Game Design as a Game

Robert H. Thompson and Steven L. Tanimoto
Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 98195
robthomp@cs.washington.edu, tanimoto@cs.washington.edu

***Abstract*—The software engineering process for games has enough special structure that it can be formulated as a kind of game itself. This, in turn, permits the teaching of game construction in a unique way with new potential to motivate students. We present a new game design client program for an existing collaborative problem-solving website known as CoSolve. The client was built with an emphasis on increased interaction and fine control over a problem's state. With this comes the opportunity to more easily design and test games in the CoSolve space. It is our hope that this will teach and inspire student users to learn more about game design, problem posing, and programming in general.**

***Index Terms*—Game design, problem solving, collaborative computing, software engineering, education, meta-game.**

Introduction

Game design has significant potential as an educational activity [1]. State-of-the-art games require coding skills just as formidable as for any other software type. Interesting problems can arise from human-computer interaction, artificial intelligence, systems and other aspects of computer games. Perhaps most important however is the connection between playing games and learning game design, because of games' motivational power. Games are a rapidly expanding form of entertainment and most young students have played and enjoyed at least one. Framing game design as a game itself can interest students unfamiliar with the process of game development, after which they can be gradually exposed to more complex ideas. Competition and working towards task-oriented goals are types of activity that can be leveraged to engage students even as they learn how to use the same elements themselves in their own game designs [2].

Tool Design Criteria

There are many ways to frame game design. When framed in the context of open-source software development, it has been found that complex and uniquely organized communities can form to optimize design output [3]. Our project, on the other hand, is concerned with the use of a problem-solving framework for the game design process. Our intent is not so much to offer a robust design tool as to provide a tool that employs a unique structure and centers on motivating students who are new to the activity of game design.

A. The CoSolve System

Our research group at the University of Washington has developed a web-based system called CoSolve for research in computer-supported collaborative problem solving [4]. CoSolve uses the state-space search methodology as a structure for the problem-solving process. (A well-known proponent of this approach is Herbert Simon [5].) One interpretation of what CoSolve does is that it offers scaffolded problem-solving experiences to users, offering specific actions that can be taken at each step of the problem-solving process. A CoSolve "template" represents a class of problems, each of which has an initial state and which uses a set of operators defined with the template. The initial state and operators together specify a "problem space." Such a problem space is a potentially infinite set of discrete states that can be constructed by applying transformations to the initial state or other states in the set.

When a team of users begins to solve a problem with CoSolve, one member of the team selects an existing template and creates a new "solving session" associated with that template. The solving session is represented to the users as a dynamic tree, which starts out containing only one node: a root that represents a realization of the initial state. The team members, accessing the tree through their own web browsers, build new nodes in the tree in order to construct a branch that leads to a goal node, thus representing a solution. To build a node, a user selects an existing node, chooses on operator from a given list of operators that are applicable to the selected node, and (if required) specifies the values of any relevant parameters to the operator. The CoSolve server receives this information and applies the specified operation to the specified state, thus computing an explicit representation of the implied new state. The view of the user who initiated the operation is immediately updated with a new node, while the other users are notified that an update has taken place and given the option to update their own views.

A problem template may specify a goal state, but it is not required to do so. Thus the CoSolve framework is applicable not only to solving problems with concrete solution criteria but also to working on design problems with fuzzier goals. A session tree representing the states visited in a design space is actually a history of the design process. Such a process may involve significant branching, and it may contain the work of many different people.

The CoSolve system is comprised of (1) a server-side content management system (Drupal) augmented by a special module to handle state creation and operator application, and (2) various associated CoSolve clients. Different clients can be used to view the same information, often presenting that information in different forms to better suit an individual user's needs or personal device for accessing the internet. This structure is shown in Fig. 1.

