A GAME DEVELOPMENT COURSE FOR NON-CS MAJORS

Lubomir Ivanov

Department of Computer Science, Iona College (UNITED STATES)

Abstract

This paper describes our experience in creating and teaching an introductory, undergraduate Game Development course for non-Computer Science majors. The hands-on course uses the Unity game engine to teach a variety of Computer Science (CS) topics and inspire students to consider majoring in Computer Science. The paper discusses the course objectives, the approach to presenting the material, and the course outcomes in terms of student success and CS majors recruitment.

Keywords: Game development; Game Engine, Unity; Non-majors; CS education.

1 INTRODUCTION

Most colleges and universities offer an introductory Computer Science (CS) course for non-majors – students who either do not intend to major in Computer Science or are undecided. In addition to providing a service to other departments, the non-majors course often offers student-recruitment opportunities for the CS Department: A number of students typically become interested in the subject area, and select Computer Science as their major.

Much has been written about the non-majors course and its issues, topics coverage, and overall focus [1-6]. Rather than reiterate everything that has been said, we will simply point out that the main difficulty in making the non-majors course truly successful comes from the fact that the course serves a broad mix of students with diverse backgrounds, levels of preparation, and academic and non-academic interests. As a result, it is difficult to select a set of topics, which will interest most students. The two common course design strategies adopted by most schools are:

- Teach a traditional CS1-type course, emphasizing problem analysis and solutions design. This approach usually involves an introduction to a specific programming language like C++, Java, Python, etc.
- Discuss a broad mix of traditional and modern CS topics, highlighting the impact of Computer Science on other disciplines and, in general, on all areas of modern life. In this approach, topical breadth rather than depth is emphasized.

Iona College has traditionally opted for the latter approach – a broad introduction to Computer Science and its applications. Our non-majors course is part of the College Core, and includes a brief introduction to Javascript or Python, as well as some basic elements of client-side web design using HTML and CSS. All students except CS, Science, and Mathematics majors are required to take the course as part of fulfilling their College Core requirements. However, the annual course evaluations routinely indicate a broad dissatisfaction with the course, regardless of instructor, programming language choice, or topics coverage. The consensus among students is that the course is “boring” and has little to do with their career paths.

A redesign of the Iona College Core during 2015-16 provided an opportunity to diversify the set of courses offered to non-majors. The first new, “special topics” non-majors course was Game Development. It was offered initially during the Spring’16 semester, and offered again during the Fall’16 semester. The course is now routinely offered every Fall and Spring semester, and has been supplemented by other “special topics” courses such as Robotics and Web Design.

Game development is, of course, not a new course topic [7-23]. Faculty have long recognized the potential of game development to attract student interest and drive student creativity. Over the last 10-15 years, game development courses for CS majors have been offered at many universities and at all levels – from freshman to senior. In recent years, the availability of powerful and easy to use game engines (Unity, Unreal) and frameworks (MS XNA) has led to the integration of game development
into traditional Computer Science courses, including the capstone course [19-22], and to the creation of new game development courses [9, 10, 14, 16, 17], concentrations and majors [23]. The vast majority of these courses, however, are intended for CS majors, and offered either at the CS1/CS2 level or as upper-level electives.

This paper describes our experience in creating, teaching, and assessing a game development course for non-CS majors. We outline our motivation and course objectives, present our instructional methodology and topics covered in the course, and share some experiences and lessons learned. We also discuss the course outcomes in terms of student performance and recruitment of new CS majors.

2 MOTIVATION

The primary motivation for creating the “Game Development” course for non-CS majors was to generate broad student interest in Computer Science and improve the quality of our students’ educational experience. Of course, we anticipated that the course will inspire some students to pursue a degree in Computer Science. Another important motivation was to highlight to students a challenging, stimulating, and rewarding career path in the ever-growing gaming industry.

