M A R C D A V I S
www.marcdavis.me

P U B L I C A T I O N S
info@marcdavis.me

Programming with Characters

Bibliographic Reference:

Marc Davis and Michael Travers. "Programming with Characters." In: *Proceedings of the First International Conference on Intelligent User Interfaces in Orlando, Florida*, ed. Wayne D. Gray, et al., ACM Press, 269-272, 1993.

PROGRAMMING WITH CHARACTERS

Michael Travers & Marc Davis

MIT Media Laboratory
20 Ames Street
Cambridge, MA 02139
(617) 253-0608
mt or mdavis@media.mit.edu

INTRODUCTION

Programs are hard to build, and even harder to understand after they are built. We lack intuitive interfaces for visualizing and manipulating many parts of programs and the ways in which these parts interact. Constraint systems have addressed these problems. We generalize some of the notions inherent in constraint systems to agent-based systems, and explore the use of animated characters as interface representations of agents. In particular, conflict detection and resolution is dramatized by the use of characters and their emotions. The history of their interactions is presented as a narrative using video and storyboard techniques. Building programs out of agents and enabling users to manipulate program parts by interacting with simple animated characters can aid relatively unskilled users in understanding and modifying complex systems.

FROM CONSTRAINTS TO AGENTS

We chose to explore the question of whether agents could be used as the basis of an interactive design and programming environment. Most direct manipulation environments (i.e., MacDraw) are easy to use, but lack intelligence and flexibility. The user cannot extend the system beyond the simple operations provided and the system cannot learn new methods or tasks.

These issues have been addressed by interactive constraint-based systems such as SketchPad [4] and ThingLab [1]. Constraints are computational objects which combine *declarative* and *procedural* characteristics. The declarative part of a constraint specifies a condition that the constraint will attempt to maintain (e.g., "The button width is greater than the button text length."), while the procedural part specifies techniques for bringing about the given state (e.g., "Move the right edge of the button until the button width is greater than the button text length."). The major computational issue for constraint systems is how to resolve conflicts involving multiple constraints. SketchPad used numerical relaxation for this purpose, while ThingLab used a combination of planning techniques and relaxation. A further problem of the constraint-based approach is the difficulty of visualizing constraints and their interrelations so that they can be understood and altered by the user.

To appear in the proceedings of the 1993 International Workshop on Intelligent User Interfaces

We chose to investigate a new technique which, like constraints, makes use of the power of combining declarative and procedural functionality in an interactive network, but which offers a new approach to the above problems. We represent programs as a collection of *agents*, which are simple, individual mechanisms that accomplish a particular task. The notion of agent derives from Minsky's usage in *Society of Mind* [3], which envisions the mind as a network of interacting "mindless" parts. An agent has goals and methods for achieving its goals, which may rely on other agents achieving their goals. Constraints may be understood as a specialized type of agent.

The ability to learn and to adapt to new situations enables an agent-based approach to resolve conflicts among multiple agents. We have explored a strategy which utilizes insights from Minsky's theory as well as some aspects of case-based learning. If two or more agents are in conflict, the agents involved remain unchanged; instead, a new supervisor agent is created which knows how to manage the agents involved in the particular conflict situation as well as any new agents which may have been created through user intervention. This strategy is based on Minsky's notion that one learns, not by debugging old agents, but by adding new agents that know *when* the old agents are applicable and when not. The concrete example of the conflict and its resolution is stored in a case library to which the supervisor agent refers in managing its supervisees. As the case library grows, supervisor agents are able to find resolutions to new conflict situations by referring to similar situations stored in the case library.

Reconceptualizing computational processes in terms of agents also facilitates the design and visualization of complex interactive systems. As agents ourselves, we bring to programming and to human-computer interaction a powerful cognitive and affective framework for dealing with other agents. Our assumptions about how things with

agency behave, interact, and grow can be put to use in designing more intuitive systems.

FROM AGENTS TO CHARACTERS

Our notion of agents is related to but somewhat divergent from recent work on "interface agents" [2]. An interface agent is an intelligent intermediary between a user and a computer system, often presented as a video image of a person or animated character. It is an "agent" in the sense of a travel agent who acts on behalf of the user. Our agents, on the other hand, have their own goals (which, to be sure, derive from that of the user or system designers). Rather than acting as intermediaries between the user and a computational environment, in our approach, the network of agents constitutes the underlying computational environment itself.

How should agents be presented to the user? We are investigating the use of cartoon characters as a metaphor for computational agents. Unlike "interface agents," which represent the user to the computational environment, our characters represent agents (which make up the computational environment) to the user. The stereotyped actions and general lack of intelligence in agents suggest that cartoon characters are a better interface representation than more human-like characters.

