A Community College Blended Learning Classroom Experience through Artificial Intelligence in Games

Titus Barik*†, Michael Everett†, Rogelio E. Cardona-Rivera*, David L. Roberts*, Edward F. Gehringer*
*Department of Computer Science, North Carolina State University
{tbarik, recardon}@ncsu.edu, robertsd@csc.ncsu.edu, efg@ncsu.edu
†Wayne Community College, Goldsboro, NC, michael.everett@sas.com

***Abstract*—We report on the experience of teaching an industry-validated course on Artificial Intelligence in Computer Games within the Simulation and Game Design department at a two-year community college during a 16-week semester. The course format used a blended learning just-in-time teaching approach, which included active learning programming exercises and one-on-one student interactions. Moskal's Attitudes Toward Computer Science survey showed a positive and significant increase in students in both interest (W(10) = 25, p = 0.011) and professional (W(10) = 49.5, p = 0.037) constructs. The Felder-Soloman Index of Learning Styles (n = 14) failed to identify any statistically significant differences in learning styles when compared to a four-year CS1 class. In the final class evaluation, 8 out of 13 students (62%) strongly or very strongly preferred the blended learning approach. We validated this course through four semi-structured interviews with game companies. The interview results suggest that companies are strongly favorable to the course content and structure. The results of this work serve as a template that community colleges can adopt for their curriculum.**

I. Introduction

Community college students face a unique set of challenges that differ in many ways from those of traditional four-year University college students. A significant number of community college students enter their programs lacking study skills and the academic maturity needed to handle more rigorous, self-directed approaches used in traditional colleges [1]. Many of these students have responsibilities outside of school, placing further demands on their time and financial resources. For these students, community colleges must serve as a critical bridge between academia and industry and provide them with a practical and industry-focused education that prepares them for the workforce. At the end of a two-year program, students are expected to have sufficient training so that they can be placed into entry-level positions in their respective fields immediately after graduation. Given these student and instructional differences, we believe that educators should not tacitly assume that pedagogical approaches and experiences that have been successful in traditional institutions can be applied with equal success at the community college level; these approaches must be once again validated in a two-year setting.

Students in community colleges typically restrict themselves to a focused curriculum that emphasizes skills practical for industry. As such, community colleges must make continual efforts to ensure that their courses are relevant and useful to the industry jobs students are seeking. This task cannot be accomplished in isolation, and can only be successful when academia and industry collaborate in curriculum design. We argue that this collaboration must occur not only at the program level, but also at the individual course level, so that each course itself is validated as being appropriate to the types of skills that industry demands.

It is with this understanding that we report on the experience of teaching an elective, pilot course on Artificial Intelligence (AI) in Computer Games within the Simulation and Game Design curriculum at a two-year community college during the 16-week Fall 2012 semester. An advisory committee comprised of both academic and industry members identified the following three criteria for success: 1) to evaluate if teaching computational thinking through game design can positively increase attitudes about Computer Science; 2) to assess student reactions to a blended learning classroom experience; and 3) to validate whether the learning objectives of the course are relevant and useful to industry.

The course format used a blended learning approach, which consisted of discussions, in-class active learning programming exercises, and one-on-one student interactions with the instructor.[1] We minimized the use of formal lecture; when necessary, we delivered lecture content using a just-in-time teaching approach [2] to address observed student difficulties. The course content can be summarized as five weeks of learning the Python programming language through a game context that emphasized algorithmic thinking, with the remaining semester time spent on AI for games topics.

To evaluate instructional compatibility with pedagogical techniques used in four-year institutions, students completed the Felder-Soloman Index of Learning Styles (ILS) questionnaire [3] ($n = 14$). To identify changes in student attitudes, students completed the Moskal's Attitudes Toward Computer Science survey [4] ($n = 10$) twice during the semester. To determine whether our blended learning classroom approach was well-received, students completed a final evaluation ($n = 13$) with 5-point Likert-item questions. Finally, to validate our course, we performed semi-structured interviews with four local game companies.

The results of this work serve as a template for community colleges considering the adoption of specific pedagogical approaches in the classroom or for colleges who wish to adopt a prepared, industry-validated AI course in its entirety. The results also highlight the effectiveness of using a blended learning approach at the community college level.

[1] We recognize that the definition of blended learning varies widely in the education community. We are less interested in constructing a strict definition, and far more interested in whether or not the techniques are useful to students.

