Board Game Strategies in Introductory Computer Science

Ivona Bezáková
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, NY 14623, U.S.A.
ib@cs.rit.edu.edu

James E. Heliotis
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, NY 14623, U.S.A.
jeh@cs.rit.edu.edu

Sean P. Strout
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, NY 14623, U.S.A.
sps@cs.rit.edu.edu

ABSTRACT

We present three open-ended freshman projects where students design and implement their own player strategies for well-established board games: Quoridor by Mirko Marchesi (Gigamic), San Francisco Cable Cars by Dirk Henn (Queen Games), and The aMAZEing Labyrinth by Max J. Kobbert (Ravensburger). Unlike modern computer-based games, most board games are inherently discrete. For example, the board tends to have a fixed number of allowed positions for the game pieces and every player performs a search through a finite number of possible moves to decide which move to take next. As such, designing a player strategy for a board game provides a very natural context for basic data structures, searching algorithms, and other concepts typically covered in a freshman-level computer science sequence. Furthermore, the project allows for continual improvements to one's strategy, targeting both beginners as well as more advanced programmers.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Algorithms, Experimentation, Performance

Keywords

Introductory computer science, Learning in context, Basic data structures and algorithms, Open-ended project

1. INTRODUCTION

Motivating a full class of students in a freshman computer science course is challenging, especially if the students come from a variety of backgrounds and previous programming experiences. We address this situation by developing CS2-level projects where students design and implement player strategies for well-established (yet not necessarily widely known) board games. Board games provide a natural context for introductory data structures and algorithms, as both the layout of the game and the playing style (i. e., turn-taking) are typically discrete. Depending on the game, it might require the students to implement a graph search, or continually update and search through a 2D data structure. Previous research has demonstrated that learning to program in context increases student motivation [23, 24].

The students implement player modules that compute their players' moves and keep track of the current game state. We provide the "engine," a program that makes round-robin calls on the student modules to make their moves, enforces the rules of the game, and graphically displays the current state of the game. It also informs the student modules of the other players' moves so that the modules can update their data structures to reflect the current state of the game.

To achieve full credit we simply ask our students to consistently beat an unsophisticated or random playing strategy – we feel that this difficulty is compatible with a CS2-level project. In other words, we do not base the actual grading on competitiveness. However, the motivated student has the opportunity to keep improving their strategy and have it tested against their classmates' strategies in the end-of-term tournament that we dubbed the "Battle Royale." Our preliminary findings indicate that board games not only provide a suitable context but that programming player strategies further increases student motivation by being open-ended and providing opportunities for experimentation.

1.1 Related work

Games have been used as a motivation tool [2, 5, 6, 16] in computer science curriculum for several decades and have flourished into an active area in computer science educational research. The majority of approaches using games for educational purposes fall into these categories: game design and development (for a sample of recent publications, see, e. g., [3, 25, 13, 17, 28]), and playing games to learn (e. g., [8]). "Game design and development" typically means to implement a human-controlled game, not a player strategy, with student excitement often stemming from the accompanying multimedia experience [26, 19].

Our projects do not follow either approach – we use a well-established board game to provide the context for the concepts from introductory computer science courses and ask the students to design and implement a player strategy. As far as we know, the closest works related to our projects are Davies' Uno [22], the last part of Bayliss' Unreal Tournament [3], and Kurkovsky's games for mobile devices [14]. Davies' Uno is a CS1 project where students use variables, conditionals, and simple loops to experiment with possible player strategies for the card game Uno. This project provides an excellent context for introductory programming concepts but, naturally since it addresses CS1 audience, not introductory data structures and algorithms. Bayliss' students spent the majority of the time implementing the Unreal Tournament (i. e., doing game development), with the last three weeks devoted to programming their own bot behaviors. Unlike in our case, the nature of the game is continuous where bots turn, move and face opponents in shooting matches. In the end, the bot-behavior part received much less attention than the traditional game development.

