Knowledge-Based Systems 23 (2010) 295–301

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Explaining how to play real-time strategy games

Ronald Metoyer [a], Simone Stumpf [a,*], Christoph Neumann [b], Jonathan Dodge [a], Jill Cao [a], Aaron Schnabel [c]

[a] Oregon State University, Corvallis, OR 97331, USA
[b] Hewlett Packard, Corvallis, OR 97330, USA
[c] 9Wood, Inc., Springfield, OR 97477, USA

ARTICLE INFO

Article history:
Available online 22 November 2009

Keywords:
Strategy
Explanation
Real-time games
User study

ABSTRACT

Real-time strategy games share many aspects with real situations in domains such as battle planning, air traffic control, and emergency response team management which makes them appealing test-beds for Artificial Intelligence (AI) and machine learning. End-user annotations could help to provide supplemental information for learning algorithms, especially when training data is sparse. This paper presents a formative study to uncover how experienced users explain game play in real-time strategy games. We report the results of our analysis of explanations and discuss their characteristics that could support the design of systems for use by experienced real-time strategy game users in specifying or annotating strategy-oriented behavior.

1. Introduction

Artificial Intelligence (AI) research has shifted focus in recent years from board games such as Chess or Go to real-time strategy (RTS) games, such as that shown in Fig. 1, as test-beds for learning complex behavior. RTS games are typically carried out in a two-dimensional world in which multiple players concurrently compete for resources, build armies, and guide them into battle. Winning the game necessitates executing a strategy by placing game-playing units in a spatial environment and giving them tasks to do at the right time. RTS games are particularly appealing to AI because of the many levels of complexity involved in the game play, such as resource management, decision-making under uncertainty, spatial and temporal reasoning, adversarial reasoning, etc. [2]. Such challenges are also present in many other domains that require strategy execution, including air traffic control, emergency response management, and battle planning.

The typical approach has been to learn behavior from many instances of game play log data. However, this approach cannot be applied if there is sparse training data or if, in the extreme case, there is just a single game trace. This challenge could be overcome by allowing end users, not adept in machine learning, to inform learning algorithms about salient features and tasks by supplementing the game trace with explanatory annotations or demonstration. Facilitating additional user feedback to learning algorithms has been shown to produce improvement to learning in other domains [21].

Expert explanations could be used in a natural programming approach [14] as building blocks for the design of annotation or demonstration systems. While researchers have investigated expert game play in traditional games as well as more recent action games [17], to our knowledge, there has only been limited research that has investigated the explanations of experienced players for real-time strategy games.

By studying explanations of game play, we aim to uncover a user vocabulary that includes objects, spatial aspects, and temporal constraints. We also aim to help the design of annotation and demonstration tools for machine learning systems that operate on minimal user examples by trying to understand how behavior is enacted in game play. The contributions of our research are (1) coding schemes useful for transcripts of real-time strategy game explanations (2) identification of the content of game play explanations (3) identification of the structure of game play explanations and (4) a set of design implications for real-time strategy game annotations.

In this paper, we describe a formative user study designed to understand how experienced users explain strategies and game play. We begin by discussing the related literature and then describe our experimental design to capture explanations, including our methodology for coding the data. We present the results of our analysis and discuss the trends and characteristics of the explanations and how this information may be used to inform the design of end-user programming environments for agent behavior as well as for annotating strategy for machine learning systems.

* Corresponding author.
E-mail address: stumpf@eecs.oregonstate.edu (S. Stumpf).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2009.11.006

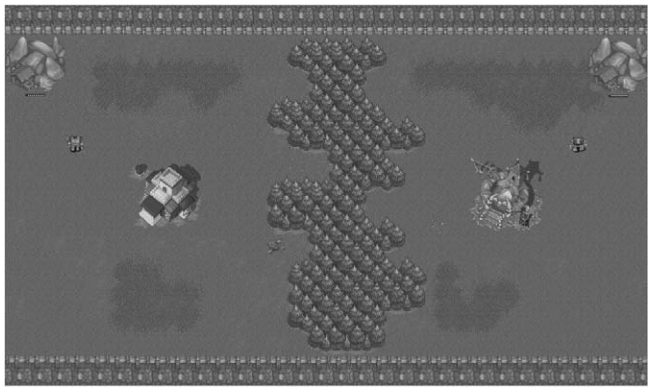

Fig. 1. Our customized version of the "Nowhere to Run, Nowhere to Hide" map for the real-time strategy game, Wargus.

