The GIVE-2 Nancy Generation Systems NA and NM

Alexandre Denis, Marilisa Amoia, Luciana Benotti, Laura Perez-Beltrachini, Claire Gardent, Tarik Osswald
INRIA Grand-Est, LORIA-Nancy
54603 Villers les Nancy Cedex, France
{denis,amoia,gardent}@loria.fr, {luciana.benotti,perez.laura,tarik.osswald}@gmail.com

Abstract

This paper presents the two instruction generation systems submitted by the team TALARIS at INRIA-Nancy to the GIVE-2 challenge 2010. The first system (NM) aims to provide high-level instructions by giving freedom to the user and relies on an extension of the Dale and Reiter's incremental algorithm to generate the referring expressions. The second system (NA) uses a more constrained directive navigational strategy and relies on Reference Domain Theory (Denis, 2010) to generate a wide range of referring expressions including pronouns, ellipsis and alternative phrases. The results show that despite NA is more successful than NM, NM is considered a bit more enjoyable by the subjective assessment.

1 Introduction

The last few years have witnessed a considerable growth of systems providing autonomous software agents and agent-based dialogue applications, such as pedestrian navigation assistance (Baus et al., 2002), tutorial systems (Callaway et al., 2006), etc. Accordingly, the need of services facilitating the interaction with such applications has increased.

In this context the role of instruction giving systems for real time task solving, i.e. systems which aid human users to achieve some task-oriented goal by means of natural language generation (NLG) is crucial to allow universal access to these services.

Recently, the Generating Instructions in Virtual Environments GIVE challenge (Byron et al., 2007), a shared NLG task, has raised extensive attention in the research community providing an infrastructure to evaluate instruction giving systems over the internet thus permitting task-based human evaluation at a much lower cost than standard lab-based experiment, thereby still providing results which show high correlation with those gathered in a classical lab-setup (Koller et al., 2009). The systems participating in the first edition of the challenge, GIVE-1 (Byron et al., 2009), had to deal with discrete worlds and mostly generated aggregations of step-by-step instructions using various strategies such as landmark-based navigation (Striegnitz and Majda, 2009), level-based discourse planning (Dionne et al., 2009) and dynamic level adaptation (Rookhuiszen and Theune, 2009).

The challenge of this year, GIVE-2, has two innovative aspects. First, it implements continuous navigation, i.e. permits continuous movements in the virtual environment. Second, the challenge focuses on evaluating some new aspects of instruction giving systems, such as for instance the emotional impact of the instruction giving strategy on users.

This paper describes the two instruction giving systems submitted by the team TALARIS at INRIA-Nancy to the GIVE-2 Challenge 2010. The first system (NM) provides a context aware extension of Dale and Haddock's (1991) incremental algorithm for the generation of referring expressions. The properties used in the descriptions include the set of static absolute properties defined in the domain such as color, shape, and a set of dynamic properties defined during the human-machine interaction, such as the relative position of user and target object in the virtual world.

Further, by applying the same referring strategy to both objects and locations, the system produces high level path descriptions thereby generating more user-friendly navigational instructions such as *"Search the red room with the coach"*. Finally, the descriptions of objects generated by the system are overspecified, in that they also include a description of the function of the object in the virtual world. This extension improves the alignment between user and system (common ground), for instance *"To open the door, push the green button behind you"*.

The second system (NA) is based on a more directive navigational strategy only providing expected directions, while relying on Reference Domain Theory (Denis, 2010) to produce a wider variety of referring expressions, including demonstrative pronouns, ellipsis and alternative phrases, for instance *"Push a blue button"*, *"Not this one! Look for another one!"*.

This paper is structured as follows. In Section

2 and 3, we present the architecture of the NM and NA systems respectively. Section 4 describes the general messaging strategy used by both systems. In Section 5, we discuss the results our system achieved in the GIVE-2 challenge 2010. Section 6 concludes with some final remarks.

2 The NM System

In this section we describe the NM system. The instruction giving strategy implemented in this system aims at providing context aware referring expression generation and user-friendly instructions enhancing user learning about the virtual domain. In order to meet these desiderata, the same referring mechanism is used for both objects and locations. Further, the descriptions of objects are overspecified in that they include a reference to their function.

