

STRATEGY GAME PROGRAMMING PROJECTS

Timothy Huang

Department of Mathematics and Computer Science

Middlebury College

Middlebury, VT 05753

(802) 443-2431

huang@middlebury.edu

ABSTRACT

In this paper, we show how programming projects centered around the design and construction of computer players for strategy games can play a meaningful role in the educational process, both in and out of the classroom. We describe several game-related projects undertaken by the author in a variety of pedagogical situations, including introductory and advanced courses as well as independent and collaborative research projects. These projects help students to understand and develop advanced data structures and algorithms, to manage and contribute to a large body of existing code, to learn how to work within a team, and to explore areas at the frontier of computer science research.

1. INTRODUCTION

In this paper, we show how programming projects centered around the design and construction of computer players for strategy games can play a meaningful role in the educational process, both in and out of the classroom. We describe several game-related projects undertaken by the author in a variety of pedagogical situations, including introductory and advanced courses as well as independent and collaborative research projects. These projects help students to understand and develop advanced data structures and algorithms, to manage and contribute to a large body of existing code, to learn how to work within a team, and to explore areas at the frontier of computer science research.

We primarily consider traditional board games such as chess, checkers, backgammon, go, Connect-4, Othello, and Scrabble. Other types of games, including video games and real-time strategy games, might also lead to worthwhile programming projects. Nevertheless, traditional strategy games have several advantages: they are familiar to many students, easy to learn, and focussed on problem solving rather than real-time response (though competitive play often involves the use of time clocks).

By “computer players,” we refer to computer programs that select moves to play and function as virtual opponents for human players or for each other. While the design and construction of a program that allows humans to play against each other can be an interesting task in its own right, we think that the development of an intelligent game-playing program provides even greater challenges and learning opportunities.

The remainder of this paper is organized as follows. Section 2 discusses the motivation for using strategy games in course projects. Sections 3 and 4 describe how Connect-4 and Mancala can be used in introductory and advanced courses, respectively. Sections 5 and 6 examine how Chinese checkers and go can be used for independent and collaborative research projects, respectively. Section 7 summarizes our experiences and explores other possibilities for projects based on strategy games.

2. MOTIVATION

Computer science is not a spectator sport. Students learn much more from actually solving problems, designing algorithms, and writing programs than from attending lectures and taking notes. Hence, programming assignments play a crucial role in computer science courses at all levels, and the content of those assignments merits careful consideration.

Strategy games have many characteristics that make them attractive programming projects. Their simple rules help students quickly understand the requirements of a computer player, and their well-defined winning conditions provide straightforward measures of a computer player’s performance. The task of building a program that performs “intelligently” challenges students in a way that is highly motivating and fun. Also, the development of computer players requires students to learn many concepts and skills. They design data structures to represent board positions, develop algorithms to search game trees, and evaluate tradeoffs between different heuristic evaluation functions. For more complex systems, they read and contribute to a substantial body of existing code. For team projects, they learn to discuss ideas and coordinate development efforts with fellow students. Lastly, the wide range of difficulty, from tic-tac-toe and Connect-4 to chess and go, provides a variety of project possibilities for students at all levels.

3. CONNECT-4 IN CS2

We begin by describing Connect-4, a board game sold by Milton-Bradley, and showing how a programming project based on Connect-4 has been used in a CS2 course on data structures and algorithms. Students come to CS2 with varying backgrounds. Some have taken only one previous computer science course, while others already have substantial programming experience. The limited number of move choices in Connect-4 make it simple enough for beginning students, while the actual goal of playing effectively keeps it challenging for more advanced students.

Connect-4 is played with black and red checkers pieces. Players take turns dropping one piece at a time into the columns of a vertical grid of seven columns and six rows. Each piece falls until it sits right above the topmost piece in the column or hits the bottom of an empty column. The player who first connects four pieces of his or her color in any direction – horizontally, vertically, or diagonally – wins the game (see the sample game in Figure 1).

