The TCD system for GIVE-2

Dermot Hayes McCoy
Trinity College Dublin
Dublin
hayesmcd@tcd.ie

Ielka van der Sluis
Trinity College Dublin
Dublin
vdsluis@scss.tcd.ie

Saturnino Luz
Trinity College Dublin
Dublin
luzs@scss.tcd.ie

Abstract

This paper describes the aims, design and architecture of the Natural Language Generation system with which we enroled in the GIVE-2 challenge. We present and discuss the results of our system as reported in (Koller et al., 2010).

1 Introduction

In this paper a Natural Language Generation (NLG) system is described and evaluated, which was developed for the Generating Instructions in Virtual Environments (GIVE) challenge. The challenge invites researchers to submit a system that improves the baseline system provided. The task in this challenge is to guide a user through a virtual environment and achieve a goal through the use of efficient natural language instructions. The user's goal in this case is to obtain a trophy hidden in the environment; the user 'wins' by obtaining this trophy within a given time limit and loses by either setting off an alarm, failing to obtain the trophy within the time limit or by simply giving up. The user needs to find and press a series of buttons in a strict order to win the game. Pressing a button in the wrong order may result in the sequence being reset or the alarm being triggered. Furthermore, moving to certain areas within the environment outside of the correct sequence may also result in an alarm. These tight constraints mean that acting in a random fashion, or paying an insufficient degree of attention to the instructions provided will almost always result in failure for the user. Notwithstanding this, the onus is on the system to provide clear, precise and efficient instructions to the user so as to enable them to complete the task successfully.

The system detailed here uses as a starting point the "proof-of-concept" example system developed by (Koller et al., 2009) and builds upon it. In particular it was aimed to: (1) reduce the number of system instructions; (2) increase the system's clarity; and (3) bring about more natural sounding and less repetitive system output. The following sections provide a structural description of the system, a presentation of the results of its entry in the GIVE challenge and a comparison of the results with the other systems participating in the GIVE challenge. Finally, a discussion of these results is provided along with potential future improvements.

2 Baseline Architecture Overview

The pre-existing GIVE-2 system used for our system provides the entire means of creating the environment, displaying this environment to the user, enabling the user to interact with the environment and displaying instructions to the user, therefore their details are not discussed here (but see (Koller et al., 2009)). A planner built into the GIVE system is utilised to obtain the correct series of actions the user must undertake based on their current status and the current status of the environment. Thus, the description of our system's architecture deals only with the task of calling and utilising the planner correctly and transforming its output into natural language instructions for the user. The sole output of our system consists of textual instructions to the user. These are overlaid onto the interface of the user by the GIVE system. Five different "callback" methods determine the system's flow of execution. These are called automatically by the GIVE system in certain circumstances. A particular callback method is chosen and executed based on both input from the GIVE environment and the user's actions. A UML 2.0 activity diagram representing this program flow is provided in Figure 1. Here rounded oblongs represent activities the system undertakes, diamonds represent choice points where flow can potentially diverge and notched rectangles represent external signals into the system (i.e. the callback methods).

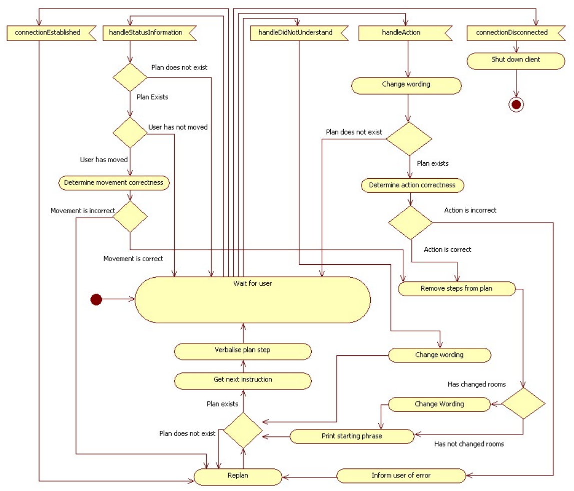

Figure 1: Activity diagram of the program flow.

