Who's Got Game (Theory)?

By

Erik Jackson Blankinship

B.A. English Literature
University of Maryland
College Park, MD, 1997

M.Ed.
Harvard University, School of Education
Cambridge, MA, 1998

S.M. Media Arts and Sciences
Massachusetts Institute of Technology
Cambridge, MA, 2000

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2005

Author ____
Erik Jackson Blankinship
Program in Media Arts and Sciences
September 2005

Certified by ____
Walter Bender
Executive Director & Senior Research Scientist
MIT Media Laboratory

Accepted by ____
Dr. Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Who's Got Game (Theory)?

By

Erik Jackson Blankinship

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Media Arts and Sciences.
September 2005

Abstract:

Many players enjoy the challenge of outwitting computer opponents in strategy games. Devising strategies to defeat a computer opponent may enhance certain cognitive skills (e.g., analysis, evaluation, planning). This thesis takes a constructionist approach to gaming, hypothesizing that players may learn more about strategic planning by building their own computer opponents and then playing them to understand how their strategic theories play out in real experiments.

I have developed a graphic toolkit for designing strategy games and computer opponents. The goal is to help students learn the underlying mathematical and computer science theories used to win these games. The tools have been designed to eliminate the overhead of using conventional programming languages to build games and focus students on the pedagogical issues of designing and understanding game theory algorithms. I describe the tools as well as initial evaluations of their effectiveness with populations of teenage students.

Teenagers in this study posed their own problems, in the form of games they designed, and then hypothesized about winning strategies. Of their own volition, most teenagers iterated on their strategic designs, reformulated problems and hypotheses, isolated variables, and informed next generation versions of this tool with astute suggestions. The toolkit designed for this thesis has a low floor, making it easy for people to quickly start playing with mathematical concepts, and a high ceiling for sophisticated exploration.

Thesis Supervisor: Walter Bender
Title: Executive Director & Senior Research Scientist

Thesis Committee

Thesis Advisor

Walter Bender
Executive Director & Senior Research Scientist
MIT Media Laboratory

Thesis Reader

Brian K Smith
Associate Professor
School of Information Sciences and Technology
College of Education
The Pennsylvania State University

Thesis Reader

Mitchel Resnick
LEGO Papert Associate Professor of Learning Research
MIT Media Laboratory

Thesis Reader

Jon Orwant
Research Director
France Telecom

Acknowledgements

I would like to thank my committee: Walter Bender and Brian Smith for their friendship and guidance, Mitchel Resnick for inspiring me to come to the Media Lab (and helping me to graduate), Jon Orwant for inspiration and clarity.

I would like to thank a few good friends: Pilapa Esara, Bakhtiar Mikhak, Sunil Vemuri, and Georg Essl.

I would like to thank my fellow students in Electronic Publishing and Explanation Architecture, along with those in the other Media Lab research groups.

I would like to thank my teachers at the Media Lab and Media Lab Europe.

I would like to thank members of the Media Lab staff, notably Missy Corley, Felice Gardner, and Linda Peterson.

I would like to thank the generous sponsors of the Media Lab.

I would like to thank my undergraduate friends (well, not any more) from East Campus, notably: Michael Baker, Max Van Kleek, Jim Roewe, Cliff Frey, Sagara Wickramasekara, and Andy Marsh.

I would like to thank some friends: Lisa Clarke, Federico Grau, Benjamin Duncan, Paul Edwards, Natasha Abramova.

I would like to thank my family: my mother and father, my brother Adam, my cousin Olivia Crosby. I would also like to thank my family's pets: Oscar, Chewbacca, MacDougall, MacDuff, Banjo, Buttercup and Blue.

I would like to thank the Hulk for his incredibility, Spiderman for his amazingness, and Thor for his mightiness. I would also like to thank the Wizard of Frobozz.

This dissertation is dedicated to my uncle Holmes Crosby.

Table of Contents

Table of Figures

1 Introduction

Life is full of choices, many of which lead to unexpected outcomes. Therefore, learning how to plan ahead to find multiple paths to satisfactory results is a useful life skill. Because games are repeatable and enjoyable, they provide ample opportunities for learning about decision-making and problem solving in situ. However, learning to play a game well does not necessarily transfer to better decision-making skills. Algorithmic analyses of options in games, and in life, provide another means for reflection on the decision-making process. Opportunities to learn about systemic decision-making processes are limited by the amount of computation required to do it effectively; the opportunity to design algorithm-based decision-making systems is limited almost exclusively to computer programmers. Most others only see the effects of these systems, most often when they play computer games. But most people never understand, let alone design, these systems themselves.

This thesis describes how to make systematic decision-making concepts more widely available with the development of a new computational tool. The pedagogical goal was to provide constructionist opportunities for learning about strategy in the familiar context of strategy games. By constructionist, I refer to the educational research tradition of providing learners with well-designed modeling tools with which they can build robust mental models (Kafai and Resnick 1996). I started with an initial design for this tool and worked through multiple enactments with students in gaming workshops, making changes to better scaffold students' learning. The decision-making skills modeled by this tool introduced students to a systematic way to approach problem solving: enumeration and evaluation of all options. Students using my tool saw how they could design computational processes to help make decisions that impact them directly.

The tool described in this dissertation models a game-theoretical decision-making process. Game theory has practical applications in varied fields to predict and analyze behavior. For example, military strategists and economists (Weintraub 1992) have used it for planning and prediction. The theory has also been used for analysis of decision-making behavior in other fields such as theology (Brams 1980) and law (Baird, Gertner et al. 1994). As these examples illustrate, systematic decision-making has useful applications in various contexts. However, for many teenagers—who are the focus of my study—the practical applications are first and foremost games. Playing games is fun. I follow the constructionist tradition of helping students design computer games but take a new approach in that my tool de-emphasizes programming and focuses instead on teaching the decision-making system.

I begin this chapter by describing how games have been used in educational settings to teach decision-making. Then, I explain the approach of this thesis to learning about a decision-making process via algorithm modification and animation. I also introduce my design contributions and an overview of my evaluation contributions. I conclude this chapter with a preview of the rest of the dissertation document.

1.1 Educational Games

Card games have been studied as informal opportunities for learning arithmetic such as counting, evenness, oddness, quantities, addition, subtraction, and for introducing probability (Golick 1998). Decks of cards have also been used to teach adults advanced mathematics such as elementary statistics, set theory, abstract and linear algebra, and some game theory, although not necessarily by playing games (Baker 1999). The player or teacher who knows to look can find mathematical relationships within many games.

However, most educational games are about learning traditional school subjects extrinsic to the games themselves. The earliest examples are from the mid 17th century: decks of standard playing cards decorated with illustrations and text captions of educational subjects such as geographic locations, historical events, and heraldry (Tilley 1967). The educational value of these decks is more as instructional material than game. A recent example is the US Military's Iraq deck given to GIs to acquaint them with enemy names and faces.

Even if intended for memorization of facts, playing cards that maintain their suit and rank can still be used to play games. Many instructional computer games, in contrast, offer only drill and practice repetition. Marketed as making math easy, these games fail to recognize that easy games aren't much fun; *hard fun* is what constitutes good game playing (Papert 1998). Arguably, these instructional games fail to sustain children's interest because they fail to integrate "learning" and "playing".

1.1.1 Simulations: Decision Making in Context

Simulation games, in which students' decisions are intrinsic to the outcome of the game, are more compelling than instructional games. For example, role-playing simulations in social studies classes, such as "Model U.N." in which groups of students take the roles of countries in diplomatic negotiations, contextualize the decision-making process with pretense of international importance. However, because these are interpretive simulations, it is difficult to reflect upon and analyze all decision-making factors during or after a game.

In games with more structured rules, such as strategy games, decision-making factors are more easily enumerated. Such was the goal when Educational Development Center (EDC) in Cambridge, Massachusetts, developed and tested a board game in which students moved Eskimo game pieces to hunt caribou game pieces (moved by other players) (Carlson 1969). If the hunters didn't cooperate, they would perish from starvation and lose the game. The game was inspired by Harvard educator Jerome Bruner to focus students on the interdependence of man and his environment. The rules of the game were designed to teach "that it is through careful organization of technological and human resources that man is able to master his physical environment." That is, the rules of the game were an artifice for simulating real-world decisions and their outcomes.

Around the same time, computer simulations were introduced into classrooms to study their effectiveness as learning tools. These early games were text-based simulations such as *The Oregon Trail* and *Sumer*. In these simulations, children made decisions such as how many supplies to buy to survive a cross-continental trip or how much grain to plant to feed an ancient city. Children's decisions were entered via a keyboard and then descriptive outcomes were generated based on equations in the game's software. Within a few years, the original source code of *The Oregon Trail* was published alongside graphs of the game's governing equation (Rawitsch 1978):

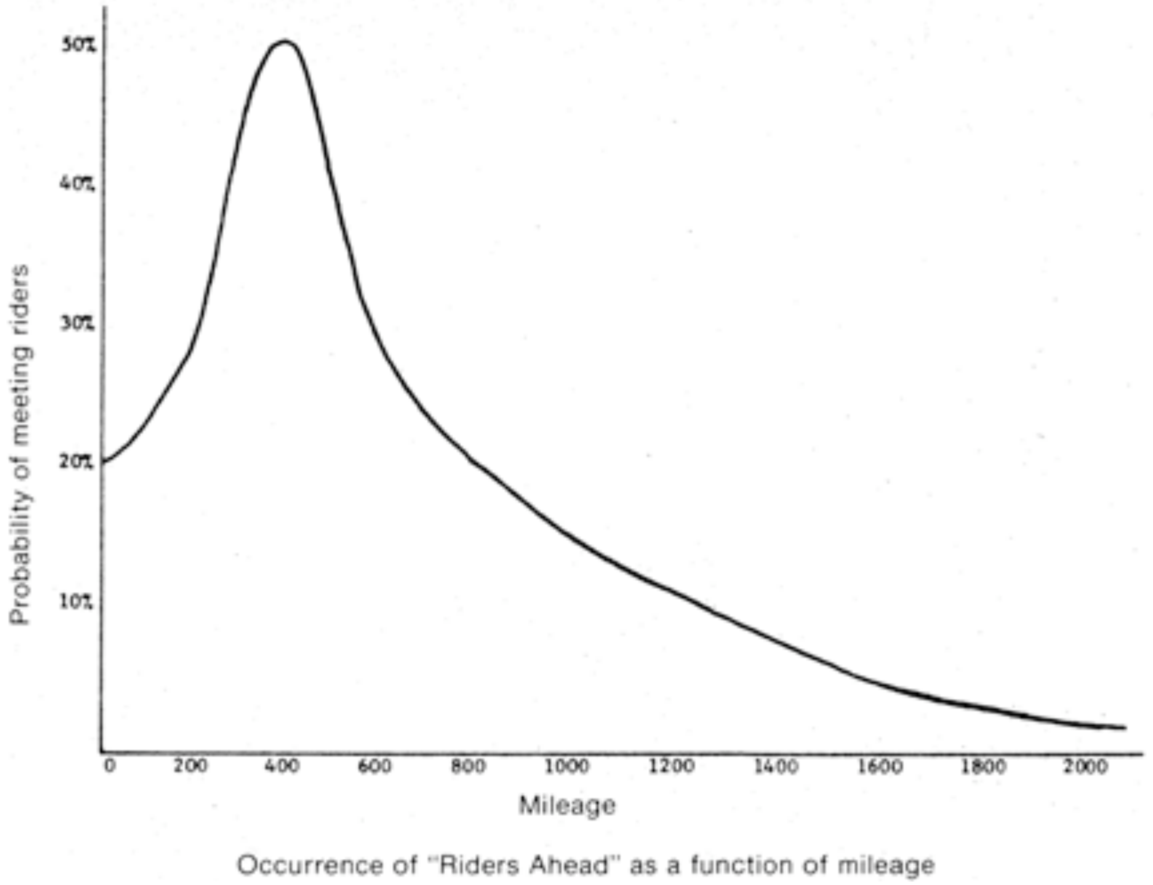


FIGURE 1.1 A GOVERNING EQUATION FROM THE OREGON TRAIL SIMULATION GAME

With this graph, players could strategize about how to play better. While some game players might argue that this takes away the enjoyment of playing a simulation since it reveals the outcome of decisions, it is hard to argue against the educational merit of seeing how the algorithm impacts game play. Access to this algorithm is the equivalent of "looking under the hood" of a simulation.

These simulations are predecessors of the more sophisticated *SimCity* and *Civilization* graphic simulations, which have been studied by educational researchers as contexts for learning history and urban planning (Squire 2003). However, a problem with using these games as learning tools is that players can only intuit the governing equations of these games and are therefore bereft of many learning opportunities for nuanced analysis (Starr 1994). Instead, students playing these games "learn the properties of a virtual world through interacting with its symbology, learning to detect relationships among these symbols, and inferring the game rules that govern the system" (Squire 2002). It would be beneficial to educators if the governing equations were available for analysis and graphing like the source code to *The Oregon Trail* was decades ago. It would be even better if students could modify these equations themselves to see their effects on simulated behavior.

1.2 Game Modifications

Children are adept at modifying the games that they play. This flexibility has been well documented by social scientists in England (Opie and Opie 1969) whose longitudinal study of variations in children's folk customs, including children's schoolyard and street games, revealed subtle differences throughout England. Playground game variations come from the children themselves. Swiss psychologist Piaget's interviews with children found that younger boys attributed the origins of the rules for marbles to their fathers or God (or a conflation of both) and thus considered them immutable, while older boys credited learning the rules from peers and acknowledged they were meant to be changed (Piaget 1965). The members of this gerontocracy also reflected on the need to balance rules whenever changes were made so as to ensure a fair game.

Modification is a feature of some popular computer games in kids' culture today, albeit only where the computer-game designer has provided ways for players to plug-in their data. For example, different "skins" for a video game can change the graphics, or different map files can create new maze games. Although making modifications to a game provides a sense of authorship (Fine 1983), the modifications are mostly cosmetic. The original computer programmer has already defined the rules of play; all the game players are changing are variables. However, with an appropriate digital representation for game rules and processes, it is possible to modify the rules of a computer game with design primitives such as "turn", "move", "hand", "round", etc. (Orwant 1999).

For this thesis, I studied two-player, turn-based games because they provide opportunities for learning classic strategies associated with game theory. Instead of studying just one example strategy game, I created game-design tools that can be used to design a variety of games, all of which could be strategized about in similar ways. I did not want to design an exemplary educational game and then make it *more educational* by using it to help students learn about strategy. Rather, I wanted students to make modifications to their own games, to the point of being able to create them from scratch, so that they would have access to more strategic scenarios. And by being able to create and modify asymmetric games, children would have opportunities for reflection on balance and fairness in relation to strategy.

1.2.1 Technical Contribution: The Gaming Calculator

Although some computer games allow modifications that can change the look, feel and even game mechanics, simulated behavior remains the ghost in the machine. Since the allure of many computer games comes largely from an always-available computer opponent, my inspiration was to extend game modification to the algorithms that govern simulated behavior. My design goal was to make complex modifications to these algorithms relatively simple without ceding to full-blown computer programming which is overwhelming and not tailored to learning strategy.

Game playing algorithms use *logic* to make decisions based on game *data* (Rabin 2000). A well-designed separation of logic and data makes it possible for the same game-playing algorithms to play different games. What I have done in this thesis is to use data to modify not just games, but to modify the logic of the algorithms that play these games. The contribution of this approach to learning about strategy is a focus on high-level problems, not the maintenance of control-structures and the resultant syntax errors associated with programming.

Consider an analogy to another tool—graphing calculators—found in most secondary math classes today. Graphing calculators allow students to plot *f(x)* and to rapidly explore how functions operate. They provide unlimited exploration within a limited problem space. The pedagogical strength of graphing calculators is not in replacing students' learning opportunities, but enhancing them by facilitating more inquiries than are possible by rote calculations. Graphing calculators are also computers, which motivated students can program to explore more complex computational ideas. In this spirit, I introduce a *gaming calculator*: as a graphing calculator is designed to help students solve problems related to equations, the gaming calculator helps to solve game-theoretical mathematical problems.

A graphing calculator, as the name suggests, literally *graphs* mathematics to help students identify patterns hidden in equations. Similarly, the gaming calculator helps students find the mathematics hidden within games by graphing the decision-making process into flow-chart representations. Since the decision-making modeled with this tool is an algorithmic process, diagrams are animated to elucidate the sequential steps of that process. These diagrams represent changes to both the modified game logic and the current game data, providing custom learning aids for every move in a game. Therefore, a contribution of the gaming calculator is that animated diagrams and graphics reveal the decision-making process to help learners understand the function of their strategic designs.

The gaming calculator makes debugging easier by linking relevant source code to its representation in the diagrams and vice versa. For example, clicking on a line of code that describes how to evaluate moves overlays informative graphics onto the game board; selecting a different move from a diagram updates the game board and the overlaid graphics accordingly. These visual scaffolds facilitate debugging by coupling the description of a process to a representation of the process unfolding.

1.2.2 Pedagogical Contribution: Testing Strategic Hypotheses

My pedagogical focus was to help students learn about strategy by designing strategies. A guiding instinct throughout my research was that winning games is fun and therefore would hold the interest of students when they were learning the mathematics. This instinct proved right as I conducted research workshops with teenage students.

Their use of the gaming calculator led to my second contribution: initial understandings of how teenagers used the gaming calculator to develop hypotheses about strategies and test those hypotheses by creating computer opponents. Through my workshops, I was able to identify conceptual difficulties students had as they tried to understand the mathematics of game theory. For instance, teenagers did not know how to systematically plan ahead nor did they have a method for selecting their best move in a competitive game.

While I had initial hypotheses about these difficulties, more emerged as I worked with students and those led to further development of scaffolds in the gaming calculator and additional trials with teenagers. This process of iteration—refining the software, testing with teenagers, refining again based on what was learned—was used to not only discover the conceptual difficulties but also to inform the design of supports to assist learning.

1.3 Overview

- Chapter 2 is an extended example of one teenager's learning story with a prototype gaming calculator.
- Chapter 3 contextualizes this work within the constructionist tradition.
- Chapter 4 describes my iterative process of design and evaluation with evidence of how students in my workshops used the gaming calculator to learn about strategy.
- Chapter 5 describes how this thesis' design approach can be extended to enable exploration of decision-making in other types of games that children like to play.

2 Extended Example

As part of this thesis work, I ran a series of eleven game workshops over six months to study what teenagers learned about decision-making when they used prototype gaming calculators to create and win strategy games. In all, 25 teenage students responded to postings and advertisements for various game workshops. In my early workshops, we played chess variants so that I could better understand the problems students had understanding strategy in a familiar game. Later workshops included both game design and strategy design. This chapter is an extended anecdote describing how one of these later workshops ran; I show the reader how a real teenager used the gaming calculator. At the end of the chapter, I summarize what this student learned at the workshop.

2.1 Introduction

I met Jim, age 14, on a summer afternoon at his school's library. He had responded to a flyer advertising game workshops that was sent to the members of his junior high school's math and chess club. His membership in such a club and his interest in my workshop indicated he was a self-motivated teenager with an interest in solving problems. The activity appealed enough to bring him back to school over summer vacation.

In an informal interview, I learned that Jim had no computer programming skills, although he had some experience making web pages with HTML. He liked to play strategy games and computer games but had never designed one of his own. He was keen to know what we were planning to do that afternoon; I outlined for Jim the afternoon's agenda: I was going to teach him how to design his own strategy game, which would consist of designing game pieces, a game board, and winning conditions. Then, he was going to design a computer opponent to play the game he had just designed. I stressed that our goal that afternoon was not to create an unbeatable computer opponent, but rather to understand the process by which his opponent worked.

2.2 Game Design

Jim asked, "What kind of strategy game are we designing? Real-time strategy?" If you are unfamiliar, real-time strategy games are a popular genre of video games in which players control semi-autonomous armies in real time. This was a common question from students in my workshops, indicating that teenagers today are savvy and discerning game players.

"No, we are designing an old fashioned, two-player strategy game—a turn-taking game. Games like chess and checkers. But your game can be different than those games. For example, you can make game boards of different sizes and boards with barriers. You can make pieces that can move and capture in different ways. Once you make a board and some pieces, you can put them together into a game and each side can have different pieces. You can also make up your own ways to win your game by describing those to the computer. Don't worry if this sounds overwhelming. We're going to go through this design process step by step."

I asked Jim what sort of game he would like to design, and he wasn't sure. I suggested some popular book and movie titles as source material, and he suggested one of his own: Harry Potter. (I later learned that his e-mail address name was a variant of Harry Potter, and that he was a big fan of the book series). With that decided, I booted up the gaming calculator and got Jim started designing his game.

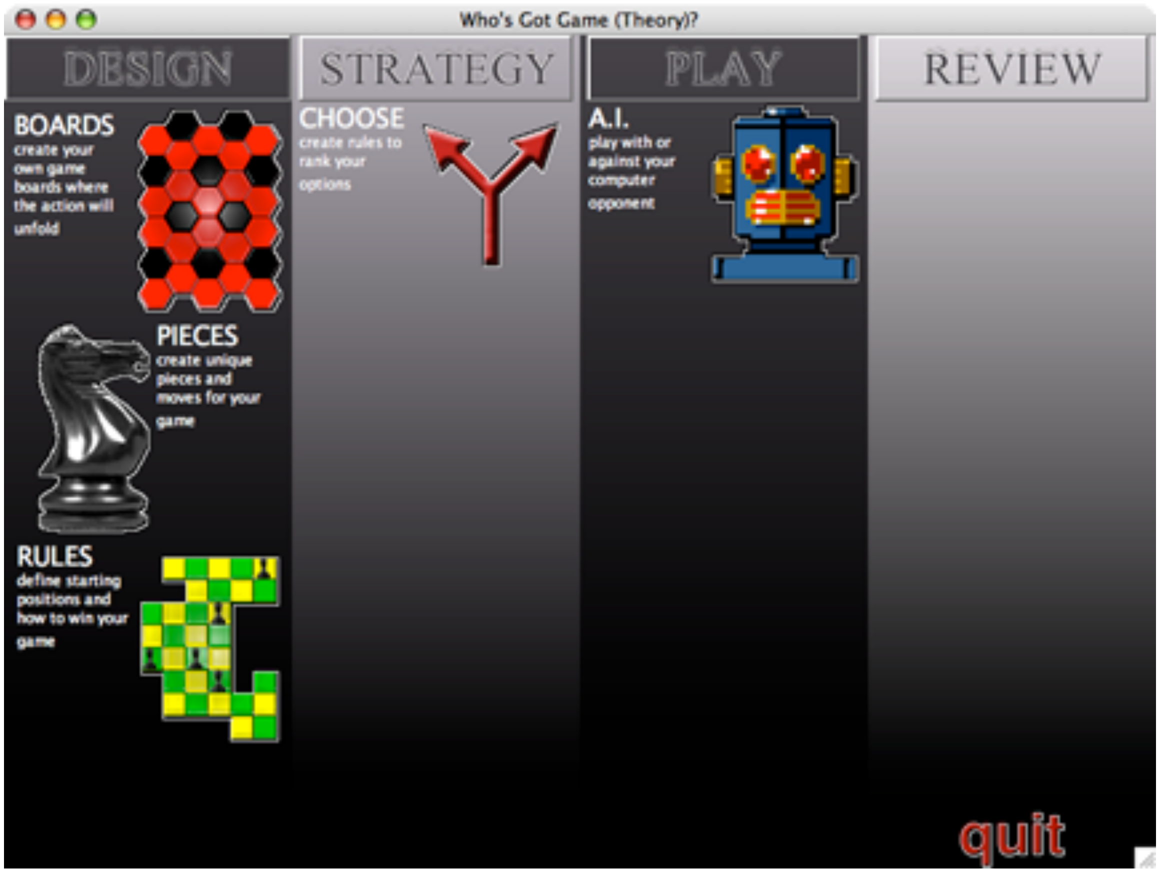

Who's Got Game (Theory)?
DESIGN
STRATEGY
PLAY
REVIEW
BOARDS
create your own game boards where the action will unfold
PIECES
create unique pieces and moves for your game
RULES
define starting positions and how to win your game
CHOOSE
create rules to rank your options
A.I.
play with or against your computer opponent
quit

FIGURE 2.1 THE MAIN MENU OF THE GAMING CALCULATOR

2.2.1 Board Design

I started by showing Jim how to design his own game board. The board-design screen came up, consisting of a blank 4×4 tiled game board and a few design palettes to the side. I showed Jim how we could change the dimensions of the game board in the *Size* palette, by adjusting *Width* and *Height* sliders to create a 5x5 board. Jim increased the size of the board to 5x5. Jim asked why we couldn't make a larger game board, at least one the size of a real chessboard, and I told him that it was easier to learn by starting with smaller games to minimize complexity.

