Content determination through planning for flexible game tutorials

Luciana Benotti and Nicolás Bertoa

NLP Team, Universidad Nacional de Córdoba, Argentina
benotti@famaf.unc.edu.ar, nicobertoa@gmail.com

Abstract. The goal of this work is to design and implement an agent which generates hints for a player in a first person shooter game. The agent is a computer-controlled character which collaborates with the player to achieve the goal of the game. Such agent uses state of the art reasoning techniques from the area of artificial intelligence planning in order to come up with the content of the instructions. Moreover, it applies techniques from the area of natural language generation to generate the hints. As a result the instructions are both causally appropriate at the point in which they are uttered and relevant to the goal of the game.

1 Introduction

Nowadays, most game tutorials make the player follow a fixed script. Hence having a good or bad tutorial is still an art that depends on how good the script is at faking some freedom (for instance, by foreseeing possible reactions of the player) in order to make it more entertaining. *Natural language generation* can offer game designers techniques that, in the future, may transform tutorial design not in an art but in a science by providing efficient algorithms for managing the players knowledge and dealing with the players reactions in a dynamic way.

The goal of this work is to design and implement an agent (a computer-controlled character) which is able to generate hints that help a player advance in a game situated in a 3D virtual world. Such agent uses state-of-the-art reasoning algorithms such as *planning* in order to come up with the content of the instructions which are relevant at each point in the interaction. Furthermore, it applies techniques from situated natural language generation such as *common ground management* to generate the context-aware hints.

We believe that our contribution is interesting for the game industry as well as for the players. For players, this work is a first step towards more flexible, effective and entertaining tutorials; in order to sustain our claims we performed a human based evaluation. For the game industry, good tutorials are expensive to develop because it is costly to script all the appropriate game behaviors caused by different potential player reactions (some of which might never be triggered). Our approach can be seen as a way of automatically producing scripts that change according to the unpredictable player behavior and the changing environment, reducing the burden on the developer.

The paper proceeds as follows. In Section 2 we briefly review previous work using planning for natural language generation in virtual environments. Section 3 describes the game environment where we implemented the game tutorial, and discusses the process of planning problem generation from the game environment. Section 4 presents the process used to generate an instruction according to a plan. Section 5 describes the integration of our generated instructions into the game interaction as well as the management of common ground between the player and the agent. Section 6 presents the results of our human evaluation. Section 7 concludes the paper.

2 Natural language generation and planning

Natural language generation (NLG) is a subfield of Artificial Intelligence (AI) and Computational Linguistics. It is concerned with the construction of computer systems which can produce understandable texts in English or other human languages from some underlying representation of information. NLG systems combine knowledge about language and the application domain to automatically produce reports, explanations, help messages, and other kinds of texts.

The area of NLG is a new area of research, with less than a couple of decades of experience [10]. However, it is a rapidly developing field that has put forward promising techniques in the last few years [1], in particular thanks to the shared challenges proposed for comparing current NLG technologies. One of such challenges is called GIVE (Giving Instructions in Virtual Environments). GIVE [8] is particularly relevant for our work since it applies NLG techniques to the problem of generating natural language instructions in a 3D virtual world. As a result several techniques investigated by GIVE's research community are directly applicable to this work. In the GIVE task, human players try to solve a treasure hunt in a virtual 3D world that they have not seen before. The computer has a complete symbolic representation of the virtual world. The challenge for the NLG system is to generate, in real time, natural-language instructions that can guide the player to the successful completion of their task. Only the player can effect any changes in the world, by moving around, manipulating objects, etc. Figure 1 shows a screen-shot of the user's view on the 3D world. On the top of the picture, the current instruction given by the NLG system is displayed.

The NLG tasks and techniques that are relevant for GIVE and for this work are content determination through planning, situated generation of referring expressions and common ground management; we will briefly discuss them.

The task of content determination consists in deciding which information to communicate to the player such that the information is appropriate at the current point in the interaction. Such task can be implemented using the inference technique of planning [3]. As a result, the instructions are both relevant to the goal of the game and causally appropriate at the point in which they are uttered.

The situated generation of referring expressions area [10] provides algorithms for coming up with descriptions of objects so that the player can identify such

Fig. 1. The player's view during the GIVE Challenge

objects (e.g. "the door in front of you") taking into account both static (e.g., color) and dynamic properties (e.g., visibility) of an object.