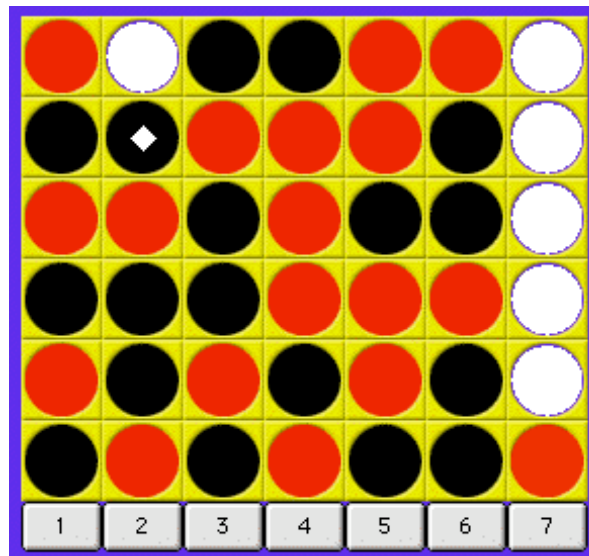


Figure 1: With red (dark gray) to move, red can win by playing on top of the marked piece in column 2.

From a game-theoretic perspective, Connect-4 is solved. In 1988, Victor Allis proved the correctness of nine strategic rules that guarantee a win to the player who moves first [1]. Based on these rules, he wrote a Connect-4 program called VICTOR. Using standard search techniques and a database of 500,000 positions, VICTOR always wins when playing first on a standard 7 x 6 board. Although game theorists and AI researchers have moved onto other games, the actual process of writing a Connect-4 computer player remains a formidable task for introductory computer science students.

In our CS2 course, we provided students with most of the skeleton code for the game, along with an executable so they could see how their finished programs ought to work. We first asked students to write a computer player that chooses moves without using game-tree search. Our goal was for them to read the existing code and determine how to build on it. We then asked students to implement a computer player based on minimax search. Our goals were for them to understand the data structure representing the board, to construct a search algorithm using that data structure, and to devise a heuristic evaluation function. Finally, we held an optional tournament on an 8 x 8 board (whose game-theoretic result is not yet known).

Overall, we were pleased with the effectiveness of the Connect-4 assignment. For many students, it was their first experience reading a relatively large body of existing code, and they found the exercise both challenging and worthwhile. Other students found that they learned the most in the process of implementing minimax search, even though they were given the actual algorithm in pseudocode. Many students, not just the more advanced ones, enjoyed the light-hearted tournament and the opportunity to experiment with different evaluation functions.

4. MANCALA IN UPPER-LEVEL ARTIFICIAL INTELLIGENCE

We continue by describing the ancient game of Mancala and showing how a programming project based on Mancala has been used in an upper-level course on artificial intelligence (AI). An AI course provides the natural setting for a project

involving strategy games. After all, alpha-beta search and the development of heuristics were important early contributions of the field. Students in the course tend to have a substantial amount of programming experience, though usually in only one or two languages.

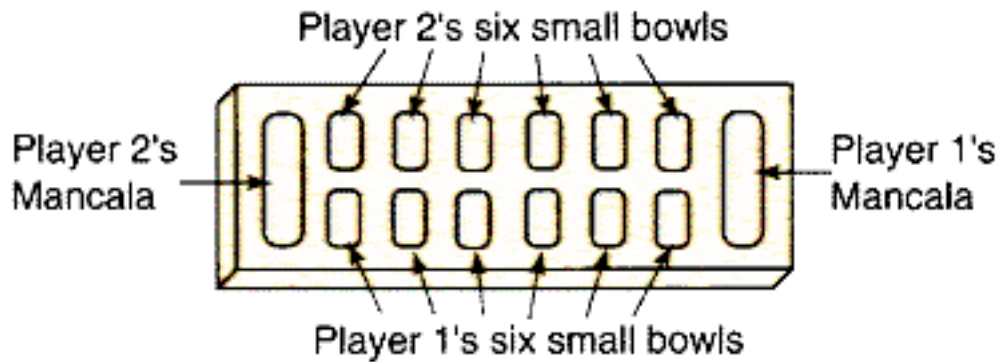


Figure 2: Empty Mancala board. The game starts with a fixed number of pebbles (usually 4 or 6) in each small bowl.

