AC 2010-1563: USING THE TETRIS GAME TO TEACH COMPUTING

Yung-Hsiang Lu, Purdue University

Yung-Hsiang Lu is an associate professor in the School of Electrical and Computer Engineering. In 2008, he was one of the three recipients of Purdue "Class of 1922 Helping Student Learn Award." In 2004, he obtained the National Science Foundation Career Award. He obtained the Ph.D. degree from the Department of Electrical Engineering at Stanford University. This study is supported in part by NSF CNS 0722212 "CPATH EAE: Extending a Bottom-Up Education Model to Support Concurrency from the First Year." Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the view of the sponsor.

Guangwei Zhu, Purdue

Guangwei Zhu received Bachelor's degree in Automation at Tsinghua University, Beijing. He is currently a Ph.D. candidate and teaching assistant in Electrical and Computer Engineering at Purdue University. He received Magoon's Award in Teaching Excellence in Spring 2009. His research interests include control theory, applied mathematics and object orient design and programming.

Cheng-Kok Koh, Purdue University

Cheng-Kok Koh received the B.S. degree with first class honors and the M.S. degree, both in computer science, from the National University of Singapore in 1992 and 1996, respectively. He received the Ph. D. degree in computer science from University of California at Los Angeles in 1998.

Currently, he is an Associate Professor of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana. His research interests include physical design of VLSI circuits and modeling and analysis of large-scale systems.

Cheng-Kok Koh received the Lim Soo Peng Book Prize for Best Computer Science Student from the National University of Singapore in 1990, and the Tan Kah Kee Foundation Postgraduate Scholarship in 1993 and 1994. He received the GTE Fellowship and the Chorafas Foundation Prize from the University of California at Los Angeles in 1995 and 1996, respectively. He received the ACM Special Interest Group on Design Automation (SIGDA) Meritorious Service Award in 1998, the Chicago Alumni Award from Purdue University in 1999, the National Science Foundation CAREER Award in 2000, the ACM/SIGDA Distinguished Service Award in 2002, and the Semiconductor Research Corporation Inventor Recognition Award in 2005.

Using the Tetris Game to Teach Computing

Abstract

This paper presents a programming project using the Tetris game as a platform to integrate various computing concepts typically scattered in multiple courses, including discrete mathematics, algorithms, data structures, networking, linear algebra, and object-oriented design. The project is divided into four stages: (1) generating Tetris pieces, (2) developing single-player games, (3) developing two-player games through networks, and (4) devising game strategies for competition. Students' feedback suggests that this project is effective in integrating these concepts, promoting innovation, and motivating students.

1 Introduction

Technology has become an integral part of everyday life and has profound impacts on economy, education, as well as national security. A "technologically literate" person is characterized by three dimensions: knowledge, thinking and acting, and capabilities [1]. It is critical for citizens to understand the potential, limitation, and risk of technology when forming their opinions about public policies, for example, whether paper verifications should be printed by electronic voting machines [2], how privacy should be protected in government's computers [3], and whether government should subside broadband networks [4].

A critical step in improving technological literacy is increasing the interest in learning, formally or informally, concepts related to technology. This can be achieved in many ways, such as organizing outreach activities, presenting programs in mass media, or encouraging students to take courses related to technology. In recent years, the interest in studying technology-related fields, namely STEM (science, technology, engineering, and mathematics) has been steadily declining in USA; declining enrollments in STEM have caused great concerns [1, 5]. Many factors contribute to this decline of interest; among them, students often fail to understand and to integrate concepts from different areas. As a result, students cannot recognize the importance of, for example, mathematics and physics in their everyday lives. Many students enjoy playing computer games and games usually need collision detection. The calculation requires analytical geometry; the law of reflection and surface materials govern objects' speeds and directions after collisions. Computer games have been frequently used as examples to improve students' interests in studying STEM subjects. Some educators use computer games to study how students learn [6, 7]. This paper presents a project that

