Novice-friendly Authoring of Plan-based Interactive Storyboards

James Skorupski and Michael Mateas

Expressive Intelligence Studio
University of California, Santa Cruz
{jskorups, michaelme}@soe.ucsc.edu

Abstract

Story Canvas is a visual authoring tool for the creation of interactive, generative stories. Aimed at authors without a technical background in computational storytelling, our system takes an existing author goal-based narrative planning architecture and adds a highly visual authoring and reading interface to the technology, using the language of storyboards and comics as a framework for both authoring and interacting with the resulting narratives. In this paper we describe Story Canvas and its evolution from our previous authoring work, including how our interface choices have been driven by our previous experiences with non-technical authors, and describe the details of translating the visual authoring constructs into story plans within the story generator.

Introduction

The authoring of compelling interactive and generative stories traditionally demands expertise in computational models of story structure and execution, as well as the background knowledge to formulate compelling plot arcs, rich dialog, character conflicts, and other story elements. The rarity of individuals with this cross-disciplinary experience in the relevant technical and creative backgrounds motivates our work in this area. To address this scarcity, we have created Story Canvas, a novice-friendly authoring tool for interactive, generative stories that presents a visual authoring and reading interface based on the spatio-temporal language of storyboards and comics. It is a major evolution of our previous interactive story authoring system, Wide Ruled, and is designed in many ways as a response to our experiences with that system in our multiple class room evaluation sessions (Skorupski 2009).

Our underlying story planner is based on the UNIVERSE story model introduced by Michael Lebowitz in 1985, in which he described an HTN-style model of story structure and execution based on hierarchically-arranged author goals that represent the story intentions of an author, and plot fragments (tasks) that consist of ordered steps (including subgoaling) to accomplish goals (1985). Wide Ruled, our previous work, is a UNIVERSE-based interactive textual story authoring system utilizing a traditional GUI complimented by non-technical narrative terminology, natural-language descriptions of technical plan components, and step-by-step guidance for the more complex tasks (Skorupski 2007, 2009)[1]. While it allows many users with little or no programming experience ("non-technical" or "novice" authors) to create interesting stories, it still suffers from a number of conceptual hurdles for these authors. Creating complex precondition constraints, binding and referencing data variables, and managing large story hierarchies have all proven to be troublesome in previous evaluations of Wide Ruled. Story Canvas addresses these difficulties by abstracting away from the underlying story planning model and handling some of these complexities automatically. It introduces a richer, visual method of storyboard-based storytelling and provides a visual interactive representation of the high level story structure that we hope will allow larger audience of novice authors to utilize the power of our computational storytelling model. Story Canvas generates stories in the reactive planning language ABL (Mateas & Stern 2002). While our system utilizes only a subset of the entire ABL language, the implicitly concurrent capabilities of the planner allowed us to easily implement interactive features of the story model, and allow for future enhancements of the underlying story model such as fully concurrent story plans, and integration into external game engines (McCoy et al 2008, Weber et al 2010). In this paper we introduce the Story Canvas interface, describe how its interface components and features are motivated by our previous experience with the Wide Ruled story authoring environment, and describe our novel method of translation from the storyboard interface to the underlying reactive planning code for the story generator.

Related Work

The motivation behind our chosen storyboard model of interaction is based on previous analysis of comics and

This material is based upon work supported by the National Science Foundation under Grant No. 0747522.