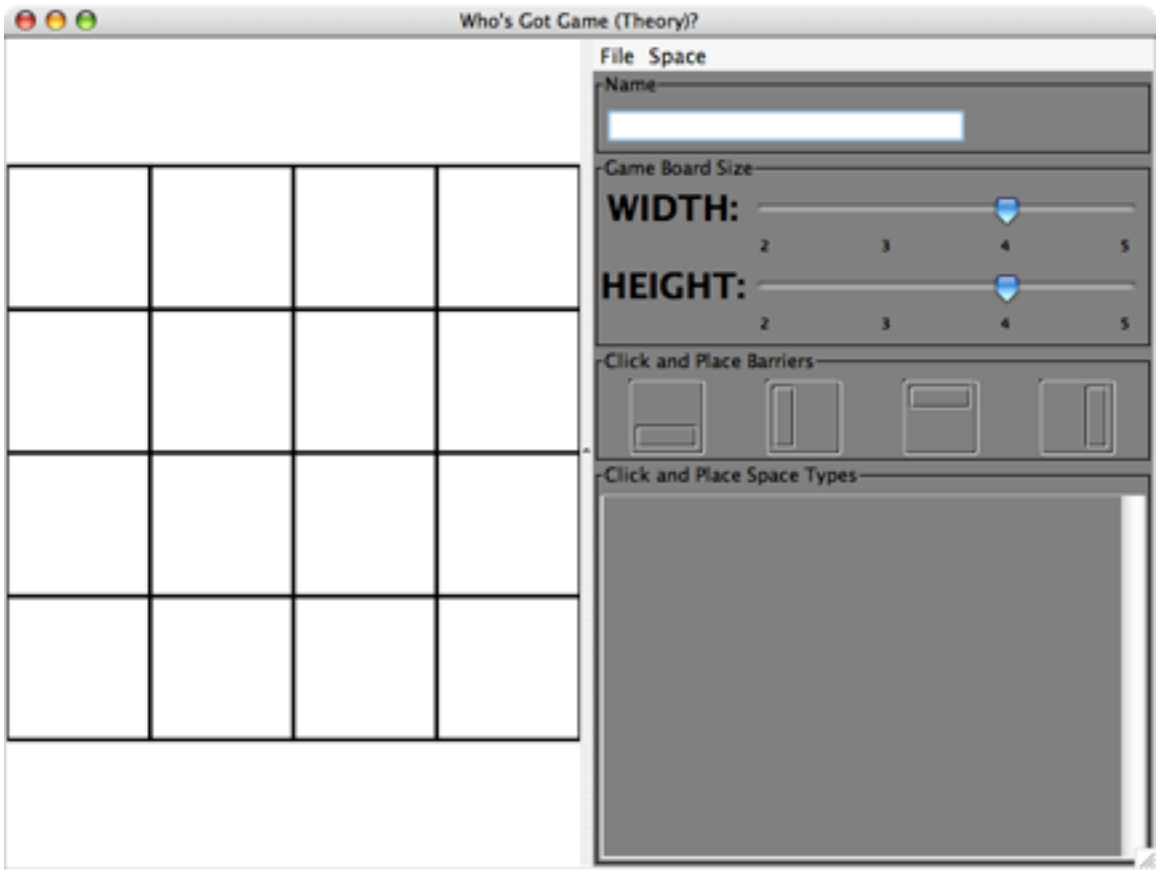


FIGURE 2.2 THE DEFAULT BOARD-DESIGN SCREEN OF THE GAMING CALCULATOR

Next, I showed Jim how we could decorate the game board by adding some graphics. On the menu bar, I selected the *Add Color Space* menu item and picked red from a color dialog, which added a red button to a *Spaces* palette. Then, I added a blue button to the *Spaces* palette. I explained how we could also go on the Internet and get some graphics to add images to the spaces palette. Now that we had a few choices on our palette, I clicked on the red button, which changed our mouse pointer from the default arrow to a small game board pointer, indicating that the function of the mouse had been changed. Then we clicked a space on the game board, which colored that space in, together with its mirror space on the other side of the game board. I explained, “If you want to erase the graphic that you put on a space, just click the space again and it will be cleared.”

Jim asked, “Can you only make symmetric game boards?”

“Yeah, it’s designed this way so that if both players have the same game pieces, then strategies are interoperable regardless of which side you’re playing. But, as you’ll see, each player can have different game pieces and different starting positions. Another reason the game board is symmetric is to keep from having to spin the board around if you’re designing a strategy for both sides. It helps keep things from getting confusing.”

The remaining board-design palette was labeled *Barriers*, and I next showed Jim how this worked. The palette had four buttons shaped like squares, each one with a thick barrier on a different cardinal side. Clicking on the button with a heavy barrier on its bottom side updated the graphic of our mouse pointer into a miniature version of the button , indicating that the palette was active for adding barriers to the game board. Then I clicked on a space on the game board, which dropped a thick graphic barrier on the bottom of that space. A barrier was also added on the other side of the game board, making the game board's barriers symmetric as well. "Only pieces that can hop, like checkers or the knight in chess, can move over barriers."

Upon learning how to design his own game board, Jim created a board that represented one of the dungeon passages in Hogwarts, a school in the Harry Potter story. Using the barrier function, he added dungeon walls through the middle of the game board to create a corridor, then cleared away some of the barriers to make passageways into the corridor. Then, he colored in the corridor with blue tile and colored the outside area with grey tile. Jim saved his work to disk, eager to learn what we'd do next.

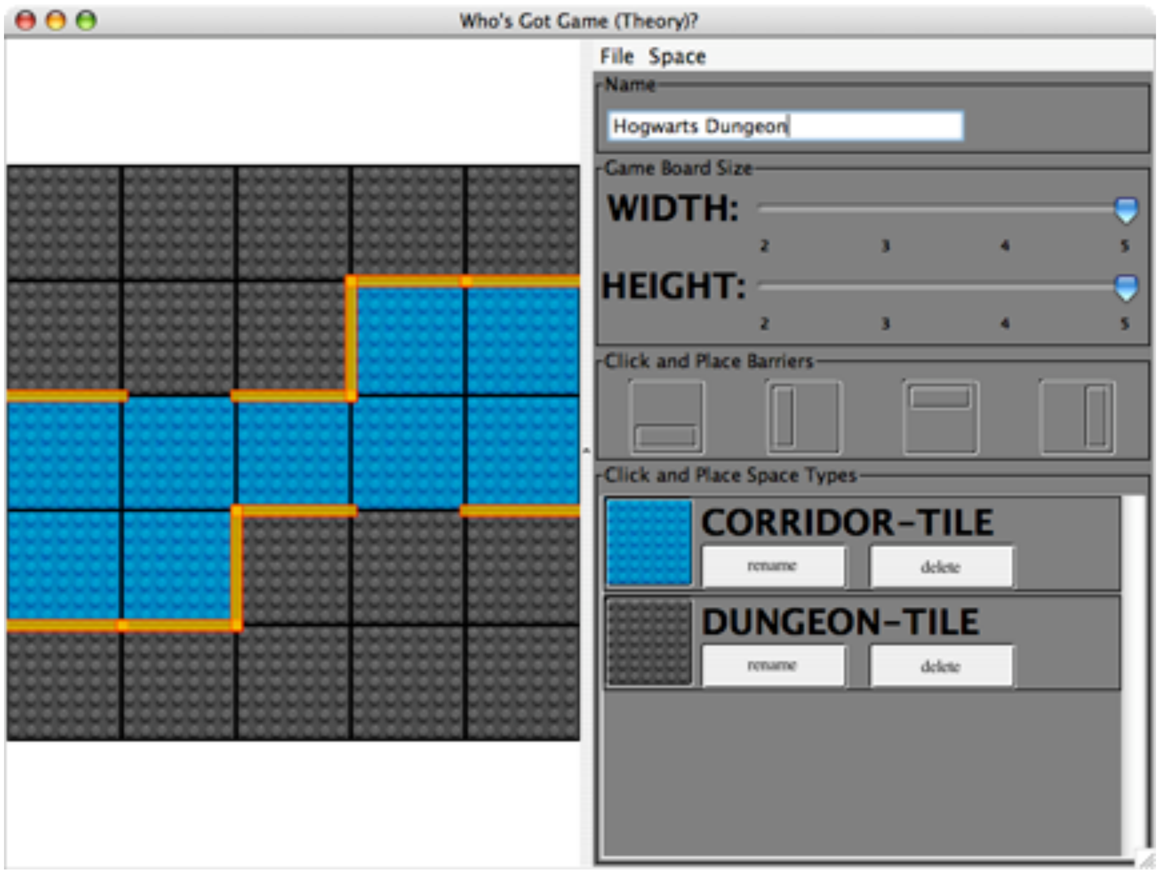

Name: Hogwarts Dungeon

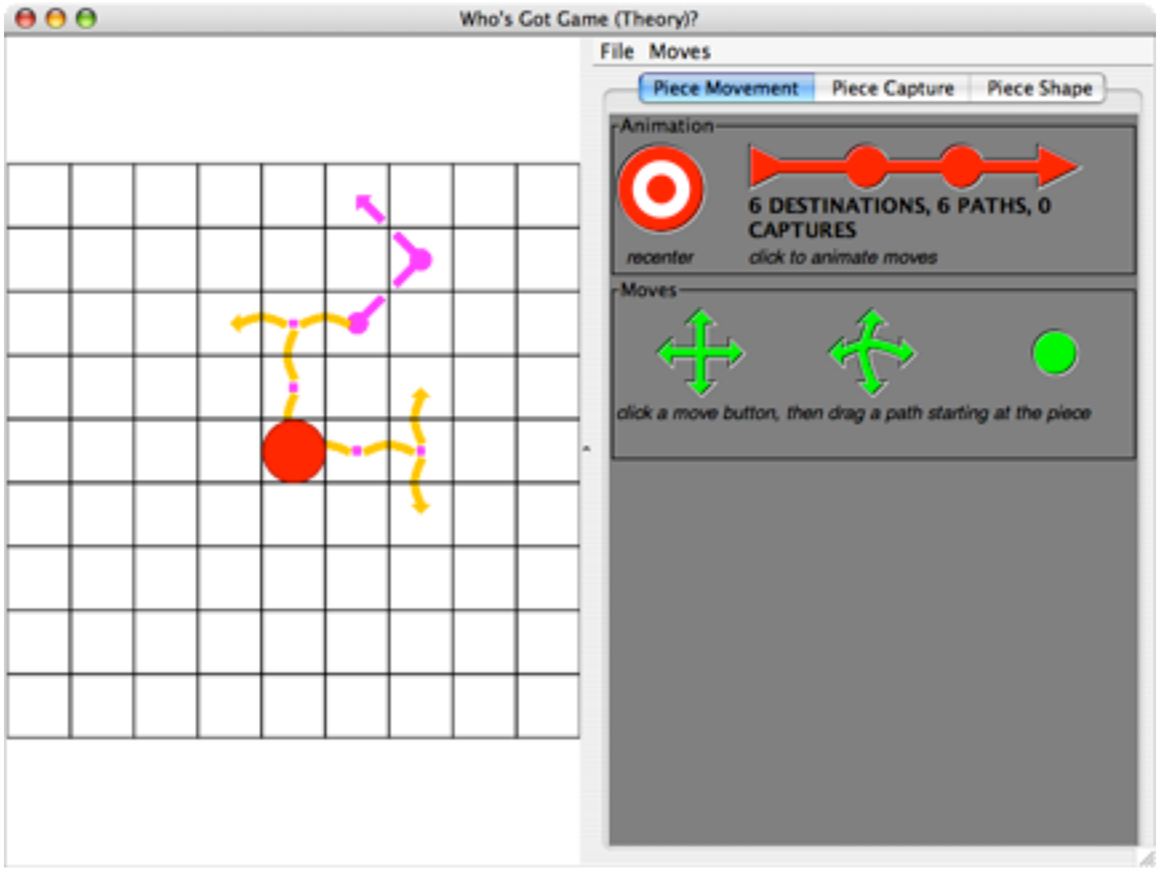
FIGURE 2.3 A HOGWARTS GAME BOARD DESIGNED WITH THE GAMING CALCULATOR'S BOARD-DESIGN TOOLS

2.2.2 Piece Design

I opened up the piece-design tool and said, “Let me show you how we define the way pieces move around the game board you just designed.”

On screen was a large 9×9 game board with a single red piece in the middle shaped like a checker. A palette to the side was titled *Moves* and had buttons labeled *Move*, *Hop*, and *Stop*. I clicked on the *Move* button, which turned the cursor into a small compass rosette and indicated that the palette was enabled.

Clicking on the game piece on the board, I dragged a path through adjacent game spaces, which left a graphic trail on the game board extending from the game piece. I explained to Jim, “If we added this piece to our game, this is the path it could move along. Let me show you how it moves by animating it.” I clicked on the *Animate Moves* button in a side palette, and the game piece slid from its starting position along the graphic trail that we had drawn on the game board. When it reached the end of its trail, it reappeared at the center of the board. Jim’s facial expression brightened, indicating the he thought this was pretty cool—the game software had some graphics that moved.



6 DESTINATIONS, 6 PATHS, 0 CAPTURES

click a move button, then drag a path starting at the piece

FIGURE 2.4 THE PIECE-DESIGN SCREEN OF THE GAMING CALCULATOR

I explained to Jim, "If we want to make a piece jump over any barriers, or over other pieces, we can add hop moves onto the game piece's path." I clicked on the *Hop* button, and then clicked and dragged on the game board to extend the path graphic we had already started. While moves were drawn onto the game board as straight lines, hops were drawn as small arcs. When I next clicked the *Animate Moves* button, the game piece first slid along its initial move path, as before. Then when it reached the new hop segments of its path, the game piece hopped from space to space. "We can extend the way a piece moves by drawing a branch on to the existing paths we've made. This is how we'd design a piece like the knight from chess; it can fork at the end of his move."

Jim asked, "But how do we make a piece stop someplace other than at the end of its path? Can't we stop a piece somewhere along the way? How would we design a piece like the chess queen?"

"That's easy. Just click on the *Stop* button in the *Moves* palette, and then, on the game board, click on spaces along your game piece's path to add stops. Now, let me show you how you can let your game pieces capture other game pieces."

In the *Captures* palette, I showed Jim a list that had been automatically generated to show all of the final stops that our game piece could make. Selecting one of these stops from the list highlighted that stop on the game board. These stops included the ones that we had just added along our game piece's path and the ends of every path. Above the list of stops were two target-shaped buttons, one labeled *Capture* and the other labeled *Move Only On Capture*. I clicked on the *Capture* button and then clicked on a selected space along the game piece's path. This added a colored explosion graphic to the game board at this space, indicating a capture.

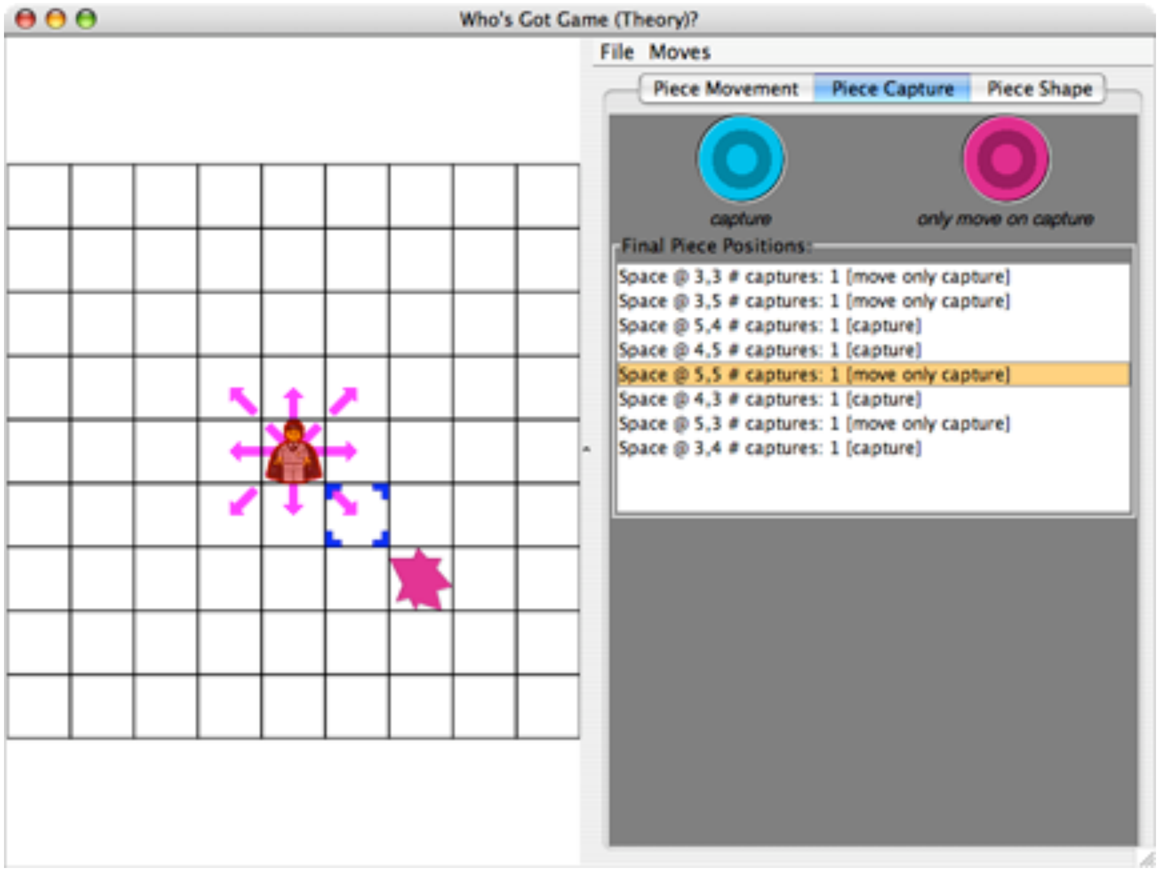
"We aren't limited to just landing on an opponent's piece to make a capture. We can make it so we capture pieces that we've just hopped over, or a piece in front of us, or both. We could even make it so that if we stop here, we capture pieces all over the game board." Jim thought this was a pretty interesting feature.

"Okay, I get it. This is like a make your own freaky-deaky chess program. What does that *Move Only On Capture* button do?"

"That's so you can make pieces that capture like the pawn in chess. The pawn can move diagonally, but only when there is an opponent's piece there to capture. You can add that sort of limitation to how your game pieces move."

Next, I showed Jim how we could change the look of the game piece in the *Appearance* palette with buttons for changing the name and color of the game piece. I also showed him how we could use a letter of the alphabet, rendered with any of the computer's available fonts, as the graphic. I explained how we could load in graphics from the Internet to use as game pieces (and how, if a graphic had a transparent background, the game piece token would automatically take the shape of the visible part of the graphic so it would look more like a game piece).

Upon describing the process by which Jim could design his own pieces, he thought for a bit and came up with some ideas. Jim made a Harry Potter piece that moved a little bit like a king in chess, but it did not capture by landing on opponent's pieces. Instead, Harry used his wand to zap pieces one space away from him in the direction of his move. Also, Harry could not move diagonally unless there was an opponent's piece to capture. Then, Jim made a professor piece that moved like Harry but that could move diagonally without capturing. Finally, Jim made a ghost piece that could hop diagonally, thereby moving through the dungeon walls of the game board. The ghost piece could capture by landing on an opponent's piece. Jim saved all of his pieces to disk, each as a separate file.



```
Space @ 3,3 # captures: 1 [move only capture]
Space @ 3,5 # captures: 1 [move only capture]
Space @ 5,4 # captures: 1 [capture]
Space @ 4,5 # captures: 1 [capture]
Space @ 5,5 # captures: 1 [move only capture]
Space @ 4,3 # captures: 1 [capture]
Space @ 5,3 # captures: 1 [move only capture]
Space @ 3,4 # captures: 1 [capture]
```

FIGURE 2.5 A HARRY POTTER GAME-PIECE DESIGNED WITH THE GAMING CALCULATOR'S PIECE-DESIGN TOOLS

2.2.3 Rules Design

Now that Jim had designed both a game board and game pieces, he was ready to put these pieces together to make a game. He opened the rules-design screen, which was empty except for palettes labeled *Player A Pieces*, *Player B Pieces*, and *Winning Conditions*. Using an open file dialog, we loaded in Jim's game board file, and it appeared in the game design window. Then, in the same fashion, Jim loaded in all of the game pieces he had just designed. The pieces were added to both the *Player A Pieces* and *Player B Pieces* palettes.

I explained, “Here in the rules-design toolkit, we put your pieces onto the game board in their starting positions. Player A and Player B have all of the same pieces available to them, but they don’t have to be symmetrical or even have the same pieces.

“Where you place pieces is where they will be at the beginning of your game. If you have pieces that only advance forward, like pawns in chess, you need to make sure you start them on your side of the board so they have somewhere to go. Player A’s side is on the bottom of the board, and Player B’s side is on the top of the board.”

Jim said, “So I’ll put Harry on Player A’s side and the Hogwarts staff on Player B’s side.” He clicked on a Harry Potter piece in the *Player A Pieces* palette, which updated the appearance of his pointer on the screen to a letter *A*, and then clicked a space on the game board, thereby stamping a Harry Potter piece onto the game board at that space. The game piece on the board was tinted red, indicating it was one of Player A’s pieces. Jim stamped one more Harry Potter game piece onto the board. Then Jim switched to Player B’s palette, selected a Professor piece, and stamped two of those onto the game board, then did the same for his ghost. “Those seem like good places to start the game,” he says, “but I don’t think two Harry Potters can beat all of these other guys in a chess-like battle.”

“You don’t have to make this a capture game like chess. You can have different winning conditions for your game. For example, you can define winning as getting a piece to a certain location, maybe crossing the board successfully.”

“Oh. Then how about Harry has to cross to the other side, like sneaking around Hogwarts, and all of the Hogwarts staff are trying to capture him. That’s how they’d each win.”

I told Jim, “That sounds good. Let me show you how we can define winning conditions for your game. The first player to reach a winning condition ends the game because only one player can win.” In the *Winning Conditions* palette, I showed Jim an empty list with *Add*, *Delete* and *Edit* buttons at the bottom. Jim clicked on the *Add* button, and this brought up a new window with a modifiable sentence composed of pull-down menus and other widgets. The sentence read, “If [1] or more [Harry Potter] pieces [of mine] are [on the board] at this location [image of game board with highlighted spaces], [ ] then I win the game.” (In this sentence, everything in brackets is a widget that Jim could adjust the value of).

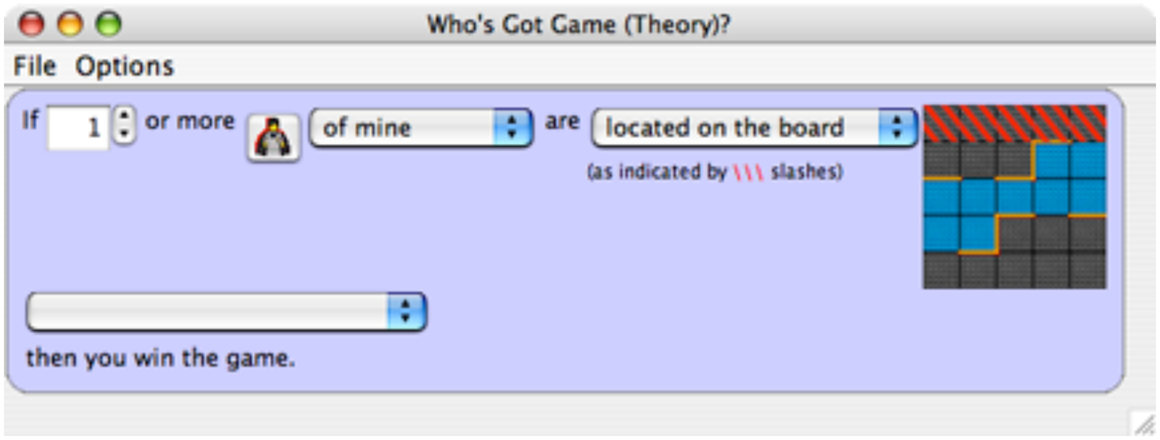


FIGURE 2.6 DEFINING WINNING CONDITIONS WITH THE GAMING CALCULATOR

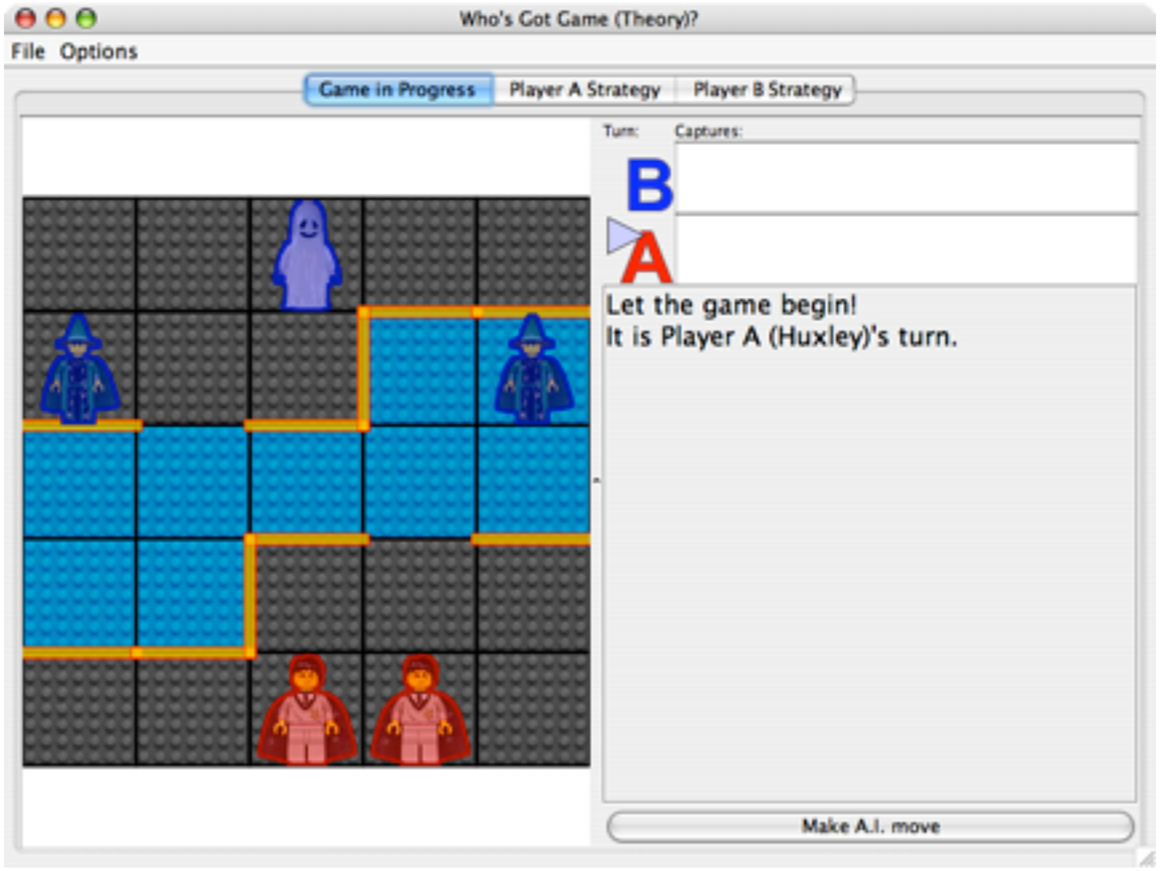
I explained to Jim that he could describe one of his winning conditions by modifying this sentence. The default sentence already described Jim's winning condition for Harry pretty well, except that Jim had to click on the back row of the game board to specify that those were the spaces where Harry had to be to win the game. Then, Jim saved his changes, and his winning condition was added to the *Winning Conditions* list as "Harry Potter on the Board".

Jim clicked *Add* again to create the winning condition for Player B's Hogwarts staff. Jim used a pull-down menu to change [on the board] to [captured], which caused the miniature game board to disappear. Then, Jim changed the number of pieces at the beginning of the sentence to [2]. His sentence now read, "If [2] or more [Harry Potter] pieces [of theirs] are [captured], [ ] then I win the game." Jim clicked save, and that winning condition was listed on the *Winning Conditions* list, shortened to "2 of their Harry Potters caught".