2. Related work

While the machine learning and AI communities have focused on real-time strategy games as a domain of interest for several years, to our knowledge, none of the research has attempted to understand how *people* describe their strategies. Instead, much of the literature in these areas is concerned with identifying a language for representing problems and solutions in complex learning and planning domains, such as Wargus [22]. Ponsen et al. for example, incorporate knowledge into their AI algorithms by hand coding domain knowledge for planning within the Wargus domain [15,1]. Rather than finding a representation for *machines* to use for learning, we are interested in finding a language or representation for *people* to use for demonstrating or annotating behavior for a machine.

Notations for specifying behavior can be found in the end-user agent programming domain which has applications in many fields including robotics, video games, and education. Agent programming approaches generally fall under either *direct programming approaches* or *programming by demonstration*. In the direct programming case, some research has addressed the challenge of programming agent behavior by developing specialized APIs or code construction environments to support novice users of a general purpose programming language. For example, the RoboCode project [11] allows a student to use and extend a Java API to define the behavior of a virtual tank within a 2D simulated environment. Alice [4] employs techniques such as drag-and-drop construction and live method execution to assist the user in programming agents with an object-oriented textual notation. In a similar fashion, Agentsheets [18] supports end-user programming of agents by using an object-oriented notation that is augmented by fill-in forms and live execution within an environment that emphasizes the use of a 2D grid as a means to organize the simulation space [6]. Whereas these approaches focus on reducing fundamental challenges associated with general purpose programming languages, the focus of our experiment is to inform a notation which is grounded in the language of end users of a RTS game.

Programming by demonstration (PBD) systems have been shown to lower barriers to programming agents by allowing the user to simply demonstrate the proper behavior. Examples of such an approach are KidSim [20] and ToonTalk [7]. As noted by Modugno et al. [12] and Repenning [19], PBD still requires a notation to allow for editing and high-level specification and the form of that notation can affect the effectiveness of the PBD system.

3. Experiment design

In order to explore how behavior is explained by users in real-time strategy games our formative study followed a dialogue-based think-aloud design, in which we paired an experienced user with a novice to help elicit explanations of game play. The experienced user was asked to play the game while at the same time explaining what he or she was doing. The novice was able to observe the experienced user and ask for clarifications when necessary.

The dialogue-based think-aloud setup allows reasoning to be made explicit as explanations are given in a natural, interactive way through typical social communication with their partners. As an experienced user showed the novice how to play the game, he was able to draw attention to what mattered and to justify his actions in the context of situations as they arose. In response, novices were able to ask questions that clarified the experienced users' actions or explanations, drawing out additional details that may have otherwise been unclear.

We chose Wargus as the RTS environment for our study. Wargus [22], and its commercial equivalent *Warcraft II*, allows users to command orcs or humans in a medieval world that pits the player in battle against opposing factions. We used a contained battle scenario shown in Fig. 1 where the computer controlled the opponent. In this simple map, the player starts off on one side of a barrier of trees that separates him or her from the enemy. There are a variety of ways that enable a player to play this scenario and overcome the enemy but the game is simple enough to be completed within a reasonable time.

Ten students participated in this study and were compensated for their time. Participants were assigned roles as experienced users and novices based on their experience with *Wargus* or *Warcraft II*. Participants with more than 20 hours of experience were considered experienced users, while novices had less than 2 hours of experience. Experienced users and novices were then randomly paired. Overall, the participants were made up of nine males and one female with an average age of 22.8 years. Experienced users consisted of five males, average age of 20.2 years, and novices were four males and one female, average age of 25.4 years.