1. Create a class called `Marine`, using the `Tank` class as a starting point. The `Marine` class should have the following properties: `name`, `position`, `armor`, and `damage`. That is, the programmer should be able to initialize the marine by performing the following:

```
m = Marine("Joe", (3, 2), 2, 5)
```

Next, add a `__str__` method to this class so that when `m` is printed, the following is output:

```
>>> print m
Joe has 2 armor and 5 damage.
He is located at (3, 2).
```

Submit the file `marine.py`.

(a) Pre-lab

3. In this section, we will work with the `PredatorTank`, which is a variation of the `Tank` class as discussed in the text.
   (a) Create a class called `PredatorTank`. It should have an initialization method (`__init__`) that takes in a `name`, `health`, `attack`, `position`, and `range`.

(b) In-class Exercise

4. Create an air unit called `Bomber`. A bomber has the following properties: `name`, `position`, `health`, `attack`, `firing_range`, `accuracy`, and `air`. Since a bomber is an air unit, `air` should always be set to `True` and does not have to be explicitly specified when creating the Bomber.

(c) Homework

Fig. 1. A representative example of how class assignments carry from pre-lab to homework.

II. Course Syllabus

This 16-week course introduces artificial intelligence concepts with an applied focus in video game development and has been designed to be a part of a two-year community college program. The course is intended to be taught in a lab setting where each student has access to a computer workstation. Upon completion of the course, students should be able to understand the basic concepts, and implement algorithms, for AI in games.

The objective of the course was to teach computational thinking through game programming, so that students could relate theoretical classroom concepts to their existing experiences with playing video games [5]. A secondary objective was to design a modular course structure that can serve as a starting point for other community colleges. In this section, we discuss the specifics of the course format as well as the course topics. The class met once a week for 3 hours in the evening. However, the effective class time, after accounting for three 10-15 minute breaks as well as administrative overhead, was reduced to approximately two hours.

A. Course Format

Grounded in Revised Bloom's Taxonomy (RBT) [6], we adopted a just-in-time teaching approach [2] in our course, with some variations: we had less emphasis on web-based materials, integrated the classroom and laboratory sessions, and implemented blended learning by having computer programming tasks as the focal point for all activities. Students completed pre-lab exercises before class, completed active learning sheets in-class, and had a homework assessment on the related material after class.

Our pre-lab exercises were less exploratory. Instead, the exercises explicitly tasked students with surveying the text to remember and understand course material for the associated class session. Because of this, they were graded on a pass/fail basis. For example, the task in Fig. 1a. asks the students to modify an existing piece of code available in their textbook by changing a `Tank` to a `Marine`. To successfully complete the task, the student must rename the file, rename some of the properties, and change a single statement that prints these properties to the screen.

The majority of in-class time was spent completing exercises for the topic, with minimal lecturing from the course instructor [7]. Through the use of modified[2] active learning sheets [8], students completed in-class exercises that built upon the completed pre-lab exercises. Students were offered an opportunity to apply their pre-lab knowledge to new problem contexts, without introducing new material. Students would first attempt to solve the exercises without instructor assistance. During this time, we walked around the room to give individual attention to students as they solved the assigned problems. This approach was essentially an early-alert system that allowed us to quickly assess student understanding of the material. After 5-15 minutes, depending on exercise complexity, we regrouped as a class and together worked toward obtaining a correct solution. With the guidance of the instructor, students analyzed the problems to gain insight into the relationship between pre-lab and in-class material. Here, we provided the rationale for the problem solutions, analyzed tradeoffs and alternative approaches, and discussed how these materials might be applied to games they have played in the past. Continuing the previous example, for the task in Fig. 1b we instructed students to implement an initialization method based on their pre-lab, and performed the aforementioned tasks to explain how this method relates to the programming task as a whole.

After class, students were given a homework assignment that was a continuation of the class topics. The homeworks provided an opportunity to combine the concepts from class in a novel way and in more depth. Because of increased difficulty of homeworks, students were encouraged to work together, but had to submit work individually. Finalizing the ongoing example, for the task in Fig. 1c, we asked students to extend the behavior demonstrated in class by having them create unit types that support both air and ground modes. This required students to modify their existing Tank firing logic because of a constraint that Tanks can only fire on ground units (not shown).

[2]Instead of fill-in-the-blank type responses, our question prompts required students to supply short answers. This style was appropriate to our course format because we did not deliver active learning sheets within a normal lecture as Lau [8].

