Submitted to AIIDE 2010 by Philip Hanson and Charles Rich, WPI.

A Non-Modal Approach to Integrating Dialogue and Action

Abstract

We have developed and demonstrated an experimental authoring and run-time tool, called Disco for Games, that supports the creation of games in which dialogue and action are integrated without the need for changing modes. This tool is based on collaborative discourse theory, in which utterances are treated as actions, and has a number of additional benefits including better modeling of interruptions, automatic dialogue generation, plan recognition and automatic failure retry.

Introduction

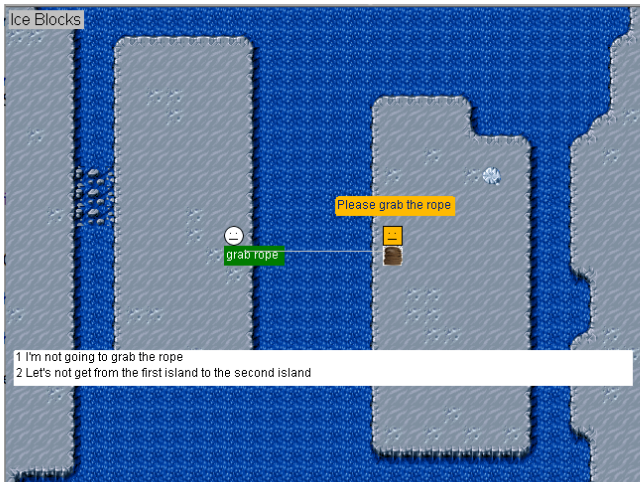
A problem with current computer games that include both dialogue and action is that dialogue and action are supported in two distinct user interface *modes*. Each mode has very different affordances and controls, and you usually need to explicitly switch between them. For example, dialogue mode typically provides a display of what the nonplayer character (NPC) is saying and menu of utterances from which the player can choose, while action mode typically provides affordances and controls for moving and manipulating objects in the environment, aiming and firing weapons, fighting, etc. This has been true for major 3D games right up to the present time, e.g., Deus Ex (2002), Elder Scrolls IV: Oblivion (2006) and Dragon Age: Origins (2009).[1]

This modal approach to dialogue and action is a problem because it is not a good reflection of natural human interaction in which talking and acting are usually seamlessly interwoven. It also encourages game design in which characters are divided into "talkers versus fighters," which tends to make both types of characters less rich than possible.

Furthermore, we believe that the artificial separation of dialogue and action into distinct user interface (UI) modes is a symptom of a deeper dichotomy between the technologies used to implement dialogue and action. In this work, we address both the UI issue and the underlying technology issue by applying a basic tenet of collaborative discourse theory (see more below), namely to *treat utterances as a type of action*. We have developed an experimental authoring and run-time tool, called Disco for Games (D4g),[2] based on this approach and have used it to build a simple 2D puzzle-adventure game with a non-modal interface. Using D4g to build games has other benefits, which we point out below as well.

[1]The only game genre that has completely avoided this problem is the niche of interactive fiction (text adventure) games, in which both dialogue and action are uniformly handled with text.

[2]This tool was originally called Tizona in (Author 2010).



1 I'm not going to grab the rope
2 Let's not get from the first island to the second island

Figure 1: Screen Shot of Ice Blocks Level

A Demonstration Game

Our demonstration game is titled *Secrets of the Rime* and involves primarily the player and a sidekick character, both researchers in Antarctica, who find themselves cut off from their research base and must take the long way back. Collaboration is an integral part of game play, with most puzzles requiring coordination of player and sidekick actions.

screen is composed of four levels, each containing a puzzle to be solved or task to be completed:

- *Ice Blocks*: cross a river with two islands (see Figure 1)
- *Ice Wall*: get past a wall that is too high to jump
- *Shelter*: build a shelter with materials at hand
- *Walrus Cave*: solve riddles posed by a Sphinx-like walrus

At the start of each level, the player and sidekick are located near the left edge of the map area. The next level and their ultimate goal, the research base, are always to the right. At the moment in the Ice Blocks level shown in Figure 1, the sidekick (square orange head[3]) has just crossed from the first to the second island, thrown back a rope to help the player (round white head) cross the water, and said "Please grab the rope" (shown in bubble above the sidekick's head).

