Development of Casual 2-D Game Laboratory Exercises in Introductory Computer Graphics Programming Course

Ge Jin
Purdue University, Calumet
ge.jin@purduecal.edu

Magesh Chandramouli
Purdue University, Calumet
mchandr@purduecal.edu

Abstract

Introductory programming is an important foundational course in computer science, information technology, computer graphics technology, and other computing-related disciplines. However, it is a challenging task for an instructor to raise and maintain student interest in programming at the freshman level programming course. This difficulty has resulted in high dropout rates in computing-related majors in higher education worldwide. In this paper, a series of hands on laboratory exercises are described, focusing on casual 2-D games to increase the students' interest in an introductory computer graphics programming course. The Pong game laboratory exercise was developed to teach the conditional statements and keyboard/ mouse interaction. The Tetris game laboratory exercise was designed to reinforce the usage of array data structure and boundary checking. The Maze game laboratory exercise was developed to convey the knowledge of recursive programming. In addition, the Angry Bird game laboratory exercise was designed to teach programming using open source libraries. Students have shown great interest in these casual game laboratory exercises and provided positive feedback in the end-of-semester course evaluations.

Introduction

Teaching a foundational programming course is a challenging task. The difficulty of passing a programming course at the freshman level has resulted in high dropout rates in computer science, information technology, computer graphics technology, and other computing-related majors in higher education worldwide [1]. Computing educators have stated that learning programming languages and acquiring problem-solving skills are time-consuming and difficult tasks [2, 3]. One important reason is that the current programming languages are mainly designed for industrial use and hence are too complicated to teach programming foundations. Jenkins [4] indicated that the programming language for an introductory course should be chosen for an educational purpose rather than popularity in industry.

Several key components influencing student motivation must be considered when discussing programming instruction and learning. Driscoll [5] stated that instructional material must appeal to learners and must motivate learners in their goal achievement. Keller and Litchfield claimed that considering student motivation “is particularly important because it pertains to a person’s basic decisions as to whether or not to accept responsibility for a task and to pursue a given goal” [6]. As applied to instruction, Talton and Fitzpatrick noted, “A long-standing difficulty in the development of introductory courses in computer graphics is balancing the educational necessity of ensuring mastery of fundamental graphics concepts with the highly desirable goal of exciting and inspiring students to further study by enabling them to produce visually interesting programming projects” [7].

Many researchers in education have shown that the use of instructional methods other than the traditional lecture format is much more effective in facilitating student learning. Methods such as active learning [8], problem solving [9, 10], and project-based learning [11, 12] are encouraged as ways of exciting students, providing real-world, problem-solving experiences, and increasing transfer of critical skill sets from the classroom to the workplace, especially in introductory programming instruction [13] and other technology-based learning [14, 15]. To allow students to relate new learning to existing skills/knowledge without cognitive overload, teaching in technology environments should include as much contextual content as possible.

A freshman programming course is an important foundational course in computer science and graphics education. Educators in these disciplines have proposed various ways to enhance the learning of programming concepts in introductory programming courses. Recently, Hernandez et al. taught fundamental programming principles to freshmen students through a 2-D Game Engine: GameMaker [16]. Game Engine allows the students to understand fundamental programming principles without having to learn the complex syntax of programming language. Similarly, Kazimoglu et al. felt that digital game play could be an effective method for computational thinking programming instruction [17]. Additionally, Papastergiou showed that a digital gaming method of learning was both more motivational and more successful than traditional non-gaming methods in teaching computer memory basics to high school students [18]. However, Holzinger et al. cautioned that although dynamic media can be an effective learning tool, excessive use of such methods could lead to cognitive overload if not utilized judiciously [19].