Invented in Africa over 3000 years ago, Mancala is played on a board with 12 small bowls arranged in two rows facing each other. At both ends of the rows are larger bowls, or Mancalas. Each player owns one row of bowls and the Mancala to the right of that row (see Figure 2). The game begins with some number (usually 4 or 6) of pebbles in every small bowl and no pebbles in the Mancalas. The goal is to collect the most pebbles in one's Mancala. At each turn, a player chooses one of his non-empty bowls, removes the pebbles from that bowl, and distributes them one at a time in counter-clockwise fashion into the subsequent bowls, including his own Mancala and the opponent's small bowls but excluding the opponent's Mancala. If the last pebble lands in his own Mancala, the player gets another turn. If the last pebble lands in one of his empty bowls, and the opponent's bowl directly across the board is non-empty, then the player moves that pebble and all of the pebbles in the opposing bowl directly to his Mancala. The game ends when one player has no stones in his small bowls. The other player then moves all the pebbles remaining in his small bowls to his own Mancala, and the player with the most pebbles wins. There exist many variations on these rules, though most are quite similar.

In our AI course, we provided students with the rules of Mancala and asked them to implement both the control program for playing legal Mancala games and a computer player based on minimax search with alpha-beta pruning. The goals were for students to understand and implement the alpha-beta search algorithm and to gain substantial experience programming in Lisp, a language that most students had not learned before the course.

Overall, the Mancala assignment met our goals, though we uncovered some areas for improvement. While many students found that programming in an unfamiliar language posed unwelcome challenges, they agreed that it ultimately gave them a new perspective on algorithmic problem solving. For example, the Mancala rule that allows multiple turns (when the last pebble lands in the Mancala) complicates the task of determining what moves are available at any given time. One approach involves treating each selected bowl with a separate level in the game tree. An alternate approach involves treating each sequence of selected bowls as a single level in the game tree. In weighing the tradeoffs between each approach, students discovered the

expressive power of list structures for storing and representing sequences of arbitrary length. Many students also found it instructive to implement alpha-beta pruning. While some understood the algorithm in theory, most found that they did not fully grasp how it proceeded or how it reduced search time until they actually wrote the search code themselves.

Some students felt that writing all the code for the Mancala control program was tedious. We think that students should sometimes have the opportunity to carry a programming project all the way from design to full implementation and testing, and that this would be reasonable for a self-contained subject such as Mancala. Nevertheless, in some situations it may be more appropriate to provide students with some skeleton game code and to have them write only the move generation, search, and heuristic evaluation functions. Another concern was that the simplest evaluation function – the difference between the number of stones in each player's Mancala – was good enough to discourage experimentation with other evaluation functions. One way to encourage further investigation is to provide students with the executable for a computer player that defeats other players using such an evaluation function.

5. CHINESE CHECKERS AS INDEPENDENT STUDY PROJECT

In this section, we describe the game of Chinese checkers and show how it has provided a rewarding platform for independent projects. Students who pursue independent projects tend to be highly motivated and to have completed most of the requirements for a computer science major. An independent project provides the opportunity for a student to explore topics outside or beyond the scope of traditional course offerings. Among the topics that a project based on Chinese checkers could cover are complex heuristic evaluation functions, multi-agent coordination and competition, real-time reasoning, and graphical user interfaces.