[1] Wide Ruled is freely available at: http://eis.ucsc.edu/Wide_Ruled

storyboards, and the success of domain-specific visual programming languages. The sequential art of storyboards have been an extremely effective spatio-temporal visualization technique for films, comics, graphic novels, computer animation, and game design. It has been the subject of extensive analysis (Eisner 1985, McCloud 1993), and provides a background for the design of our authoring system and the visual metaphors used to interact with the underlying model. With Story Canvas, we have moved the author further away from the underlying code and structured editor style of our previous work and stepped into the visual domain. While general visual programming has traditionally been argued as ineffective and inefficient for large and complex systems (Whitley 1997), domain-specific visual languages have been successful in many practical instances. These include the sound processing and music synthesis tools such Max/MSP and Pure Data, and user interface builders in programs like Eclipse and Visual Basic/C++ . Closer to our domain of interactive media, systems such as Storytelling Alice and Game Maker[2] have successfully provided visually enriched game authoring experiences, but require some programming of textual scripts to create complete games (Kelleher et al 2007). Kodu, introduced in 2009, is a successful and completely visual game programming environment, but it is limited to a uniform model of physical movement and interaction, without support for the abstract computational structures and alternative non-physical interactions often desired in interactive narratives (MacLaurin 2009). Storyboards and comics are not a new interface metaphor, and have previously been used to automatically visualize generated story plans from domains authored in a non-visual fashion (Jhala et al 2008, Pizzi et al 2008). From the space of comic-based programming work, previous research by Fernaueus and Kindborg et al discusses graphical rewrite rules as well as more complex general programming constructs represented in comic form (2006, 2007). While this work focuses on purely graphical modifications of visual objects, our system combines the explicit visualization of story objects with representations of abstract logic and computational constructs specific to the domain of UNIVERSE stories. Gingold's *Comic Book Dollhouse* used author-created comic strips representing static story graphs and used simple graphical triggers to allow reader navigation of the graph (2003). Our work aims to combine the flexibility of a general visual programming language and with the simplicity and resilience of a structured editor by utilizing the specific domain of UNVERSE stories to inform our visual design and user interaction, and relying on this restricted domain to avoid the potential visual clutter and scaling pitfalls of general visual programming.

Story Model

The story generation model in our system is based on the HTN-style UNIVERSE model of story generation (Skorupski 2009). It models the story structure (planning domain) as a set of hierarchical plans that encompass one or more ways to accomplish a story goal for the author. Like Wide Ruled, a Story Canvas story contains a set of author-created story objects, represented as "Characters" or "Environments", each with associated attribute-value trait pairs, and relationships to other story objects, along with a strength value for each relationship. Stories also contain a set of "Plot Point Types", which define types of episodic attribute-value data stores which are then instantiated as "Plot Points" and utilized only during the story generation process. "Author Goals", with optional story object parameters, are the primary unit of story planning in this tool, each containing one or more "Plot Fragments" that describes a set of actions that fulfill its parent author goal. Plot fragments have precondition constraints that must all be true before execution. These constraints rely on the current state of the story world during generation and can also bind story objects (characters, environments, and plot points) and their attributes for later use in that same plot fragment. Additionally, plot fragments contain a set of sequential "Story Actions" that can modify the story world during execution. These actions can modify story characters, environments, and plot point instances captured in the precondition, create and delete plot point instances, calculate new values, and output parameterized text (in the form of narration, speech and thought bubbles).

Story generation in Story Canvas begins with a top-level initial author goal, which randomly selects amongst all executable plot fragments with valid preconditions, and then sequentially executes all of its contained story actions to successfully complete a story. If the generator encounters a story action that pursues another author goal, the process repeats and a new plot fragment is selected to accomplish the subgoal. The relationship between author goals and their associated plot fragments describes a potential tree-like space of stories, in which a single generated story is represented as a traversal of the tree, as seen in figure 2. The result of this story generation process is a single story instance among many potential stories, represented in Story Canvas as a series of storyboard panes visualized on the screen, as is seen in Figure 5. Readers of the story interact with these visual stories using a modified form of Wide Ruled's interactivity model, which involves author-specific interactive actions that trigger interleaved goals at any time during generation (discussed later).

The Story Canvas-ABL Connection

Each story world in our system is represented by an ABL program that contains story object information along with hierarchical behaviors that mirror the relationship between author goals and plot fragments in the Story Canvas story model. In our case, a single story world represents a single ABL "agent" that is executing a story plan while interacting with the interface to the user. The author goals and plot fragments are translated into ABL behaviors, and the initial story object information is translated into types and instances of ABL Working Memory Elements (or

[2] http://www.yoyogames.com/make

WME's) (Mateas & Stern 2002). Each time the author makes changes in the storyboard interface, Story Canvas generates updated code with the corresponding changes. During generation, this ABL agent is launched, and proceeds to asynchronously communicate with our storyboard interface, transmitting user interactions to the planner, and receiving information about the resulting story plan as it is generated. Figure 1 depicts the architecture of these translation and generation steps. In this diagram, the "ProxyBot" element is an asynchronous arbitration layer than handles communication with the Story Canvas interface. The generation architecture is similar to that of the ABL Wargus and ABL Starcraft project architectures (McCoy et al 2008, Weber et al 2010).