TABLE I. INDUSTRY VALIDATED SYLLABUS FOR A 16-WEEK SEMESTER AI IN GAMES COURSE

Module	Topic	Learning Objectives
1	Introducing Python	Basic Python syntax and semantics, such as numbers, strings, variables, conditionals, loops, and functions. The list, tuple, and dictionary data structures.
2	Exploring Python	Additional exercises on concepts from the previous module. Classes as a means to organize data. The game loop as building block for all games. Basic random number generation as a means for adding variation in games.
3	Introducing Pygame	Review of classes. Loading sprites and backgrounds; blitting (drawing) objects to the screen. Event handling and keyboard logic.
4-5	Numerical Python	Review of list operations, importing libraries. Converting lists and tuples to `NumPy` arrays. Fundamentals of Newtonian Physics, which include displacement, velocity, and acceleration. Vectors, including Cartesian coordinate systems, plotting, dot products, Euclidean distance, and vector normalization.
6	Movement Algorithms	Review of integer and floating point operations and subtle pitfalls. Additional practice on vector operations implemented using simplified Newtonian Physics model from the previous modules. Game loop updates for updating avatar position. Frame vs. time updates. Seek movement algorithm as a fundamental AI primitive.
7	Pathfinding I	Double-ended queues, and algorithmic costs and tradeoffs for list and double-ended queue operations. Data structures, e.g., stacks and queues implemented using double-ended queues. Static paths, as found older games, implemented through list rotations. Tile graphs as the basic building blocks of two-dimensional world representation.
8	Pathfinding II	Breath-first search implementation using nested lists as a world representation. A simplified edge finding function for locating adjacent nodes. Discussion of generalized search with explored and frontier lists, and choice of distance functions and their effects. Mathematical operations for quantization (translating graph nodes to pixel coordinates and back).
9	Decision Making I	Review of dictionaries and how they can be used to implement Blackboard architectures. Hard-coded decision trees as sequences of conditional statements. Five-state decision tree system involving AI that can fire on the opponent, seek the opponent, find ammo items, find health, and panic when it is out of ammo and low on health.
10	*Practical Midterm Exam*	
11	Decision Making II	Review of Manhattan distance. Multiple mechanisms (references) for accessing unique objects in the context of dictionaries. Visual comparison of decision trees and finite state machines (FSMs). Implementing FSMs in Python using object-oriented programming. Methods as a means to add behavior to classes. FSMs as a technique to modularize AI behavior. Implementation of a schoolyard game of tag through FSM behavior.
12	Learning I	Review of world representations and blackboard architectures. Review of random number generators as a means to emulate intelligence. Artificial stupidity; how games cheat to give the perception of intelligence (for example, fog of war). Implementing finite state machines in a hide and seek game setting.
13	Supporting Technologies I	Technologies that are not directly game AI, but help support games in general. Recasting AI algorithms and demonstrating applicability across domains. Finite state machines as an example of maintaining user states in a chat system. Event-based management in networks, as opposed to games, through the `Twisted` framework.
14	Supporting Technologies II	Continuation of previous module; modifying networking code to implement a text-based online game. Discussed the limitations of AI, and role and need for multiplayer games. The use of AI agents that can act as players.
15	Comprehensive Final Review	Wrap-up of course topics in an interview context. Main take aways and course highlights at a high level. Interviewing techniques for industry positions in entry-level game AI. Application of course topics to general programming careers.
16	*Practical Final Exam*	

Students were evaluated using a midterm and final exam, which had notable differences from evaluations most students were familiar with. To simulate industry settings, all exams were open-book and open-notes, and we allowed the use of the Internet. More importantly, all problems were programming exercises that required students to submit source code within the 3-hour time window. Thus, no multiple choice or fill-in-the-blank questions were employed. Students were notified in advance of the specific subject matter to be evaluated. In much the same way that in-class exercises were built on pre-lab exercises, the exams provided an opportunity to independently demonstrate learning from the in-class exercises. To our knowledge, this exam style is still not widely used in computer science; indeed, at the community college where this course was taught and evaluated, it was the first within the Information Systems department to implement these techniques.