Despite its name, the game of Chinese checkers originated from a European board game called Halma, which was developed around 1880. Chinese checkers is played by two to six players on a six-pointed star-shaped board. There are 121 positions on the board – 10 in each of the star points, and 61 in the middle area (see Figure 3). Each player starts with 10 pegs all in one star point. The goal is to be the first to move all 10 pegs to the star point directly across the board. At each turn a player can either slide a peg to an adjacent empty position or can make a sequence of one or more jumps with a single peg. Each jump consists of moving the peg over any adjacent peg into the position on the other side of that peg, which must be empty. At no time are pegs removed from the board during play.

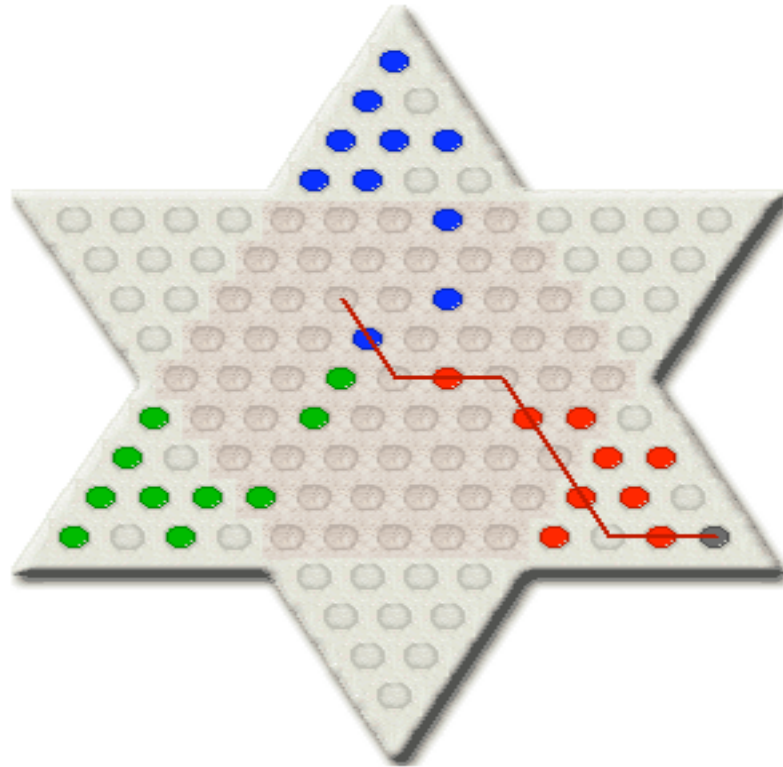


Figure 3: The path shows a peg moving from the tip a star point to the center of the board via a sequence of jumps

We supervised several students on informal independent projects in the context of designing and implementing computer players for a Chinese checkers contest. The contest involved teams of one to three undergraduate students who wrote Chinese checkers players that communicated with a server program via a simple interface language. With constraints on the amount of computing time allowed to choose moves, each contestant played a series of games against one, two, and five other computer opponents. Unlike traditional programming contests, where all the programming is done on the day of the contest, student participants in this contest relaxed on contest day while watching their programs compete with each other.

Our students enjoyed the opportunities to practice their coding skills, explore advanced topics, and participate in a competition. To begin with, they wrote code to generate all legal moves in a given position and to communicate using the Chinese checkers interface language. To select among the available move choices while managing a limited amount of computation time, they implemented iterative-deepening search algorithms and experimented with a variety of heuristic evaluation functions. After getting their players working, they built graphical user interfaces and added test suites to more easily visualize, assess, and tune the performance of their players. Furthermore, they investigated approaches for adapting their search algorithms for games with more than two players. Such games provide additional challenges, such as the possibility of alliances being formed and broken among some of the players.

All of the students who participated in the Chinese checkers contest thought that it added a valuable dimension to their undergraduate experience. They enjoyed the opportunity to study and implement time-constrained search algorithms, to experiment with different evaluation functions, and to build graphical user interfaces. They also

appreciated the experience of sharing, discussing, and evaluating each other's ideas. One possible improvement would be to place greater emphasis on multi-agent issues. The option for up to six Chinese checkers players at one time provides a natural opportunity to explore this area of AI research.

