A Comparative Study of Programming Agents in POSH and GOAL

Rien Korstanje[1], Cyril Brom[2], Jakub Gemrot[2] and Koen V. Hindriks[1]
[1]*Delft University of Technology, EEMCS, Delft, The Netherlands*
[2]*Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic*

Keywords: Agent Programming, User Study, Novice Programmers, Advanced Programmers, Virtual Environment.

Abstract: A variety of agent programming languages have been proposed but only few comparative studies have been performed to evaluate the strengths and weaknesses of these languages. In order to gain a better understanding of features in and their use by programmers of these languages, we perform a study which compares the two languages GOAL and POSH. The study aims at advancing our knowledge of the benefits of using agent-oriented languages and at contributing to the evolution of these languages. The main focus of the study is on the usability of both languages and the differences between novice and more advanced programmers that use either language. As POSH requires Java programming experience, we expected novice POSH programmers to perform better on the tasks than novice GOAL programmers whereas we hypothesized this difference would not be observed between more advanced programmers. However, results suggest that there is no significant difference. The study does suggest that general experience and tooling support can make a difference. Analysis of the tasks and the observations made about the use of the languages, moreover, suggests ways to improve the experimental design in such a way that differences in usability of the frameworks could be established.

1 INTRODUCTION[1]

A variety of agent programming languages have been proposed in the literature, see e.g., (Bordini et al., 2005; Bordini et al., 2009), but only few comparative studies have been performed to evaluate the strengths and weaknesses of these languages. In order to gain a better understanding of the use of language features by programmers it is important to perform studies on various programming tasks in these languages. Such studies advance our knowledge of the benefits of agent-oriented programming and may contribute positively to the evolution of the paradigm.

In this paper we report on a study we performed in which the agent programming languages GOAL (Hindriks, 2009) and POSH (Brom et al., 2006) are compared empirically. Both languages take a different approach to agent programming. GOAL is a rule-based language for programming the decision making of cognitive agents that uses an embedded logical language for knowledge representation (e.g., Prolog). POSH is based on Behaviour Oriented Design and distinguishes between two layers: a high level rule-based planning layer on top of a layer that provides support for executing low level actions and senses.

The main aim of the study has been to analyse whether the different set of language features provided impact the performance of programmers on the same set of tasks. Moreover, we are interested in gaining a better understanding of the learning curve associated with agent-oriented programming languages. Given that the POSH framework builds directly on top of Java whereas the GOAL language uses Prolog for representing knowledge, we hypothesized that the learning curve would be less steep for POSH than for GOAL. More in particular, we expected novice programmers in POSH to perform better than novice GOAL programmers. For the coding tasks that we designed, we expected this difference to disappear for more advanced programmers that received more extensive training in programming in either language.

The study consists of a comprehension task and a number of increasingly more complex coding tasks. The coding tasks involved programming an agent that controls a character in a virtual environment called Emohawk that runs on top of the Unreal engine. Emohawk provides a fairly complex environment but is not as complex as the Unreal Tournament gaming environment. Using Emohawk enabled us to design

[1]This work also has been presented at the Second AAMAS Workshop on Cognitive Agents and Virtual Environments (2013)

Korstanje, R., Brom, C., Gemrot, J. and Hindriks, K.
A Comparative Study of Programming Agents in POSH and GOAL.
DOI: 10.5220/0005799601920203
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2*, pages 192-203
ISBN: 978-989-758-172-4

challenging but not too complex tasks. A basic agent program was provided to start with that subjects could inspect first and extend thereafter. By providing a basic program not only a useful starting point was provided but time needed to complete the tasks was also reduced. Subjects were asked to extend the basic agent in a number of consecutive coding tasks. During the experiment we collected data on the time it took to complete each task and asked participants to rate the usefulness of language features.

The paper is organized as follows. Section 2 discusses related work. In Section 3 we briefly discuss the virtual environment used in our study whereas Section 4 compares the agent programming languages used. In Section 5 we discuss the experimental design of the study. Section 6 presents results, which are further discussed in Section 7. Section 8 concludes.

2 RELATED WORK

In this paper we report empirical results that allow us to compare two agent programming languages. We can also compare agent programming languages analytically and we mention two examples of work in that area. (Hindriks et al., 1998) formally compare the agent languages AgentSpeak(L) and 3APL and shows that the former can be embedded into the latter. It follows that 3APL is at least as expressive as AgentSpeak(L) and any program that can be written in AgentSpeak(L) can also be coded in 3APL to do the same task but not necessarily vice versa. (Bryson, 2003) presents a theoretical comparison between POSH and other action selection mechanisms such as Environmental Determinism, Finite State Machines and Basic Reactive Plans. POSH was found to be more expressive than these other frameworks for programming reactive agents.

We briefly discuss some related studies that also aim at empirically evaluating agent programming languages. (Lillis et al., 2012) evaluate a toolkit called ACRE for conversation management between agents. The focus of this study is on agent communication and compares the performance of subjects that used the toolkit with subjects that did not have access to it. Results suggest that the toolkit can improve code quality and reduces the code base but not necessarily effort. The Agent Factory platform was used in the study but use of language features was not analyzed.

(van Riemsdijk et al., 2012) present a qualitative analysis of multi-agent programs for UNREAL TOURNAMENT written in GOAL by first year BSc computer science students. The aim of the study was to gain insight into practical aspects of agent development and to better understand some of the problems that agent programmers face. The method used in (van Riemsdijk et al., 2012) has been to analyse the multi-agent program code that was developed by the students. The paper discusses the actual use of language features in this code. The study did not involve an experimental design with programmers as subjects nor a comparison with other languages.