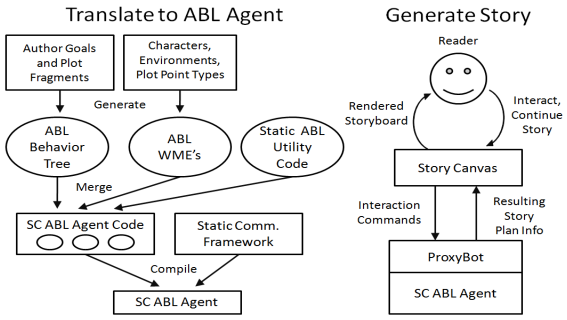

Figure 1. The Story Canvas planning architecture, depicting the translation from the story model to ABL plans, and the generation of a story instance.

Authoring in Story Canvas

Constructing a Story Canvas story world requires the author to create a set of objects upon which the story structure acts (characters, environments, plot point types), as well as the story structure itself (author goals and plot fragments). Here, we will focus on the two most complex aspects of the interface: authoring the story hierarchy, and creating plot fragments. In the following sections, we will describe the interface for authoring each component, how this interface has evolved from its equivalent in Wide Ruled as a response to our previous lessons learned, and finally, how this feature is implemented in the ABL planner.

The Author Goal and Plot Fragment Hierarchy

The interface for viewing and modifying the goal-fragment story space is shown in figure 2, which depicts a sample murder mystery story in the Story Canvas interface. Goals are represented by dotted outlines around groups of labeled plot fragments. This hierarchical layout is automatically generated and adjusted as new author goals and fragments are created and old ones are deleted by the author. In order to deal with potentially very large story trees, the interface can be panned and zoomed across the hierarchy. Here the author can create, delete, and edit goals, arrange fragments, and create empty fragments and delete existing ones. The editing of individual plot fragments is more complex and described in the next section. It is important to note that this story hierarchy is not a strict direct acyclic graph. Plot fragments can subgoal author goals recursively, represented by the dotted arrow connecting two author goals in the tree. In addition, a goal can be subgoaled by more than one plot fragment throughout the story, and therefore appear in multiple places in the hierarchy.

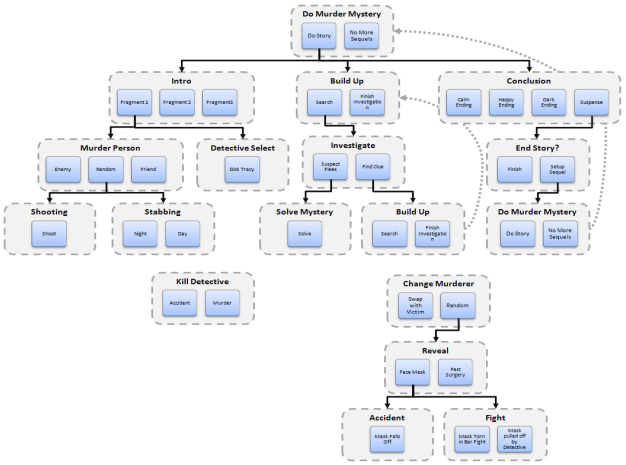

Figure 2. The Story Canvas story structure interface. Groups of plot fragments are contained within author goals, and each plot fragment can be connected to one or more author goals, represented by edges in the hierarchy.