I explained to Jim that we could add more winning conditions. For example, Harry could capture one of the Hogwarts teachers to win, in addition to winning by getting to the other side of the game board. Jim thinks it over and decides to keep his game as is and saves his game file to disk. He is ready to play.

2.3 Strategy

From the main menu, Jim clicks on the *A.I.* button, and this brings up the game-playing screen. After loading his game file, Jim's Hogwarts game board with all of his game pieces on it at their starting positions appears on this screen.



Let the game begin!
It is Player A (Huxley)'s turn.

FIGURE 2.7 THE GAME-PLAYING SCREEN OF THE GAMING CALCULATOR

To the side are two small panels labeled *A* and *B*, which I explain to Jim are where captured pieces will end up. The *A* and *B* panels also indicate whose turn it is with a small indicator arrow. The arrow points to the *A*, and I explain that games always start with player A (but that that might change in a future version).

I suggest to Jim that we try playing his game, and he agrees. He clicks on one of his Harry Potter pieces and drags a ghost image of it to a neighboring space on the game board. When he releases the mouse, the ghost image disappears and the game piece itself slides from its current location to the selected space. The turn indicator arrow also moves to point to *B*.

I then refer Jim to a button on the screen labeled *Make A.I. Move*. I explain that pressing that button tells the computer to decide which move a player should make. I explain that for now, pressing *Make A.I. Move* just makes a random move, but that we will design a strategy soon. After a few moves against the random player B, Jim has won his own game and the computer displays a message that reads, “Player A Wins! Winning Condition: Harry on the Board.”

Jim comments, “I guess my game is pretty easy to win when there isn’t any real competition.”

2.3.1 Number of Moves

We select the *Restart Game* menu item, and the game pieces are returned to their designated starting positions. Before we play again, I ask Jim how many possible moves he has on his first turn. He looks over the game board and says that he has two moves. I ask him how many countermoves Player B has, he thinks it over, and says they have 4 moves. Neither of these answers is correct, so I guide Jim through a counting process to make sure that we are both using the same definition for number of moves. I point to the first Harry Potter figure on the board and ask Jim how many spaces it can move to on this turn. He answers, “two”. Then I point to the other Harry and ask how many spaces it can move to on this turn, and he answers “two”.

“So how many moves, in total, can you make on your turn?”

“Oh, I guess four.”

“And how many countermoves can Player B make on their turn?”

“Um, I guess… let me count… they have ten moves.”

“Is that right? If you move Harry here, how many moves can they make in response?”

“Ten.”

“And if you move Harry here, how many moves can they make?”

“Ten again. I see, they can make… a total of forty moves if I count everything they can do after every move I can make.”

“Let me show you a way to diagram all of these moves and countermoves.”

I click on the *Player A Strategy* tab, and the game board disappears and up comes a new set of screens. I explain to Jim, “This is your strategy dashboard, where you can get information on your strategy and also where you can program your strategy. Here is a diagram of all of your moves and countermoves: right now it is only showing your four possible first moves.”

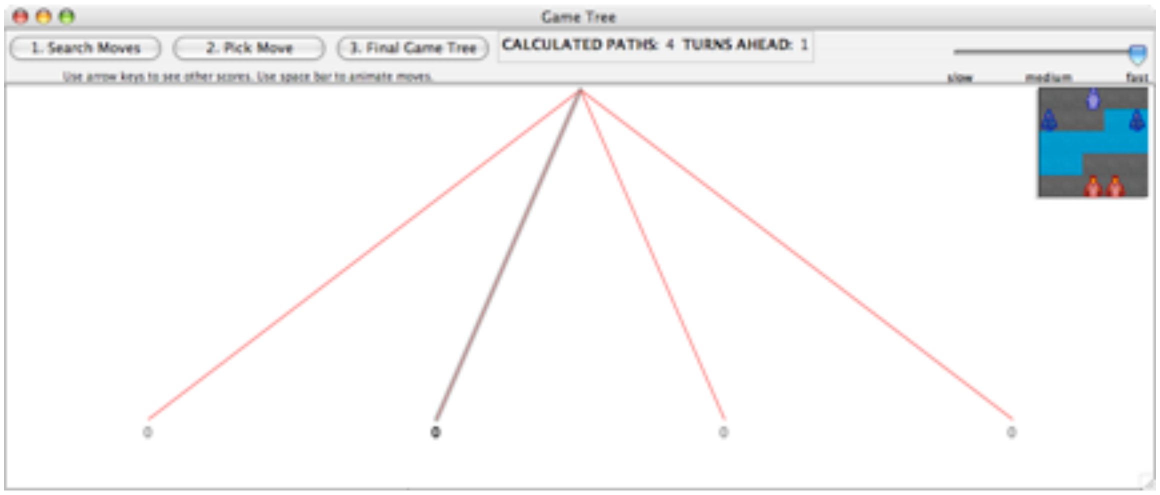
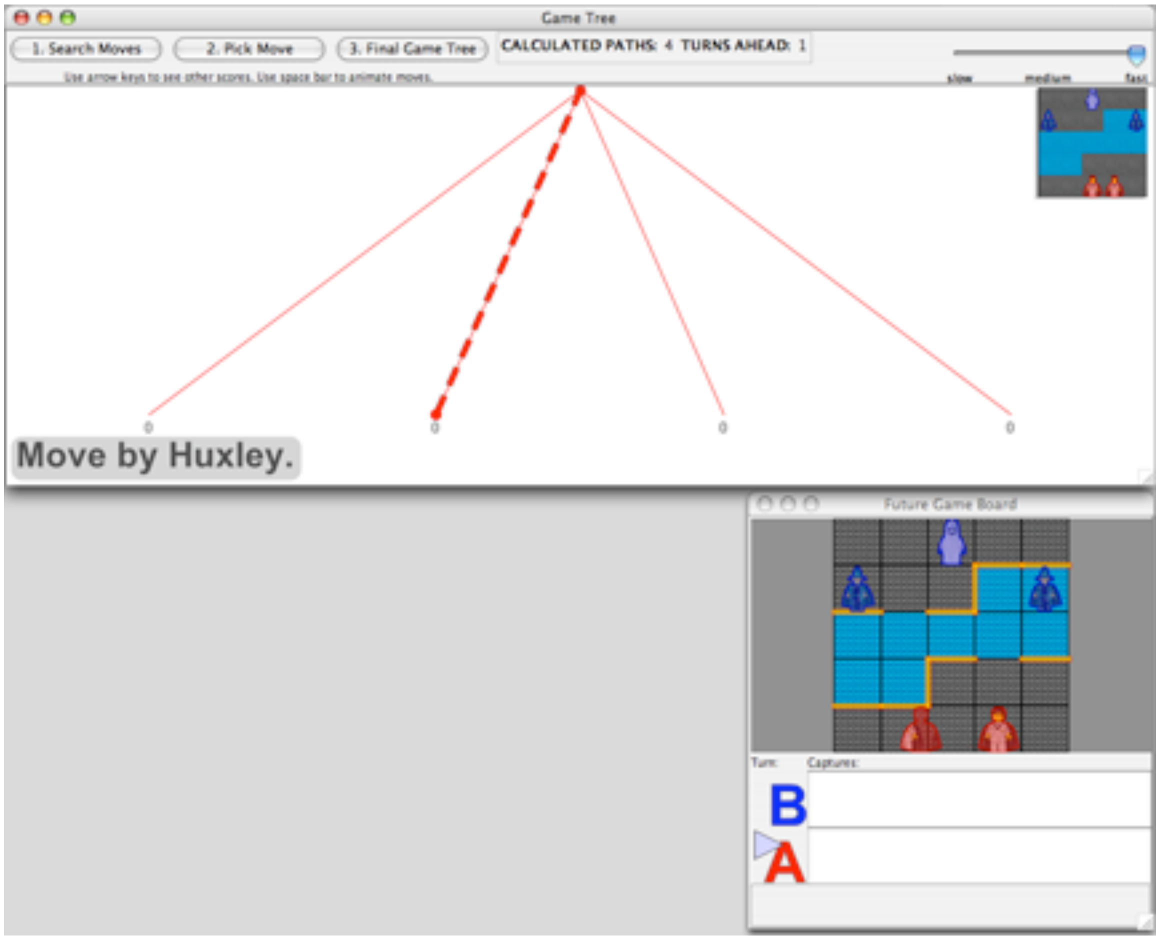

Game Tree
1. Search Moves
2. Pick Move
3. Final Game Tree
CALCULATED PATHS: 4 TURNS AHEAD: 1
Use arrow keys to see other scores. Use space bar to animate moves.
slow
medium
fast
0
0
0
0

FIGURE 2.8 THE GAME TREE DIAGRAM

I continue, "This diagram is called a *Game Tree*. It is like a road map, and you are at the top of the road and you have four ways you can go. Each line represents a move you can make. We can use the arrow keys to select and highlight different moves."

Jim clicks the arrow keys on the keyboard, and with each click, a different line on the diagram is highlighted. Changing the highlighted move with the keyboard also changes the configuration of game pieces on the small *Future Game Board* window in the corner. I explain to Jim, "That is a possible future game board. It shows us what will happen if we make the highlighted move. We can see an animation of the move that leads us to this possible future game board by double clicking on the *Game Tree*."

Jim double clicks and the pieces on the *Future Game Board* momentarily fade away and then reappear in their starting positions. Then one Harry Potter piece moves by sliding into the space to its right. While this move is animated, that branch on the game tree is highlighted with a red colored dashed line and a label below the *Game Tree* reads, "Move by Huxley".



CALCULATED PATHS: 4 TURNS AHEAD: 1

Use arrow keys to see other scores. Use space bar to animate moves.

FIGURE 2.9 THE GAME TREE AND FUTURE GAME BOARD PREVIEW OF ONE OF HUXLEY'S POSSIBLE MOVES

Jim asks, “Who’s Huxley?”

“Huxley is a name for Player A. It helps to distinguish Player A from Player B if we give them names, so we call Player A “Huxley” and call Player B “Robotron.” Now, let me show you how we can look ahead to see all of the countermoves that you counted out earlier, the moves that Robotron can make on his turn.”

I refer Jim to a slider labeled *Number of Moves Ahead*, which has three options: 1, 2 and 3. Jim moves it from 1 to 2, and the *Game Tree* graphic changes: now a bunch of blue lines appear under each of the lines representing Huxley’s four moves. I explain that these represent all of the countermoves that Robotron can make on his turn. (I also remind Jim that blue is the color assigned to Player B and that the colors help to clarify whose turn is whose).

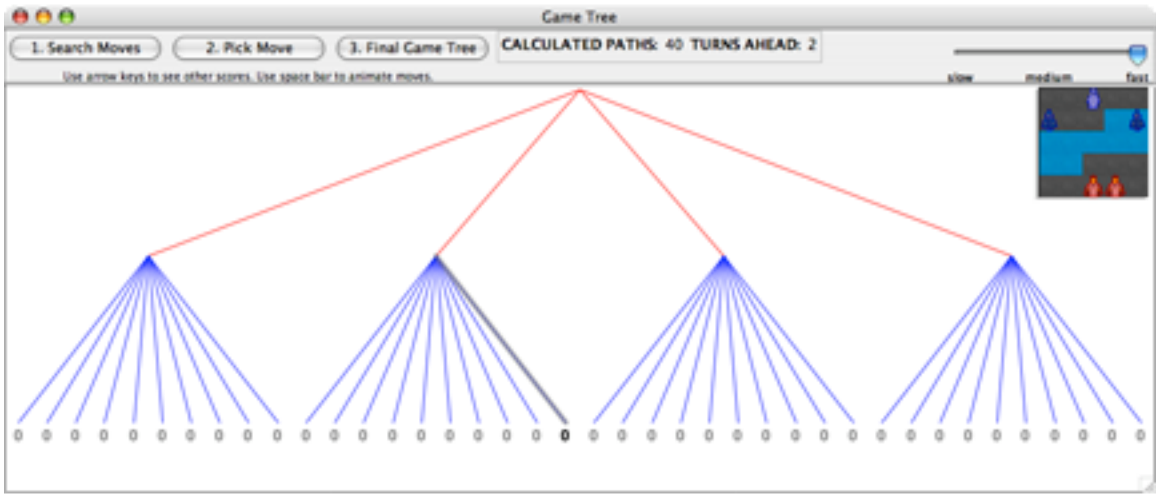

Game Tree
1. Search Moves
2. Pick Move
3. Final Game Tree
CALCULATED PATHS: 40 TURNS AHEAD: 2
Use arrow keys to see other scores. Use space bar to animate moves.
slow
medium
fast

FIGURE 2.10 LOOKING TWO TURNS AHEAD ON THE GAME TREE

Then Jim cranks the slider up to 3 moves ahead and the *Game Tree* gets really thick with branches: under every counter move are a new cluster of red moves. Jim says, "That's a whole lot of stuff when you consider everything everyone can do."

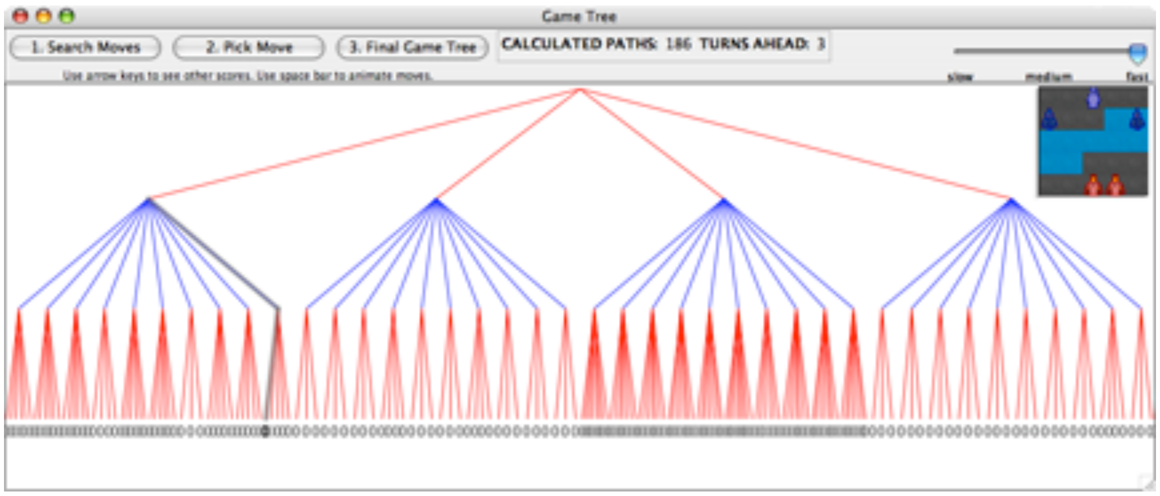

Game Tree
1. Search Moves
2. Pick Move
3. Final Game Tree
CALCULATED PATHS: 186 TURNS AHEAD: 3
Use arrow keys to see other scores. Use space bar to animate moves.
slow
medium
fast

FIGURE 2.11 LOOKING THREE TURNS AHEAD ON THE GAME TREE

I explain, "Even though the computer can plan for all of these moves, right now it doesn't know which of these moves is any good. Do you know which of these four starting moves is better then the others?"

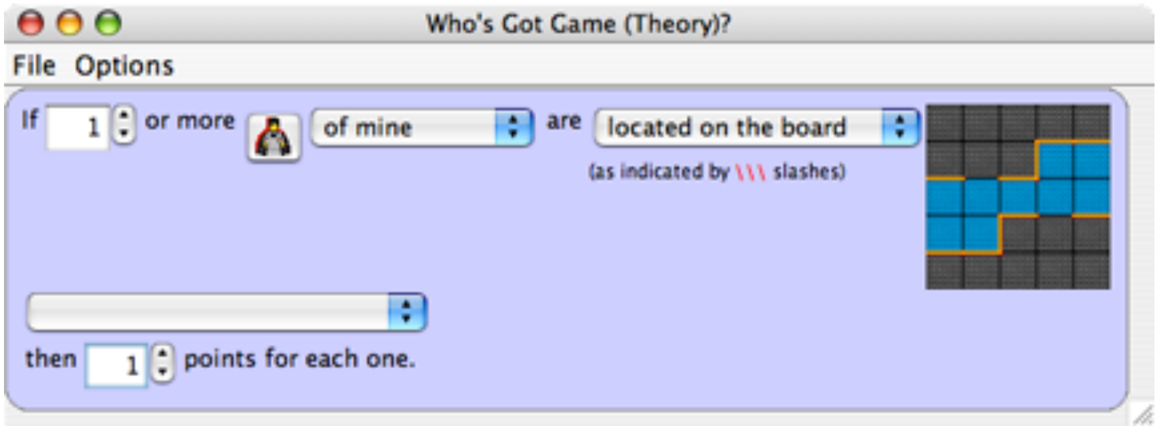
Jim looks at them and says, "They're all pretty much the same, I think."

"Do you think so? We are going to give Huxley hints so he knows which moves are better than others. Then, Huxley will use those hints to come up with a number score for all of these choices on the *Game Tree*. Right now, every move that Huxley knows about is worth 0 points—they're all the same to him, unless it is a winning or a losing move.

Let me show you how we create hints for Huxley to use to figure out which moves are better than others."

2.3.2 Evaluations

Under *Player A's Strategy* tab, there is a panel titled *Hints* with an empty list and three buttons titled *Add*, *Delete* and *Edit*. The *Hints* panel looks like the one Jim used to define his game's winning conditions, but the *Hints* panel has an additional column with the heading *Points*. Jim clicks on the *Add* button, and a sentence screen like the kind he used to define winning conditions comes up, but this sentence ends with "… worth [1] points for each one." instead of "…and I win the game."


Who's Got Game (Theory)?

File Options

If 1 or more of mine are located on the board (as indicated by \\\ slashes)

then 1 points for each one.

FIGURE 2.12 DEFAULT HINT DESCRIPTION INTERFACE

Also, when Jim clicked the *Add* button, the *Game Tree* graphic faded just a little bit. I explain to Jim that after he creates his hint, that it will be used to assign a number score to all of those moves on the *Game Tree*.

"Making hints is like making winning conditions, but the difference is that a number score is assigned to every piece described by your hint. For example, you could assign a score of 1 point for every piece of yours that is on the game board. But you can also make hints that assign points for pieces related in some way to other pieces. For example, you can assign points for pieces that can be captured by other pieces, or spaces that other pieces can move to, or pieces that can be captured by other pieces, but can't catch those pieces. You can define those relationships by using the *relationship* pull-down menu. You could have used these relationships to define winning conditions also."

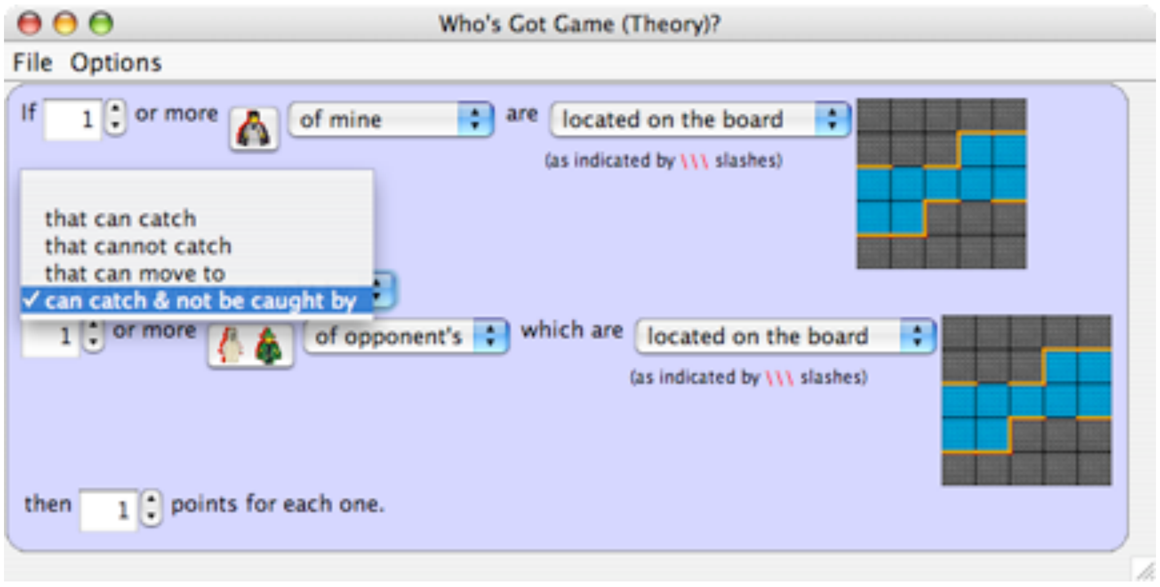

Who's Got Game (Theory)?
File Options
If 1 or more of mine are located on the board
(as indicated by \\\ slashes)
that can catch
that cannot catch
that can move to
✓ can catch & not be caught by
1 or more of opponent's which are located on the board
(as indicated by \\\ slashes)
then 1 points for each one.

FIGURE 2.13 HINT DESCRIPTION INTERFACE WITH RELATIONS PULL-DOWN MENU SELECTED

Jim thinks this over, and then asks, “Okay, so, what’s a good hint?”

“Well, what is important in your game? You lose your game if all of your pieces are caught. You win your game by getting at least one of your two Harry Potters to the other side of the game board. What hint could help you know that you’re making good decisions towards reaching that goal?”

“Having both of my Harry Potters still in the game is good.”

“Okay, let me show you how you tell the computer that that is a good hint. This hint is like Harry’s winning condition, only now you want to say that it is good for a Harry to be anywhere on the game board.” Jim specifies this by using a pull-down menu to make the hint sentence read *anywhere on the board*, which automatically selects all of the spaces on the miniature game board.

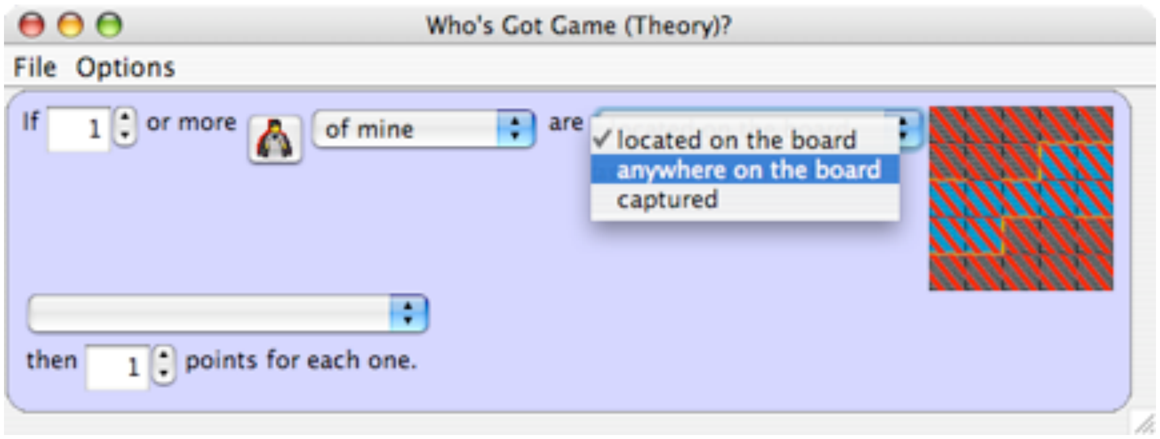
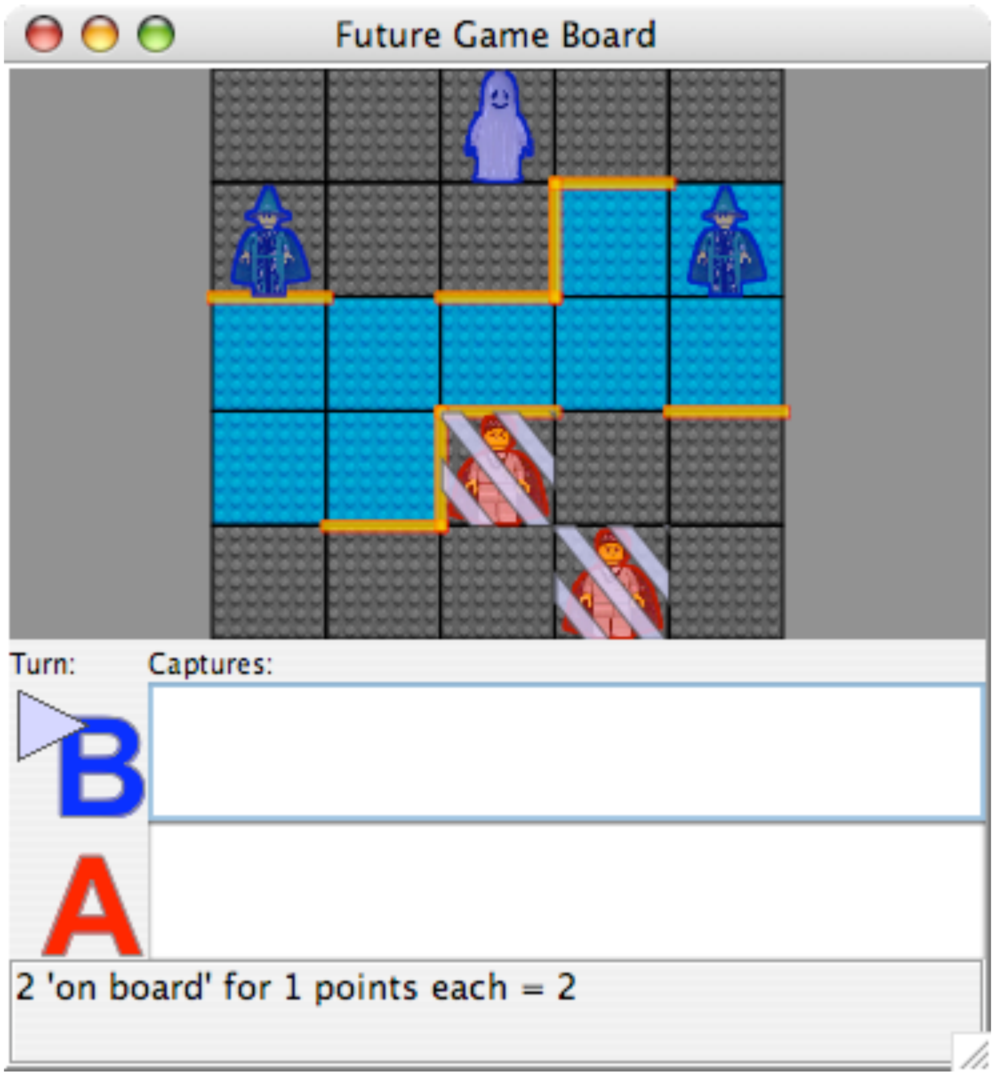


FIGURE 2.14 SELECTING ANYWHERE ON THE BOARD WITH THE HINT DESCRIPTION INTERFACE

Next, I direct Jim to the last part of the sentence that assigns points and explain, “Now, you want to assign a point value for each Harry on the board. By default that value is set to 1 point per Harry.” Jim decides that 1 point per Potter is good enough for now.