However, we are still exploring the relationship between agents and the characters that represent them. If a character represents just one agent, it might seem too stupid even for a cartoon. In our current implementation, we use a single cartoon character to represent a collection of agents that all work towards a common goal. On the other hand, this representation will make it difficult for the user to view interactions between those agents. One flexible solution would be to allow the user to recursively peer into the heads of characters, which might show multiple, smaller, stupider characters working within.

The simplicity and predictability of cartoon characters, as well as their affective appeal, make them well suited to the construction of narrative scenarios for explaining the interactions of agents and for resolving conflicts between them. We currently use an interactive storyboard (see below) which makes use of the user's understanding of narrative and comic-strip conventions in visualizing conflicts between agents and facilitating conflict resolution. Clicking on a character plays its video clip which expresses the emotional state the character has in relation to the actions it took at that point in the story. Representing conflicts between agents by means of character and story is a fortunate match because much of our narrative comprehension focuses on the recognition and resolution of conflicts. With the storyboard, users can understand the origin of a conflict situation by means of a video story, whose happy ending they can create by interactively teaching the characters new skills which enable them to resolve their conflict.

SCENARIO: INTERFACE TOOLS DESIGNER

We have been exploring our ideas about agents and characters within the domain of the Macintosh Common Lisp Interface Tools Designer (IFT). IFT is a direct-manipulation, graphical environment for creating interface objects (dialogs, menus, buttons, etc.) in Macintosh Common Lisp. An example of designing a dialog box and buttons in IFT is shown in Figure 1.

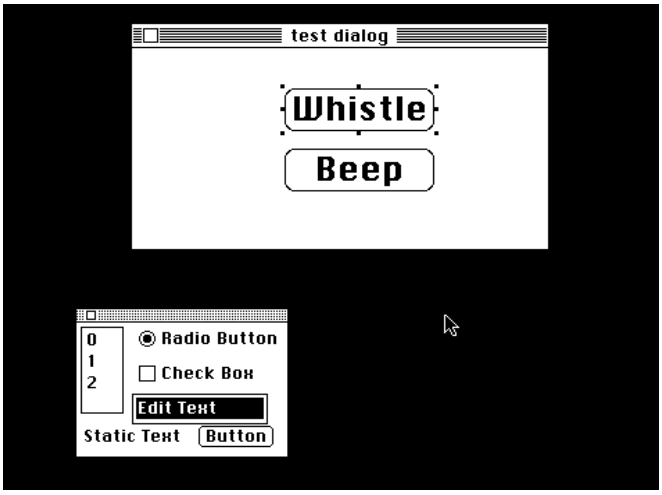

Fig. 1: Macintosh Common Lisp Interface Tools Designer

Our system augments IFT with agents that perform tasks such as keeping objects aligned or sized to fit their contents. In our scenario these agents form two characters: Ren and Stimpy. We borrow the Ren and Stimpy characters from the popular cartoon show of the same name which airs on the Nickelodeon cable television channel. Ren and Stimpy are cartoon characters well suited to the representation of agent conflict: they are simple characters with singular driving passions (Ren is choleric and Stimpy sanguine) whose lives are a series of conflicts and reconciliations. In the world of IFT, Ren is the character who cares about the alignment of objects (see Figure 2), while Stimpy is concerned about making sure that containing objects accommodate the objects they contain (see Figure 3).

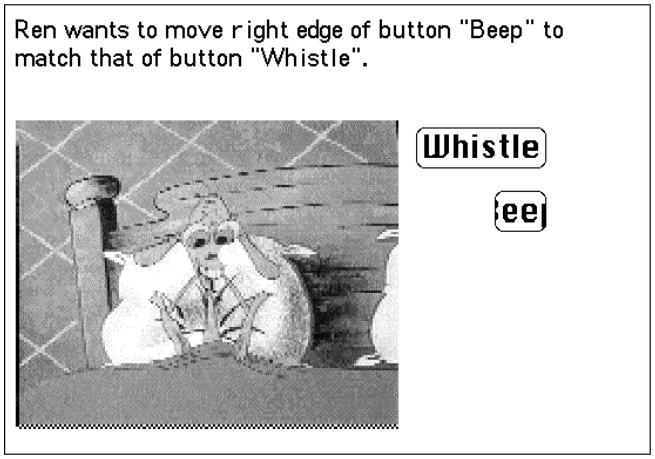

Fig. 2: “Ren”

In our example, Ren and Stimpy come into conflict over two buttons, "Whistle" and "Beep." Through manipulating the buttons Whistle and Beep in the Interface Tools Designer, the user brings about a situation in which the methods which Ren has for accomplishing his goal (in this case the right alignment of button Whistle and button Beep) and the methods which Stimpy has for accomplishing his goal (in this case making sure that the button Beep is large enough to fit its button text) come into conflict. Once the conflict is detected, a video storyboard is created which presents the history of the conflict to the user (see Figure 5). By clicking on the frames of the storyboard QuickTime movies are played which express the emotional states of Ren and Stimpy; the captions for each frame explain Ren and Stimpy's motivations and the dilemma they have gotten into. Thumbnails of the disputed objects depict key phases in the progression of the conflict. Thus the story of Ren and Stimpy's conflict is presented to the user.