2.1 Instruction Giving Strategy

The instruction giving strategy behind the NM system is summarized in Figure 3.

```
1: plan ← simplify(originalPlan)
2: while (plan ≠ ∅) do
3:    currentGoal ← PLAN.popNextAction();
4:    goalLocation ← currentGoal.getLocation();
5:    if (playerRoom = goalLocation) then
6:       describe(currentGoal)
7:    else
8:       describe(goalLocation)
9:    end if
10: end while
```

Figure 1: The Instruction Giving Framework

The original game plan provided as a service by the GIVE-2 challenge infrastructure is first abstracted to a list (*plan*) of high level "macroactions" representing the local goals the user have to accomplish during the game, i.e. a *push* or *take* action. A macroaction (see Figure 5) contains a reference to the `type` of action to be performed, to the `object` which should be manipulated, to the `location` (i.e. the room in which the object can be found) and to the `function` of the object in the virtual world (e.g. *open door* d_2).

```
macroaction(  type:push,
              obj:b3,
              function:open(door(d2)),
              location:room2)
```

Figure 2: An example of macroaction describing a "push-button" action. The goal of this action is to open a door.

System-user interaction is triggered by the position of the player. When the player is in the room in which the next scheduled action can be performed (*playerRoom = goalLocation*) the system generates instructions describing how to accomplish the action (*describe(currentGoal)*). Otherwise, the system outputs navigational instructions describing the room in which the next action can be performed (*describe(goalLocation)*). The fundamental step in generating such instructions consists in providing a referring expression describing the referent object.

The following subsection describes how the generation of referring expressions is realized within the system.

2.2 Generating Referring Expressions

The generation of referring expressions implemented in the system is based on a context aware extension of (Dale and Haddock, 1991) incremental algorithm. Our algorithm thus makes use of the notion of dynamic context and takes into account time dependent properties of the context, i.e. the positions of user and virtual agents and their states, as they are crucial for content determination.

In a dynamic environment in fact, not only the set of relevant individuals (the distractors) but also the set of properties which count as distinguishing, might change over time. To capture these changes, the set of individuals of a dynamic context is defined in terms of visual context, i.e. in terms of the subset of individuals of the domain that are visible to the user/player at a given time.

Further, the set of distinguishing properties used to compute a description for a target referent includes the set of *absolute or static* properties, i.e. properties which do not change during the interaction, such as the color or the shape of objects but also their (absolute) position with respect to particular objects that may be called "landmarks" and the set of *dynamic* properties, i.e. properties which might change during interaction, such as the positions of objects relative to other objects and to the instruction follower. For instance, the position of a button on the wall with respect to the instruction follower (on the left/right, in front of) is a relative position and thus a dynamic property.

```
1: if (object.hasType=room) then
2:    generateDefDescr(object)
3: else
4:    generateDefDescr(object)
5:    object.addFunction()
6: end if
```

Figure 3: Generating Descriptions of Rooms and Objects

As, given the set of properties taken into account, a distinguishing description for a given referent might not exist or might not be found, in some cases the context aware incremental strategy

generates underspecified reference. When this happens, the system outputs an indefinite description of the referent with a property including its relative position with respect to the player.

Summarizing, the referring expression generated by the context aware incremental strategy can be:

- a distinguishing description of the object, if a unique description of the object exists: *"The blue button"*.
- a distinguishing description containing absolute positions, i.e. positions with respect to landmark objects: *"The green button on the left of the picture"*.
- a distinguishing description including relative positions: *"The third button on your left"*.
- a non-distinguishing description including relative positions: *"A red button behind you"*.

As shown in (Foster et al., 2009), first describing the goal of an action to the instruction follower an then giving the instruction helps improving performance of an instruction giving system. Building on this idea, the system presented here generates overspecificied descriptions containing information about the *effect* of actions to enhance user learning. Thus, after a description for a target object has been generated, an *effect* property is automatically associated (through the definition of object function in the correspondent macroaction) to an individual on the basis of the specification of the object manipulating actions in the virtual world setting (*object.addFunction()*). To amplify the learning effect, this property is highlighted in the surface realization; in fact it is not localized in the definite noun phrase but realized as an appositive infinitive clause. So for instance, the strategy outputs referring expressions of the following type: *"To move the picture, push the red button on your left"*.

2.3 Navigational Instructions

Contrary to GIVE-1, in the GIVE-2 challenge systems are required to cope with continuous movements (rather than discrete steps as in GIVE-1). Thus, high level navigational instructions can be generated. As described in 2.1, the abstract plan (*plan*) we build up from the one provided by the GIVE-2 infrastructure permits us to associate each task-relevant action to the location in which it can be performed.