2.1 Atoms

An Atom object details all necessary information concerning a single action in the GIVE world. Actions undertaken by the user can be represented by a single Atom and the series of instructions generated by the planner can be represented by a series of Atoms. In this way the user's actions can be directly compared to the plan to determine their correctness. The first part of an Atom object is the predicate which describes the type of action to undertake. There are three possible predicates: move, manipulate and take. The first concerns the user moving from one position to another, the second concerns the user pressing a button or otherwise interacting with an object in the world and the third concerns the user taking the trophy and completing the challenge. The rest of the Atom describes further information on the particular action, such as a direction (for move actions) or a button identifier (for manipulate actions).

2.2 Callback Methods

The *connectionEstablished* and *connectionDisconnected* methods are called automatically upon connection or disconnection of the GIVE client to the NLG system respectively. The *handleAction* method is called whenever the user manipulates (clicks with the mouse) an object within the GIVE environment and the *handleDidNotUnderstand* method is called whenever the user presses a specific key indicating they do not know how to proceed. Furthermore the *handleStatusInformation* method is called automatically five times per second regardless of the user's actions. The *connectionEstablished* and *connectionDisconnected* callback methods are relatively straightforward. After establishing a connection the method initialises all necessary starting elements, such as a referring expression generator (REG) and the planner. It then calls upon the planner to create a new plan based on the user and environment information, instructs the system to transform the first element of the plan to natural language and presents it back to the user. Disconnecting involves simply calling upon the client to shut down. The *handleAction* callback method is only called when the user manipulates an object within the world. Upon being called, it first must determine whether the action undertaken was the correct one or not. The action in the user environment is represented by an Atom object and the plan itself consists simply of a list of Atom objects. Therefore the user action can be directly compared to the next action in the plan in order to determine its correctness. This comparison is done both here and in the *handleStatusInformation* callback method. The latter performs similarly, but in this case is mainly concerned with whether the position of the user within the world is correct. Each of these methods results in one of two possible scenarios: either the user is in error, or what they have done up to this point is correct. If in error the user is informed of this fact and the planner is called upon to make a new plan, if not then the user can continue as normal, receiving the next instruction if appropriate. The final callback method is the *handleDidNotUnderstand* method. The user can call this method by pressing a specific key if they ever feel lost or confused as to what to do. In this environment, instructions to the user are relatively straightforward. Calling upon this method rewords the instruction provided to the user, randomly replacing the non-essential elements with those from a list of suitable alternatives whilst keeping the same content. It is hoped through these means to remove any misunderstandings or ambiguities in language whilst maintaining the functional meaning of the instruction.

2.3 Referring Expression Generator

In the baseline system a referring expression generator (REG) is used to denote objects in the world based on their characteristics and spatial relationship to other nearby objects. These objects include both manipulable types and non-manipulable types. The manipulable types in the GIVE environment currently exist only as buttons, which can be manipulated (clicked on) and may affect the world in some manner. The non-manipulable objects can exist in a number of forms, such as chairs, plants, etc. They serve no purpose except for use as references to the user. The referring expression generator takes in as inputs the object in question as well as a list of all other visible objects. It then returns a descriptive phrase for the object which can be combined with others as part of the text provided to the user. For instance instead of simply instructing the user to 'Go to the button' it might say 'Go to the blue button by the lamp'. The referring expression generator used here is the same as that of the baseline system with only minor modifications.

3 Design Aims

Our system is intended to improve upon the baseline system in the following ways:

(1) Reduce the number of System Instructions
The original system provides very low level commands instructing the user in what is sometimes an overly simplistic step-by-step fashion. For instance, it might give a series of five move instructions in a row to simply cross a room to a doorway on the other side. We felt that this was unnecessary and could be replaced with a single instruction if that doorway is visible. A huge number of instructions given in a short space of time can be confusing and difficult to read for the user. Equally, providing a low number of very complex instructions can also engender confusion, therefore an ideal middle ground needs to be found. The issue lies partly in the structure of the GIVE environment and the raw instructions received by the planner. The world itself is split into many small sections, often many within the same room; therefore the planner lists them all individually as locations that need to be moved to in sequence. To decrease the number of instructions given to the user our system culls many of these, presenting only those seen as vital to the user, namely those telling the user to leave the room and those telling the user to move to or press a button. It is hoped that this method will reduce the number of distractions to the user whilst still giving them enough information to complete the task.