The study session began after a brief paper tutorial, in which the participants were familiarized with units, resources, and basic game-playing instructions. The experienced users were asked to play the game while "thinking aloud" about what they were doing. The novices were instructed to ask the experienced users about any detail they did not understand. Each experiment session lasted approximately 35 min, during which two games were played by the experienced user. We used the Morae [13] system to record the screen as the game was played, and to record the interaction between experienced users and novices. All screen, video and audio data was automatically synchronized. After the session, we captured background and demographic information in the post-session questionnaire in addition to subjective evaluations and comments about game play during the study.

4. Methodology

In order to analyze the think-aloud data, we used content analysis to develop coding schemes for describing and understanding how game play is communicated [9]. We developed two sets of codes: the first set captures the *content* of explanations (Table 1) while the second captures the *structure* of explanations (Table 2). We now describe our code development process in more detail.

In order to facilitate analysis, the audio of the experienced user and novice interactions was transcribed, and supplemented with time codes and information about game actions and gestures. Transcripts then were broken up into coding units. In our approach, each sentence a participant uttered was segmented into one or more units. Sentences were broken up at coordinating conjunction (e.g. 'and', 'or') or subordinating conjunction (e.g.

Table 1
Content coding scheme.

Code	Subcode	Description	Example
Object	Enemy object	Important objects that are under the control of the opposing player	"*They* have no more *peons*"
	Fighting object	Units that are used for fighting	"My *archers*"
	Production object	Units that are involved in producing resources/are resources	"I am building a *town hall*"
	Environmental object	Object that is part of the game environment, not under the direct control of the game players	"I wanna not cut down those *trees*
	Unspecified object	Player refers to an object indiscriminately	"My *guys* here"
Action	Building/ producing	When the action described in the statement refers to building or producing things	"I am going to build a farm"
	Fighting	When the action described in the sentence refers to fighting	"And you only attack one guy at a time"
Quantity	Unidentified discrete	A reference to object quantity but vague amount	"*Armies* of peasants are good"
	Identified Discrete	Reference to object quantity with a specific amount stated	"I want *a barracks*"
	Comparative	Reference to object quantity in comparison to an (sometimes unspecified) reference point	"We need *more* farms"
	Absolute	Reference to quantity extremes	"I went in and killed *all* their grunts"
Temporal	Ordering	Referring to the sequence in which things have to happen	"They will tend to attack military units *before* peons"
	Timing	Referring to an absolute time	"Are those trees *now* wide enough to go through?"
	Speed	Referring to the speed at which things have to happen	"Be *really fast* in the early game"
	Repetition	How often things have to happen	"Do that *again*"
Spatial	Distance	A relative distance between two objects (e.g. close to, away from)	"I am trying to keep my archers *away* from fighting"
	Point	A specific place	"Is that a hole right *there*"
	Size	Absolute reference to an object's length or space	"Let's have them chop where the gap is *kind of big*"
	Arrangement	Specific spatial arrangement of objects	"And if you can get some archers *along the border* killing their peons"

Table 2
Structure coding scheme.

Code	Description	Example
Fact	A statement or opinion about how the world works, a current event, or a future outcome	"Farms supply food"
Depend	Language that reflects a dependency of one thing on another or a constraining fact. A statement that reflects a forced or enabled course of action due to a limiting or satisfied constraint	"Building archers requires wood"
Do	Prescriptive instructions on how to behave, in particular, talk about manipulating concrete things and taking concrete actions	"Build a farm"
Goal	A statement of intent or desired achievement that is non-specific about means or actions	"Block them from reaching your ranged units"
History	A statement that describes an action or event that has already occurred in the past	"They had ranged units and I did not"
Mistake	A statement that negatively describes an action or event that has occurred in the past	"It would have been good if I had gotten archers early on"
Question	A statement where further clarification is requested	"Are those trees now wide enough to go through? "
UI	A statement that refers to software-specific features	"Control-1 just makes them group 1"

'because', 'since'). Sentences were left intact if they contained correlative conjunctions (e.g. 'either. . .or', 'if. . .then').

