A Game-Building Environment for Research in Collaborative Design

Steven L. Tanimoto, *Fellow, IEEE*, Tyler Robison, and Sandra B. Fan

***Abstract*— Collaborative design practices are evolving rapidly today as a result of improvements in telecommunications and human-computer interfaces. We present a suite of research tools that we have built in order to evaluate a particular methodology for design based on a theory of problem solving from the field of artificial intelligence. These tools are (a) a formal specification for a class of multimedia games, (b) a game-building tool called PRIME Designer, and (c) a game engine that brings games to life. The design of these tools addresses several issues: (1) support for a common language for the design process, deriving from state-space search, (2) visual interfaces for collaboration, (3) specifications for a class of games (called PRIME games) whose affordances represent a balance between simplicity and richness, (4) educating students to work in design teams that use advanced computational services, and (5) assessing the learning and contributions of each team member. We also report on a focus group study in which four undergraduate students used the tools. Our experience suggests that users without a computing background can learn how to employ state-space trees to organize the design process, and thereby gain facilities to coordinate their individual contributions to the design of a game.**

I. Introduction

The ubiquity of computers and the growth of networks have intensified efforts to support collaborative design and problem solving [3]. We are exploring a methodology to engage teams in collaborative design that uses the "classical theory of problem solving". As a part of this effort, we are constructing tools that follow a transparency theme wherein various structures are revealed to users that have traditionally been hidden, such as design histories. We have described aspects of our methodology elsewhere [11, 12].

In this paper, we describe a suite of tools for conducting research in collaborative design following this methodology. We begin by identifying the issues of interest in the design of the tools. We then explain why we chose game design to support the research. That's followed by a discussion of prior work and related issues. We give some details on a new game format called PRIME games, and then we describe the tools. Then we describe a focus group study and provide a discussion of both game design and educational issues that arose in the research

Steven L. Tanimoto is on the faculty of the Univ. of Washington, Seattle WA 98195 USA (corresponding author, tel: 206-543-4848; fax: 206-543-2969; e-mail: tanimoto@cs.washington.edu).

Tyler Robison is a doctoral student at the Univ. of Washington, Seattle WA 98195 USA (e-mail: trobison@cs.washington.edu).

Sandra B. Fan is a doctoral student at the Univ. of Washington, Seattle WA 98195 USA (e-mail: sbfan@cs.washington.edu).

Partial support from the National Science Foundation under grants IIS-0537322 and 0613550 is gratefully acknowledged.

A. Issues of Interest

Our question of primary interest is this: Can the field of artificial intelligence give us a common language for the design process? But we are also interested in these secondary issues: effectiveness of collaboration; teaching design methodology; readiness of designers to use computational services; the use of games as collaboratively designable objects; and assessment of student learning.

B. Why Game Design?

In our research on the design process, we needed a class of "designable objects" with several properties: (a) they should be *software-based*, as a practical matter; we did not want to worry about glue, welding, chemicals, etc., (b) they should have *controllable complexity*, so that the time required to complete a design could be limited to 2 hours, (c) they should require, or at least permit, *interdisciplinary contributions*; that is, expertise from multiple fields should be needed during the design process, and (d) they should represent a *compelling goal*, in terms of reward and challenge, for undergraduates serving as human subjects.

C. Prior Work on End-User Game Design

The relevant literature for this article falls into four categories: game design by end-users, collaboration, formal design methodologies, and the use of artificial intelligence in game design. Let's now consider the first of these.

Tools such as MissionMaker [6, 7] and the Mockingbird kits [4] allow users to create their own games, within certain constraints, without the need for prior programming experience. MissionMaker allows designers, often young students, to construct "complex 3D adventure and puzzle games." These games involve players exploring 3D spaces populated with images and sounds, objects to be interacted with, characters to comunicate with, and rules that respond to events as dictated by the designer. Users are able to construct these games using a perspective similar to that in playing the game, and then can play through the games.

The game kits from Mockingbird [4] offer children the means to construct games that include stories and characters that may be personally meaningful. The company emphasizes the idea that constructing games might be more enjoyable and worthwhile than playing games.