Similarly, Kurkovsky's students focused on game development, with a short amount of time devoted to programming game elements reacting to the actions of a human player. In this case the setup was also continuous and, moreover, the students did not implement a strategy of a human player but presumably a simpler behavior of various game creatures.

Summet, et al. [27] proposed an interesting context for CS1 centered around programming robot strategies: their students implemented various robot behaviors, using physical robot devices. However, the robots do not competitively interact.

Another interesting group of recent publications centers around using board games in CS1/2. Drake and Sung [7], and Kurmas and Vanderhyde [15, 22] discuss an interesting variety of board (and dice and card) games and their relevance to CS1/2 topics; however, none of these works take advantage of implementing player strategies. On the other hand, player strategies for games like Tic-Tac-Toe are a favorite topic in introductory artificial intelligence courses [9].

2. THE BOARD GAMES

We briefly sketch the rules for all three board games; see Figure 1 for snapshots of the games.

2.1 The aMAZEing Labyrinth

Designed by Max J. Kobbert and published by Ravensburger [12], the game consists of a 7x7 tiled area, where every tile is an I-shaped, T-shaped, or L-shaped corridor. Every other row and column can be shifted. There is an extra tile that is used to do so, thereby changing the paths in the labyrinth. Each player has a wizard with a distinct home position on the board, plus a randomly generated list of treasures to collect before returning home. Treasures are depicted on some of the tiles and the treasure is collected whenever a player's wizard lands on the corresponding tile. In every turn, a player first changes the board by inserting the extra tile and then moves his or her wizard any distance along an existing path. The game works for 2-4 players.

2.2 Quoridor

Designed by Mirko Marchesi and published by Gigamic [20], this game consists of a 9x9 board and works for 2 or 4 players. Each player places a single piece in the middle of one edge of the board. Its goal is to arrive somewhere along the edge at the opposite side of the board. Additionally to their piece, each player has ten walls of length 2. In each move, a player either moves their piece to an adjacent location, or places a wall. A wall must be aligned with the grid lines and it cannot cross an already placed wall or extend out of the board. Walls are used to obstruct and therefore detour opponents' movement. However, the wall placement cannot completely block the opponent from reaching their destination.