Thus within this framework, the NM system provides navigational instructions that correspond to room descriptions such as *"The red room with the safe"*. The choice of generating this kind of descriptions is motivated by the following two assumptions. First, letting the user free to choose his own way to the target location enhances the entertainment value of the game experience. And second, highlighting in the description the properties of a room makes easier for the user to remember them, thus enhancing his awareness of the game space and his learning.

```
1: nextRoom ← currentGoal.getLocation();
2: if hasDEFDescription(nextRoom) then
3:    verbalise(DEFDescription(nextRoom))
4: else
5:    if isAdjacent(nextRoom, playerRoom) then
6:       verbalise(relativePosition(nextRoom, player))
7:    else
8:       R ← list of rooms defining the path between
         current room and goal room
9:       if (room_i ∈ R & hasDEFDescription(room_i))
         then
10:         describe(room_i)
11:      else
12:         describe(R.popNextRoomOnPath())
13:      end if
14:   end if
15: end if
```

Figure 4: Strategy for describing rooms.

Figure 4 sketches the algorithm for the generation of room descriptions. The system makes a first attempt to generate a distinguishing description for the target room by applying the incremental algorithm to the whole game domain. If a distinguishing description for the room can be found, it is outputted by the system.

(1) *"Search the green room with the table"*

If the room to be described has no distinguishing description, the system tests whether it is adjacent to the player room. If this is the case, a description is outputted including the relative position of the target room with respect to the player.

(2) *"Go to the next room on your right"*

Otherwise, the same procedure is recursively applied to the list of rooms along the path between player and goal location.

Further, during the game the system records the rooms visited by the player so that "return"-statement such as (3) can be also generated.

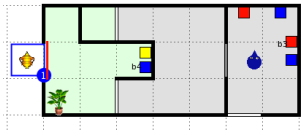
(3) *"Return to the room with the couch"*

2.4 An example interaction

We now illustrate how the system interacts with the user by focusing on some example scenario. In Scenario 1, the next scheduled action in the plan is a "push-button" action to be performed on button b_3.

The player is in the same room of the target button (*room*$_2$) thus, the system outputs a description of the action to be performed and of the referent

Scenario1:

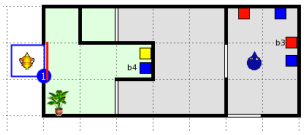


Local Goal:	push:button(b_3)
Goal Location:	$room_2$
PlayerRoom:	$room_2$
Sys:	[Describe local goal] *"To open the door, push the red button on your right"*
User:	[Turns right and push to the red button]
Sys:	[Acknowledges success] *"Great. This was the right button!"*

object b_3 which is visible to the user e.g. *"Push the red button on your right"* and further explains what is the goal of the action e.g. *"To open the door"*.

Suppose the user follows the instruction, finds and pushes the right button. The game scenario is now as in Scenario 2. The system has acknowledged the success of the user action and updated the local goal to a push-button action on button b_4 which is in $room_4$.

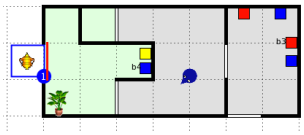
Scenario2:



Local Goal:	push:button(b_4)
Goal Location:	$room_4$
PlayerRoom:	$room_2$
Sys:	[Describes location of local goal] *"Search the green room with the plant"*

As target button and user are in different rooms, the system generates an instruction for guiding the user navigation and thus describes the room containing the referent to the player e.g. *"Search the green room with the plant"*. This is possible because in the given game scenario, the room has a unique distinguishing description.

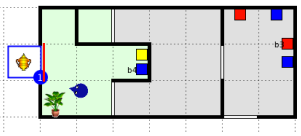
Scenario 3:



User:	[Searches the green room]

In Scenario 3 the user is still searching for the target room. When the user has reached the green room (e.g. Scenario 4) the systems generates the description of the action to be performed in that room and a description of the target object b_4 that in this case is not visible e.g. *"To move the picture, push the button behind you"*.

Scenario 4:



User:	[Has found the green room]
Sys:	[Acknowledges success] *"Great!"*
Sys:	[Describes local goal] *"To move the picture, push the blue button behind you"*

The green room $room_4$ is now marked as visited by the system which next time will use a "return"-statement to refer to it e.g. *"Return to the green room with the plant"*.

3 NA System

In this section we describe the NA system. This system differs from NM both on the move instruction level, by assuming a directive strategy, and on the push instruction level, by favoring focus over description.

3.1 Instruction giving

Like the NM system, the NA system does not directly rely on the plan returned by the planner because of its too fine-grained granularity and builds an higher level plan. Whereas the NM system adopts a very coarse granularity, by describing rooms instead of directions – when possible (e.g. *"Search the green room with the plant"*), the NA system systematically favors directions. 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 instructions are similar to the macroactions of the NM system with two differences. First, they do not specify the function of the instruction, such as opening a door for a button, and second they maintain the sequence of smaller actions they are gathering.