Lessons learned: visualizing large story spaces. Wide Ruled used a simple hierarchical list to depict its story structure, which proved to be unwieldy for large story spaces. The move to a zoomable, pannable visual hierarchy interface allows authors to deal with much larger story spaces. A story world in our system can be disconnected into multiple hierarchie: a single author goal is designated as the start goal (in figure 2, the top-most goal is the start goal) before story generation begins, but independent story trees may exist in the story world at the discretion of the author. This interface makes it easy to spot distinct hierarchies (see the two small hierarchies at the bottom of figure 2). Multiple hierarchies can be a remnant of an incompletely-authored story or a failure to include an appropriate author goal in a plot fragment (a bug), or they may be intended for activation during generation by an interactive action specified by the author.

Implementation: From story trees to behaviors. Story Canvas goals and fragments are converted into ABL behaviors for execution. All fragments for a given author goal are converted into multiple behaviors with the same method signature, and ABL's behavior arbitration

```
sequential behavior MakeDrama () {
 precondition { }
 subgoal Introduction();
 subgoal TheEncounter();
 subgoal Conflict();
 subgoal TheFight();
 subgoal LoversRevealed();
 subgoal ChooseLoverOrHusband();
 subgoal Conclusion();
}
...
```

Figure 3. A plot fragment for the initial author goal "MakeDrama" for our example story, in (abridged) ABL code.

mechanism analyzes preconditions and automatically selects a valid plot fragment during story generation (Mateas & Stern 2002). Figure 3 shows the ABL-encoded example of a single plot fragment for the author goal `MakeDrama` in an example story. Here, we have no precondition, and sequentially execute seven author goals in order to complete the story. These author goals may have plot fragments that in turn call other author goals, resulting in our hierarchical structure.

Plot Fragments

Story Canvas presents a plot fragment as a set of story panes for both the precondition and the story actions, and hides some of the complex computational constructs that caused problems for non-technical authors in our previous evaluations of Wide Ruled. Figure 4 shows a sample plot fragment being edited in Story Canvas. Precondition constraints are contained in the left-most pane, and the rest of the panes contain story actions. Below each story pane is a smaller pane, which contains abstract, non-visual constraints and story actions, that affect elements that will not be visible to a reader of the resulting story (plot point constraints and modifications, and calculations of new values are located in these panes). If an action pane only has non-visible actions, then it will not be displayed during story generation. As a plot fragment is created, the author can zoom and pan in this interface, resizing elements at will to fit into each story pane. While there is always a single precondition pane, there can be any number of story action panes in a single plot fragment.

Plot Fragment: Authoring Preconditions

During generation of a story in Story Canvas, the precondition for each plot fragment is a list of requirements that determines whether it is eligible to be selected as a possible way to complete an author goal. These requirements are a list of constraints on the traits and relationships within characters, environments, and plot points, and every constraint must be true simultaneously for the plot fragment to be valid and ready to use within a story. In addition to providing an eligibility test, preconditions bind characters, environments, and plot points to local variables for use and modification within the plot fragment.

As seen in figure 4, these constraints are represented as a graph of character and environment icons. Because preconditions are meant to capture characters, environments, or plot points, which are dynamically selected during story generation, their appearance is not known during the time of authoring. These "blank" icons therefore represent unknown, potential story objects, and correspond to objects placed into the story action panes on the right. Intra-object constraints on individual objects, such as "age > 19" or "name != John" are contained within the small box hovering next to each icon. Inter-object constraints, which relate two dynamically bound objects, are represented by an arrow between the icons, showing that those two objects have an inter-dependency

Lessons learned: eliminating temporary variable bindings. This process of naming and referencing a variables within a plot fragment was a major problem for non-technical users authoring stories in Wide Ruled (Skorupski 2009). The graph-based approach constraint authoring approach avoids the tedious and confusing management of variable name bindings and references. We have eliminated explicit name management by visually linking story objects to the bindings in the constraint graph.

Implementation: From graphs and text boxes to ABL preconditions. Figure 5 shows the ABL code that corresponds to the plot fragment depicted in Figure 3. The precondition block in figure 5 depicts the binding of character, environment, and plot point information stored in various WME objects. Within ABL, binding a field to a local variable is represented by the `::` symbol, while standard java-style comparison operators are used to compare fields to variables and literals (Mateas & Stern 2002). The management of variable names is completely hidden from the author. Rather, constraints are depicted as graph relationships between objects in the authoring interface.