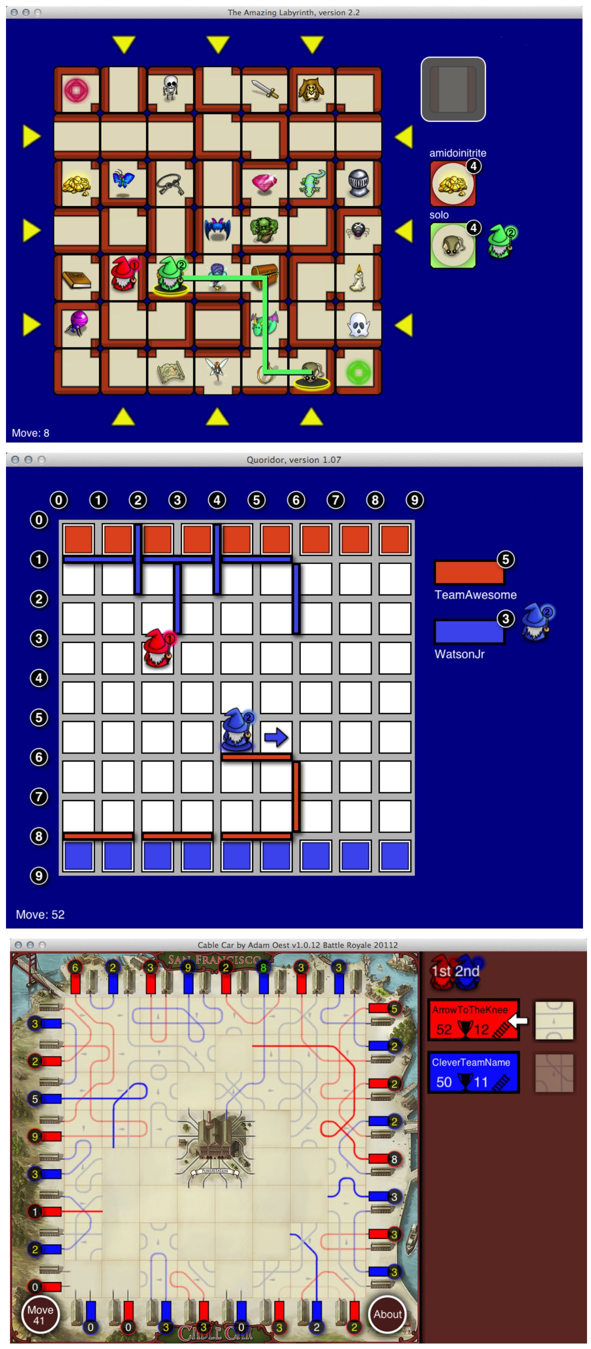

Figure 1: Labyrinth, Quoridor, and San Francisco Cable Cars depicted by our engines. The figures show games in progress, in each case a battle between the top two student players of the respective academic year.

2.3 San Francisco Cable Cars

Designed by Dirk Henn (art by Michael Menzel) and published by Queen Games [11], this game for 2-6 players is played on an 8x8 initially empty board, with 32 cable car stations distributed along the sides of the board. The rules specify which stations belong to each player. Each player gets a randomly drawn tile that specifies four connections, each between two of the eight end-points on the tile's sides. In every move, a player rotates their tile and places it on the board adjacently to an already placed tile or to a side of the board. By doing so, the players build cable car routes (theirs or their opponents' – or both). The game finishes after all tiles have been placed and the player with the overall longest cable car routes wins.

3. HOW THE PROJECT WORKS

3.1 The student module

The students implement three functions:

An initialization function. This function is called by the engine at the beginning of the game. The parameters of the function specify the initial state of the game, including the number of players, the student's player ID (to determine order of play), the initial game layout (e. g., in Labyrinth, the initial tile layout), etc. Students initialize their data structures accordingly – from this moment on the engine does not communicate the current game state; it is the student's responsibility to keep track of it.

A move function. This function is called whenever it is the student's turn to move. The student returns a move object specifying the move of their player.

An update function. This function informs the student of other player's moves, so that the student can update their data structures accordingly.

As a side note, to help the students with their strategies in Labyrinth, we opted to reveal the sequence of treasures to be collected for each player. Similarly in Cable Cars, we opted to reveal the tiles currently held by every player.

3.2 The engine

The engine starts by initializing all student modules. Then, it calls the first player's move function, followed by a call to each player's update function[1], informing them of the first player's move. Then, the engine calls the second player's move function, followed by the update function of the other players. This scenario continues until the game is over.

3.3 The project assignments

To guide the students during this project, we split it into four assignments. Roughly speaking, the first assignment is a "get started" part that asks the students to implement some useful function, for example graph search in Labyrinth. The second assignment is a "single player mode," where the students implement their move function but they do not need to worry about other players yet. In the third assignment, a "multi player mode," the students update their data structures based on other player's moves – they need to be able to play the full game. In the fourth "strategy" assignment we ask the students to design and implement a strategy that beats an unsophisticated player that we provide.

[1]Our design is evolving to inform the player module of its own move as well. This allows us to sometimes provide additional useful information, and it makes the student code's handling of game state updates more consistent.

We opted to have the students do the first assignment individually to fully understand how the engine works. For the subsequent assignments they work in teams of two. We devote three problem-solving (active learning) and laboratory sessions to the first three assignments, typically in the second, fourth, and seventh week of the (10 week) quarter.

3.4 Battle Royale

On the Saturday between the end of classes and the start of final exams, a friendly competition called Battle Royale takes place. An auditorium is outfitted with computer screens for the currently playing game and for a chart showing the winners of each round. Pizza is provided in the area just outside the auditorium for the students, instructors, and their guests. This voluntary competition is completely outside of the grade scheme for the course itself. Instead, prizes are awarded to the top teams (gift certificates at the college book store).

