The Loria Instruction Generation System L in GIVE 2.5

Alexandre Denis
INRIA Grand-Est, LORIA-Nancy
54603 Villers les Nancy Cedex, France
denis@loria.fr

Abstract

This paper presents the instruction generation system L submitted by the LORIA and TALARIS team to the GIVE challenge 2011 (GIVE 2.5). The system L takes the same approach to instruction generation than its predecessor the system NA that participated to the GIVE challenge 2010 (GIVE 2), the two systems are almost the same except minor modifications. We present the strategy of these systems, namely a directive, low level, navigation strategy ("Go left") and a referring strategy based on focus and sub-contexts (Denis, 2010) ("Not this one! Look for the other one"). These strategies were successful, as shown by the GIVE 2 challenge, but also had some deficiences we tried to fix for GIVE 2.5. We explain these deficiencies and how we fixed them in GIVE 2.5. We eventually present the preliminary results that show that the system L, like the system NA, achieved a very good result both in objective and in subjective metrics.

1 Introduction

The GIVE challenge (Byron et al., 2009; Koller et al., 2010) is a framework that enables to evaluate instruction giving systems in a 3D setting. Players connect to the framework and are paired randomly with a system that will guide them through a 3D maze to retrieve a trophy. Each system must develop its own strategy to instruct the player to move (*navigation strategy*) and to push buttons to open doors or deactivate alarms (*referring strategy*). The systems must also make sure to monitor the player behaviour and provide him the necessary feedback to put him back on track if he performs wrong actions. From this framework we can draw two kinds of results, the objective results (task success rate, duration, number of words, etc.) and the subjective results (overall evaluation by the player, friendliness, etc.), see (Koller et al., 2010). These two metrics are both helpful to assess the quality of the systems.

We describe in this paper the system L, developed by the LORIA laboratory that participated to GIVE 2.5. The system is very close to the system NA that participated to the former challenge GIVE 2 (Denis et al., 2010). Thanks to GIVE 2 metrics, we were able to draw some interesting conclusions about the efficiency of the system and we tried to improve the existing flaws for GIVE 2.5. In section 2, we first present the previous system NA, and describe its navigation and referring strategies. We then show in section 3 what was wrong with the NA choices, in which situations it was not optimal, and how we circumvented the problems in the system L. We conclude in section 4 with the preliminary results and show that the performance of the system L is better than system NA.

2 NA System

In this section we describe the NA system that participated to the GIVE 2 challenge (Denis et al., 2010). We first present the whole instruction giving strategy and the main loop. Then we present the two kinds of instructions at hand, *move instructions* and *push instructions* and how these two instructions are both verbalized and monitored. We also describe three mandatory components, namely the replanning mechanism, the acknowledgement and warning system and the messaging manager.

2.1 Instruction giving

Like other systems, the NA system relies on the plan returned by the planner provided with the framework. However, it does not directly rely on this plan because of its too fine-grained granularity and builds an higher level plan. The general idea to build the high-level plan (or *instruction plan*) is to iterate through the plan returned by the planner (or *action plan*) and gather move actions. For instance, when a move action takes place in the same room than a push action, the move action and the push action are gathered into a single push instruction. Or when two move actions take place in the same room, they are gathered into a single move instruction. The plan is iterated and a rule-based matching algorithm rewrites the actions into instructions.

```
instr   (push(b₃),
         actions:(move(r37,r42), push(b₃)))
```

Figure 1: A push instruction gathering a move and a push action

Following the plan consists in providing the instructions at the right time, and monitoring the success or failure of actions. The main loop thus consists of two parts:

- pop a new expected instruction from the instruction plan when there is no current one
- evaluate the success or failure of the expected action and verbalize it

For each instruction, two functions have then to be specified:

- how to verbalize the instruction ?
- how to monitor the success or failure of the instruction ?

We now detail these two functions for both move and push instructions as they were implemented in NA.