The two main issues with teaching this type of course to non-majors are the programming aspects and the large amount of computer graphics/audio terminology. Some faculty have tried to avoid the complexities of teaching programming by using visual languages and environments [11, 18]. However, we believe that, as an important aspect of software development and Computer Science, programming should be a part of the students’ course experience. If a student wishes to pursue Computer Science as a major, he/she should be aware of what is really involved in developing a complex software system such as 3D game. Moreover, introducing students to programming in the introductory course helps them build essential skills and algorithmic thinking as early as possible. The key to success is balancing the amount of programming and concepts with the creative side of game development (writing the game story, creating the visual and sound elements, etc.) and the marketing of the finished product. This is particularly important for non-CS students, who may be majoring in English, History, Fine Arts, Business, and any number of other fields. Bringing in interdisciplinary elements into the Game Development course ensures that most students are excited about at least some aspects of the game development process and eagerly contribute to the course projects. This interdisciplinary emphasis illustrates to students that Computer Science is in a truly symbiotic relationship with just about every other field of human endeavor, and, regardless of which career path they choose, Computer Science will have an impact on their professional and personal lives.

3 THE GAME DEVELOPMENT COURSE

3.1 Course Objectives

To measure the success of the course with regard to student performance, we adopted a number of student learning outcomes (SLOs), based on which the assessment at the end of the semester is carried out. We expect that, by the end of the semester, students enrolled in the course will be able to:

- Write a simple game plot, including a pre-story, and a non-linear game-play storyline
- Use game engines and graphics and audio tools to create game components
- Use a scripting language to create game scripts
- Integrate game components into a multiplatform game
- Work in a team of software developers
- Present and defend creative work in a public forum
- Discuss the social/ethical/psychological aspects of game design and their impact on people and society

An additional measure of success for the CS Department is the number of students who decide to pursue a Computer Science degree after taking the non-majors Game Development course.
3.2 The Unity Game Engine

The main tool used for game development in our new course is the Unity game engine. It was introduced by Unity Technologies in 2005, and has since become one of the most popular and powerful platforms for developing 2D/3D games. Unity is widely used by individual developers as well as by small companies and large game corporations. The latest version, Unity 5, is a fully functional game engine, with all advanced features enabled and freely available to individual developers and non-for-profit academic institutions. Unity is an integrated environment, which combines a number of sophisticated components such as the PhysX physics engine, the Mecanim animation system, a self-contained terrain editor, and more. Unity is also seamlessly integrated with the Monodevelop code editor, so that any changes to the code made in Monodevelop are transparently compiled by Unity to .NET dlls and inserted into the game. Errors are displayed in the Unity console, and are fairly descriptive even for beginners. In an effort to make game development easy and intuitive, Unity has fully automated many aspects of game development, including complex coding issues such as shading and navigation. Thus, for the casual developer, the coding aspects of the projects are relatively few and fairly simple. This makes Unity particularly well suited for beginners, who gain knowledge and experience writing short Javascript or C# scripts, based on the Unity API. Another major advantage of Unity is that the developed games can be easily ported to multiple platforms such as Windows, Linux, Mac OS, iOS, Android.

3.3 Student Makeup

Since the Game Development course is intended for non-CS majors, it has no prerequisites. However, during the initial offering of the course, a permission from the CS Department Chair was required before a student could register for the course. Due to the limited number of sections offered, this measure ensured that only students with a relatively high SAT-Math scores could enroll in the course. During the initial offering of the course, there were twenty students enrolled, only four of whom had any prior programming experience. The enrollment included twelve freshmen, four sophomores, and four seniors. The upperclassmen represented several non-CS majors (Accounting, Mass Communications, Psychology, and English), but the majority of the freshmen were undecided.

During the Fall'16 offering of the course, the Departmental Chair permission requirement was lifted. The enrollment consisted of eighteen freshmen, two of which had a limited prior experience with Unity, but not with programming. The distribution of intended majors included English, Business, Criminal Justice, Biology, Mass Communications, and Psychology.

3.4 Course Material

The course is offered in a lab environment, where each student works on an individual workstation (half the machines are Windows-based, the other half are Macs). The lectures are hands-on, with students experimenting in real-time with the material covered in class.