[3]Please excuse the "programmer art."

```
[Achieve CrossRiver] -live
  Player says "We need to get to the other side."
  [Get from the near side to the first island by making a bridge] -done
    [...] -done
      Sidekick says "How do you want to get from the near side to the first island?"
    Player push an ice block into the water. -succeeded
    [Player walk to the first island] -succeeded
      Sidekick says "Please walk to the first island."
  [Get from the first island to the second island using the rope] -live
    [...] -done
      Sidekick says "How do you want to get from the first island to the second island?"
    Player says "It's too far for me to swim. There's a rope, though..."
    Sidekick says "Well, I can make it across. I'll send you a postcard from the other side."
    Sidekick swim from the first island to the second island.
    Sidekick throw the rope.
    [Player grab the rope] -live <-focus
      Sidekick says "Please grab the rope."
  [Get from the second island to the far side]
```

Figure 2: Automatically Generated Interaction History at Moment Shown in Figure 1

The player is *completely free at any time* to either move up, down, left or right using arrow keys or to choose from the utterances shown in the dialogue menu shown at the bottom of the screen using number keys. Finally, notice the green rectangle with "grab rope" that has just popped up as the player has approached the end of the rope. This indicates that if the player moves a bit further, he will perform this action. For details of the other levels, see (Author 2010).[4]

The interface to this game is non-modal and the underlying implementation, described in the following sections, integrates dialogue and action into a single representation.

Interaction History

To begin to understand the representation underlying D4g, please refer to Figure 2, which contains the *interaction history* automatically generated by D4g at the moment shown in Figure 1 (with a few distracting details replaced by ellipses). Interaction histories are an additional benefit of using D4g. They are a textual visualization of D4g's internal state representation, which show the past, present and expected future of the current interaction between the player and an NPC. They are an invaluable tool for debugging game designs and may also serve as an entertaining post-play review for players.

Underlining has been added to the history in Figure 2 to highlight the interleaving of utterances and actions. Notice that the player has pushed an ice block into the water between the near side and the first island creating an ice bridge (still visible in Figure 1), over which the player and sidekick walked. Then, after some conversation, the sidekick swam to the second island, threw the rope, and asked the player to grab it.

The bracketed lines in interaction histories represent *goals*. The indentation shows the recursive decomposition of goals into subgoals or primitives. Primitives are either utterances or actions. The status of each goal is indicated by flags: -done, -succeeded and -failed indicate completed goals; -live indicates goals that are ready to start or in progress; future goals that are not ready to start have no flag (such as the last line in the history).

Finally, notice the "focus" pointer on the third-to-last line in Figure 2, which indicates the goal that is currently at the top of D4g's *focus stack*. All of this goal's parents (goals that are above it and less indented in the history listing) are below this goal on the focus stack. The focus stack is an additional benefit of using D4g, because it allows characters to properly handle interruptions in conversations (see details in (Author 2010)).

[4]There are also videos of the first two levels available at http://www.cs.wpi.edu/∼rich/d4g/IceBlocks.avi and IceWall.avi.

Current Technologies

Almost all dialogue characters in games are currently programmed using dialogue trees (Despain 2008), which are an effective, if laborious, technique for giving the designer complete control over what the NPC says and the player's utterance choices at every step in a conversation. In order to reduce labor, different paths in a dialogue tree often join to the same point, and there can even be cycles—thus dialogue trees are technically directed graphs rather than trees. Nevertheless, dialogue trees are well-known to suffer from a "combinatorial explosion" of options as conversations get longer. We are not trying to fix this problem with dialogue trees, though it may be improved somewhat by D4g's automatic dialogue generation capabilities discussed below.

The technologies currently used to implement action characters are more varied. Ad hoc code, finite state machines and decision trees are common options; recently, behavior trees (Isla 2005) have become popular.

The problem with this current technical state of affairs is that if a developer wants to implement a character capable of both dialogue and action, that character needs in effect to be implemented twice: once with dialogue trees and once with an action technology. Furthermore, coordination between the states of the two subsystems typically entails an ad hoc collection of flags, hidden variables, etc., which is labor intensive and error prone. The result is that few integrated characters are created.

Disco for Games

Disco for Games (D4g) uses a single representation, namely hierarchical task networks (Erol, Hendler, and Nau 1994), which subsume both dialogue trees and a certain kinds of behavior trees.[5] In this representation, utterances are treated as just another type of action, so that utterances and actions can be freely interleaved at as fine a granularity as desired. Furthermore, because there is a single engine managing the entire interaction between the player and NPC, the problem of ad hoc flags and hidden variables to coordinate between dialogue and action is obviated.