D. Collaboration in Game Design

Any game designer working in a team will need to find ways to collaborate with teammates. Collaborative software often provides affordances for articulation work [7, 9] and awareness [2]. Articulation work is the process of deciding upon a division of the work into separate units required to

complete a project, and how to integrate it back together. Awareness is maintaining knowledge of other group members' activities. Some version control systems may use "lock" icons to show when another developer has taken exclusive-write privileges on a file; this is a rudimentary awareness affordance. Our tools support collaboration in the game design process through the use of "roles." We describe this further in Section III.

E. Formal Design Methodologies

Much of the theoretical work on design methodologies has been developed by architects such as Christopher Alexander or in the field of software engineering. Some of these ideas are codified in the "metadesign" approach of Fischer et al [3]. Our own design methodology grows out of work done in the 1960s by members of the artificial intelligence research community. The key ideas have been well articulated by Herbert Simon [9]. We elaborate on this methodology in Section III.

A different type of formal design methodology for games (Church [1]) focuses on patterns of player experience, in terms of subgoal formation, awareness of possibilities, player control, and unforseen consequences of actions. This approach is particular to games, whereas ours is a general design methodology that we happen to apply to games.

F. Artificial Intelligence in Game Design

The work of Togelius and Schmidhuber [13] addresses the use of artificial intelligence techniques to create content for games. In that work, the rules of the game are created automatically by the computer with a fitness function based on Koster's theory of fun. The search space is explored via a hill-climbing algorithm using this fitness function. Their work illustrates how the mechanics of a game itself (as opposed to an agent inside the game) can be created with the help of artificial intelligence techniques. In our work we use structures from artificial intelligence while still keeping humans in the loop. Our design system represents possible game configurations as states in a search space, but instead of automatically picking one, it allows users to inspect and traverse the search space, and exercise their own judgment in picking successor states.

G. Overview of the Game-Building Environment

There are three components to the suite we describe here: (a) a game class specification called PRIME games, (b) a game-design tool, and (c) a 3D game engine for playing the games. Each is described in one of the next three sections.

II. The PRIME Games Specification

We have designed a class of computer games to meet several objectives: (a) integration of four disciplinary components in the design process (architecture, engineering, music, and computer science); (b) simplicity; and (c) richness of possible player experiences.

A. Rationale for PRIME Games

While the main reason to develop a class of games has been to create something to support our research, we also found a lack of open game formats that would support the type of games we needed. The lack of standards can be attributed to an industry tendency toward proprietary formats [5]. The PRIME acronym stands for "Puzzle Rooms with Image and Music Experiences". The four disciplinary components of a PRIME game are these: (a) the architectural layout, including the selection and placement of wallpaper, background music, puzzles, and doors; (b) image puzzles in which secret messages are embedded in scrambled images and which must be unscrambled during the game; (c) music puzzles in which melodies are permuted and must be unpermuted during the game; and (d) logical rules that control opening of doors, awarding of points, playing of audio clips, and textual announcements.

The current version of the PRIME specification is 1.0. Games in this format are limited to 9 square (or cubical) rooms in a 3 by 3 layout. Any adjacent pair of rooms may have a door between them. Each room may have background music that starts playing when the player enters the room. Each wall may have wallpaper (given as an image file) as well as an image puzzle or a music puzzle. When a player solves a puzzle, credit is awarded in the form of a "magic phrase" which can be uttered to perform actions such as opening a door or gaining points.

B. Standard Affordances and their Specification

A PRIME game is described in an XML file, and the file must be accompanied by any image and audio files referred to in the XML. The following example is part of the XML file describing a game called Mozart's Maze that we use as a running example in this paper.

```
<prime_game name="Mozart's Maze" version="0.1">
  <layout>
   <overall_default_wallpaper name="Fcm.jpg">
     </overall_default_wallpaper>
   <room num="0" name="ROOM-1"
bg_audio="Ssmp.mp3">
     <wall loc="N">
       <wallpaper name="Sburg.jpg"></wallpaper>
     </wall>
     <wall loc="E">
       <door num="12" into="1" state="open"></door>
     </wall>
     <wall loc="S"></wall>
     <wall loc="W"></wall>
   </room>
```

Figure 1a. Portion of a PRIME game XML file, showing header information and contents of the first room.

As can be seen in Figure 1a, a room is described as an entity inside the layout and having four walls. Each wall in the room can have its own features, including wall-specific wallpaper, or a door. A wall can also hold a puzzle, though this is not shown in this example.

