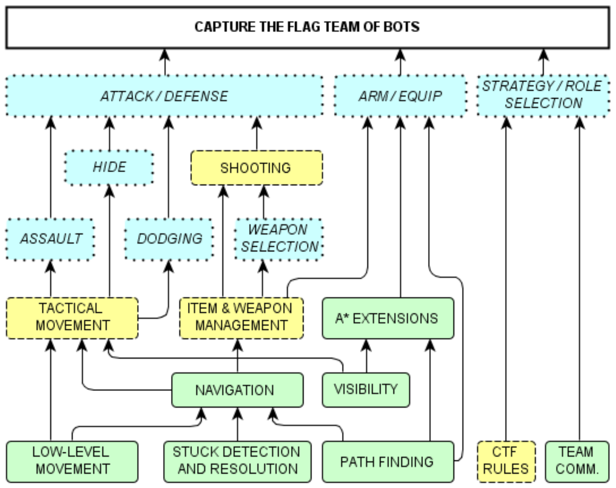


Figure 1. Decomposition of the CTF bot behavior into general behavior issues (solid border), UT2004 specific knowledge or sub-behaviors (dashed borders) and task specific sub-behaviors (dotted borders). Arrows depict dependencies between individual topics.

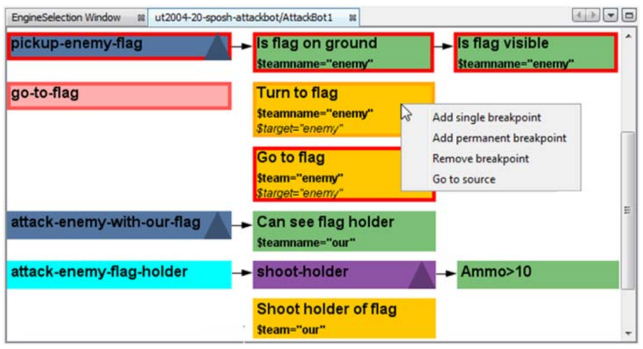


Figure 2. Visualization of the part of the CTF bot behavior (matches Fig. 3, left column) as presented by the yaPOSH editor.

3.4 Classes Overview

The ultimate goal is to build a team of bots for the capture-the-flag game mode. In order to successfully implement behaviors for the team, students need to be taught various programming skills, techniques and algorithms both generic and UT2004 specific (Fig. 1).

As the CTF task is daunting, students are presented with simpler tasks first, which gradually teach them about different behavior issues, aspects of UT2004, the Pogamut API and algorithms and programming techniques required to implement team of CTF bots (Tab. 1). Tasks respect the dependencies from Fig. 1 and behaviors for latter tasks always build on experiences from doing the prior ones.

Key aspect of the classes is the behavior decomposition: new low-level concepts are introduced one at a time and are always accompanied with behavior development. Following assignments then reuse the same low-level functionality, increase complexity of the behaviors and let the student take advantage of his experience in the previous tasks without letting him to copy-paste a previous solution. For example, students learn to hide from other players and reason tactically about their movement in the Hide&Seek task. This is similar to the tactical movement of the CTF bot, the very same API calls are used and the general philosophy is alike, but the student needs to rethink the behavior decomposition to balance positioning for a good shot (new in CTF) with hiding from the enemy (known from Hide&Seek). On a similar note, the students learn how to create assault behavior while programming DM bot, but in CTF bot, the assault must be balanced with handling the flag and team cooperation and thus requires a redesign of the behavior decomposition.

3.5 Lesson Structure

Each lesson is divided to theoretical and practical part. Theoretical part lasts around 45 minutes and we explain the goal of the lesson - which is usually to implement one of the presented bot types - and all the necessary theoretical knowledge and API calls that will be needed to solve the lesson task. This is then followed by a practical part (45 minutes), where the students begin to solve the problem. The tasks are mandatory and they are set in a way the students usually will not complete the implementation on the lesson, but need to finish it at home. To further motivate the students, we organize three tournaments during the lessons for the fundamental tasks such as basic movement (Tag! Bot), reasoning (Hide&Seek bot) and combat behavior (DM Bot). All solutions of students from these task are automatically submitted to the tournament and the results are then shared and commented on. This improves student engagement in the lessons and motivates the students to spend more time on the assignments.