The Computer Graphics Technology (CGT) program at Purdue University, Calumet, does not include many programming courses at the freshman and sophomore levels. To better prepare the students to meet industry requirements and to enhance their employability by sharpening their programming skills, it is essential to teach introductory graphics programming course(s) at the first/second year. The undergraduate computer graphics technology program at Purdue University, Calumet, focuses on areas such as multimedia design, web design and development, computer animation, game development and graphic design. Formerly, CGT students would take programming courses (Java or C++) from other academic departments, but these often would be too discipline-specific to meet the needs of the CGT students. To resolve this issue, a CGT-specific computer graphics programming course was developed. Since CGT students tend to be visually oriented and are exposed to various graphics design and game development tools, it was beneficial to teach fundamental programming concepts using simple 2-D game laboratory exercises. It was also believed that the 2-D casual game laboratory exercises would serve the dual purpose of enhancing general programming concepts and inculcating computer game development skills.

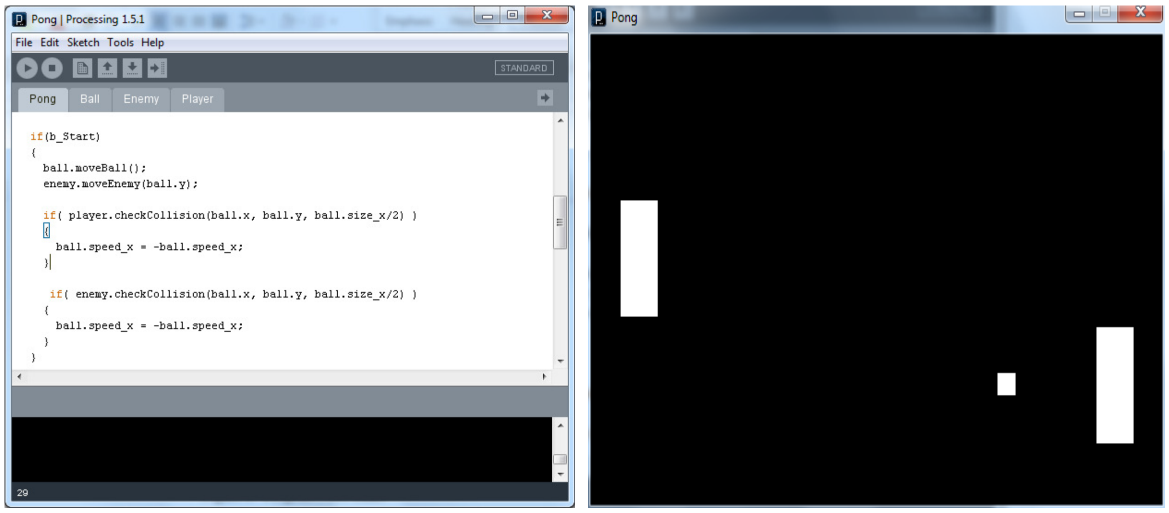
This paper proposes the use of casual 2-D game laboratory exercises to raise the students' interest in introductory programming course. Four casual 2-D games—Pong, Tetris, Maze, and Angry Birds—were chosen to reinforce the conditional statement, array, recursion, and API programming concepts respectively. Visual programming exercises can quickly communicate programming concepts, especially for students with visual learning style preferences [20-22]. Overall, the results suggest that the visual and casual 2-D game laboratory has helped students learn complex concepts. An open source programming environment called Processing was chosen to design and implement laboratory exercises in an introductory graphics programming course.

Methods

Some programming concepts are generic and span across different languages. These include data, variables, scope, functions, loops, iteration, conditionals, recursion, etc. Illustrations can quickly communicate programming concepts, especially for students with visual learning style preferences [20-22]. Besides these basic programming elements, students need to know a very important programming paradigm, object-oriented programming (OOP). Extremely successful programming languages like Java and C++ are based on OOP. The casual game laboratory exercises would focus on one or more fundamental programming concepts.