6. GO AS COLLABORATIVE RESEARCH PROJECT

In this section, we describe the game of go and show how it has provided a rewarding area for collaborative research. Invented in Asia over 3000 years ago, *go* (called *weiqi* in China and *baduk* in Korea) is almost certainly the oldest strategy game still widely played. Although go players in Europe and the US number only about 100,000 and 20,000, respectively, go players in the Far East number over 25 million. Each year, Chinese, Japanese, and Korean newspaper companies spend the equivalent of millions of US dollars to sponsor tournaments for the roughly 700 professional go players in the world.

Go is played with white and black stones on a grid of 19 x 19 intersecting lines (sometimes 9 x 9 or 13 x 13 grids are used). Players take turns placing one stone at a time onto one of the 361 intersections. Once placed, stones do not move; however, stones can capture stones of the opposing color by surrounding them completely on all four sides. The game ends when both players pass. The player who has surrounded the most intersections (points of territory) with his or her stones wins the game. Figure 4 shows an example game on a 13 x 13 board.

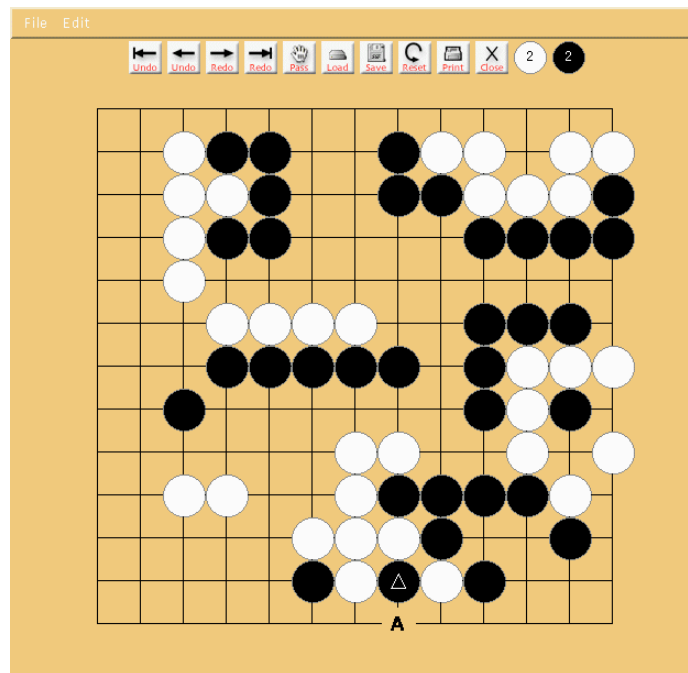


Figure 4: Midway through an example game on a 13 x 13 board.

From a computer scientist's perspective, what stands out about go is that it remains a perplexing computational problem. Unlike the best computer programs for Connect-4, Mancala, Chinese checkers, western checkers, chess, backgammon, and Scrabble, all of which play at or better than the level of the best human players, the best programs for go play only at the level of an advanced beginner [2].

The standard approach based on game-tree search fails for two primary reasons. First, the search tree for go is much larger than most other games. For example, the average branching factor in chess is about 35, whereas the average branching factor in go is about 200. Second, heuristic evaluation functions for go positions tend to be inaccurate and computationally expensive. Unlike chess, where the difference in material is easily computed and provides a reasonable estimate of which side is winning, go positions have no easily extracted features from which to quickly estimate the game state. Although researchers continue to make incremental improvements to the performance of go programs, most believe that the goal of competing with the world's top human players remains decades away.

We started a formal go research project in the Fall of 1999 with three main goals: to better understand the computational challenges posed by go, to explore AI approaches for improving the performance of go-playing programs, and to provide collaborative research opportunities for undergraduate students. To date, we have designed and developed much of the infrastructure to support go playing, both on a local machine and over the Internet (via IGS, the Internet Go Server). This includes a full graphical user interface as well as support for saving and loading games in SGF (smart game format), annotating board positions, and printing board snapshots (see Figure 5). We've also implemented a standard game-player based on minimax search. As mentioned earlier, however, search trees in go are extremely large, and efficient evaluation functions are difficult to construct. Indeed, many opportunities remain for students to contribute to the design of the computer player.