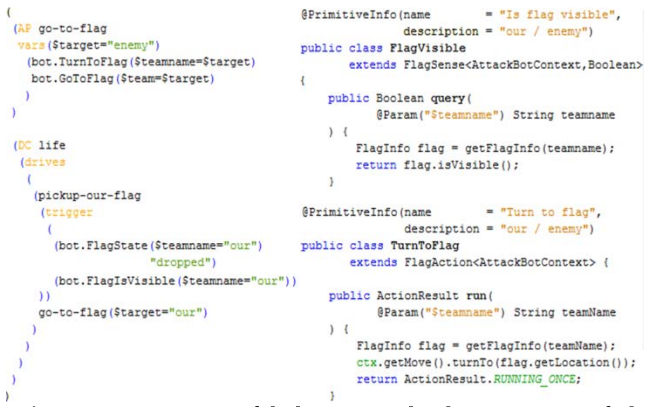
3.6 Behavior Oriented Design and yaPOSH

Behavior modeling approach taught during classes follows the principles of Bryson's Behavior Oriented Design (BOD) [21]. "*BOD is a methodology for developing control of complex intelligent agents, such as virtual reality characters, humanoid robots or intelligent environments.*" [22] As such, it is applicable to video game characters and their behaviors. The methodology encourages iterative development that consist of multiple design-implement-test cycles. Originally, the methodology is bound with the POSH language that provides language constructs for respective behavior primitives recognized by the methodology (e.g. action sequences). We adapted the methodology for the use within the Java language by mapping POSH language constructs to Java language templates (based on combination of method calls, if-then rules and finite state machines) that provides user with similar expressiveness while sustaining the favorable properties of POSH language like self-documentation and the separation of behavior structure and primitives and their implementation.

Not to teach BOD in Java only, we have created own adaptation of POSH, so called yaPOSH (implemented in Java). Additionally, as yaPOSH plan Lisp-like syntax (Fig. 3, left column) has been found confusing to our users, we have developed graphical editor for yaPOSH plans and integrated it tightly into NetBeans Java IDE. The yaPOSH editor provides way for custom behavior primitives definition (in the form of Java classes, Fig. 3, right column), allows to structure the behavior via drag&drop actions and features plan debugger (Fig. 2) that allows to place breakpoints on behavior primitives during bot runtime.

Later during the course, when students become familiar with the methodology as well as somewhat versed in UT2004 environment mechanics and related API, they are introduced to yaPOSH where they are forced to exercise the methodology in its pure form (without behavior structure hacks that are possible in Java such as execution of actions from different places in parallel).

The key point of BOD demonstrated by the use of yaPOSH is the separation of behavior structure and its implementation. If the game rules were the same, the yaPOSH behavior plan could have been reused between different environments (Fig. 2); only the implementation of behavior sensors and actions would have differed (Fig. 3, right column). The same applies to the Java mapping. If a student decomposes the action-selection without references to the environment API wrapping all API-dependent behavior structures such as sensors and actions into separate method calls, the action-selection code would again remain the same between different environments (similar to the way multi-platform software is written). The approach promotes good coding habits such as code readability and maintainability as discussed by Bryson [21]. Therefore, the course classes crosses the boundaries of UT2004 and its environment even though they are built around it.



```
(
 (AP go-to-flag
  vars($target="enemy")
   (bot.TurnToFlag($teamname=$target)
   bot.GoToFlag($team=$target)
  )
 )

 (DC life
  (drives
   (
    (pickup-our-flag
     (trigger
      (
       (bot.FlagState($teamname="our")
                      "dropped")
       (bot.FlagIsVisible($teamname="our"))
     ))
     go-to-flag($target="our")
    )
   )
  )
 )
)
```

```
@PrimitiveInfo(name        = "Is flag visible",
               description = "our / enemy")
public class FlagVisible
       extends FlagSense<AttackBotContext,Boolean>
{
    public Boolean query(
            @Param("$teamname") String teamname
    ) {
        FlagInfo flag = getFlagInfo(teamname);
        return flag.isVisible();
    }

@PrimitiveInfo(name        = "Turn to flag",
               description = "our / enemy")
public class TurnToFlag
       extends FlagAction<AttackBotContext> {

    public ActionResult run(
            @Param("$teamname") String teamName
    ) {
        FlagInfo flag = getFlagInfo(teamName);
        ctx.getMove().turnTo(flag.getLocation());
        return ActionResult.RUNNING_ONCE;
    }
```