Development of Pong Game Laboratory Exercise

One of the fundamental programming concepts is conditional statement. Graphics examples may help students to transcend the abstract level and take the learning process to a more concrete level. This is because a visual example allows students to actually see the result of conditional statement via the graphic output window. Pong is one of the first arcade games created in 1970s. In this game, the "ball" will bounce back from the side walls until a player scores a point. The logic behind the ball bouncing off the sidewalls is essentially a conditional statement. If the vertical position of the "ball" is less than 0 (the upper boundary of screen) or higher than the screen height (the lower boundary of screen), the ball will change its speed along a vertical direction. Without the conditional statement, the ball will disappear from the upper or the lower boundary of the screen. In this way, the students can immediately interpret the result of conditional statement in a visual fashion. This logic of this conditional statement is illustrated in Figure 1, left image. The similar logic of conditional statements will be applied to the collision between the "ball" and the player paddle, as well as the opponent's paddle.



```
if(b_Start)
{
  ball.moveBall();
  enemy.moveEnemy(ball.y);

  if( player.checkCollision(ball.x, ball.y, ball.size_x/2) )
  {
    ball.speed_x = -ball.speed_x;
  }

   if( enemy.checkCollision(ball.x, ball.y, ball.size_x/2) )
  {
    ball.speed_x = -ball.speed_x;
  }
}
```

Figure 1. Pong game laboratory exercise for conditional statement

Development of Tetris Game Laboratory Exercise

The concept of variable is the first bottleneck for freshman programing course. Simply stated, a variable is a data-holder or a computer memory used to hold data. Just as in the real-world different containers are used to hold different types of materials, different variables are used to store different types of data. A more advanced data concept for an introductory programming course is array. There are several issues when students start to use array data structure. The first issue is array index out of bound, where the valid array index ranges from 0 to array size minus 1. The second issue is mapping a real-world problem to an array data structure.

The first example we used to map real world problem to an array data structure is simple a Tic-Tac-Toe game. In this demo example, we created a 3 x 3 integer array to represent a Tic-Tac-Toe map. This example demonstrates that a traditional board game can be represented by a simple array structure and illustrates the importance of separating the underlying data structure (3 x 3 integer array) from the graphical displays. The Tic-Tac-Toe game requires the students apply graphical drawing functions, mouse input functions, object-oriented design, and array data structure to solve the problem. Below are several important tasks that student need to complete:

- A class is designed to represent the Tic-Tac-Toe map. A data member of a 3 x 3 integer array is to represent the map. Initially the array is initialized with a value of 0.
- In the main function, a 3 x 3 grid is drawn on the display screen. For each grid, it checks the value of the Tic-Tac-Toe array. If the array element value is 0, nothing was drawn; if it is 1, we draw a circle, and if it is -1, we draw an "X" symbol.
- If the player clicks on the display screen, the mouse click position will be mapped to the array index, and the Tic-Tac-Toe array element value will be set to 1.

- The computer will check the remaining array elements, choose the best place, and set the array element value to -1.
- Whenever a player or computer sets an array element value, an evaluation routine will check the three consecutive values along horizontal, vertical, and diagonal directions to determine the winner.

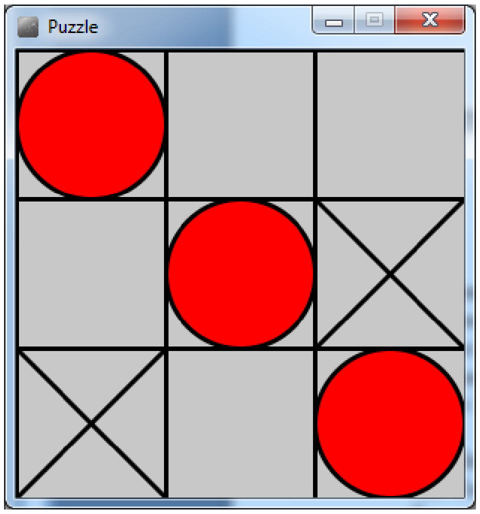


Figure 2. Tic-Tac-Toe game for array concept

A more advanced laboratory exercise is Tetris, a famous 2-D game that was developed by two Russian programmers. After completing the simple Tic-Tac-Toe game, students are able to map the Tetris game to array data structures. The display screen is mapped to a 10 x 20 integer array. The Tetris block is mapped to a 4-element integer array. The Tetris block is moving inside a 10 x 20 map to see if the block touches the lower boundary of the map or any block previously occupied the map grid. The rotation of the block is essentially changing the 4-element array value of Tetris block object. Below are several important tasks that student need to complete:

- A Tetris map class is designed to represent the display screen of the Tetris game. A data member of 10 x 20 integer array is to represent the map. Initially, the array is initialized with value of 0.
- A Tetris block class is designed to represent the block of the Tetris game. A data member of a 4 x 2 integer array is to represent the shape of the Tetris block. The initial value of the shape array is determined by the type of the block.
- In the main function, a 10 x 20 grid is drawn on the display screen. For each grid, it checks the value of the Tetris map array. If the array element value is 0, nothing was drawn; if it is 1, a solid rectangle is drawn on the grid, location indicating a block has occupied that grid.
- At any time, there is an active block continuously moving down; if the player pressed a left or right arrow key, the block move left or right. If the player pressed the up or down key, the block rotates clockwise or counterclockwise to change the shape.

- After each movement of the block, the computer will check if the block touches the lower boundary of the map or any block previously occupied the map grid. If a collision is detected, the computer will check if a horizontal line is occupied by blocks. If so, that grid line is removed from the map array.
- Next, a new block object is created and repeats the game play. And if the block occupied the top grid line, the game ends.

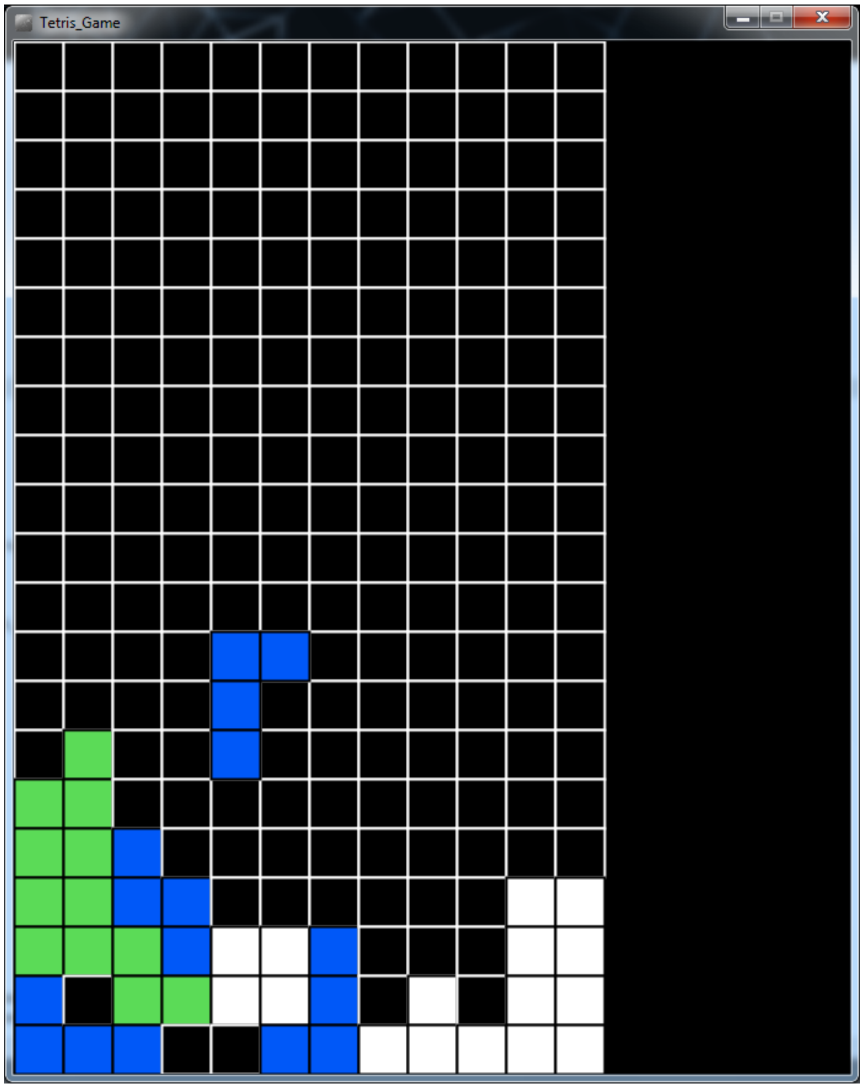


Figure 3. Tetris game laboratory exercise for advanced array concept

Development of Maze Game Laboratory Exercise

One important concept in programming is recursion. Solving a Maze game requires the knowledge of array data structure and recursive programming skills. In the Maze game laboratory exercise, the maze is represented by two-dimensional integer array. A value of 1 in the array element indicates the wall, and a value of 0 indicates the path. A recursive function is created to find out one of the maze path from the entrance to the exit. The recursive function works as follows:

- If the current location is the exit point, the recursive function returns with success, which means we find the solution.
- If the current location is not an exit point, we mark the current location as visited by using value of 2; then we continue to look for the 4 neighboring array elements. If we find any one neighboring array with a value of 0, we recursively call the solve maze function.
- If no valid path exists, we mark the current cell as a dead end, by using value of -1, and return.
- Once the recursive function returns with success, the path from the entrance to the exit will be displayed by marking all the array elements with value of 2.

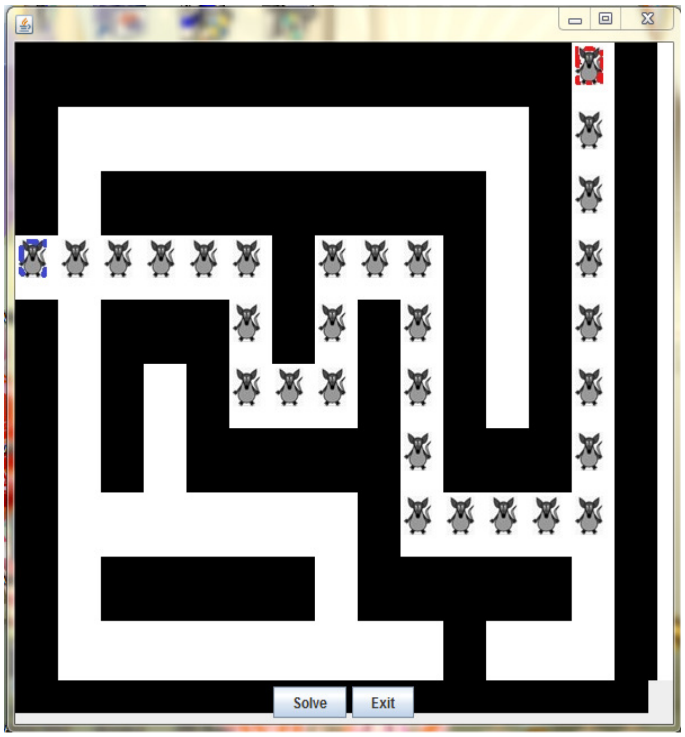


Figure 4. Maze game laboratory exercise for recursion

Development of Angry Birds Game Laboratory Exercise

At the final project period of the introductory programming course, we decided to teach the students how to use open source libraries to create a prototype Angry Birds game. We decided to use Fisca library, a wrapped JBox2-D java library, for the processing. Both the Angry Birds class and the Pig class extend the FCircle class of the Fisca library. The wood blocks extend the FBox class. The actual shape of the angry bird, pigs, and wood blocks are represented using images. The pigs and wood blocks were set as passive dynamic objects, and angry bird object was set as active dynamic objects. In addition, the Angry Birds class has methods to deal with the mouse input. Once the user selected the angry bird and dragged to a shooting position, initial force and direction were calculated and applied to the bird. The bird will fly to the wood blocks and pigs to interact with them.