Figure 3 shows the component architecture used to build our demonstration game with D4g. At design time, authors specify a *task model* using an XML formalism we have defined, Disco for Games Markup Language (D4gML), which is a macro extension of the ANSI/CEA-2018 task modeling standard (Rich 2009) (task modeling and D4gML are discussed further below). In addition, authors specify the usual other game content, such as graphics models, textures, audio files, etc.

At run time, D4g reads in the author's task model and uses it to manage all of the dialogue and action interaction between the player and an NPC. D4g is implemented on top of Disco, which is a successor to the Collagen (Rich, Sidner, and Lesh 2001) collaborative discourse manager. Finally, there is the game engine, which loads and uses the other design-time game content (see below for more detail on the connection between D4g and a game engine). In the case of *Secrets*, the game engine is Golden T (www.goldenstudios.or.id). All of the run-time components are implemented in Java and are open source.

Task Modeling

One of the basic principles of collaborative discourse theory (Lochbaum 1998), which leads to the utterance as action approach, is that the structure of a conversation, especially

[5]The exact definition of a behavior tree is still in flux.

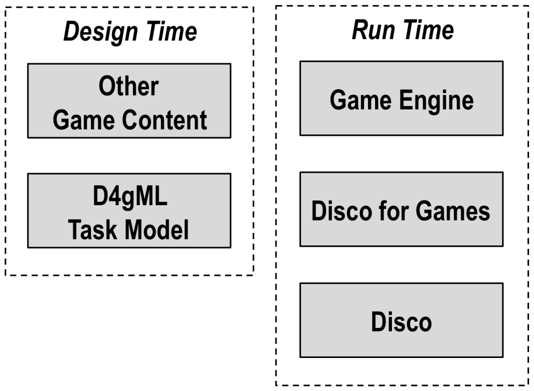


Figure 3: Component Architecture

one involving a collaborative activity, follows the underlying task structure. For example, Figure 4 shows the task model that controls the entire interaction in the Ice Blocks level of *Secrets* described above. This diagrammatic notation highlights the key features of a task model, but there are important additional details in the full ANSI/CEA-2018 specification, such as a JavaScript *applicability condition*, which can optionally be provided for any action, goal or decomposition. Goals and actions may also have a postcondition, which determines whether they succeeded or failed (see Game Engine API below).

Goals and actions in a task model diagram are shown as ovals; goal decompositions are shown as diamonds. Actions are distinguished from goals by not having any decompositions. Goals are connected to decompositions using dashed lines; the steps of a decompositions are shown using solid lines.[6] Steps are by default unordered; (partial) ordering constraints can be added using arrows. Restrictions on actions, such as whether only the NPC (sidekick) or player

[6]Figure 4 uses a common diagrammatic abbreviation in which if there is only a single possible decomposition for a given goal, such as CrossRiver, the diamond node is omitted.