(2) Increase the System's Clarity
A further problem that comes about in the original system lies in the manner in which it reports user errors. Every time the user does not follow an instruction correctly the system informs him of this fact and instructs him to remain still while a new plan is created. However, these errors are nearly impossible to avoid due to small size of the room sections, as described above. Very minor deviations from the assigned path result in an error message yet the system does not give its instructions in an exact enough manner to avoid this occurrence. Whilst it is certainly possible to provide instructions in terms of exact course and distance this would be overly formal and is not generally used in common parlance. Furthermore, such stringent adherence to the plan on the part of the user may not be necessary to achieve the goal, which is why many instructions were removed as part of Aim 1 above. Our system of handling user errors allows the user greater freedom. The solution arrived at is for the system to act in much the same manner as before when a user does not follow the plan (i.e. make a new plan, taking into account the user's new position and instructing them from there) and yet not to inform them that they are in error, making the replan occur seamlessly and invisibly. Thus if a user moves to a section of a room, or even to another room altogether that is not listed in the plan they are not told that they are in error, but simply instructed on how to proceed from their new location. It is of course possible for the user to fail the task based on their movements (i.e. if they step on an alarm tile), but as this immediately ends the game an error message would be superfluous.

It should be noted that this has only been changed for user movements, not for button pushing actions. Pushing an incorrect button can drastically upset the plan sequence and so the user is informed of this.

(3) Produce More Natural, Less Repetitive Output
Although the GIVE Challenge is a measure of how well a system can instruct the user to perform tasks it is also desirable to provide instructions in a manner that is as natural and human-like as possible. The original system suffered from a large degree of repetition and could only express instructions

in a single, standard form. If a human being was giving instructions a variety of sentence structures would be used. Furthermore, if they found that they were misunderstood they would reword their instruction in the hopes that the listener would understand if the content was presented in a different manner. It is hoped that by varying the sentence structure the user would perceive the system as friendlier, more natural and less robotic. To do this three types of phrases were identified, each of which could be modified yet still retain the same meaning: a starting phrase, a leaving phrase and a door phrase. For each type of phrase a pool of such phrases with identical meanings was developed. Each time a new phrase is needed a random one is chosen from the pool. The final instruction may combine elements from all three pools and in this way create a large degree of variety in the instructions given.

4 Implementation Details

The following is a description of the most important activities represented in the activity diagram in Figure 1

Change Wording
This activity changes the wording of the instructions sent to the user and relates clearly with Aim 3 above. Specifically it aims to maintain the core meaning of the instruction whilst changing the non essential 'fluff' surrounding it. There are three types of phrases that can be added: a starting phrase, a leaving phrase and a door phrase. A starting phrase is an introductory phrase used to start a command, a leaving phrase is used when instructing the user to exit an area and a door phrase is used when instructing them to find or move through a door. Initially a random phrase of each type is selected from a pool of phrases whose meaning should ideally be identical. For instance, a leaving phrase might be 'Now we need to leave this room' but another randomly chosen alternative could be 'We need to go to a different room to continue'. Starting phrases include expressions such as 'Brilliant' and 'On to the next step'. These are intended both as an acknowledgement of a completed action and as a lead in to the next instruction. Door phrases tie in with leaving phrases to describe an exit for the room. Examples include 'Go through the door.' and 'Exit through the doorway.' The Change Wording stage of the system randomises the chosen phrase of each type, giving new results. These new phrases will then be used every time a phrase of their type is requested until the "Change wording" activity is called upon again.

Print starting phrase
This activity prints the currently selected starting phrase out to the user. It is used as an acknowledgement of a correct action and as a lead in to the next instruction.

Replan
This creates a new plan using the GIVE planner. A plan consists of a a sequence of Atom objects, representing each individual movement or action to be undertaken by the user.

Get next instruction
This takes the next Atom instruction from the plan and sends it to the 'verbalise plan step' stage.