In Figure 1b, the XML specification of (most of) an image

puzzle is given. The puzzle is identified by a number (for the computer) and a name (for the designers and players). The embedded secret message in this example is "Papageno." There are some operators specified here as part of the puzzle: Horizontal flip, and Shuffle rows twice. These are options that a player will have when working out a solution to the puzzle. (Note, some operators of the real puzzle have been left out of this excerpt for reasons of space and simplicity.) The correct sequence of operator applications, given in this excerpt, tells a game engine how to determine whether the player has solved the puzzle. The puzzle specification is completed with a sequence of "procimage" declarations. Each of these serves to associate with a particular operator sequence a preprocessed image that the engine can display.

```
<image_puzzle num="1" name="Symphony #40 in G minor">
    <message text="Papageno"></message>
    <source role="1" file="papageno.jpg"></source>
    <operator role="0" name="Horizontal flip" op="T_0">
    </operator>
    <operator role="1" name="Shuffle rows twice" op="T_1">
     </operator>
    <correct_sequence seq="(3, 1, 2)"></correct_sequence>
    <proc_image seq="()" filename="Puzzle-1-img-ver--.jpg">
    </proc_image>
    <proc_image seq="(0,)" filename="Puzzle-1-img-ver--0.jpg">
     </proc_image>
    <proc_image seq="(0, 0)" filename="Puzzle-1-img-ver--0-0.jpg">
    </proc_image>
    <proc_image seq="(0, 1)" filename="Puzzle-1-img-ver--0-1.jpg">
     </proc_image>
    <proc_image seq="(0, 2)" filename="Puzzle-1-img-ver--0-2.jpg">
     </proc_image>
```

Figure 1b. Portion of a PRIME game XML file, showing part of an image puzzle specification.

These images are computed by the design tool at game export time, so game engine need not have an image processing facility. Only five declarations are shown in this excerpt. In a normal puzzle, there are many more.

The XML specification for a music puzzle is similar in structure to the description of an image puzzle. What's scrambled is a melody rather than a digital image. In order to save space, we do not show the music puzzle XML here.

One more type of item specification in a PRIME XML file is that for a rule. A rule tells the game engine to do something when certain conditions are met. Each rule therefore has a conditions section and an actions section. The example in Figure 1c has a single condition and a single action. The condition is satisfied when the player first gets to room 3. The action increases the score by 10 points.

We do not list all the possible conditions and actions that can be represented in PRIME XML files here. However, some of them have to do with setting and testing boolean variables called flags. The rule designer can set up any number of flags to maintain the state of the game such as whether particular challenges have been met, etc.

```
<rule num="0" name="RULE_1"
    condition_type="AND">
  <condition event="Reached" arg="3">
    </condition>
  <action event="Score" arg="10">
    </action>
</rule>
```

Figure 1c. Portion of a PRIME game XML file, showing a rule specification.

III. The Game Design Tool

Our game design tool is called PRIME Designer. It is an interactive tool that makes the game design process into a sequence of "design acts". The interface of this tool is based on state-space search trees [9] and its diagrammatic features were described in [12]. The tool is intended for teams of up to four designers, with each member using a separate copy on his or her own personal computer. Before we describe the tool itself, we first present the general theory of problem solving on which it is based, and the adaptation of this theory to the realm of design.

A. The Classical Theory of Problem Solving

Our reliance on a theory of problem solving as the foundation for our design tool is consistent with the statement, "Designers solve problems." (from the Preliminary Report on the NSF Workshop on Science of Design: Software and Software Intensive Systems, p.10). The theory is better known in the artificial intelligence community as the theory of state-space search. The two fundamental constructs of this theory are the *state* and the *operator*. A state represents the progress made, at a particular point in time, along a particular line of attack, towards solving a problem. For example, in solving the "Towers of Hanoi" puzzle (in which a series of rings of diminishing sizes is to be moved from a first peg to a third peg, making use of a second, intermediate peg, where only the topmost ring on a peg can be moved at any one time, and it must never be placed on top of a ring of smaller size), a state consists of one arrangement of the rings on the pegs.

An operator is a scheme for making a "move." For example, an operator for the same puzzle could be "Move a ring from Peg 1 to Peg 2." This operator would only be applicable if there is a ring on Peg 1 and if any ring on Peg 2 is larger than the ring to be moved.