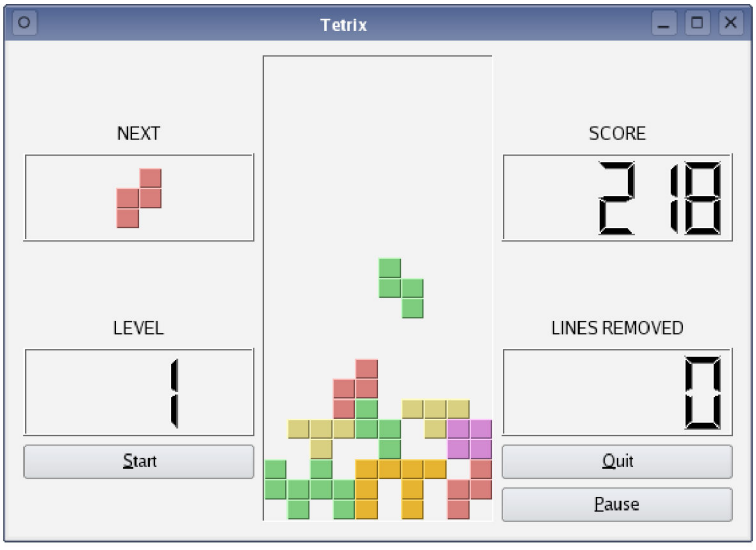

Figure 1: Snapshot of Qt's Tetrix example.

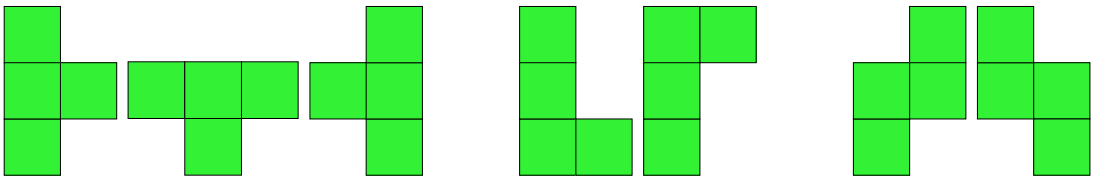

Figure 2: (a) Three pieces with different orientations; Tetris allows rotations so these pieces are the same. (b) and (c) Two pairs of pieces that are mirrors but cannot be obtained by rotations; these four pieces are considered unique in Tetris.

uses a computer game to help computer engineering students integrate many subjects learned in different courses.

Tetris is one of the most popular computer games. Tetris has seven pieces, each with four squares. A player rotates a piece or move it horizontally as it falls. When a horizontal line is completely filled by squares, the line is eliminated. The player's score increases as a new piece enters the Tetris window or when a line is eliminated. Figure 1 is a snapshot of a Tetris game. A player's goal is to maximize the number of eliminated lines given a sequence of Tetris pieces. Tetris is a 2-dimensional packing problem and it is NP-complete [8]. Many problems related to Tetris are also NP-complete, even for off-line games when the sequences are known in advance [8]. It is computationally expensive to find the minimum height or the maximum number of cleared rows. Some researchers consider Tetris as an optimization problem [9]. Some educators use Tetris for teaching game development [10–15].

In the fall semester of 2009, we used Tetris as a semester-long project in a course of object-oriented programming. This course teaches both Java and C++; the Qt library is used to create graphical user interfaces for C++ programs. The students had taken at least two programming courses (C Programming and Advanced C Programming) as prerequisites. Many students had taken or were taking discrete mathematics and data structures. The project extends the original Tetris program by adding pieces of 5, 6, or 7 squares per piece. The project requires students to:

- create programs with graphical user interfaces and handle user inputs.
- use timers and handle timer events.
- apply permutations and combinations to generate additional Tetris pieces.
- communicate with another program through the Internet using TCP sockets.
- develop intelligent strategies to rotate and horizontally move pieces.

The main purpose of adopting a popular computer game as a course project is to motivate students in learning these concepts and skills.

The project is divided into four stages: (1) generating Tetris pieces, (2) developing single-player games, (3) developing two-player games through networks, and (4) devising game strategies for competition. The forty-six students were divided into fifteen teams of three or four people per team. They changed teammates in different stages so that the students could interact with more classmates and learn different programming styles. Students decide which language (C++ or Java) to use and they may change languages in different stages. The detail of each stage is explained in the following sections.

