How to compare usability of techniques for the specification of virtual agents' behavior? An experimental pilot study with human subjects

Jakub Gemrot[1], Cyril Brom[1], Joanna Bryson[2], Michal Bída[1]

[1] Faculty of Mathematics and Physics, Charles University in Prague,
Malostranske namesti 25,
118 00, Prague 1, Czech Republic
[2] University of Bath,
Bath, BA2 7AY, United Kingdoms

Abstract. Reactive or dynamic planning is currently the dominant paradigm for controlling virtual agents in 3D videogames. Various reactive planning techniques are employed in the videogame industry while many reactive planning systems and languages are being developed in the academia. Claims about benefits of different approaches are supported by the experience of videogame programmers and the arguments of researchers, but rigorous empirical data corroborating alleged advantages of different methods are lacking. Here, we present results of a pilot study in which we compare the usability of an academic technique designed for programming intelligent agents' behavior with the usability of an unaltered classical programming language. Our study seeks to replicate the situation of professional game programmers considering using an unfamiliar academic system for programming in-game agents. We engaged 30 computer science students attending a university course on virtual agents in two programming assignments. For each, the students had to code high-level behavior of a 3D virtual agent solving a game-like task in the Unreal Tournament 2004 environment. Each student had to use Java for one task and the POSH reactive planner with a graphical editor for the other. We collected quantitative and qualitative usability data. The results indicate that POSH outperforms Java in terms of usability for one of the assigned tasks but not the other. This implies that the suitability of an AI systems-engineering approach is task sensitive. We also discuss lessons learnt about the evaluation process itself, proposing possible improvements in the experimental design. We conclude that comparative studies are a useful method for analyzing benefits of different approaches to controlling virtual agents.

1 Introduction

Reactive planning is currently the dominant paradigm for controlling virtual agents in 3D videogames and simulations. Prominent reactive planning techniques used in the industry are derivations of finite state machines (FSMs) [1], and more recently, behavior trees [2]. Technically, these are implemented in a scripting language, be it a

general-purpose language such as Lua [3] or a special-purpose language tailored at a particular game, such as UnrealScript [4], or hard-coded in a game's native language, typically C++ [5]. Advantages and drawbacks of different approaches used by the industry have been commented on widely [6,7,8].

At the same time, academic action-selection systems for AI planning are becoming increasingly mature, and the question arises whether they have advantages over the solutions employed presently by the industry. These systems include decision making modules of several cognitive architectures, e.g., Soar and ACT-R [9, 10], stand-alone BDI-based programming languages, e.g. GOAL [11], and stand-alone reactive planners such as POSH [12]. It has been already demonstrated that some of these systems, for instance Soar [9], POSH [13], GOAL [11] and Jazzyk [14], can be used for controlling virtual agents acting in game-like environments. From the perspective of efficacy of code execution, these systems are sluggish and can be considered as prototypes only at the present stage of maturity; however, they could potentially outperform some industry solutions in terms of usability (from the programmers' perspective), re-usability (of parts of code) and agent's cognitive performance, as assumed, for instance, by part of the academic community studying BDI-based languages [15].

Sound empirical data demonstrating the alleged advantages of different reactive planning technique, both industrial and academic, are generally lacking. Tyrell analyzed various robotics and ethology-based action selection mechanisms in terms of agent performance given approximately equal amounts of time devoted by a programmer [16]. This work was extended by Bryson in an effort to provide an evaluation for her own POSH action selection. [17]. Tyrrell's system was to test a single action-selection mechanism over a large number of "lifespans" by agents inhabiting an extremely rich and varied environment. The complexity of the environment lead to enormous variation in the results, so statistical significance was determined by running enough trials to compare the standard error rather than the standard deviation.

Bryson also provided a more theoretically formal but less rigorous comparison of POSH action selection to FSMs, showing that POSH plans were able to express action an intelligence was likely to choose to do in a more efficient way than an FSM [18]. However, none of these studies engaged programmers other than the authors themselves in the mechanisms' evaluation. In contrast, Hindriks et al [19] conducted an extensive qualitative analysis of the code of 60 first year computer science students developing (in teams of five students) three Capture The Flag agents for the videogame Unreal Tournament 2004 (UT 2004) using GOAL agent programming language. Hindriks's team aimed at "providing insight into more practical aspects of agent development" and "better understanding problems that programmers face when using (an agent programming) language" and identified a number of structural code patterns, information useful for improvements to the language. However, that study was not comparative and did not report the programmers' feedback.

Here, we are interested in a complementary approach, namely feasibility of quantitative comparative quasi-experimental studies (as used in psychology and social sciences) for investigating usability of action selection systems from the users' (programmers') perspective. We specifically address the usability issue as opposed to the efficiency or performance issue. This perspective encompasses various objective

and subjective measures, such as steepness of the learning curve, time spent by development, programming vs. testing time ratio, number of bugs made by the programmer, subjective attitude towards the technique etc. We designed and conducted a pilot study with the following objectives:

a) to investigate the subjectively-perceived usability of an academic action selection system designed to be useful for programming agents' behavior, when compared to perceived usability of an unenhanced classical programming language; this mimics the situation of game programmers considering using an academic system they are not familiar with for programming in-game artificial intelligence;

b) to compare the quality of solutions implemented in the academic action selection system and in the classical programming language; this measure plays an important role in the adoption of new systems in general;

c) to consider whether the experimental method *per se* is useful and whether (and under which conditions) it can produce helpful results.

We have been running a course on virtual agents development for computer science students at Prague University since 2005. Students are taught various techniques for controlling virtual agents [20] and trained to program their behavior in the virtual environment UT 2004 (similarly to Hindriks et al.). For that task, our integrated development environment Pogamut [21] is used by the students. In the academic year 2009/10, we turned the final exam for the course into a scientific experiment engaging 30 computer science students in two programming assignments lasting 3 hours each. Each student had to code the high-level behavior of a 3D virtual agent solving a game-like task in the UT 2004. The conventional language and the language underlying the academic system were both Java. We use Java because its learning curve is less steep than that of C++ (a more usual game development language) and because our students are expected to be at least to some extent familiar with Java. For the academic system, we used the POSH reactive planner with a graphical editor. This is because POSH has been already demonstrated for controlling UT agents [13] and because POSH has previously been investigated by our postgraduates and integrated into Pogamut.

For both the tasks and in both programming environments, the students' task was to organize low-level action and sensory primitives to produce complex behavior, but not to program the primitives as such. The drag-and-drop graphical editor we developed for POSH disguised its Lisp-like underlying plan syntax students might have struggled with. The study was only possible because the Pogamut platform provided the same development environment for both tasks and allowed us to predesign the same sets of behavior primitives, isolating the features of the language as the subject of the study.

We collected various quantitative and qualitative usability data in four questionnaires. Our main hypothesis was that subjects' attitude towards POSH would be at least as high as towards Java. As this is a pilot study, we kept the research question as simple as possible. Of course, for practical, commercial application of POSH, it would be an advantage to specifically identify its benefits compared to Java (and other systems), but this was not our aim for this study and is left for future work.