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

Figure 5: A push instruction gathering a move and a push action

Following the plan consists in providing the in-

structions 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 ? and how to monitor the success or failure of the instruction ? We detail these two functions for both move and push instructions.

3.2 Move instructions

3.2.1 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"*.

However, 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 6 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"*.

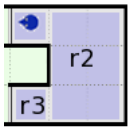


Figure 6: Example of U-turn

3.2.2 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 3.4). If the player reaches the goal region of the move instruction, then the success is raised and the current expectation is erased.

3.3 Push instructions

3.3.1 Verbalizing push instructions

Given the structure of the instruction plan, a push instruction can only take place in the same room than where the button is located. 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 for the reference to buttons (Salmon-Alt and Romary, 2000; Denis, 2010). The main idea of this theory is that the reference 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 reference 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 but-

tons. 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).

3.3.2 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 3.2.2) then the designation process fails.

3.4 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.

3.5 Acknowledging

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 an 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.

3.6 Alarm warning

If the player steps on an alarm the game is lost. It is therefore quite important to warn the player about alarms. The NM system just 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 we assume in the NA system that this was not sufficient, and thus add 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 pass by alarms, a timer forbids uttering several alarm warnings. But if the timer goes off, new alarm warnings could be potentially produced.

4 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. Both systems NM and NA rely on the same 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. *"Move forward"*) 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.

5 Discussion of the results

In this section we analyze the evaluation data collected by the GIVE-2 Challenge (Koller et al., 2010) on our two systems. We discuss the objective metrics proposed in the challenge as well as other objective metrics we have calculated ourselves. We perform an error analysis of the lost and canceled interactions, analyzing the reason for the low task success achieved by our systems in comparison with the task success achieved by the systems that competed in GIVE-1 (Byron et al., 2009). Finally, we interpret the observed subjective metrics.

5.1 Objective metrics

Across worlds, the NA system achieved an average of 47% task success while the NM system achieved an average of 30%. This task success rate was more stable across the different evaluation worlds for the NA system than for the NM system (see Figures 7 and 8).

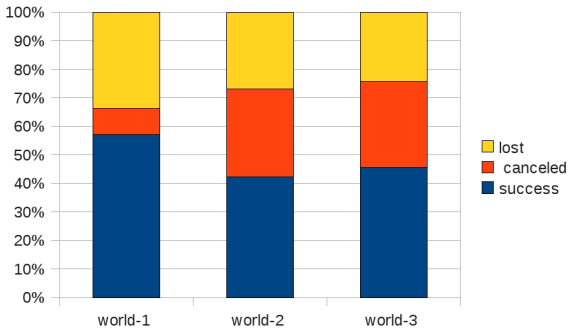


Figure 7: Success per world for the NA system

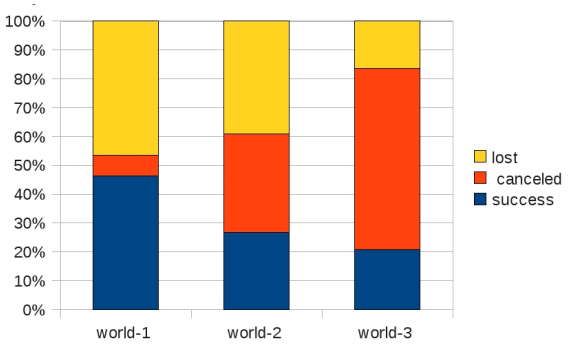


Figure 8: Success per world for the NM system

In particular, the NM system achieved only a 21% task success in world 3. We consider that this result is due to the fact that the navigation strategy of the NM system attempt to produce a distinguishing description for the target room using static properties of the game world (as explained in Section 2). This strategy was much more effective in world 1 because such descriptions exist for most rooms. In worlds 2 and 3, most rooms are indistinguishable using this strategy (see (Koller et al., 2010) for a 2D representation of the worlds). As a result the NM system switches to the strategy which includes the relative position of the target room with respect to the player (e.g. *"Go to the next room on your right"*) as explained in Section 2. However, the implementation of this strategy is buggy as discussed in Section 5.2. These observations correlate with observed subjective metrics discussed in Section 5.3.