After having played the video storyboard through, the user selects a character to interact with in order to solve the narrative conflict. The user trains the selected character in a

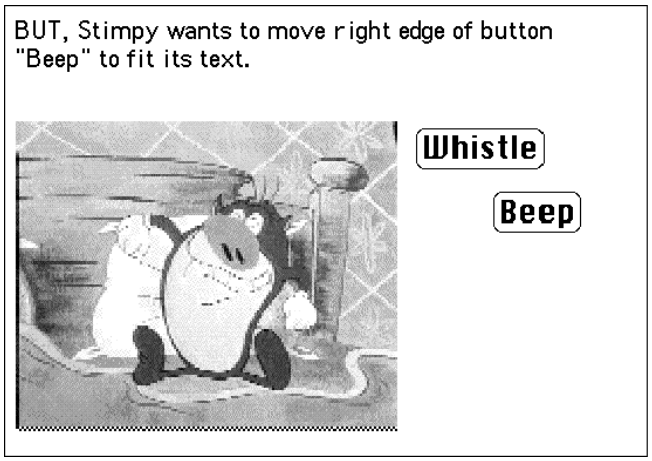

Fig. 3: “Stimpy”

new skill which enables the characters to avoid the conflict. This training takes place through direct manipulation of the objects involved in the conflict between the characters. The user offers a solution by showing the selected character what it should do in order to avoid the conflict situation. The character then responds to the user in text explaining its understanding of the proposed solution which the user confirms if correct. Behind the scenes, this process of interactive debugging results on the one hand, in the creation of a new agent within the character (i.e. a new goal-method pair which in effect adds a new method to the character for achieving the character's goal), and on the other hand, in the storage of this conflict resolution within the case library. When the system encounters this (or a similar) situation again, the conflict manager refers to the case library in order to manage which agents in the relevant characters get activated so as to avoid the potential conflict situation.

The system architecture (Fig 4) thus includes multiple characters, each of which can have multiple agents (but all must share the same goal), and a central conflict manager with associated case library.

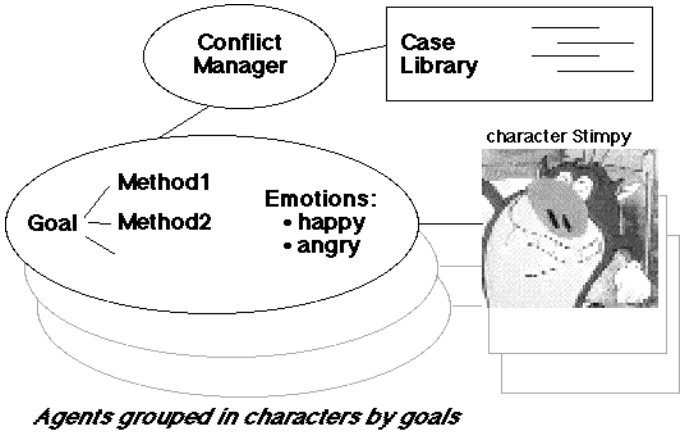

Fig. 4: System architecture with detail of one character

After having resolved the conflict situation, the user sees an expanded video storyboard showing in one frame an ebullient character pleased at having found a solution to the conflict and in the next frame a raucous dance and song, entitled "Happy Happy Joy Joy," celebrating the reconciliation of Ren and Stimpy. The conflict resolved, the story over, the video storyboard departs until the next time that Ren and Stimpy get into a conflict which must be visualized to the user so that through "narrative debugging" the user can program the system's characters to better meet the user's needs.

FUTURE WORK

We are currently exploring extensions and improvements to our existing system in the following areas: increasing the range and complexity of narratives which depict the interaction, conflict, and resolution between characters;

supporting better integration between the video storyboard representation and the actual objects of characters' concern (e.g. showing characters manipulating and fighting over computational and interface objects); and improving the selection mechanism for video clips to allow automatic retrieval of relevant segments. These and related issues are being explored by the authors in the context of ongoing doctoral research.

CONCLUSIONS

Our prototype points toward future systems which draw upon and augment users' cognitive, affective, and experiential capabilities. By building programs as sets of agents, representing these sets of agents as cartoon characters, and affording user interaction with these characters and their behaviors by means of narrative and video storyboard techniques, our system supports the activity of non-programmers in a complex computational environment.