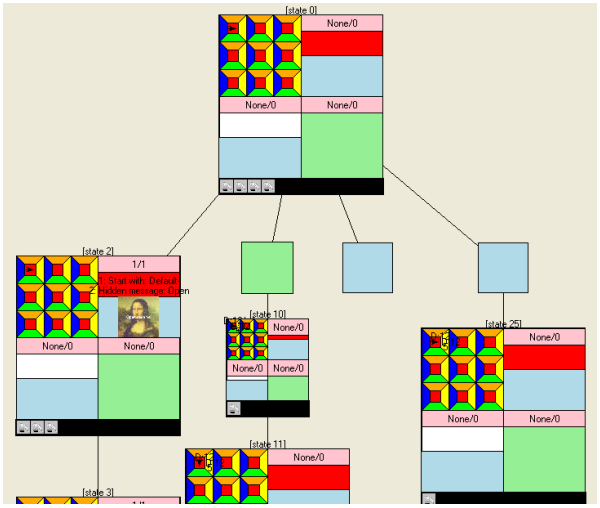

Figure 2. A view of the design history tree in a PRIME Designer session.

The theory is concerned with such issues as the representation of states, the rate of growth of the number of possible states, as a function of how many moves are made, methods for automatically creating and testing the states (search algorithms). A solution to a problem may correspond to a particular state or to a sequence of states known as a path. For the Towers of Hanoi problem, it is the path starting with the initial state (having all rings on Peg 1) and ending at the goal state (having all rings on Peg 3) that represents the solution.

Next we discuss how state-space search is used in design.

B. Design Acts

A design problem can be formulated as a state-space search problem by specifying an initial state (usually the "empty" design) together with a set of operators. The operators generally add some element to a state, producing a new state in which there is one more element of the design than there was before. For example, to design a house, one might begin with an empty floor plan and, with an operator, add a living room. With another operator, a bedroom might be added, and another operator might select the location of the bedroom relative to the rooms already added. When an operator is chosen by a designer to apply to the current state, the designer commits a design act. Another kind of design act is selecting, from all of the states constructed so far, a state to be the current state. This may involve a judgment about which state is the best so far.

C. Collaboration via Roles

Each member of the game design team carries out specific duties in his or her specialty area, and is able to view the contributions of others. Currently these duties are predefined in the system, e.g. one team member is in charge of creating certain types of puzzles, another is responsible for logic rules concerning points earned in the game, etc.

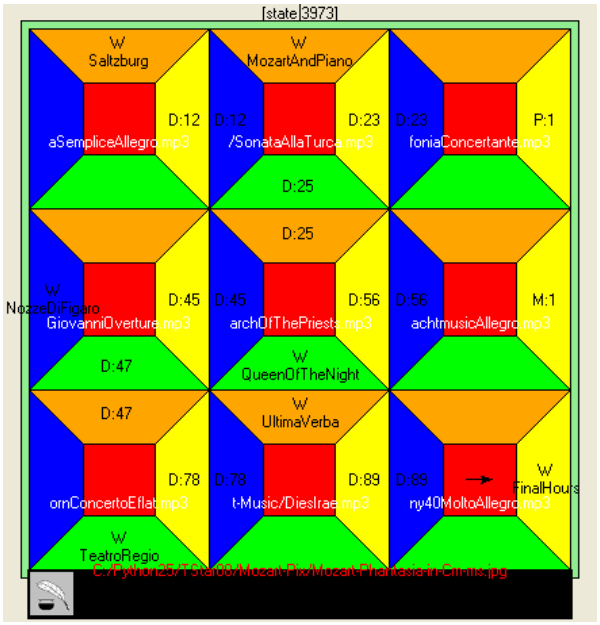

Figure 3. Enlarged node as viewed by the design team's architect.

After members finish their parts, they can merge their designs, and are able to see all the specific design decisions their teammates made. Additionally, our game designers can comment on other's design decisions by adding *annotations* to any given state. In the following subsections, we describe each of the four roles in details.