Agent programming languages have been empirically compared in only a few studies. (Gemrot et al., 2012b; Gemrot et al., 2012a) aimed at evaluating differences between programming agents in Java or POSH for the virtual environment Emohawk. The study, however, found that subjects using POSH and those using Java performed similarly. Both studies asked subjects to write code from scratch, i.e., no code was provided initially. The second study also included a comprehension task. Both studies lasted eight hours, making fatigue a potential factor that may have influenced the results found.

As we use a virtual environment called Emohawk that runs on top of the Unreal gaming engine, we briefly mention related work in the area of programming gaming agents. (Heckel et al., 2009) evaluated a tool called BehaviorShop and showed that the IDE allowed users with little to no experience in Artificial Intelligence to create high-level behaviors that control virtual characters. Subjects that were only given a brief lecture on the basic concepts of BehaviorShop were able to create fairly complex interactions. In our study we also found that, given more general experience with programming, a short tutorial was sufficient for subjects to code agents that control virtual characters. Our study is comparative, however, and different from (Heckel et al., 2009) it also investigates which language features are used.

3 EMOHAWK ENVIRONMENT

A challenging aspect of designing a comparative study for agent languages concerns the choice of environment with which the agents interact. Toy examples, such as the Blocks World discussed in (van Riemsdijk et al., 2012), for example, do not call for the use of more advanced features of a language, whereas the use of complex gaming environments such as UNREAL TOURNAMENT may pose too many challenges for a programmer to solve within the necessarily limited time available during an experiment (Gemrot et al., 2012a; Gemrot et al., 2012b).

For our study we have chosen to use the Emohawk environment. Similar to UNREAL TOURNAMENT, Emohawk runs on top of the Unreal engine.

But it provides a very different environment featuring a small town with several locations of interest and several characters that can display text, emoticons and animations (Bída and Brom, 2010). The size of the Emohawk environment is small compared to maps in UNREAL TOURNAMENT and thus presents less of a challenge than the UNREAL TOURNAMENT game.

Like UNREAL TOURNAMENT, Emohawk is accessible through a software environment called Pogamut. Pogamut provides functionality to manage characters in the Emohawk environment by means of modules. These modules provide a wide range of functionality and support various forms of locomotion, displaying emotions, communication with other bots, information, etc. They also manage information about the world and allow an agent to query this information. By combining modules, an agent can be programmed to do more complex tasks. For example, an agent can request the Players module for the nearest visible player (character) and then tell the Navigation module to bring another character to that player.

For comparing the performance of GOAL and POSH programmers, an environment should be used that provides similar access to environment primitives in both cases. This poses an additional (technical) challenge as agents written in either language must be able to interact with the same environment in a similar way. Even though this concerns a mainly technical aspect of our study, the way access is provided to the environment thus needs to be carefully designed to make sure that connecting to the environment does not significantly change the tasks that subjects are asked to complete. This also rules out, for example, the use of the UNREAL TOURNAMENT gaming environment in our study because of the significant differences between the interfaces that are available for connecting GOAL and POSH to this environment.

We have used Pogamut for connecting both GOAL and POSH to the Emohawk environment. POSH is integrated with Pogamut and can connect directly with it. In order to connect GOAL to Emohawk, an environment interface had to be designed and created that provides similar functionality. We used the Environment Interface Standard for this (Behrens et al., 2011). In order to make the same functionality provided by Pogamut available for GOAL and POSH programmers, we restricted the set of Pogamut modules that were made available in POSH to those that were present in the interface designed for GOAL or for which similar functionality could be provided in GOAL otherwise. Technically, this was realised for POSH by creating a custom context that provides access to the selected modules. For GOAL, the same functionality was either made available through percepts and actions in the interface or via Prolog code. The Prolog code in a GOAL agent represents knowledge that is provided by Pogamut modules that operate on more basic data available in the environment.

4 LANGUAGES USED: GOAL & POSH

Before we proceed, we introduce the main features of the agent programming languages GOAL and POSH used in our study and provide a brief comparison which highlights the main differences and similarities between the languages.

4.1 The GOAL Programming Language

GOAL is a *logic-based* agent programming language for programming *cognitive agents* (GOAL, 2015; Hindriks, 2009). GOAL agents maintain a mental state that consists of *beliefs* and *goals*. Agents derive their choice of action from their beliefs and goals. GOAL agents also use a *knowledge base* to represent conceptual and domain knowledge. The version of GOAL used in this study uses Prolog, a declarative programming language used for representing the knowledge, beliefs, and goals of an agent. A Prolog program consists of *Horn clauses*, which are logical rules and simple facts (Shapiro and Sterling, 1994).

GOAL is a *rule-based* language. The design philosophy of GOAL is that writing agent programs essentially means writing rules that determine for each situation that the agent finds itself in what it should do in that situation. Rules are ordered which allows for imposing a priority on what needs to be done first.

GOAL agents execute a reasoning cycle that consists of *two phases*. The purpose of the first phase is to process all *events* such as percepts and messages and make sure that the agent's mental state is up-to-date. In this phase the GOAL agent retrieves and processes all *perceptual information* available from the environment to update the beliefs and goals of the agent. In the second phase of the cycle agents decide what to do next. Typically, in this phase one *environment action* is selected and sent to an environment.

The concept of a *module* is a key programming construct in GOAL for structuring and writing larger agent programs. A module basically is a container for a set of rules. A GOAL agent program then is a set of modules. The `event` module corresponds to the first phase in the agent's cycle and is designed to support event processing whereas the `main` module corresponds to the second phase and is designed to support decision making. In addition, a special `init` module is available for initialising the mental state and other components of an agent. A programmer can also add and write its own set of modules for structuring and organizing code. GOAL also provides support for *communication* between agents.