3.5 Grading

We de-emphasized the competitive aspect of the project for grading purposes – a correctly implemented strategy that used basic data structures and searching algorithms easily earned a top score. (For a full score in functionality we only require the player strategy to consistently beat the weaker of the modules that we provide with the engine, i. e., a randomly playing strategy.) We based our decision to suppress competitiveness on research suggesting that underrepresented groups tend to dislike, and not perform well, in a competitive environment [10]. We had very favorable experience with our proposed grading scheme: a vast majority of the teams eventually beat the provided module, reporting to have learned the underlying concepts and stating that the project "was fun."

4. WHY BOARD GAME STRATEGIES

We based the project on programming board game strategies for a number of reasons:

Discrete context to demonstrate basic CS concepts. Board games tend to be discrete. We argue that a discrete model fits better with the introductory courses and basic concepts such as searching algorithms and data structures ranging through arrays, lists, queues, and graphs.

In the Labyrinth project students stored the data in a 2D array (some even contemplated a 1D array) or as a graph representing the labyrinth, using either the adjacency matrix or adjacency lists. The variety led to a nice problem solving discussion of advantages and drawbacks of each data structure. Additionally, independently of the strategy, the students needed to search for their current treasure, search the tile layout for possible paths, and enumerate through several possible moves (for example different insertion locations for the extra tile) to choose what they consider to be the most appropriate move. Different strategies, ranging from heuristics to searching through the configuration space while anticipating the opponents' moves, may be incorporated.

Similar algorithms and data structure concerns relate to the Quoridor project. Even though the maze does not continually shift, the students needed to account for wall placement. This added an interesting dimension as the students needed to keep track of objects (game pieces) on the board as well as on the grid lines (walls). Unlike in Labyrinth, in Quoridor it is desirable to move along a shortest path, requiring the students to implement this functionality.

Cable Cars brought a set of different programming concerns. Since a cable route never splits, this project did not require graph search – a simple linear traversal was enough. The main hurdle to deal with was the representation of the game state, namely the connections of the route segments in the tiles placed on the board.

Established games. Board games are well-tested. If they were not interesting, they would not achieve popularity. We have found this preferable to the difficult task of inventing games that are both well-designed and interesting [4].

Room for experimentation. There is a wide variety of strategies. (If there were only one dominant strategy, the game would loose its appeal.) Typically there are some obvious strategies that are good enough to beat a random player, but the teams have the option to go beyond the required part of the project and experiment further. This means that the project has a clearly defined objective, yet it is open-ended.

Variety. There is a wide variety of board games [1] to choose from, allowing us to rotate through games and reduce the risk of plagiarism from previous students.

Visual feedback. The students get immediate visual feedback on the performance of their modules.

Why have we decided to have our students implement player strategies instead of doing traditional game development? While game development provides an interesting and relevant context for basic computer science concepts (and often for advanced concepts as well), the design of a player strategy is much more open-ended, providing motivation to complete the work, and to keep working on the project beyond the typical call of duty. Also, game development assignments often motivate students to put a lot of effort into the multimedia experience [18]. This tendency is natural, since the stimulation of the senses sparks student excitement. However, in our experience multimedia development can detract from data structures and algorithmic concepts. By providing the engine which includes the graphics, we give the students the satisfaction of a visual feedback, yet they concentrate on the implementation of the concepts in the underlying data structures and algorithms.

Another advantage of using a real-life established board game is that we are able to show the students the physical game and leave it in the student center for them to play (and study its properties!) during their leisure time.

5. GAME SELECTION CRITERIA AND EXPERIENCES

We started with the following list of selection criteria for the board games.

- Reasonably simple rules.
- Underlying data structures and algorithms that fit within the scope of CS2-level courses.
- Well-ranked at an on-line game review website [1]. (This implies the existence of a variety of strategies and an overall appeal of the game.)
- Non-violent and ethnicity- and gender-neutral.

Over the years we started adding other criteria, that, while not deal-breaking, influenced our game selection decisions.