D. Designer Views by Role

Each team member sets his or her "role" to either architect, image puzzle designer, music puzzle designer, or rule-base designer. It is also possible to select "all roles" to see all four aspects of the game being designed. Figure 2 shows an all-roles view, whereas Figure 3 shows the architect's view of one node, and Figure 4 shows the image puzzle designer's view of one node. Figure 5 shows the music puzzle designer's view, and Figure 6 illustrates the rule-base designer's view.

The role of the architect on the design team is to place game items in rooms or on walls. Most importantly, the architect specifies where doors go. The architect can place puzzles (that have been created by the team's image puzzle designer and music puzzle designer) on walls. He or she can also specify background music for each room and wallpaper for each wall, room, or for the entire set of rooms.

The image puzzle designer creates one or more image puzzles that can be placed by the architect in the maze. Each image puzzle has three elements: a hidden image, a hidden textual message, and a sequence of scrambling transformations that effectively hide the image and the message. When a game player works on an image puzzle, he or she tries to unscramble the image and the textual message by finding the sequence of image transformations that inverts the scrambling done by the image puzzle designer.

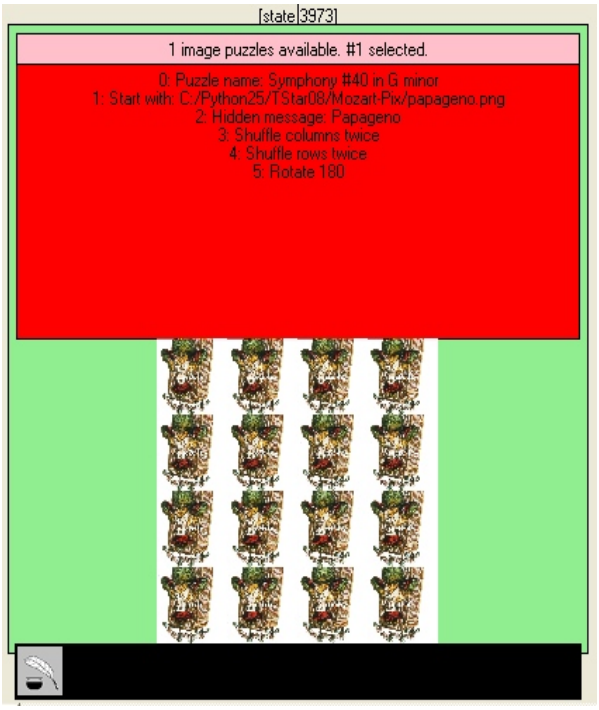

Figure 4. A node as viewed by the team's image puzzle designer.

The music puzzle designer creates one or more music puzzles. A music puzzle consists of a background image, a musical melody that is represented as a string of note names (i.e., A, B, ..., G) and a sequence of permutations that scrambles the melody. A player who works on a music puzzle tries to unscramble the melody by creating a sequences of permutations that inverts the scrambling. The current version of the partially unscrambled melody is played after each step of puzzle solving.

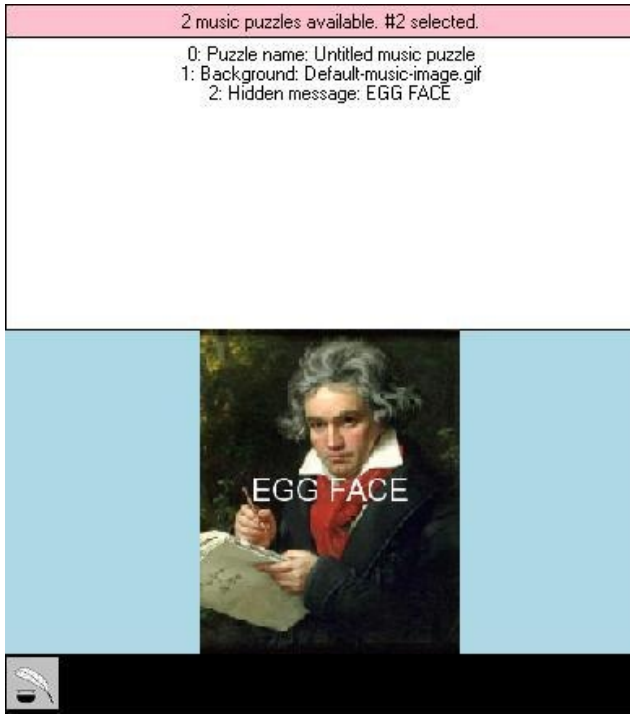

Figure 5. Music puzzle designer's view.