As reported in (Koller et al., 2010), the distance traveled until task completion and the number of executed actions are similar for both systems, however the average time until task completion is considerably smaller in the NA system. We think that this can be explained by three reasons. First, the NM system gave more information about the task goals than the NA system, increasing the number of words per instruction (e.g. the NM system generates *"To open the door, push the blue button on your right"* when the NA system generates *"Push the blue button on your right"*). This is confirmed by the *words per instruction* metric of the Challenge. Second, the NM system gave more feedback about the task progress than the NA system (e.g. *"Great! You have to find other 2 buttons to open the safe."*), increasing the number of instructions per game. This is confirmed by the *instructions* objective metric (see (Koller et al., 2010)) and the Q3 subjective metric (see Section 5.3). Third, the NM system gave coarser grained instructions for navigation which did not specify exactly which action to do next (e.g. *"Search the red room"*). This correlates with the number of times that the user asked for help: the user asked for helped more than twice as often in the NM system than in the NA system. In sum, we think that all these factors added up to result in the longer completion time observed in the NM system. NA instructions are shorter and cognitively lighter (e.g. *"Yeah! This one!"*) allowing the player not only to read them fast but also to react fast.

5.1.1 Our objective metrics

In addition to the objective metrics collected by (Koller et al., 2010), we calculated additional measures related to the referring process. These measures are calculated on all games (success, canceled, lost). We only consider the first time a button is referred to, such that we avoid any effect due to memory.

- Success rate of the first reference. A reference is successful if, given a push expectation, the next pushed button is the expected button. In other words, it is a failure if a wrong button is pushed or if, because of replanning, a new push expectation is raised.

- Average duration of the first reference only if it is successful (in seconds).
- Average number of instructions of the first reference only if it successful.

	success rate	duration	instructions
world-1			
NA	88 %	10.6s	0.8
NM	84 %	12.1s	1.9
world-2			
NA	70 %	12.2s	2.5
NM	61 %	11.9s	5.5
world-3			
NA	91 %	8.5s	0.9
NM	79 %	8.4s	2.1

Table 1: Our objective metrics. Success rate of first reference, average duration and number of instructions.

On all worlds, NA obtains a better success rate than NM while uttering a fewer number of instructions. However, on world 2 and 3, NM is slightly faster than NA. The good overall success rate of NA may come from the negation e.g. *"Not this one!"* which prevents the player to push wrong buttons, and the higher number of instructions of NM is probably related to the direction-related instructions e.g. *"Push the red button on your right"* that change more often than the NA demonstrative strategy when the player is turning. On the other hand, the direction-related instructions improves slightly the duration. We observed that the NM referring strategy is much faster in the rooms containing similar objects. For instance, in the world 2, the parlor room is filled with blue buttons. The NA strategy takes an average of 14s to refer to the buttons of this room while NM takes only 10.1s. This big difference is caused by the NA strategy that forces the player to check visually each button until he can find the right one (*"Yeah! this one!"*) while the NM strategy provides a direct direction to it.

Some situations are problematic for both systems. The lowest success rate is found in world 2, especially in the hall which contains a grid of buttons. For the buttons *bhall* composing the grid, the success rate drops to 51% for NA and 43% for NM. It is interesting to note that the two strategies have actually two kinds of deficiencies. The NA system is failing because of the presupposition raised by the indefinite *"Push a blue button"* that any blue button could work. On the other hand, the NM system is failing because it produces very long descriptions like *"Push the blue button above the green button and on the left of the green button"* that are immediately replaced by new descriptions. Those two errors are discussed in details in the following section.

5.2 Error analysis

As argued in (Koller et al., 2010), it is surprising that the best performing system of the GIVE-2 Challenge achieves only the 47% of task success. This result is much lower than the task success achieved by the systems that participated in GIVE-1 (Byron et al., 2009). Two potential reasons for such low task success may be that either the systems are not good enough or the task itself is not engaging and then the users cancel or lose due to their lack of interest. The GIVE-2 task is indeed not engaging as made evident by the result of the emotional affect measures reported in (Koller et al., 2010). However, we have observed in the game logs that most cases in which a user cancels or loses the one that is to blame is not the user (who gets bored) but a contradicting or confusing behavior from the part of the system.

We first analyzed the lost games. In these cases, the performance of both systems could be greatly improved by a more elaborated alarm warning mechanism. The NM system has no warning system, and the NA alarm warning is quite simplistic, although it's effect is already observable (see Figures 7 and 8).

Analyzing the reasons for cancellation (and hence lowering its percentage) is more complex. There is only one reason why users lost, they step on an alarm, but there are many reasons why users cancel. Below we list the most frequent causes of cancellation that we observed in the logs of our systems.

Based on our observations in the logs we conclude that the navigation and the generation of referring expressions is much more difficult in GIVE-2 setup that in GIVE-1. This is due to the multiplication of positions and orientations of the player in the continuous world offered by GIVE-2. The amount of relative positions and the fact that the user can (and does) move very fast make instruction giving and monitoring a much more challenging task, putting timing in a crucial role and making the interleaving of instructions and the preservation of coherence essential.