2.2 Move instructions

2.2.1 Verbalizing move instructions

The verbalization of a move instruction consists basically in providing the direction to the goal region. If there is a door located at the goal region, the verbalization is *"Go through the doorway + direction"*, and if there is not, the verbalization is simply *"Go + direction"*. The direction is computed by taking the angle from the player position to the goal region, and we only consider four directions *"in front of you"*, *"to your right"*, *"to your left"* and *"behind you"*.

Nevertheless, there could be cases in which the goal region of the high level move instruction is not the most direct region. For instance, the room in figure 2 being shaped like an U, the player has to move to region $r3$, but because the moves to $r2$ and $r3$ are in the same room, they are aggregated in a single move instruction. But if we would directly utter the direction to the goal region $r3$, given the player orientation we would utter *"Go to your left"*. Instead, we need to consider not the goal region of the move instruction but the different regions composing the expected move. The trick is to take the region of the last low-level move action composing the move instruction which is theoretically visible (modulo any orientation) from his current position. The computation takes into account visibility by testing if an imaginary ray from the player position to the center of a tested region intersects a wall or not. Thus, in this case, because a ray from the player to $r3$ intersects a wall, it is not chosen for verbalizing while $r2$ is picked and the produced utterance is eventually *"Go behind you"* (this instruction has been changed in the system L to *"Turn around"*, see section 3).

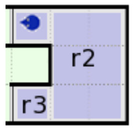

Figure 2: Example of U-turn

2.2.2 Monitoring move execution

The evaluation of the move instructions takes care of the lower action level. It simply tests if the player stands in a room for which there exists in the lower action level a region in the same room. In other words, a region is not on the way if it is located in a room where the player should not be. If this is the case, the failure of the move instruction is then raised (see replanning section 2.4). If the player reaches the goal region of the move instruction, then the success is raised and the current expectation is erased.

2.3 Push instructions

2.3.1 Verbalizing push instructions

Given the structure of the instruction plan, a push instruction can only take place in the room of the target button. The push instruction is actually provided in two steps: a *manipulate* instruction that makes explicit the push expectation *"Push a blue button"*, and a *designation* instruction that focuses on identifying the argument itself *"Not this one! Look for the other one!"*. The verbalization of the manipulate instruction does not make use of the focus, it only describes the object. On the other hand the verbalization of the designation instruction first updates the focus with the visible objects and then produces a referring expression.

This two steps referring process makes it easier to work with our reference setup. We tried applying Reference Domain Theory (RDT) for the reference to buttons (Salmon-Alt and Romary, 2000; Denis, 2010). The main idea of this theory is that the referring process can be defined incrementally, each referring expression relying on the previous referring expressions. Thus, after uttering a push expectation, a domain (or group) of objects is made salient, and shorter referring expressions can be uttered. For example, after uttering *"Push a blue button"*, the system can forget about other buttons and focus only the blue buttons. Expressions with one-anaphora are then possible, for instance *"Yeah! This one!"*. Spatial relations are only used when there is no property distinguishing the referent in the designation phase of the referring process. These spatial properties are computed, not from the player point of view, but to discriminate the referent in the domain, that is as opposed to other similar objects. For instance, we could produce expressions such as *"Yeah! The blue button on the right!"*. Vertical and horizontal orderings are produced, but only three positions for each of them are produced left/middle/right and top/middle/bottom. We also found it important to have negative designation instructions such as *"Not this one"* when there are focused buttons in the current domain that are not the expected buttons. Thanks to the referring model, we just have to generate *"Not"* followed by the RE designating the unwanted focus. More details about the use of Reference Domain Theory in the GIVE challenge can be found in (Denis, 2010).

2.3.2 Monitoring push execution

The evaluation of the success of a push expectation is straightforward: if the expected button is pushed it is successful, and the push expectation is erased such that the main loop can pick the next instruction, if a wrong button is pushed or if the region the player is standing in is not on the way (see section 2.2.2) then the designation process fails.