2 Stage 1: Generating Pieces

In the first stage, students developed algorithms to generate the additional pieces of 5, 6, or 7 squares. There are 280 different pieces with 4, 5, 6, or 7 squares per piece (excluding one 7-square piece that has a hole at

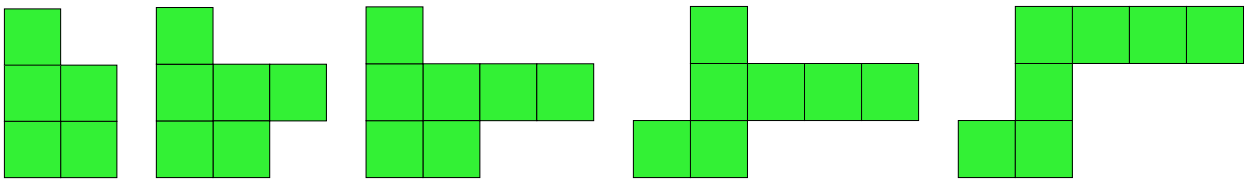

Figure 3: Five different Tetris pieces: (a) 5 squares, (b) 6 squares, (c)-(e) 7 squares. (c) and (d) are different by shifting the bottom row.

the center). Tetris allows rotations; hence, the three pieces in Figure 2(a) are considered the same. Tetris does not allow mirrors and the two pairs in Figures 2(b) and (c) count as four unique pieces. Figure 3 shows five different pieces of 5, 6, or 7 squares. In this stage, students applied their knowledge about permutation and combination to generate the new pieces. The students also applied linear algebra to rotate pieces and detect duplicates.

The students used several different ways to generate the pieces. The most common approach starts with a two-square piece and grows the piece by adding another square in six possible locations. These 3-square pieces are then used to generate 4-square pieces by adding another square in different locations. This process continues recursively until 7-square pieces are generated. This is a bottom-up approach.

Another approach adopts a top-down strategy by using *integer partition*. To generate 7-square pieces, the value 7 is partitioned into the sums of positive integers; for example, $7 = 4 + 3 = 1 + 3 + 2 + 1 = 2 + 2 + 3$ are three different ways to partition 7. Each number in the partitions represents the number of squares in a row. Figure 3(c) and (d) use the partition of $1 + 4 + 2$ and Figure 3(e) uses the partition of $4 + 1 + 2$. The rows are then shifted to produce different pieces. Figure 3(c) and (d) show two different pieces by shifting the last row.

Another common algorithm uses a grid by selecting different and connected cells to fill. In this stage, performance was not critical but students were advised not to use an algorithm that was "apparently inefficient." An example of an inefficient algorithm is as follows. A piece has at most 7 squares and a 7×7 grid can be used to indicate the locations of the squares. There are C_7^{49} ways to select 7 squares in this grid of 49 squares. Among the $C_7^{49} \approx 8.6 \times 10^7$ combinations, most are invalid because the squares are not connected. Hence, this algorithm is inefficient.

Both the top-down and the bottom-up approaches of generating Tetris pieces produce duplicates. Duplicates may occur for two reasons: (1) adding a square to two $(k-1)$-square pieces at different locations may produce identical k-square pieces, or (2) rotating some pieces may produce identical pieces. After the pieces are generated, duplicates are eliminated. Most of the students used 2-dimensional arrays (i.e. matrices) to represent the pieces. To detect duplicates, the students applied their knowledge in linear algebra by using matrix transformations.

3 Stage 2: Developing One-Player Games

The second stage handles user inputs, rotates and moves the falling pieces, detects collisions, and eliminates lines. This stage is designed to convey the concepts of object-oriented programming. Essential techniques include class, encapsulation, inheritance, event handling, and file input/output. In order to compete in the final stage, all teams had to agree on the original orientations of pieces. A file was provided to all teams and the pieces were defined in the file. By reading this file, the students' programs did not have to generate the pieces when the programs started and could reduce the initialization time. Students learned the differences between off-line computation and on-line computation. The internal representations of a piece could vary, for example, using 2-dimensional arrays or adjacency matrices. Different representations required different ways to handle rotations. The programs had to handle user inputs from keyboards and mouses. Moreover,

the keys could be reconfigured by players and the programs had to be flexible enough to redefine keys at run-time.

Students encapsulated methods into corresponding classes and avoided static functions (similarly to functions in C). For example, rotation should be a member method of the Tetris piece class because rotating each piece produces a unique result. There are up to 280 different pieces and this makes it impractical writing a "C-Style" function using more than 200 cases inside a `switch` block. For the students that had no prior experience in object-oriented programming, the crucial concept is the combination of data and operations by creating member functions. This stage has relatively few requirements for algorithm design. Hence, students could focus on writing well-structured programs in the anticipation of the next two stages. This stage provided important experience writing extensible programs that could be improved by adding new features later.