5.2.1 NA system

We observed the following bugs for the NA system:

- The system gets mute.
- There is a faulty calculation of left/right. In order to discriminate similar buttons, the referring process in the NA system makes use of the spatial ordering by projecting the buttons coordinates onto the user screen. However, this projection is confusing when two buttons are almost aligned on the X or Y axis, but different on the Z axis (that is when one

is close while the other is further). The result are clumsy expressions *"the button on the right"* immediately followed by *"the button on the left"*. It would be necessary to consider close/far properties to avoid this problem.

We also observed the following situations that are not bugs *per se* but are related to the choice of the strategy itself:

- Alarm warnings may be useful but they also pop up at any moment in the NA system. As an unexpected result, they could appear right in the middle of the referring process, breaking then the anaphoric reference. For instance, *"Push a blue button"*, *"Warning! There is an alarm around!"*, *"Yeah! This one!"*. One way to deal with this problem would be to really model reference to every object including alarms instead of hard-coding some of the instructions. The result would then be *"Yeah! This yellow button!"* instead of the one-anaphora (see (Denis, 2010)).

- One of the most problematic feature of the referring process of the NA system is the use of indefinite as a separated instruction. The first step of the reference process does not make use of the focus, and thus allows to produce indefinite expressions such as *"Push a blue button"*. The main problem is the presupposition that any blue button can satisfy the expectation. This is not problematic in most cases given that the next designation instructions remove this presupposition by making use of the focus. However, we observed many problems when the player acts before the designation instruction has been uttered, that is by pushing any blue button. This indeed can happen because the first push instruction is a *mandatory* message and as such stays on the screen for few seconds. The solution to this problem would be to collapse the first push instruction and the next designation one, for instance by uttering in a single utterance *"Push a blue button. Yeah! This one!"*. This would then prevent any action based on a wrong presupposition.

- Relying massively on the focus is sometimes not the good strategy. This is the case when the room has a particular shape: there is a part of the room filled with blue buttons, while the expected referent is in another part of the room not directly visible. If after receiving *"Push a blue button"*, the player stands in the part of the room with many blue buttons, he may spin around receiving continuously *"Not these ones! Look for another one!"*. Indeed, the instruction is correct. It is hard to figure why the player does not explore the room looking for another blue button. We think that it may be caused by context divergence: while the system is in the context of the whole room, the player is in the context of the part of the room filled with blue buttons. He may then assume wrongly that the system makes a mistake somewhere, especially if the system demonstrated a faulty behavior previously. To solve this, we may switch to another granularity if we observe too much time between instruction uttering and execution, for instance by adopting the same strategy than the NM system or by raising a move expectation before the push expectation.

5.2.2 NM system

As for the NM system, we observed the following bugs:

- The system gets mute.

- The system says *"Go to the left"* and there is a wall that blocks this direction.

- The system says *"Go to the 9th room on your right"* and there is no such room. The calculation of the the relative position of the target room with respect to the player is bugged. This is particularly problematic in world 3, where the number of rooms and their positions make the navigation very difficult for the user given the system strategy.

These remarks apply to the strategy itself:

- The grid setup of world 2 is confusing and causes many cancellations or wrong buttons pushing. Definite referring expressions are generated frequently but only shown for such a short time that it is impossible to read them, e.g. *"Push the blue button above the green button and on the left of the green button"* gets immediately replaced by *"Push a blue button"* when there are any visible blue buttons and only one of them is the intended referent.

- We observed contradicting directions when the room has a non-rectangular shape, e.g. *"Go on the right behind you"* [Player turns 180 degrees] *"Go to the left behind you"*. This can be avoided by taking into account a lower granularity of the goal regions, like in the NA system (cf section 3.2.1).

- In world 3, the system is replanning every time the user enters a wrong room. The consequence is that almost the only message actually displayed to a player moving too fast is *"Wait a sec, I'm checking the map"*. This

problem can be avoided like in NA by raising a wait expectation before replanning (cf section 3.4).

- The directions switched too quickly as the user was moving. The NM strategy relies too much in the relative position of the player to let him move around freely. A possible solution to this problem is to enjoin the player to stop moving, for instance by raising a wait expectation.

5.3 Impact on the subjective metrics

In this section we analyze how the strategies of the two systems impact on the subjective metrics collected during the challenge (Koller et al., 2010). Figure 9 shows the subjective metrics for our systems (notice that a higher number means a better performance, no matter whether the question is positive or negative). In general, the observed subjective metrics correlate with the objective metrics, that is, NA system does better in general than NM system, across worlds.