Verbalise plan step
Creates and outputs to the user an Atom object converted to an instruction in natural language. The system output created here is often a combination of a number of text strings appended to each other to form sentences. The operation first obtains the predicate of the plan step which tells the program what type of action is called for (e.g. 'move', 'manipulate' or 'take'). If the predicate is unrecognised then the step is simply returned in its raw form (i.e. unprocessed by the natural language generation system). The 'take' option is used solely at the end of the task when the user, being in the room that contains the trophy, can access the trophy. If 'take' is found to be the predicate then a single hard-coded utterance is used to instruct the user to take the trophy. The 'manipulate' option is used to tell the user to press a specific button ('the target object'). The target object, as well as the list of all other visible objects is passed to the referring expression generator. This returns a referring expression which is then included in the template-based instruction to the user.

The 'move' option deals with a variety of possible circumstances dependent on the values of the two attributes of the predicate 'move' that represent a 'from' and a 'to' location. There are two possible results here: either the values of these attributes are the same (in which case the user is being instructed to stay in the same room) or they are different (in which case the user is being instructed to leave the room). In both cases an instruction to the user is constructed from a series of concatenated strings.

If the user should leave the current room a natural language phrase is constructed from three sources. A string is obtained from the leaving phrase pool, another from the door phrase pool and also a direction string which indicates in which direction the user should move to reach the relevant location. A discretiser transforms the direction from an exact heading to a more natural expression. For example the phrase ‘slightly to the right’ is used instead of the ‘+12.6°’. The strings are combined and presented to the user as a single instruction. If the user should stay in the same room the plan step is removed and ignored unless it refers to an area directly adjacent to a button, as described in Aim 1. If the plan step does instruct the user to move to a region directly adjacent to a button then our system produces a natural language instruction consisting of three parts. Firstly, if the user is already beside a button but it is not the correct one listed in the plan then a string is added informing them of this. A second part of the response is created from a referring expression that describes the location of the button in relation to other nearby objects. This is again constructed from the referring expression generator based on the objects currently visible and near to the user. The final part of the response gives a direction to turn based on the user’s current direction using the discretised form as above. Again, all elements are concatenated and presented as a single instruction.

Determine Action Correctness
This activity deals solely with “manipulate” actions. As the output of the planner consists of a series of Atoms representing individual actions and movements it is possible to directly compare the user action with the next action listed in the plan. If the Atom representing the recently undertaken user action is identical to the next Atom in the plan then the action is deemed to have been correct. If they are not identical the action is incorrect.

Determine Movement Correctness
Determining whether user movement is correct is similar to determining action correctness in that it compares the provided user movement Atom to the Atoms in the plan. The difference in this case is that the user may have performed a number of movements in the time between status updates or may have taken a slightly different route to that in the plan, as per Aim 2. The system therefore steps through the plan from the start until it gets to the point in the plan that instructs the user to move to their current location. The number of steps taken is recorded and indicates the number of correct actions that have been undertaken. If the user’s current location is not in the plan then a zero is recorded to indicate they have performed an incorrect movement. If any other number is recorded the movement is deemed to have been correct.

Remove Steps from Plan
This activity removes one or more elements from the start of the plan. Current user actions are always compared to the first action in the plan; therefore as actions are completed correctly they must be removed from the plan to enable the next action to be examined. If a manipulate action has been undertaken then a single Atom is removed. If, on the other hand, a movement has been undertaken then a variable number may be removed as found in the ‘Determine Movement Correctness’ stage (i.e. the number of correct steps taken).

5 Results

Results from the GIVE challenge are based on data obtained from 154 users trying to solve the task in three virtual worlds (i.e. World 1, World 2 and World 3, cf. (Koller et al., 2010)) that varied in difficulty. Results are divided into two sections: objective results and subjective results. There were overall seven teams entering the GIVE 2 challenge. All results here were obtained directly from the GIVE 2 report (cf. (Koller et al., 2010)). Objective results were obtained automatically by the GIVE system unobtrusively and without direct user input. They include categories such as task success rate and average time taken. It should be noted that, bar the initial category of task success, all objective results were based solely on five users who successfully achieved the goal. Subjective measurements, on the other hand, are taken from all users who participated in the task (154 subjects). They took the form of a questionnaire which appeared regardless of how the task was ended: whether by completion, failure or cancellation. The questionnaire invited the user to rate various aspects of the NLG system from a value of -100 to +100. Subjective questions were divided into two categories: quality assessments and emotional assessments. Quality assessments should theoretically have an overlap with the objective measurements whereas emotional assessments should cover less directly measurable criteria, such as the friendliness of the instructions for example. There also existed a single, ideally all-encompassing, question: ‘Overall, the system gave me good directions’, which is intended to cover both areas.

