Multiplayer Board Game Strategies in the introductory CS curriculum

PI: Ivona Bezáková, Co-PIs: James Heliotis and Sean Strout
Department of Computer Science, Rochester Institute of Technology
Student developers: Adam Oest and Paul Solt

Abstract
Many board games provide a natural context for the use of basic data structures and search algorithms taught in a typical introductory CS sequence. Unlike traditionally used programming assignments where students implement the actual game, we provide the game “engine” and ask the students to implement player strategies. The engine graphically displays the current state of the game and cyclically calls the individual player strategies to perform their moves. The students need to apply the same algorithms as if programming the rule checker for the game. And with the added strategy component, the project becomes open-ended, leaving space for continued improvements and experimentation.

A sample game: Quoridor
- By Mirko Marchesi, published by Gigamic Games.
- Played on a 9x9 initially empty grid, the players want to get their pawn across the board.
- For 2 or 4 players (here we describe the 2 player version).
- The players start from opposite sides of the board - their pawn is placed in the middle of their side:
- Every player has 10 walls of length 2.
- In every move, a player chooses to either
  (a) move their piece to an adjacent location in a direction not blocked by a wall, or
  (b) place a wall along the grid lines, aligned with the grid intersections, not crossing an existing wall and entirely inside the grid.
A player can never block another player from reaching their destination.

Intro CS content in Quoridor
- Need to keep the current state of the board.
  There are multiple possibilities, for example:
  - recording the wall locations in a 2d array (careful, the walls are "in-between" the array cells!),
  - keeping a list of wall positions,
  - recording adjacency relations between the grid cells and updating them with new wall placements.
- Additionally, need to check the validity of a move (have all walls been used? is the wall placed at a valid location? does it overlap with any existing walls?).
- Finally, use breadth-first-search (BFS) to get the pawn to the destination along the shortest possible route.

Benefits of the strategy component
We aim to get students more engaged in their introductory courses and CS in general. By adding strategies into the picture, the students
- observe and experiment with different levels of functionality,
- become aware of topics beyond the introductory sequence (for example, several of our students self-studied parts of AI/machine learning to improve their strategies),
- have fun competing against their peers and instructors.
Supplemental information

Implementation from the student's perspective
Students need to implement three functions:
- `init()`: called by the engine at the beginning of the game, to initialize their data structures (the parameters inform the student of the number of players and other essential information)
- `move()`: called by the engine whenever it is their turn to perform a move; the student needs to compute and return the move of their player
- `last_move()`: called by the engine after each move, informing the student of the move; the student needs to update their data structures

Grading
We target students at all programming levels. Anyone who consistently beats a random/unsophisticated player gets full score. If students implement additional functionality (i.e., a better strategy), they get rewarded by participating in a college-wide tournament, not by added points in the course.

Project parts and other logistics
We split the project into four parts, roughly split as follows:
- Part 1: set up the data structures, implement basic searching (e.g., find a shortest path between two locations in Quoridor)
- Part 2: play the game as a single player, generating valid moves
- Part 3: play the game with multiple players, taking into account their moves
- Part 4: implement a strategy that beats a random player
We do not specify which data structures or algorithms to use. We hold problem solving sessions where students discuss their approaches before they start implementing them. The students are welcome to change their approach after the problem solving sessions.

Evaluation of educational impact
To evaluate our hypothesis that implementing strategies is more exciting than implementing a rule checker for a game, in 2012/2013 we:
- Split the students randomly into two groups: the study group and the control group.
- Students in the study group implement player strategies.
- Students in the control group implement a rule checker.
- The project parts are set up so that the study group and the control group students need to use similar type of data structures and algorithms. The problem solving sessions are identical for both groups.
- At the end of the term, we will evaluate the impact of the strategy-writing project both qualitatively by pre- and post-surveys, as well as quantitatively by analyzing the students' grades. Thanks goes to our evaluation team: Trudy Howles (the main evaluator), Doug Baldwin, and Sage Miller.

Student demographics
The project is a graded component in CS2 (2nd quarter). Over 400 students (computer science, software engineering, computer engineering, computational math, applied math, and other majors) take the course.

Game selection criteria
- simple rules
- well liked (e.g., on boardgamegeek.com)
- multiple strategy choices (none clearly better than others)
- game converges (to prevent never-ending matches)
- reasonably easy to display (grid-like games work well)
- aligns with introductory CS content
- for 2 (and ideally also 4) players

Development so far
2009/10:
The aMAZEing Labyrinth by Max Kobbert, published by Ravensburger.
2010/11:
Quoridor by Mirko Marchesi, published by Gigamic Games.
2011/12:
San Francisco Cable Cars by Dirk Henn, art by Michael Menzel, published by Queen Games.
2012/13:
Quoridor. We decided to go with a previously used game to minimize unknown factors during the main project evaluation.

What comes next
- Currently using Python. Java support is in the testing stages.
- Evaluation of strategy vs. rule checker implementation.
- Web interface for online tournaments.
- Additional games.