Figure 3. Snippets of behavior code that are part of the CTF bot behavior (matches Fig. 2). Left column: yaPOSH plan and its Lisp-like syntax. Right column: Sensor `FlagVisible` and action `TurnToFlag` implementations within the Java language.

```
// pickup-enemy-flag
if (   isFlagOnGround("enemy")
    && isFlagVisible("enemy"))
  goToFlag();
else {
  goToFlag_Reset();
  // attack-enemy-with-our-flag
  if (canSeeFlagHolder("our"))
    attackEnemyFlagHolder();
}
```

Figure 4. Translation of yaPOSH behavioral plan from Fig. 2 into Java code using if-then rules.

4. Evaluation

In order to improve the Pogamut platform and yaPOSH editor, we are collecting opinions about their use from students during the final exam of the course. We are monitoring how students are satisfied with the course (lectures and classes separately) and how they are satisfied with tools they had to use. We are collecting both objective and subjective data.

Table 2. Summary of answers for Q2: What do you think about the Pogamut+UT2004 platform?

Category & Sample answers	**Count**		
	2011	2013	2014
No problem / Perfect / Easy to start or work with / Enjoyed [2014] *"Very easy to start with and create fun bots quickly, found no bugs during the semester. Great library even by professional standards with good documentation."* [2013] *"Easy to understand, easy to learn and yet very powerful tool."* [2011] *"Great for its purpouse. Simple to write relatively complex behaviours."*	2	7	7
Lot of functionality, but after a few use, easy to work with [2014] *"Very intuitive, after a few months, I still remember everything we were taught."* [2011] *"Lot of functions, but they are very easy to use."*	6	2	5
Some caveats or issues, but easy or interesting or fun to work with [2014] *"Very easy to start with, some tricky caveats await for those who will play with it, but all-in-all a very good platform for writing bots."* [2013] *"Sometimes, documentation is not sufficient, but good overall."* [2011] *"It's sexy in the way it can co-operate with an AAA title like UT. The actual API seems however a bit cluttered and cumbersome, which makes it quite difficult to learn on your own."*	8	6	2
Hard corners / Bugs / Some things could be better or should be added [2013] *"There a few things missing, like looking behind when the bot is running."* [2011] *"It's quite buggy, but nice education tool. When it gets fixed, it'll be great."*	4	3	0
Too complex [2011] *"There are so many possibilities that it's hard to choose the right one some-time."*	2	0	0

Table 3. Summary of answers for Q3: Compare coding of bot behaviors in Java-ONLY to yaPOSH+Java.

Category & Sample answers	**Count**	
	2013	2014
A. yaPOSH Acknowledgements		
It is easy to create behaviors with preimplemented primitives; you do not even need to be a pro-grammer).	5	5
Easy to orient in / Easier to change behaviors /yaPOSH brings better behavior structuring	5	5
B. yaPOSH Editor Criticism		
Editor not user friendly / Bugs / Plan visualization should be better	12	2
C. yaPOSH Behavior Structuring Criticism		
Behavior plan structure limitations	5	6
Necessity to create class for every behavior sense or action	2	2
D. Java Behavior Structuring Critisim		
Java tends to spaghetti code / Hard to orient in the code / Not seeing the whole behavior / Harder to change the behavior structure	5	11
E. Java Acknowledgements		
Java execution sematics is more clear.	3	0
If you can manage to keep the code tidy, Java is easier to orient in.	2	0

4.1 Subjects

We report data and opinions from three years of the course 2011 (22 males), 2013 (18 males) and 2014 (13 males, 1 female). We used Chi-square test to confirm that differenc-es between groups are not significant (age, number of AI lectures studied, man-months spent programming in any language, motivation to study behavior development). We do not report data on students from 2012 as they were not working with Java and yaPOSH.