Though many languages have been used for teaching introductory programming [9], we chose Python because of previous successes reported by four-year Universities and high schools, and we expected these reported benefits to apply equally well at the community college level [10], [11]. As presented by Grendel et al. [11], we also considered Python's minimal syntax, dynamic typing, expressive built-in types, and immediate feedback to be positive factors when considering its adoption. Most importantly, Python has an orthogonal design that allowed us to simplify the language, such that advanced language features, including decorators, list comprehensions, and operator overloading, were omitted from the course entirely.

To support game-specific features, the following additional libraries were required: Pygame,[3] for abstracting low-level game engine details (such as graphics, user input, and event management); NumPy,[4] for vector operators; and Twisted,[5] for event-driven networking support. Students used the built-in IDLE editor to write their Python programs, using simple `print` statements when debugging was necessary. All of the software used in the class is open source and therefore freely available to the students for use at home. In short, we offered a general-purpose programming language that was loosely coupled with the underlying game engine library so that the concepts learned in the course could easily transfer to other programming environments.

[3] http://www.pygame.org
[4] http://www.numpy.org/
[5] http://twistedmatrix.com/

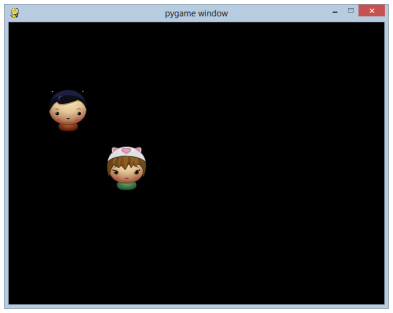

Fig. 2. Students experiment with avatar movement. The boy is controlled using the keyboard by the player and the girl is an AI agent.

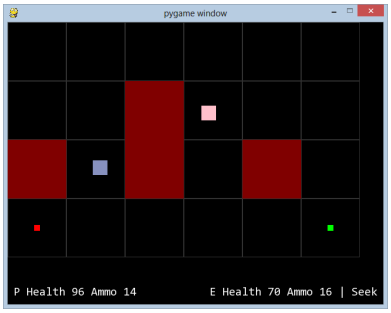

Fig. 3. Students implement a finite state machine in a tile-based world. Colored tiles are obstacles and hiding places for the player. Small red (bottom left) and green (bottom right) rectangles are ammo and health items, respectively, which provide opportunities for the AI to perform additional decision making.

B. Course Topics

The course used two textbooks. The first textbook was Python-specific and introduced the students to the Python programming language from a games perspective and the Pygame framework [12]. Throughout the course, this book was largely used as a reference. The second was an Artificial Intelligence (AI) textbook used in many four-year undergraduate programs [13]. Despite the fact that the AI text was not specifically designed for the community college level,[6] we hoped that its comprehensive coverage would allow students to continue pursuing their interests in game AI well after the completion of the class. As mentioned in Section II-A, class sessions assumed that students had read the text beforehand.

An overview of the topics can be found in Table I. We tried to strike a balance in course topics that satisfied the needs of the industry, but at the same time was appropriate for the capabilities of community college students. We presented traditional computer science algorithms through a game-focused lens. As one example, rather than offer a theoretical coverage of vectors as an abstract mathematical construct, we motivated the need for vector operations as a practical method to simplify AI movement in games programming. As another example, rather than directly cover the concepts of stacks and queues, we motivated the need for these data structures through their use in AI pathfinding, a staple of games. Throughout the course, we emphasized analyzing tradeoffs between different data structures, such as lists, tuples, and dictionaries. Nested lists were found to be most suited to representing a tile-based world; tuples were useful in representing coordinates; and dictionaries were useful in storing agent properties that needed to be accessed by a key (such as a player's health or ammo).

It was important to us that concepts in the course be cumulative because we felt that students could appreciate their efforts if they could tangibly see how their simpler implementations could be reused to develop more complex components. For example, students implemented tuples so that they could implement distance functions. Distance functions were used to implement seek behaviors, which were then used in their implementation of path finding, which is then used within finite state machines to move toward different goals, and so on. Concretely, we began with a simple environment devoid of obstacles and a world representation, as shown in Fig. 2. Students used this environment until Module 7: Pathfinding I (see Table I) as a platform for reinforcing mathematical foundations as well as developing their programming language skills. When the simpler environment representation became insufficient for expressing advanced concepts, students moved to a more complicated representation as shown in Fig. 3. The latter representation provided affordances for discussing more sophisticated topics that required a world model, and was used for the remainder of the course.

[6]The book assumes a moderate knowledge of mathematics. Additional class time may be required to cover these pre-requisites.