Finally, common ground management is the task of generating grounding acts when appropriate according to the behavior of the player. Grounding acts are utterances that, in a strict sense, do not add new information to the discourse but instead they reinforce old information. For example, if the NLG system tells the player "take the left green kit" and the player turns slightly left to face an object, the system may generate a positive grounding act such as "yes" if that was the right object, or "no" otherwise [11].

These three techniques are useful for generating hints that convey appropriate information to the player and that consider the player reaction in order to reinforce the information. In [5] the authors propose to cast all three tasks as a planning problem. Such integration is proven to be successful for small and discrete game worlds, however it does not scale to continuous game worlds in which the player can move freely (and continuously) in 3 dimensions. As a result in this paper we propose to use planning for handling only content determination while we use more traditional NLG methods for the other two tasks.

3 Finding plans from a game state

The design of game characters decision making is one of the most challenging tasks when designing a game; deciding what the expert game partner is to say next is, by no means, an exception to this rule. In most games, the characters decision making is either implemented in an ad-hoc way which is scripted for that particular game, or designed as a state machine which needs to be kept small because of scalability issues [9]. A believable game partner cannot get away with a decision making process that is either scripted or small. First, the agent needs to react appropriately to infinitely many player reactions which cannot be predicted and scripted. And second, it needs to reason over complex

game states in a goal directed way in order to be able to give instructions that are both appropriate in the current game context and relevant to the game goal.

Our approach to implement the decision making process of the expert game partner (also called content determination in the NLG area) is to use an off-the-shelf automated planner. The planner that we use is FastForward (FF). FF [7] can handle big planning problems (with over a million objects) and return plans of thousands of actions in seconds. Moreover, it can handle our game environment (with a couple of hundreds of objects and plans with a maximum length of 250 actions) in a few milliseconds. Automated planners have reached a maturity level in which they can be used in real time applications even if the need of re-planning is high. In our setup there is a high need for re-planning because the behavior of the player is non-deterministic and unpredictable.

The game environment for which we have implemented our game tutorial is first person shooter game (FPS). The game scenario, called Igor[1], was developed using the Irrwizard framework[2] and extended in C++. As in most FPS games, the player is situated in a 3D world where he can perform several actions such as walk, jump, climb stairs, shoot and pick up different items which have different effects. The goal of the game is to kill a creature that is wandering in the 3D world. The creature cannot be killed only by shooting at it because it has a self healing mechanism that needs to be turned off first by deactivating a series of power rays in a given order. The NLG agent, which guides the player through the tutorial, knows which is the right sequence of rays as well as the position of each of these rays. It is also able to recognize other items (such as poison or health) and is aware of their effect.

The agent must extract information from the environment and represent it in the standard planning language: PDDL [6]. PDDL is intended to express the physics of a domain, that is, what facts are true in the world, what actions are possible, and what the preconditions and effects of actions are. The agent represents the number of rays that the player needs to pick up and its order. Also, the agent codifies in the planning problem the game waypoints [9] and adjacencies which are used by the planner to find the nearest path between the player and the items to pick up. Moreover, the agent constantly verifies the health level of the player and enemy and updates its condition (attacking, wandering, dead etc) in the planning problem.

Once the agent has collected all the information about the current state of the game evironment, it generates a planning problem and sends it to the planner. Then the planner generates a plan that the agent will use to generate the instructions. The planner takes, on average, 8.61 milliseconds to find a plan for our game planning domain (with a standard deviation of 7.46 milliseconds). Therefore, the planner can be integrated smoothly into the real time interaction of our middle sized-game.[3]

---

[1] https://sites.google.com/site/nicolasbertoa/igor

[2] http://irrwizard.sourceforge.net/

[3] The maximum length of a plan in our game environment is 250 actions (notice that plans and plan length varies depending of the current state of the interaction).

4 Using plans to decide what to say

This section explains how the agent uses the plan obtained from the planner. The plan is composed by a series of steps that the player must follow to meet the plan goal. When the agent has a new plan, it must decide what instruction to say next. Suppose that the planner gives us the plan steps ⟨ MoveTo (w4,w5), MoveTo (w4,w5), MoveTo (w4,w5), MoveTo (w4,w5), MoveTo (w4,w5), MoveTo (w4,w5), TakeKey(blueKey,w11)⟩. This plan contains two types of actions, MoveTo and TakeKey. The first type of action indicates that the player has to move from one way-point to another, and the second indicates the ray that the player hast to pick as well as the way-point where it is located.