ACKNOWLEDGMENTS

We would like to acknowledge the support of our sponsors at the MIT Media Laboratory as well as the support of the Mitsubishi Electric Research Laboratories in Cambridge, Massachusetts. Also thanks go to our advisors and colleagues at the MIT Media Laboratory and especially the members of the Narrative Intelligence Reading Group.

REFERENCES

1. Borning, Alan. ThingLab: A Constraint-Oriented Simulation Laboratory, Xerox TR SSL-79-3, 1979.
2. Laurel, Brenda. "Interface Agents: Metaphors with Character." In: The Art of Human-Computer Interface Design. ed. Brenda Laurel. Addison-Wesley, Reading, Massachusetts, 1990.
3. Minsky, Marvin. Society of Mind. Simon & Schuster, New York, New York, 1985.
4. Sutherland, Ivan. Sketchpad: A Man-machine Graphical Communications System, MIT PhD Thesis, 1963.

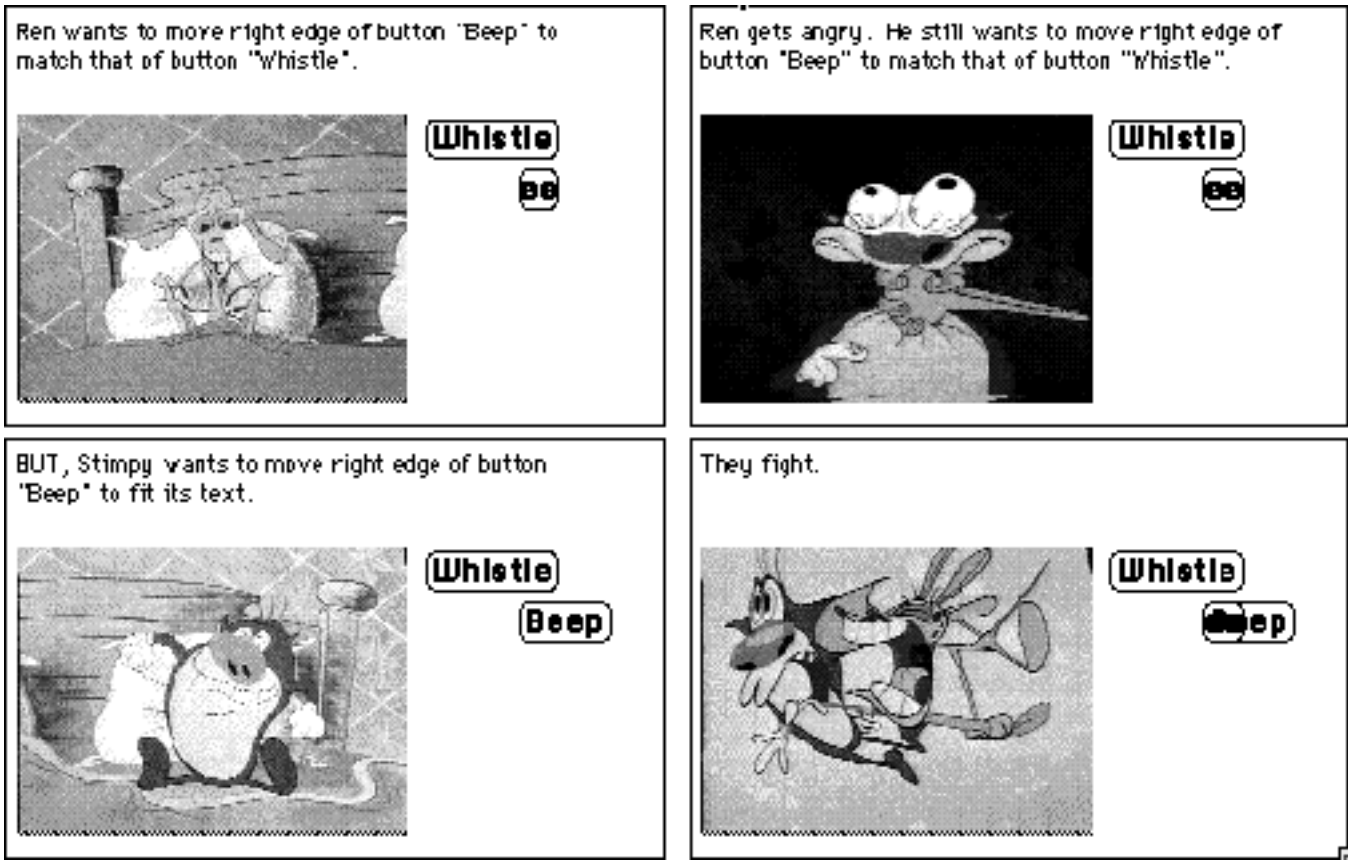

Fig. 5: First four panels of the video storyboard for Ren and Stimpy