Plot Fragment: Authoring Story Actions

Story action panes are depicted to the right of the precondition pane in figure 4. Actions can subgoal author goals, modify objects captured in the precondition, create/delete plot points, calculate new values, display static or captured objects with varying poses and composition, and output parameterized text in the form of narration blocks, speech bubbles, or thought bubbles. Modification of the traits of characters/environments is

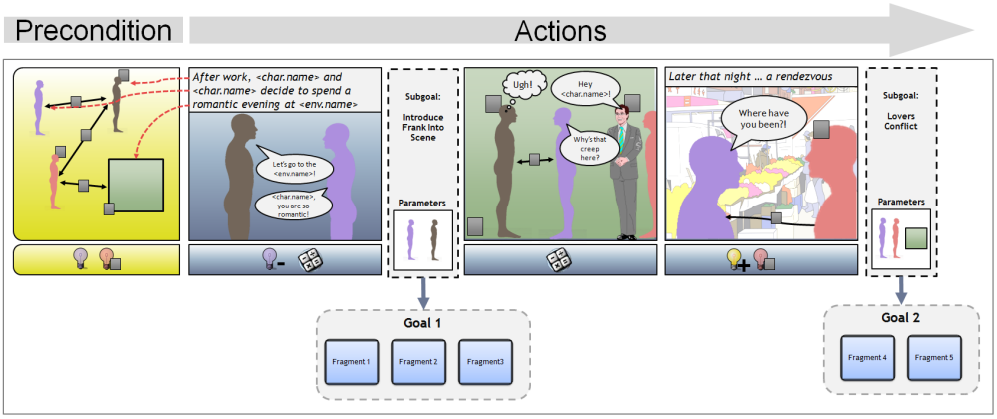

Figure 4. The Story Canvas plot fragment editor. Silhouetted objects represent dynamically-bound story objects, and images represent static elements.

```
sequential behavior TheEncounter () {
 precondition {
  (CharWME id::chID1 Name::name1)
  (CharWME id::chID2 Name::name2 Husband==chID1)
  (CharWME id==chID1 Friend==chID2)
  (CharWME id::chID3 Name::name3 Loves==chID2)
  (CharWME id==chID2 SecretlyLove==chID3
    FavPlace::place)
  (EnvWME id::envID1 Name==place Description::desc
    daytime==true)
  tensWME = (TensionPPWME DramaLevel > 0.5)
  progWME = (ProgressPPWME HusbandSuspects == false)
 }
 ...
}
```

Figure 5. The (abridged) ABL code generated for the precondition story pane in figure 4.

shown as a small box hovering next to the unknown icon of a captured object or static image of pre-selected object. Editing of relationships between characters is depicted as a graphical link between avatars, similar to the inter-object constraints in the precondition pane. Plot points are modified (indicated by small box), created (indicated by a plus sign), and deleted (indicated a minus sign) in the bottom non-visible pane, along with the creation of calculated values. Like in the precondition pane, dynamically bound characters are displayed as colored silhouetted icons, and statically chosen objects are rendered with their associated image directly in the authoring interface. Parameterized text can be displayed as a narration block (as seen in the first and last story panes in figure 4), a thought bubble, or a speech bubble. The locations, sizes, and orientations of captured and static story objects, narration, speech and thought bubbles, as well as text size, are customizable by the author, to allow for varied and interesting compositions.