These steps form a plan to pick up the ray blueKey as illustrated in Figure 2. Now, the agent has the problem of deciding which plans steps to verbalize, that is, it needs to do content determination based on the plan steps. Notice that the naive approach of verbalizing all plan steps would result in a very repetitive and over restrictive game partner. Therefore we made the content determination process of our agent dependent on the area which is currently accessible to the player. We define this area as the set of waypoints that are directly visible to the player or visible by turning around on his current position, we will say that these waypoints are *visible 360*.

The left drawing in Figure 2 shows that there are only three visible waypoints of the plan at 360 degrees from the position of the player (waypoints 4, 5 and 6). The others are blocked by the environment. From these waipoints, we verbalize the one that has a referring expression which uniquely identifies it. Waypoints 4 and 6 are in the middle of rooms, so, there is no referring expression that unambiguously describes them. But in the case of the waypoint 5, there is a door, so we can use a referring expression to uniquely identify the object involved. As a result, in the situation illustrated in the left drawing the instruction "go through the door in front of you" is generated.

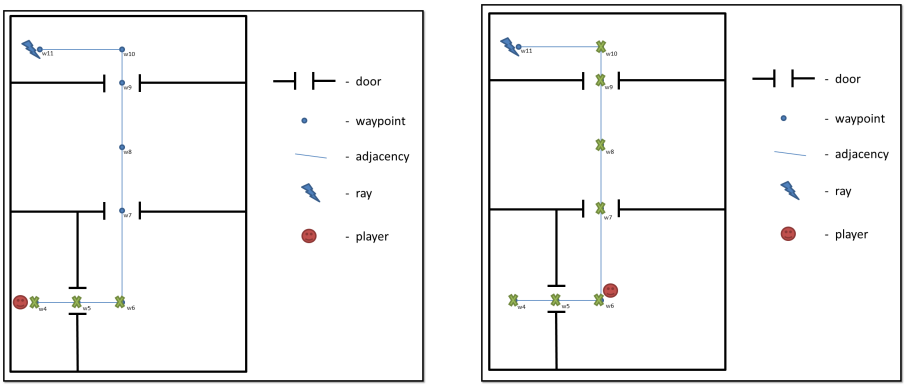

Fig. 2. Path of a sample plan with waypoints visible 360 highlighted.

However, what happens in the case when there is more than one object that can be uniquely identified? The right drawing Figure 2 shows this case. There are seven visible waypoints of the plan at 360 degrees from the position of the player, and three of them have an object that can be uniquely identified: 5 (has a small door), 7 (has a big door), and 9 (has some stairs). In this case, the agent generates an instruction which refers to the the furthest visible object in the path of the plan and verbalizes "see those stairs in front of you?" before saying "take them". This strategy is used when referring to objects that are far away from the player, see Section 5 for details.

Fig. 3. Instructions generated using the plan and the player's visibility

The screen-shots in Figure 3 illustrate the process we just explained. In the screen-shot we have drawn the visible waypoints and the paths between them. The arcs between waypoints illustrate the actions in the plan. In the screen-shot in the top of Figure 3 the planned actions could be verbalized as "go forward, go through the door, go forward, go forward, do you see those stairs in front of you?, take them, go forward". As we said, we have decided to verbalize the last action in the sequence which contains a referring expression which uniquely identifies the object involved in the action. In this example, this is the case for the action "do you see those stairs in front of you?" since "those stairs" are a distinguishing referring expression (given the current position of the player) for the stairs that are further away. We say that the first actions, namely "go forward, go through the door, go forward, go forward" are *left tacit* [2] and they are expected to be inferred by the player given the causal constraints of the world (in simpler terms, in order to see "those stairs" better the player will need to approach them). The screen-shot at the bottom of Figure 3 ("Turn left"), illustrates an instruction whose goal is to make directly visible a waypoint that is visible 360.

5 Guiding an unpredictable player in a changing game

Once the agent generates the instruction, it will show that instruction to the player. Of course, the player may decide to follow the instruction or to do something else. A good game tutorial should handle both situations robustly; we

discuss our strategy in Section 5.1. Furthermore, not only the player but also our environment can behave in a non deterministic way. The game partner should be able to sense and react to a changing environment in an appropriate way; Section 5.2 addresses this issue. In both of these situations grounding acts can be used to reinforce instructions that were already communicated but were not successfully accomplished yet; we illustrate how positive and negative grounding acts are used in Section 5.3.