The course begins with a top-level introduction to game design, focusing on game genres, story and script writing, basic game psychology, socio-economic, age, and gender issues in gaming, and the use of serious games in other disciplines. Emphasis is placed on the significance of creating an immersive experience for the user by providing concrete game rules, manageable objectives, properly-timed user feedback, incremental increase in gameplay complexity, and an intuitive game interface.

The next few lectures introduce students to Unity. While fairly intuitive, Unity can be overwhelming for a complete beginner due to the rich terminology and the large set of tools for manipulating the 3D environment. Therefore, the instructor has to be careful not to inundate students with too much information in too short a time. Students are given sufficient time to get accustomed to the various windows, hierarchies, and “gizmos” that Unity provides. Hands-on exercises like creating simple shapes and texturing them with prefabricated or scratch-built material make students feel more comfortable within the Unity environment, and help them learn the associated terminology (scenes, objects, transforms, components, meshes, renderers, maps, etc.). A subsequent introduction to the Unity terrain editor and the notion of “prefabs” (prefabricated objects) gives students even more confidence and practice in creating and manipulating virtual worlds. At this point more advanced topics

---

2 Unity 5 is free as long as the developer(s) profits do not exceed a certain threshold.
such as lighting, shadows and the fundamentals of the physics system (rigidbodies, colliders, event triggers) are introduced.

With the introduction of game physics, the need for scripting becomes evident. In Unity, the developer writes independent scripts using the Monodevelop editor, which is a part of the Unity package. Once the developer saves a script file and returns to the Unity window, the scripts are automatically compiled and integrated into game. For students with prior programming experience, this type of coding feels strange at first. However, we have found that it works well for students with no prior programming background, since they can focus on relatively small and simple scripts and still achieve fairly intricate visual effects within the game. Unity scripting can be done in C#, Javascript, or Boo (a Unity native language). We felt that, for complete beginners, Javascript will offer the most seamless introduction to programming.

Teaching the fundamentals of programming to absolute beginners does take a few lectures. As is customary in a first CS course, sufficient time must be allocated to basics such as variable declarations, types, assignments, and output. However, Unity API calls can be mixed in almost immediately. Unity requires scene initialization code to be in a Start() method, and code to be executed every frame to be in an Update() method. Therefore, students are also immediately exposed to the concept of a “function” (“method” is initially introduced as synonymous with “function”). To use the Unity API, students have to get accustomed to the “dot-notation”, which is introduced in a very straight-forward manner: Variables of different types store different types of data. If we want to have a “complex” variable that stores multiple data items, we would need to declare a special custom type – a class. Variables declared using this special type are called “objects”. To access the individual items stored in such an object, we need to separate with a “dot” the name of the variable from the name of the item (or function) we are trying to access. The first time the course was offered, we found it really surprising how quickly students got used to these concepts and began using “dot notation” correctly and consistently. We believe that this is partly due to the fact that, while familiarizing themselves with the Unity environment, students have already become accustomed to the notion of a visual object in a Unity scene consisting of multiple items (properties) such as a transform, a mesh renderer, a rigid body, etc. In turn, each property may consist of other sub-properties, e.g. the transform has position, rotation, and scale sub-properties, each of which has “X”, “Y”, and “Z” components. Thus, once students get accustomed to visual “objects” being implemented in this “matryoshka-doll”-style, it becomes relatively easy to adapt to the equivalent “dot-notation” coding convention. More importantly, students quickly realize the equivalence between graphical and Javascript objects. This makes the discussion of object instantiation from a Unity-prefab/Javascript-class very natural. By the time of the first test – a third into the semester – most students are comfortable with the object-oriented nature of Unity.

Once students become accustomed to Javascript programming, new visual and coding elements can be introduced: Particle systems, fire and water effects, underwater swimming scripts, script-driven scene fade-in/fade-outs, and “magic” game objects activated through touch/pickup scripts make the game experience more immersive and challenging. Students write scripts to maintain the player’s game statistics such as health and score points. Using the Unity UI system, they learn how to create overlays to display relevant information and update it dynamically through scripting. Switching between successive game scenes is also done with scripts.