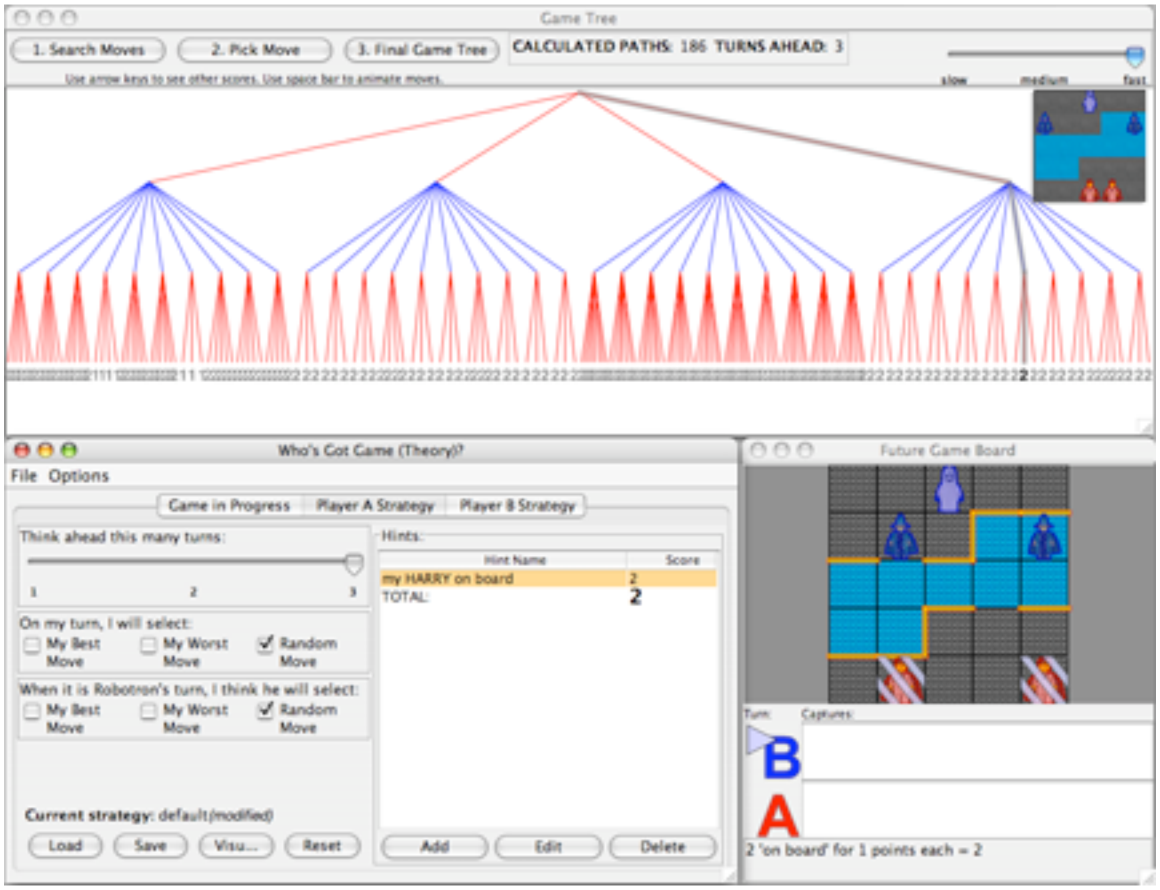
I refer Jim to the *Future Game Board*, which highlights the location of his two Harry Potters on the game board, and underneath a sentence reads, “2 ‘on the board’ for 1 points each = 2 pts.” I explain to Jim that this particular *Future Game Board* is worth two points when we use his hint to score it. I ask Jim to increase the point value on his hint sentence, and when he does this, the sentence on the *Future Game Board* changes to read “2 ‘on board’ for 2 points each = 4 pts.” Jim decides he doesn’t like that change and returns it to 1 point per Potter.



2 'on board' for 1 points each = 2

FIGURE 2.15 THE FUTURE GAME BOARD HIGHLIGHTS PIECES DESCRIBED BY A HINT

Jim saves his hint and this returns him to his *Hints* list, where his hint is now listed. There is also a tabulated TOTAL at the bottom of the list. Since there is only one item on the tabulated list, Jim's hint scored at 2 points, the total number of points is also listed as 2.



CALCULATED PATHS: 186 TURNS AHEAD: 3

Use arrow keys to see other scores. Use space bar to animate moves.

Think ahead this many turns: 1 2 3

On my turn, I will select: My Best Move | My Worst Move | [✓ Random Move]

When it is Robotron's turn, I think he will select: My Best Move | My Worst Move | [✓ Random Move]

Current strategy: default(modified)

Hints:

Hint Name	Score
my HARRY on board	2
TOTAL:	2

2 'on board' for 1 points each = 2

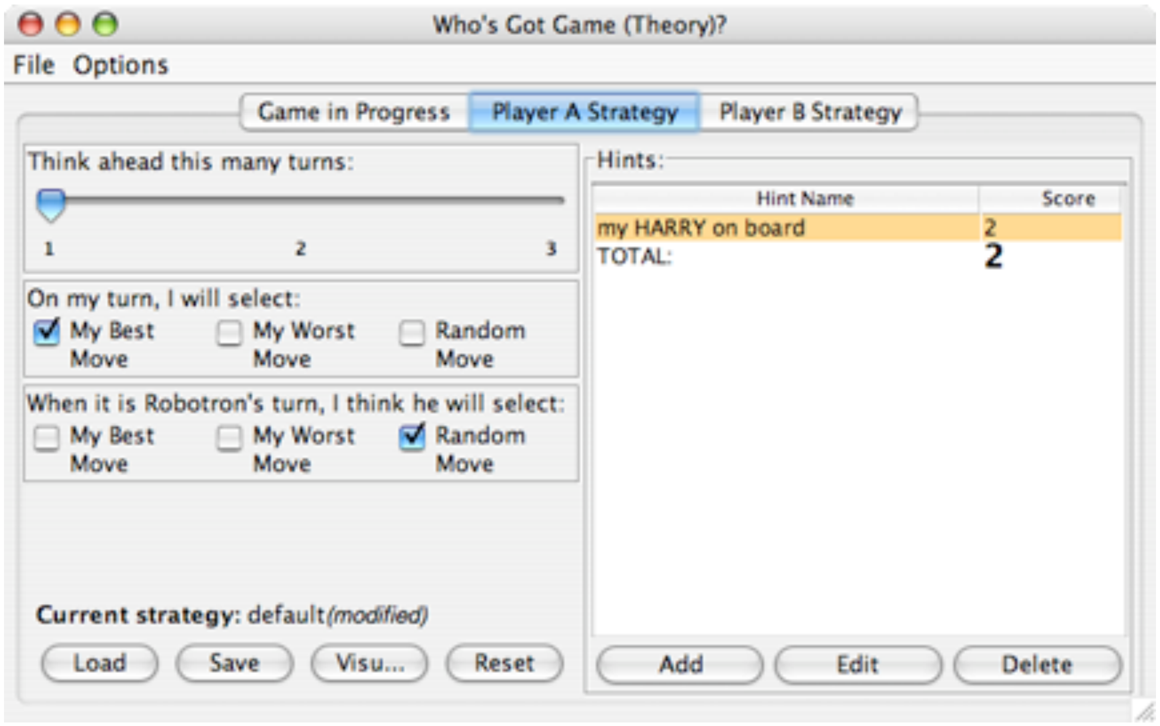
FIGURE 2.16 A HINT IS ADDED TO THE HINTS LIST

On the *Game Tree*, the highlighted move is also scored at 2 points. The other branches at the bottom are now sprinkled with the numbers 1 and 2. I explain to Jim that each of these different number scores indicates how many Harry Potters are on the board depending on which moves are made. Jim browses his mouse through the *Game Tree* graphic and as he passes each branch, a different configuration of game pieces shows up on the *Future Game Board*, each one highlighting the location of the Harry Potter pieces on the game board. A few of these game states only have one Harry, as is indicated by the number 1 on those branches of the game tree.

2.3.3 Selecting Moves

I say, "Now we're going to program Huxley to make the best move when it is his turn. Let's start by keeping it simple and return the *Think ahead this many turns* slider back to 1." Huxley's four starting moves are now the only choices on the *Game Tree*, and each of these moves is scored at 2 points.

I ask Jim which of these moves he should instruct Huxley to make on his turn. Jim thinks about this a second and then says, "Well, these are all the same, but Huxley would pick the move with the highest score. Those are his best moves." I show Jim where he can specify this rule under the *Player A Strategy* tab in a sentence panel that reads, "On my move I will pick: my best move | my worst move | a random move". A sentence ending can be selected by clicking a radio button. "A random move" is currently selected, which is the default setting. Jim clicks "my best move."



Think ahead this many turns: 1 2 3

On my turn, I will select: [✓ My Best Move] My Worst Move Random Move

When it is Robotron's turn, I think he will select: My Best Move My Worst Move [✓ Random Move]

Current strategy: default*(modified)*

Hints:

Hint Name	Score
my HARRY on board	2
TOTAL:	2

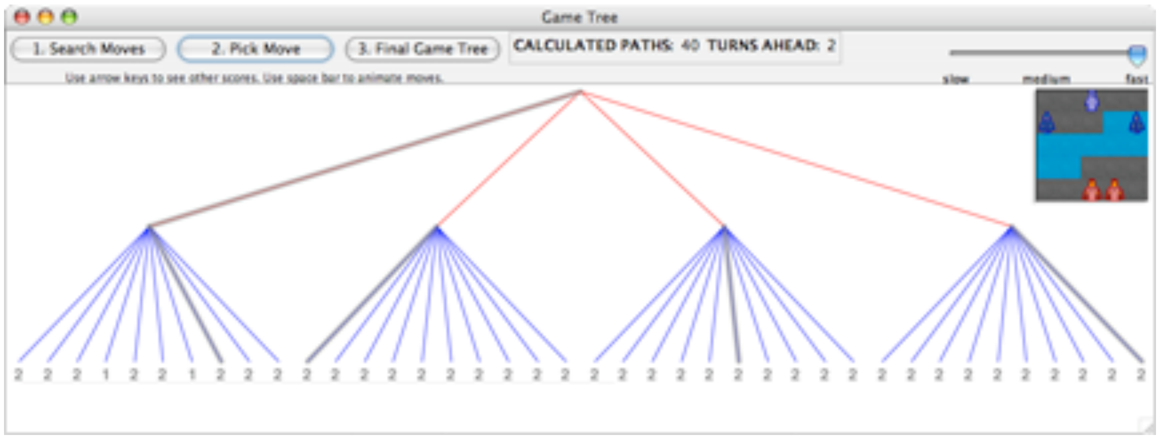
FIGURE 2.17 SELECTING "MY BEST MOVE"

I say, "Now, let's make this a little more complicated. Let's set the *Think ahead this many turns* slider up to 2 so that Huxley is planning ahead for what Robotron might do on his turn." This change brings back the second tier countermoves on the *Game Tree*.

"Now, as we have it programmed, Huxley thinks that Robotron, on his moves, is going to make a random move. That is specified here:" I refer Jim to another sentence panel with multiple endings that reads, "I think that Robotron, on his turn, will pick: my lowest score | my highest score | a random score". It is currently set to the default value, "a random score".

I ask Jim, "What sort of move do you think Robotron will pick on his turn?"

Jim thinks for a bit, and then selects, "my best move", and four of the highlighted branches on the second tier are switched to select 2-point moves. One of the highlighted branches on the second tier connects to a highlighted branch on the first tier. I explain to Jim, "the highlighted path on the *Game Tree* shows both the move that Huxley plans to make and also what Huxley expects to happen on the next turns. Huxley is making his decision based on those expectations. Let me show you the process Huxley goes through to make that decision."

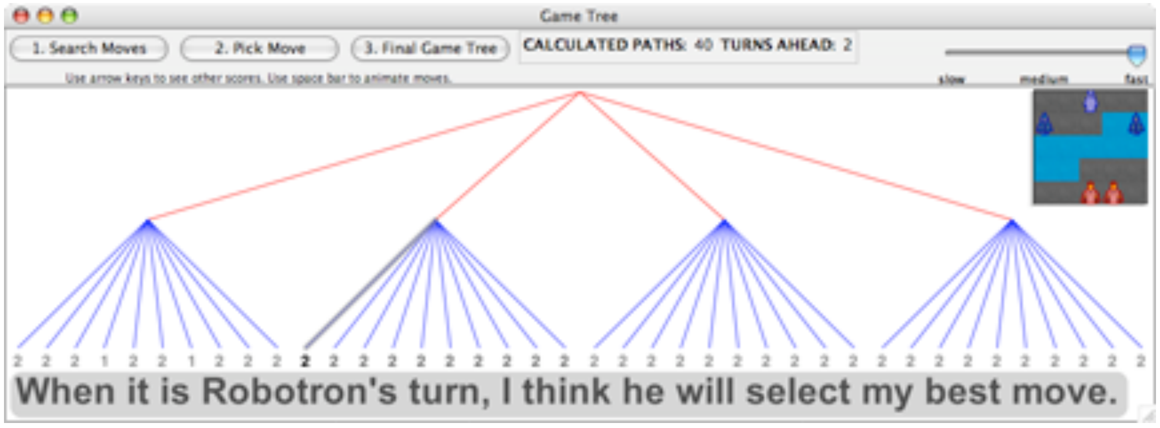


CALCULATED PATHS: 40 TURNS AHEAD: 2

FIGURE 2.18 THE GAME TREE WITH EXPECTED MOVES AND COUNTERMOVES HIGHLIGHTED

I explain, "Huxley knows that Robotron has a range of countermoves. Looking here on the *Game Tree*, we see that Huxley also knows how good these countermoves would be for him: scores of 1 and 2. As you have programmed Huxley, he expects that Robotron will pick a high scoring move on his turn."

While explaining this to Jim, I mouse through the four clusters of countermoves, and the *Game Tree* responds by highlighting a 2-point countermove branch as I pass through each cluster. A label on the *Game Tree* also updates to read, "When it is Robotron's turn, I think he will select my best move."



CALCULATED PATHS: 40 TURNS AHEAD: 2

When it is Robotron's turn, I think he will select my best move.

FIGURE 2.19 MOUSING THROUGH LOWER BRANCHES OF THE GAME TREE HIGHLIGHTS EXPECTED COUNTERMOVES

I continue, "Huxley assigns a score to each of his four initial moves based upon these expectations of what Robotron will do next." Mousing up to the top tier of the *Game Tree*, a highlighted trail is drawn from the top of the tree down to a 2-point branch at the bottom. A label on the *Game Tree* also updates to read, "On my turn, I select my best move."

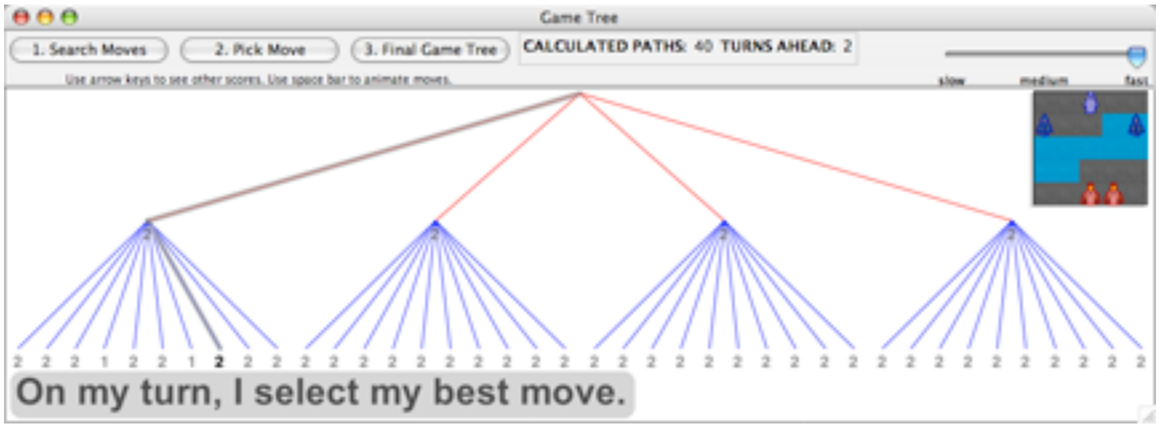

Game Tree
1. Search Moves
2. Pick Move
3. Final Game Tree
CALCULATED PATHS: 40 TURNS AHEAD: 2
Use arrow keys to see other scores. Use space bar to animate moves.
slow
medium
fast
On my turn, I select my best move.

FIGURE 2.20 MOUSING THROUGH THE UPPER BRANCHES OF THE GAME TREE HIGHLIGHTS PLANNED AND EXPECTED MOVES

“Let me recap the decision-making process. First, Huxley thinks of all his possible moves and countermoves. Then, Huxley scores all of those possible countermoves. Next, Huxley makes an assumption about what Robotoron would do in all of those possible scenarios. Finally, Huxley makes a decision about which move to make based on those assumptions. Let me show you an animation of that process:”

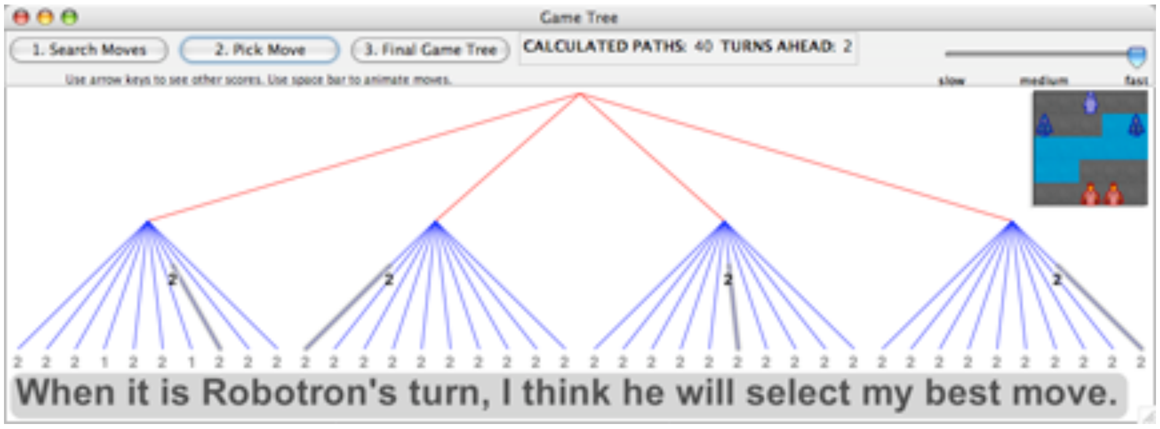

Game Tree
1. Search Moves
2. Pick Move
3. Final Game Tree
CALCULATED PATHS: 40 TURNS AHEAD: 2
Use arrow keys to see other scores. Use space bar to animate moves.
slow
medium
fast
When it is Robotron's turn, I think he will select my best move.

FIGURE 2.21 ANIMATION OF EVALUATION SCORES CLIMBING BRANCHES OF THE GAME TREE

I click the *Pick Move* button on the *Game Tree* and that begins an animation of a high ranked score in every cluster of countermoves climbing up their branch, leaving a trail behind them. When those scores reach their junctions with the first tier, then only one score continues to climb to the top of the *Game Tree*.

I say, “Now that we understand Huxley’s decision-making process, let’s examine this particular decision he is making a little more closely.” I move the mouse into one of the clusters of countermoves on the *Game Tree*, thereby highlighting a move scored at 2-points which updates the pieces on the *Future Game Board*. “Now, this is a preferable outcome. This is what Huxley expects will happen. Let’s preview the moves and countermoves that would happen in order for this to be the outcome.” I double click on the *Game Tree*, and the pieces on the *Future Game Board* return to their starting positions, and then slide through the move and countermove leading to the 2-point future state. As the game pieces move, the selected branch of the *Game Tree* is highlighted with a dashed line and labels on the *Game Tree* are updated to read, "Move by Huxley" and "Countermove by Robotron".

"We both agree that this selected move is a good outcome. But, let me ask you a question: is this a likely outcome? When Robotron makes his move, he doesn't have to choose a 2-point move; he could choose a 1-point move. This is not as good of an outcome for us—we've lost one of our Harry Potters in this possible future. Let's watch how this could happen." I select a 1-point countermove branch and double click on the *Game Tree*. The move and countermove leading to that possible future play out on the *Future Game Board*, resulting in a Harry Potter capture.

Jim thinks about this, and agrees it is rather unlikely that his opponent would pass up an opportunity to take one of his pieces. "Yeah, I guess you're right. Robotron wouldn't make a move that leaves me so well off. He wants to beat me."

"So how can you program Huxley to think that way?"

"I can change his programming in this sentence panel to read: 'I think Robotron will pick my worst move'. But this way of thinking is a little confusing."

"What do you mean?"

Jim paused for a while. "His best move is my worst move. What is bad for me is good for him. I was thinking earlier that I was somehow describing what was good for him… but it's all about what's good for me… and Robotron keeping me from getting there."

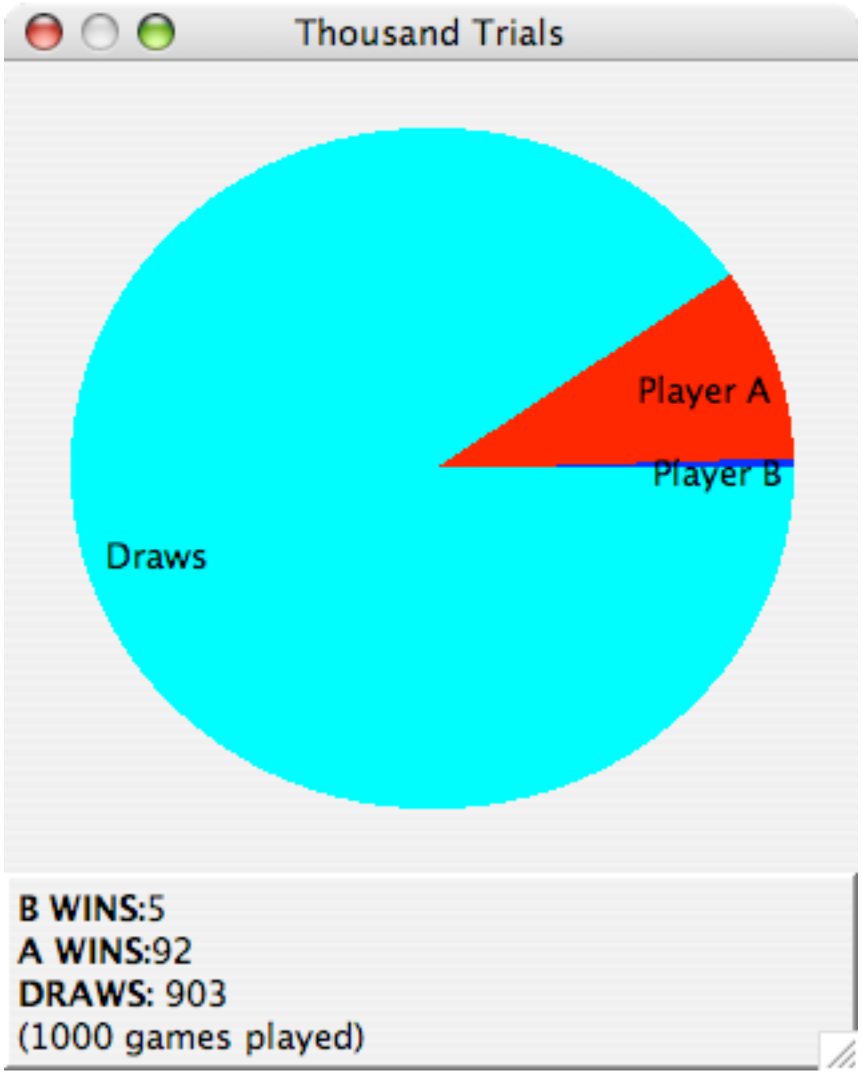
2.4 Strategy in Context

I suggest that we play Jim's game by clicking the *Make A.I. Move* button on both Huxley and Robotron's turn. When Huxley makes his move, Jim sometimes looks baffled and remarks, "what a stupid computer." When I ask Jim what he means, he says that there were clearly good moves Huxley could have made but did not. Jim says that better hints are in order after we finish playing this game through.

When it is Huxley's turn in the game, we refer to the *Game Tree* graphic and browse the different scores of 1 and 2 to see what Huxley is thinking about. After a few turns, the letters W and L also appear on the *Game Tree*'s branches. I explain that these letters represent potential wins and losses.

"Winning is an infinitely high number, and losing is an infinitely low number. So when Huxley has to decide on his turn between a 1, a 2, and a win, he will pick the win since that is, by far, the highest number."

After playing a game through to the end (a win for Huxley), I suggest that we try playing again a few more times. But instead of playing turn by turn, I show Jim a way to run through many games at top speed with a menu option called *Thousand Trials*. Selecting this option brings up an animated pie graph. One of the pie slices is colored Player A red, another pie slice is colored Player B blue, and the last slice is colored cyan and labeled "Draws." The colored pie slices change size every few seconds, reflecting the outcome of games being played inside the computer's memory. Underneath the pie graph is an information panel listing the number of Player A wins, Player B wins, and draws as the total number of games played grows towards one thousand.



B WINS:5
A WINS:92
DRAWS: 903
(1000 games played)

FIGURE 2.22 THE THOUSAND TRIALS

Jim asks, "How does a game draw?"