5.1 Dealing with unpredictable behavior

Our strategy for dealing with the player's unpredictable behavior is to find a new plan (that is, to replan) every time the player gets *too far away* from the current plan. The crucial point here is to define what it means to get too far from the current plan. Since we base our content determination procedure in the player visibility we also use the same strategy to decide when to replan. The agent will replan when all waypoints in current the plan are no longer visible 360 for the player.

Let's illustrate our strategy by an example. Figure 4(a) shows an example scenario and the waypoints of the plan which are visible 360.

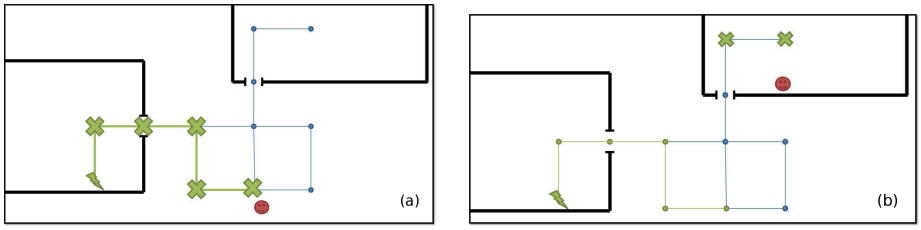

Fig. 4. Re-planning example.

Now suppose that the player does not follow the instructions and ends up in the situation illustrated in Figure 4(b). In this new situation, re-planning is needed because none of the waypoints in the current plan (marked in green) is visible anymore from the current player position.

Summing up, the agent will re-plan when there are not visible waypoints of the plan, because is very difficult that the agent achieves the reorientation of the player towards the goal. This strategy results effective (and not overly restrictive) while guiding the player to the game goal as shown by our evaluation results in Section 6.

5.2 Dealing with a changing game environment

Suppose we have 4 ray items: blue, green, violet and red and the correct sequence to deactivate the enemy's defense mechanism is red, blue and red. The left drawing in Figure 5 shows the plan that the agent obtained from the planner (the

plan is simplified not to show move actions for presentation simplicity). What the planner cannot handle (without replanning) is the fact that, when the player picks up some object of the game, the object repositions randomly in another way-point. For example, in the right drawing in Figure 5 we can see that the obtained plan is incorrect because the red ray was repositioned at a different way-point. The planner assumes a deterministic environment, we deal with a non deterministic environment, such as our game, by means of replanning. That is, when the position the environment changes in a non deterministic way (for example when a ray changes its position randomly), the agent replans.

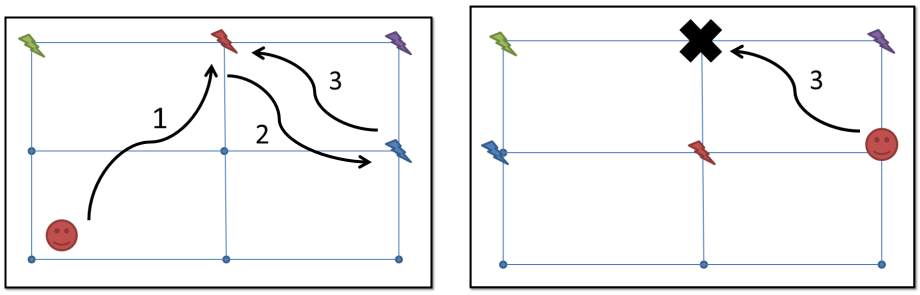

Fig. 5. Re-planning example due to non-deterministic environment.

5.3 The importance of the grounding acts

Grounding acts are utterances that, in a strict sense, do not add new information to the discourse but instead they reinforce old information. Grounding acts are not required from an informational point of view of communication, however, they play an important role in order to cope with changing environments, and the unpredictable behavior of the player.

The Figure 6 shows a case of the positive grounding act "Get this ray" which follows the instruction "Turn left to see the ray" when the ray becomes visible. This utterance is a positive grounding act because, it does not add new information to the discourse. The player should be able to figure out that he should pick up this ray, otherwise, why would have the agent guided him to it? However, natural language is ambiguous and the player may not draw this conclusion so the reinforcing achieved by the grounding act is indeed useful.