Previous research has not provided any coding schemes applicable to our investigation. In order to develop coding schemes suitable to our aims, we employed an affinity diagramming approach to develop codes by examining random transcript sub-portions in a team setting [5]. After initial identification of candidate codes, we refined them iteratively and tested the reliability and coverage of coding application. In the refinement process, a candidate coding scheme included definitions of potential codes and corresponding examples from the transcripts. The candidate codes were applied independently by researchers to a randomly chosen transcript section and agreement measures were calculated. Any codes that proved difficult to apply were further refined and integrated into a revised candidate coding, which was in turn applied to a new random transcription section. Once sufficient agreement between the coders was reached, the remaining transcripts were coded by individual researchers.

Agreement measures are useful in developing codes that provide coverage of the area under investigation, and that can be consistently and reliably applied by different researchers [3,16]. For the first coding scheme, the content codes, multiple codes could be applied to the same unit, making standard Kappa unsuitable as an agreement measure. We therefore calculated agreement between researchers using the Jaccard index, which is the intersection of two researchers' codes divided by the size of their union. We reached an overall code agreement of 80.12% for the content coding scheme.

For the second coding scheme, the structure codes, we used a slightly modified process to account for three raters using mutually exclusive codes. We calculated agreement in two different ways. We first calculated a simple agreement measure by using the proportion of actual agreements over possible agreements, applied pairwise between all three researchers; the average agreement over three researchers for the structure code set was 83.44%. We also calculated Fleiss' Kappa for this code set, which was 0.69 (agreement over 0.61 is usually considered substantial [10]).

5. Results and discussion

One of our contributions is the development of two coding schemes that allow the structure and content of explanations to be explored within the realm of real-time strategy games (see Tables 1 and 2). These coding schemes could also be re-used or adapted for other domains that feature dynamically changing environments with spatial and temporal constraints.

5.1. What concepts are used in explanations

Understanding the content of user explanations can help in identifying concepts that an annotation language or demonstration system should cover. We analyzed the content codes (Fig. 2) to understand what aspects were frequently mentioned in RTS explanations.

References to *Objects* (sometimes called entities) of the game environment occur most frequently (72.1%). Not surprisingly, participants talked most frequently about their own units (e.g. such as *Production* and *Fighting* units) but objects relating to the *Enemy* are referenced very often (15%). This indicates that an important aspect of game play is monitoring the activity of one's opponent.

Spatial and temporal aspects of game play are important areas for learning. One challenge may be that explanations could be too vague or too infrequent to be able to generate good examples from which to learn. Surprisingly, we found that experienced players expressed *Spatial*, *Temporal*, and *Quantity* concepts frequently throughout the game, in 11.4%, 19.5%, and 28.9% of the coded units, respectively. Participants were also very specific about these concepts. *Spatial* concepts occurred mostly in terms of *Point* specific locations (7.2%) such as "here" or "at the farm", *Temporal* concepts occurred most often as *Timing* statements (9.8%), such as "now" or "at the end" while participants often described a specific *Identified Discrete* quantity (12.1%) or *Absolute* value (6.9%), such as "two units at a time" or "all". Even when they were not able to give a discrete quantity, they were able to give *Unidentified* values (5.5%), such as "little bit", or *Comparative* amounts (6.0%), such as "more footmen than archers". This indicates that experienced users tended to be very concrete in their explanations *while* playing the game. They were able to refer to particular numbers of objects, at particular locations, and indicated particular times at which events occurred.

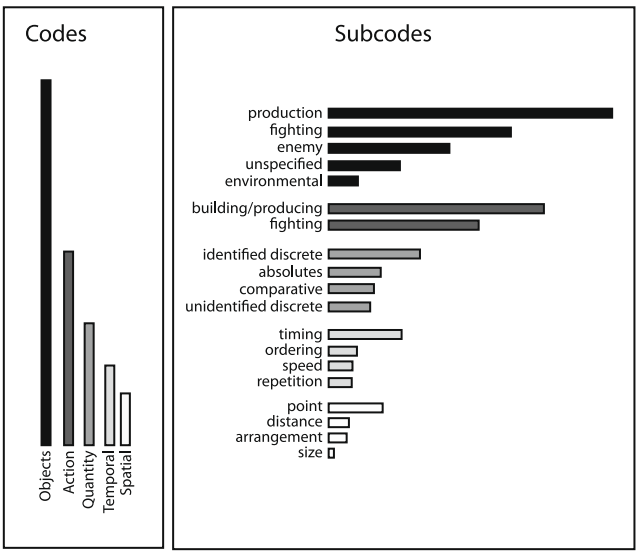

Fig. 2. Frequency of content code occurrences over all transcripts.