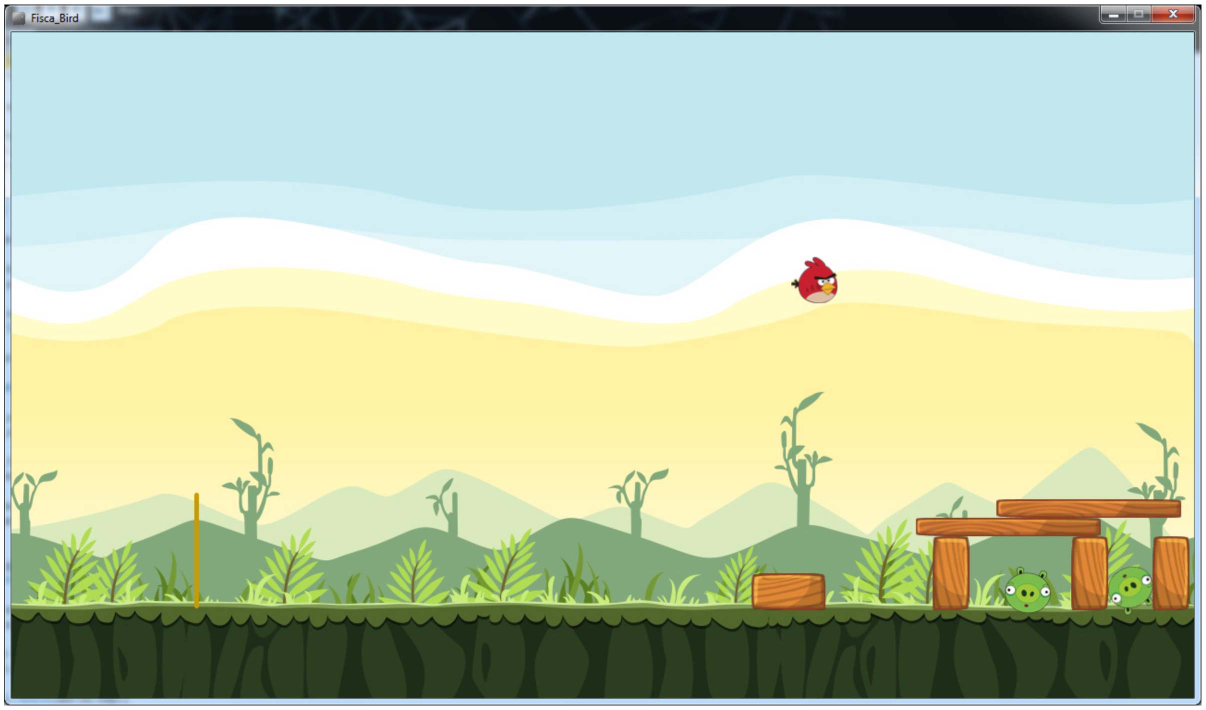


Figure 5. Angry Birds game laboratory exercise for open source library

Results and Discussion

These 2-D game laboratory exercises were designed to cover essential programming concepts as well as advanced topics such as array, recursion, and open source library. The exams/quizzes and labs/projects were designed to evaluate the students' learning outcomes. Midterm and final exams consisted of 100 questions that tested student understanding of graphics programming concepts. Table 1 illustrates the number of questions in each sub-category and the students' performance in the corresponding sub-category. The exam result indicates that students performed extremely well in understanding 2-D graphics concepts. The results also show that approximately 80% of the students correctly answered basic programming questions. These questions covered concepts such as variable, conditional statement, function, recursion, array, OOP, and event-driven programming. Overall, the results suggest that the 2-D causal game laboratory exercises have helped students learn complex concepts.