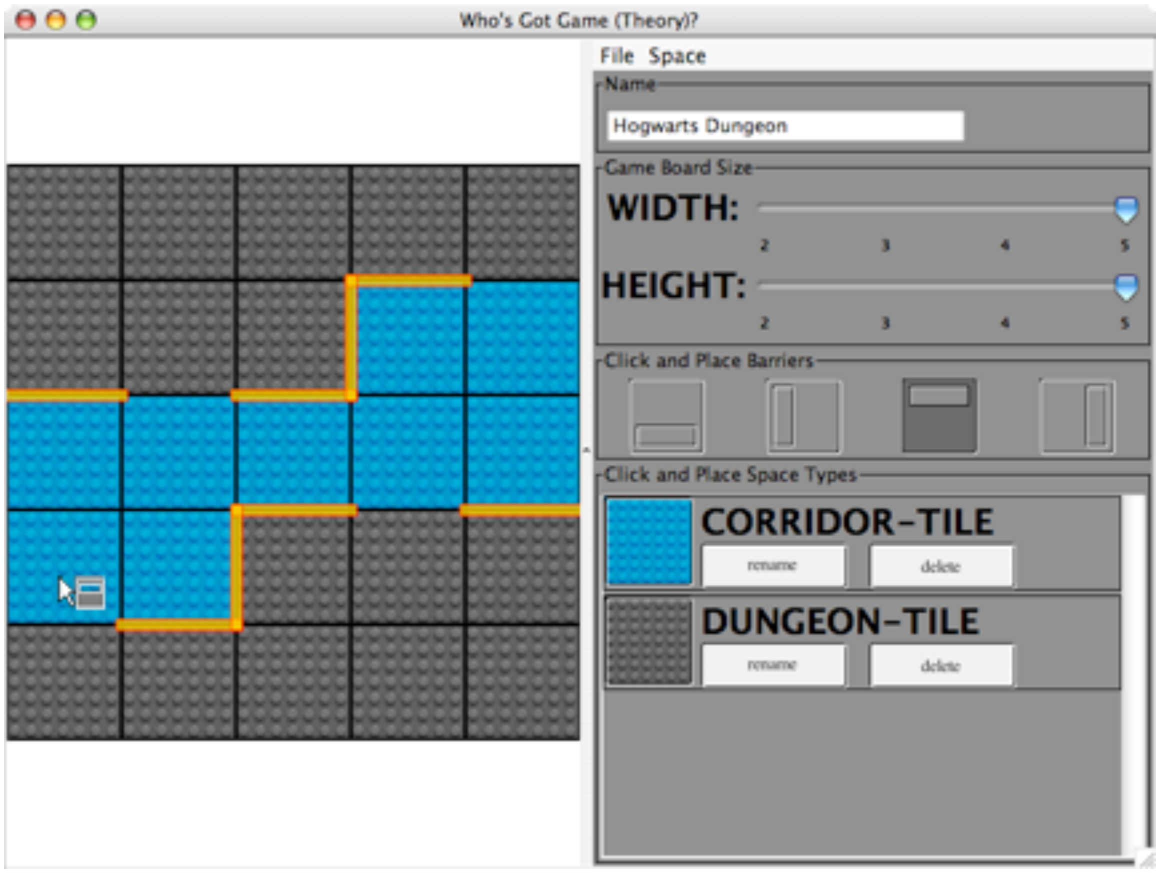
"There is a draw if the game pieces are in the same spaces six times or if one of the players has no moves left to make on their turn."

The pie graph oscillates wildly at first, and then, after a few hundred games, settles into a general shape: a lot of draws (90%), some wins for Player A (9%), and only a sliver of wins for Player B (1%).

2.4.1 Changing the Game

"Harry is winning, but he isn't winning as much as he should. I think my game is too hard for Harry," says Jim. "Can I change the game?"

I tell him that is a good idea and that the toolkit is designed so that he can easily make changes to the game board, to the pieces, or the rules. Jim pops out of the game-playing screen and back into the board-design screen, and then deletes one of the barriers. A barrier is removed from the mirror side of the game board as well.

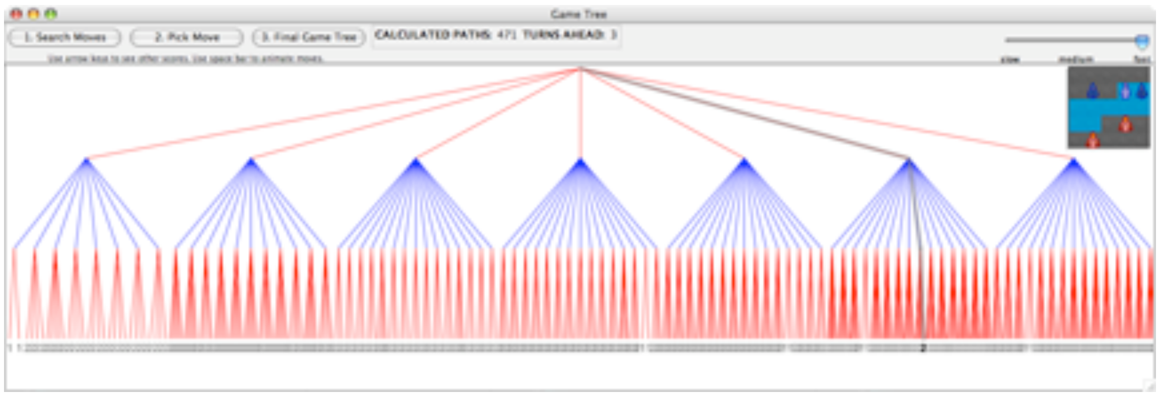


Hogwarts Dungeon

FIGURE 2.23 REMOVING A BARRIER WITH THE BOARD-DESIGN TOOLS

"That's a secret passage," Jim explains. "But it's not too secret since everyone on the game board knows about it. But that's okay."

He returns back to the game-playing screen and then reloads his game with the new secret passage. Jim plays his game for a bit, moving through one of the new passages. The strategy he had designed earlier remained intact, so I tell Jim to take a look at his *Game Tree* to see the changes. The game tree now has many more red branches then before his changes to the game board, and under every one of these new branches are many more blue countermove branches.



CALCULATED PATHS: 471 TURNS AHEAD: 3

Use arrow keys to see other scores. Use space bar to animate moves.

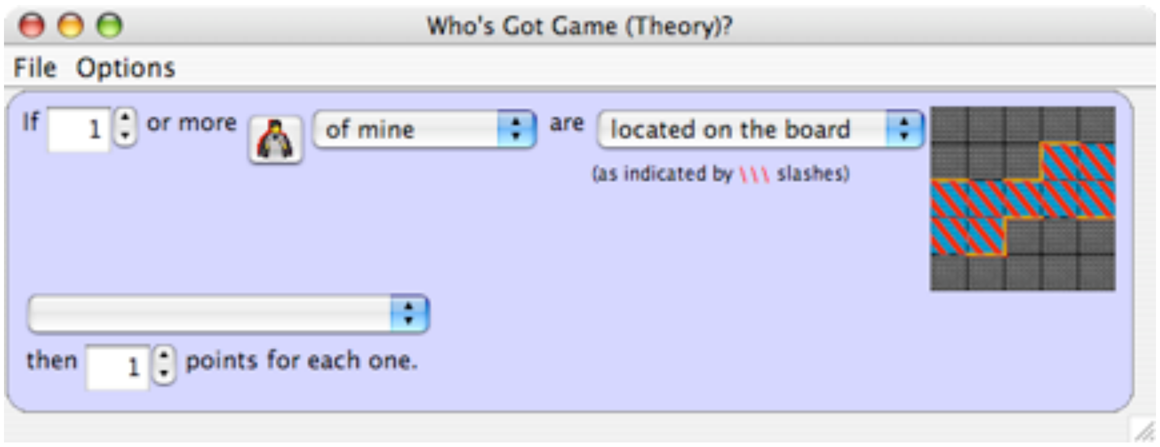
FIGURE 2.24 THE GAME TREE WITH MORE BRANCHES

Jim tries running his strategy through the *Thousand Trials* again. As the pie graph changes shape, Jim sees that his change has only made it slightly easier for Harry Potter pieces to get across the game board. It shouldn't be this hard for Huxley to win; Jim acknowledges that his strategy needs some work.

2.4.2 Different Heuristics

I suggest that Jim think about what behavior to expect from the one hint he has given Huxley, and he replies, "Well, I guess Harry just looks ahead and tries to avoid getting captured but never makes it very far. All he knows is to avoid getting captured, and he can't think far enough ahead to find the other side of the board."

Excited by this insight, Jim asks if he can add another hint to his strategy. I explain that he can make many hints, and they will be summed together to score moves. Jim's new hint is to assign 1 point for every Harry Potter in the middle corridor, which is the region halfway across the game board.



If 1 or more of mine are located on the board

(as indicated by \\\ slashes)

then 1 points for each one.

FIGURE 2.25 A HINT FOR A HARRY POTTER GAME-PIECE TO BE IN THE MIDDLE OF THE BOARD

Now Jim retries the *Thousand Trials* and finds that he is winning most of his games. I ask him what he's done and he replies, "Well now Harry knows where to go. Before he was being hesitant, looking ahead to make sure he didn't get caught, but he didn't think at all about where he had to go. Now Harry thinks about keeping himself safe, but he also knows to get closer to his goal. And I know how to make him even better."

Jim then adds another hint; this one says that it is worth 1 point for Harry to capture any of his opponent's pieces. Then Jim reruns the *Thousand Trials* and does considerably worse than he expected.

Jim moans, "This isn't what is supposed to happen!"

Prompted to explain, he thinks and says, "I guess Harry is being too aggressive. I mean, he thinks that zapping a teacher is just as good as getting to the other side of the board. But when he is looking to zap teachers, they can zap him too. Harry thinks that capturing pieces is just as important as getting where he has to go to win. I need to change that."

2.4.3 Scaling Heuristics

Jim changes his original "1-point per Harry Potter on the board" hint up to 3 points and reruns the *Thousand Trials*. Jim is proud to see that Huxley is winning many more games than before.

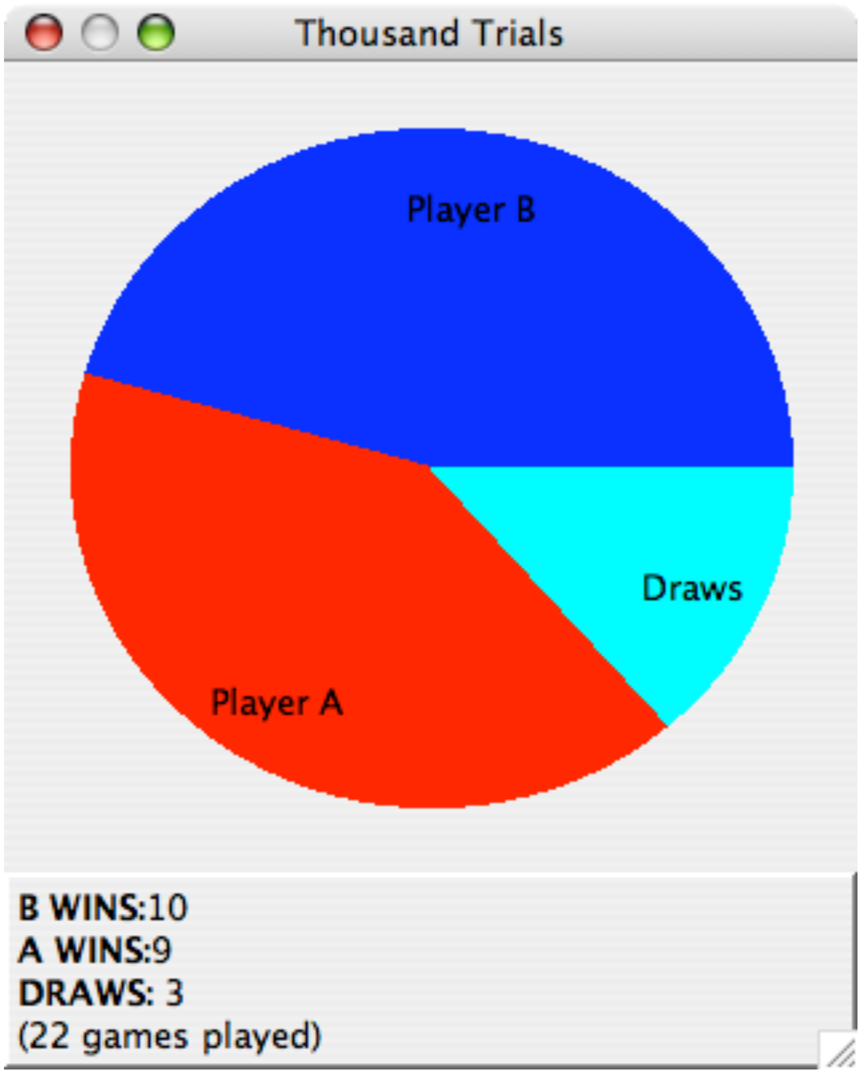
Jim explains, "Now Harry knows to stay alive, which is most important. Harry also knows to get to the other side by going through the corridor, and, if he sees a teacher that is easy to zap on the way, he'll capture them too. But he won't go out of his way and get into trouble."

2.5 Opposition

I congratulate Jim, "Well, I think you've done a good job, but you do know Robotron has just been making random moves. The Hogwarts staff was not even looking for Harry unless he happened to be under their noses on their turn. Let me give Robotron a hint of his own and see how the game turns out."

I switch over to Robotron's control panel and add a hint that assigns 1 point for every Harry captured. I explain, "Since all the Hogwarts staff has to do is capture two Harry Potters, a pretty good hint is capturing at least one. Capturing the second Harry is a winning move, so we don't need to give Robotron that hint." I set Robotron to look one move ahead and to pick his maximum score on his move.

We run the thousand trials, and Jim sees that he now has some competition: Robotron is winning a few games. Then, I crank up Robotron's *Think ahead this many turns* slider to 3, and update his other sentence to read, "On Huxley's turn, I think he will pick my worst move." Then, we run the *Thousand Trials* again, and we see that Huxley and Robotron are just about equal now in wins. I explain to Jim, "You made Harry think ahead to be cautious and avoid getting caught, but no one was really looking for him. Now I've made it so everyone at Hogwarts is planning ahead to cut Harry off to keep him from getting away!"



B WINS:10
A WINS:9
DRAWS: 3
(22 games played)

FIGURE 2.26 THE THOUSAND TRIALS WITH NEW STRATEGIES AND A MORE EQUAL DISTRIBUTION OF WINS

We are just about of time for our workshop, and Jim says he really liked playing Harry Potter and would like to do it again sometime. He has some new ideas for strategies he would like to try out and also some ideas for how to change the game.

2.6 Summary

"It was as though the potion was illuminating a few steps of the path at a time: He could not see the final destination… but he knew that he was going the right way…" (Rowling 2005)

- Harry Potter and Half-Blood Prince

In this citation from the Harry Potter book series, protagonist Harry has just swallowed a good luck potion that gives him intuition about where to go and what to do. The effects of the good luck potion are a little bit like what Jim programmed Harry to plan for and anticipate in this chapter.

Let's review some of the things Jim learned by playing with this software toolkit:

- **Don't paint yourself into a corner.** By designing his own game, Jim learned that it was good to keep his pieces' options open by removing barriers that limited their movement. The more moves his pieces had available, the more ways Jim could find a way to win.

- **When you get to a fork in the road, stop and ask for a road map.** Jim had a difficult time enumerating all of the choices he had available to him in his own game. Enumerating all of his countermoves was exhausting. The *Game Tree* graphic helped him to understand and preview all of his choices.

- **If you can't see the end of the road, you're going to need to ask for directions.** Early in the game, the *Game Tree* wasn't very helpful in finding that moves led to a win until Jim created some hints. Once Jim created hints, he was able to think systematically about his choices.

- **I think that you think that I think…** Jim had to anticipate what his opponent was going to do in order to win games. This also led to his insight that "what is good for them is what is bad for me", which is a way of explaining the minimax algorithm.

- **You win some, you lose some.** By testing his strategy iteratively in the *Thousand Trials*, Jim was able to get a better sense of performance benchmarks then when just playing his game turn by turn.

3 Design Rationale

In this chapter, I explain my approach to making some complicated mathematical ideas understandable and useful to students. I begin by comparing formal theoretical and algorithmic representations of minimax with my own constructionist representation; over the years, the minimax algorithm has been used to effectively play various strategy games with computers, so I explain where my thesis fits in that tradition. Next, I introduce previous efforts to help children become computer strategy-game designers and explain how this work fits in that tradition. I conclude the chapter with a constructionist explanation and rationale for design features of the gaming calculator.

3.1 Game Theory

After playing poker with friends in 1928, mathematician John Von Neumann speculated about an optimal strategy. His approach to this problem went beyond the rules of poker to a theoretical analysis of all games. Game theory, aptly named, describes mathematically provable methods to increase the probability of winning games. Put differently, it is the systematic evaluation of all choices, often in competitive situations.

Von Neumann's analysis of strategy was not simple; his game theory requires an understanding of advanced mathematics and notational systems: it took a mathematician to turn parlor games into homework. Consider, Von Neumann's Minimax Theorem, a core game-theoretical contribution:

$$\max_{X} \min_{Y} X^T \mathrm{A}\, \mathrm{Y} = \min_{Y} \max_{X} X^T \mathrm{A}\, Y = v$$

FIGURE 3.1 THE MINIMAX THEOREM (WEISSTEIN)

This thesis makes some core ideas of game theory available to children by eliminating such formalisms and making the math relevant to their experience playing games.