Some concepts that were expressed are more abstract or complex, and may require specialized support. Some explanations referred to the spatial *arrangement* or *distance* of objects to each other in the game, while temporal constraints such as *ordering*, *speed*, or *repetition* were also mentioned.

Design implications: Users pay attention to aspects under their control as well as to aspects that are outside their realm of manipulation. Annotations need to account for monitoring of these outside factors, which may lead, in turn, to changes in future choices of actions.

Any annotation or demonstration interface should account for and provide a means for specifying or choosing these specific concepts possibly through mouse pointing (point locations), time indicators for both discrete and comparative (now, early, as soon as, etc.), and a broad range of quantity selection mechanisms such as number entry for object quantities, object group selection, and selection/deselection of all objects. In addition, annotation and demonstration tools need to lend support to the user to easily specify more complex concepts that puts various objects in relation to each other.

5.2. Explaining how to win

Choosing the right strategy and executing it correctly helps the user win the game. We investigated the structure of how experienced users explained the strategy and necessary actions. Fig. 3 shows the set of structure codes and their distribution over all game transcripts. Participants mentioned *Goals* less frequently than expected (7.9%). While some participants used *Goal* codes more than others, it was surprising to us that experienced users did not provide high-level explanations of their strategy more frequently, especially considering that experienced users summarized their strategies succinctly with general, high-level descriptions in the post-study questionnaire.

It appears that *Goal* as a high-level intent was only one way in which a strategy could be described by experienced users. In a *Do* code, an experienced player gave instructions on how to behave, focusing on specific actions in the pursuit of an intended strategy. In our study, experienced users employed *Do* more frequently than *Goals* (12.1%). Experienced users on the whole tended to explain their strategy during the interaction by using a finer granularity, in which they made detailed reference to what to do in the context of the game.

Do and *Goal* should be considered as a spectrum in which to explain strategy. While most experienced users preferred to explain strategy in terms of prescriptive instructions in pursuit of a higher-level goal which is not necessarily verbalized, others tended to employ high-level descriptions of general intent instead. When these two codes (*Do* and *Goal*) are considered in combination, they made up a considerable amount of explaining of what to do to win the game (20%).

Understanding when strategy explanations occur is important in deciding when to make annotation or demonstration capabilities available. A reasonable but naive assumption would be that strategy is stated at the beginning and then enacted or decomposed into smaller tasks in the remainder of a game. In our study, strategy explanations in the form of *Do* or *Goal* were found interspersed throughout both games – even for the second game in a study session, in which participants could have omitted strategy explanations since they had already been covered previously.

Design implications: Our results show that experienced users provided many explanations of their intended behavior, but that they had a preference for choosing a certain level of granularity in which to express the strategy. Experienced users that chose high-level strategy explanations tended to provide fewer detailed, fine-grained strategies, and vice versa. The variance of users'

preference for detail is an important factor to consider for notations in order to provide a match to the granularity of expression. Furthermore, notations for expressing strategy that are only available at the beginning of the game, and force decomposition in a top-down fashion, may run counter to how users prefer to explain strategy. In our study, strategy explanations were made in a situated way throughout, drawing on the surrounding context. Our findings imply that behavior annotation and/or demonstration could possibly benefit from environments that are tightly coupled to game play and that allow annotation and demonstration *within* the game context. In addition, annotation strategy behavior *within* the context of the environment should provide a means for detailed prescriptive instructions and the intent behind them while annotations outside of the environment may still benefit from a higher-level, general mechanism for specifying the strategy.

5.3. Explaining what to notice

Actions in RTS games depend on the context in which they are enacted. What to do may draw on certain features of the situation, require constant monitoring, and may have to be adjusted based on unexpected outcomes. We were interested in how the context of game play is communicated in RTS games.