The rest of the paper proceeds as follows. We introduce POSH in Section 2 and detail the methods of our study in Section 3. The results are presented in Section 4 and discussed in Section 5, and Section 6 concludes.

2 POSH

POSH action selection was originally developed in the late 1990s in response to criticism of what was at the time an extremely popular agent design approach (at least in academic discussion): the Subsumption Architecture (SA) [27]. SA was used to produce considerable advances in real-time intelligent agents, particularly robotics. It consists primarily of two components: a highly modular architecture where every action is coded with the perception it needs to operate; and a complex, highly distributed form of action selection to arbitrate between the actions that would be produced by the various modules. Although extremely well-known and heavily cited, the SA was seldom really used outside of its developers. Bryson hypothesized that the emphasis on modular intelligence was actually the core contribution of SA, but that the complexity of action selection, while successfully enforcing a reactive approach, confused most programmers who were not used to thinking about concurrent systems.

POSH was developed then to simplify the construction of action selection for modular AI. Briefly, a programmer used to thinking about conventional sequential programs is asked to first consider a worst-case scenario for their agent, then to break each step of the plan to resolve that scenario into a part of a reactive plan. Succeeding at a goal is the agent's highest priority, so should be the thing the agent does if it can. The programmer then describes for the agent how to perceive that its goal has been met. Then for each step leading up to the goal the same process is followed: a perceptual condition is defined allowing the agent to recognize if it can take the action leading most directly to its goal [12, 18]. The actions are each small chunks of code that control the agent, so-called behavior primitives (see Tab. S2 – all supplementary figures and tables can be found in the appendix), and the perceptions are sensory primitives (Tab. S4).

After a period of experimenting with the system, Bryson embedded POSH in a more formal development methodology called Behavior Oriented Design (BOD). BOD emphasizes the above development process, and also the use of behavior modules written in ordinary object-oriented languages to encode the majority of the agent's intelligence, and to provide the behaviour and sensory primitives. BOD includes a set of heuristics for recognizing when intelligence should be refactored either from a plan towards a behavior module or from a module into a plan. BOD and POSH have now been adopted or recommended by a number of leading thinkers and toolkits in AI, including Pogamut [21], RePast [28] and AIGameDev [6].

Recently, a graphical editor for POSH plans has been developed as part of the Pogamut effort. Its new version is used in the present study (Fig. S1).

3 Method

3.1 Experimental design

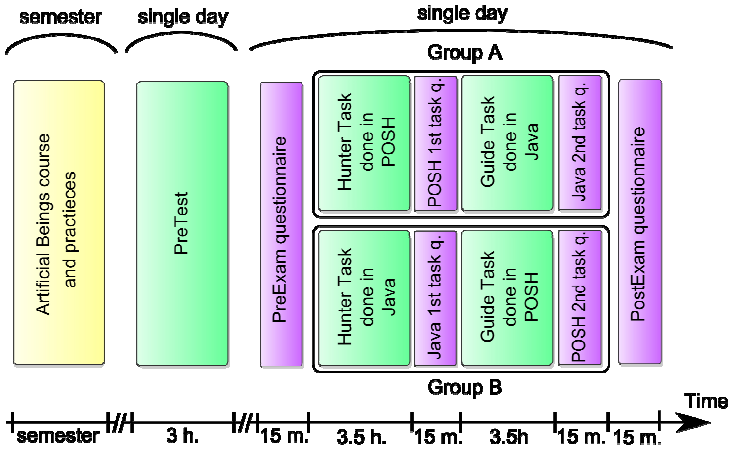
As explained earlier, the study compares the usability of an academic reactive planner, POSH, and an unenhanced classical programming language, Java. Low-level

behavior primitives were prepared for both groups in advance by the authors of the study. The set of primitives were fully sufficient for solving the presented tasks.

The study was set in an AI course for computer science students in Charles University in Prague. The syllabus of course is described in [20, 22]. Subjects were given a pretest (3 hours) after the course to ensure that they have acquired elementary skills for solving sub-problems from the final exam. Only subjects that have passed the pretest were admitted to the final exam.

The final exam was structured to obtain comparative data on Java and POSH usability. In the final exam, each subject had to solve two tasks, the Hunter Task (3 hours) and the Guide Task (3 hours), see Sec. 3.3. Subjects were split into two groups, Group A and Group B. Group A was instructed to solve Hunter Task in POSH first and Guide Task in Java second while Group B was instructed to solve Hunter Task in Java first and Guide Task in POSH second. For both tasks, syntax highlighting was available for Java and a graphical editor for POSH plans (Fig. S1).

Figure 1. The course of the experiment.



Subjects were given 4 questionnaires in total during the exam (15 minutes each). There was a 30 minutes long break for a light lunch between the tasks. The course of the experiment is summarized in Fig. 1. Subjects were informed that the study will take about 8 hours in total in advance, but the structure and the exact content were revealed only during the study. The assignments were administered immediately prior to each task and the subjects given 30 minutes to read them.

3.2 Participants

We recruited 30 students for the study out of 52 attendants of the AI course. The study was the course's final exam and if students succeeded in its both parts, they were given a final grade based on their agent's performance. Students had the option of withdrawing from the study if they preferred a different kind of final exam.

We excluded 3 students from the analysis due to data incompleteness. In total, we analyzed data from 27 students of which 2 were female. Students were sampled into two groups. Due to the low number of subjects, the groups were not assigned to conditions entirely at random. Rather the students were ranked by their ability as determined by their pretest performance, and then the two groups were matched with as close to equal sums of rank status as possible. The number of students according to their years of study and assigned groups is presented in Tab. S1.

3.3 Materials

The Course. The students attended an introductory course on the control of virtual characters. The course is intended for students without previous AI or 3D graphics knowledge but with previous programming experience. Only students from the second or a higher year of study can attend. The course comprises of 12 theoretical lectures (90 minutes each) and 6 practical lessons at computers (90 minutes each). The theoretical classes are detailed in [20, 22]. During practical lessons, the students are taught how to work with Pogamut 3 platform library (2 lessons) and develop behavior of virtual agents using both Java (2 lessons) and POSH (2 lessons) [23].

The Pretest. The general aim of the Pretest was to rule out subjects that were not sufficiently prepared for the final exam. Unprepared subjects would bias the data as they would likely fail during the final exam which would influence their answers in questionnaires.

The Pretest task was to create an agent capable of exploring the environment of UT2004 game and collect items of a specific type only. The agent had no adversaries in this task. Subjects were not given behavior primitives in advance; they had to create them in Java for themselves. Regarding programming of a high-level behavior, subjects had the opportunity of choosing between Java and POSH. This approach was chosen to test the level of subjects' comprehension of the Pogamut library so that they would be able understand behavior primitives provided to them during the final exam.

Three programmers skilled in VR technology solved the pretest task in advance to calibrate the difficulty of the test. The time allotment (3 hours) was at least three times longer than average time needed by these programmers to finish the task. Subjects had 3 attempts to pass the Pretest. Most passed on their first attempt.