The other screen-shot in the Figure 6 illustrates a case of negative grounding acts with "This is not the ray you need", which follows the instruction "we need to find the green ray", when the player hovers the mouse pointer on the red ray. With this instruction, the agent prevents the player from re-activating the protection mechanism of the creature as a result of picking a ray in the wrong sequence. Again, in a strict sense, the negative grounding act is not necessary because the player has already been told that the next ray that is needed is

Fig. 6. Grounding acts that can be generated as a reaction to the player actions

green. However, the player may not remember this and may try to take the red ray which is directly in front of him.

The screen-shots in Figure 7 illustrate a typical example of common ground creation between the agent and the player. The screen-shot in the top shows the instruction "Do you see that ray in front of you?". With this instruction the agent wants that the player to focus his attention on a particular ray. As a result it is to be expected that the player gets closer to the ray. In this new situation, the agent generates the instruction "Pick it up". It is clear from this instruction that the agent wants the player to pick up the blue ray in front of him, but we can see that neither the ray nor its position or color are included in the instruction. The pronoun can be used because the intended ray is already in the the common ground between the agent and the player.

Fig. 7. Multi-utterance instructions which create and use the common ground

6 User evaluation

In this section we describe the results of the human evaluation of our agent. For our evaluation we used objective and subjective metrics. Objective metrics were collected by logging the player behavior and the subjective metrics by asking players to complete a questionnaire. We used the same metrics that are used in

the GIVE Challenge [8] to evaluate systems in terms of the effectiveness, naturalness of instructions and the engagement of the interaction. We gathered 10 volunteers for the evaluation. The demographic characteristics of the volunteers that we collected are the following: they were all male, the average age was 20 years and all of them were gamers. This subject population is the target market of the kind of game we implemented. The results that we obtained using the objective metrics are shown in Figures 8 and 9.

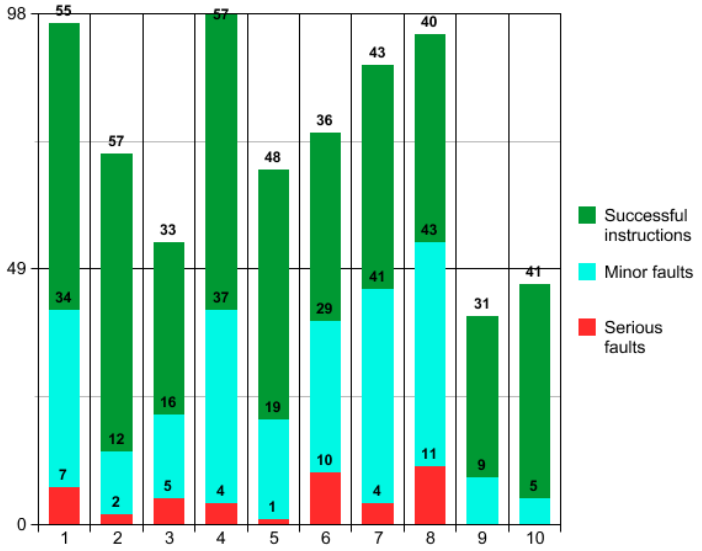

Fig. 8. Objective metrics: Successful instructions, minor and serious faults per player.

The Figure 8 shows the number of successful instructions, minor faults and serious faults. An instruction is considered successful if the player did exactly what the system asked him to do, an instruction is considered a minor fault if the player deviated from the instruction without causing a replanning, and an instruction is a serious fault if the agent had to replan after uttering it. Their averages are 44.1, 28.9 and 4.4, respectively. We can see that there were few serious faults, this suggests that the agent was successful in the task of guiding the player. Also, the players deviated from 40% of the instructions and, according to the game logs, this is correlated to the presence of enemy which causes the player to not follow the plan because he is busy shooting. Finally, approximately 60% of the instructions were directly successful, the player did exactly what the agent asked him to do in more than half of the cases.

The Figure 9 shows the average of the other objective metrics we collected. The successful instructions were completed quickly, which suggests that the instructions were easy to understand and execute. Also, players were able to complete the level quickly without visiting all the map waypoints (they only visited, in average, 65.5% of the map).

	Average	Standard deviation
Time of completeness per instruction (seconds)	28.9	17.04
Waypoints traveled	65.5	18.76
Playing time (minutes)	2.4	1.29

Fig. 9. Objective metrics: average times and distance traveled.