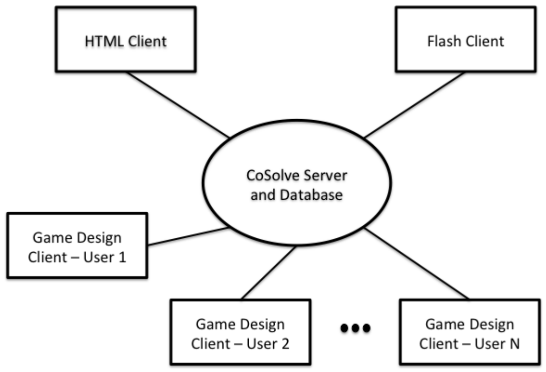

HTML Client
Flash Client
CoSolve Server and Database
Game Design Client – User 1
Game Design Client – User 2
•••
Game Design Client – User N

Fig. 1. Relationship between the CoSolve server and various clients. Users interact with the server through one of several clients.

Since different clients can access and change the same content at any time, CoSolve supports both synchronous and asynchronous collaboration. Changes are always additive. In addition to creating new nodes, a user can create textual annotations associated with any node. One node may carry any number of annotations, and the annotations of a single node may be written one user or many users. Nodes cannot be deleted or altered after they are first generated. Thus users do not have to worry about their annotations being unlinked or disappearing. When a user makes additions to a tree, any other users currently editing it are informed and given the option to view these changes immediately or at a later time. The changes will also automatically be included for new users viewing the tree at any time after that point.

B. Gaming Elements in Problem Solving and Design

Our previous experience with the CoSolve project has shown that the problem-solving process in CoSolve can be treated as a game, even if the problem itself is not presented that way. The following is a list of game-like elements we've observed during CoSolve activities:

- Short and long-term goals
- Levels
- Points/Score
- Competition/Leader Boards
- Collaboration

The long-term goal of any user in CoSolve is often to solve a particular problem. Short-term goals can take many forms, but they may involve escaping a local minimum in the state space, out-scoring another user or team, or even just better understanding how a problem is structured. Different instances of a problem or different problems themselves can be thought of as levels. Once a user team has solved a problem to their satisfaction they can move up to another. A state evaluation function can provide a score that users are encouraged to optimize. Users often end up collaborating or competing in the same solving session to find superior states. The overall contribution a user has made to a problem-solving session is easily measured and compared with those of other members of the team. Another member of our research group, Tyler Robison, has created client functionality that displays aggregate data about a solving session including users' contributions [6], which can be thought of as a leaderboard. Users sometimes interact with the problem solving process as if they are playing a game. Right now, the primary obstacle to a better game-design experience is the lack of sufficient interactivity in the CoSolve client program. This makes many problem types, and design problems in particular, more difficult to tackle than necessary.

C. Overcoming Current Client Drawbacks

The standard CoSolve client runs in Adobe Flash and was written in ActionScript. (We will refer to this client as the "Flash client.") An example solving session as viewed through the Flash client is shown in Fig. 2. The root node is the problem's initial state and branches exploring different problem solutions can be seen. The usual way to interact with states in the Flash client is through lines of text. New states are generated by typing in a line of parameter values to be parsed by an operator[1]. States are shown as static PNG images that are generated by the server at the same time as the state is created. If specified in the template, multiple PNG images can be made for the same state to show different aspects of the state, but they are all still static. Because processing is done on the server, the Flash client does not have the ability to manipulate states in any more organic or complex way. The Flash client also offers no opportunity for augmentation of the solver interface by template developers, unless they are CoSolve site developers. With this in mind, we have developed a new client built to be an effective game design tool and to be more interactive and moldable in general.

THE NEW TOOL'S AFFORDANCES

This section describes the new tool's features and its potential for different types of collaboration.

A. Standard Tool Features

Rather than a static visualization as in the Flash client, the Game Design Client (GDC) is implemented as dynamic JavaScript code that runs in the browser on the client's side. States are maintained and manipulated in the client rather than on the server, so new modes of interaction beyond just text are now possible. For example, an object in a game space can be moved much more intuitively with a mouse than by typing in object coordinates as one would have to do in the Flash client.

[1] An experimental method that converted a single click into a coordinate string that was applied to an operator as a parameter was made as well.