Task Hunter. The Hunter Task was designed as a game-like scenario. Subjects were to create an agent (called Hunter) that explores the environment collecting blood samples of another computer-driven agent called Alien either by finding them around in the environment or by shooting Alien. Alien was an adversary agent that was capable of killing Hunter when nearby. If Hunter or Alien got killed, they were restarted in the environment far from each other. In addition, Hunter started with no weapons. Thus, the AI behavior must correctly prioritize the following intentions: 1) finding a weapon, 2) collecting blood samples, 3) responding to Alien. For instance, the Hunter agent should stop pursuing a blood sample item and responded to Alien if Alien has approached, otherwise Hunter could be killed resulting in the loss of weapons and blood samples collected so far.

In contrast to the Pretest, subjects were given a full set of behavior primitives (`canSeeEnemy`, `runToItem`, `shootEnemy` etc., see list in Tab. S2) that were sufficient to solve the task. All behavior primitives were carefully commented inside the code to make their usage clear. Action primitives did not contain any decision making logic, e.g., `shootEnemy` action did not contain any checks whether the agent has a loaded weapon to shoot from or whether the enemy is close enough for the weapon to be effective. Such logic was to be created by each subject using proper sensors, e.g., `hasWeapon` and `getEnemyDistance` (example can be seen in the Fig. S1). The task was again solved by two skilled programmers in advance using these primitives and their feedback was used to adjust them.

After filling in a pre-exam questionnaire, each subject was given the assignment written on the paper and was provided a sufficient time (30 minutes) to read it and ask questions to clarify any ambiguities. Group A was then instructed to solve the task in Java while Group B in POSH. Time allotment was 3 hours, which is roughly three times more than was required by the skilled programmers. Both groups had the same set of primitives. The POSH version of the primitives differed only in implementation details so that they could be easily used inside POSH reactive plans.

Group A and Group B were working in parallel in two different rooms. Subjects were not allowed to cooperate on the solution but they were allowed to utilize any documentation about the Pogamut library available on the Internet [24].

Task Guide. The Guide Task was designed to be more cognitive than the Hunter Task. Subjects were to create an agent called Guide that can find a Civilian agent inside the environment and guide it back to its home. The Civilian agent was created to wander aimlessly around the environment far from its home unless the Guide agent instructed it otherwise. The Guide agent must communicate with the Civilian agent if it wants the Civilian agent to follow its lead. The communication has a fixed and rather simplistic protocol described in the assignment (see Tab. S3).

Communication was reliable and the two agents could hear each other up to a specific distance. Apart from finding Civilian, there were three obstacles that Guide had to overcome in order to successfully lead Civilian home. First, Civilian was willing to start to follow Guide only if it can see it. Second, if Civilian lost Guide from view, it stopped following. Third, Civilian was created to be absent-minded and ceased to follow the Guide agent from time to time for no reason. Thus, the challenge was not only to find Civilian and persuade it to follow the Guide agent to its home, but also to constantly observe whether Civilian is doing so.

As in the previous task, subjects were given a full set of behavior primitives (Tab. S4) and the task was tested by two skilled programmers both in Java and POSH. The only exception was the handling of the communication was always in Java, but it was sufficient to write three lines of Java code to solve the task in the POSH variant.

Group A was instructed to solve the task in POSH while Group B in Java. Everything else (the assignment description, the space for questions, the prohibition of cooperation, the allowance of Internet usage, slight differences in the POSH primitives) remained the same as in the previous task.

3.4 Questionnaires

Every subject was given four questionnaires in total. Questionnaires were: 1) PreExam questionnaire, 2) Hunter Task questionnaire (in Java and POSH variants), 3) Guide Task questionnaire (in Java and POSH variants) and 4) PostExam questionnaire. The timing of administration of each questionnaire is pictured in Fig. 1.

The PreExam questionnaire contained questions about the subject's biographical background and their AI/Agent/Programming literacy. Only relevant results are presented in this paper. The main questions for the present interest are: "How many person-months of programming/AI/Java experiences do you have?" and "How many hours have you spent experimenting with Pogamut at home?"

The two task questionnaires were designed to elicit data about comprehensibility of sensory and behavior primitives and subjects' preferences for the programming formalism used in the task. The main questions for present interest are: "Did you find POSH/Java sensor/action primitives comprehensible?" (1: I had a lot of troubles understanding them, 3: I did not understand a few primitives, 5: I had no troubles at all, everything was perfectly clear.), "Did you find the number of POSH/Java sensor/action primitives sufficient?" (1: totally insufficient, 3: I had to create a few for myself, 5: totally sufficient), "Which formalism do you prefer, Java or POSH?" (1: strong Java preference, 2: weak Java preference, 3: can not tell which is better, 4: weak POSH preference, 5: strong POSH preference.

The PostExam questionnaire contained many questions about the comfort of the Pogamut library API, Java, POSH GUI and other features of the Pogamut platform. It also contained the final question about the overall preference between POSH and Java: "Which formalism do you generally prefer for high-level behavior specification, POSH or Java?" (1: strong POSH preference, 2: weak POSH preference, 3: can't tell which is better, 4: weak Java preference, 5: strong Java preference). Subjects were also given a space for a free-text explanation of their answer.

The POSH/Java preference question was given three times in total and they have appeared in both (POSH/Java) variants of task's questionnaires. Our aim was to observe subject preferences with regard to the different tasks (Hunter Task vs. Guide Task) they had to solve as well as their overall. The questionnaires were not anonymous so we were able to pair them with concrete agents later on (see 4.2).

3.5 Data analysis

Answers of subjects from questionnaires of both groups were analyzed. We used χ^2-tests of independence to confirm that both groups had same or different language preferences. As the number of subjects in each group is rather small, we have grouped subjects with Java/POSH preferences into 3 classes (instead of 5) for the purpose of the χ^2-tests. Answer 1-2 is considered as *Java preference*, answer 4-5 as *POSH preference* and answer 3 as indifference.

Additionally, all agents were tested for quality. We executed a corresponding task scenario for every agent 15 times and checked whether the agent fulfilled the task's objective within the time frame of 10 minutes. We marked every run with either 0 (agent failure) or 1 (agent success). Average number of successes was counted as the

agent success rate (ASR). Even though every run was identical (the same environment setup was used, the same starting positions of bots were used, the same random seeds, etc.), we had to perform multiple runs due to small non-determinism caused by UT2004 and by asynchronous execution of agents' behaviors which resulted in different outcomes from the behavior deliberations.

ASR was taken as the degree of agent quality. An ASR of 1 indicates the agent always succeeded, while an ASR of 0 indicates the agent always failed – real values could fall between these. Logistic regression was used to identify relationships between the agent quality and the chosen technique, subject experiences and their understanding of the provided primitives. The regression was made for every task/group combination (4 regressions) as well as for all agent runs for Task 1 and for Task 2 (combining data from Group 1 and Group 2) and is presented in 4.2.

There were 4 questions testing subject understanding of the behavior primitives. For the subsequent analysis, we averaged responses of these questions and used this average as the *Primitives apprehension* variable.