III. Methodology

Student data was collected anonymously in full accordance with research protocols at the authors' respective institutions. Students did not receive extra credit for participating in the study. The course consisted of $n = 18$ students (17 male, 1 female), but student data was discarded for students under the age of 18, and for students who did not consent to releasing their data. The course was offered as an elective, and students self-selected to enroll in the course, which is a potential bias.

To identify changes in student attitudes, students completed the Moskal's Attitudes Toward Computer Science survey [4], once immediately before the core AI component of the course and again at the end of the semester. The purpose of this qualitative instrument is to better understand factors that discourage students from pursuing degrees in computer science. The survey measures attitudes across five constructs: confidence, interest, gender, usefulness, and professional. Because of student demographics, the gender construct questions were removed from the survey. To score the survey results, the 4-point Likert-item survey responses are re-coded to a numerical scale which ranges from 1 to 4; negatively phrased questions were reverse coded such that a high score is always a positive attitude. The score for a construct is simply the sum of the responses for that construct.

To reflect on their own learning preferences, students also completed a Felder-Soloman Index of Learning Styles (ILS) questionnaire [3] at the end of the tenth class session. This is a 44-question instrument that can be used to assess preferences on four dimensions: active/reflective (ACT-REF), sensing/intuitive (SEN-INT), visual/verbal (VIS-VRB), and sequential/global (SEQ-GLO). The scoring methodology categorizes students for each of these dimensions ranging through, for example, Strongly Active, Moderately Active,

TABLE II. ATTITUDES TOWARD COMPUTER SCIENCE SURVEY

	Construct							
	Confidence $p = 0.362$		Interest $p = 0.011$		Usefulness $p = 0.172$		Professional $p = 0.037$	
Student	Pre	Post	Pre	Post	Pre	Post	Pre	Post
SA	28	29	36	40	24	24	16	16
SC	28	27	35	38	24	24	13	13
SD	28	29	35	37	21	24	7	16
SE	23	29	30	40	18	24	11	13
SF	20	16	18	22	18	18	10	12
SJ	23	26	27	38	22	24	11	11
SK	29	23	36	32	24	18	12	13
SL	29	29	29	32	22	23	14	12
SP	23	27	25	34	17	21	11	16
SQ	28	28	36	40	24	24	12	15

Balanced, Moderately Reflective, and Strongly Reflective. For this research, we compared the ILS distributions for each dimension against a typical four-year University CS1 program [14], to determine if statistically significant distributional differences exist between two-year and four-year populations.

To evaluate perceptions of the course, students completed a 13-question final evaluation consisting of 5-point Likert-type items. We provided this to the students directly before their final exam.

To validate if the course is relevant and useful to industry, we conducted four semi-structured, in-person interviews with game companies within the Raleigh, North Carolina area. The structure was as follows: a discussion about the course topics, the choice of Python as a programming language, and the blended learning approach of the class. The interviews were conducted over four months, and each session ranged from 1.5 to 3 hours. The first author was both the instructor and interviewer, which may have resulted in social desirability bias.

IV. STUDENT-CENTRIC RESULTS

In this section, we discuss the results of the Moskal's Attitudes Toward Computer Science survey [4], the Felder-Soloman Index of Learning Styles questionnaire [3], and the course evaluation.

A. Attitudes Toward Computer Science

Computer games have been cited as a motivational tool to teach Computer Science concepts [15]. Consequently, we were interested whether students who completed an Artificial Intelligence in Games course would have a positive increase in their attitudes toward Computer Science. Students completed Moskal's Attitudes Toward Computer Science instrument [4] directly before the AI component of the course, and again at the end of the semester. Students were randomly assigned a unique identifier, and the results for students who completed both pre-test and post-test surveys are shown in Table II. A Wilcoxon matched pairs signed-rank test was performed between the two trials, and constructs for students' interests in CS (W(10) = 25, p = 0.011) and students' beliefs about professionals in CS (W(10) = 49.5, p = 0.037) increased significantly.

Since students take many courses during a semester, it should be noted that there are several confounding factors that prevent us from attributing this affect to our course alone. There is also a survivorship bias, in that students who dropped the course before the final survey did not complete both surveys. It is indeed plausible that students who dropped the class had negative attitudinal changes. However, when taking into account the student evaluations, we believe that some component of these positive attitudinal changes were a result of our course.