4 Stage 3: Developing Two-Player Games

The third stage extends the project to consider two players. The two players may use different keys of the same keyboard on the same computer. The players may also play through the Internet. This is the most difficult stage due to the wide range of new concepts to learn. First, students changed their team partners in each stage. Thus, in the third stage, each student had worked with two different groups of classmates. Starting from the second stage, each team had to choose a code base among the team members' code from the previous stage and learn the pros and cons of different programming styles from their partners. Second, this stage required many new features and students learned the importance of writing well-structured code that could be easily extended. Third, the students learned how to use built-in classes for network communication. Moreover, they had to organize their programs so that the algorithms for determining rotations and positions could be improved in the fourth stage.

We provided a game server to which two game clients were connected. For simplicity, the clients and the server were fully synchronized: each move or rotation command from the client received an acknowledgment from the server. A client could send another command only after receiving the acknowledgment of the previous command. The two players competed by sending eliminated lines to the bottoms of the opponents' windows. Figure 4 shows an example. When a line is eliminated, the line is sent to the bottom of the opponent, without the most recently filled squares. In this example, when the rotated `T` piece falls, two lines are sent to the right side simultaneously. When these two lines are eliminated again by the right player, the lines are completely removed so that squares do not accumulate. The programs had to keep track of the sources of squares on the board: from a new piece or from the opponent. In a two-player game, speed is also important. If a player can eliminate more lines and send them to the opponent, the former has a better chance of winning. The students also learn how to make robust programs. One requirement of the third stage is that a program must not crash (for example segmentation fault or exception). Run-time exceptions had to be caught. Common problems include being unable to find the server, unexpected disconnection by the server or by the opponent. Even though the server sends only messages allowed by the specification, the other client may send unexpected messages. In the fourth stage competition, one team's program crashed when the opponent sent the team's name in a format different from the specification. The first team should have reported a wrong format from the opponent without an uncaught exception.

5 Stage 4: Devising Strategies for Competition

In the final stage, the students' programs competed across the Internet using the game server. A program had to pass a qualification by eliminating at least 200 lines in a sequence given in advance. The sequence contained mostly 4-square pieces with occasional pieces of 5, 6, or 7 squares. Among the fifteen teams in the class, ten teams passed qualification. The teams that passed the qualification competed against each other. The last four teams entered the semifinals and the final competitions. Figure 5 shows the server's view of the competitions. An additional constraint was added in this stage. To prevent any player from sending an

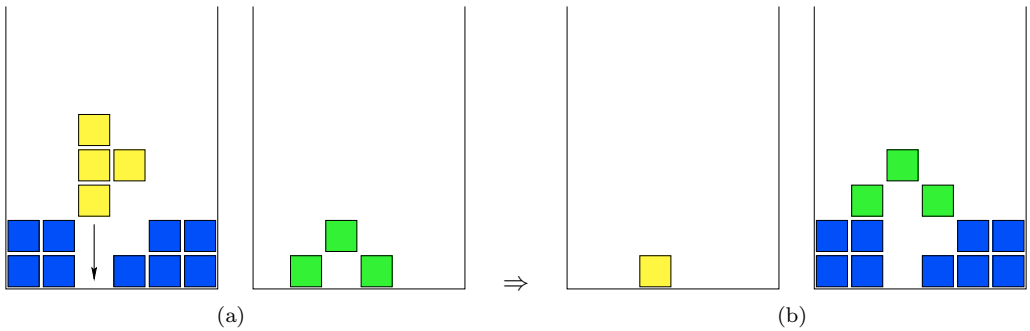

Figure 4: When a line is eliminated, the line is sent to the bottom of the opponent's window without the most recently filled square or squares. In this example, two lines are eliminated when the rotated `T` piece falls. The two lines are sent to the right player.

excessive amount of commands, the server enforced a 20-millisecond limit. If a command was sent within 20 milliseconds after another command, the later command was discarded. A common strategy for a human player would try to minimize the following numbers (1) the maximum height, (2) the number of holes, and (3) the variations of heights. Meanwhile, a player would try to maximize the number of lines eliminated. One difficulty is determining the relative weights of these factors. We invited a professor whose reach area specialized in artificial intelligence to explain game-playing strategies used in computers.

6 Evaluation

Judging from the students' codes, documentations and course feedbacks, we found that through this four-stage group programming assignment, many students demonstrated strong capability in writing network-enabled programs, designing effective and efficient algorithms for artificial intelligence and team collaboration.