4 Results

4.1 Comparison of the two groups with regards to subjective Java/POSH preference

The attitude of the students towards the languages in the two tasks is shown in Fig. 2, 3, S2-S7 together with their means and standard deviations.

Regarding the first task, Group A exhibits a strong preference to POSH (Hunter in POSH) while Group B (Hunter in Java) was more indifferent.

Figure 2. Left: Group A, Hunter Task (in POSH), Java/POSH preference. Right: Group B, Hunter Task (in Java), Java/POSH preference.

Ans.	#	%
1	0	0
2	2	15.4
3	1	7.6
4	3	23.1
5	7	53.9
Mean	4.15±1.14	

Ans.	#	%
1	0	0
2	5	35.7
3	4	28.6
4	1	7.1
5	4	28.6
Mean	3.29±1.27	

The contingency table of Java/POSH preference after the first task is shown in Tab. 1. The preferences in Group A and B are not significantly different (p-value = 0.12).

Table 1. Contingency table of the Java/POSH preferences after the first task.

	Java pref. (1-2)	*Can't decide (3)*	*POSH pref. (4-5)*	*Total*
Group A	2	1	10	13
Group B	5	4	5	14
Total	7	5	15	27

Concerning the second task, Group A (using Java) was indifferent and Group B (using POSH) exhibited preference to Java (Tab. 2). The preferences in Group A and B are not significantly different (p-value = 0.36). In general, the students shifted their preference to Java after the second task, which is summarized by Tab. S5.

General preference between Java and POSH, as assessed by PostExam questionnaires, is not a clear one. The preferences in Group A and B were significantly different with Group A preferring POSH while Group B preferring Java (p-value = 0.01) (summarized in the Tab. 3).

Figure 3. From left to right: i) Group A, Guide Task (in Java), Java/POSH preference, ii) Group B, Guide Task (in POSH), Java/POSH preference, iii) Group A, PostExam, Java/POSH preference, iv) Group B, PostExam, Java/POSH preference.

Ans.	#	%	Ans.	#	%	Ans.	#	%	Ans.	#	%
1	2	15.4	1	3	21.5	1	0	0	1	3	21.5
2	4	30.8	2	7	50.0	2	3	23.1	2	6	42.8
3	1	7.6	3	2	14.3	3	2	15.4	3	4	28.6
4	4	30.8	4	1	7.1	4	3	23.1	4	1	7.1
5	2	15.4	5	1	7.1	5	5	38.4	5	0	0
Mean	3.00±1.41		Mean	2.29±1.14		Mean	3.77±1.19		Mean	2.21±0.86	

Table 2. Contingency table of the Java/POSH preferences after the second task.

	Java pref. (1-2)	*Can't decide (3)*	*POSH pref. (4-5)*	*Total*
Group 1	6	1	6	13
Group 2	10	2	2	14
Total	16	3	8	27

Table 3. Contingency table of the general Java/POSH preferences as answered in the PostExam questionnaire.

	Java pref. (1-2)	*Can't decide (3)*	*POSH pref. (4-5)*	*Total*
Group A	3	2	8	13
Group B	9	4	1	14
Total	12	6	9	27

4.2 Comparison of the two groups with regards to objective task solution quality

Logistic regression was used to identify relationships between an agent's quality (dependent variable) and chosen technique (Java or POSH), subject experiences and apprehensions of provided primitives. The parameter for the group was statistically insignificant and was left out from the model for the sake of simplicity. We have created 3 models (using data from both Group A and B, from Group A only and from Group B only) for both tasks (6 models in total).

Models description. The models' parameters are summarized in Tab. 4. Some dependencies between model variables and agent's quality are presented in Figs. S8 –

S10. Every figure contains graphs for Task 1 (left) and Task 2 (right) models separately. Models using data from both groups contain the additional discrete variable *Technique* (Java / POSH), therefore they are visualized with two graphs separately in each picture (for the Java and POSH cases separately). As all models amount to a function from the n-dimensional space (yielded from the Cartesian product of model variables' ranges) into <0;1> (agent success rate, model dependent variable), every presented graph can be seen as a planar cut through chosen variable of the whole model's n+1-dimensional graph where all other variables are fixed at data's means.

Tasks comparison. Task 1 was solved considerably better by subjects from higher years of study (Fig. S8, left). The data for Task 1 also suggests that subjects' comprehension of provided primitives affects the quality of their agents (Fig. S9, left); this is more pronounced in Group A's subjects. Additionally, solutions from Group B (implementing the Hunter agent in Java) indicate correlation with previous Java experiences (Fig. S10, left). The chosen technique (Java or POSH) did not influence the agents' success (see first row *POSH-influence* column in Tab. 4) in Task 1.

The interpretation of results of Task 2 is not as clear. Task 2 was also sensitive to Java experience as well as primitive comprehension (Fig. S10, S9 right), but results were more widely distributed this time. Also, agents of Group B driven by POSH did considerably worse than agents of Group A that were controlled by Java (see the fourth row *POSH influence* column in Tab. 4).

Table 4. Logistic models of agent success with respect to programming technique, subject's year of study, his/her experiences and primitives comprehension. Every row contains the parameters of one model. Column *POSH-influence* (discrete variable) explains how the probability of an agent's success changes when the agent was programmed using POSH (present only when data from both groups are used). All other columns (continuous variables) show how respective variables contribute to ASR. *Odds ratio* describes how the variable influences the probability of an agent's success. Values greater than one indicate that the probability grows proportionally with the variable and vice versa. Values in bold are discussed in Section 5.

Data used	Model fit comp. against empty model	POSH influence		Year of study		Java experience		Pogamut used at home		Primitives comprehension	
	P-Value	Odds ratio	Sig.	Odds ratio	Sig.	Odds ratio	Sig.	Odds ratio	Sig.	Odds ratio	Sig.
GA+B, T1	10^{-12}	**1.10**		**2.08**	***	1.08		0.96		**2.58**	***
GA, T1 (POSH)	10^{-10}	X		**2.10**	**	1.19		1.04		**1.24**	***
GB, T1 (Java)	10^{-6}	X		**1.81**	***	**1.30**	**	0.96		**0.74**	
GA+B, T2	10^{-5}	**0.44**	**	0.88		**1.11**	**	1.05	.	1.58	*
GA, T2 (Java)	0.057	X		0.99		**0.91**		0.91	*	2.37	**
GB, T2 (POSH)	10^{-7}	X		0.81		**1.09**		1.23	**	1.46	

Significance (P-Value): 0 < *** < 0.001 < ** < 0.01 < * 0.05 < . < 0.1

5 Discussion

This pilot study compared the usability of an academic reactive planning system to the usability of a common programming language when applied to programming the behavior of virtual agents in 3D game-like tasks. The POSH reactive planner empowered by a graphical editor of plans was chosen for the former and the Java programming language for the latter. This quantitative experimental study is, to our knowledge, the first in the field of virtual agent programming techniques (but see also [29]). The purpose of the study was twofold. First, we aimed at investigating objectively the usability of the two techniques, making a small step towards the grand goal: isolating features that contribute to usability of different approaches to control virtual agents in 3D videogames and simulations. Secondly, we aimed at answering the question whether the chosen experimental method *per se* is promising for future studies. We now discuss these two points.