TABLE III. INDEX OF LEARNING STYLES AGGREGATED SURVEY RESULTS, COMPARING OUR AI CLASS WITH A 4-YEAR CS1 CLASS

	Dimension (A-B)							
	ACT-REF $p = 0.234$		SEN-INT $p = 0.484$		VIS-VRB $p = 0.356$		SEQ-GLO $p = 0.054$	
Preference	AI	CS1	AI	CS1	AI	CS1	AI	CS1
Strong-A	1	13	0	24	4	64	0	8
Moderate-A	3	56	4	53	7	80	3	49
Balanced	7	123	6	101	1	70	6	139
Moderate-B	1	24	2	31	2	28	4	20
Strong-B	2	6	2	13	0	12	1	2

B. Index of Learning Styles

We hypothesized that one of the differences between community college students and traditional University students was that the two student populations have significantly different learning styles, and that these differences can be used to inform the pedagogical approaches in the class. To evaluate this hypothesis, we compared our results against a student population from an Introduction to Computer Science course at a four-year University [14]. Due to the small number of students in our AI class, Fisher's exact test was applied. These results are shown in Table III. We were unable to identify any statistical differences between the two populations, though SEQ-GLO appears to be on the threshold of significance. Thus, as with the CS1 population, a Shapiro-Wilk normality test reveals that, for all dimensions, we were unable to reject student populations as significantly different from normal (all $p > 0.05$). Consequently, our interpretation of this result is that pedagogical approaches that are explicitly based on ILS can potentially be applied with success in community colleges.

C. Student Evaluation

We had 18 students initially enroll, 4 of whom dropped before the final exam. One student did not complete the final evaluation. Though our reports reflect some level of survivorship bias, our course retention rate of 78% is comparable to the Simulation and Game Design program average retention rate (78%).

We expected a progression in student perception of difficulty, with the pre-lab exercises being easiest and the midterm exam being most difficult. Using the Wilcoxon matched pairs signed-rank test, we were unable to find any statistical significance to confirm this expectation. We believe that this is in part due to the small number of students. Qualitatively, we observe a monotonic increase in difficulty between the in-class exercises and the out-of-class homework. Our explanation for this is that students receive help during class from the instructor.

10 out of 13 students (77%) felt that the practical exam format was appropriate or very appropriate and that they would

TABLE IV. STUDENT EVALUATION AGGREGATED RESPONSES

Qn	Question Text	Scale	Likert-item Counts				
			1	2	3	4	5
Q1	The following question asks you to rate the difficulty of the course materials.	Very Difficult—Very Easy					
	How difficult were the questions for the pre-lab exercises?		–	5	8	–	–
	How difficult were the questions for the in-class exercises?		–	5	5	3	–
	How difficult were the homework assignments?		–	8	4	1	–
	How difficult was the midterm exam?		2	5	5	1	–
Q2	How often did you read the textbook material before coming to class?	Never—All of the Time	1	5	4	3	–
Q3	How often did filling out the in-class exercises cause you to miss important parts of the lecture?	Never—All of the Time	4	4	3	2	–
Q4	The midterm exam was a practical exam. How appropriate was this exam format for this course?	Very Inappropriate—Very Appropriate	–	1	2	4	6
Q5	If given the option, to what degree would you avoid or prefer this exam format for future exams?	Very Strongly Avoid—Very Strongly Prefer	–	–	3	5	5
Q6	During class, the instructor would go to each student to check their progress during in-class exercises. How useful were these one-on-one interactions with the instructor?	Very Useless—Very Useful	–	–	1	2	10
Q7	This course used a blended learning classroom approach. If given the option in your future classes, would you avoid or prefer classes that used this teaching style?	Very Strongly Avoid—Very Strongly Prefer	–	–	5	3	5
Q8	Please indicate your level of agreement with the following statements.	Strongly Disagree—Strongly Agree					
	The in-class exercises encouraged me to attend the lectures.		–	1	1	5	6
	The in-class exercises make the lectures interesting.		–	–	2	7	4
	The in-class exercises helped me better understand the lecture material.		–	1	1	3	8
	The in-class exercises helped me complete the homework.		–	–	2	5	6
	The assigned textbook was a useful resource in the class.		–	3	5	3	2
Q9	This course used the Python programming language for all programming activities. How difficult or easy was it to learn the Python language?	Very Difficult—Very Easy	1	3	4	4	1
Q10	For future programming tasks, how likely would you be to use Python as your preferred language?	Very Unlikely—Very Likely	–	2	3	4	4
Q11	How important do you feel the material in this class is in obtaining an entry-level game developer position in industry?	Not at all Important—Extremely Important	–	–	3	6	4
Q12	Compared to other classes that you have taken at this community college, how would you rate this course?	One of the Worst—One of the Best	–	–	3	1	9
Q13	Overall, how satisfied were you with this course?	Very Dissatisfied—Very Satisfied	–	–	3	3	7