The GOAL language is distributed with an Integrated Development Environment for coding, testing, and debugging in the Eclipse environment. It provides the usual program editing tools as well as tools to analyse and debug code.

4.2 The POSH Programming Language

POSH is a reactive action selection mechanism introduced in (Bryson, 2001) to simplify the construction of action selection for modular AI; see (Gemrot et al., 2012b) for details on the POSH language. A programmer used to thinking about conventional sequential programs is asked to first consider a worst-case scenario for his/her 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 and attempted whenever the agent can. It is the task of the programmer to write code that allows an agent to recognize that it can meet a goal. This needs to be done for all sub-goals: A perceptual condition must be specified that allows the agent to recognize if it can take an action that makes progress towards its goal (Bryson, 2001). Actions are small code chunks that control the agent briefly, so-called behaviour primitives.

(Bryson, 2003) provides a development methodology called Behaviour Oriented Design (BOD) for developing POSH agents. BOD emphasizes the above development process, and also the use of behaviour modules written in ordinary object-oriented languages (in our case, in Java) to encode the majority of the agent's intelligence, including its memory. These modules provide the behaviour and sensory primitives; methods calls are the interface between a high-level POSH plan and the low-level code of the behaviour modules. A POSH plan can be organized hierarchically in a tree-like structure.

Technically, a plan consists of production rules, very similar to modules in GOAL. Rules are also ordered by priority. In POSH, the antecedent or condition of a rule is a method call (i.e., a sensory primitive). The head of a rule is either a method call, i.e., a behaviour primitive, or a set of sub-rules also called a *competence*. At run-time, a POSH plan is periodically evaluated in order to determine the action, i.e., which behaviour primitive, the agent should execute.

A graphical editor for POSH plans is available and used in the present study.

4.3 Comparison

It is clear that GOAL and POSH share some features for writing agent programs that are similar. Both languages, for example, are rule-based. Rules are used in GOAL as well as in POSH for deciding which action the agent has to perform in a particular situation. Even though the terminology used differs, we can at least conceptually compare language features. In order to enable comparison, we introduce five important concepts related to programming agents and discuss how each language provides support for these concepts.

The Agent's State. A state represents everything that the agent knows about its environment. A POSH agent maintains a state by means of various Java modules which process messages received from the Pogamut environment that enables controlling characters in Emohawk (see also Section 3). A GOAL agent maintains a mental state by processing percepts received from the Emohawk environment via an environment interface that connects the agent to the Pogamut environment. A GOAL agent not only updates its beliefs but may also update its goals when it receives new percepts.

The Plan. In both languages, a plan consists of condition-action rules that allow an agent to decide on what to do next given its current state. In GOAL such a set of rules is called a module. In POSH such a set of rules is called a POSH plan.

Rule Conditions. Rules are condition-action rules. Rule conditions in GOAL consist of Prolog queries performed on the agent's beliefs and/or goals. Conditions of POSH rules cannot directly query the agent's state but use so-called 'senses' to evaluate rule conditions.

Rule Actions. POSH rules contain primitive and aggregate actions. Through primitive actions POSH has access to a large number of high and low level actions provided by Pogamut (Gemrot et al., 2009). A GOAL agent only has access to actions that are provided by means of an environment interface. In our study, that means that a GOAL agent cannot directly access actions that are made available by the Pogamut environment and only has access to actions in that environment that are made available explicitly in the environment interface used by the agent.

Perception. In GOAL the `event` module can be used to process percepts and store facts in the mental state. POSH has no such feature and relies on functionality provided by Pogamut to do this.

5 EXPERIMENTAL DESIGN

In our study, we compared how programmers used two agent programming languages. Subjects also used the Integrated Development Environments (IDEs) that were available for both languages. Subjects were provided with an initial agent and asked to complete several tasks which measured their ability to comprehend and modify agent program code. The coding tasks were divided into several sub-tasks that each built on the results of the previous task. Subjects were also asked to complete three questionnaires. Subjects were asked to program in either GOAL or POSH, and based on their experience classified as either novice or (more) advanced programmer. We thus have four different groups: GOAL novices, advanced GOAL programmers, POSH novices, and advanced POSH programmers.

5.1 Aim of the Study and Hypotheses

The aim of the study is to establish whether there are differences related to the usability of the programming languages GOAL and POSH. More specifically, we are interested in the learning curve of both languages. We therefore looked in particular at differences between novices that just started learning a language and more advanced programmers that had received more extensive training and practice.

The first hypothesis we formulated relates to the question how quickly programmers are able to familiarize themselves with either GOAL or POSH and how long it takes before they can successfully complete a coding assignment. We speculated that because POSH requires programming in Java, a language familiar to all participants, and GOAL requires programming in Prolog, novices would do better when using POSH rather than GOAL. This seems a reasonable hypothesis given that Prolog is perceived as difficult to learn for novices (Pane and Myers, 1996). Moreover, POSH plans, a key feature of the language, are represented visually which is generally considered to facilitate novice programmers. We thus hypothesized that there would be a difference in the learning curve which would it make easier for POSH novices to perform well on the tasks of our study.

Hypothesis I. POSH novices perform better on all tasks than GOAL novices.

Here, performance is measured in terms of successfully meeting the criteria of the task (see below) and the time-to-complete. Generally, time-to-complete is considered a relevant measure for establishing differences in usability.

As discussed, both GOAL and POSH provide similar though distinct language features for programming an agent at a conceptual level. Both languages provide the features needed for solving the tasks of the study. For a programmer that has been relatively well familiarized with either language, given the quite reasonable complexity of the tasks, we expected therefore that it should be possible to perform well on these tasks. For this reason, we did not expect advanced programmers to perform very differently. Therefore we hypothesized that more experienced programmers would achieve similar results in more or less the same time. In case we would observe different performance, we would take this as an indicator that there would be a real and significant difference between the languages.