5.1 Results

Summary of the data. The answer for the question of usability of Java and POSH has two sides which are intertwined. First, there is a subjective answer of comfort in using a chosen system as presented in Sec. 4.1. Second, there is an objective answer that comes of assessing the quality of agents as presented in Sec. 4.2.

Regarding the subjective answer, there are two main outcomes. a) Subjects, in general, reported that they preferred POSH for the first task (Fig. 2, S2, S3; Tab. 1) while they preferred Java for the latter (see Fig. 3, S4, S5; Tab. 2). b) Group A subjects tend to prefer POSH while Group B subjects tend to prefer Java (Tab. 3).

The objective answer as showed by logistic regression indicates several outcomes. c) students in a higher year of study tend to perform better in the first task while there was no such influence in the second task (see Fig. S8) d) previous Java experience was important in Task 1 in Group B (using Java in that task) but not in Task 2 in Group A (using Java in that task) (Fig. S10, left; Tab. 4), e) comprehension of the provided primitives was high in general (Fig. S9 left; means in both tasks were higher than 4.1) and seems to influence ASR a bit (Fig. S9 left; Tab. 4), f) the first task was done equally well in both POSH and Java (see *Odds ratio* of *POSH influence* in the first row of Tab. 4) while in the second task, subjects using POSH performed significantly worse (see *Odds ratio* of *POSH influence* in the third row of Tab. 4).

General comments. Arguably, the main underlying theme is that the data indicates different outcomes for the two groups. Why? Let us start with comments on distribution of subjects into Group A and B with respect to major variables (Comment 1), proceed with comments on several uncontrolled variables that may have influenced the outcome (Comments 2, 3, 4), and finally return to the individual outcomes A-F above.

1. Is the average programming experience of the subjects the same for the two groups? Tab. S6 indicates that Group B may have consisted of slightly more Java experienced subjects, but the difference between the groups is rather small. Data for the total previous programming experience look similarly (note that mean is not a

useful aggregative variable here since the learning curve is not linear). Students from Group B also have higher years of study on average (A: mean=3.3; SD=1. B: mean=4.4; SD=1.5). This is the outcome of the rank-based sampling procedure, which will be commented in Sec. 5.2. For present purpose, it is important that Group B may have comprised slightly more experienced programmers on average than Group A.

2. Subjects were undergoing a coding marathon as the final exam lasted 8 hours so the results from the second task could have been biased by subjects' tiredness. However, it seems reasonable to assume that both groups were equally tired.

3. It may be that the second task is harder in general, independently of the tiredness. We did not consider the complexity of tasks beforehand; therefore we have asked *post hoc* four independent VR experienced programmers to judge tasks' complexity out of the assignments (they did not perform them, we have just presented them written assignments) and task suitability for the chosen technique. The second task was perceived as easier only by one of them; the others thought that the second task is harder. Their comments regarding the suitability of techniques diverged.

4. It also may be that POSH fits better for solving the first task while Java for the second. This idea is actually supported by free-report parts of questionnaires. Some subjects indicated that Java was more suitable for the second task while none the other way round. Some subject's comments to the 2nd task:

"There were more if-then rules in the first task than here, therefore POSH would have suited the first task more, using it here was mere overkill."

"Using POSH for this task would be a nuisance."

"In contrast with the first task, this was too complex to niggle with POSH plan graphical editor. It was better to address it in Java."

Main interpretation. In our opinion, the most plausible explanation of the results is that they are produced by combination of two effects: the fact that the second task can be more easily solved using Java (unlike the first task), and the fact that the graphical drag&drop editor and POSH (it is not clear which of these or whether both of them together) is more appealing to a less skilled audience and such an audience can use it more effectively than Java. This statement agrees with Results (A) and (B) and partly with (C), and is further supported by Comments 1 and 4. Of course, our data only indicates that this can be the case; a useful hypothesis for further testing rather than a conclusive result.

It is also possible that the essential difference was that Task 2 was best completed by altering or adding to the provided the primitives. Because of the way POSH was introduced with the emphasis on the graphical tool, most subjects appeared to feel obliged not to alter any Java code while they were in the POSH condition. One student did provide an exceptionally good agent in Task 2 by combining POSH and altered Java primitives. This strategy is more in keeping with the way POSH is presented in the academic literature as a part of a development methodology (Behaviour Oriented Design) rather than a stand-alone approach. However, only one exceptional programmer tried this strategy.

Another way of looking at the data is that POSH scored surprisingly well (Tab. 1, 8) given many subject's initial Java experience but no initial POSH experience. Investigation of steepness of the learning curve might be fruitful in the future. Useful

information could also come out of studies of programmers already skilled in using an agent-based technique. Sadly, finding such a subject pool is presently a difficult task.

It is not surprising that understanding the primitives (Result (E)) has a positive effect on ASR. In fact, the influence is rather small, which is most likely caused by a ceiling effect: the average understanding of primitives was high in general, suggesting that our primitives were well chosen, prepared and documented.

Several questions remain open. We do not know why there was no influence of the students' years of study on the agents' performance in the 2nd task (Result (C), 2nd part); perhaps the assignment was not sensitive enough, or perhaps the difference on the 1st task indicated more advanced students become more adept at a new problem more quickly, whether through learning more quickly or due to being less stressed by exam conditions.

Concerning Result (D), it is not surprising that previous Java experience was important in Task 1 in Group B but not A, because the former group used Java. We do not know why previous Java experience had no influence on Group A in Task 2; again perhaps the 2nd assignment was not sufficiently sensitive to this variable. Also the sensitivity to previous Java experience in Task 1 suggests that classical programming languages are not as suitable for less-experienced programmers such as game designers as higher-level graphical tools and planning languages are.

Generalization. The results of this study indicate that academic techniques may in certain cases provide advantages over classical programming languages, but it is too soon to generalize based on the results of one study performed on two particular approaches and tasks. More studies are needed to obtain more conclusive data for further supporting or refuting such a claim. Nevertheless, it is a good sign for developers of various agent-based languages such as Jason [26] or GOAL [11]. Closer examination is needed to identify different complexities underlying virtual agents' development. Such examination may help recognize possibilities and limits of various techniques and uncover their strong and weak points. For instance, it may be that when augmented by drag&drop graphical editors (as POSH was in our study), some of these languages may be better suited than scripting languages for people with mediocre programming skills, such as some game designers. We believe that without such analysis the gaming industry would unlikely embrace academic techniques for virtual agent's development.

5.2 Lessons learned

As the comparative study of different techniques usable for virtual behavior development is new, we report lessons learned and suggest improvements for future studies. The main lessons are:

1) Performing the study in two consecutive parts promotes biased data on the second part due to subjects' tiredness. This can be addressed by altering the experiment design either by a) dividing subjects into 4 groups giving every group only *one* combination from the task-technique pairs, which would however require at least twice as many subjects, or b) by dividing each group into two subgroups, which

would solve both tasks each but in the reverse order; that would allow the statistical computation of the effect of tiredness, or c) to perform the second task in another day.