prefer this exam format for future exams. More importantly, 8 out of 13 students (62%) strongly or very strongly preferred the blended learning approach to class. The other 5 out of 13 students (38%) were indifferent, and no students indicated that they would avoid this class format.

12 students (92%) found the one-on-one interactions with the instructor to be useful, despite having only a few minutes of interaction time with the instructor for each exercise. In our opinion, this is because the instructor could quickly steer otherwise perplexed students in the correct direction.

The department allows students to have up to 4 penalty-free absences before being dropped from the course. Of the 13 students who completed the course, 6 absences in total were recorded across all students, and 8 students had perfect attendance. 11 students (85%) agreed or strongly agreed that in-class exercises encouraged them to attend the lectures, reported that the exercises made the lectures interesting, and reported that the exercises helped them complete the homework. Thus, if students did not find class to be useful, we would have expected many more students to have absences.

10 students (77%) believed the class was very or extremely important in obtaining an entry-level game developer position, which shows that students perceive the course as being relevant. 9 students (69%) rated this course as one of the best they have taken at our community college. In general, 10 students (79%) were satisified or very satisfied with the course.

V. INDUSTRY COURSE EVALUATION RESULTS

The selected companies constitute what we believe to be a representative sampling from the diverse types of game studios in the marketplace: Redstorm (RS),[7] for traditional AAA game development; Vicious Cycle (VC),[8] for game engine design; Virtual Heroes (VH),[9] for serious research games; and Spark Plug Games (SP),[10] for mobile and independent game development. Programmers who directly worked with Artificial Intelligence in some capacity within their company evaluated our curriculum and contributed feedback for three measures: selection and ordering of topics, choice of Python as a programming language, and the use of blended learning.

With few exceptions, all four companies found the course topics and topic organization to be appropriate to the game industry. When we designed the course, we envisioned the sections on Supporting Technologies as optional, but all of the interviewed companies felt that this section was necessary to show how game programming concepts can transfer to other programming domains. However, VC felt that Supporting Technologies should only be a single class, using the gained class time on Learning in AI. RS suggested that students should implement a simple, but complete game (such as Asteroids or Pacman), rather than using prototype

[7] http://www.redstorm.com
[8] http://www.viciouscycleinc.com
[9] http://www.virtualheroes.com
[10] http://www.sparkpluggames.com

environments. Decision trees and finite state machines were considered to be an essential topic for all companies, but they differed on the importance of pathfinding. RS suggested that the topics of decision trees and pathfinding be reversed, since pathfinding is a relatively complex topic. Specifically, RS, VC, and VH indicated that all modern game engines support pathfinding, and that breadth-first search could be covered in less detail or abstracted entirely as a result. VC and VH also indicated that the concept of navigation meshes should be presented as an alternative to navigation points.

We were concerned that our course did not make use of an industry game engine, such as Unity.[11] However, the use of Python and Pygame was well-received by all companies. From our interviews, it was clear that companies are less interested in knowledge of particular tools and more interested in students' programming capabilities. For example, RS was interested in whether students understood the thought process behind programming and VH was similarly interested in students who have a base-level knowledge of programming. SP liked Python because it allowed students to focus on concepts, as opposed to focusing on language idiosyncrasies. In terms of entry-level opportunities, RS liked the use of Python because they considered scripting to be a stepping stone into other areas; their entry-level developers begin as scripters. VC entry-level hires also begin in a scripting or support role, and thus found Python to be appropriate. At VH, all entry-level hires start out as generalists, and at SP, their game programmers work on all aspects of the game. Thus, we conclude that teaching through Python is not a significant barrier to entering the game industry, and we recommend that community colleges examine their balance between tool-focused courses and concept courses.