Because the Minimax Theorem describes a decision-making process, it can be translated into an explicit algorithm, making it a good candidate for solving with computational methods. Even before computer hardware was available, Alan Turing designed software to play the game of chess and computed a game by hand. (Since even before their inception, computer games have been highly motivating activities!). The "computer opponent" lost, but Turing's original algorithm, a step-by-step implementation of Von Neumann's theorem, is applicable to any number of decision-making problems when abstracted:

```
function MINIMAX-DECISION(game) returns an operator
    for each op in Operators[game] do
        VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return

function MINIMAX-VALUE(state,game) returns a utility value
    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

FIGURE 3.2 THE MINIMAX ALGORITHM (RUSSELL AND NORVIG 1995)

Even when expressed in this explicit algorithmic representation, minimax remains impossibly complex for most people. However, if the minimax algorithm's operations are described in colloquial language, the idea isn't nearly as esoteric. The crucial idea of minimax works roughly like this: *anticipate the worst things your opponent could possibly do to you and then plan to make the best of those bad scenarios*. A more elaborate explanation can be found in many textbooks, and, for the most part, instructional aids like textbook descriptions and diagrams are how these concepts are taught to students.

I take a constructionist approach to game-theoretical mathematics by translating the minimax algorithm into natural-language syntax easy enough for secondary-school students to manipulate. I hypothesized that students could learn the math of game theory by experimenting within the familiar context of various strategy games, instead of being lost in abstract formalisms. Consider the natural-language description of minimax found in the toolkit described in this dissertation:

> Think ahead this many turns: [3]
> On my turn, I will select [my best move].
> When it is my opponent's turn, I think they will select [my worst move].

By changing the options in this description of the minimax algorithm, students can, for example, change how far ahead to search, or alter their assumptions about how their opponent will play. In addition to winning competitive games with minimax, the toolkit allows students to easily modify the algorithm to play anti-games in which the goal is to lose (maximin), for both sides to lose (minimin), or to assume that their opponent wants them to win (maximax), to play randomly, or assume their opponent will play randomly. This strategy-description language is coupled with real-time animation and descriptions of the decision-making process to help learners understand the effect of their changes. The overarching pedagogical goal of this approach is to make the mathematics of decision-making useful to students by representing it as a design medium.

3.1.1 Thesis Scope

There are many sorts of games, but this thesis is limited to one type: two-player, turn-based, zero-sum games of complete information; in short, games like chess and checkers. Conway (Berlekamp, Conway et al. 2001) is known for most fully exploring mathematical representations of this genre of games, referred to in game-theoretical parlance as combinatorial games.

Chess, and the games generated by the gaming calculator, are games of **complete information** because both players can see all of their opponent's pieces and therefore can calculate all potential moves. The games are also **zero-sum**, as only one player can win, and when that happens, the other player has lost. Allowing only games of this type eliminates certain strategies: since both players can see all of their opponent's options, probability is out of the scope of the decision-making process. In Chapter 5, I describe how my pedagogical approach can be applied to other game genres.

3.2 Deep Blue, Wide Blue, and Kiddie-Pool Blue

Since Turing's hand-computed chess strategy lost its first game, it has been a long-standing goal of computer science to create a computer chess opponent good enough to beat expert human players. This goal was achieved by IBM's Deep Blue supercomputer in a series of games against Grand Master Kasparov in 1997. Deep Blue won by brute force: using the minimax algorithm, amongst others, it searched ahead billions of possible moves and countermoves. However, although Deep Blue is very adept at playing chess, it doesn't know how to play any other games; even tic-tac-toe and checkers are beyond it.

Recognizing this deficiency, some computer-science research has focused on the development of a general-purpose opponent for any strategy game (Pell 1993; Orwant 1999). Keeping with IBM's nomenclature, these general-purpose computer opponents could be referred to as Wide Blues: give them the rules to a strategy game and they'll play. These general-purpose game players use the minimax algorithm to search through game options, but how they evaluate their choices has to be more flexible than Deep Blue since they don't know which game they'll be asked to play.

If Deep Blue can only play one game really well, and Wide Blue can play any number of games reasonably well, then my software toolkit could be called Kiddie-Pool Blue: it can play a variety of strategy games, so it is reasonably wide, but it needs user intervention to help it play any of these games well, and it is designed for students (and kiddies of all ages). By allowing students to design and modify games and then experiment strategically within those games, I have sought to provide robust learning opportunities for understanding strategy within various contexts.

3.3 Learning Programming versus Learning Strategy

In the late 1960s, researchers began to study how children could design and program their own computer games (Papert and Solomon 1970). The first documented educational computer-game design research was a project teaching 7th graders how to program the relatively simple strategy game of *NIM*. In *NIM*, two players alternate taking 1, 2, or 3 matches from piles; they win by not taking the last match. Theirs was an ambitious project for young students: learning a programming language, then building an entire game, including the game representation, interface, and an automated strategy. A first version of the opponent made random moves and a later opponent could simulate "thinking ahead" by picking up an odd number of matches, thereby always leaving the other player with a match to pick up on their turn. Programming this particular strategy is helpful for learning how to win *NIM*, but is almost never applicable to any other game.

Later, Brady and Emanuel argued that, instead of students learning about low-level computer programming, students could use a high-level strategy language to work on the more interesting problems inherent in computer games (Brady and Emanuel 1978). They created a *NIM*-strategy language that consisted of if-then rules, a relatively straightforward way to describe decision-making. They also proposed expanding their strategy language to describe ways to win more sophisticated games, but eschewed teaching children to program game-tree search algorithms, instead advocating that concepts like minimax could be represented in a better way. This thesis is, in some ways, a development and implementation of Brady and Emanuel's proposed strategy language and an evaluation of what students learn when they use it to play games.

3.4 Constructionism

Piaget's **constructivist** learning theory attributes knowledge acquisition to learners actively constructing knowledge structures in their mind. **Constructionist** learning theory asserts that providing real-world construction materials (e.g., construction paper, toy blocks, computational toolkits) can help learners build robust mental models (Papert 1980). In other words, constructionism aims to provide appropriate modeling materials for mental models. Computers, the machine of machines, provide nearly unlimited constructionist opportunities because of the variety of possible design activities. Another benefit of computation is that processes must be explicitly represented; this creates opportunities to consciously reflect upon the process by which design decisions are made.

My game-design research was informed by constructionism, as I wanted my students to gain mathematical fluency through design activities. A core tenet of constructionist theory is that people learn best when they build personally meaningful artifacts, and this is one reason why the gaming calculator allows students to be both game and strategy designers. My approach was to have students create their own math problems (their games) and then solve them (their strategies), all the while explicitly supporting logical processes and mathematical discovery.

3.4.1 Debugging Processes

The processes by which most software works is opaque; we intuit behavior through interactions on the screen (Turkle 1995). When software breaks or doesn't work as expected, we often have no way to "look under the hood" to try and figure it out ourselves. Part of the problem is that computer processes are most often represented statically in text files. It is hard to see how these descriptions of software's control structures, iterations, and recursions are tied to the execution of these processes.

Some specialized programming environments provide an animated representation of processes to help learners understand how software functions. The original *Logo* programming language is helpful in this way, as every programming step literally left a trail behind it, a visual trace of behavior upon which the programmer could reflect while debugging. But the usefulness of visual paths in *Logo* is a special case in that *Logo* was used to teach geometry and paths define geometric shapes.

When programming complex behavior, visually tracing the steps for debugging the decision-making process is not always straightforward or easy. In the computer programming toolkit *ToonTalk*, algorithmic processes are designed by staging cartoon characters and process execution is visualized as an animation (e.g., a mouse runs on screen to sum numbers, a bird transfers information between processes) (Kahn 2000; Kahn 2001). These animations visualize steps as they occur at a granular level and must be viewed in succession to glean their context as part of a larger process.

In contrast, algorithm animation in the gaming calculator provides a high-level diagrammatic overview of procedures described by game tree algorithms. My rationale for using diagrams is to make algorithmic processes easier to understand: diagrams make decision-making processes visible and traceable. Animated diagrams go further by indicating the current state of a complex process, thereby providing an opportunity to visually and chronologically trace progress. Because diagrams in the gaming calculator are dynamically generated to represent the specific context of students' games, and because they are linked to graphical representations of those games, the diagrams are high-level tools for reflection: they help the students learn about decision-making. The specific details of the graphic representation are discussed in the next chapter.

The gaming calculator's animations are designed to make debugging easier. If your program is not working as expected, you have likely encountered a procedural bug in your own thinking about a problem. Visualizing the steps of a procedure provides a representation to help identify and understand what isn't working.

Supporting the learner in debugging assists learning in the constructionist tradition, as reflecting on the steps of a complex process can lead to deeper understandings. For example, thinking about the explicit steps involved in juggling (Papert 1980) or knot-tying (Strohecker 1991) can help learners understand how simple steps can create complexity. Similarly, reflecting on the steps in a decision-making process can help learners to fix mental models of a strategic thinking process.

3.4.2 Collaborative Learning

Well-designed social experiences can be used to construct meaning in the mind of a learner. Good constructionist environments provide students with an audience for their work in order to support collaborative learning. For example, students who were given the explicit task to design computer games to teach other students mathematics took their role of teacher seriously and based their designs on what they thought other students knew (Kafai 1995). Collaboration can be integrated into the design of a constructionist software environment, creating an implicit peer-tutoring environment like an online

MUD (Bruckman 1998). With these examples in mind, I designed my software and workshops to support collaboration around student-designed games.

It might seem like an oxymoron to create a collaborative learning environment around a competitive activity, but students in my workshops moved between these two modes with ease, with a greater emphasis on collaboration. As is discussed in Chapter 5, students savored competitions, and when preparing for a competition, would sometimes work in secret. After each game series, students would compare the performance benchmarks of their strategies and then discuss, examine, and appropriate each other's code to figure out what did and did not work.

I created a modular game system so that game boards, pieces, and strategies are interchangeable (e.g., you can import chess pieces into your checkers game as well as try out a strategy from one game in another). This was a design decision to encourage collaborative design projects. Indeed, students in one workshop designed their game collaboratively on separate machines and then integrated their pieces on one computer and into a final product (well, not exactly final; they continued to revise as they went along). I also made the source code of every object viewable and editable so that students could build on each other's work. This design decision was informed by the experiences of students who used Bruckman's MUD design toolkit *Moose Crossing*, in which students created virtual objects of personal interest and showed them off to other students, who would then view the source code of these objects and subsequently modify them to make their own objects.

Computer programming is an open-ended activity and so unique problems are likely to occur. Students who solve a problem can support the learning of other students. A few times, students in my workshops were able to explain their strategies to me when I couldn't understand what they had done—even the teacher becomes a student in a constructionist environment.

3.4.3 Authenticity

The gaming calculator hopes to teach the mathematics of decision-making through design and testing. The design task for the students is two-fold: game design and strategy design. If the pedagogical goal is to teach about strategy, and the gaming calculator supports learning strategy through design, why did I chose to make game design such a prominent feature? Why did I not just pick one game and limit my inquiry into how students design strategies for that game? There are two answers to this question.

My first answer relates to teaching decision-making: having students make different games creates different decision-making situations and provides more varied learning opportunities. Integrating game-design tools allows a learner to see how, for example, changing their game board's size or removing barriers (as described in Chapter 2), impacts how well the same strategy performs in a new context. Many students reigned in their game pieces' movements after seeing that they were too powerful; or if students' pieces were limited to only moving in one direction, they added the ability to retreat.

Moreover, when learners create their own games, they can directly control the complexity and thereby manage the pace of their own inquiry.

The second answer in favor of game design is that allowing children to create their own games and problems provides them with an authentic learning experience. The word authentic is an overloaded term so let me unpack some of its definitions (Shaffer and Resnick 1999). Within the educational research community, authentic refers to students engaging in the same activities as professionals, asking similar questions to those that professionals ask. A second definition is more in line with constructionism and involves students answering hard questions that they have posed themselves, questions that they have a personal interest in answering.

I argue that game design, when coupled with strategy design, enables authentic learning under both definitions of the term. Real professionals in computer science—and the natural, military, and other sciences—create game-like computer simulations and then experiment with their simulations to evaluate different outcomes. Also, computer games themselves are a billion-dollar industry (Entertainment Software Association 2005), and designing games is a profession that many children aspire to and emulated in my workshops.

Under the second definition, the gaming calculator allows students to experiment with solutions to problems that they have created: problems in the form of games. This authenticity motivated the students. They were compelled by the complexity of games generated from simple rules of their own design. Also, as I describe in the next section, being able to design a game about a fantasy world, and then see it come to life, was very motivating to some students.

3.5 Metaphors and Meaning

I chose to support the design of games with movable and modifiable pieces, in part to encourage metaphorical thinking. Consider these two chess pieces, one from Korean chess and the other from western chess. They are distant relatives descendant from some early proto-chess originating in South Asia millennia ago.

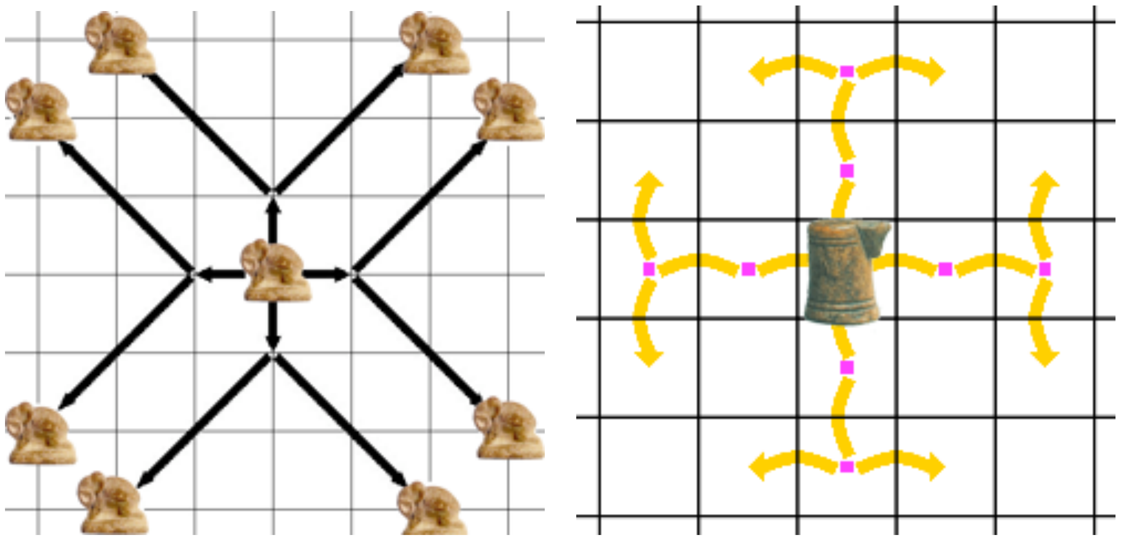


FIGURE 3.3 DIFFERENT CHESS PIECE MOVEMENTS

The change of graphics from an elephant to a horse is cosmetic but carries with it the metaphor of transport by animal, and thus holds meaning for the players. It also helps us to trace its shared lineage. The forking movements of both pieces extend the animal associations, even if only vaguely representative of the movements of real animals. Relative to the movements of the other chess pieces, they make sense: horses can leap over things; elephants can charge far ahead. That there is a metaphorical connection at all is suggestive to many people, especially to children. The shape, names, and movements of many game pieces are evocative and convey meaning.

A metaphorical connection between a moving game piece and some real world counterpart is a leap of abstraction. Not every moving piece game has metaphorical mappings; game tokens in backgammon and checkers, for example, have no obvious counterparts. Selecting between a top hat and a boot in Monopoly also doesn't mean much, although people have favorites and make personal associations with their representative token. When moving a representative game token, players' language can switch point-of-view from "I make this move" to "My piece moves here". So, although not every game piece is a metaphor, pieces that move can be imbued with volition.

Where pieces move is important to game play, and some games also convey meaning through their game boards; where a game is supposed to take place and what it represents can be important to understanding the game. While *Monopoly* only resembles Atlantic City with place names, the association is clearer when we buy hotels to place on the properties. We can assume *Candyland* is a sweet place, even if we've never been there, because spaces on the game board are decorated with lollipops. *Chutes and Ladders* set you back or advance you across the board. In Chinese chess, metaphors are also tied to movement rules: palaces restrict the movement of some pieces and a river prevents other pieces from crossing. In contrast, however, are western chess and checkers boards, which are just tiled grids.

My decision to study games with moving pieces was, in part, to leverage people's metaphorical associations and create personally meaningful games and thus meaningful strategic behavior. When asked if they wanted to design a game, many students were dumbstruck and overwhelmed until I suggested that they use a fictional story world of their choosing as source material. The toolkit was simple enough that within half an hour, students were able to create meaningful game environments that they felt were representative of rich fictional story worlds. Some students' games took place in the twisty corridors of the Matrix movies, the dungeons of Harry Potter's school, and in the mountains and fields of Tolkien's Middle-Earth. In each of these games, graphics and barriers were carefully placed to best represent these fictional landscapes. Choice of game pieces and their movements and captures were also carefully designed to represent their source material, often with interesting discussion about how to best translate a complex character's appearance into pixels and movement into vectors.

3.5.1 Piece Design by Example

In lieu of textual description of rules for how a computer should carry out of its instructions, programming-by-example researchers have developed graphical toolkits to build video games without the need to explicitly codify rules (Smith, Cypher et al. 1994). Instead, the user manipulates graphics on the screen, and the toolkit infers what the game's rules might be. For example, moving a train icon from a track icon to another track icon could create the rule to move trains onto available track icons, thus programming a rail-based locomotive. This line of programming-by-example research was furthered by inferring rules for board games (McDaniel 1999).

Since I limited the scope of games designable with the gaming calculator to chess-like games, in which players and machine already understand the rules of play, I was able to leverage that knowledge when designing the gaming calculator. This allowed for game designers to engage in some programming-by-example-like activities, such as defining the movements of game pieces by drawing their path with a mouse. By no means is this thesis a contribution to this field of research; I just mention the influence here as a tip of the hat.

3.5.2 Syntonic Pieces

The language of some students when learning about strategy with game pieces moved to an interesting new level: they spoke from the game pieces' point-of-view. For example, the language used to explain the minimax decision was ascribed to their game pieces' literally "looking around the corner" and the resultant behavior as "being cautious". The students' language is weak but intriguing evidence of learning transference: taking a mathematical concept and attributing it to behavior in another context, in this case a fictional story scenario.

This descriptive language occurred because strategizing with motile game-pieces facilitates *syntonic learning* of decision-making (Papert 1980). I use the term syntonic learning as defined by Papert to describe how the movement of one's own body can be used to reason about mathematics. The available movements of game pieces on the game board are clearly demarcated in such a way that any child who has played hopscotch can understand the options of his or her game piece in terms of his or her own body movements, thereby enabling body-syntonic knowledge. That is, children could embody their game pieces and put themselves on the board because the children and the game pieces share the same volition. But more interesting is how games empower ego-syntonic thinking. Because in the decision-making context of a game there are defined and explicit goals, the game piece and the student are moving towards those goals together. Students' reasoning about their options from the perspective of the pieces is a personal way to understand systematic problem solving.

While making an association between the moves of something real and a game token is a form of abstraction, ironically, the term "abstract strategy games" is used for games with weak real-world metaphors. Some abstract strategy games, e.g., go and line-and-dot games, make excellent candidates for teaching about planning because every game state shows the aggregated history of the game. In these games, a comparison of the current game state to any future game state clearly communicates what has changed. In contrast, such comparisons with moving-piece games are more difficult because of their transitive nature. In Chapter 5, I explore how abstract and other types of strategy games could be incorporated into a future version of the gaming calculator.

In the next chapter, I explain why I chose specific representations to model different parts of the decision-making process and evaluate how students responded to them in my workshops.

4 Design and Evaluation

In this chapter, I interleave design rationales for my software with formative evaluations of my workshops since the two were iterative. Over a six-month period, I conducted eleven workshops with twenty-five students and made many updates to the software based on problems students encountered. I began with an initial design and did multiple enactments and analyses (Design-Based Research Collective 2003) to get the gaming calculator and my workshops to work as well as described in Chapter 2.

I started with a two-week pre-test of the gaming calculator in an inner-city junior high school. The most important thing I learned from this experience is that not every teenager knows how to play strategy games or cares to. I spent a good deal of time explaining how game pieces move to students who did not know checkers, chess, or any board games. To help explain how these games work, I made a large tiled game board on the floor of their classroom with masking tape and played games in which the students were human game pieces. This was a fun and effective way to introduce these games to the students and created a few rousing discussions about strategy. The class also practiced playing games on paper game boards. Using the gaming calculator, we also designed our own game together as a class, which most every student enjoyed. However, when we started to strategize about how to win this game, the students who had just learned what a strategy game was had a difficult time with the first step in the strategic process, enumerating their moves. By then, our two weeks together were up.

The students who enjoyed this activity approached me later, after class, to find out more about the gaming calculator. This self-selected group of students was more disposed to learning about strategy than their peers. While these interested students worked on strategy designs after school, one of their reluctant friends complained to me, “why can’t we just play games without thinking so much?” While this activity wasn’t for him, students interested in strategy were good candidates for my workshops.

This chapter begins with a description of a diagnostic test I conducted at the beginning of most of my workshops. Next come sections that describe the modeled decision-making process designed into my software: identify your options (4.2), clarify your goals (4.3), evaluate your options (4.4), and then make your move (4.5). Each of these sections provides a rationale for my representation and an evaluation of how students understood what was represented. Then, in Section 4.6, I describe how students created hypotheses and tested them with the gaming calculator. I conclude this chapter with a summary of the problems my students had and how I overcame them.

Let me begin by explaining how the majority of my workshops were structured. Teenage students responded to flyers advertising “Do you like games? Learn the Secrets of Game A.I.” Flyers were posted at game shops, sent to chess, math and computer clubs, and handed out to interested neighbors. Based on response, 2–3 hour workshops were scheduled for teenagers at game shops, schools, and their homes.

Of the 25 students in my workshops, 7 responded to notices sent to computer clubs, 6 responded to flyers sent to school math and chess clubs, 3 responded to flyers posted at game shops, and 9 were colleagues' children, neighbors and neighbors' friends. 17 of the students were 13-14 years old, 5 were 15-16 years old, and 3 were 17-19 years old. Only 3 students were female.

4.1 Diagnostics

To ascertain what students knew about strategy and decision-making, I conducted informal pre-tests at the beginning of my workshops. First, I would ask how many moves could be made in the game we were playing. Then, I would show them a 1-ply game-tree graphic with numbers on it and ask which move they would make. I would repeat these questions with a 2-ply and 3-ply tree, asking them to explain their decision-making process. Every student was unfamiliar with these diagrams but nodded understanding after the representation was explained. Students could pick their best move on a 1-ply game tree, while 2- and 3-ply game trees presented more of a challenge. Many students would run their finger along the horizon states of the game tree and pick the branch with the largest spread of large numbers. This indicated that they knew larger numbers were better and represented what was good for them, but that they were not considering their opponents' choices.

I would then introduce them to the toolkit in the way outlined in this chapter. I would explain that the goal was not to just win games, but to learn the process by which a computer could win games. Many times in my workshops, I sensed that students "got it" based on their non-verbal language and familiarity with the gaming calculator's interface. So when it seemed to me that students understood the systematic decision-making process modeled by the gaming calculator and could read a game tree, I would give them the same diagnostic tests conducted at the beginning of the workshop. I wanted to know if they could describe the minimax process outside of the context of the game being played. It took a minimum of two tests before any of my participants were able to explain the minimax process satisfactorily.

Once they understand the concept, however, they were comfortable enough to apply it in other situations. When asked how they would design their agent to lose a game, they would think and then describe maximin. With a smile they'd say, "That would be a funny game." I would ask them if they could design that behavior in the gaming calculator and they would, and then proceed to happily lose games. I would conduct my test at the end of the workshop to confirm that they understand minimax.

4.1.1 Diagnostic Games

In my early workshops, we played a chess variant of my design, "baby barrier chess". These early workshops were freewheeling, as we figured out together what worked and what didn't, both technically and pedagogically. In my later workshops, "baby barrier chess" was shown as an example game and then students would use my design tools to create their own game that we would play for the rest of the workshop.

"Baby barrier chess" is played on a 5×5 game board, with symmetric barriers obstructing the movements of all pieces but the knights. The pieces and winning condition are the same as real chess: checkmate of your opponent's king. The game was specifically designed so that within two moves, you could easily put your opponent in check if they weren't planning ahead—a lesson my gaming calculator facilitated.

4.2 Choices

Identifying every good option in even a simple game—and in life—is not a trivial problem. When students were asked to identify their possible moves in whichever game they were playing, they had a difficult time enumerating all of them. Their estimates of the total number of possible moves were always considerably lower than the actual number. When asked to count their opponent's countermoves, their estimates were even worse. Moreover, the moves students identified were usually those that were obviously good; moves with subtle advantages were overlooked. When all moves seemed equal, the ability to count options degraded further: students would often respond to questioning about the number of moves with a shrug and an "I don't know."

Students were not able to think about this problem in an organized way. Only with prompting could they count the moves available to each of their pieces. At my suggestion, a few students did this by hand, a tedious, sloppy and imprecise exercise.

The gaming calculator was designed to make the identification of choices easier and more systematic. The primary element that accomplished this was an animated, interactive graphic. One way to illustrate possible moves could have been to sketch the moves directly onto a graphic of the game board, similar to how a football coach would sketch out a play. But this method quickly becomes overly complex with a tangle of lines representing possible moves, as seen in Martin Wattenberg's visualization of starting chess moves (see Figure 4.1):

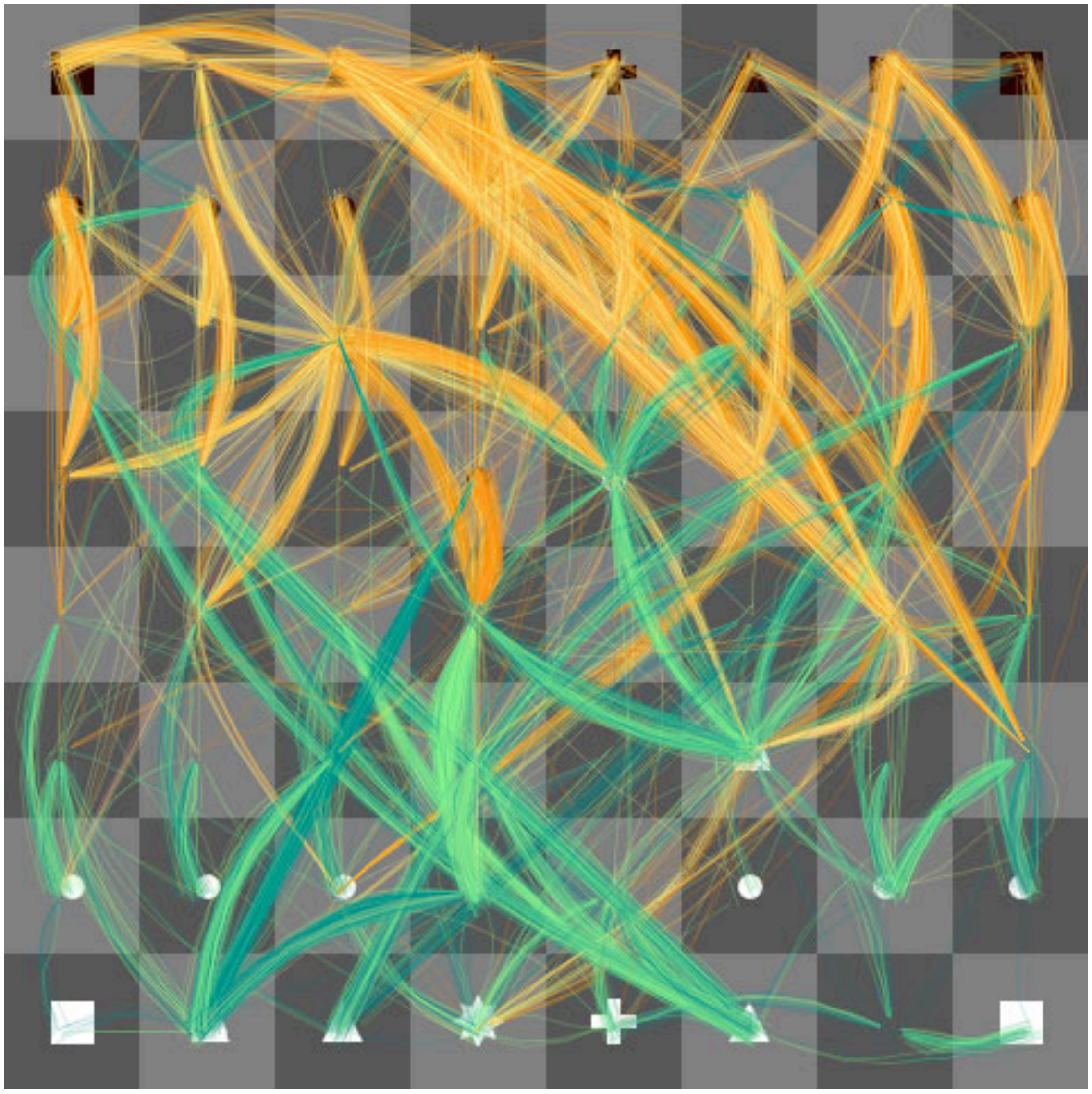


FIGURE 4.1 CHESS OPENINGS VISUALIZATION (WATTENBERG AND WALCZAK 2005)

Shown here are only the best moves; Wattenberg's visualization prunes away less attractive options. Instead of this approach, I use a common abstraction, a graphical representation of choices known as a game tree (See Figure 4.2). Each branch of the game tree represents a possible move in the game, and each junction represents a choice.

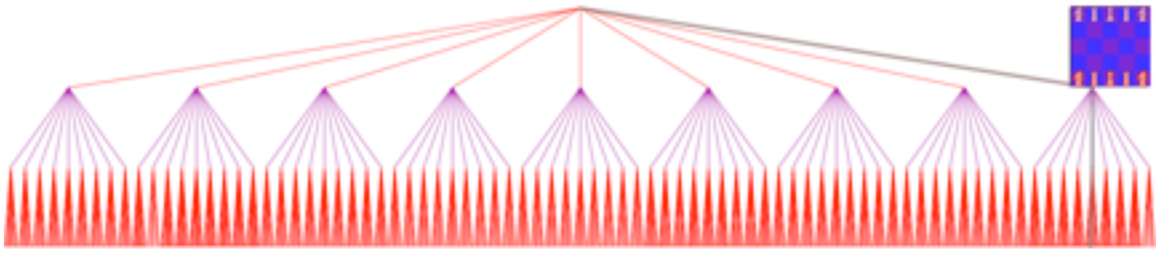


FIGURE 4.2 A GAME-TREE GRAPHIC OF THE FIRST THREE MOVES AND COUNTERMOVES OF BABY BARRIER CHESS

One goal of this abstraction is to reduce the cognitive load on the learner by clearly delineating choices. I had several other assumptions about why this representation offers advantages to a learner. In a multi-piece game, this representation focuses attention on all of a game player's choices, not just the choices of particular pieces. In addition, when looking more than one move ahead, the sequence of moves and countermoves is clear. A third advantage is that instead of a mesh of complexity, complexity is made manageable by fanning out all of the moves. Finally, a game tree allows students to more accurately estimate the number of moves by a glance at the density of the graph.

When I turned on the game tree function in my software, many kids were awe-struck. They were surprised at the sheer number of choices, and perhaps more importantly, they immediately identified the mistake of their previous estimates. When I showed the game tree to one of my students in a repeat diagnostic test, he said to me, "Ah, I know what this is about now." The number of choices became clear and important.

4.2.1 The Future Game Board

There are tradeoffs in using a game-tree representation. Foremost is the cognitive distance between the abstract game tree and the concrete game board. To help the students bridge this gap, I added a "future game board" in a separate window to function as a "crystal ball", showing students what possible moves on the game tree would look like. The future game board always corresponds to the branch currently highlighted in the game tree. Using the cursor keys, the student is able to select different branches on the tree to see possible game moves (the sheer number of moves in many games necessitated the use of the keyboard instead of the mouse when selecting individual moves).

At first, many students confused the graphic of the future game state with the current game state. This is understandable: the game board and the pieces are the same, just in a slightly different position. To remedy this problem, I designed it so that the actual game in progress was not visible when the future game board was shown. In retrospect, it might have been helpful if I had surrounded the future game board with a cloudy or dream-like border.