2) It would be beneficial to administer one more questionnaire during the pretest to obtain the initial preferences of subjects regarding the techniques compared in the study. In general, several other variables could be controlled better, e.g. the task difficulty (see also Comments 2, 3, 4 in Sec. 6.1).

3) The analysis should be complemented with qualitative studies to gain more insight. This may have several forms. a) Interesting data can be obtained by analyzing the agent code as has been previously done by Hindriks and colleagues [19]. We may still do this with the code from the present study. b) Focus groups or structured interviews can be conducted after the main study to obtain more precise explanations for subjects' preferences and their solutions' quality. c) Questionnaires should encourage subjects to describe reasons for their preference (the importance of this has been highlighted in Comment 4 in Sec. 5.1).

4) Attention should be paid to the evaluation's tasks. Each task should be judged not only for its general difficulty by programmers skilled with VR technologies, but also for its difficulty regarding the technique being tested. In general it is presumably a good thing to make assigned tasks varied so that an over-general conclusion is not reached without adequate justification. After the evaluation, subjects should be asked for their own assessment of the tasks to check if it correlates with the experts'. Note that both subject and expert assessment should be checked against actual quantifiable results.

5) The sampling procedure should be carefully considered. Evidently, even a rank-based sampling may produce unequal groups (with respect to some variables). When there are a lot of variables and a relatively small sample size, such an outcome may be inevitable. The sampling procedure will also be different for different questions asked, e.g., if one would like to assess group of experienced Java programmers against inexperienced ones, the criterion for sampling would be previous Java experience.

6) Pretests are important in order to ensure that students have certain minimal skills for the main study, e.g. from the present study the ability to understand behavior primitives. Pretests are also important for obtaining data for the sampling procedure.

5.3 Future work

Our results clearly indicate a need to continue with comparative studies and to begin to identify the different aspects of the complex task of virtual behavior development. We are considering performing another study this year, taking into account the lessons learnt, possibly utilizing GOAL [10] as an academic reactive planning technique that is based on the BDI paradigm. We may also run the same test again but with POSH clearly set forward not as an alternative to Java but rather as a way to supplement it. AI action selection systems are intended to simplify the development of agent intelligence, not to replace it.

6 Conclusions

This pilot study compared an academic reactive planning technique (namely POSH) against a common programming language (namely Java) with respect to their usability for programming behaviors of virtual agents in 3D game-like tasks. The study has investigated the performance of subjects' agents with respect to the technique used as well as subjects' preferences towards the techniques.

The conclusion, stated with caution, is threefold. First, from a general perspective, POSH scored comparable to Java. Second, in a more fine-grained manner, usability of Java and POSH seem to be task-sensitive and subjectively perceived usability of the techniques as well as objective quality of the subjects' agents with respect to the techniques may change with subjects' programming experience. Third, the experimental method is useful, but should be complemented by other approaches.

Taken together, these are promising news for agent-based control mechanism developers. Future studies are needed and they should focus on isolating mechanisms' features that contribute most to the mechanisms' usability for different target groups of users, e.g., game designers vs. programmers.

Acknowledgement. Students' assignments were developed at Charles University in Prague as part of subproject Emohawk developed under the project CZ.2.17/3.1.00/31162 that is financed by the European Social Fund and the Budget of the Municipality of Prague. The subsequent research was partially supported by grant P103/10/1287 (GA ČR) (C.B., J.G., J.B.), SVV project number 263 314 (J.G., M.B.), research project MSM0021620838 (MŠMT ČR) (C.B.) and by students grant GA UK No. 0449/2010/A-INF/MFF (M.B.). We thank our students. The questionnaires were designed by J.G. and C.B. Human data were collected respecting APA ethical guidelines.

References

1. Fu, D., Houlette, R., "The Ultimate Guide to FSMs in Games," AI Game Programming Wisdom II, Charles River Media (2004): pp. 283-302
2. Champandard, A. J.: Behavior Trees for Next-Gen Game AI. Internet presentation. URL: http://aigamedev.com/insider/presentations/behavior-trees (18.1.2011)
3. Schuytema, P.: Game Development with Lua. Charles River Media (2005)
4. UnrealScript programming language. URL: http://unreal.epicgames.com/UnrealScript.htm (18.1.2011)
5. Schwab, B.: AI Game Engine Programming. 2nd edition. Charles River Media. (2008)
6. AiGameDev community. URL: http://aigamedev.com/ (18.1.2011)
7. Rabin S.: AI Game Programming Wisdom series. URL: http://www.aiwisdom.com/ (18.1.2011)
8. Gamasutra webpage. URL: http://www.gamasutra.com/ (18.1.2011)
9. Magerko, B., Laird, J. E., Assanie, M., Kerfoot, A., Stokes, D.: AI Characters and Directors for Interactive Computer Games, Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference, San Jose, CA, July 2004. AAAI Press (2004)

10. Best, B. J. & Lebiere, C.: Cognitive agents interacting in real and virtual worlds. In Sun, R. (Ed) Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation. NY, NY: Cambridge University Press. (2006)
11. Hindriks, K. V., van Riemsdijk, M. B., Behrens, T., Korstanje, R.,Kraaijenbrink, N., Pasman, W., de Rijk, L.: Unreal GOAL Bots: Conceptual Design of a Reusable Interface. In: Agents for games and simulations II, LNAI 6525, pp. 1-18. (2010)
12. 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)
13. Partington, S.J., Bryson, J.J.: The Behavior Oriented Design of an Unreal Tournament Character. In: Proceedings of IVA'05, LNAI 3661, Springer-Verlag (2005)
14. Köster, M., Novák, P., Mainzer D., Fuhrmann, B.: Two Case Studies for Jazzyk BSM. In: Proceedings of Agents for Games and Simulations: Trends in Techniques, Concepts and Design, F. Dignum, J. Bradshaw, B. Silverman and W. van Doesburg (ed.), AGS 2009, LNAI 5920 (2009)
15. Dignum, F.; Bradshaw, J.; Silverman, B.G.; Doesburg, W. van (Eds.): Agents for Games and Simulations proceedings. LCNS 5920, Springer-Verlag. (2009)
16. Tyrrell, T.: Computational Mechanisms for Action Selection. Ph.D. Dissertation. Centre for Cognitive Science, University of Edinburgh (1993)
17. Bryson, J. J.: Hierarchy and Sequence vs. Full Parallelism in Action Selection', Simulation of Adaptive Behavior 6, Paris (2000) pp. 147-156
18. Bryson, J. J.: Action Selection and Individuation in Agent Based Modelling', in Proceedings of Agent 2003: Challenges of Social Simulation, Argonne National Laboratory (2003) pp. 317-330
19. Hindtriks, V. K., van Riemsdijk, M., Jonker, B., C. M., 2011, An Empirical Study of Patterns in Agent Programs: An Unreal Tournament Case Study in GOAL, PRIMA 2010.
20. Brom, C.: Curricula of the course on modelling behaviour of human and animal-like agents. In: Proceedings of the Frontiers in Science Education Research Conference, Famagusta, North Cyprus. (2009)
21. Gemrot, J., Brom, C., Kadlec, R., Bída, M., Burkert, O., Zemčák, M., Píbil, R., Plch, T. Pogamut 3 – Virtual Humans Made Simple. In: Advances in Cognitive Science, Gray, J. eds, The Institution Of Engineering And Technology (2010) pp 211-243
22. Brom, C., Gemrot, J., Burkert, O., Kadlec, R., Bída, M.: 3D Immersion in Virtual Agents Education In: Proceedings of First Joint International Conference on Interactive Digital Storytelling, ICIDS 2008 Erfurt, Germany, LNCS 5334, Springer-Verlag, Berlin (2008)
23. Artifical beings course, practical lessons slides. URL: http://diana.ms.mff.cuni.cz/pogamut-devel/doku.php?id=lectures (18.1.2011)
24. Pogamut 3 platform documentation. http://diana.ms.mff.cuni.cz/main/tiki-index.php?page=Documentation (25.1.2011)
25. Artificial beings course, final exam package. URL: http://diana.ms.mff.cuni.cz/pogamut-devel/doku.php?id=human-like_artifical_agents_2009-10_summer_semester_exam_info (18.1.2011)
26. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, Ltd. (2007)
27. Brooks, R.A: Intelligence Without Representation, Artificial Intelligence 47 (1-3) (1991) pp. 139-159.
28. Bozada, T.A., Perkins, T. K., North, M. J., Kathy, K. L., Simunich, L., Tatara, E.: An Applied Approach to Representing Human Behavior in Military Logistics Operations. In: Fall Simulation Interoperability Workshop, Simulation Standards Interoperability Organization, Orlando, FL USA (September 2006)
29. Desai, N.: Using Describers To Simplify ScriptEase. In: Master Thesis. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada. (2009)