The rule-base designer's responsibility is to create a set of IF-THEN rules to control the operation of the game. Each rule has a list of conditions that must be met for the rule to fire, plus a list of actions to be taken when the rule fires.

To an extent, the team members can work independently on their own components of a game. However, the components must work together if the game is to be an enjoyable experience for its players. The messages and images involved in puzzles should fit the themes of the rooms in which they are placed. Solving them should, with the help of appropriate rules, lead to new opportunities (i.e., doors opening). Formal communication among the team members is only required for merging their contributions, but informal communication and adjustment are needed all through the design process, so that the components will work together and all support a consistent theme or story.

E. Facilities for Combining Work

Partial designs by different designers can be combined through tree merging operations. Most nodes represent incomplete games. In the current version of PRIME Designer, the work done by each team member must be combined using a 4 step process. First each team member saves their work locally and then extracts the path (from root to leaf node) representing the contribution they wish to share. Next, each team member uploads this contribution to the PRIME Designer server. Third, a designated team member downloads the single-path trees of the other members, forming a tree with separate branches for each of the contributions. Fourth, the four paths are merged using a special menu option in the tool. The resulting tree contains one very long path produced by taking all the operator sequences involved in all the contributions and concatenating them. The leaf of this long path represents the final design.

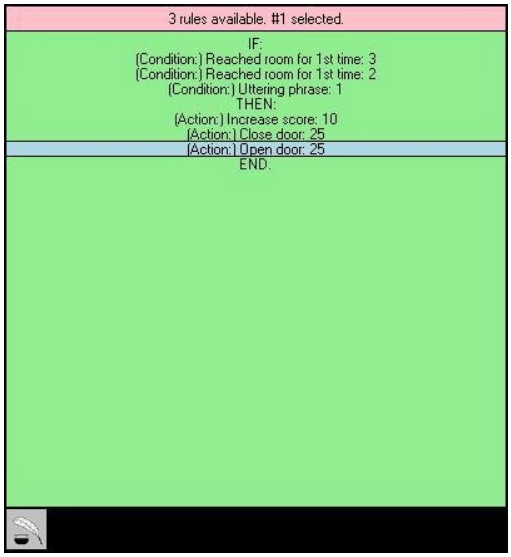

Figure 6. State view of a rule-base designer.

IV. Game Engines

Our suite of tools includes three game engines for playing PRIME games. The simplest is a textual engine; when using this engine, no images are shown and no audio is played (only their filenames are shown). There is a 2D graphical game engine that graphically shows the doors, wallpaper and the image puzzles in a game. Music puzzles operate with simple sound generation. This engine shows an architect's view of the game at all times, making it useful for debugging a game, but usually too easy for players.

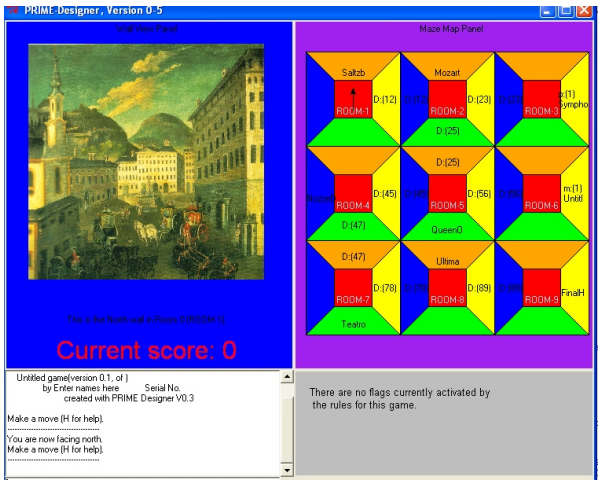

Figure 7. The 2D graphical game engine. Player commands are given through the text input pane.

The 3D engine, in contrast, offers players a more immersive experience, comparable in some respects to what many game players expect. Except when a special birds-eye view is enabled (see Fig. 8), players can only see the room they are in, plus whatever they can see through open doors. They move through the game using navigation keys that work geometrically -- more like driving a car than commanding an agent. Figure 9 gives a player view within the example game Mozart's Maze.

Figure 8. Birds-eye view of the Mozart's Maze game in the 3D graphical game engine.