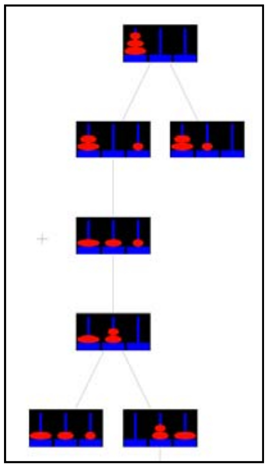
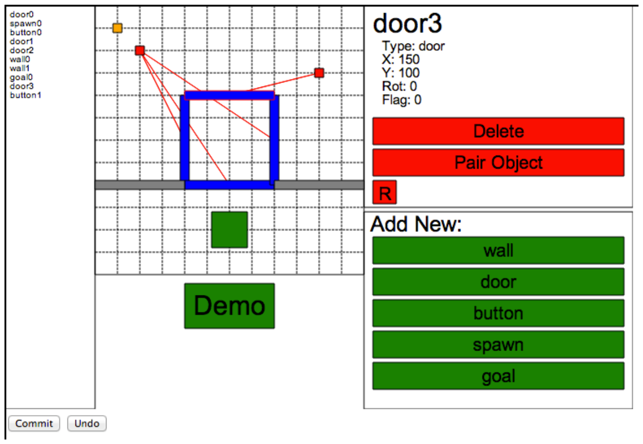


Fig. 2. A view in CoSolve of a tree representing the portion of the state space explored by a team for the Towers of Hanoi problem. Such a tree is displayed by the preexisting Flash client. Branches represent the exploration of alternatives.


door0
spawn0
button0
door1
door2
wall0
wall1
goal0
door3
button1

door3
Type: door
X: 150
Y: 100
Rot: 0
Flag: 0

Fig. 3. Screenshot of new game design client interface. The main work area is shown in the center with object list on the left and detailed object view on the right.

This is not a deficiency of the Flash plugin but rather due to how the Flash client was designed. With JavaScript, there are several libraries available that can make these behaviors easy to implement, such as the RaphaelJS library that offers Scaling Vector Graphics with click-drag functionality.

To illustrate the capability of this new client as a platform for game design tools, we have made an example problem template TGDC1 based on a predecessor of CoSolve, which was called PRIME Designer [7]. PRIME Designer is a standalone Python program that corresponds to a CoSolve template plus an interactive control program. It supports collaborative game design with several users working together while inhabiting different design roles: architect, image-puzzle designer, music-puzzle designer, and game-logic designer.

Each game in the class of games enabled by the new TGDC1 template and the GDC client involves a large room with a top-down view in which users can place walls, doors, and other interactive game objects. Objects are placed with the mouse, and they snap to grid intersection points automatically. An example of the design interface is shown in Fig. 3. Object behavior is defined by binary relationships between objects. For example, if a button object is paired with a door object, the door will be closed or opened when the button is pressed. While this type of association cannot fully describe the complex behavior one might see in a modern game, it can serve as part of an easily understood introduction to more advanced types of scripting.

B. *Instant Playtesting*

Another key feature of the GDC is that at any time, a user can click an onscreen "Demo" button, which engages an embedded Unity web-player to generate an interactive scene from the current problem state, an example of which is shown in Fig. 4. This is one of the features that distinguishes the GDC from other online collaborative design tools. The ability to instantly playtest and interact with a game while it is being designed is a large motivating factor for students. For new users just starting to comprehend the design process, we can present a complete pre-designed game scenario, allowing them to play through it, and then step back through the state-space, seeing how the scene was built piece-by-piece by different users and even viewing design alternatives that eventually were abandoned. For more experienced users, this is a way to quickly playtest a design and discover any flaws or deficiencies.