The Figure 10 shows the results of the subjective metrics. From these metrics we learned that the players considered that the instructions were too repetitive (no syntactic or lexical variability). Also, we realized that the instructions where not visible enough time for the players to read them comfortably. We were pleased to observed that the statements that were related to entertainment obtained very high percentages, which suggests that the player had fun, that is the main objective of any game. Another strong point is that the instructions were considered clearly worded and were understood. Also, players perceived that most of the time the agent helped the players when they were confused.

Number	Statement	%
1	The system used words and phrases that were easy to understand	80
2	I had to re-read instructions to understand what I needed to do	70
3	The system gave me useful feedback about my progress	50
4	I was confused about what to do next	75
5	I was confused about which direction to go in	80
6	I had no difficulty with identifying the objects the system described for me	80
7	The system gave me a lot of unnecessary information	10
8	The system gave me too much information all at once	75
9	The system immediately offered help when I was in trouble	80
10	The system sent instructions too late	50
11	The system's instructions were delivered too early	10
12	The system's instruction were visible long enough for me to read them	35
13	The system's instructions were clearly worded	90
14	The system's instruction were repetitive	90
15	I really wanted to kill that creature	95
16	I lost track of time while solving the overall task	80
17	I enjoyed solving the overall task	90
18	Interacting with the system was really annoying	55
19	I would recommend this game to a friend	70
20	The system was very friendly	70
21	I felt I could trust the system's instructions	60

Fig. 10. Results of the subjective metrics.

We consider that these results are encouraging, in particular because one of the main goals of our agent was to be able to participate in an engaging interaction while still providing useful instructions whenever the player needed help.

We are considering different techniques to improve the agent in those characteristics in which it did not get such good results (for example, by doing corpus based generation in order to avoid being repetitive [4]).

7 Conclusions

In this paper we have presented an agent which is able to generate hints that help a player win a level in first person shooter game. Such agent uses state of the art reasoning such as planning in order to come up with the content of the instructions and state of the art techniques from natural language generation to context-aware generate referring expressions and grounding acts.

Currently, most game tutorials make the player follow a fixed script which must contemplate as many situations as possible. When a possible reaction of the player is contemplated, the developer must add new code, increasing the development cost. Our approach is a way of automatically producing scripts that change according to the unpredictable player behavior and the changing environment, reducing the burden of the developer. Our evaluation shows that natural language techniques have a lot to contribute to the game community.

References

1. Alonso-Lavernia, A., De-la Cruz-Rivera, A., Grigori, S.: Generation of natural language explanations of rules in an expert system. In: Gelbukh, A.F. (ed.) CICLing. Lecture Notes in Computer Science, vol. 3878, pp. 311–314. Springer (2006)
2. Benotti, L.: Incomplete knowledge and tacit action in a dialogue game. In: Workshop on the Semantics and Pragmatics of Dialogue. pp. 17–24. Italy (2007)
3. Benotti, L.: Implicature as an Interactive Process. Ph.D. thesis, Université Henri Poincaré, INRIA Nancy Grand Est, France (2010), supervised by P. Blackburn. Reviewed by N. Asher and B. Geurts
4. Gandhe, S., Traum, D.: Creating spoken dialogue characters from corpora without annotations. In: Proceedings of Interspeech. Belgium (2007)
5. Garoufi, K., Koller, A.: Automated planning for situated natural language generation. In: Proceedings of the 48th ACL. Uppsala (2010)
6. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Tech. Rep. R.T. 2005-08-47, Università degli Studi di Brescia, Italy (2005)
7. Hoffmann, J.: Where 'ignoring delete lists' works: Local search topology in planning benchmarks. Journal of Artificial Intelligence Research 24, 685–758 (2005)
8. Koller, A., Striegnitz, K., Gargett, A., Byron, D., Cassell, J., Dale, R., Moore, J., Oberlander, J.: Report on the second NLG challenge on generating instructions in virtual environments (GIVE-2). In: Proceedings of the International Natural Language Generation Conference (INLG). Dublin (2010)
9. Millington, I., Funge, J.: Artificial Intelligence for Games. Morgan Kaufmann, Burlington, MA, USA (2009)
10. Reiter, E., Dale, R.: Building natural language generation systems. Cambridge University Press, New York, NY, USA (2000)
11. Traum, D.: A Computational Theory of Grounding in Natural Language Conversation. Ph.D. thesis, Computer Science Dept., U. Rochester, USA (1994), supervised by James Allen