5.1 Objective Results

In terms of direct task success the results from our system were poor. The proportion of users completing a task using our NLG system was 3%, compared to an average of 22%, indicating 7th place for that category. However, in most other objective measures our system’s results were highly favourable, producing a 1st place result in four separate areas: those of duration, distance, number of actions and number of instructions. In terms of number of words per instruction however, ours again came in 7th.

5.2 Subjective Results

The subjective questionnaire results showed relative average performance compared to the other participants. Overall though, users reported generally negative feedback for most NLG systems. The summed total of all quality-related questions resulted in a score of +373 for the best performing system, compared with -183 for the worst performing. Considering the maximum limits of 1500 in this category, however, these results are not as divergent as they seem. In subjective quality our system obtained a summed result of -44, attaining 5th place. In particular the results showed that our instructions were quite clear and understandable (Q1, Q2, Q13), and were delivered at an adequate pace (Q8) with few timing issues (Q10, Q11). Shortcomings were brought to light in the ability of the system to adequately describe the environment (Q4, Q5, Q6), as well as recognising when prompts were necessary (Q3, Q9, Q12). All systems had issues with robotic sounding dialogue, although ours was better than most, coming in second place. In terms of emotional assessment most systems were again found lacking, with only one achieving a positive result (+20). Our system obtained a result of -88, placing it 4th in this category. Overall, however, users found most systems unenjoyable (Q16, Q18, Q20) as well as annoying (Q19). Furthermore, users found that they rarely lost track of time while playing our system (Q17) and proclaimed a lack of trust in our system’s directions (Q22). On a more positive note however they did find our system friendly (Q21).

6 Discussion

6.1 Success Rate and Difficulties

To a large degree most of the failings with our NLG system were shown in the objective, automatically obtained results. An overall success rate of 3% is remarkably low for such a task and is further brought to light in the 0% success rate that users achieved in World 2 and World 3. To a large degree our system was not suited for the environments presented in the challenge. Whereas our system worked admirably in the sample environment presented to potential contestants many aspects which had not been a major focus in our development were strenuously tested in the competition environments. A primary concept was the need for an extremely precise referring expression generator (REG). Many locations in the world required precise definition of objects within complex spatial relationships, such as in Figure 2 below (a green button surrounded by blue ones). Whereas our system could describe objects in terms of other, nearby ones, it could not do so to the level of detail necessary here. Other problems arose due to more simplistic issues, such as our referring expression generator being unable to express the colour yellow for example. This challenge highlighted the necessity of creating a robust referring expression generator, capable of dealing with complex scenarios. In contrast our REG had only minor modifications from the baseline as it was not a major focus in our development plans.

Similarly to the issues involving the REG, the new environments presented a number of challenges designed to test the limits of the NLG systems. Numerous issues arose which resulted in highly confusing output to the user. For instance, rooms in the aforementioned sample environment were large and clearly laid out. In the final challenge environments, however, such distinctions are not so clear cut. Doorways in the GIVE en-

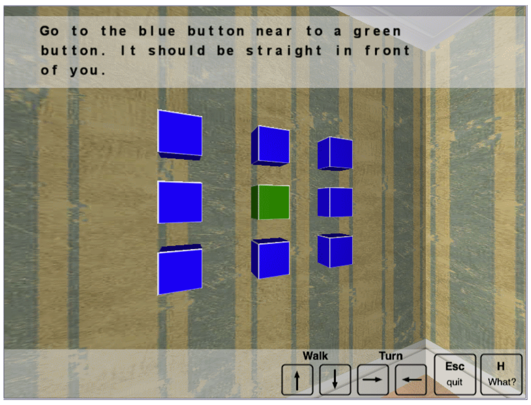
Go to the blue button near to a green button. It should be straight in front of you.