- Playable by 2 and 4 players. Two player mode is important for our grading scheme (described below). The four player mode is not as crucial but it became a Battle Royale tradition and it gives the students a chance to play against multiple opponents at once and see what that does to their strategies.
- Few ties. Some games (e. g., Tsuro [21]) often end up in a tie, especially in the two player mode. This does not make for an interesting strategy development or match-watching.
- Not much randomness. If there is a lot of randomness, the game might be fun for real people but the strategy may not determine the winner.
- Some randomness. Quoridor does not include any randomness and many students opted to not use a randomized strategy. Thus, if there was a tie, we could switch the starting order for a second match but the third rematch was identical to one of the two earlier matches. We are addressing this by including different starting positions for the game.
- Game progress. With Labyrinth some students implemented overly defensive strategies where a match consisted of blocking the opponent from reaching their destination. This led to never-ending games. We started enforcing a maximum number of moves to solve this issue, as well as limiting the amount of time allotted for each player's computation.
- Reasonable visual display. Thus far we avoided 3D games, as well as games that do not use a bounded play area (such as Carcassonne [29]).

6. COURSE SETUP AND STUDENT DEMOGRAPHICS

We used The aMAZEing Labyrinth project in 2009/10, the Quoridor project in 2010/11, and the San Francisco Cable Cars project in 2011/12. We used the projects in the CS2 course in the winter quarter (December through February). The course covers introductory data structures and algorithms such as graphs, breadth and depth first search, backtracking, and hashing. The course was taken by 202 students in the first year, followed by about 350-400 students in both subsequent years. The course is offered in multiple sections of 50-60 students each, taught by a variety of instructors (overall, 10 instructors have taught the course so far). The majority of the students declared themselves as computer science majors, followed by software engineering majors, then computer engineering majors, as well as applied math and bioinformatics majors. Additionally, one or two sections of the course were offered to up to 90 students in the spring quarters. Thus far over a thousand students have taken part in this project.

7. PRELIMINARY FINDINGS

The project motivated the students to understand basic computer science concepts and see beyond the "computer science is just programming" view that we frequently encounter with freshman students. Many students enjoyed the experimenting part and were eager to learn more about upper-level topics such as artificial intelligence. These sentiments are captured by the following sample of student comments:

- *I am proud of every part of this project. When introduced to the idea at the beginning of the quarter I was flabbergasted and did not believe I could accomplish such a task but now that I have I am very confident that computer science is where I belong.*
- *Being very interested in AI, I'll probably return to this program down the road as I learn more about artificial intelligence.*
- *It was a fun project, and I might try to improve my player to win more often, or at least not get closed in so easily. I should probably stop making it rotate to the left every time.*
- *Overall, it was a pretty good project.*
- *The project overall is very interesting and fun. I really enjoy it.*
- *I think the competition was a good idea, since it gave a lot of people motivation to work harder on the project.*
- *The project wasn't necessarily hard. Meeting the minimum requirements was definitely not that much of a challenge. It was the competitive aspect that provoked me to do a bit more than what is required.*
- *The project was fun. (several times)*
- *Nice project idea, can't wait until CS3.*
- *The project was helpful and enjoyable. I liked the way the class was set up.*
- *The project is what made this course interesting. Rather than 4 more individual, disjointed, labs, the project gave us a chance to put several things we had learned together into one thing.*
- *I really like cable car. It's a creative game, and it was challenging to design an AI for but yet it was doable.*
- *I think competition is a great way to foster creativity and pride in one's accomplishments.*
- *This was such a cool project. Not only did we learn a lot, it was a fun experience. I really enjoyed the fact that we could write the code the way we wanted. Overall it was an awesome experience!*
- *Overall, however, the project was interesting and useful in demonstrating practical application of programming skills taught in class.*
- *I really liked the project and how it related to the course materials.*
- *Overall, I believe this project was a very good opportunity for me to improve my programming abilities. Along with the algorithms that we covered in class, the project allowed me to try some new techniques to improve the strategy for the many different parts. Without this experience, I don't think I would be as confident about my programming abilities as I am following its completion.*
- *I guess what I'd like to communicate is that while I don't think I learned much in terms of coding or algorithms over the course of this project, I learned an incredibly important design lesson, which was not to over-complicate things. I'd say this project provides a good testing grounds for students with some self-taught programming knowledge, or students with knowledge from prior courses, to realize what parts of their current programming strategies are stupid.*
- *It was tough. I barely pulled through. Probably worth it.*

We encountered several negative comments as well. Mainly they were about team collaboration and not specific to this project, thus we do not cite them here.