We took three of these subjective metrics, namely Q0, Q5 and Q6 and we compared NM and NA behavior taking into account the world. Q0 metric evaluates the overall instruction quality of each system. We consider that Q5 metric evaluates the quality of the reference strategy and Q6 the quality of the navigation strategy. Figure 10 shows Q0 metric for each system and each world. By comparing Figures, 8, 7 and 10 we observe that the subjective metric of overall evaluation is correlated with the task success, and then heavily dependent on the world of interaction.

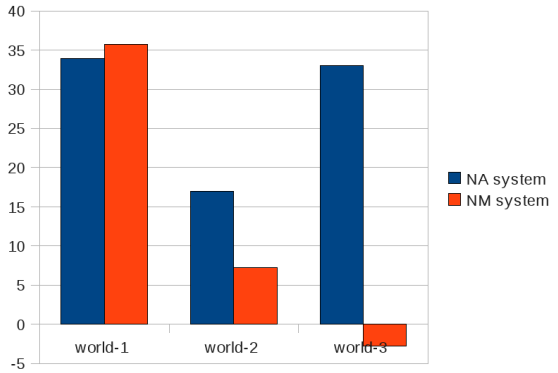


Figure 10: Subjective *overall evaluation* per world and per system

This made us wonder whether there is a subjective metric in which NM system does considerably better than NA system for a given world configuration, and we found that this is the case for Q5 and world 1. Figure 11 shows Q5 metric for each system and each world.

This result nicely illustrates that the NM navigation strategy is well suited for worlds with a small

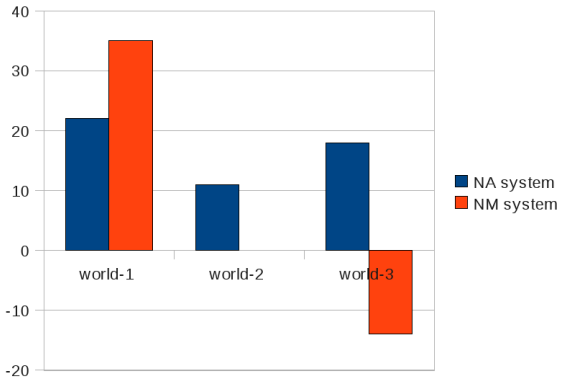


Figure 11: Subjective *Direction clarity* (Q5) measure per world and per system

number of rooms which are distinguishable by definite descriptions, resulting in instructions such as *"Return to the room with the lamp"*. In worlds with a larger number of rooms, like the worlds 2 and 3, the NM strategy is indeed confusing as proved by Q4 and Q5. The freedom given to the player makes him complain about the little help he received from NM as shown by Q9. In such worlds, the NA strategy being more directive leads to a better subjective assessment on those metrics.

On the level of reference (Q6), the NA strategy, that refers to objects by relying on ellipsis and focus, gives better results than the NM descriptive strategy, and this is stable across worlds. This is coherent with the objective metrics in Table 1 where NA is more successful and relies on a smaller number of instructions. However, even if players evaluated positively the feedbacks given by NM (Q3), they complain about the amount of unnecessary information as proved by Q7, maybe because knowing the function of the objects provided by NM is not a crucial information for the execution of the task. This unnecessary information may also be at hand in question Q8 where the players consider to have received too much information at once.

Interestingly, we observed that although NA and NM share the same messaging system the scores related with timing are different. On the one hand, they receive almost the same score about the "too late" assessment (in Q10). But on the other hand, NA performs better than NM about the "too early" assessment (in Q11). This difference may be explained by the NM strategy that refers to objects that are not yet known to the player such as *"Search the red room with* the safe*"* or *"To open* the door*, push the green button behind you"*.

Like all the participating systems, NA and NM do not have a very good score at the emotional level (questions Q16 to Q22). This may be caused by the task itself which is very repetitive. However,

Subjective Metric	NA	NM
Q0: Overall evaluation of the system	36	18
Q1: The system used words and phrases that were easy to understand	62	54
Q2: I had to re-read instructions to understand what I needed to do	40	8
Q3: The system gave me useful feedback about my progress	9	11
Q4: I was confused about what to do next	29	9
Q5: I was confused about which direction to go in	21	8
Q6: I had no difficulty with identifying the objects the system described for me	18	13
Q7: The system gave me a lot of unnecessary information	15	10
Q8: The system gave me too much information all at once	31	8
Q9: The system immediately offered help when I was in trouble	32	4
Q10: The system sent instructions too late	38	39
Q11: The system's instructions were delivered too early	39	12
Q12: The system's instructions were visible long enough for me to read them	6	-14
Q13: The system's instructions were clearly worded	32	23
Q14: The system's instructions sounded robotic	-4	-2
Q15: The system's instructions were repetitive	-31	-28
Q16: I really wanted to find that trophy	-11	-8
Q17: I lost track of time while solving the overall task	-16	-18
Q18: I enjoyed solving the overall task	-8	-4
Q19: Interacting with the system was really annoying	8	-2
Q20: I would recommend this game to a friend	-30	-25
Q21: The system was very friendly	30	20
Q22: I felt I could trust the system's instructions	37	24