C. *Design as Collaboration*

The collaborative aspect provided to the GDC by CoSolve is a relatively unique feature in the realm of game design. Traditional game engines are desktop-based and lack any sort of collaborative aspect beyond the use of standard version-control software. The GDC runs in a browser without any plugins, allowing anyone with interest to participate, and automatically preserves the game design at all points of its development. This allows users joining a design in-progress to view the history that led to its current state. User activities can be monitored and compared to each other. Unsuccessful offshoots can still be studied to garner any useful aspects they may still have. Since every state can also be played at any time,

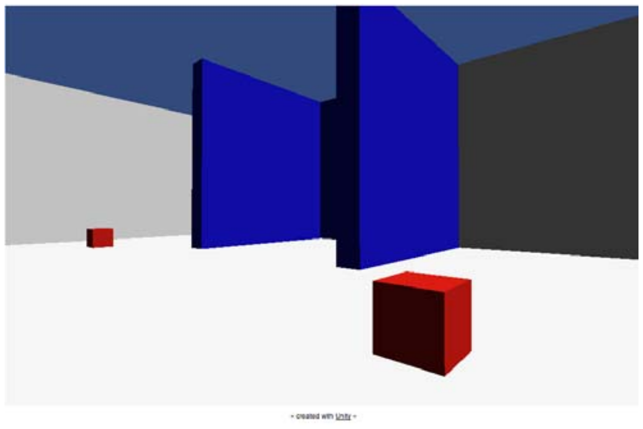


Fig 4. Screenshot of Unity playtest scene. This scene was generated from the same state shown in Fig 3.

quick comparisons can be made between two or more states. All states can be commented upon and organized by these comments as well, encouraging dialog between users. This means the client facilitates all three types of what Alex Games calls "dialogic interactions" [8]. Designers interface with the physical design of the game while also interacting with other designers and players. Dialog can take place as design iteration in addition to standard text discussion.

D. Design Roles

To further explore the potential for collaboration, students working on the same design can be assigned different roles. Should users desire to specialize in one particular aspect of the game design process, roles are just as easily assigned as in PRIME Designer, and since the visualization in the GDC is not entirely dependent on the design state, a template could be made that offered a completely different set of tools and visuals depending on user role. For example, an "artist" user's interface could include a rudimentary drawing program for texture generation. A "sound design" user's interface could include an interface to upload custom sound files to the game scenario. Roles offer a way for students to specialize on one particular aspect of game design, and roles can help distribute the students' efforts in an effective way.

The Game Design Client as a Game

We have described the GDC as it stands now, making full use of the already game-like system that CoSolve offers but not yet adding more. However, it is easily expanded in many different directions beyond the natural framing.

A. Design-Based Goals

Design-based goals could be tested for and recognized. A problem template could contain functionality to prompt users to "Make a game that uses all object types" or "Make a game that takes more than 5 minutes to complete" and recognize when these goals are met. These are explicit goals that can work in tandem with the personal goals users create for themselves discussed already. These goals can also encourage competition both within a group working on the same solving session and between separate groups working on different ones. This means teams themselves are also encouraged to collaborate. In a classroom setting, the client could be used for teams to compete against one another. Different goals, even when applied to the same initial design state, can be treated as different levels. Perhaps a team that meets a design goal is allowed to start a new design "level" based around a different, more complex goal.

B. Rewards as Buildup

Game object types or behavior could be introduced to users as they perform more basic operations, thus reducing the chance of overwhelming new users with too many choices. In addition to smoothing the learning process, this can also serve to present a reward or milestone to a student. Again coming back to goals, providing feedback about measured, in-game progress and making it clear to players is a powerful motivational tool in games today. If a student reaches a milestone such as "used all available object types in their game design" and as a result unlocks a new set of objects, we can assert that he or she has used a desirable set of tools and is now very likely to try out the newly presented ones.

C. Evaluation Functions

In a very similar vein, evaluation functions could be used to quantify game design quality. Evaluation functions have already been used in CoSolve in a variety of ways to measure state quality. Simple examples of measures for session quality include the depth and breadth of a solving-session tree. Students could be evaluated on the fraction of nodes they personally made compared to their group's total, or on how often they built on work done by others. Designs could be evaluated on how close they come to certain reference designs or how well they conform to established game genres, or on how complex their behaviors are. Since each node a in CoSolve session tree can be annotated by any user, that means that designs can be scored on the quantity or quality of dialogue, as represented in the annotations, that the users expressed during the process of coming up with the design. These metrics can also be a way for students to think critically about the inner workings of the GDC, as they may become curious about how exactly these values are being created. For when a student desires to know how the GDC works or even when they see a need to expand its current features, we have designed the GDC to accommodate these more ambitious users.