Hypothesis II. Advanced POSH and GOAL programmers perform similarly on all tasks.

Finally, given that novice and more advanced programmers participated, we hypothesize that advanced programmers outperform novices. This does not necessarily have to be the case as advanced programmers may complicate things more than novices but seems a very reasonable assumption to make.

Hypothesis III. Advanced programmers outperform novices on all tasks.

5.2 Tasks

In task-based evaluations, the relevance of the results depends directly on the relevance of the tasks with respect to the purpose of the study. Given that we want to compare two agent programming languages, the design of relevant tasks is a challenge. In order to be able to perform a reasonable comparison, we need to avoid very basic or trivial tasks but at the same time design tasks that can be performed within a reasonable amount of time. (Gemrot et al., 2012b) illustrates that it is difficult to strike the right balance and report that in their comparative study of POSH and Java subjects became fatigued during an 8 hour session and could not successfully complete the given task. To avoid problems like this, we decided that the duration of the experiment should not take longer than 4 hours. Moreover, multiple tasks were designed that aim to measure performance on different skills, including program comprehension and code writing. A between-subjects design was used, i.e., each subject was asked to complete the tasks only in one language.

Due to time constraints in general and the inexperience of the novice users, for the coding tasks, we did not consider it feasible for subjects to create a rather complex agent from scratch. Therefore the choice was made to ask subjects to extend a given agent, by means of a set of incremental tasks. Each consecutive task was designed to be more difficult, starting with a rather basic task and ending with a task that required coding of more complex behaviour.

At the start of the experiment, subjects were provided with some time to familiarize themselves with the development environment of the language and were asked to run a given agent connected to the Emohawk environment. This allowed subjects to try and play with the software they were asked to use for completing the tasks. At this stage, we could provide additional explanation where needed, address questions, or resolve potential issues and thus make sure that very basic issues unrelated to the study would not arise during the experiment itself.

Subjects were provided with an initial agent program that performs a basic scenario in the Emohawk environment. The agent controls a virtual character and makes it do some work at home for a little while before it then decides to get a cup of coffee. The action of working and the need for coffee are graphically displayed in the environment by means of emoticons. After deciding to get coffee, the agent walks to the coffee shop, uses emoticons to order a double espresso, and then returns home to make a renewed attempt to work again. This agent was used in the comprehension task and provided the starting point for the coding tasks. To ensure comparable difficulty of the tasks subjects needed to complete, the agent was written in both POSH and GOAL following a predefined specification. We also made sure that the GOAL agent was able to provide similar functionality as the Pogamut framework provides (on top of which the POSH agent runs) by means of adding several Prolog rules as knowledge to this agent. For more details on the tasks and the instructions that subjects received, please see the materials we provided to subjects (Materials, 2015).

Subjects were also provided with the documentation that is distributed with GOAL and POSH. For POSH, this material consists of the Pogamut source code and documentation accessible through the IDE. For GOAL, its programming manual and a manual of the environment is provided. The lecture materials from the tutorial that introduced the language to novices were also provided to all subjects.

The experiment consists of two parts: the first part concerns a program comprehension task and the second part involved code writing.

5.2.1 Comprehension Task

For the comprehension task, subjects were provided with an agent that controls an entity in the Emohawk environment as discussed above that was written in either GOAL or POSH. They were asked to examine the program code and to answer a number of questions about the behaviour that the agent will display, the flow of control of the agent, the structure of the program code, and the interpreter that executes the agent. The comprehension task was designed to measure how well subjects comprehend the agent program provided to them. It also allowed subjects to familiarize themselves with the program code that they needed to extend in subsequent tasks.

5.2.2 Coding Tasks

In the coding tasks subjects were asked to extend the agent program they examined during the comprehension task in various ways. The tasks were presented in order of increasing complexity and each task builds on results obtained in the previous task. The main aim of this incremental set of tasks is to create an agent that implements the following scenario in the Emohawk environment: two people (entities controlled by agents) meet in a park, have a conversation, and thereafter visit the cinema together. The five coding tasks asked subjects to write code that:

1. makes the agent walk to the park, a location in the Emohawk environment,
2. makes the agent walk from that location to the cinema,
3. creates a second agent that is able to do the same (1,2 above) and makes the agents wait for each other in the park,
4. simulates a conversation between both agents, and
5. makes the agents walk side by side to the cinema.

Subjects were asked to write down the time they finished a question or task. This provided us with a measure of progress for all subjects throughout the experiment. To check whether subjects successfully completed a task we ran the agent, observed its behaviour, and checked whether this behaviour met a predefined list of criteria.

5.3 Questionnaire

Subjects were asked to complete three questionnaires to collect qualitative data. One questionnaire was given before the actual experiment, one questionnaire was given in between the two main parts of the experiment, and one questionnaire was given after completing the coding tasks (or when time ran out). Each questionnaire took about 15 minutes.

5.3.1 Experience Questionnaire

The first questionnaire we designed to obtain information about the subject (e.g., age), about prior programming experience in relevant languages, and about their experience with popular IDEs (e.g., Eclipse). Subjects were asked to rate their own skill in programming with Java, Prolog, POSH, and GOAL. Java is required for programming a POSH agent but also gives a subjective rating of programming experience in a common and popular language that is used extensively in education at both universities where students were recruited. Experience with Prolog is needed to program a GOAL agent.

The results from this questionnaire also provided us with a means to check whether the different groups that got different treatments (based on the various experimental conditions) were balanced in the following sense. Groups should have more or less the same experience in programming, should be familiar with at least one popular IDE, and age should be similarly distributed over groups.