2.4 Replanning

It is often the case that the expected instructions are not executed. A simple way to handle wrong actions would be to relaunch the planning process, and restart the whole loop on a new instruction plan. However, we need to take into account that the player may move all the time and as such could trigger several times the planning process, for instance by moving in several wrong regions, making then the system quite clumsy. To avoid this behavior, we simply consider a *wait* expectation which is dynamically raised in the case of move or push expectation failure. As other expectations, the two functions, verbalize and evaluate have to be specified. A wait expectation is simply verbalized by *"no no wait"*, and its success is reached when the player position is not changing. Only when the wait expectation is met, the planning process is triggered again, thus avoiding multiple replanning triggers.

2.5 Acknowledging and warning

Acknowledging the behavior of the player is extremely important. Several kinds of acknowledgments are considered throughout the instruction giving process. Each time an action expectation is satisfied a *positive* acknowledgement is uttered such as *"great!"*, or *"perfect!"*, that is when the player reaches an expected region or pushes the expected button. We also generate acknowledgements in the case of referring even if the identification expectation is not represented explicitly as an action. When the player sees the expected button, we add *"yeah!"* to the generated referring expression. This acknowledgement does not correspond to the success itself of the action, but just warns the player that what he is doing is making him closer to the success. *Negative* acknowlegdments are also uttered, when there is an expectation failure ("no no wait") or when there is a visible button that could be the referent ("not this one").

However it is as necessary to warn the player when something went wrong as warning him that something *could* go wrong. Indeed, if the player steps on an alarm the game is lost. It is therefore quite important to warn the player about alarms. The NA system first provides a warning at the beginning of the game by explaining that there are red tiles on the floor and that stepping on them entails losing the game. But it also embeds an alarm monitor. If at any time, the player is close to an alarm, the system produces an utterance *"Warning! There is an alarm around!"*. In order to avoid looping these messages when the player passes by alarms, a timer forbids uttering several alarm warnings. But if the timer goes off, new alarm warnings could be potentially produced.

2.6 Messaging

Message management in a real-time system is a critical task that has to take into account two factors: the moment when an instruction is uttered and the time the instruction stays on screen. NA relies on a messaging system in which we distinguish two kinds of messages, the *mandatory* messages and the *cancellable* messages. Mandatory messages are so important for the interaction that if they are not received the interaction can break down. For instance, the manipulate instructions (e.g. *"Push a blue button"*) are crucial for the rest of the referring process. In the case they are not received, the player does not know which kind of button he has to press. Cancellable messages are messages which could be replaced in the continuous verbalization. For instance, the designation instructions (e.g. *"Yeah! This one!"*) or the direction instructions (e.g. *"Go straight"*) are continuously provided, each instruction overriding the previous one. We cannot force the cancellable messages to be displayed a given amount of time on the screen because of the fast update of the environment. Both types of messages are then necessary:

- if we would have only mandatory messages, we would risk to utter instructions at the wrong moment because of the delay they would stay on screen.
- and if we would have only cancellable messages, we would risk to miss critical information because they can be replaced too fast by next instructions.

The system then maintains a message queue in an independent thread called the message manager. Each message, either mandatory or cancellable, is associated to the duration it has or can stay on screen. The manager continuously takes the first message in the queue, displays it and waits for the given duration, then it displays the next message and so on. Before a new message is added to the queue, the message manager removes all pending cancellable messages while keeping mandatory messages. It then adds the message, and if the current displayed instruction is cancellable it stops the waiting.

3 Improvements in system L

We present in this section some of the problems of system NA and how we fixed them for GIVE 2.5 in system L.

3.1 Navigation strategy

While the NA navigation strategy was quite effective, and in general praised by the subjective assessment, it required some modifications. Some players were confused with the "doorway" verbalization either because they were not native speakers and did not know the word, or because they did not consider it as a natural wording. Indeed, because there was no visible door and only openings in walls, this verbalization was confusing. In system L, it simply has been removed and a shorter instruction "Go + direction" has been preferred. Moreover, thanks to the free-text feedback, we found out that the verbalization "Go behind you" of NA was clearly inappropriate, several players complaining about its non-naturalness and we replaced it by a simpler "Turn around" in system L.