Figure 2: Example situation to illustrate the need for highly precise referring expressions.

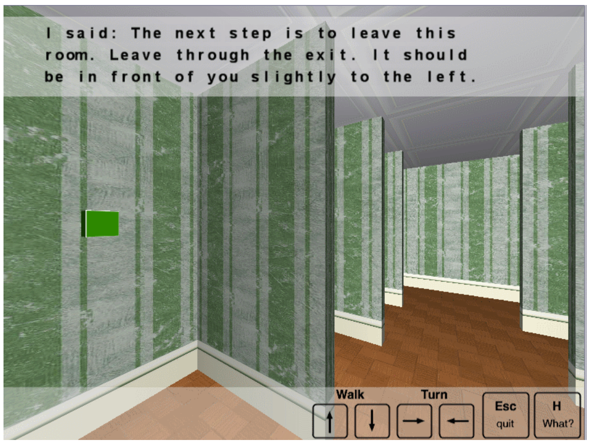
I said: The next step is to leave this room. Leave through the exit. It should be in front of you slightly to the left.

Figure 3: Example situation to illustrate likely discrepancies in the definition of a room.

vironment are simply defined as gaps between rooms. In World 3 however, often extremely small rooms are placed side by side. This results in the system referring to rooms and doorways which, although represented internally in the GIVE environment as such, do not have such appearance to the user when traversing the environment. This is shown clearly in Figure 3, which is an in-game view of the same World 3. Here the planner is attempting to lead the player to the sole button in view, yet each alcove is represented internally as a separate room, and the gaps between as doorways. We see quite clearly the problems a typical user would encounter, in that to the user both (s)he and the button may be in the same room, yet to the GIVE environment they are in entirely separate locations.

Similar environmental problems come about with regards to alarm tiles. If a user steps on an alarm tile the game is immediately ended with a loss, yet the user is never informed in advance that this will happen. They are given a brief tutorial on how to move around and interact with the environment yet must rely on the NLG system to guide them around alarm tiles. This is never a problem in World 1, as no correct solution would lead them over an alarm. In the competition environments this can often be the case though. Such an environment is pictured in Figure 4. In this scenario the actual route chosen by the GIVE planner instructs them to move in a direction which would trigger an alarm. It is difficult for the NLG system to avoid this event; in order to do so it would have to plot a series of small movements around every alarm, which would be quite difficult to achieve for an unknown environment. A much simpler solution is simply to tell the user not to step on any red alarm tiles at the very beginning of the challenge, in which case they can use their own skill and intuition to avoid triggering them. Unfortunately in our system such a measure was not implemented. Indeed it is not entirely clear whether it is the duty of the NLG system or the in-built GIVE tutorial to provide such knowledge. However, even with this knowledge and sufficient instructions the user may be unable to avoid setting off an alarm; some environments such as World 3 provide a maze of alarm tiles which must be avoided solely through the skill of the user.

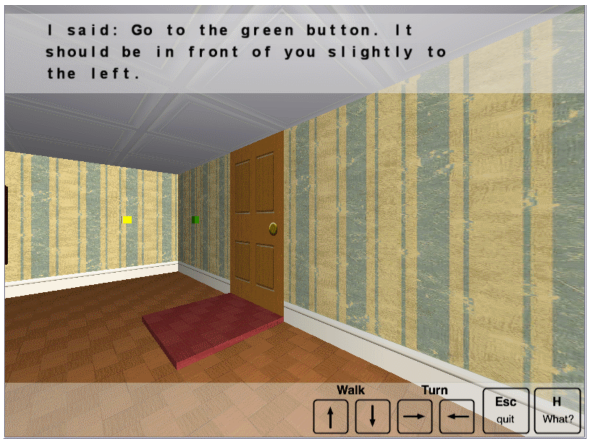
I said: Go to the green button. It should be in front of you slightly to the left.

Figure 4: Example situation to illustrate difficulties with alarm tiles.

6.2 Our Aims

In Design Aim 1 we made the decision to remove many of the steps that the planner created to achieve the goal. The rationale was that a large amount of them were unnecessary and only served to confuse the user. Again, in the sample environment this proved to be the case but the final competition environments were much more complex.