5.3.2 Post Comprehension Questionnaire

The second questionnaire was given to subjects after they finished the comprehension task. This questionnaire asked subjects to rate on a five point Likert scale (strongly disagree to strongly agree) how useful they found selected language features for *understanding* the agent program provided to them. Questions were posed in the form "was feature X helpful". Subjects could also select a "not applicable" option for each feature. Additionally, they were asked to explain their rating in words.

5.3.3 Post Coding Questionnaire

After completing the second set of coding tasks, a final questionnaire was handed out to subjects. This questionnaire asked subjects to rate the usefulness of a language feature for completing the *coding* tasks. For each feature they were asked to rate their agreement with the question "was feature X helpful" on a five point Likert scale (strongly disagree to strongly agree). For each question there also was the option to select "not applicable". Additionally, subjects were asked to explain their rating.

5.4 Treatments and Subjects

Based on the use of either GOAL or POSH and the level of experience, we obtain four treatments: (i) POSH novice, (ii) GOAL novice, (iii) advanced POSH programmer, and (iv) advanced GOAL programmer. We use 'novice' and 'advanced' here in the sense that subjects that we label 'novice' had little experience with a languages whereas 'advanced programmers' had received a more extensive training. Subjects assigned to the novice condition were only given an introductory tutorial on GOAL or POSH of about two hours well before the experiment took place. The tutorial consisted of a basic lecture on language features that were needed to complete tasks and a demonstration of how to use the IDE of the language.

Subjects for the study were recruited from students attending University Delft University of Technology with experience in GOAL and students from University Charles University in Prague having experience with POSH.[2] None of the Delft University students were familiar with POSH and only very limited experience with GOAL was reported by the Prague University students. Therefore, we could assign Prague University students to the GOAL novice treatment and Delft University students to the POSH novice treatment. GOAL novices should at least have some experience with Prolog whereas POSH novices should at least have some experience with Java. Results shown in Table 1, obtained from the first questionnaire where subjects were asked to report on their own programming experience in terms of man months a language had been used, demonstrate that this was indeed the case. From self-reports on the use of either language and the skill rating we also can conclude that the subjects assigned to the expert treatments indeed were more advanced than the other students (e.g., skill is rated significantly higher; $p_{GOAL} < 0.005$, $p_{POSH} < 0.005$).

28 students from Charles University participated. For 18 of these students participating in the experiment was part of their final exam and students were graded on the performance of their agent. These students were assigned to the advanced POSH programmers condition. The other 10 students voluntarily signed up for the experiment and were assigned to the GOAL novice condition. 11 students from Delft University of Technology participated, all of which vol-

[2]All Delft students participated in the Multi-Agent Systems course in which GOAL is taught, see http://ii.tudelft.nl/trac/goal/wiki/Education. All Prague students participated in the course Artificial Beings (Brom, 2009) in which POSH is taught. Students took part in the study at the end of or directly after completing the course.

Table 1: Self-reports on Experience and Skill.

	Novice	Expert
GOAL Usage	0.2 (0.3)	0.7 (0.3)
GOAL Skill	1.5 (0.6)	2.8 (1.1)
Prolog Usage	1.8 (2.2)	0.7 (0.3)
Prolog Skill	2.2 (1.2)	3.0 (1.0)
POSH Usage	0.0 (0.0)	0.3 (0.4)
POSH Skill	1.0 (0.0)	2.5 (0.8)
Java Usage	5.5 (4.4)	2.6 (2.9)
Java Skill	3.6 (0.5)	2.5 (1.3)

unteered. These were divided equally over the POSH novice (6 students) and advanced GOAL programmers (5 students) condition, stratified by the results of a Java skill test. All subjects were given an incentive with an equivalent value of 20 Euros.

The total of 39 students that participated included 4 women and 35 men. Analysis of age shows there is a clear difference in age distribution (one way ANOVA, $p < 0.001$; $F = 15.01$) between groups assigned to different conditions. This was the case because, contrary to the expectations, a large number of senior students enrolled in the course in Prague from which subjects were recruited. Finally, analysis of experience with IDEs shows that almost all students except for a few students assigned to the GOAL expert condition had experience with either the Eclipse, Netbeans or Visual Studio IDE.

6 RESULTS

In this Section, we report on the use of language features by and the performance of subjects in the experiment. For both language usage as well as performance (time-to-complete) self-reports of subjects were used. The data obtained from Likert scale questions is presented by the associated mean and standard deviation which indicates the general tendency of a group assigned to one of the four treatments. Comparisons between treatments are analyzed using chi-square or Fisher's-Exact test whenever the categorical minimum was not reached.

6.1 Comprehension Task

The comprehension task provided little difficulty for any of the subjects. Figure 1 shows that all participants finished this task in roughly the same time (one way ANOVA, $p = 0.707; F = 0.46$; time reported in minutes). Due to the low number of subjects in the POSH novice and advanced GOAL groups, we report the individual completion times. Most questions were

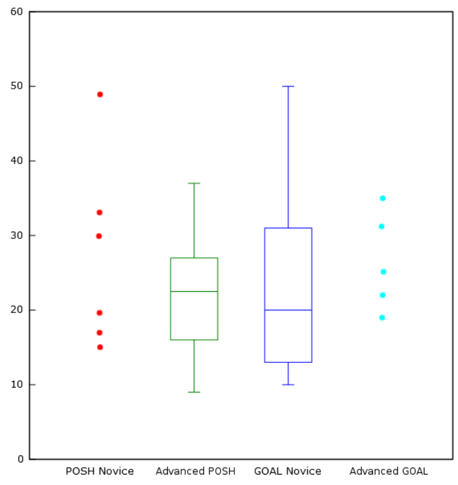

Figure 1: Comprehension task.