Figure 9: Average subjective metrics for both systems across worlds

as shown by Q18, the NM system that gives more navigational freedom to the player is considered more enjoyable than the NA system. We assume that the very directive navigational strategy of NA is much less entertaining than letting the player make his own choices like NM, even if giving him more freedom leads to more confusion as shown by Q4 and Q5.

6 Conclusion

We presented the evaluation of two instruction giving systems in the GIVE-2 challenge 2010. The first strategy (NM) aimed at providing high-level instructions, describing rooms for navigation (*"Search the room with a lamp"*) and providing goal description (*"To open the door, ..."*). The second strategy (NA) used lower level instructions, such as a directive navigation strategy (*"Go through the doorway to your left"*) and a referring strategy mostly based on ellipsis and focus (*"Not these ones! Look for the other one"*). The results show that the NA strategy is more successful than the NM strategy. The high-level instructions strategy of NM proves to be confusing and cognitively overwhelming. On the other hand, players favor the kind of feedbacks provided by NM strategy and found it more enjoyable and less robotic than the NA strategy.

The data collected in the GIVE-2 experiment contributes to the identification of weak points and strengths of our systems' strategies and implementations. But also it provides a detailed view of different situations in which one strategy performs better (or worse) than the other. In the future we will focus on investigating how to integrate or combine the different strategies into a system which takes advantage of the strengths and can dynamically adapt different strategies to different situations. For instance, one possible way to combine both strategies is to start from the NM strategy, at a high level of description and giving more freedom to the instruction follower, and in case of execution problems to switch dynamically to a lower level. However, the concrete details of this combination have yet to be worked out. An important thing to consider is the goal of an instruction generation system. Do we want to make it as faster as possible? enjoyable? or to have some learning effect? or all of them?

References

Jörg Baus, Antonio Krüger, and Wolfgang Wahlster. 2002. A resource-adaptive mobile navigation system. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 15–22, New York, NY, USA. ACM.

Donna K. Byron, Alexander Koller, Jon Oberlander, Laura Stoia, and Kristina Striegnitz. 2007. Generating instructions in virtual environments (GIVE): A challenge and an evaluation testbed for NLG. In *Proceedings of the Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*, Washington, DC.

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.

C. Callaway, M. Dzikovska, C. Matheson, J. Moore, and C. Zinn. 2006. Using dialogue to learn math in the LeActiveMath project. In *Proceedings of the ECAI 2006 Workshop on Language-Enabled Educational Technology*.

Robert Dale and Nicholas J. Haddock. 1991. Generating referring expressions involving relations. In *Proceedings of the 5th Conference of the European Chapter of the ACL, EACL-91*.

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

Daniel Dionne, Salvador de la Puente, Carlos León, Pablo Gervás, and Raquel Hervás. 2009. A model for human readable instruction generation using level-based discourse planning and dynamic inference of attributes. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 66–73, Athens, Greece, March. Association for Computational Linguistics.

Mary Ellen Foster, Manuel Giuliani, Amy Isard, Colin Matheson, Jon Oberlander, and Alois Knoll. 2009. Evaluating description and reference strategies in a cooperative human-robot dialogue system. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, California.

Alexander Koller, Kristina Striegnitz, Donna Byron, Justine Cassell, Robert Dale, Sara Dalzel-Job, Johanna Moore, and Jon Oberlander. 2009. Validating the web-based evaluation of NLG systems. In *Proceedings of ACL-IJCNLP 2009 (Short Papers)*, Singapore.

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.

Roan Boer Rookhuiszen and Mariët Theune. 2009. Generating instructions in a 3d game environment: Efficiency or entertainment? In *Intelligent Technologies for Interactive Entertainment*, volume 9 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 32–43. Springer Berlin Heidelberg.

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.

Kristina Striegnitz and Filip Majda. 2009. Landmarks in navigation instructions for a virtual environment. In *Proceedings of the Workshop on the First NLG Challenge on Generating Instructions in Virtual Environments (GIVE-1)*, Athens, Greece.