Removal of many of the low level instructions may have even resulted in worse performance than the original system.

In Design Aim 2 a decision was made to remove the error messages concerning the user's movement. This was done, again, for the reason that it may be confusing to the user to report error messages for very minor discrepancies between the user's actions and those listed in the plan. Such error messages were simply suppressed yet the system continued to replan as normal. This solution worked well for World 1, where the layout, and therefore plan, were relatively simple. The complexity of the competition environments however again proves to be a difficulty, as the in-built GIVE planner can not keep up with the large number of highly complex replans needed. This then causes a delay in the user instructions while the planner and the NLG system struggle to keep up. Often an instruction shows on screen that is clearly old and no longer relevant to the user, presumably causing much confusion. Some of the comments left by users support this view. For instance, one states that the

'System seemed to update way too slow'

and that

'I was never sure whether the information it gave me were meant for my current position or for an earlier one'

Another user said that

'Acting according to instructions just isn't possible when you never know if those instructions correspond to your situation, or the situation you've been in some time ago'

The original error message instructed the user not to move until the replan was complete. This solution, while resulting in a slower and much less fluid user experience, would probably bring about a greater success rate. It should be said though that the speed of the planner is affected by the computational speed of the user's hardware, therefore some users may have had an exacerbated or lessened effect than that described here.

Whilst it must be said that the success rate of users operating under our system was quite poor, those who did succeed performed, on average, far better than with any other NLG system in the competition. In terms of time taken, distance travelled, number of actions taken and number of instructions given our system excelled. Clearly those who could operate without low-level prompting for every single step of a task found the less obtrusive system created through Design Aims 1 and 2 superior to the alternatives. It is felt that this approach would be highly suited to a series of more natural environments, rather than those specifically based on complexity.

6.3 Subjective Measurements

The users' subjective results also proved to be reasonably positive. The main negative feedback was found to be with the system's descriptive ability and with its ability to deliver prompts when necessary. These problems fit quite closely with both the referring expression generator issues and the planner issues as described above. Taking those issues out of the account the analysis is much more positive. Users found the system to be clear, concise, easy to understand and quite human-like in its speech. This is interesting when one views the statistics showing that our system produces the longest instructions: clearly this is not an issue if the content within remains of high enough quality. Some users did apparently have difficulty reading all the text before it disappeared though. One user states:

'I liked the game but I'm a German pupil and it was really hard to read as fast as the system told me'

The length of time the instructions remain on screen therefore may need to be increased. Altogether though these results fit quite well with our goal stated in Design Aim 3. Unfortunately, despite our efforts to vary content users still found it to be quite repetitive; however this was a universal problem across all systems and was shown to be much less prevalent in our system than most others. All systems also received negative remarks concerning the emotional effect of the challenge. As stated in (Koller et al., 2010) though this may be more due to the structure of the task rather than the NLG system involved. In particular our system got bad marks for trustworthiness, which presumably follows directly from its low success rate. It also got low marks for its ability to make users lose track of time, which is again most likely due to the low success rate and its resulting short average play times. Users did find the system friendly however, which could be a positive indication for the choice of language used.

Overall the system did not perform as well as was hoped, although this mainly in the area of direct task success, which is only one of many evaluation areas. Other scoring measures proved much more positive, indicating in particular that the three design aims do show a degree of promise. If the issues mentioned in this section are addressed and the system better tailored to the type of environments it is likely to achieve then it could achieve far better results in future competitions.

Acknowledgments

This research is supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Trinity College Dublin.

References

A. Koller, K. Striegnitz, D. Byron, J. Cassell, R. Dale, S. Dalzel-Job, J. Moore, and J. Oberlander. 2009. Validating the web-based evaluation of nlg systems. In *Proceedings of ACL-IJCNLP 2009 (Short Papers)*.

A. Koller, K. Striegnitz, A. Gargett, D. Byron, J. Cassell, R. Dale, J. Moore, and J. Oberlander. 2010. Validating the web-based evaluation of nlg systems. In *Proceedings of INLG 2010 (Generation Challenges'10)*.