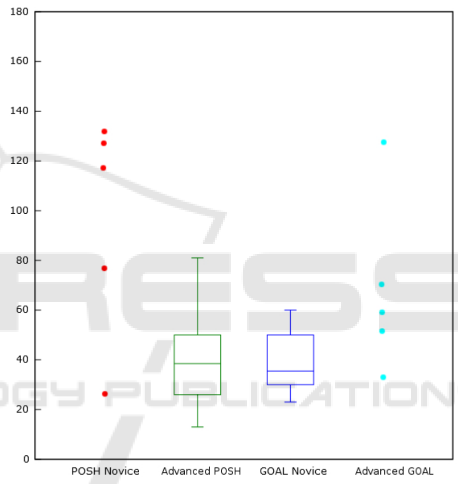

Figure 2: Coding tasks 1-3.

answered correctly. Only one question (out of 8 questions) about the number of execution cycles that were needed to perform a specific part of program code turned out to be difficult for novices. This is also reflected in the high self-rated satisfaction with the completion of the task (3.9 ± 1.1) for all groups.

6.2 Coding Tasks

The coding tasks were considerably more difficult and not all participants were able to complete all tasks in time. The difficulty of these tasks is also reflected in the slightly lower satisfaction ratings (3.4 ± 1.1) for these tasks. In particular, some subjects in the POSH novice and advanced GOAL programmers groups were not able to complete the 4th and 5th task. As we do not have completion times for all subjects, these tasks are not included in the discussion below.

The 4th coding task, simulating a conversation using emoticons, was found to be difficult for quite a number of subjects. Most of these subjects had problems with coding the right length that the emoticons should be displayed. The 5th task, where one agent should be coded to follow another, was difficult for all subjects. Most subjects had problems with establishing in their code that the the first agent should start following the other agent (task 5). Given that all subjects had trouble with completing this task, the task may have been too challenging.

Analysing the cumulative completion times for the first three tasks (see Figure 2), the advanced POSH and GOAL novice groups from Prague use significantly less time than the groups from Delft University X ($p < 0.001$; $F = 9.31$). This may be explained by difference in age, years of enrolment, and years of programming experience, but separately these only moderately correlate (R^2 of 0.31, 0.26, 0.17, respectively).

Looking more closely at the scatter plots in Figures 3, 4, and 5 shows that there are clear outliers for each of the three plots, including in total 5 subjects of the POSH novice and advanced GOAL programmers groups. While there is a difference in performance between the groups from Delft and Prague University, further analysis with Tukeys HSD shows there is little difference within these groups ($p_{Delft} = 0.77$; $p_{Prague} = 0.99$). The programming language thus does not appear to influence the completion times for the subjects from Prague University. A similar observation applies to the subjects from Delft University but due to the low number of subjects no firm conclusion can be drawn here.

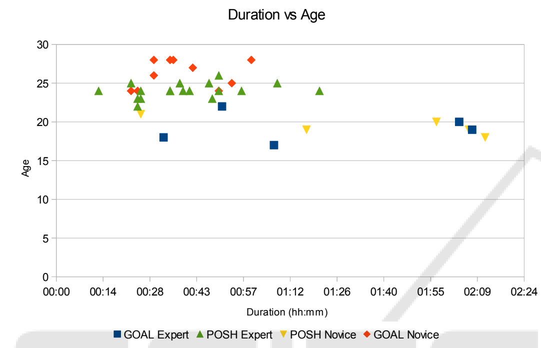

Figure 3: Cumulative time-to-complete versus age.

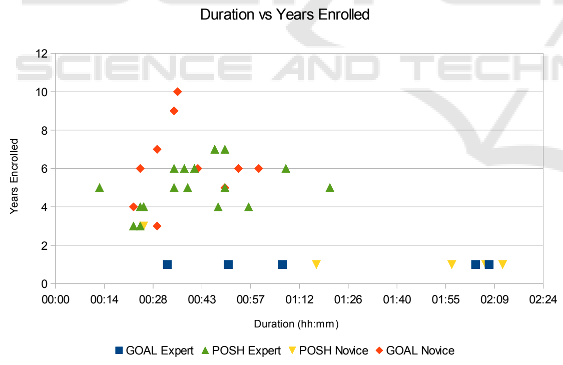

Figure 4: Cumulative time-to-complete vs years enrolled.

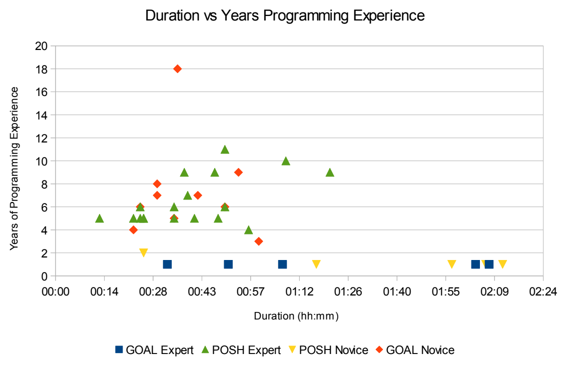

Figure 5: Cumulative time-to-complete vs experience.

6.3 POSH Language Feature Usage

Tables 2 and 3 show the average ratings given by subjects of how useful they found a POSH feature for the comprehension and coding tasks, respectively; see Section 4 for discussion of these features. Most subjects used almost all POSH language features while completing a task. Both novices and advanced programmers rated features of the language close to a rating of 4.0 indicating that all features were useful with only one exception: Advanced programmers did not find the textual representation of POSH plans useful but novices indicated that they did find it useful. Novices also appear to rely more on Java when they are asked to program more complex agent behaviour.