Figure 9. Player's view in the first room (Salzburg) of Mozart's Maze, using the 3D graphical game engine.

An image puzzle initially appears to a player as a scrambled image (Fig. 12). By selecting transformations in the right sequence, a puzzle can be solved. Music puzzles have a similar structure, but it is the notes of a tune, rather than the pixels of an image, that are scrambled and unscrambled.

The PRIME specification allows the designers to indicate both per-game wallpaper defaults and per-room wallpaper defaults. The image specified is used to cover the surfaces of rooms.

V. EDUCATIONAL ASPECTS

A. Educational Use of PRIME Designer

One motivation for our research is to come up with an approach to teaching elements of the classical theory of problem solving to students and designers. We want designers from different backgrounds to share a common language about the design process. This common language should also support the designers in controlling and monitoring computational agents that perform services related to exploring the tree of states. While we do explicitly teach about the theory before having students use the tools, the tools themselves can provide a certain amount of feedback to the students to help them learn the theory and understand their (and their team's) progress in the task.

B. Needs and Affordances for Assessment

The assessment needs for an educational tool can be classified into several groups: (a) confirmation for actions taken, (b) sources of information for user self-assessment, and (c) automatic performance evaluation. The PRIME Designer tool currently supports only the first two of these.

For confirming actions taken, there are two important aspects: first, letting users know that the system has accepted

their input and responded, and secondly, expressing the confirmation using the language of design in such as way as to reinforce the user's knowledge of the language. The menu of commands for state-dependent actions contains an item "apply one operator" so that the user knows that the function being chosen is an operator. Then, after the operator has been applied, the tree is updated to show a new state, and the state is labeled to show that it is a state, e.g., "State 29." Thus the interface's feedback reinforces the vocabulary of the classical theory.

Sources of information for user self-assessment are provided in several forms. Figure 10 shows some of the means by which team members may be supported (by the current implementation of PRIME Designer) in their efforts to assess their own progress. First, there is the tree display, which can help a user see how far they have come and what detours they may have taken along the way. Shown in the lower left of Fig. 10 is the displayed list of postings on the server; from this one can infer how many of one's teammates have reached a design milestone. The tool also outputs two textual streams – one to the Python console, containing short messages to confirm I/O operations and activity during longer operations such as exporting games, and the other stream to a log file, where a more permanent record of the session is kept for research purposes.

Figure 10. Affordances with potential use in self-assessment. The tree diagram can be used to show the separate contributions of team members (top). The console text stream (center) reports on the success, failure or status of actions such as input and output, the list of server postings (lower left) indicates milestones being met by team members, and the log-file stream (lower right) shows what aspects of the session are being recorded.

VI. Focus Group Study

We conducted a formal evaluation session in which four paid undergraduate students were introduced to the tools and given an opportunity to design a game. The session lasted approximately two hours and 30 minutes. After 45 minutes devoted to background questionnaires and an introduction to the tools, the students began designing their own game. They were given the choice of starting a game from scratch or extending Mozart's Maze; they chose to start from scratch. At the end of the session, they had many elements of a playable game, though some aspects of their game were not yet debugged. Here are some of our observations: although the role of architect involved the most work, the role of rule-base designer was the most difficult one, requiring thinking at a more abstract level than the others and requiring a mental model of the game as a whole. The puzzle designers had few constraints on their contributions and only had to coordinate their work with the architect and rule-base designer at an artistic and strategic level, not a technical level. The rule-base designer used in interesting design strategy: to "follow" the architect from room to room and write rules associated with each room reached in the order reached. One misconception exhibited by team members was that the individual puzzles and rules could be created in separate branches of their trees and yet contribute to one design. With path merging, this is true, but normally, each branch in a tree represents an alternative design sequence rather than a component of a design. There was also confusion about how to "undo" a design decision. Backing up in the tree and creating a branch to "undo" an early choice of operators also "undoes" the applications of operators subsequent to the early one. It took the team a while to realize this. In terms of collaboration and communication, the architect and rule-base designer had the greatest needs to coordinate their work. To save time and reduce the learning load on the team, the authors took care of uploading, downloading, path extraction and path merging to combine the contributions of the team. We have begun work on a new version of the design tool that will simplify this.