Template as Educational Tool

To make full use of the TGDC1 template and GDC client as a motivator to students for further exploring game-design theory and programming in general, we have structured the client to be malleable on several layers of increasing complexity and required knowledge.

A. Options Without Coding

Should a student desire finer control over the GDC behavior, most likely to give it more affordances, they can take some steps without having to write code directly. Objects can be designed and placed in Unity scenes without programming knowledge. While it is true that eventually one must start programming to have full control over the GDC and accompanying Unity scene, we see it as a tool to educate and motivate students just starting to explore the field of game design rather than a robust tool on its own.

B. Popular Language Makes Coding Transition Easier

If users do decide to start programming, there are still steps that can be taken to gradually explore the possibilities that are available. The GDC's JavaScript is a popular language for budding programmers, being featured in learning sites such as Codecademy and Code School. New game object types can be added to Unity and the GDC with minimal coding to change the types of games that can be created. The object types currently available support the generation of a puzzle style of game. The player-camera is itself an object type, and thus can be easily switched to a 3rd person camera. Trigger-zone object types could be added to create more scripted behavior, and hostile non-player characters (NPCs) could also be added to create a more action-oriented game. The Unity engine contains a large number of pre-built assets that can be made available for game designers and GDC designers to use as well. Should a student desire to add custom behavior to game objects, one of the scripting languages Unity offers is JavaScript, making the transition into scripting familiar and easier. It is not the goal of the project to create a complete game design tool that does not require any programming. Instead we want to present a simple tool whose inner workings are as accessible as possible for students that want to self-select for a greater challenge.

CONCLUSION

We have succeeded in making a more interactive game design client that maintains the current CoSolve collaborative state-space process model. With this interactivity, the process of using the GDC can itself be framed as a game, and take with it all of the motivational aspects inherent to games. The GDC offers a simplified view of game design, ideal for students just starting to explore the field, while also harboring the potential for augmentation should a student wish to explore further. While our current set of affordances for the game-design process has a limited range of functions at this time, new features are easily implemented by interested students of varying skill levels. Our educational goal is to produce students familiar enough with design and coding through their experiences with the GDC that they are better prepared to tackle game design and programming problems on an advanced level.

REFERENCES

[1] Claypool, K. and Claypool, M. “Teaching software engineering through game design,” in Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (ITiCSE '05), Caparica, Portugal, 2005, pp. 123–127.

[2] Scacchi, Walt. “Competitive game development: Software engineering as team sport,” Keynote talk delivered at 2nd International Workshop on Games and Software Engineering. Zurich. June 9, 2012. Powerpoint presentation slides at http://www.ics.uci.edu/~wscacchi/GameLab/GAS2012-Scacchi-Keynote.pdf

[3] Scacchi, Walt. “Free and open source development practices in the game community.” IEEE Software. 2004. 0740-7459/04. pp. 59-66.

[4] Fan, S. B., Robison, T., and Tanimoto, S. L. “CoSolve: A system for engaging users in computer-supported collaborative problem solving,” Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing. 1-3 Oct. 2012, pp.205-212.

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

[6] Robison, T. Opening up the Collaborative Problem-Solving Process to Solvers. Ph.D. dissertation, Dept. of Computer Science and Engineering, University of Washington, Seattle, 2012.

[7] Tanimoto, S. L., Fan, S. B., and Robison, T. “A game-building environment for research in collaborative design,” Proc. 2009 IEEE Symposium on Computational Intelligence and Games. Sept. 7-10, 2009, Milan, Italy, pp.96-103.

[8] Games, Ivan A. “Three dialogs: A framework for the analysis and assessment of twenty-first-century literacy practices, and its use in the context of game design within *Gamestar Mechanic*”. E-Learning Volume 5 Number 4, 2008. pp.396-417.