6.4 Language Features GOAL

Tables 4 and 5 show the ratings given for the usefulness of GOAL features. Although all main GOAL language features were used for completing the comprehension task, this is not the case for the coding tasks. Generally speaking, advanced programmers use more GOAL language features except maybe for Prolog (the knowledge representation language on top of which GOAL runs). It is not entirely clear how to explain these findings but they suggest a difference in learning curve between POSH and GOAL. Perhaps novices that used GOAL, which all had initial experience with Prolog, may more quickly resort to what they already know than more advanced programmers (a similar effect was observed for POSH). However, due to the low number of subjects as well as the presence of outliers, it is difficult to draw any firm conclusions. We

Table 2: POSH Feature Rating: Comprehension Task.

	Novice		Advanced	
	Usage	Rating	Usage	Rating
Senses	100%	4.2 (0.4)	94%	4.2 (0.7)
Actions	100%	4.0 (0.6)	100%	4.2 (0.8)
Plans	100%	4.8 (0.4)	100%	4.6 (0.8)
Textual	100%	3.8 (1.0)	94%	1.8 (0.8)
Java	100%	3.0 (1.2)	83%	2.4 (1.2)
Overall		4.0 (0.0)		4.0 (0.7)

Table 3: POSH Feature Rating: Coding Tasks.

	Novice		Advanced	
	Usage	Rating	Usage	Rating
Senses	100%	3.8 (0.4)	100%	3.9 (0.9)
Actions	100%	3.6 (0.5)	100%	3.9 (0.9)
Plans	100%	3.8 (0.4)	100%	3.8 (1.1)
Textual	NA	NA	NA	NA
Java	100%	4.4 (0.5)	100%	3.9 (1.3)
Overall		3.8 (1.1)		3.7 (0.8)

Table 4: GOAL Feature Rating: Comprehension Task.

	Novice		Advanced	
	Usage	Rating	Usage	Rating
Actions	90%	2.4 (1.2)	80%	3.0 (1.4)
Modules	90%	4.9 (0.3)	100%	4.4 (0.9)
Events	70%	2.3 (1.1)	100%	2.8 (1.1)
Knowledge	100%	2.8 (1.1)	100%	3.4 (1.5)
Prolog	100%	4.3 (0.7)	100%	3.8 (1.3)
Overall		4.6 (0.5)		5.0 (0.0)

Table 5: GOAL Feature Rating: Coding Tasks.

	Novice		Advanced	
	Usage	Rating	Usage	Rating
Actions	22%	4.5 (0.7)	80%	3.3 (0.6)
Modules	89%	4.6 (0.5)	100%	4.4 (0.9)
Events	22%	4.0 (0.0)	40%	4.0 (1.4)
Knowledge	67%	4.2 (1.2)	20%	5.0 (N/A)
Prolog	100%	4.6 (0.7)	80%	4.0 (1.4)
Overall		4.4 (0.7)		3.2 (1.3)

are also not completely sure that all language features were required for completing the tasks.

7 DISCUSSION

The most surprising finding has been that advanced POSH programmers as well as GOAL novices (both subjects from Prague University) completed tasks much faster than advanced GOAL programmers or POSH novices (both subjects from Delft University). As a consequence, contrary to our initial expectations, all three hypotheses are rejected. In our study, GOAL novices performed better than POSH novices (and not vice versa), advanced POSH programmers performed better than advanced GOAL programmers (and not similarly), and advanced GOAL programmers did not clearly outperform GOAL novices.

Nevertheless we can draw some interesting conclusions based on our findings. First, the fact that Java is used in combination with POSH did not yield a significant difference as we expected. It is important to note that subjects that used POSH did not exclusively use Java for programming the agent but also used POSH rules for implementing the agent's decision making. Second, an alternative interesting explanation of the performance of subjects from Prague University is suggested by the fact that these subjects were all more experienced programmers. Our findings here suggest that, generally speaking, having *more programming experience improves performance more than additional training in any particular programming language*. One might speculate whether more experienced programmers have developed a mental model that is more generally applicable to arbitrary programming language. However, a simpler explanation may be that they are more experienced in using tools for developing programs. In particular, we observed that the more advanced programmers made more use of various IDE features. Better handling of the tooling provided with a language may be quite beneficial for programming in general, which would point towards the *importance of tool support*. As development of most agent programming languages has focused more on the language features than on the tooling to make these languages accessible to programmers, a shift in focus may thus be called for, although more work is needed to confirm this.

Given these results and the fact that differences between GOAL novices and advanced POSH programmers and between advanced GOAL programmers and POSH novices are small, we can conclude that it remains difficult to establish relations between language features and programming performance. In this as well as previous studies that compared POSH with Java (Gemrot et al., 2012a; Gemrot et al., 2012b) no significant effects of the programming language on the programmer's performance was found.

It thus remains difficult to design experimental tasks for comparative studies of (agent) programming languages. One issue associated with the experimental design that we used is the necessarily short amount of time that is available for completing tasks. Advantages of programming languages that might not show up in a short study may still manifest themselves in much larger programming projects which take weeks or even months. Other issues may be more specific to the particulars of agent programming languages. It may, for example, be the case that successfully completing tasks that involve virtual environments such as Emohawk require that more time is spend on observing and testing the agent's behaviour in that environment (if only because virtual characters take more time to perform actions). A significant amount of time thus may be used to this end compared to the time that is actually spend on programming. As a consequence, the use of a virtual environment may make it more difficult to measure the effect of a programming language on completing a task. Based on observations during the experiment we do not believe there has been a significant difference in amount of testing for either of the platforms used. For future work, we therefore suggest to also measure the actual time spent on various subtasks such as testing.