Some teams developed strategies beyond our expectation. In the final stage, winning teams must have better strategies for both network communication and artificial intelligence. Timing is a crucial factor in winning. For example, the champion developed an adaptive strategy to achieve a higher speed in dropping pieces and sending eliminated lines to the opponent. The team measured the response time from the server to determine how soon the next command could be sent. This approach is actually similar to TCP Tahoe, a congestion control mechanism that is widely used in the Internet. Some groups developed computer players that could horizontally slide falling pieces to fill overhang holes. Sliding moves are unavailable in most existing Tetris programs; sliding is particularly difficult in a networked game. If the command is sent too early, the command is ignored due to collision. If the command is sent too late, the falling piece has landed and is no longer moveable. Several teams developed different algorithms and made them compete to select the better algorithms. One team designed an algorithm with several tuning parameters whose values were automatically improved through feedback using self competition. Meanwhile, some students improved their algorithms by "trial-and-error" without systematic approaches. Some students were fluent in applying linear algebra in rotations even though some others used unnecessarily complex algorithms due to the lack of mathematical skills in matrix manipulations. Some students showed appreciation of the close relationships between generating Tetris pieces and discrete mathematics.

Students' feedback suggested that most students were strongly motivated to learn many different concepts related to Tetris, from linear algebra to networking, from artificial intelligence to feedback tuning. They also learned teamwork and team dynamics. The students wrote comments such as "The class was hard, but I learned a lot." "Good course. I learned a lot and enjoyed developing games." "I wish I had taken this class earlier on." "This has been my favorite class and I have definitely learned the most." "Most of my interview questions have been supported by this class alone. I would recommend this class to any computer engineer as I believe it is one of the most important." Many teams created demonstration videos and posted them