4.2 Exam Structure

As a final exam, students are asked to solve two tasks (create two behaviors). The first task is named GuideBot; students had to create a bot that is capable to search for other friendly agents and guide them. The second task is named GuardBot; students has to extend the existing GuideBot behavior to include active protection of the guided agent against hostile bot present in the environment. The GuideBot behavior had to be extended with simple combat sub-behavior and the switching between *guiding* and *guarding* must be solved.

The second task also contains a twist; students did not receive the same pre-prepared GuideBot behavior for the extension, they are given the behavior created by other student. This design introduced variables to control for, but required the participants to work with behavior development differently; they had to read and understand existing behavior first before they could actually start extending it (for more info see [27]). As the student can receive bad code, they are not actually graded according to the outcome of the second task but only for the first.

Students from 2011 and 2013 were split into two groups according to the tool they were using (Java-ONLY or yaPOSH+Java). Students from 2012 were using yaPOSH+Java only.

4.3 Measured Variables & Asked Questions

Here we report time students need to finish respective exam tasks (T1 for GuideBot task, T2 for GuardBot task). and answers to the following questions. Q1. What do you prefer for behavior development, Java-ONLY or yaPOSH+Java? (11-Likert like scale, 0 - Java-ONLY, 10 - yaPOSH+Java). Q2. What do you think about the Pogamut+UT2004 platform? Q3. Compare coding of bot behaviors in Java-ONLY to yaPOSH+Java. Questions Q2 and Q3 are formulated vaguely in order to avoid leading questions and to obtain the broad range of opinions. Question Q3 was added into the questionnaire in 2013.

4.4 Results

Solution times are summarized in Tab. 4. Tool preferences (Q1) are summarized in Tab. 5. Comments to the Pogamut platform (Q2) and differences between plain Java and yaPOSH+Java (Q3) are summarized in Tab. 2, resp. Tab.3.

Table 4. Solution times for respective tasks, groups and years.

	Task 1 Time Mean (SD)		Task 2 Time Mean (SD)	
	Java ONLY	yaPOSH +Java	Java ONLY	yaPOSH +Java
2011	162,9 (32,3)	170,6 (32,3)	F	F
2013	N/A	79,4 (25,9)	N/A	112,5 (25,9)
2014	64 (34,3)	57,9 (26,3)	73,3 (27,9)	119,3 (48,9)

Table 5. Tool preferences (Q1) for respective years.

	Tool preference				
	Strong Java Pref. [0]	Java Pref. [1-4]	Neut. [5]	yaPOSH Pref. [6-9]	Strong ya-POSH Pref. [10]
2011	4	6	5	6	1
2013	3	4	4	5	2
2014	1	1	5	3	4

5. Discussion

Here we discuss the main highlights of our course and methodology.

The Pogamut platform is matured. Looking at the Tab. 2 we can see clear shift towards positive feedback between both year 2011 - 2013 and 2013 - 2014. In 2014, the majority of students rate the platform either as "Perfect" or "Easy to use".

yaPOSH editor user experience has been improved. Looking at the Tab. 2**,** C, we can see drop in the number of objections to the editor. The majority of objections were resolved (between 2013-2014) by creating wizard for action and senses creation.

Students spend less time with the platform and yaPOSH. Looking at Tab. 4, we can see that times required to solve final exam tasks have dropped dramatically between 2011 and 2014. We link this drop with the fact the Pogamut platform is now stable and many yaPOSH usability issues (as well as bugs) have been cleared out. This trend can be seen in Tab. 5 as well, where students from 2014 prefer yaPOSH over Java more than in previous years. Additionally, students from 2013 and 2014 were able to finish the task, c.f. students from 2011.

Recognition of well-formed behavior structure and its presentation in yaPOSH. Looking at Tab. 3, A and D, we can see that many students report that it is hard to orient in behavior code in Java, whereas yaPOSH editor ease this by providing visual representation that has fixed structure. Interestingly, this recognition projects more to the negative feedback for Java (Tab. 3, D) than positive comments to yaPOSH (Tab. 3, A). We consider this as an indirect proof that students learned BOD and accustomed to the structuring of behaviors according to the methodology.