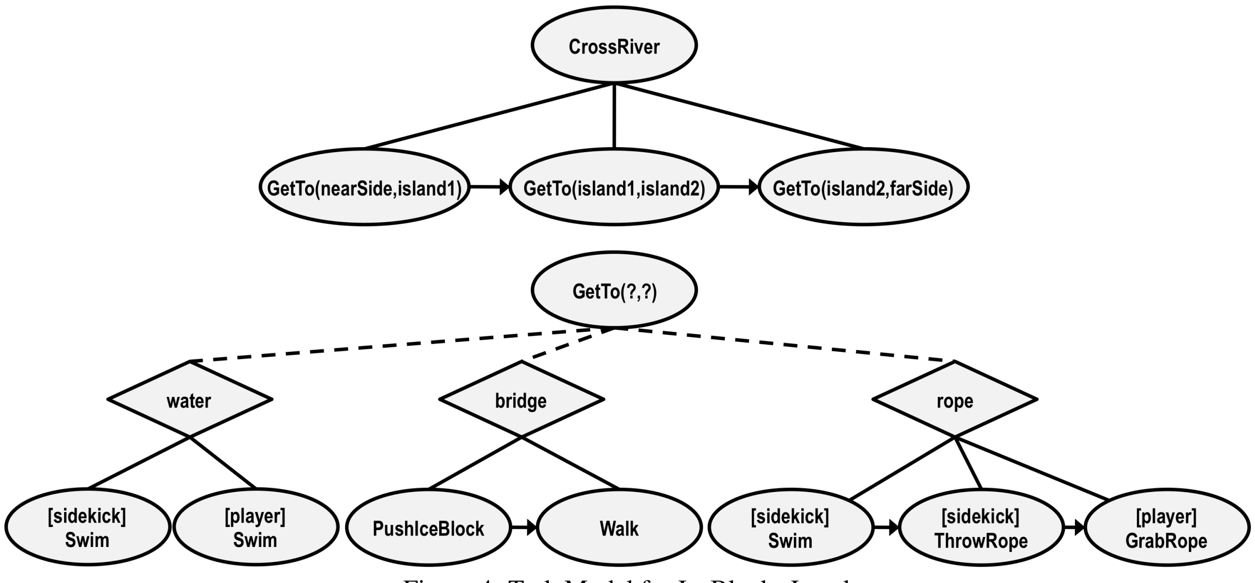
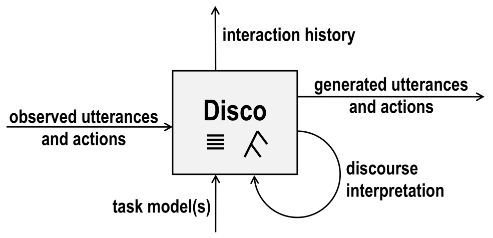


Figure 4: Task Model for IceBlocks Level


interaction history
generated utterances and actions
observed utterances and actions
Disco
discourse interpretation
task model(s)
Figure 5: Disco Collaborative Discourse Manager

may perform it, are indicated in square brackets. This formalism is sometimes called a hierarchical task network. The semantics of these models is similar to and/or trees, where diamonds are *and* nodes and ovals are *or* nodes.

Thus in Figure 4 there is one top-level goal, CrossRiver, which is always decomposed into three totally ordered steps (subgoals), each of which is an instance of a goal called GetTo, with its two parameters (*from* and *to* location) appropriately bound. GetTo has three alternative decompositions, corresponding to swimming, making a bridge, or using a rope. Notice that the two steps of the 'water' decomposition are unordered, which means that in different playings of the game, the player and sidekick can swim across in different order. This is a minor example of the kind of playing variation that task models can provide in general with very little effort.

Figure 5 summarizes how Disco manages the NPC-player interaction. The heart of Disco is the discourse state representation, based on collaborative discourse theory, which consists of a plan (task) tree and a focus stack. The main process in Disco is discourse interpretation, which updates the discourse state by explaining how the observed utterances and actions of the player and NPC relate to the loaded task model(s). For example, part of this interpretation involves updating the liveness flags discussed in Figure 2. The main output of Disco is to generate, for the player, a list of utterance menu options which make sense in terms of the current discourse state, and for the NPC, the single most appropriate utterance or action for it to perform next. As mentioned earlier, Disco can also produce interaction histories at any time for debugging or review.

Automatic Dialogue Generation

An observant reader may have noticed that there are no utterances in the task model for the Ice Blocks level (Figure 4), and yet there are many utterances by both the player and the sidekick in the corresponding interaction history (Figure 2). This is because one of the benefits of using D4g (that comes from Disco) is the automatic generation of dialogue based on the current discourse state. The automatic dialogue generation is based on a lightweight semantics for utterances (Sidner 1994) and a generic rule framework (Rich et al. 2002), neither of which space permits describing here. A simple and typical example will have to suffice.

The sidekick utterance in the middle of Figure 2, "How do you want to get from the first island to the second island?", was automatically generated by a rule triggered by the fact that the current discourse focus is a goal (GetTo) with multiple decompositions, none of which has yet been chosen or started. In addition to generating this sidekick utterance in this state, Disco also automatically generated the appropriate player menu at this point based on the applicable decomposition alternatives for the current goal, which in this case is only 'rope':

1 Let's get from the first island ... using the rope
2 Let's not get from the first island to the second island

The second menu choice above is generated from another generic rule that always gives the player the option to reject the current goal.