VII. Discussion

Designing with PRIME Designer could be considered playing a kind of game in its own right. We are thinking of adding a scoring system that might add an extra measure of motivation to the designers to meet milestones. While PRIME Designer trees might be compared to game trees of games like Checkers, etc., PRIME teams are assumed to be cooperating rather than competing. Nonetheless, a scoring scheme could provide both individual and team scores.

The most obvious limitation of the current PRIME class of games is probably the restriction to 3 by 3 arrays of square rooms. We could drop it, but the architect's responsibilities during game design would then significantly exceed the responsibilities of the other designers.

Figure 11. A subset of the puzzle images precomputed by PRIME Designer.

An interesting implementation issue in our tool suite concerns where to put specialized processing for puzzle-solving. In principle, a player solving an image puzzle would be doing actual image processing. However, to simplify our game engines, that activity is simulated by having the design tool precompute most of the images that a player is likely to need to see during the activity. If the image puzzle designer makes a puzzle complicated (by scrambling the target image with a long sequence of transformations), then many images will be necessary. Figure 11 shows thumbnails of some of the images automatically produced for the Papageno puzzle in Mozart's Maze. The XML file associates one image with each sequence of transformations a player might try. When the player tries sequences not in the precomputed list, a special "dead end" image is shown by the engine.

Our work thus far shows how it is possible to structure the game-design process as a kind of game itself, using state-space search trees in an explicity way. Future work includes additional testing of the tools with users and reimplementing the design tool for the web.

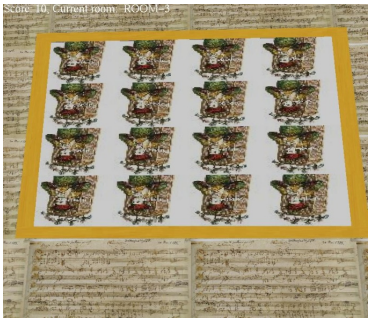

Figure 12. View of Papageno image puzzle in Mozart's Maze.

VIII. Acknowledgements

The authors would like to thank Earl Hunt, Brian Johnson, Daryl Lawton, Andrew Morozov, Linda Shapiro and Kofi Weusijana for participating in discussions of PRIME games.

References

[1] D. Church. "Formal abstract design tools." Game Developer, August, 1999. Republished online at Gamasutra: http:// www.gamasutra.com/features/19990716/design_tools_01.htm

[2] P. Dourish, and V. Bellotti, "Awareness and coordination in shared workspaces". Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work, pp.107-114, ACM Press, New York, NY, USA, 1992.

[3] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, "Meta-design: a manifesto for end-user development", *CACM* , Vol. 47, No. 9, 2004, pp. 33-37.

[4] T. Gilbert. "Game Making is the Game: Lessons Learned from Mockingbird" (presentation at GDC 2009, San Francisco, CA) http://mockingbirdgames.com/wp-content/uploads/2009/04/ game-making-is-the-game-presentation.pdf

[5] K. Hoet, "Open Game Format". 14 Oct. 2008. Blog at http://crossthebreeze.com/2008/10/14/open-game-format/.

[6] Immersive Education. MissionMaker. Product website at http://www.immersiveeducation.com/missionmaker/.

[7] C. Pelletier, "Making Games: Developing Games Authoring Software for Educational and Creative Use: Non-Technical Summary", RES-328-25-0001. Swindon: ESRC. (2007).

[8] K. Schmidt, and L. Bannon. "Taking CSCW seriously". Computer Supported Cooperative Work, Vol. 1, No. 1, 1992, pp.7–40.

[9] H. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge MA: MIT Press, 1996.

[10] A. Strauss, "Work and the division of labor". The Sociological Quarterly, Vol. 26, No. 1, 1985, pp.1–19.

[11] S. Tanimoto, "Enhancing state-space tree diagrams for collaborative problem solving". *Proc. DIAGRAMS 2008*, also at http://www.cs.washington.edu/ole/d08.pdf.

[12] S. Tanimoto, and S. Levialdi, "A transparent interface to state-space search programs". *Proc. ACM SoftVis 2006*, Brighton, UK.

[13] J. Togelius, and J. Schmidhuber, "An experiment in automatic game design", *Proc. IEEE Symposium on Computational Intelligence in Games* 2008, Perth, Australia, 2008.