All companies found the blended learning approach to the classroom to parallel the type of work performed in industry. RS liked the interactivity of the course, and how it built on concepts from first principles. VH felt that the course format accurately resembled the way that game programmers solve problems in industry. VC stated that having immediate feedback is valuable, and that the course offered several opportunities to practice debugging code — an essential skill. The practical midterm and final exam formats were also well-received. In general, companies preferred the problem-based learning format to lecture-based learning. In our interviews, we found that companies are interested in what projects students have completed more than the courses they've taken. In particular, SP liked that the course provided students with deliverables that could be discussed during interviews. VH also indicated that it was very important for students to have completed projects when applying for entry-level positions.

VI. Conclusion

While we have shown a positive increase in attitudes within the interest and professional constructs, we are unable to isolate these increases to our course alone. Future replications of this course may mitigate some of these confounding factors. The results of our final class evaluation demonstrate that by having a blended learning approach, no students are marginalized when compared with alternative classroom formats, but a significant number of students stand to have an improved experience. Finally, our industry interviews suggest that game companies prefer a blended learning approach to a traditional lecture because it more closely matches their industry practices.

[11] http://unity3d.com/

Acknowledgments

The authors would like to thank Steve Reid (Managing Director and Executive Vice President) and Greg Stelmack (Expert Game Systems Engineer) of Redstorm Entertainment, an Ubisoft Entertainment company; Wayne Harvey (Vice President/CTO), Allan Campbell (Gameplay Programmer), and Alfred McNulty (Gameplay Programmer) of Vicious Cycle; Jerry Heneghan (Director of Product Development) and Oliver Gray (Computer Game Programmer) of Virtual Heroes; and John O'Neill (President/CEO) of Spark Plug Games for their expertise and feedback.

References

[1] P. Smittle, "Principles for Effective Teaching in Developmental Education." *Journal of Developmental Education*, vol. 26, no. 3, pp. 10–16, 2003.

[2] T. Bailey and J. Forbes, "Just-in-time teaching for CS0," *ACM SIGCSE Bulletin*, vol. 37, no. 1, p. 366, Feb. 2005.

[3] R. Felder and J. Spurlin, "Applications, reliability and validity of the Index of Learning Styles," *International Journal of Engineering Education*, vol. 21, no. 1, pp. 103–112, 2005.

[4] A. Hoegh and B. M. Moskal, "Examining science and engineering students' attitudes toward computer science," in *29th IEEE Frontiers in Education Conference*, Oct. 2009, pp. 1–6.

[5] A. Lenhart, J. Kahne, E. Middaugh, A. Macgill, C. Evens, and J. Vitak, "Teens, Video Games, and Civics: Teens," *Pew Internet & American Life Project*, 2008.

[6] D. R. Krathwohl, "A revision of Bloom's Taxonomy: An overview," *Theory into Practice*, vol. 41, no. 4, pp. 212–218, Oct. 2002.

[7] P. Carter, "An experiment with online instruction and active learning in an introductory computing course for engineers," in *Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE '09*. New York, New York, USA: ACM Press, May 2009, p. 103.

[8] K.-K. Lau, "Active learning sheets for a beginner's course on reasoning about imperative programs," *ACM SIGCSE Bulletin*, vol. 39, no. 1, p. 198, Mar. 2007.

[9] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '07. New York, NY, USA: ACM, 2007, pp. 204–223.

[10] D. Ranum, B. Miller, J. Zelle, and M. Guzdial, "Successful approaches to teaching introductory computer science courses with Python," in *ACM SIGCSE Bulletin*, vol. 38, no. 1, Mar. 2006, p. 396.

[11] L. Grandell, M. Peltomäki, R.-J. Back, and T. Salakoski, "Why complicate things?: Introducing programming in high school using Python," in *Proceedings of the 8th Australasian Conference on Computing Education*, Jan. 2006, pp. 71–80.

[12] W. Mcgugan, *Beginning Game Development with Python and Pygame: From Novice to Professional*. New York, NY: Apress, 2007.

[13] I. Millington and J. Funge, *Artificial Intelligence for Games, Second Edition*, 2nd ed. Burlington, MA: Elsevier, 2009.

[14] J. Allert, "Learning style and factors contributing to success in an introductory computer science course," in *IEEE International Conference on Advanced Learning Technologies*, 2004, pp. 385–389.

[15] D. Cliburn, "The Effectiveness of Games as Assignments in an Introductory Programming Course," in *The 36th Annual Frontiers in Education Conference*, 2006, pp. 6–10.