Table 1. Question distribution and student success rates in midterm/final exam

	Midterm (50 Questions) and Final (50 Questions)	
	Number of Question	Success Rate
Variable	10	79.6%
Conditional Statement	12	86.6%
Loop	6	89.2%
Array	9	78.9%
Functions	14	72.2%
Graphics Transformation	9	95.1%
Object Oriented Programming	14	84.5%
User Interaction(Keyboard/Mouse)	12	89.8%
Images/Text	5	76.7%
Processing Specific Functions	9	89.5 %
Total	100	83.9%

Two types of end-of-semester course evaluations were conducted: course learning outcomes and instructor evaluation. The course learning outcomes are presented in Table 2. To further confirm the positive results demonstrated in this section, the following question was included in the final instructor evaluation: "I am glad that I took a graphics programming course in my freshman year": 50% of the students replied "Strongly Agree" and 50% selected "Agree." This confirms that students are receptive to the idea of learning CG concepts using a programming medium.

Table 2. Question distribution and student success rates in midterm/final exam

Survey Question (1-Lowest, 5-Highest)	Score
1. I learned basic programming concepts such as variables, data types, selection and repetition control structures, arrays, files, and methods and functions from this course.	4.375
2. After taking this course, I can create simple object-oriented applications.	4.19
3. After this course, I can understand the similarities between Processing and Java.	3.875
4. After this course, I can apply knowledge to different computer graphics software applications.	4.3125

Conclusion

This paper proposed using a series of 2-D casual game laboratory exercises in first year undergraduate programming courses. The study explored the feasibility of attracting students to the programming laboratory tasks with 2-D game topics. With these exercises, students can create their own simple games with fundamental programming concepts such as conditional statements, array, recursion, and open source library. In this paper, a series of hands on laboratory exercises were described, focusing on casual 2-D games to increase the students' interest in an introductory computer graphics programming course. The Pong game laboratory exercise was developed to teach the conditional statements and keyboard/mouse interaction. The Tetris exercise was designed to reinforce the usage of array data structure and boundary checking. A Maze game laboratory exercise was developed to convey the knowledge of recursive programming. In addition, the Angry Birds exercise was designed to teach programming using open source libraries. The paper addressed the issue of reducing "cognitive overload" by minimizing perplexing jargon or complex programming terminology and using graphics illustrations as an effective means to communicate programming notions. With casual game laboratory exercises, students could actually see the visual result via the graphic output window, which facilitated understanding important programming concepts. Students had shown great interests in these casual game laboratory exercises and provided positive feedback in the end-of-semester course evaluations.

References

[1] Kinnunen, P., & Malmi, L. (2006). Why Students Drop Out CS1 Course? *Proceedings of the 2006 International Workshop on Computing Education Research*, 97-108.

[2] Lahtinen, E. Ala-Mutak, K., & Jarvinen, H. (2005). A Study of the Difficulties of Novice Programmers. *ACMSIGCSE Bulletin*, *37*(3), 14-18.

[3] Gomes, A., & Mendes, A. J. (2007) Learning to Program—Difficulties and Solutions. *Proceedings of the International Conference on Engineering Education*, 283-287.

[4] Jenkins, T. (2002). On the Difficulty of Learning to Program. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 53-38.

[5] Driscoll, M. P. (2005). *Psychology of Learning for Instruction*. (3rd ed.). Needham Heights, MA: Allyn and Bacon.

[6] Keller, J. M., & Litchfield, B. C. (2002). Motivation and Performance. In R. A. Reiser & J. V. Dempsey (Eds.), *Trends and Issues in Instructional Design and Technology* (pp. 83-98). Upper Saddle River, NJ: Pearson Education.

[7] Talton, J. O., & Fitzpatrick, D. (2007). Teaching Graphics with the Opengl Shading Language. *ACM SIGCSE Bulletin*, *39*(1), 259-263. doi: 10.114.

[8] Prince, M. J., & Felder, R. M. (2006). Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. *Journal of Engineering Education*, *95*(2), 123-138.

[9] Jonassen, D. H. (2002). Integration of Problem Solving into Instructional Design. In R. A. Reiser & J. V. Dempsey (Eds.), *Trends and Issues in Instructional Design and Technology* (107-120). Upper Saddle River, NJ: Pearson Education.