We also report preliminary evaluation results stemming from a post-quarter survey from winter 2011/12. The students responded to a list of statements with their sentiments ranging from "not true at all" to "very true". Out of 217 student responses, 152 students (70%) agreed with "Having to develop the algorithms helped me to see that the project was more than just programming."; 139 students (64%) agreed with "I enjoyed the competition aspect of the project.", 134 students (62%) agreed with "The project helped me to better understand computer science."; 132 students (61%) agreed with "I enjoyed the experimentation with different problem solving strategies." The majority of the other responses were neutral: the disagreeing percentages are 14%, 18%, 19%, and 18%, respectively. The main project evaluation will take place in 2012/13; see Section 8.1.

8. FUTURE PLANS

8.1 Formal evaluation

In 2012/13 we will compare strategy design to game development. We will split the students randomly into a control group and study group. The control group will implement parts of the engine, namely rule enforcement, keeping track of players in the game, score computation, but not graphics. The study group will design and implement player strategies as before. Both groups will need to keep track of the current game state and implement very similar search algorithms to generate/verify moves. Both groups will end the quarter with a voluntary mini competition: Battle Royale for the study group and a "best engine" for the control group (this will be judged by a panel of instructors). We will compare the groups through normal grade keeping but also with before and after surveys that measure how much they learned during the quarter, their interest in the project, their motivation to keep improving their code, and their enthusiasm for computer science in general.

8.2 Programming language support

All the game engines from the previous years ran under Python 2.6 due to the graphical environment Pyglet that is, unfortunately, not supported in later versions of Python. Through a very different approach developed in the summer of 2012, we have now successfully migrated to Python 3. The new architecture has also allowed us to add a Java version of the engine. The reason is that the graphical component has moved to a web browser, thus minimizing the amount of code that has to be rewritten when a new language is adopted. Once stable, these engines, along with the supporting documentation, will be available for use by other instructors and institutions.

8.3 Web interface

In the future, we envision a web site hosting a variety of tournaments, some devoted to a single course, as well as those where students can try their strategies against those written by teams from different universities.

9. ACKNOWLEDGMENTS

We gratefully acknowledge NSF funding, award ID 1044721. The Labyrinth and Quoridor engines were developed by Sean Strout and Paul Solt. The Cable Cars engine development and overhaul of the Quoridor engine in 2012 was done by Adam Oest.

Additionally, we are thankful for the feedback of our evaluation team led by Trudy Howles, with consultants Doug Baldwin, and Sage Miller.

10. REFERENCES

[1] BoardGameGeek / Gaming Unplugged Since 2000, a database of player reviews, session reports, images, and news. `http://www.boardgamegeek.com/`.

[2] T. Barnes, H. Richter, E. Powell, A. Chaffin, and A. Godwin. Game2Learn: building CS1 learning games for retention. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '07)*, pages 121–125, 2007.

[3] J. D. Bayliss. Using games in introductory courses: tips from the trenches. In *Proceedings of the ACM SIGCSE '09*, pages 337–341, 2009.

[4] B. Burd, J. Goulden, B. Ladd, M. Rogers, and K. Stewart. Computer games in the classroom, or, how to get perfect attendance, even at 8 am. In *Proceedings of the ACM SIGCSE '07*, page 496, 2007.

[5] D. C. Cliburn. The effectiveness of games as assignments in an introductory programming course. In *Proceedings of the Thirty-Sixth ASEE/IEEE Frontiers in Education Conference (FIE '06)*, 2006.

[6] D. C. Cliburn and S. Miller. Games, stories, or something more traditional: the types of assignments college students prefer. In *Proceedings of the ACM SIGCSE '08*, pages 138–142, 2008.