3.1.1 Referring strategy

The changes in navigation strategy were purely cosmetic. On the contrary, despite its efficiency, the NA referring strategy had some serious flaws and thus required deeper modifications. In NA we separated the referring process into two steps that could make use of different discrimination features, the first step for instance did not make use of focus or spatial relationship, the second step used focus and spatial relationship but only to disambiguate between visible buttons. However this strategy was failing in at least two cases:

- the first descriptive step was not working well if the player was too close to a button. Because in most cases an indefinite referring expression was uttered e.g. "Push a blue button", it raised the presupposition that *any button was appropriate*, and if the player was too close to a matching button, he would directly press it, even if it was the wrong one.
- the second step mostly based on focus was not working well in rooms where *a lot of similar buttons were present.* The player would receive first "Push a blue button" and would continuously receive instructions like "Not this one! Look for another one!". He would then have to turn around, looking at each blue button until he would find the right one.

These two cases have been found out either by looking at the raw datas, situations where wrong buttons are pushed, or duration between the instruction and the actual push, or by looking at the subjective assessments of the players.

The common solution to these problems was to introduce player-relative spatial discrimination e.g. "to *your* left", as was done by another system that participated in GIVE 2, the NM system, see (Denis et al., 2010). For system L we also relaxed the difference between the two referring steps and both use a combination of focus, description and relative direction. In the first step, to address the indefinite presupposition problem, we forbid the simple utterance like "Push a blue button" but included either focus-based discrimination like "Push a blue button but not this one", or player-relative spatial discrimination like "Push a blue button, it is on your left".

Unfortunately time prevented us to model correctly the pronoun anaphora in the RDT framework (Denis, 2010) and it has been hardcoded. The player-relative discrimination also helped a lot to solve the second issue, instead of looking at each button, the player was directed immediately to the intended referent.

4 Results and conclusion

The preliminary results of the GIVE 2.5 challenge are consistent with the results of the GIVE 2 challenge (Koller et al., 2010). While the system NA achieved 47% average task success in GIVE 2, the preliminary results show that the system L achieves 67.2% in GIVE 2.5, and like last year it is in the top three systems. It is also the fastest system while using the smallest number of words to achieve task success. On the subjective level, the system has been positively evaluated and is in the first group for almost all metrics. The weakest point is shown by the *task progress feedback* metric. The system receives its lowest mark for the evaluation item "The system gave me useful feedback about my progress". This result is normal, and in line with the previous challenge, since the system does not provide any information about the task at hand but only gives move and push instructions. Other systems that participated to GIVE 2.5, for instance system C, are much more talkative (hence taking more time) and describe in details the task and the current progress like the remaining number of buttons. However, if giving task feedback is a necessary feature, we could question how much these approaches are task independent and if we could draw some general principles underlying the verbalization of task progress.

References

Donna Byron, Alexander Koller, Kristina Striegnitz, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2009. Report on the First NLG Challenge on Generating Instructions in Virtual Environments (GIVE). In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 165–173, Athens, Greece, March. Association for Computational Linguistics.

Alexandre Denis, Marilisa Amoia, Luciana Benotti, Laura Perez-Beltrachini, Claire Gardent, and Tarik Osswald. 2010. The GIVE-2 Nancy Generation Systems NA and NM. Technical report, INRIA Grand-Est/LORIA.

Alexandre Denis. 2010. Generating Referring Expressions with Reference Domain Theory. In *Proceedings of the 6th International Natural Language Generation Conference - INLG 2010*, Dublin Ireland.

Alexander Koller, Kristina Striegnitz, Andrew Gargett, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2010. 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.

Susanne Salmon-Alt and Laurent Romary. 2000. Generating referring expressions in multimodal contexts. In *Workshop on Coherence in Generated Multimedia - INLG 2000*, Mitzpe Ramon, Israel.