In Wide Ruled, story actions were strictly ordered lists of actions. In the domain of storyboards, actions exist on a 2D plane, and may not clearly reveal the underlying linear ordering of the actions in the ABL story plan, which can affect the resulting displayed information. In order to deal with this ordering ambiguity, Story Canvas enforces the rule that parameterized text, calculations, and modifications to objects may only reference information

```
sequential behavior TheEncounter () {
 precondition {...}
 mental_act {
   workingMemory.delete(tensWME);
   double newWifeVal =
     workingMemory.get(chID2).getWifeVal() – 0.1;
 }
 act showStoryPane(1, chID1, chID2, name1, name2, envName);
 subgoal IntroduceFrank(chID1, chID2);
 mental_act {
   workingMemory.get(envID1).setVisited(true);
   workingMemory.get(chID2).setSuspicious(true);
   workingMemory.get(chID2).setWifeVal(newWifeVal);
   workingMemory.get(frankID).setConfusion(2.0);
   double newSecLove =
     workingMemory.get(chID1).getSecretLoveVal() – 0.1;

 }
 act showStoryPane(2, chID1, chID2, chID3,envID1, name1);
 mental_act {
  workingMemory.get(chID1).setSecretLoveVal(newSecLove);
  workingMemory.add(new SuspicionPPWME(chID3));
  progWME.setHusbandSuspects(true);
 }
 act showStoryPane(3, chID1, chID3);
 subgoal loversConflict(chID1, chID2, envID1);
}
```

Figure 6. The (abridged) ABL code generated for the story action panes in Figure 4.

contained within previous panes. This is seen in figures 6 and 7, where new trait values are calculated before their use in the next pane. Subgoaling in a plot fragment is displayed as a narrow story pane with a dotted outline containing an author goal name and the story objects to be passed as parameters to that author goal. The author goal to be subgoaled, along with its contained plot fragments are shown on screen below the current plot fragment for quick access to the editing views for those elements.

Lessons learned: Hiding named variable creation and reference. Like in the precondition pane, we hide variable bindings and references within the story action panes to simplify the authoring process. Whenever a story action item needs to use the information stored in a captured story object (or value created in a previous pane), the user double clicks on the object or calculation to be edited and they can then select from a list of available information: previously calculated values, any trait or relationship value within previously captured or used story objects, or any attribute of a captured plot point. No variable names are generated – a user selects the required piece of data from a list of icons and attributes names that appear during editing and a this connection is then visually represented as an arrow to the referenced object or calculation. These arrows, shown stemming from the narration text box in the first action pane of the fragment in figure 4, are only visible when an object/calculation is selected, to avoid cluttering.

Implementation: From story panes to ABL plans and back. Plot fragments are translated into ABL behavior code like that shown in figure 6. A single behavior encompasses all the story panes in a plot fragment. In the code sample, the `mental_act` construct is a block of raw java code that we use to modify the working memory state of the objects in our story and calculate new values. All the actions for a single pane are executed in the order they were created, followed by a command (or "primitive action" preceded by the `act` keyword) called `showStoryPane(…)` that gathers and passes along the dynamically bound story plan information relevant to the corresponding graphical storyboard pane to the Story Canvas interface. This command immediately sends the bound plan information to the interface which accesses stored assets (story object graphics and visual composition information) that are rendered to the screen for the reader.

Reading and Interactivity

The reading and interaction interface for a generated story is very similar to the authoring interface, as shown in figure 7. The plot fragment shown in figure 3 is displayed to the reader, with unknown icons filled in with characters and environments, and object modification, information reference arrows, plot point modifications, and calculation actions hidden from view. This interface displays a continuous comic scrolling from left to right until the story is finished, using the arrow buttons to control the generation pace and browse previous panes. Story Canvas implements an asynchronous goal execution model of interaction, in which story authors specify a set of

"Interactive Actions" for each plot fragment, which can be executed at any time during the execution of that fragment. These actions are separate goals, which can modify the story world and output completely new panes to the screen. During generation, when an interactive action is activated, the current fragment is halted, and the interactive action is executed completely, allowing for a dynamically interleaved story with full authorial control over what a reader can do at different times during generation.

Lessons learned: Contextually-relevant interactive actions., Wide Ruled had a similar interactivity model with a global set of interactive actions. User feedback indicated that creating global interactions that make sense at every point in a story is difficult, so Story Canvas allows the author to associate interactive actions with specific plot fragments (Skorupski 2009). Those actions only appear during the execution of their associated fragments.

Implementation: Reacting to input. At its core, the ABL reactive planner is designed to execute concurrent plans and interact with ever-changing inputs from a dynamic external "world". In our case, this "world" is the reader interacting with our storyboard interface. By using the "daemon behavior" design pattern, described by Weber et al, we are easily able to constantly monitor for input from the interface with a parallel behavior, and interrupt execution of the current story plan at any point in time to wait on a user's command or to process a request to activate an interactive action (2010).