Just seeing a future game state does not explain how you would get to that state. Therefore, I used animation to show how game pieces would move from the current to the future game state. When the animation played out on the future game board as moving pieces, the corresponding branch on the game tree was animated simultaneously with a dotted line (the line was meant to resemble a treasure map trail; follow these steps to get to this destination).

4.2.2 Scaffolds to Debugging

I found in my early workshops that the simplest things could confuse students. For example, the language used to explain the process of moves and countermoves could get confusing. Referring to Player A and Player B was not sufficiently distinct to ensure that students knew who was who. Similarly, the word "opponent" became slippery because it had multiple meanings: they were designing a computer opponent to play on their behalf and also playing against a computer opponent.

As a fix, I gave each player a personal name so that moves were more clearly attributed. The programmable agents were named Huxley and Robotron (after some of my childhood computers). These monikers solved the problem of misassociation. I also used the names in dynamically generated explanations of the game-tree search process. For example, "Countermove by Huxley" would be displayed when an appropriate section of the game tree was highlighted.

The introduction of names had a secondary effect: students' language changed to indicate that they were programming an agent, a psychological explanation, instead of just designing rules, a mechanistic explanation (Resnick 1994). Huxley became a robot who "knew" and "could do" things. Their language made it clear that what were ascribing behavior to be implemented by an agent that followed their instructions. Debugging became fixing how Huxley would play the game (Suthers, Connelly et al. 2001).

In addition to naming conventions, the moves of each player were differentiated by color. When designing a games' rules, students selected the color for each player, and these colors were used to both tint the game pieces on the board and their branches on the game tree.

4.2.3 Bonsai Trees

In my first workshop, the representation of the game tree looked very different. I used an off-the-shelf widget for visualizing tree structures. When a branch was unfurled in this visualization, branches on the same ply were pushed down, often off the screen, to make space. Not being able to visualize all of the moves and countermoves at one time proved very confusing for these students. In the next design iteration, I implemented the graphic game tree (as depicted in Figure 4.2) and understanding improved.

With this improved game tree graphic, many things relevant to strategies became apparent to students. For example, one way an opponent can prune your game tree most dramatically is to capture pieces. Some of these captures generated unique "bonsai" game trees. I hypothesized that sparse branches would be easy to interpret as severely limited choices. I found that the students could interpret these trees, but only after some exploration with the gaming calculator. For example, two students were playing a game of their own design in which both players had only two pieces. When the tree first depicted a capture of either piece, the students were intrigued. They viewed those game states on the future game board to see what was happening. Once they saw their pieces fly off the board and correlated that with the highlighted branch, they could interpret the rest of the tree.

In addition to changing the game tree by the moves they made, students could affect the tree dramatically by the elements they chose when designing their game. The game-design toolkit was designed to allow variations that had impact on the game tree: e.g., to enlarge and to reduce the number of moves. For example, game-board barriers are a design element that limits a piece's motility. When you add a barrier to a board, you readily see that your choices are reduced. In designing their games, students added and removed barriers to see how the number of choices available changed. This was a quick process that led to many iterative design checks. Similarly, student designers would add a new move to one of their game pieces and then check to see how their choices expanded. The ability to "only move on capture" was added to create expansions of the game tree in specific situations.

When designing asymmetric games, students were often intrigued by game trees that showed a different number of choices available to each player. Students were able to explain that this interesting diagram represented the fact that one player had more choices than the other.

4.3 Goals and Hints

Games are different from many simulations in that they have defined goal states and reaching a goal state terminates the simulation. A winning condition is desirable pedagogically because it makes design decisions motivating and testable, ideas that I will discuss in Section 4.6. For now, I want to describe how the toolkit makes it possible for students to choose and describe goal states as well as evaluations of intermediary game states.

A common way to describe winning conditions in many strategy games is as a description of how pieces are related to one another on the game board, for example, three tic-tac-toe pieces in a row, or my piece can capture your king (check). To help students describe conditions in this way, the toolkit uses simplified natural-language syntax. While only allowing for a rudimentary expression of location, this approach reduces the burden on students by eliminating the need for programming, so they can focus their efforts on learning strategy.

Because game states can be defined as relationships between spaces, pieces, and other pieces, the problem of defining a goal state lends itself to a fill-in-the-blank sentence structure, such as "if piece _____, owned by player _____, located _____, is next to piece _____, owned by player _____, located _____, then you _____." The blanks are filled in using pull down menus to make choices obvious and to keep students on track. My choice of a natural-language syntax was informed by (Bruckman and Edwards 1999) and settled on after many design iterations. The interface prevented logic errors by precluding mutually exclusive choices, e.g., if you had selected that pieces have to be captured, then you are not presented with menus to describe where these pieces were located on the board. The same sentence interface is used to describe winning conditions and to describe evaluation functions, the latter including the assignment of number values.

Even with the scaffolding sentences, students sometimes needed support to translate their ideas for a pattern into the available language. For example, one girl wanted to describe that limiting her opponent's queen's moves was desirable, and I had to explain how negative numbers could describe that relationship. Using pattern language helped students understand the specificity that a machine requires without forcing them to speak the language of the machine.

Here is the design pattern template for sentences that describe game states:

> [# >= 0] [pieces] [of mine|of yours] [located on board|captured]
> ([relationship to other pieces] [# >= 0] [pieces] [of mine|of yours] [located on board]) * [pts] = score

In this notation, brackets indicate required fields and parentheses indicate an optional relationship clause. Written out this way, it is somewhat opaque, but the following examples illustrate that this syntax can define a variety of patterns:

> If 1 King of theirs is captured then I win.
> If 2 Harry Potter of mine are on the board here, then I get 2 pts for each one.

In game-theoretical jargon, +1 is used to describe a winning condition. I used the word "win" since this is the intuitive term for game playing (under the hood, a winning condition was defined as Integer.MAX_INTEGER). By moving the winning condition way down the number line, the range of numbers that students could use to score game states became much broader: students could use whole numbers rather than decimals.

Let me explain how my software finds patterns. First, the minimum number of parent nodes—each representing an active game piece—has to be found. This means, to use the latter example sentence above, that at least 2 Harry Potters have to be found in the specified spaces on the board. Next, if a relationship clause is used, e.g., can move or can capture, then every actor piece is checked to see if it satisfies the conditions of that relationship, e.g., can move to these specific spaces or can capture this number of specific pieces. If the sentence pattern is defining an evaluation, then points are allocated for every matching child node. If a relationship is not specified, points are allocated for every matching parent node.

To make the computer's evaluation process transparent, the gaming calculator shows a graphical overlay of evaluation criteria on top of the game board. This helped students to understand the relationship between their pattern sentence and real game boards. In my pre-test, we spent a lot of time exploring how a number score could be assigned to a game board and we made repeated reference to the overlaid evaluation information to understand how this process worked. We projected the information on a white board and the students took turns marking up and counting the pieces on the board themselves to score a board according to their rules; then I would turn on the overlay to confirm that they were right. In Figure 4.3, a student had just marked up the 4 spaces where the two T-Rex pieces in the lower left could move.

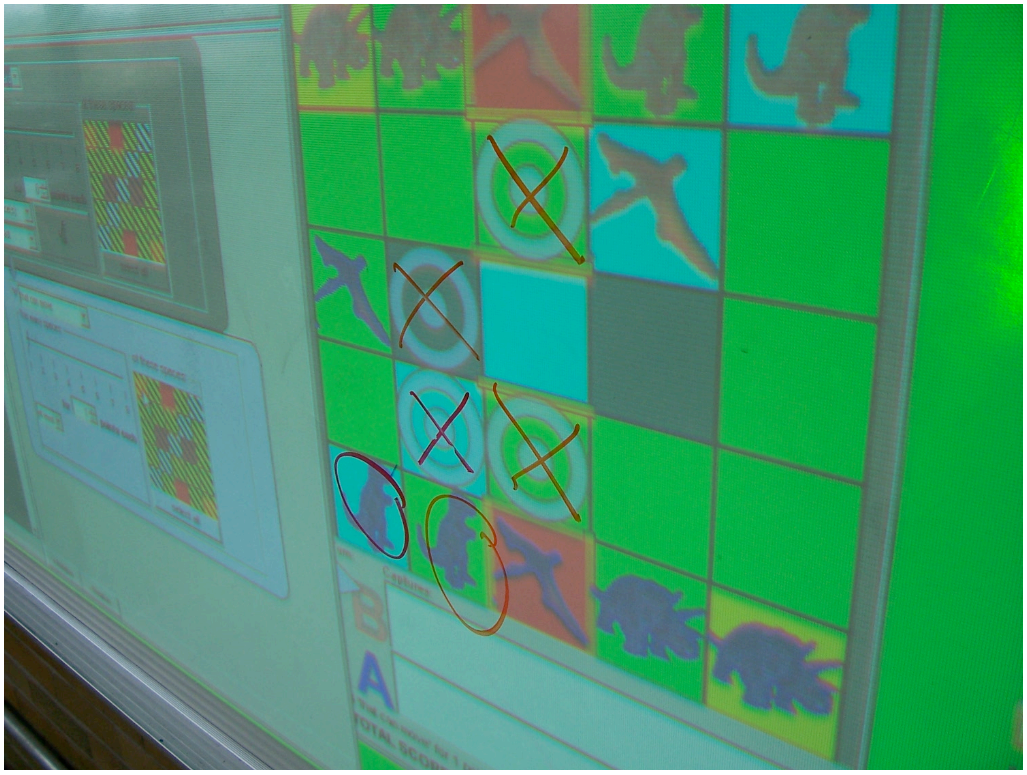


FIGURE 4.3 A GRAPHICAL OVERLAY OF EVALUATION CRITERIA PROJECTED ONTO A WHITEBOARD

Because game-state hints might describe multiple pieces and various relationships, an overlaid search result must be able to convey a lot of information. In my design, if child-node spaces were identified by a hint, these spaces were highlighted on the game board; mousing over these spaces would reveal the locations of their parent-node pieces. For example, in the game state depicted in Figure 4.3, mousing into one of the target graphics would highlight the T-Rex piece that could move there. Contextual graphics would represent the type of relationship: explosion graphics showed pieces that could be captured and targets showed spaces where pieces could move.

4.3.1 Collaborative Competition

My design goal was to create a collaborative learning environment in which evaluation criteria were appropriable. It might seem contradictory that in competitive workshops in which students wanted to beat each other in their games, students would share and discuss strategies. However, there was open discussion of what worked to win games; the process became a collaborative inquiry.

One of my design goals was to make hints easy enough to understand so that other students could read and modify each other's work. This led me to add the ability for students to "view source" of each other's hints to reveal how they worked (Blankinship, Smith et al. 2004). I also designed my workshops to facilitate the exchange of code and ideas. Before students tested their strategy against the default computer opponent, I asked them to present their strategy's hints to the other students. I would encourage them to read their hints aloud. Then we would see how their hints performed in the thousand trials; other students took careful note of which strategies performed well. The student whose strategy performed best received extra attention from the other students, who asked for details about how the strategy worked; they peaked at the hints after the trials.

The gaming calculator supports the design of games with multiple winning conditions. This means that each player can have many and differing goal states. For example, you can create a game that is winnable by either capturing a certain piece or moving to a certain space. I hypothesized that with more than two ways to win, students could experiment with multiple strategies that might require thinking more broadly about their evaluation criteria. Only one of the games designed in my workshops had multiple winning conditions for both players: capture all of your opponent's dinosaurs or get to the other side of the game board. However, once the game rules were shaken out and relatively balanced, it turned out to be nearly impossible to get to the other side of the game board without being captured first. Students' strategies focused almost exclusively on capturing other pieces.

When students created games with asymmetric winning conditions, it had the interesting effect of helping to clarify the minimax algorithm. When playing games with symmetric goals, students would be distracted by trying to evaluate how good the other player's moves were from the other player's perspective (which is not how minimax functions; this is teased apart in the next section). When playing games with asymmetric goals, students only focused on evaluating how good *their* moves were, and it was therefore easier to understand that their opponent would prevent them from getting to their best moves. An example of this is in the Harry Potter story from Chapter 2.

4.4 Systemic Evaluation of Choices

I would begin many workshops by asking students how they played strategy games such as chess and checkers. Most students answered that they played randomly until they found a killing move or a memorized tactic such as a pin or fork. Some would explain that they played defensively, which when asked to explain, meant they tried to keep from losing pieces.

The lack of a sophisticated strategy could be ascribed to the earlier finding that most students were not aware of the number of choices available. But even when they saw all of their choices on the game tree, they still had a difficult time identifying the relative merits of each move. I displayed numerical scores on the horizon nodes of the game tree to make differences clear, but without an evaluation function, every branch has the score of 0.

I introduced the concept of heuristics with an analogy to the familiar search game "you're getting warmer/you're getting colder". Like the player of this game, a computer opponent needs guidance to know which moves bring it closer to the goal state, "red hot". I explained how we could play this game by calling out numbers instead of qualitative temperatures ("you're burning up" can become 500), and that is how we had to program Huxley.

By this point in the workshop, students had already created their winning conditions and were familiar with the pattern language and sentence interface used to describe game states. After I prepared the ground with the "hot/cold" analogy and introducing game states on the tree, students, with little prompting, knew how to create a hint, and they would apply it to see the effect. Commonly, their hint would be to count the number of their pieces on the game board for 1 point each. When only searching ahead one move, this often did little to differentiate the moves. But when we would change the look-ahead function to 2 or 3 moves, differences became obvious. Inevitably, depending on what they did, they could lose pieces.

Sometimes, to make the point that moves that look similar can be distinguished from each other, even without looking far ahead, I would create a hint that gave 1 point for every space a piece could move to. This hint created a 1-ply game tree with lowered scores on the ends: a center-heavy number line. When asked if they could explain this number line, students would pause and then describe that pieces along the borders of the game board could not move to as many spaces and therefore had lower values. They were able to interpret how the numbers reflected real game conditions.

I also explained that we could create multiple hints so that the computer could evaluate game states various ways, and that these hints would be summed together. The visual representation of these multiple hints was as a simple tabulation with a "TOTAL" at the bottom. Double-clicking any of the line items would open the hint in an editor. The design of the toolkit synchronized the display of the selected evaluation in the tabulation, the overlay on the future game state, and the corresponding selected tree branch. The same evaluation number was also displayed on the tree, the tabulation, and the future game state to clarify how the same score was represented in various parts of the decision-making process.

Many students would create multi-hint strategies, and with almost every new hint, the number lines on their game trees would update. In the example from Chapter 2, the student designing the Harry Potter game increased the score for his self-preservation hint to a larger number, and this made moves that kept Harry on the board stand out numerically on the game tree. This method of dramatically increasing the value of a hint to see where it is reflected on the game tree can be used as a search tool: e.g., where are those few moves where Harry can capture but not be captured?

When developing strategies for chess variants, one student used the game tree in an interesting way. She would enter a new hint and then see how the number-line would change, browse through highly scored states, and tweak one of those hints even further so as to better describe what was already a good move. In general, the students who created hints for chess variants had much more varied point scales than students who made their own games. For example, they would assign a rank value for each of the different chess pieces, and then use this scale as a basis for other valuations. This scale, used by different students in different workshops, seemed to be straight out of chess-strategy guidebooks.

Many students recognized the need for a scale that ranked hints according to their importance, but were unsure of what the upper bounds of this scale should be. I explained that winning was an infinitely high number and their scale was up to them. Some students wondered if just putting in a larger number would result in better performance, but as one student told another, “really big numbers just make you feel good about yourself, they don’t make you play better”. He then explained that what was important was how much larger one number was to another, relative scale.

One common problem students had in understanding scale related to capturing pieces. Students would often create a hint that said it was worth the default 1 point to be able to capture another piece. But when they played, their agent would rarely capture anything. I would explain that they needed to specify capturing as good, not just being able to capture. In other words, their hint said that it was good to be in striking range and nothing about the actual strike. Then they would add a hint specifying the need to capture other pieces, but they would leave the value at the default of 1 point. This only brought about slightly better behavior than before. I would then explain that the computer now valued equally being able to strike and the actual striking. Students would quickly increase the value of striking to create the desired behavior. In creating these two rules, the students had made a better capturer then they would have made with one rule, and they understood this.

4.4.1 Complexity from Simplicity

The toolkit gave students control over the level of complexity in their games, thus helping them see how different rules affected issues of balance and how simpler rules sometimes made strategy more relevant. There was only one game design in my workshops that was simple enough for a student to identify a guaranteed winning strategy in two moves. This circumvented the need for all of my software’s visualization tools.

This game was simple to win because of an overly powerful piece: a dinosaur that could run all the way across the board, eating as it went. What is most interesting about the student who was able to identify the winning strategy is that when we slightly decreased the power of this piece, and thereby increased the complexity of the game, he was able to determine a winning strategy in five moves within a few minutes—again, without a game tree. An avid chess player, he commented that this game needed to be more like chess, with simpler rules, so as to create strategic scenarios that he could not solve in his head.

4.5 Choosing

After enumerating all of the moves and evaluating them, it is finally time to make a move! Many students were excited that we were actually going to play the game, not just think a lot about it; little did they know that picking a move was probably the most difficult conceptual problem they would encounter in the game. Even at this stage of the workshop, most students were still highly motivated because they wanted to create a machine that would win.

If only looking 1-ply ahead, every student found it easy to pick the best move on the game tree. This was confirmed both in my diagnostic tests and when students were using the gaming calculator. Programming Huxley to select that best score was also straightforward to most students given the interface I created for describing this behavior. Students had to complete this sentence: On my move, I will pick [my best move] [my worst move] [a random move].

Clicking “best move” would then highlight the branch on the game tree with the highest number score. If multiple branches were scored the same, then one of those branches would be selected at random. I would explain that “best” meant the largest number value and remind them about the hot/cold game analogy.

Next, we asked our agent to look two turns ahead, and the game tree would fill out with our opponent’s available countermoves to every one of our moves. When asked which of these countermoves our opponent was most likely to select on his turn, students would pause. They had a similar sentence to complete to describe this process: “When it is Robotron’s turn, I think he will pick [my best move] [my worst move] [a random move].”

After ruminating about this, nearly every student selected maximax, “my best move”! How they came to understand what their selection meant is interesting. When asked why they made that choice, they would often explain that they assumed the opponent, Robotron, would pick *his* best move on his turn. They were reading the sentence as if the opponent spoke it. The explanation students gave when playing symmetric games was that their opponent would pick its best move on its turn, just as they picked their best move on their turn.

I would re-read the command sentence to them aloud with an emphasis, “When it is Robotron’s turn *I think* he will pick *my* best move”. Often times the student would look at me for confirmation—that’s right, right? It was clear that ambiguity of the language interfered with the students’ ability to understand the strategic concept.

I would next remind them that all of the scores on the game tree represented their agent’s moves and how good they were to their agent. “If Robotron makes one of these countermoves, this number represents how good that countermove is for you.” This explanation just led them to look at me plaintively. Some students would then select “worst score” or “random move”, but when asked why, they just said they were pretty sure “best move” wasn’t right.

I did not want to give them the answer but wanted to help them figure it out on their own, so I used the game tree to scaffold this investigation. This is how I did it: I would pick one of the hints they had already created, and edit it to increase its value so that when the game tree was reloaded with new scores, there was sure to be large relief between many choices on their horizon game states. I felt it was important to keep the student’s hints, but just make one stand out a little more for this exercise.

Then, I would switch the game tree’s look-ahead function to 3 moves, extending the reach of the game tree and then selected maximax. Next, I would find and select one of the highlighted higher scored game states on the tree, which brought up the corresponding future game board. We would discuss why this game state was scored as high as it was, making it clear that we both understood why it was considered a desirable future scenario. It was agreed: it would be great if, in two moves, this is the situation we would find ourselves in.

Then, I would animate the sequence of steps that would lead to this game state. I would ask if what we just saw was a plausible sequence of events: would any decent computer opponent make the move that made such a great move possible for us? For example, if Robotron has a chance to capture one of our pieces on his turn, would he reasonably not make that capture? Would he just step out of the way? Especially when not making that capture means Robotron's pieces could be captured when it is our turn?

This series of questions was asked while mousing over the region of the game tree that corresponded with the move being discussed. I designed the game tree animation to be responsive to the location of the mouse to help facilitate this sort of conversation. When I placed the mouse into the lower branches, representing the moves three turns ahead, all of the scores were labeled on those branches. Moving the mouse up to a branch above, to a previous move, displayed the selected score from the bottom branches (max, min, random). In this way, I was visually representing how the searches for scores were being propagated backwards. I would explain this process as "sentries who have scouted ahead sending back their report from the horizon".

In addition to this responsive display of information, I also used an animation of the minimax decision, in which the game-state scores were animated climbing up the branches. When students first saw this animation, they were often wowed that so much information was moving on the screen at one time. I would focus their attention on the game branch under discussion by pointing at that region of the animation and replaying it. The animation was designed to show the minimax decision occurring on all branches simultaneously so as to impress upon students the process in which a systemic decision was being made: every computed move and countermove was being considered. I inferred that students understood this process as I explained it to them, but some students later told me that it was not until the end of the lesson that they understood what was being represented by this busy animation.

While having this conversation, I would toggle between the preferable maximum game state on the horizon and a neighboring lower game state. This would show the outcome of the selected move on the future game board. The lower game state was not as desirable an outcome, but, the student would agree, was more likely to be the move available to us after our opponent moved.

This visualization and discussion led to many students having an "aha!" moment, and they would explain to me, "So, let me see if I understand this… Robotron's best move is my worst move. What is best for him is what is worst for me." Many students would mull this idea around in their mind for a while before agreeing on it. At this point, the student would usually change the sentence for programming their computer opponent to read, "I think Robotron will pick my worst move".

By the logic of minimax, evaluations of what is a good move are always made from the perspective of the person playing, and an opponent's goals are completely dependent on your own. Only evaluating the state of the game from your perspective is somewhat counter-intuitive; most of my students thought they should evaluate their opponent's moves independently of their own. Being aware of the minimax decision-making strategy could be helpful in areas beyond games.

4.6 Hypothesizing with the Game Calculator

In many workshops, students learned to design their own strategy by playing against a default computer opponent. Before playing, students would watch me create or load a default strategy into the *Player B Strategy* tab as I told them that Player B's behavior could be changed later. After playing a few games, I would let students guess the rules of this default strategy. Without even knowing how to explicitly model a strategy of their own, many students were able to deduce their opponent's simple rules for behavior: it looked one move head and only wanted to capture pieces. Once they understood how their opponent worked and how to use the gaming calculator to model their own decision-making process (as described in the previous sections), it did not take long before they suggested new competitions.

In this section, I describe how students posed their own problems, hypothesized about outcomes, tested their solution in trials, analyzed results, and then began this process again with new problems. How this line of inquiry played out in my workshops is described in two anecdotes: one in which students played a chess variant and another in which a group of students played an asymmetric game of their own design.

This section presents evidence to support the claim *that students designed and tested computer game strategies in ways that resemble the scientific methodology of generation and testing of hypotheses*. I conclude this section with an explanation of why my gaming calculator was conducive to this inquiry.

4.6.1 Me Versus Me

Two boys, each aged 13, were playing "baby barrier chess", on their own laptops. We were at a gaming shop that specializes in the sales of fantasy role-playing games and customizable card games, and a third of the store was filled with large folding tables set up for gamers; we worked back there. The two boys had responded to advertisements posted at the store for my "Learn the Secrets of Game A.I." workshop.

Once they understood how the toolkit worked, the boys were ready for some new challenges. One of the boys had an idea—an idea that excited both boys: could they run the computer opponents they had just designed against each other? I said that that sounded like a good idea to me; we just needed to get a disk to copy one of their strategy files to the other laptop. But no, I had misunderstood; they wanted to see how their computer opponents faired against themselves (but they also liked the idea of a competition as well).

I was intrigued that they had proposed their own problem. As I guided them through how to save their Player A strategy to disk, and then load it into the *Player B Strategy* tab, I asked them how they had come up with this idea. They replied that it would just be cool to see what would happen.

I wasn't sure of the outcome myself, so I asked the students what they expected to happen. What were their hypotheses about the outcomes of these mirror matches? The boys, while hyped to see the matches unfold, paused for a moment and thought about this. The boy who had proposed the idea said, "…well, they should be equal, I mean, both should win the same amount." And when asked why, he answered, "Because they both have the same strategy. They will both do the same thing." The other student agreed that this made sense. I repeated their hypotheses aloud, to make sure everyone understood their expectations and their rationale.

We started the Thousand Trials on both laptops, and the results started to stream in (at this early workshop, I had not yet implemented the pie graph, so results appeared line by line in a debug console). The boys were taken aback by the results: Player B was winning almost all of the games on both of their laptops! We confirmed that both Player A and Player B were using the same strategy file. They thought this was particularly interesting but did not say much beyond exclamations such as, "What is going on?"

I prompted them, "Is this what you thought would happen?"

They replied, "…no…"

"What do you think is happening? What's going on that could make these results?"

"I don't know… Player A always goes first, so maybe the player who moves first in this game can get caught easier…"

The other boy continued, "Yeah, they put themselves out into the battlefield first so they get hit first and… it's all downhill from there."

His friend jumped in, "Chess might be the same way… too big to test it though."

I said, "That's an interesting theory. Would you like to change something and try it again?"

The boys thought for awhile, and then one said, "Well, even if we change the strategy, it will be the same: they will each play the same and whoever goes first will still lose. And we've already done that since we each did a different strategy on these two laptops." The boys nodded their heads at each other; they had confirmed their findings in each other's independent experiments.

Let me recap what had happened here: the students proposed a problem of their own to be solved, a problem they did not know the answer to. When prompted, they were able to hypothesize about an outcome. They created the experimental conditions themselves using the gaming calculator and then ran their own experiments in a thousand trial runs. The results from these trials disproved their hypothesis, and when prompted, they theorized as to why.

While these students instigated their own problem, I cannot underestimate the role I played in structuring these students' investigation. I helped to keep the students' line of inquiry alive by asking probing questions and suggestions. In short, I played the role of a teacher; without my intervention their line of questioning and reasoning might have been derailed.

4.6.2 Asymmetric Star Wars

Another example of experimentation with the toolkit began when two boys from a school chess and math club responded to mailed flyers advertising my gaming workshops. When they arrived, the boys settled on making an asymmetric strategy game themed around the popular Star Wars movies. We only had one laptop, so the students worked together on the design of their game board and pieces. Their game's winning conditions were to capture every game piece of their opponents—they had devised a death match.

I was not sure that their game was well balanced; the two designers, however, were pretty sure that they had created a balanced game because they had thought carefully about every design decision as they made it. To test their game, I suggested that we run a control study first: just random moves versus random moves to see how things shook out. The students thought this was a good idea too, and we launched the thousand trials to discover that Player A and Player B fared equally well. I was impressed. From my own experience, developing balanced asymmetric games was difficult, let alone on a first try. The thousand trials allowed them to prove the balance of the game, which served as a control for their later experimentation.

As per the format of my workshops, I loaded in the default Player B strategy and the students proceeded to learn about the decision-making process through explorations of their choices with the game tree. Working together, these two students created a 5-hint strategy. By the time they understood how the toolkit worked and their strategy was able to defeat the default computer opponent they were not sure which of their hints was responsible for this outcome. I asked them what they thought was responsible for their strategy's efficacy, and they said they were not sure and that was why they wanted to figure it out. They proposed a series of tests to determine what was actually effective in their strategy.