Notice that to avoid the internal node names from the task model appearing in the utterances above, a set of textual templates (using Java's formatted printing facilities) has been associated with goals, actions and decompositions in this model. Disco also supports an even greater level of customization, in which a whole phrase can be substituted for another phrase, which is how the more articulate "It's too far for me to swim. There's a rope though ..." has been substituted for option 1 above.

The fact that *all* the utterances in the Ice Blocks level are automatically generated from the simple task model in Figure 4 suggests a potentially powerful two-step development methodology: First debug the game design (task model) with the automatically generated dialogue—think of this as "programmer dialogue" by analogy with "programmer art." Then customize the final dialogue as much as desired.

Plan Recognition

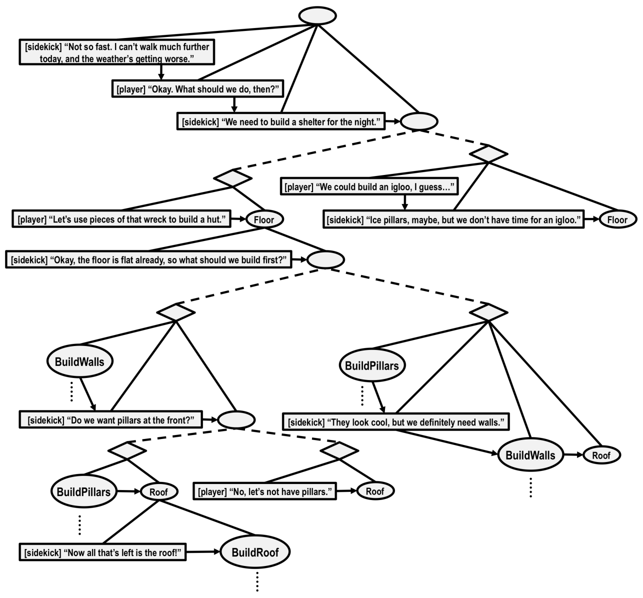
Before leaving this decomposition choice example, it is worth briefly pointing out another related extra benefit of using Disco, namely *plan recognition*. Plan recognition in general involves inferring implicit goals and decomposition choices from observed actions. When the player and sidekick were still on the near shore, the sidekick asked the first decomposition choice question in the history,"How do you want to get from the near side to the first island?", and the following player menu was generated (a different two decompositions were applicable):

1 Let's get from the near side ... by swimming
2 Let's get from the near side ... by making a bridge
3 Let's not get from the near side to the first island

However, instead of responding with an utterance on this occasion the player just *pushes* an ice block into the water, and as can be seen in the history, Disco correctly inferred the decomposition choice. This example again illustrates the advantage of a non-modal approach to integrating dialogue and action. For more details on the importance of plan recognition in collaborative dialogue, see (Lesh, Rich, and Sidner 1999).

D4g Markup Language

Of course, we do not expect all dialogue to be automatically generated. In fact, the third *Secrets* level, Shelter, starts with some traditionally authored dialogue between the player and sidekick. Figure 6 shows part of the task model for this level. Due to space restrictions, this figure is probably too small to easily read the text in each node (all of which appears



[sidekick] "Not so fast. I can't walk much further today, and the weather's getting worse."
[player] "Okay. What should we do, then?"
[sidekick] "We need to build a shelter for the night."
[player] "We could build an igloo, I guess..."
[player] "Let's use pieces of that wreck to build a hut."
Floor
[sidekick] "Ice pillars, maybe, but we don't have time for an igloo."
Floor
[sidekick] "Okay, the floor is flat already, so what should we build first?"
BuildWalls
BuildPillars
[sidekick] "Do we want pillars at the front?"
[sidekick] "They look cool, but we definitely need walls."
BuildWalls
Roof
BuildPillars
Roof
[player] "No, let's not have pillars."
Roof
[sidekick] "Now all that's left is the roof!"
BuildRoof

Figure 6: Part of Task Model for Shelter Level

again in Figure 7), but the basic structure is clear from the node shapes, particularly since we have used rectangles to indicate actions that are utterances.