One problematic aspect of game play and the actions that a player could carry out is that there are potentially a myriad of features of the situation which could matter. How does an experienced player communicate which of these features to attend to? One such way is by statements that express *Facts*, which draw attention to certain features in the game that are important. In addition, *Depend* statements draw out constraints that need to be met in these particular features and situations.

Experienced players focused on highlighting the important features and constraints frequently. In our experiments, *Fact* and *Depend* structure codes combined occurred in 45% of the transcript (34.6% and 11.7%, respectively) (See Fig. 3). We also found that *Fact* and *Depend* occurred constantly as the games proceeded.

Design implications: An interface for annotating or demonstrating strategy behavior should provide a simple and efficient means for describing the important current features in a situation. This allows a user to efficiently select important features that the behavior depends on. For demonstration or annotation for machine learning, for example, the context describes the important features that the system needs to take into consideration and feature selection is often a difficult problem in machine learning.

5.4. How concepts are used in strategy and context

In order to complete the picture of game-playing instructions, it useful to consider what is explained and how it is explained at the same time. To do so, we computed the co-occurrence of structure codes with content codes.

Fig. 3. Frequency of structure code occurrences over all transcripts. Fact occurs in 35% of the total transcript segments.

To calculate co-occurrence, we counted, over all transcripts, the number of times a content code appeared in the same unit with a particular structure code. We computed the percentage of co-occurrence for each content/structure code pair by dividing the number of co-occurrences by the sum of the total number of times each of the two codes appeared in all transcripts. Fig. 4 shows an example of the co-occurrence computation for *Enemy* and *Fact* over all transcripts.

The pattern of co-occurrence is complex but there are some patterns that occur across the codes (Fig. 5). When giving explicit instructions (*Do* codes), participants talked mainly about *Production objects* and *Fighting Objects* and the act of *Producing/building* (12.6%, 9.3%, and 15.3%, respectively). Additionally, they tended to reference both *Unidentified discrete* and *Identified discrete* quantities and *Point* specific locations (8.5%, 10.5% and 9.0% respectively). This means that participants frequently gave specific instructions about 'where' to place 'how many' buildings and/or units for resource accumulation or battle. In contrast, *Goal* codes most frequently appeared with *Fighting* (9.2%), *building/producing* (6.5%), and the associated *Enemy* (6.7%) and *Fighting Objects* (6.3%). Additionally, they often mentioned *Goal* in concert with *Timing* (7.7%), *Arrangement* (6.7%), *Distance* (6.1%) codes. It appears that participants' specification of a higher-level strategy tended to be more concerned with laying out complex spatial concepts, coupled with specific temporal aspects.

Some patterns can also be discerned in explanations of what to notice. *Facts* frequently involve all *Objects* but in particular *Enemy objects* (13.0%) and *Fighting objects* (13.1%), whereas *Depend* codes most frequently co-occurred with *Production objects* (14.8%) and *Building/Producing* (15.3%). Both *Fact* and *Depend* also co-occur often with *Timing* (6.6% and 6.4%, respectively) while *Depend* occurs more frequently with all kinds of *Quantity* references than does

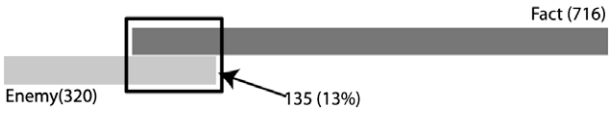

Fig. 4. Diagram demonstrating the co-occurrence calculation for *Enemy* and *Fact*.