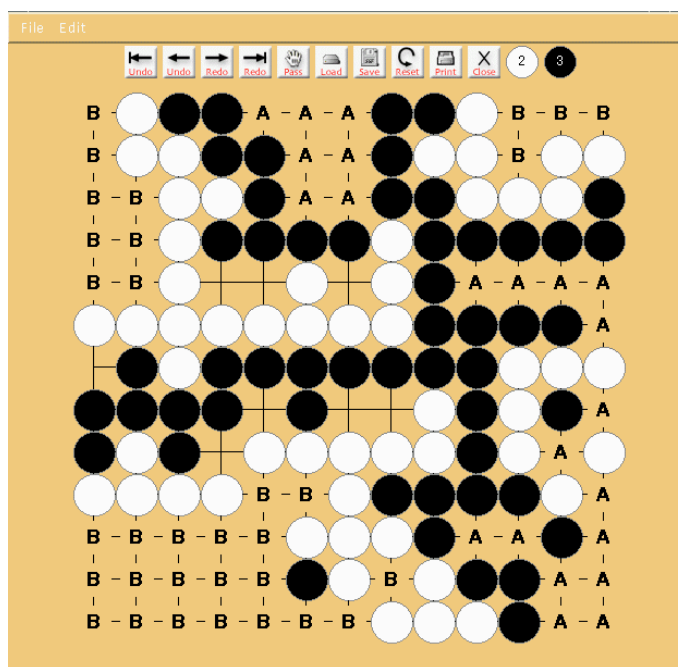


Figure 5: Annotated board position. Black's territory is marked with "A", and white's territory is marked with "B".

The opportunities for improving the performance of the computer player range from straightforward to extremely difficult. Thus, any student with an interest in computer go can find something appropriate to pursue. For example, two students built the graphical user interface depicted in Figures 4 and 5, and one student added support for annotating boards and printing board snapshots. Another student

investigated and implemented a selective search routine for identifying a class of go patterns known as ladders. This routine will eventually become an integral component of a more sophisticated board evaluation function.

Overall, our go project has provided an ideal environment for collaborative research with undergraduate assistants. The problems are easy to understand yet difficult to solve. We continue to explore alternative approaches such as reinforcement learning, neural networks, and genetic algorithms. These all help give students a deeper understanding for and appreciation of some of the work at the frontiers of artificial intelligence research.

7. CONCLUSIONS AND FUTURE WORK

We have described four strategy games – Connect-4, Mancala, Chinese checkers, and go – and have shown how they have been used as programming projects in introductory and advanced courses as well as independent and collaborative research projects. In each case, the game provides an easily understood, well-defined, and engaging problem with many possibilities for student learning and growth. In particular, they provide an ideal vehicle for students to learn about advanced data structures and algorithms, to build and enhance larger-scale programs, to work with one another, and to learn about current research. Online examples of Mancala and Chinese checkers games based on these projects can be found at [3].

We continue to explore other strategy games and their pedagogical benefits. For example, backgammon and Monopoly provide opportunities to discuss probability theory and uncertain reasoning, while Scrabble and bridge raise questions about reasoning with incomplete information. Although researchers may lose interest in a game once a computer player is developed that can defeat all humans, the games themselves continue to provide challenging, instructional, and fun projects.

REFERENCES

- [1] Allis, V. A Knowledge-based Approach of Connect-Four. Report IR-163, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, The Netherlands. 1988.
- [2] Burmeister, J. and Wiles, J. “The Challenge of Go as a Domain: A Comparison Between Go and Chess.” In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems*. Perth, Australia. 1995.
- [3] <http://www.jgames.com>