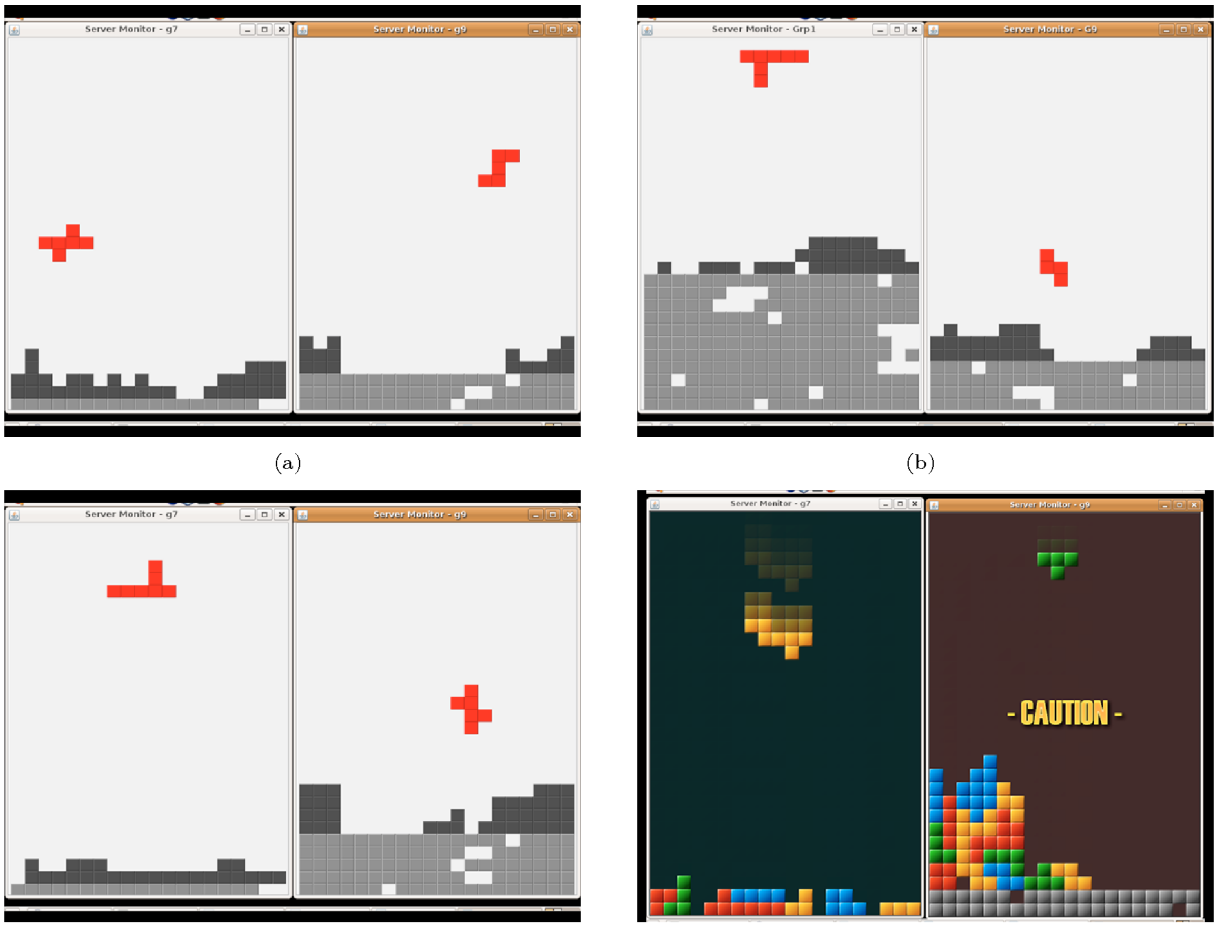

Figure 5: Screenshots of two-player Tetris competition. (a)-(c) The red pieces are falling. The light gray squares at the bottoms are eliminated lines from the opponents. (d) When a line from the opponent is added to the bottom, the window shows "Caution." The tail of a piece is added to provide visual effects.

on Youtube.

7 Conclusion

This paper presents a semester-long programming project using a variation of Tetris. The project is divided into four stages with clear goals for each stage. The project integrates the concepts covered in several courses. Through teamwork and competition, many students developed excellent programs that include graphical user interface, networking, and artificial intelligence.

References

[1] Committee on Technological Literacy. Technically Speaking: Why All Americans Need to Know More About Technology. http://www.nap.edu/catalog.php?record_id=10250, 2002.

[2] David L. Dill and Daniel Castro. The U.S. should ban paperless electronic voting machines. *Communications of the ACM*, 51(10):29–33, 2008.

[3] Jaikumar Vijayan. Will security concerns darken Google's government cloud? http://www.computerworld.com/, September 17 2009.

[4] IDG News Service. Fcc's national broadband plan: What's next? http://news.idg.no/cw/art.cfm?id=64A5457F-1A64-67EA-E4A95D435AFC2864, March 16 2010.

[5] Committee on Prospering in the Global Economy of the 21st Century. Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future. http://www.nap.edu/catalog.php?record_id=11463, 2007.

[6] Elliot Soloway. How the Nintendo Generation Learns. *Communications of the ACM*, 34(9):23–ff., 1991.

[7] Kate Sanders and Lynda Thomas. Checklists for Grading Object-Oriented CS1 Programs: Concepts and Misconceptions. In *SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 166–170, 2007.

[8] Erik D. Demaine, Susan Hohenberger, and David Liben-Nowell. *Tetris is Hard, Even to Approximate*, pages 351–363. Springer, 2003.

[9] Xingguo Chen, Hao Wang, Weiwei Wang, Yinghuan Shi, and Yang Gao. Apply ant colony optimization to Tetris. In *Annual Conference on Genetic and Evolutionary Computation*, pages 1741–1742, 2009.

[10] Carolina Cabral, Juana Dehanov, José Miguel Salles Dias, and Rafael Bastos. Developing games with Magic Playground: a gesture-based game engine. In *ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 361–362, 2005.

[11] Guillaume Chanel, Cyril Rebetez, Mireille Bétrancourt, and Thierry Pun. Boredom, Engagement and Anxiety as Indicators for Adaptation to Difficulty in Games. In *International Conference on Entertainment and Media in the Ubiquitous Era*, pages 13–17, 2008.

[12] Magy Seif El-Nasr and Brian K. Smith. Learning through Game modding. *Computer Entertainment*, 4(1):7, 2006.

[13] Jeff Sinclair, Philip Hingston, and Martin Masek. Considerations for the design of exergames. In *International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, pages 289–295, 2007.

[14] Niklas Ravaja, Mikko Salminen, Jussi Holopainen, Timo Saari, Jari Laarni, and Aki Järvinen. Emotional Response Patterns and Sense of Presence during Video Games: Potential Criterion Variables for Game Design. In *Nordic Conference on Human-Computer Interaction*, pages 339–347, 2004.

[15] Mark C. Lewis and Berna Massingill. Graphical Game Development in CS2: a Flexible Infrastructure for a Semester Long Project. In *SIGCSE Technical Symposium on Computer Science Education*, pages 505–509, 2006.