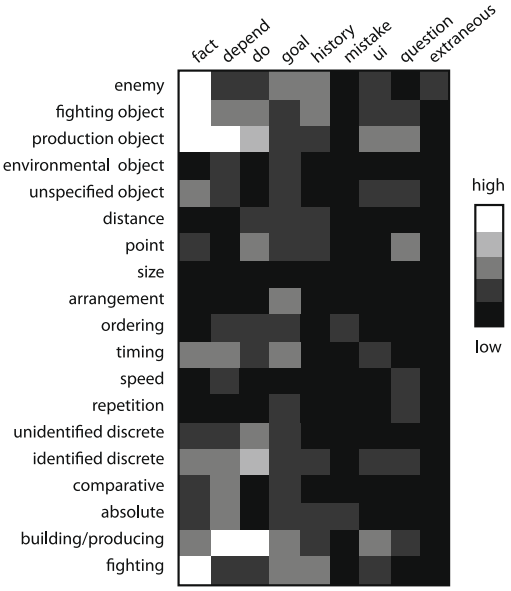

Fig. 5. The co-occurrences, over all transcripts, between structure codes and content subcodes.

Table 3
The frequency of structure codes in relation to *Question* codes (in percentages).

	Fact	Depend	Do	Goal	History	Mistake	Question	UI
Code preceded *Question*	11.2	10.3	20.3	8.3	8.0	3.3	8.4	13.0
Code followed *Question*	44.9	13.1	6.5	1.9	4.2	0.5	8.4	14.0

Fact. It seems that constraints on actions usually involve resource management but that constraints are not considered as much during monitoring opponents and battle planning. Additionally, constraints apparently are described in terms of the quantities necessary to achieve the strategy.

Design implications: In certain situations some aspects of the game play are more salient than others. In explaining strategy, specific instructions about what to do with objects may be easily given but more complex spatial concepts may need to be captured through annotations involving higher-level strategy. Similarly, constraints could be expressed easily for resources under one's own control but possibly are hard for complex battle situations involving an opponent's resources.

5.5. When more explanation is needed

Questions usually provide *explicit* requests for more information and are indications of information gaps [8]. Thus, we paid particular attention to questions that novices asked experienced users, since they indicate a breakdown in the novice's understanding.

Questions occurred frequently (9.2%) throughout the games, indicating that experienced users did not explain in sufficient detail at all times. Table 3 shows the percentage of times that a particular code preceded a *Question*. *Questions* occurred after every code, indicating that anything was liable to cause a breakdown. However, *Questions* after *Do* codes were especially frequent.

Table 3 also shows the code that immediately followed a question. This gives an indication of the type of answers that follow requests for more information. *Goal* and *Do* were not present in substantial numbers after *Questions*. It appears that experienced users did not provide answers in terms of strategy. In contrast, *Fact* (44.9%) and *Depend* (13.1%) codes most frequently followed questions. It appears that answers focused on explanations of what things were important (situation context) for the novice to consider when applying the strategy.

Design implications: The high incidence of breakdowns following actions (*Do*) indicates that notations may be useful to provide further clarification for these situations. Novel approaches in programming by demonstration, annotation, or machine learning could also generate questions that might help identify relevant information. Answers to these questions may be more likely to highlight which features to pay particular attention to.

5.6. Revisiting the past

Some explanations do not occur concurrently with the execution. Experienced players sometimes referred to mistakes as well as present or past courses of action. Mistakes were pointed out rarely and randomly (1.9%). More frequent were references to what had gone on in the past, in the form of *History* codes (7.6%). The majority of these statements occurred at or towards the end of transcripts.

Design implications: Experienced users' mistakes and reflection on the past implies that a programming or annotating environment needs to give users the opportunity to connect observed behavior to causes of that behavior. This is in line with findings of Reeves et al. [17], who found that experts become better by reflection on their own play. It is therefore natural to assume that experienced users could explain their failures and successes by reflecting on their actions. Annotation tools should allow the user to pinpoint when the strategy started to go wrong or locate the turning point for success. In addition, an annotation or demonstration interface would possibly benefit from a means for 'recalling the context' for the user to properly annotate history.

6. Conclusion

We have presented a study aimed at understanding how and what experienced users explain in RTS games. Our first contribution is the development of two coding schemes that allow a structural, content, and combined investigation. Our second contribution is an analysis of the study data and the practical implications of our findings when designing an annotation or demonstration environment for specifying and coordinating complex behavior.

Gaining a rich understanding of how RTS game play is explained by users can lead to better annotation and demonstration tools for machine learning systems, and may also provide a first step annotation in other dynamic environments in which users must make real-time decisions within specific spatial and temporal constraints.