Future Work and Conclusions

We have shown here the evolution of our latest story authoring interface, how it has evolved according to the evaluation of our previous work, and the technical underpinnings that drive its capabilities. The next steps in our work will involve evaluating our system with a new set of non-technical authors, comparing the complexity and quality of stories created with Story Canvas with those made with Wide Ruled. From a technical perspective, we will implement reactive, intelligent story analysis features that give constant feedback during the authoring process to visualize the coverage of the potential story space, and locate areas of disuse, or infinite recursion. We will also implement additional comic book storytelling techniques, such as varied story pane size, arrangement, and ordering, line drawing styles and variations of shading. Finally, the ultimate goal for this work is to allow a user to author real-time 2D or 3D interactive game experiences. These further developments will expand the creative power and application of the system by giving intelligent, active feedback on the dynamic structure of a potential stories, by embracing a larger set of visual techniques from a proven and effective storytelling medium, and ultimately by applying these methods to author richer, more dynamic types of interactive story experiences.

Figure 7. The Story Canvas reading and interaction interface for the plot fragment in figure 4.

References

Eisner, W. 1985. Comics and Sequential Art. Tamarac, FL: Poorhouse Press.

Fernaueus, Y., Kindborg, M., Scholz, R. 2006. Rethinking Children's Programming with Contextual Signs. In *Proceedings IDC 06*, Tampere,Finland.

Gingold, C. 2003. Miniature Gardens & Magic Crayons: Games, Spaces, & Worlds. Master's Thesis. School of Literature, Communication and Culture, Georgia Institute of Technology.

Jhala, A., Rawls, C., and Young, R. M. 2008. Longboard: A Sketch Based Intelligent Storyboarding Tool for Creating Machinima. In *Proceedings of FLAIRS '08*, pp. 386--391.

Kelleher, C., Pausch, R., and Kiesler, S. 2007. Storytelling alice motivates middle school girls to learn computer programming. *In Proceedings of CHI '07*. pp 1455-1464.

Kindborg, M. and McGee, K. 2007. Visual programming with analogical representations: Inspirations from a semiotic analysis of comics. J. Vis. Lang. Comp.pp99-125.

Lebowitz, M. 1985. Story Telling as Planning and Learning. Poetics 14, pp. 483-502.

MacLaurin, M. 2009. Kodu: end-user programming and design for games. In *Proceedings of FDG '09*. Orlando, Florida, April 26 - 30, 2009.

Mateas, M., and Stern, A. 2002. A Behavior Language for Story-Based Believable Agents. IEEE Intelligent Systems 17(4):39–47.

McCloud, S. 1993. Understanding Comics. New York, NY: Kitchen Sink Press/Harper Perennial.

McCoy, J, and Mateas, M. 2008. An Integrated Agent for Playing Real-Time Strategy Games. In *Proceedings of AAAI*. AAAI Press, 2008, pp.1313–1318.

Pizzi, D.; Cavazza, M.; Whittaker, A.; & Lugrin J-L. 2008. Automatic Generation of Game Level Solutions as Storyboards. In Proceedings *of AIIDE '08*, pp. 96–101

Skorupski, J, and Mateas, M. 2009. Interactive Story Generation for Writers: Lessons Learned from the Wide Ruled Authoring Tool. In *Proceedings of DAC '09*, Irvine, CA.

Skorupski, J; Jayapalan, L; Marquez, S; & Mateas, M. 2007. Wide Ruled: A Friendly Interface to Author-Goal Based Story Generation. In *Proceedings of ICVS '07*. pp. 26-37.

Weber, B.; Mawhorter, P.; Mateas, M.; and Jhala, A. 2010. Reactive Planning Idioms for Multi-Scale Game AI. To appear in *Proceedings of the IEEE CIG '10*.

Whitley, K. N. and Blackwell, A. F. 1997. Visual programming: the outlook from academia and industry. In *Proceedings of ESP '97*. pp.180-208.