Still room for improvements. Looking at Tab. 3, C, there are still rooms form improvements. The negative feedback to yaPOSH behavior structuring are mainly related to the yaPOSH inability of expressing parallel action. We are planning to resolve this issue for 2015.

6. Conclusion

In this paper we have presented the objectives and structure of practice lessons for the course on Modeling Behavior of Human and Animal-like Agents. The paper presented the challenges of behavior development education, namely the necessity to bind the classes with concrete 3D virtual environment and the implications of such a bond. The structure of classes emphasize the necessity to balance the amount of knowledge the students have to learn for respective tasks and actual behavior development. Learning objective was met by teaching BOD methodology with the use of yaPOSH we developed specifically for this purpose. More information about the classes as well as materials are freely available at our website [23].

Acknowledgment

Human data were collected with APA principles in mind.

This research is supported by the Czech Science Foundation under the contract P103/10/1287 (GACR), by student grants GA UK No. 655012/2012/A-INF/MFF and 559813/2013/A-INF/MFF.

This research is partially supported by SVV project number 267 314.

References

[1] 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. pp. 71 - 79 (2009)

[2] Burges, N.: The hippocampus space, and viewpoints in episodic memory. The Quart. Jn. of Exp. Psych., 55A(4), 1057-1080.

[3] Herbelin, B.L.: Virtual reality exposure therapy for social phobia. PhD thesis, n. 3351. EPFL.

[4] Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom C. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Agents for Games and Simulations, LNCS 5920, Springer, pp. 1-15. (2009)

[5] Teaching Objects-First In Introductory Computer Science (PDF) Stephen Cooper, Wanda Dann, Randy Pausch, SIGCSE 2003.

[6] Esper, S., Foster, S.R., Griswold, W.G.: CodeSpells: embodying the metaphore of wizardry for programming. In: Proceedings of the 18th ACM conference on Innovation and technology in computer science education, Canterbury, England, UK, pp. 249-254 (2013)

[7] DeNero, J., Klein, D.: Teaching introductory artificial intelligence with pac-man. In Proceedings of the Symposium on Educational Advances in Artificial Intelligence. (2010)

[8] Unity 3D Game Engine. http://unity3d.com/ (20.6.2014)

[9] Unreal Engine 4. https://www.unrealengine.com/blog/welcome-to-unreal-engine-4 (20.6.2014)

[10] Unreal Development Kit. https://www.unrealengine.com/products/udk/ (20.6.2014)

[11] CryEngine FreeSDK. http://www.crydev.net/dm_eds/download_detail.php?id=4 (20.6.2014)

[12] Navigation graph. http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/navigation-graph-generation-r2805 (20.6.2014)

[13] Bida, M., Cerny, M., Gemrot, J., Brom, C.: Evolution of GameBots project. In: Herrlich, M., Malaka, R., Masuch, M. (eds.) ICEC 2012. LNCS, vol. 7522, pp. 397-400. Springer, Heidelberg. (2012)

[14] Champandard, A. J.: Behavior Trees for Next-Gen Game AI. Internet presentation. http://aigamedev.com/insider/presentations/behavior-trees (20.6.2014)

[15] Scripting and Sims 2: Coding the Psychology of Little People Jake Simpson, presentation, Game Developer's Conference 2005

[16] UnrealScript. http://udn.epicgames.com/Three/UnrealScriptHome.html (20.6.2014)

[17] Papyrus scripting language for Skyrim. http://www.creationkit.com/Papyrus_Introduction (20.6.2014)

[18] Skyrim, the video game. http://www.elderscrolls.com/skyrim (20.6.2014)

[19] UDK's Kismet http://udn.epicgames.com/Three/KismetUserGuide.html (20.6.2014)

[20] CryEngine Behavior Trees. http://docs.cryengine.com/display/SDKDOC4/Behavior+Trees (20.6.2014)

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

[22] Behavior Oriented Design website. http://www.cs.bath.ac.uk/~jjb/web/bod.html (20.6.2014)

[23] Modeling Behavior of Human and Animal-like Agents Classes website. http://pogamut.cuni.cz/pogamut-devel/doku.php?id=lectures (20.6.2014)