Finally, some language specific conclusions may still be drawn based on the language feature usage that we observed. An interesting finding has been that subjects that programmed in POSH use the plan, senses and actions language features for completing the coding tasks. Subjects that programmed in GOAL, however, can make due with just the modules (consisting of rules) and the knowledge section. These language features offer more or less the same expressivity and therefore allow programmers to provide the same functionality by only using these features. This suggests that for this particular study an agent programming language that provides an efficient way to encode and execute plans that consist of condition-action rules may be sufficient. Moreover, a likely reason that explains why the event module of GOAL was not used is that completing the tasks only required writing of a relatively simple reactive agent that can derive its behavior directly from the information it perceives. It may therefore be interesting in future work to design tasks that require more complicated decision logic and/or persistent memory in order to determine when specific language features are actually used. Such tasks might reveal differences between e.g., POSH and GOAL, as tasks in which an agent is required to infer persistent beliefs from percepts might be harder to write in POSH than in GOAL.

8 CONCLUSION

In this paper, we reported the results of a comparative study of the agent programming languages GOAL and POSH. We discussed the design of an experiment that required subjects to complete both comprehension as well as coding tasks for an agent that controls a character in the virtual environment Emohawk. The tasks were designed such that they could be completed using either GOAL or POSH. As in other studies we did not yet find conclusive results that suggest the agent programming language used has a significant influence on programmer performance. We did find that general programming experience seems to be a relatively big advantage when using a (new) programming language. This may in part be due to the experience of using tooling support to develop code, which highlights once more the importance of this aspect when developing agent programming languages.

In addition, we believe that our study provides several useful clues for the design of future comparative studies that may actually yield concrete results about the effectiveness of language features incorporated in a language. In our study the tasks required programming a purely reactive agent, whereas programming a more goal-oriented agent that needs to use more complicated decision logic and/or persistent memory may highlight differences between e.g., languages such as GOAL and POSH more. As GOAL and POSH are clearly different programming languages that are used in different ways, more work on the design of comparative studies is needed to provide insights on the usability of features of these languages.

REFERENCES

Behrens, T. M., Hindriks, K. V., and Dix, J. (2011). Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295.

Bída, M. and Brom, C. (2010). Emohawk: learning virtual characters by doing. *Interactive Storytelling*, pages 271–274.

Bordini, R., Dastani, M., Dix, J., and El Fallah Seghrouchni, A. (2009). *Multi-agent programming: Languages, tools and applications, Vol. 15.* Berlin: Springer.

Bordini, R., Dastani, M., Dix, J., and Seghrouchni, A. E. F., editors (2005). *Multi-Agent Programming: Languages, Platforms and Applications.* Springer.

Brom, C. (2009). Curricula of the course on modelling behaviour of human and animal-like agents. In *Proceedings of the Frontiers in Science Education Research Conference*, volume 22, page 24.

Brom, C., Gemrot, J., Bıda, M., Burkert, O., Partington, S., and Bryson, J. (2006). Posh tools for game agent development by students and non-programmers. In *9th international conference on computer games: Ai, animation, mobile, educational & serious games*, pages 126–135.

Bryson, J. (2001). *Intelligence by design: principles of modularity and coordination for engineering complex* *adaptive agents*. PhD thesis, Massachusetts Institute of Technology.

Bryson, J. (2003). Action selection and individuation in agent based modelling. In *Proceedings of agent*, pages 317–330.

Gemrot, J., Brom, C., Bryson, J., and Bída, M. (2012a). How to compare usability of techniques for the specification of virtual agents behavior? an experimental pilot study with human subjects. *Agents for Educational Games and Simulations*, pages 38–62.

Gemrot, J., Hlávka, Z., and Brom, C. (2012b). Does high-level behavior specification tool make production of virtual agent behaviors better. In *Proceedings of the International Workshop on Cognitive Agents for Virtual Environments (CAVE'12)*.

Gemrot, J., Kadlec, R., Bda, M., Burkert, O., Pbil, R., Havlek, J., Zemk, L., imlovi, J., Vansa, R., tolba, M., Plch, T., and Brom, C. (2009). Pogamut 3 can assist developers in building ai (not only) for their videogame agents. In Dignum, F., Bradshaw, J., Silverman, B., and Doesburg, W., editors, *Agents for Games and Simulations*, volume 5920 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg.

GOAL (2015). Open source website for GOAL on github. https://github.com/goalhub.

Heckel, F., Youngblood, G., and Hale, D. (2009). Behaviorshop: An intuitive interface for interactive character design. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI Press*.

Hindriks, K. (2009). Programming Rational Agents in GOAL. In *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer US.

Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1998). A Formal Embedding of Agentspeak(L) in 3APL. In Antoniou, G. and Slaney, J., editors, *Advanced Topics in Artificial Intelligence*, volume 1502 of *Lecture Notes in Computer Science*, pages 155–166. Springer Berlin Heidelberg.

Lillis, D., Jordan, H. R., and Collier, R. W. (2012). Evaluation of a Conversation Management Toolkit for Multi Agent Programming. In *Proceedings of the 10th International Workshop on Programming Multi-Agent Systems (ProMAS 2012)*, Valencia, Spain.

Materials (2015). Experiment materials https://ii.tudelft.nl/trac/goal/wiki/Projects/PoshVsGoal.

Pane, J. F. and Myers, B. A. (1996). Usability issues in the design of novice programming systems. Technical Report CMU-CS-96-132, School of Computer Science, Carnegie Mellon University.

Shapiro, L. and Sterling, E. (1994). *The Art of Prolog: Advanced Programming Techniques*. MIT Press.

van Riemsdijk, M. B., Hindriks, K. V., and Jonker, C. M. (2012). An empirical study of cognitive agent programs. *Multiagent and Grid Systems (MAGS)*, 8(2):187–222.