Acknowledgements

The authors would like to thank the study participants and gratefully acknowledge support of the Defense Advanced Research Projects Agency under DARPA Grant No. FA8650-06-C-7605. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA.

References

[1] D. Aha, M. Molineaux, M. Ponsen, Learning to win: case-based plan selection in a real-time strategy game, in: Proceedings of 6th International Conference on Case-Based Reasoning (ICCBR-05), 2005, pp. 5–20.
[2] M. Buro, Real-time strategy games: a new AI research challenge, in: Proceedings of IJCAI, 2003, pp. 1534–1535.
[3] J. Carletta, Assessing agreement on classification tasks: the kappa statistic, Comput. Linguist. 22 (2) (1996) 249–254.
[4] S. Cooper, W. Dann, R. Pausch, Alice: a 3-D tool for introductory programming concepts, in: Proceedings of the Fifth Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges, Consortium for Computing Sciences in Colleges, 2000, pp. 107–116.
[5] K. Holtzblatt, H. Beyer, Making customer-centered design work for teams, Commun. ACM 36 (10) (1993) 92–103.
[6] K. Howland, J. Good, J. Robertson, Script cards: a visual programming language for games authoring by young people, in: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2006, pp. 181–186.
[7] K. Kahn, ToonTalk-an animated programming environment for children, J. Visual Lang. Comput. 7 (2) (1996) 197–217.
[8] C. Kissinger, M. Burnett, S. Stumpf, N. Subrahmaniyan, L. Beckwith, S. Yang, M.B. Rosson, Supporting end-user debugging: what do users want to know?, in: AVI 06: Proceedings of the Working Conference on Advanced Visual Interfaces, 2006, pp. 135–142.
[9] K. Krippendorff, Content Analysis: An Introduction to Its Methodology, Sage Publications, Inc., Thousand Oaks, 2003.
[10] J. Landis, G.G. Koch, The measurement of observer agreement for categorical data, Biometrics 33 (1) (1977) 159–174.
[11] S. Li, Rock em, sock em Robocode! <http://www-128.ibm.com/developerworks/java/library/j-robocode/>, 2002.

[12] F. Modugno, A.T. Corbett, B.A. Myers, Graphical representation of programs in a demonstrational visual shell-an empirical evaluation, ACM Trans. Comput. Hum. Interact. 4 (3) (1997) 276–308.
[13] Morae, TechSmith, <http://www.techsmith.com/morae.asp>, 2009 (accessed August 2009).
[14] B.A. Myers, J.F. Pane, A. Ko, Natural programming languages and environments, Commun. ACM 47 (9) (2004) 47–52.
[15] M. Ponsen, P. Spronck, Improving Adaptive Game AI with Evolutionary Learning, Masters Thesis, Delft University of Technology, 2004.
[16] H.M. Raghoebar-Krieger, D. Sleijfer, W. Bender, R.E. Stewart, R. Popping, The reliability of log book data of medical students: an estimation of interobserver agreement, sensitivity and specificity, Med. Educ. 35 (7) (2001) 624–631.
[17] S. Reeves, B. Brown, E. Laurier, Experts at play: understanding skilled expertise, Games & Culture 4 (3) (2009) 205–227.
[18] A. Repenning, Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments, Ph.D. Thesis, University of Colorado at Boulder, 1993.
[19] A. Repenning, Bending the rules: steps toward semantically enriched graphical rewrite rules, in: VL '95: Proceedings of the 11th International IEEE Symposium on Visual Languages, 1995, pp. 226.
[20] D.C. Smith, A. Cypher, J. Spohrer, KidSim: programming agents without a programming language, Commun. ACM 37 (7) (1994) 54–67.
[21] S. Stumpf, E. Sullivan, E. Fitzhenry, I. Oberst, W.-K. Wong, M. Burnett, Integrating rich user feedback into intelligent user interfaces, in: IUI '08: Proceedings of the 13th International Conference on Intelligent User Interfaces, 2008, pp. 50–59.
[22] Wargus, <http://wargus.sourceforge.net/>, 2009 (accessed August 2009).