Notice in Figure 6 that we freely intermix utterances, actions and goals throughout the model. We use ordering constraints (arrows) to guarantee, for example, that a particular introductory utterance precedes all the actions involved in achieving some subgoal and that another utterance (perhaps a snide comment on the quality of the work) is not live until the subgoal is completed. At each point in the interaction, Disco generates based on the task model and current discourse state, both the next utterance or action for the sidekick and an appropriate menu of authored and generated utterances for the player.[7]

Finally, notice that the structure of the first part of the Shelter task model is really just a traditional dialogue tree, which is a bit laborious to encode just using ANSI/CEA-2018. To make this case simpler, we defined a macro extension, called Disco for Games Markup Language (D4gML), which is translated down to ANSI/CEA-2018 using Extensible Stylesheet Language Transformations (XSLT). Figure 7 shows the D4gML version of Figure 6.

Like all XML formalisms, D4gML should not be entirely judged on its human readability in raw form. Typically, XML formalisms are viewed and edited by humans with significant tool support. Nevertheless, the structure of the D4gML description in Figure 7 does mimic the usual indentation conventions of dialogue trees: child elements follow the parent and sibling elements are choices. Thus, repeated nesting of elements represents total ordering. D4gML does not support partial ordering—for this one needs to drop down into ANSI/CEA-2018.

D4gML and ANSCI/CEA-2018 can be mixed in the same task model. The D4gML 'do' element allows D4gML trees to refer to tasks defined in ANSI/CEA-2018. ANSI/CEA-2018 can refer to D4gML trees by using the values of the D4gML 'id' attribute. D4gML trees can also refer to D4gML trees using 'id' values in the 'ref' attribute.

Game Engine API

Figure 8 shows the information flow involved in connecting D4g with the *Secrets* game world, running in Golden T. In order to keep D4g independent of any particular game engine, all of the code that is specific to Golden T has been

[7]Certain goals and decompositions can be marked "internal only" in the task model, which means they are never be used in generated utterances. These are shown as blank nodes in Figure 6.

```
<say actor="sidekick"
     text="Not so fast. I can\'t walk much further today, and the weather\'s getting worse.">
  <say actor="player" text="Okay. What should we do, then?">
    <say actor="sidekick" text="We need to build a shelter for the night.">
      <say id="Floor"
           actor="player" text="Let\'s use pieces of that wreck to build a hut.">
        <say actor="sidekick"
             text="Okay, the floor is flat already, so what should we build first?">
          <say actor="player" text="We need some walls.">
            <do task="BuildWalls">
              <say actor="sidekick" text="Do we want pillars at the front?">
                <say actor="player" text="Sure, let\'s go for it.">
                  <do id="Roof" task="BuildPillars">
                    <say actor="sidekick" text="Now all that\'s left is the roof!">
                      <do task="BuildRoof"/></say></do></say>
                <say actor="player" text="No, let\'s not have pillars.">
                  <say actor="sidekick" ref="Roof"/></say></say></do></say>
          <say actor="player" text="Let\'s put in some pillars at the front.">
            <do task="BuildPillars">
              <say actor="sidekick"
                   text="They look cool, but we definitely need walls.">
                <do task="BuildWalls">
                  <say actor="sidekick" ref="Roof"/></do></say></do></say></say></say>
      <say actor="player" text="We could build an igloo, I guess...">
        <say actor="sidekick"
             text="Ice pillars, maybe. But we don\'t have time for an igloo.">
          <say actor="sidekick" ref="Floor"/></say></say></say></say>
```

Figure 7: Part of Task Model for Shelter Level (Figure 6) in D4g Markup Language

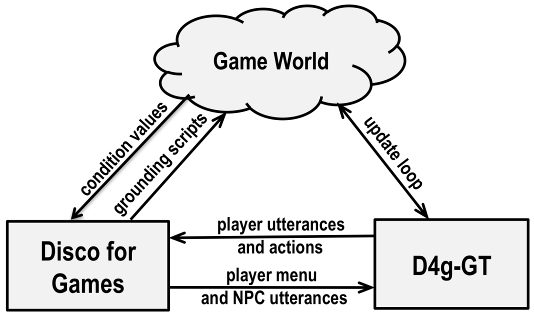


Figure 8: Connecting D4g to a Golden T Game

```
<task id="Walk">
  <input name="to"
    type="Packages.edu.wpi.secrets.Area"/>
  <postcondition>
    world.get("player").getLocation().x
      &gt;= $this.to.getWalkToLocation().x
  </postcondition>
</task>
```

Figure 9: Example JavaScript Postcondition

separated into an extension module, called D4g-GT.