[7] P. Drake and K. Sung. Teaching introductory programming with popular board games. In *Proceedings of the ACM SIGCSE '11*, pages 619–624, 2011.

[8] M. Eagle and T. Barnes. Experimental evaluation of an educational game for improved learning in introductory computing. In *Proceedings of the ACM SIGCSE '09*, pages 321–325, 2009.

[9] E. El-Sheikh and L. Prayaga. Development and use of AI and game applications in undergraduate computer science courses. *Journal of Computing Sciences in Colleges*, 27(2):114–122, 2011.

[10] S. M. Haller, B. Ladd, S. T. Leutenegger, J. Nordlinger, J. Paul, H. M. Walker, and C. Zander. Games: good/evil. In *Proceedings of the ACM SIGCSE '08*, pages 219–220, 2008.

[11] D. Henn. *San Francisco Cable Cars*. Queen Games, 2009. `http://www.queen-games.de`.

[12] M. J. Kobbert. *The aMAZEing Labyrinth Board Game*. Ravensburger, 1986. `http://www.ravensburger.com`.

[13] M. Kölling and P. Henriksen. Game programming in introductory courses with direct state manipulation. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*, pages 59–63, 2005.

[14] S. Kurkovsky. Engaging students through mobile game development. In *Proceedings of the ACM SIGCSE '09*, pages 44–48, 2009.

[15] Z. Kurmas and J. Vanderhyde. Board game project ideas for CS 1 and CS 2 (abstract only). In *Proceedings of the ACM SIGCSE '12*, page 658, 2012.

[16] S. T. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. In *Proceedings of the ACM SIGCSE '07*, pages 115–118, 2007.

[17] M. C. Lewis and B. L. Massingill. Graphical game development in CS2: a flexible infrastructure for a semester long project. In *Proceedings of the ACM SIGCSE '06*, pages 505–509, 2006.

[18] S. Ludi. The benefits and challenges of using educational game projects in an undergraduate software engineering course. In *Games and Software Engineering Workshop, as part of the International Conference on Software Engineering (ICSE 2011)*, 2011.

[19] A. Luxton-Reilly and P. Denny. A simple framework for interactive games in CS1. In *Proceedings of the ACM SIGCSE '09*, pages 216–220, 2009.

[20] M. Marchesi. *Quoridor*. Gigamic, 1997. `http://www.gigamic.com`.

[21] T. McMurchie. *Tsuro*. Calliope Games, 2004. `http://www.tsuro.com/`.

[22] N. Parlante, J. Zelenski, D. Zingaro, K. Wayne, D. O'Hallaron, J. T. Guerin, S. Davies, Z. Kurmas, and K. Debby. Nifty assignments. In *Proceedings of the ACM SIGCSE '12*, pages 475–476, 2012.

[23] M. Savin-Baden. *Facilitating Problem-Based Learning*. McGraw-Hill, 2003. Berkshire, UK.

[24] M. Savin-Baden and C. H. Major. *Foundations of Problem-Based Learning*. McGraw-Hill, 2004. Berkshire, UK.

[25] D. L. Schuster. CS1, arcade games and the free java book. In *Proceedings of the ACM SIGCSE '10*, pages 549–553, 2010.

[26] B. Stephenson and C. Taube-Schock. QuickDraw: bringing graphics into first year. In *Proceedings of the ACM SIGCSE '09*, pages 211–215, 2009.

[27] J. Summet, D. Kumar, K. J. O'Hara, D. Walker, L. Ni, D. S. Blank, and T. R. Balch. Personalizing CS1 with robots. In *Proceedings of the ACM SIGCSE '09*, pages 433–437, 2009.

[28] K. Sung, M. Panitz, S. A. Wallace, R. Anderson, and J. Nordlinger. Game-themed programming assignments: the faculty perspective. In *Proceedings of the ACM SIGCSE '08*, pages 300–304, 2008.

[29] K.-J. Wrede. *Carcassonne*. Rio Grande Games, 2000. `http://www.riograndegames.com/`.