Appendix

This section contains additional tables, figures and some additional text concerning presented study.

Reusable package. The package containing the assignment texts, Pogamut 3 platform, template agent projects and the scenario map can be downloaded from [25].

Table S1. Number of students in groups according to their types of study and years of study. Master students have number of years spent for their bachelor studies included into their years of study. Note that bachelor studies last 3-4 years typically and master studies takes usually extra 2-3 years.

Group A

Study / Year of study	*2nd*	*3rd*	*4th*	*5th*	*Total*
Bachelor	4	2	0	0	6
Masters	0	0	6	1	7
Total	4	2	6	1	13

Group B

Study / Year of study	*2nd*	*3rd*	*4th*	*5th*	*6th*	*8th*	*Total*
Bachelor	2	1	3	0	0	0	6
Masters	0	0	3	2	2	1	7
Total	2	1	6	2	2	1	14

Table S2. List of all behavior primitives that were provided in the Task 1.

Sensors		
class of primitives	*X parameter*	*Y parameter*
canSee X	AlienBlood, Ammo, Enemy, Weapon, WeaponOrAmmo	
get/know X	NavPointToExplore	
know X Y	SpawningPoint	AlienBlood, Ammo, Weapon, WeaponOrAmmo
	Spawned	AlienBlood, Ammo, Weapon, WeaponOrAmmo
get X Y	Random	NavPoint
	Nearest	NavPoint
	NearestVisible	AlienBlood, Ammo, AmmoOrWeapon, Enemy, NavPoint, Weapon
	NearestSpawned	AlienBlood Ammo Weapon WeaponOrAmmo
	AlienBlood, Ammo, Item, Weapon, DistanceToTarget	
has X	Ammo, Weapon	
is X	Moving, Shooting, RunningToItem, RunningToPlayer, RunningToNavPoint	
wantToSwitchToItem		

Actions	
run X	ToItem ToNavPoint ToPlayer
shootEnemy	
stop X	Movement, Shooting

Figure S1. Example of the code that the subjects were creating. Top: part of a POSH plan of the Hunter task as visualized by the graphical editor. Below: Hunter code in Java. The code and the plan were taken from an exemplary solution created by one of VR experienced programmers.

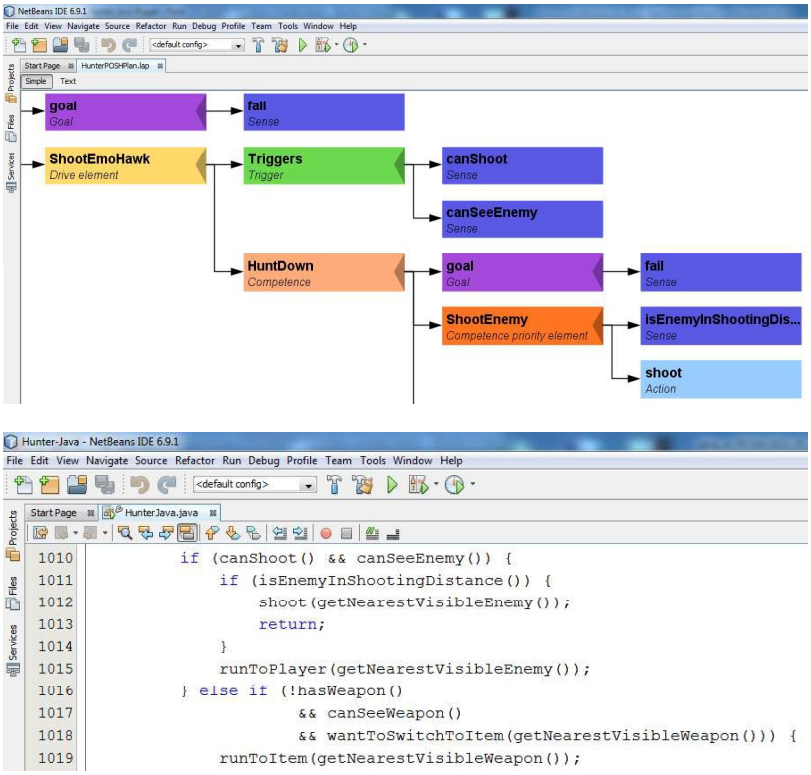


Table S3. List of possible commands that can be issued by the Guide and corresponding possible answers.

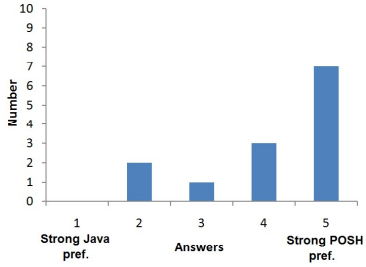
Guide commands	*Possible Civilian answers*
commandCivilianCanSee	answerAngry answerDontUnderstand answerCanSee answerCantSee
commandCivilianFollowMe	answerAngry answerDontUnderstand answerCantFollowingCantsee answerFollowingOk
commandCivilianStop	answerAngry answerDontUnderstand answerStopped
commandCivilianTurn	answerAngry answerDontUnderstand answerTurning

Table S4. List of all behavior primitives that were provided in the Task 2.