The students began a systematic process of neutralizing all of their hints but one; assigning the hints point values of 0 and then running the thousand trials to see the outcome of these neutralizations. Their "isolate-and-iterate" approach did not find the one killer evaluation responsible for winning the game, although one hint did do much better than the rest. When asked to explain, they said that it was a combination of this hint with other hints that was responsible for the peak performance.

To summarize what these students did in the context of problem solving: in an exploratory investigation, they were able to achieve a goal state but were not able to explain what was responsible for their success. To determine which factor was responsible for their success, they conducted a systematic series of tests in which they neutralized variables. While the results of these tests were inconclusive, they did identify a strong factor.

The approach of these students to testing their strategies was not unique: in three different workshops, students conducted similar isolate-and-iterate tests on their strategies. In each of these cases, students instigated these investigations on their own, and in two of these cases they continued to work on the problem after the workshop had ended. The compulsion to understand what was responsible for their winning condition was motivating.

4.6.3 Who's Going to Win?

Students can pose and test hypotheses in many domains, but modifiable strategy games are especially conducive to inquiry because they are fun, because they have elements that can be isolated and changed, and, because of their terminal states, they are easily repeatable. In short, the interesting microworld of a strategy game provides opportunities for iterative testing. Chess-like games, in particular, are uniquely testable in that they have a binary outcome: you win or you lose (or, in some cases, draw).

Since the gaming calculator's simulated microworlds have definable terminal states, they can be easily batch tested using the thousand trials feature. This feature is particularly helpful in hypothesis testing in that experiments can be easily repeated and the results are summarized in an intuitive pie chart. Game series are commonplace in chess tournaments and sporting events for this reason: the game player with the better strategy will shake out in the end despite flukes and lucky breaks. Also, because the toolkit allows games to be easily modified one element at a time, students are naturally driven to experiment with cause and effect. Pull-down menus also encourage experimentation by making choices obvious.

Games are interesting to teenagers because they are microworlds in which they have direct and understandable control over behavior. This is different from most science experiments available to young students. In many science-fair experiments, for example, students set up seed conditions to see how things will grow, then stand back and observe and measure results. This is most literally the case in potted plant triptych presentations. In contrast, experimentation in games is directly controlled by the choices of the game player, and this volition can be described in a straightforward way with the gaming calculator.

When students use my gaming calculator, the creative focus is on description of desirable game states, not on programming control structures. This focus on algorithm modification instead of algorithm implementation has the benefit of focusing students on the debugging of behavior instead of syntax errors. In other words, debugging in the gaming calculator is not about trying to make autonomous behavior work since that is a given; it is about making autonomous behavior work well, which is testable by counting wins.

4.7 Summary

Using the gaming calculator, students learned about a decision-making process in which systemic evaluation determined which of their choices was best. Students had many issues with learning this process, and the design of my software helped to scaffold their understanding. Students' attention was held throughout the learning process because of their compulsion to design winning strategies for their games.

It was difficult for students to predict the number of moves and countermoves in familiar games. To scaffold their understanding of this number and what it represented, my software produced game tree graphics that fanned out all of their choices. Students were able to understand this representation and also understand how their design decisions modified this representation.

It was difficult for students to differentiate their choices systematically. To scaffold learning this process, I employed many strategies. Foremost, I focused students on the design of descriptive evaluations of good game states since this step in the decision-making process encourages generative and creative solutions. Natural-language syntax was found to help students be the most expressive in their evaluations, and graphical overlays of their criteria over actual game boards was helpful to students understanding how their evaluations were used to generate number scores of different choices.

How to plan ahead in a competitive game, specifically how to describe the minimax decision, was very difficult for students to understand. To scaffold learning this process, I provided students with ways to modify the minimax algorithm to encourage experimentation. I found that animating the effect of students' modifications to this process helped them to understand it better. I also found that without repeated instruction students could not describe or apply this algorithm in other situations; despite many software scaffolds, instruction remained of utmost importance. I also found that instruction could come from a collaborative environment of peers around strategy games, despite the competitive nature of game playing.

I found that students, often of their own volition, would generate and test hypotheses around strategy games. Since their hypotheses were testable by winning, once their test was complete (a game or series of games was over), students would modify their hypotheses and re-test in an iterative fashion. As part of this iterative hypotheses testing cycle, I found that some students would isolate variables as a meta-strategy for understanding what made more effective strategies in their games. The gaming calculator was found to be successful in scaffolding this inquiry by enabling modification to the key steps in the decision-making process.

5 Future Work

This dissertation has described how teenagers were able to design computer games and computer opponents to play their games by modifying game-theoretical algorithms. This is a promising finding for educators who are looking for ways to get students interested in mathematics. Two of my study participants quickly realized that they were, in effect, in a specialized mathematics workshop. They confided with smiles, “This is really math class, isn’t it? We won’t tell.” Playing around with mathematics (literally!) in the familiar context of games gave numbers a practical application: winning.

This thesis’ contributions are a new computational tool, the gaming calculator, and observations of how this tool helps students to learn game-theoretical concepts. A specific result is that students were indeed able to construct solutions to complex decision-making problems through iterative hypothesis testing.

Another contribution of this thesis is supporting evidence of the more general idea that providing design scaffolding based upon high-level modification is more productive than low-level syntactic manipulations. This closing chapter explores how the gaming calculator could be extended technically to further support this general idea. One of my goals was to make the gaming calculator simple enough for non-programmers to learn quickly but expressive enough for sophisticated exploration. I believe that this goal was met, but there are many ways a next-generation version could help learners explore more strategic concepts without confusing the novice. Future versions of the gaming calculator will be far more expressive, allowing for the design of more interesting games and, as follows, more interesting strategies to win those games. Section 5.1 describes enhancements to the gaming calculator for games of perfect information (the type of games this thesis explored) and 5.2 describes how other strategy games could be supported. Section 5.3 describes how computer simulations could be modified and 5.4 looks at frenetic video games. These explorations employ data-driven artificial-intelligence modifications to focus learners’ attention on behavior, not implementation.

That many of my study participants continued to work independently at the end of workshops or asked for a copy of the software suggests that brief, intense workshops could be extended to allow more time for learners to investigate. If my comparison of the gaming calculator to a graphing calculator holds, then it is helpful to look at how graphing calculators’ efficacy in classrooms is measured. Studies of students’ performance with and without graphing calculators show that students with calculators had better comprehension in applied contexts and engaged in more problem solving activities (Burrille 2002). Technological fluency and access to a powerful exploratory tool allow these students to focus on asking and solving interesting problems, rather than working on rote calculations for every *f(x)*.

It is a fun and improbable future in which secondary school students are tested on how well they can play games as part of their formal mathematics assessment. However, integration of a gaming calculator into an ongoing educational program, such as an after-school club, an undergraduate economics course, or a gaming club, could provide many rich and varied learning opportunities. This dissertation concludes in Section 5.5 by describing how this work could be used in more educational settings.

5.1 Games of Perfect Information

Nearly every teenager in my studies, when introduced to the game-design capabilities of the toolkit, asked if there were more sophisticated game-design options. Could game pieces have hit points? Could game pieces hold different weapons? Could game pieces' capabilities change over the course of a game (e.g., grow weaker as they were hit more often?). This expectation was understandable. As games have grown in sophistication, so have kids' expectations for game design. *Dungeons and Dragons* (D&D), a paper-and-dice game with its origins in chess, is designed to make players into game designers, with its varied rulebooks providing nearly unlimited combinations of resources to customize game rules. Even if teenagers today have not played D&D, they are no doubt familiar with its influence, as D&D has served at the thematic and rule template for many computer games over the last thirty years.

Once teenagers in my study designed a game with the available feature set and began work on their strategies, they became aware of just how complex their "simple" games were. While their games provided enough of a strategic challenge for our workshops, there is no reason not to allow for the design of more complicated games. The tool should support learners posing and solving problems as difficult as they want to explore.

Even within the limited constraints of chess-like capture games, new rules could be made available for game designers, as the teens' questions suggested. For example, landing on different game board spaces could trigger interesting behaviors: addition and removal of barriers, changing the size of the game board, or maybe changing, teleporting, removing or adding game pieces. Another option would allow for certain pieces to have more power than other pieces, such as only allowing pawns to capture kings, or limiting the movements of certain pieces to regions of the game board. Many games in which a roll of the dice determines how many spaces you can move could also be implemented, since every outcome of a dice roll could be planned for. By no means are chess-like games the only type of game in which moves and countermoves can be graphed on a tree. Line-and-dot games are a popular game genre, as is go and its variants.

For every change to the rules of a game, strategies will have to be more expressive as well. In the proposed changes to chess-like games, if the game board changes while playing, strategies that refer to the game board will have to be described with a more flexible language than is used to describe static, absolute positions (as they are referred to in the current implementation). Every new game genre brings with it different ways to evaluate progress towards winning. The current implementation of modifiable sentences could be extended and dynamically configured to describe game states for a wide variety of games.

While every new game rule offers exciting new ways to play, each additional choice available to players makes for an explosive number of possible moves and countermoves, which poses memory problems for both the computer and, more importantly, for the learner. However, once the basic algorithm mechanics are understood, then opening a wider range of possibilities provides more opportunities for authentic applications of their knowledge. Ensuring a gentle introduction to these ideas with a simple game should remain a pedagogical goal, despite the insistence by many kids that they are already game experts.

5.1.1 Searching Quickly, Widely, Deeply, and Selectively

The implemented gaming calculator just scratches the surface of the many ways A.I. researchers have tackled game-search problems. Looking ahead exactly one, two, three or more turns ahead is a rather artificial (and not intelligent) way to plan in a game. For example, in the current implementation, there is no alpha-beta pruning, no quiescence, and no memory resource management. Many of these issues are, necessarily, complex and tied to low-level resource allocation problems of computer science.

To help introduce the problems inherent in any practical application of game searching, such as time and memory resources, allocation of these resources could be imposed on strategy designers (e.g., "Hey, let's play *Clue* with a 40,000-move search limit" or “Let’s set our tic-tac-toe strategies to a 10-second search”). This feature would help students to understand what makes processor speed relevant. Limiting designers to timed searches seems to be an unfair limitation, however, since the designer with the newest and fastest computer would always have an advantage.

If one is only evaluating a number of game states, then finding a way to describe search order becomes an interesting problem. A depth-first search doesn’t make much sense since designers might exceed the allocated number of searches. Breadth-first search works better in this scheme, since searches can “spill over” to the next ply. Designed and implemented in an earlier version of my software was a feature called "selective deepening". Instead of searching exactly 1, 2 or 3 ply ahead, selective deepening allowed strategy designers to break from a breadth-first search if an interesting game state was found and they wanted to explore its next moves in more depth (e.g., finding check in chess might be worth looking into). Selective deepening was disabled during user testing when early studies made it clear that my subjects were finding sufficiently interesting challenges with a straight 1, 2 or 3 ply search.

It is difficult to imagine how alpha-beta pruning could be described in a way that makes it modifiable. Nonetheless, animating its effects on state search would probably be informative. Regardless of understanding its function, once strategists saw its effect on their strategy’s performance, they would turn on this feature.

5.1.2 If-Then Rules

A straightforward way to program a computer opponent is to provide a list of if-then rules. For example, if you find a way to capture the queen in chess, don’t waste resources searching countermoves; just get the queen! This type of strategy rule is more accurately called “if and if, then”, since both an initial state and the desired next state have to be satisfied. To integrate this way of picking best moves with a game-tree search, the “if-then" state is assigned a utility function. When making the minimax decision, "if-then" scores are treated just like evaluation scores.

This feature was implemented in the first release of my software and used by some students. It turned out to be problematic. The students entered very concrete if-then rules all focused on the first few moves of the game, almost leading the computer opponent by the nose. Their if-then rules were very specific to game states, describing the position of nearly every game piece on the board. The likelihood of their game states ever being repeated was miniscule, given that some of their game's pieces could not retreat. After these first few moves, their computer opponent didn't have any heuristics regarding which moves were better than others, and the students were overwhelmed by the prospect of entering many, many more if-then rules to guide their computer opponent through an entire game.

These students' if-then rules retarded their progress towards creating even a simple evaluation function. Once they were weaned off of if-then rules and encouraged to work on evaluation functions to generally guide their strategy, they would express dismay if their if-then rules didn't fire (because the minimax decision had found a higher scored move a few turns ahead). It seemed students were distracted from making broad strategies.

5.1.3 Beyond Min and Max

In the current software, strategies can be set to select maximum, minimum, and random scores to focus learners on the most obvious choices. In earlier versions of the software, it was possible to select from a variety of number-line relationships including the median, average, and mean. Also, the ability to define an offset was enabled, so that students could select, for example, two less than the highest value. This was implemented so that strategists could make "slightly easier" opponents without having to modify their heuristics. I kept this feature disabled in my studies but did ask participants how they would make a slightly easier opponent. It was interesting that they all gave the same answer: pick a move randomly every once in a while. While this would make an easier opponent, it is more like playing against someone vaguely paying attention. Regardless, their suggestion could also be implemented into a future version to allow learners to test their ideas.

One student in my study suggested a more sophisticated way to pick best moves based on risk aversion. This astute suggestion made me revisit ways a "best move" could be defined with the gaming calculator. Future versions of the gaming calculator could provide the ability to punch into something like a command-line tool or offer a plug-in architecture to allow dedicated learners to perform more sophisticated analyses at different stages of their strategic search. I give more details about this approach in the next section.

5.1.4 Finding Patterns on the Game Board

The arrangements of pieces on a game board can be described in many ways. The evaluation sentences in the current software are a good start but hardly complete.

Implemented, but disabled to limit the focus of the evaluation, were game-board relationships such as "distance to" and arrangements of pieces in patterns (e.g., three pawns in a diagonal line).

Another implemented but disabled pattern-matching feature allowed for conditionals. With this feature enabled, evaluations would not be considered unless some game state condition was met. For example, you could more highly value your knights in chess once all of your pawns had been captured. This allows for a strategy to unfold in stages.

A future direction for describing game-state patterns would be to change how pieces are referred to. Instead of referring to game pieces by name, they could be referred to by some numbered rank (hand assigned or machine assigned). This would allow for many strategies to be interoperable between games, so that people could see how well their chess strategy would operate in a checkers game and vice versa.

I have not exhausted all of the ways to describe patterns on a game board. Furthermore, I am sure that other creative minds will find patterns that are difficult to describe succinctly with the evaluation-sentences approach I implemented. Therefore, future versions of the software should offer a command-line tool, which provides direct access to a machine representation of the game state so that motivated parties could create new tools to find patterns. This approach could also be implemented with a plug-in architecture.

5.2 Partial Information, Iterative Games and the Rest of Game Theory

Games in which you are not sure which cards you might be dealt, or which cards your opponents have in their hand, are referred to as games of partial information. Effectively strategizing for these games requires evaluation of probabilities and risk analysis. For example, when playing cards, what are the chances that your opponent has an ace in their hand? Probability is a tricky concept for many people to understand. Research could be done to find appropriate ways to represent choices involving probability to learners. Representation problems would extend to algorithm-animation considerations, since probabilistic choices would be hard to represent graphically.

Games played in rounds, like poker or iterative prisoner's dilemmas, present new strategy representation problems since players' histories can be analyzed to predict future behavior. For example, opponents might bluff, and a strategist would want to account for this behavior in their automated strategy. It is not clear how one would create a robust evaluation language to describe changing strategies in lieu of other players' histories.

Partial information and iterative games are just two features of a very large taxonomy of game-theoretical analyses of games. Not discussed here are design considerations for non-zero sum games, games with multiple players, and simultaneous games. To make each of these game types accessible to a tool like the game calculator will require research. Once appropriate representations are found for these different games, a gaming calculator could tailor its strategy-language options to that particular game's rules.

5.3 Simulations

Most behavior in computer simulations is driven by functions, in which the person playing the game, through his or her choices, indirectly controls one or more independent variables. Changing the independent variable moves the dependent variable, indirectly updating features of the simulation's behavior. For example, in *The Oregon Trail* simulation, the further you move your wagon across the country (the independent variable), the greater the chance of your wagon breaking down (the dependent variable). What makes this simulation re-playable is the element of chance (your wagon might not break down) and choice (you can equip your wagon differently to help prevent a breakdown).

How functions govern behavior in two-dimensional grid simulations, such as *SimCity*, is not as clear as it is in a "one-dimensional" command-line game such as *The Oregon Trail*. This is because multiple choices, made over time to a large area (a city), are summed in aggregate as the dependent variable. To the player, razing many buildings in the city might appear to be a number of different actions, while the simulation engine just adds the number or razed buildings into a sum, which is the independent variable. Burn down enough of your city and eventually people start to move out (the dependent variable, responsible for simulated behavior).

If the governing functions were available (as the published code and graphs for *The Oregon Trail* cited in Chapter 1 of this dissertation) they become a great asset to learners. In addition to graphing governing functions, dependent and independent variables could be highlighted on the graph, and graphical links to their input and output in the 2-D simulation could be displayed (e.g., highlight and count the razed buildings). Visualizing a simulation's governing function, maybe providing something like a simulation dashboard, could help to elucidate how simulations are designed.

Building your own simulation, and defining your own governing functions could provide a deeper and more interesting learning experience in the constructionist tradition. There are a few ways you could scaffold simulation-function design. One way would be to provide existing functions, such as hyperbolas, parabolas and even equations for s-curves, and then ask learners to map simulated behavior to these functions. This would be a "force-fit" approach and would stifle a lot of creativity.

A better approach would be to let simulation designers define their own functions and use those in their simulations. They could do this by sketching functions to describe relationships. While functions designed this way might not be expressible in concise equations, they would provide students with the ability to be authors without being bogged down in complex expressive algebra.

5.4 Frenetic Video Games

Not every frenetic video game has agency; asteroids and Tetris blocks just tumble towards you. But many video games do have agents, and, therefore, strategies for their behavior could be modeled in a new version of the gaming calculator, if the games were grid-based. The new toolkit would support simultaneous movements of multiple game-pieces. For example, in this possible *Pac-Man* scenario, there are six possible next moves:

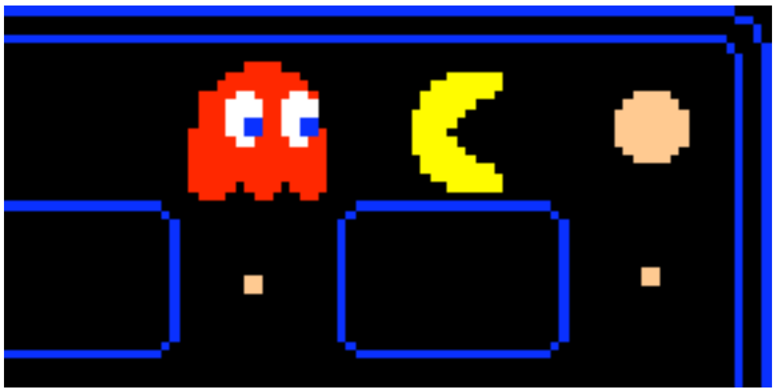


FIGURE 5.1 PAC-MAN CHASED BY BLINKY THE GHOST

Pac-Man has been modeled as an example program in many educational computer game design toolkits (Papert 1996; Begel 1997). Since programs in the software toolkit AgentSheets (Repenning, Ioannidou et al. 2000) are executed on a spreadsheet-like grid, their suggested method for programming *Pac-Man* ghosts' behavior is closest to the methods that I describe in this dissertation. In *AgentSheets*, ghosts' behavior can be programmed with a hill-climbing algorithm that works like this: Pac-Man's cell is given the highest value and lower scores are diffused into surrounding cells; ghosts' instructions are to move towards higher valued cells.

The hill-climbing algorithm is similar to the minimax algorithm in that both consider and rank their options before making a move. But the hill-climbing algorithm only ranks its immediate options; its tactic is to react to its environment. The minimax algorithm behaves in the same way if it looks only one move ahead, but the algorithm is designed to search further ahead and evaluate countermoves; its strategy is to plan ahead. The difference between these two approaches to designing behavior could be described as tactical versus strategic decision-making.

5.4.1 State Machines

Another way to design autonomous behavior for frenetic video games is to use state machines. State machines work by describing desired behavior and changes in behavior with high-level descriptive models. For example, in the video game *Pac-Man*, the default ghost behavior is to "hunt Pac-Man". After Pac-Man eats a power pellet, the ghosts switch into a new state: "run away from Pac-Man". The implementation of this behavior change, or how to program the movement of the ghosts, is cleanly separated from the description of the desired behavior. This approach is different, but in the spirit of Pane's design goals (Pane 2002); this approach, too, translates natural-language descriptions of behavior into machine rules.

5.5 Future Pedagogical Work

While it is fantastic that short, intense gaming workshops such as those studied for this research yield interesting results, there are obvious limitations to this approach. First and foremost, this researcher is not available to run every gaming workshop for every interested game player, and the tools are also not available for the long-term work that many teenagers expressed an interest in pursuing.

Many teenagers enjoyed being game designers but were rushed through the process. In future workshops, students should be allowed much more time to experiment with the game design possibilities of this toolkit. Once they develop a good sense of what makes an interesting game, then they could be introduced to my game design tools. Even this process could be slowed down to offer more time for learners to use 1-ply tactical searches until they grow comfortable with the ideas and their representation.

Many students asked for copies of the software so they could continue to play after the workshop ended. One of my students recommended turning the gaming calculator into an online website so that he could continue his work and share games with his friends. This excellent suggestion could make the educational opportunities described in this dissertation available to more people. It could conceivably enable the formation of a community of game and behavior designers.

Many people learned to make their own web pages using examples found with the “view source” feature in web browsers. Looking at other people’s HTML, they learned how to markup and thereby describe layouts. People could learn to design autonomous behavior in a similar way by viewing the source code of other people’s games and strategies.

Works Cited

Baird, D. G., R. H. Gertner, et al. (1994). Game Theory and the Law. Cambridge, Massachusetts, Harvard University Press.

Baker, R. N. (1999). Cards in the Classroom: Mathematics and Methods. Ketchikan, Alaska, University of Alaska**:** 23.

Begel, A. B. (1997). Bongo: A Kids' Programming Environment for Creating Video Games on the Web. Department of Electrical Engineering and Computer Science. Cambridge, Massachusetts Institute of Technology. **M.Eng.**

Berlekamp, E. R., J. H. Conway, et al. (2001). Winning Ways for Your Mathematical Plays. Wellesley, Massachusetts, AK Peters, Ltd.

Blankinship, E., B. Smith, et al. (2004). "Closed caption, open source." BT Technology Journal **22**(4): 151-159.

Brady, J. M. and R. B. Emanuel (1978). "An Experiment in Teaching Strategic Thinking." Creative Computing **4**(6): 106-109.

Brams, S. J. (1980). Biblical Games: Game Theory and the Hebrew Bible. Cambridge, Massachusetts, MIT Press.

Bruckman, A. (1998). "Community Support for Constructionist Learning." Computer Supported Cooperative Work: The Journal of Collaborative Computing **7**: 47-86.

Bruckman, A. and E. Edwards (1999). Should We Leverage Natural-Language Knowledge?
An Analysis of User Errors in a Natural-Language-Style Programming Language.
Proceedings of the 1999 Conference on Human Factors in Computing Systems, Pittsburgh, PA, ACM Press.

Burrille, G. (2002). Handheld Graphing Technology in Secondary Mathematics: Research Findings and Implications for Classroom Practice, Michigan State University**:** 122.

Carlson, E. (1969). Sixth Graders and Sumeria. Learning Through Games: A New Approach to Problem Solving. Washington, D.C., Public Affairs Press**:** 141-166.

Design-Based Research Collective (2003). "Design-Based Research: An Emerging Paradigm for Educational Inquiry." Educational Researcher **32**(1): 5-8.

Entertainment Software Association (2005). Essential Facts About the Computer and Video Game Industry.

Fine, G. A. (1983). Shared Fantasy: Role-Playing Games as Social Worlds. Chicago and London, The University of Chicago Press.

Golick, M. (1998). Card Games for Smart Kids, Sterling Publishing Company, Incorporated.

Jeffery, C. L. (1998). A Menagerie of Program Visualization Techniques. Software Visualization: Programming as a Multimedia Experience. J. Stasko, J. Domingue, M. H. Brown and B. A. Price. Cambridge, The MIT Press**:** 73-79.

Kafai, Y. and M. Resnick, Eds. (1996). Constructionism in Practice: Designing, Thinking, and Learning in a Digital World. Mahwah, Lawrence Erlbaum Associates.

Kafai, Y. B. (1995). Minds in Play: Computer Game Design as a Context for Children's Learning. Hillsdale, Lawrence Erlbaum Associates.

Kahn, K. (2000). "Programming by example: generalizing by removing detail." Communications of the ACM **43**(3): 104 - 106.

Kahn, K. (2001). Source Code Should be Animated. http://www.toontalk.com/Papers/ddj.pdf

McDaniel, R. G. (1999). Creating Whole Applications Using Only Programming-by-Demonstration. Computer Science Department. Pittsburgh, Carnegie Mellon University.

Opie, I. and P. Opie (1969). Children's Games in Street and Playground. London, Oxford University Press.

Orwant, J. (1999). EGGG: The Extensible Graphical Game Generator. Media Arts and Sciences. Cambridge, Massachusetts Institute of Technology. **PhD**.

Pane, J. F. (2002). A Programming System for Children that is Designed for Usability. Computer Science Department. Pittsburgh, Carnegie Mellon University. **PhD**.

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York, Basic Books, Inc.

Papert, S. (1996). The Connected Family. Atlanta, Longstreet Press.

Papert, S. (1998). Does Easy Do It? Children, Games, and Learning. Game Developer**:** 88.

Papert, S. and C. Solomon (1970). NIM: a game-playing program. LOGO memo; no. 5. AI memo; 254. Cambridge, Massachusetts Institute of Technology Artificial Intelligence Laboratory.

Pell, B. (1993). Strategy Generation and Evaluation for Meta-Game Playing. The Computer Laboratory, University of Cambridge. **PhD**.

Piaget, J. (1965). The Rules of the Game. The Moral Judgment of the Child. New York, The Free Press**:** 13-108.

Rabin, S. (2000). The Magic of Data-Driven Design. Game Programming Gems. M. DeLoura. Rockland, Charles River Media, Inc.

Rawitsch, D. (1978). Oregon Trail. Creative Computing. **4:** 132-139.

Repenning, A., A. Ioannidou, et al. (2000). "AgentSheets: End-User Programmable Simulations." Journal of Artificial Societies and Social Simulation **3**(3).

Resnick, M. (1994). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, The MIT Press.

Rowling, J. K. (2005). Harry Potter and the Half-Blood Prince. Fairfield, Arthur A Levine Books.

Russell, S. J. and P. Norvig (1995). Artificial Intelligence: A Modern Approach. New Jersey, Prentice-Hall, Inc.

Shaffer, D. W. and M. Resnick (1999). ""Thick" Authenticity: New Media and Authentic Learning." Journal of Interactive Learning Research **10**(2): 195-215.

Smith, D. C., A. Cypher, et al. (1994). "KidSim: Programming Agents Without a Programming Language." Communications of the ACM **37**(7): 54-67.

Squire, K. D. (2002). "Cultural Framing of Computer/Video Games." Game Studies **2**(1).

Squire, K. D. (2003). Replaying History: Examining learning social studies through playing Civilization III playing in urban learning environments. School of Education. Bloomington, Indiana University. **PhD**.

Starr, P. (1994). Seductions of Sim: Policy as a Simulation Game. The American Prospect. **5**.

Strohecker, C. (1991). Why Knot? Media Arts and Science. Cambridge, Massachusetts Institute of Technology. **PhD:** 554.

Suthers, D., J. Connelly, et al. (2001). Representational and Advisory Guidance for Students Learning Scientific Inquiry. Smart Machines in Education. K. D. Forbus and P. J. Feltovich. Menlo Park, American Association for Artificial Intelligence & The MIT Press**:** 7-35.

Tilley, R. (1967). Playing Cards. London, Octopus Books Limited.

Turkle, S. (1995). Life on the Screen: Identity in the Age of the Internet. New York, Simon & Schuster.

Wattenberg, M. and M. Walczak (2005). Thinking Machine 4. http://www.turbulence.org/spotlight/thinking/

Weintraub, E. R., Ed. (1992). Toward a History of Game Theory. Durham, Duke University Press.

Weisstein, E. W. "Minimax Theorem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/MinimaxTheorem.html