Zeppelin Time: Exploring the Future of Mixed Initiative in Educational Games

Philip Hanson Sarah Judd Charles Rich

May 5, 2009

1 Project Overview

Our project creates a game designed to teach students about basic programming and object-oriented concepts using text-based mixed-initiative interaction. We call this type of play interaction a *script adventure game*. Students (or players – we will use the words interchangeably in this paper) interact with the game world through a text interface to a JavaScript interpreter and a visual representation of the game world.

The game is built on the CETask reference implementation of the CEA-2018 task model standard [1]. Each room in the game is a goal or subgoal within the hierarchical task network. Subgoals within a room are designed to teach specific programming concepts, and the game interacts with students via the text interface to report progress or offer hints.

The overall interface is reminiscent of text adventure games, and the concept is similar, except for the use of a JavaScript interpreter rather than a natural language parser. Students interact with objects in the virtual environment through the use of JavaScript "objects" whose state is monitored by the game.

With regard to mixed initiative behavior, if the student selects an object to manipulate and then attempts to perform the task or subgoal but fails, the student can ask for a hint. The system will then offer progressively more informative hints as to the proper way of accomplishing the goal. Once the system runs out of hints, the player can request for the system to perform the task, at which point the system will explain what the proper method is and then perform it.

2 Our Game - Zeppelin Time

2.1 Command-Line Interface

To test our game's task model, we created a custom command-line interpreter by subclassing the `edu.wpi.cetask.guide.Guide` class. To the standard commands we added *hint*, a command that allows the player to request a hint from the system.

The organization of the task model reflects the structure of our game. There is a single overarching goal for the model called *Game*. To start playing, users start the *Game* task. Once *Game* is complete, the game is over. Each room within the game is a subgoal of the main goal, and the active 'room goal' is used to track which room the player is in. For Zeppelin Time, these room goals are *Zeppelin*, *Airlock*, and *Office*. These subgoals correspond to the three areas in our game: an airship cargo bay, an airlock between the zeppelin and a building, and finally an office within the building.

Each room goal has two or three subgoals corresponding to activities within the room. Since we are using the game to teach programming skills, players are supposed to perform these activities independently. Therefore, the post-conditions of these 'activity subgoals' are satisfied when the player has done the activity. Below each activity goal is a set of two alternative decompositions: a primary decomposition in which the player is expected to act and a secondary in which the system will perform the required actions on the player's behalf.

In a primary activity goal decomposition, where the player is expected to act, there is an external "waiting" task with the same post-condition as the activity goal. There are also several "hint" tasks that must be performed by the system. These tasks become active when the player asks for a hint using the *hint* command, which increments an internal *hints-asked-for* counter used by the hint tasks and tries to execute the next task. Each hint task has a grounding script that prints a hint to the console.

After the player has used all the available hints within an activity, the final system task in the branch has a grounding script that causes the decomposition to fail. Thus, if a player runs out of hints, the system will fail over to the alternative decomposition where the system performs the action.

2.2 Graphical Interface

The GUI consists of four parts. At the top of the screen, the user sees a representation of the world they currently exist in, often with some flavor text. In the second box, they see a description of what they are supposed to do at the current moment in this world. The next box consists of a command prompt for entering solutions to the puzzles and a button for executing these solutions. The final box displays hints, and messages explaining

3 General Format

In order for this project to be truly usable, it has to be adaptable. While we assume those teaching programming will have at least as much programming ability as the students they intend to teach, we do not assume they will have the time to figure out the CEA task engine. To this end, we provided the option of a general format.

Any game using this system will have a basic structure: a *game* composed of *rooms* which are themselves composed of *puzzles*. The rooms, in addition to

furthering the plot, each have a programming theme associated with them. This is not obvious in our general format at the moment, but it is the idea behind them.

3.1 Puzzles

Puzzles provide the backbone of the game. They need to be solved through understanding the programming concepts. They consist of a *Description*, *post-condition hints* and a *solution*

3.1.1 Description

The Description contains both the in-game reason for solving the current puzzle, and some meta-information required to solve it. For example, the very first puzzle in ZeppelinTime requires the user to set a variable that tells the system his/her name. The description, therefore, reads both "Enter your name so we can recognize you" and "the variable name that should be set is spyName"

3.1.2 Postcondition

In the CEA taskmodel, if the user has already done something, there is no reason to run the task for it. Each puzzle has a postcondition - when the user has completed the task contained within it, he/she does not need any further hints, and can continue to the next puzzle.

3.1.3 Hints

Hints help encourage a user when he/she is lost. It keeps them working on the problem, giving them the chance to try given more information rather than quickly giving up. Hints should give progressively more information about the solution to the puzzle away.

3.1.4 Solution

The solution consists of two pieces: first the computer explains what it is doing in text format. This information goes between the *text* tags. Teachers should place the script that runs the actual solution between *script* tags.

4 Typical run through

4.1 Playing the Game

First, the user is presented with a puzzle. A few possible cases can occur

1. The user understands how to do the problem immediately: S/he can type the answer into the text box next to the "Execute" button then click the execute button. The system will recognize the postcondition has been

satisfied, so it will place "Good work!" in the hint window, and the new puzzle in the what next and picture windows.

2. The user thinks s/he knows the answer, but gets it wrong. The hint window will display "Sorry, that did not help you any." The state of the world does not change beyond this

3. The user is presented with a puzzle s/he has no clue how to solve. S/he can click on the "Hint" button. A hint explaining that step will appear on the screen, in the hint text box. S/he can continue doing this until hints run out. If, at any point, the user thinks s/he now knows the solution s/he can type it into the execute box. At this point, either case 1 or case 2 will occur

4. After ruling out all hints, the user still does not know how to solve the problem. This will trigger the alternate decomposition where the system solves the puzzle for the user. The system explains, on the hint screen, what it has done for the user. After the system solves the puzzle, the state of the world changes as if the user has solved the puzzle him/herself.

4.2 Building a game

A teacher can build a game through filling out the elements of the general form. They can use our general form as an example. They would then need to run an xslt engine on that and our xslt code. This will produce a xml document that works with the CEA Task Engine.

5 Future Work

A few improvements never quite made it into our game. At the current moment, the game is very static. Each room consists of a set number of puzzles. The user goes through each room in order. In games, alternate methods exist to reach the end goal. Several methods exist merely to interact with the world. Future versions of this game should add this.

In addition, the task model provides preconditions, an easy way to only run alternate decompositions if you have/want to. This would be great for an intelligent tutor; we could keep track of how frequently a user runs out of hints in a room, opening up extra questions if the user needs extra help in. Adapting the game to users can be an interesting extension to the project.

References

[1] CEA-2018 Task Model Description Standard: Consumer Electronics Association (2007)