Sensors		
class of primitives	*X parameter*	*Y parameter*
can X Y	See	Civilian, Player
	FollowCivilian	
get/know X	NavPointToExplore	
get X Y	NearestVisible	NavPoint, Player
	DistanceTo	Civilian, NearestPlayer, Target
is X	CivilianFollowing, CivilianMoving, CivilianNear PlayerInTalkingDistance, Moving, RunningToPlayer,	

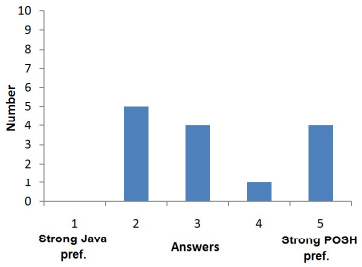
Actions		
command X Y	Civilian	CanSee, FollowMe, Turn, Stop
faceCivilian		
followCivilian		
run X	ToNavPoint, ToPlayer	
set X	CivilianSpeed, GuideSpeed	
stopMovement		

Figure S2. Group A, Hunter Task (in POSH), Java/POSH preference.



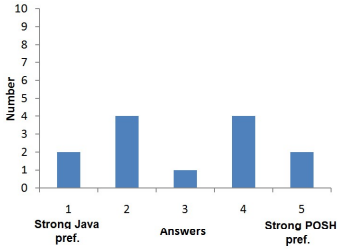
Ans.	#	%
1	0	0
2	2	15.4
3	1	7.6
4	3	23.1
5	7	53.9
Mean	4.15±1.14	

Figure S3. Group B, Hunter Task (in Java), Java/POSH preference.



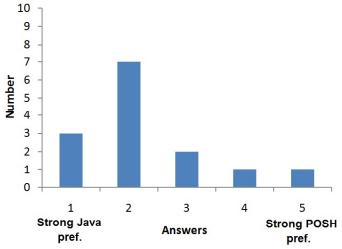
Ans.	#	%
1	0	0
2	5	35.7
3	4	28.6
4	1	7.1
5	4	28.6
Mean	3.29±1.27	

Figure S4. Group A, Guide Task (in Java), Java/POSH preference.



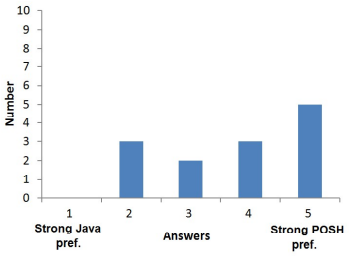
Ans.	#	%
1	2	15.4
2	4	30.8
3	1	7.6
4	4	30.8
5	2	15.4
Mean	3.00±1.41	

Figure S5. Group B, Guide Task (in POSH), Java/POSH preference.



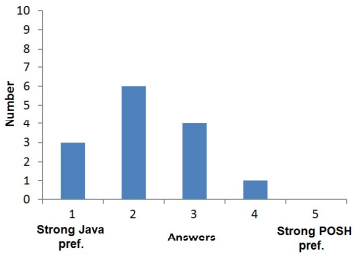
Ans.	#	%
1	3	21.5
2	7	50.0
3	2	14.3
4	1	7.1
5	1	7.1
Mean	2.29±1.14	

Figure S6. Group A, PostExam, Java/POSH preference.



Ans.	#	%
1	0	0
2	3	23.1
3	2	15.4
4	3	23.1
5	5	38.4
Mean	3.77±1.19	

Figure S7. Group B, PostExam, Java/POSH preference.



Ans.	#	%
1	3	21.5
2	6	42.8
3	4	28.6
4	1	7.1
5	0	0
Mean	2.21±0.86	

Table S5. Contingency table of the Java/POSH preferences shift.

Change in preferences of Group A

	T2 - Java	*T2 - Can't decide*	*T2 - POSH*	*Total (Task 1)*
T1 - Java	2	0	0	2
T1 - Can't decide	1	0	0	1
T1 - POSH	3	1	6	10
Total (Task 2)	6	1	6	13

Change in preferences of Group B

	T2 - Java	*T2 - Can't decide*	*T2 - POSH*	*Total (Task 1)*
T1 - Java	5	0	0	5
T1 - Can't decide	3	1	0	4
T1 - POSH	2	1	2	5
Total (Task 2)	10	2	2	14

Table S6. Table summarizing previous Java experiences in both groups (in man-months).

	0-1 months	*2-5 months*	*6-9 months*	*> 9 months*	*Total*
Group A	9	2	0	2	13
Group B	6	4	1	3	14
Total	15	6	1	5	27

Figure S8. Dependency of ASR on subject's year of study (Left – Task 1; Right – Task 2).

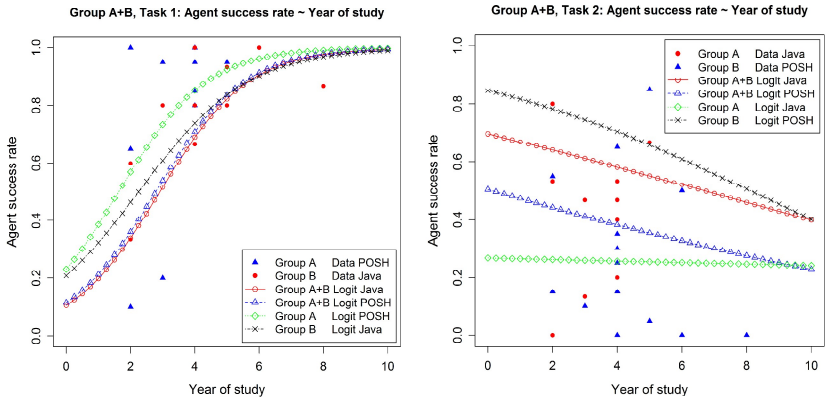


Figure S9. Dependency of ASR on primitives' comprehension (Left – Task 1; Right – Task 2).

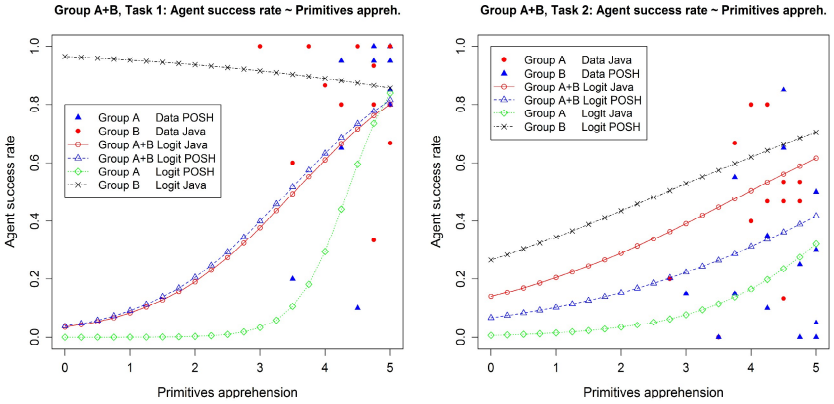


Figure S10. Dependency of ASR on previous Java experience (Left – Task 1; Right – Task 2)..