[10] Newby, T. J., Stepich, D. A., Lehman, J. D., Russell, J. D., & Leftwich, A. T. (2010). *Educational Technology for Teaching and Learning*. (4th ed.). Upper Saddle River, NJ: Pearson Education.

[11] Hadim, H. A., & Esche, S. K. (2002). Enhancing the Engineering Curriculum through Project-Based Learning. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*, Boston.

[12] Mills, J. E., & Treagust, D. F. (2003). Engineering Education—Is Problem-Based or Project-Based Learning the Answer? *Australasian Journal of Engineering Education*, 2-16. Retrieved from http://www.aaee.com.au/journal/2003/mills_treagust03.pdf

[13] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A Survey of Literature on the Teaching on Introductory Programming. *ACM SIGCSE Bulletin*, *39(*4), 204-223. doi: 10.1145/1345375.1345441.

[14] Jonassen, D. H., Howland, J, Moore, J., & Marra, R. M. (2003). *Learning to Solve Problems with Technology: A Constructivist Perspective*. (2nd ed.). Upper Saddle River, NJ: Pearson Education.
[15] Roblyer, M. D. (2004). *Integrating Educational Technology into Teaching*. (3rd ed.). Upper Saddle River, NJ: Pearson Education.
[16] Hernandez, C. C., Silva, L., Segura, R. A., Schimiguel, J., Paradela Ledón, M. F., Bezerra, L. M., & Silveira, I. F. (2010). Teaching Programming Principles through a Game Engine. *Clei Electronic Journal*, *13*(2), 3.
[17] Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning Programming at the Computational Thinking Level via Digital Game Play. *Procedia Computer Science*, *9,* 522-531.
[18] Papastergiou, M. (2009). Digital Game-Based Learning in High School Computer Science Education: Impact on Educational Effectiveness and Student Motivation. *Computers & Education*, *52*, 1-12.
[19] Holzinger, A., Kickmeier-Rust, M., & Albert, D. (2008). Dynamic Media in Computer Science Education; Content Complexity and Learning Performance: Is Less More? *Educational Technology & Society*, *11*(1), 279-290.
[20] Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning Styles and Performance in the Introductory Programming Sequence. *ACMSIGCSE Bulletin*, *34*(1), 33-37.
[21] Lewalter, D. (2003). Cognitive Strategies for Learning from Static and Dynamic Visuals. *Learning and Instruction*, *13*, 177-189.
[22] Zualkernan, I. A., Allert, J., & Qadah, G. Z. (2006). Learning Styles of Computer Programming Students: A Middle Eastern and American Comparison. *IEEE Transactions on Education*, *49*(4), 443-450.

Biographies

GE JIN, D.Sc, is currently an assistant professor in the Department of Computer Information Technology and Graphics at the Purdue University Calumet. He teaches computer game development, computer graphics, and animation, as well as computer information technology courses at the undergraduate and graduate levels. Prior to joining Purdue University, Calumet, he was a postdoctoral research scientist at the GeorgeWashington University, Department of Computer Science. Professor Jin holds a B.S. in Computer Science from Peking University, China, and an M.S. in Computer Science from Seoul National University, South Korea. He earned his Doctor of Science degree in Computer Science with a concentration in computer graphics from the George Washington University. His research spans the fields of computer graphics, virtual reality, computer animation, medical visualization, and educational game development. He is a member of the ACM SIGGRAPH, ASEE, and International Society of Virtual Rehabilitation.

MAGESH CHANDRAMOULI, Ph.D, is an assistant professor of Computer Information Technology and Graphics at the Purdue University, Calumet. He earned his B.E. degree from

the College of Engineering, Anna University, India, and an M.Eng from the National University of Singapore. He earned his Master of Science from the University of Calgary, Canada, and Ph.D. from Purdue University. His research interests include computer graphics, 3-D visualization, multi-objective optimization, genetic algorithms, virtual reality, and computer animation. Dr. Chandramouli may be reached at mchandr@purduecal.edu.