Among other things, D4g-GT contains the Golden T implementation of the UI conventions discussed earlier for displaying NPC utterances, displaying and choosing from player menus, and interacting with objects in the game world. For example, D4g sends the player utterance choices to D4g-GT as a list of strings; D4g-GT handles the display and keyboard interaction, returning the player choice (if any) as an index into that list. When the player performs an action in the game world, for example grabbing the rope, the corresponding task model action instance (with appropriate parameters) is created in D4g-GT and sent to D4g. In order to perform these functions, D4g-GT needs to be part of the main update loop of the game engine.

The other important connections between D4g and the game world are via JavaScript. ANSI/CEA-2018 uses JavaScript for both applicability conditions and postconditions, and for what are called *grounding scripts*, which are used to update the game world for NPC actions. The Java run-time environment makes calling back and forth between Java and JavaScript very easy.

Figure 9 shows the complete ANSI/CEA-2018 definition of the Walk action in the Ice Blocks level, which provides a boolean JavaScript expression as postcondition. The convention for writing such conditions is that the global JavaScript variable 'world' contains a game-specific representation of the game world. The type of task parameters in ANSI/CEA-2018 may be any JavaScript type defined within the running JavaScript engine, which also includes, via the 'Packages' syntax, all types defined in the Java environment. The postcondition of Walk verifies that the x-coordinate of the player's location is within the destination area.

Providing postconditions, especially for goals, signficantly improves the power of a task model in two ways. First, by specifying failure, postconditions allows Disco to automatically retry a goal with a different decomposition, if available. Second, by specifying success, postconditions allow Disco to detect "fortuitous" satisfaction of a goal, i.e., when the postcondition of a goal becomes true without any (or all) of its required actions being executed. This could be due to external influences in the game, such as weather, gravity, passage of time, etc.

Conclusions

We have demonstrated that it is possible to develop a non-modal game integrating dialogue and action in a principled, tool-supported way. We do not know of any similar work in games; the closest non-game systems are DiamondHelp (Rich and Sidner 2007) and LeanCuisine+ (Phillips and Scosings 2000). The tool we have built, Disco for Games, in addition to supporting this non-modal approach, also provides additional benefits, including interruption handling, automatic dialogue generation, plan recognition and automatic failure retry.

Our initial play testing has indicated that players enjoy *Secrets*, but after so much experience with modal interfaces, it took them a while to grasp the potential of the non-modal design. A number of technical challenges also remain, such as how to handle simultaneous interactions with more than two participants (serial interactions with different NPC's is no problem). We have implemented a solution for three-way conversations in the Walrus level, which will not scale.

To make D4g and/or the underlying techniques described here truly practical will probably involve developing more user-friendly tools for building task models and perhaps porting to environments other than Java.

References

Author. M.S. thesis. 2010. (suppressed to preserve blind review)

Despain, W. 2008. *Writing for Video Games: From FPS to RPG*. A. K. Peters.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: Complexity and expressivity. In *Proc. 12th Nat. Conf. Artificial Intelligence*.

Isla, D. 2005. Handling complexity in the Halo 2 AI. *Gamasutra Online* http://www.gamasutra.com/gdc2005/features/20050311/isla_pfv.htm.

Lesh, N.; Rich, C.; and Sidner, C. 1999. Using plan recognition in human-computer collaboration. In *Proc. 7th Int. Conf. on User Modelling*, 23–32.

Lochbaum, K. E. 1998. A collaborative planning model of intentional structure. *Computational Linguistics* 24(4):525–572.

Phillips, C., and Scosings. 2000. Task and dialogue modelling: bridging the divide with Lean Cuisine+. In *First Australasian User Interface Conference*, 81–87.

Rich, C., and Sidner, C. 2007. DiamondHelp: A generic collaborative task guidance system. *AI Magazine* 28(2).

Rich, C.; Lesh, N.; Rickel, J.; and Garland, A. 2002. A plug-in architecture for generating collaborative agent responses. In *Proc. 1st Int. J. Conf. on Autonomous Agents and Multiagent Systems*.

Rich, C.; Sidner, C.; and Lesh, N. 2001. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine* 22(4):15–25.

Rich, C. 2009. Building task-based user interfaces with ANSI/CEA-2018. *IEEE Computer* 42(8):20–27.

Sidner, C. L. 1994. An artificial discourse language for collaborative negotiation. In *Proc. 12th Nat. Conf. Artificial Intelligence*, 814–819.