During the last third of the semester, students learn the basics of Unity animation, which is based on the concept of a finite state machine. This presents an excellent opportunity for the instructor to “go off script” and briefly discuss the larger issues of Computer Science such as artificial intelligence, computability and complexity.

The final weeks of the course are dedicated to the issues of fine-tuning performance, porting the complete game software to different platforms (Windows, Mac OS, iOS, Android), and marketing, distribution, and maintenance of the software product.

3.5 Student Projects

In the course of the semester students are required to complete one large and several (4-5) smaller projects. Working in teams of two, students are expected to implement an original, first person action/adventure 3D game. The game is expected to have at least two levels, each with a different environment/terrain, multiple obstacles/challenges/enemies, and a well-defined success/failure outcome. Students must also implement scene transitions between levels and a UI system, which displays the player’s score and/or health points. To be successful, students must meet a number of
milestones: An original plot idea, a detailed game narrative/script, a terrain/environment for each game level, objectives, obstacles/enemies, sound, etc. Every milestone adds another layer of complexity to the overall project and accounts for a certain percentage of each student’s overall score for the semester. The final projects must be submitted according to a strict set of guidelines, which include technical documentation, a list of resource attributions, a users’ guide, and a marketing strategy.

The student final game projects are presented publicly before faculty and students from the CS and other departments. Several pictures from the end-of-the-semester student presentations are displayed in Fig. 1:

![Sample Pictures from the Final Project Presentations](image)

Figure 1: Sample Pictures from the Final Project Presentations: Some students were so excited about their completed games that they dressed in costumes similar to their in-game avatars.

In addition to the main course project, a number of individual smaller projects are assigned throughout the semester. One of these projects asks students to create an animated logo for their “company”. The logo is eventually integrated as the opening screen of the semester-long group project. Other projects allow students to practice Javascript, experiment with the PhysX system, and test finite-state machine driven third-person avatar animation.

Finally, students are required to write a 4-5-page research paper on a topic related to diversity in computer gaming: Any topic related to cultural, ethnic, gender, race, age, and national origin diversity in game-development or game-playing is acceptable. Students are expected to research the issue, present opposing points of view, and formulate and defend their personal position based on the accumulated data. This is considered a critical thinking assignment for College Core purposes, and is uploaded to the students’ learning e-portfolio accounts. The instructor must provide feedback for each individual student paper.

4 ASSESSING SUCCESS

The assessment of the non-majors Game Development course is accomplished through a set of direct and indirect measures. The direct measures include the grades for all milestone assignments, individual projects, the final project, final presentation, midterm and final exams, and the diversity research paper. The indirect measures consist of student and faculty evaluations done at the end of each semester. Students complete an online evaluation form that asks them to evaluate the extent to which the course has met each of its objectives. The course instructor submits a spreadsheet indicating the extent to which each of the assessment tools met the desired course objectives. The results of these assessments are analyzed to identify problems and find solutions.
Overall, student evaluations have been very positive as were the comments students left for the instructor as part of the survey (see Appendix at the end of the paper). Tables 1 and 2 summarize some basic statistics about the two course offerings.

Table 1. Enrollment and Survey Statistics

<table>
<thead>
<tr>
<th>Semester</th>
<th>Initial Enrollment</th>
<th>Final Enrollment</th>
<th>Responded to Survey</th>
<th>% Responded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring’16</td>
<td>20</td>
<td>19</td>
<td>12</td>
<td>63%</td>
</tr>
<tr>
<td>Fall’16</td>
<td>18</td>
<td>14</td>
<td>10</td>
<td>71%</td>
</tr>
</tbody>
</table>

Table 2. Assessment Results

<table>
<thead>
<tr>
<th>Semester</th>
<th>Course Assessment Average</th>
<th>Departmental Avg. (All CS Courses)</th>
<th>Students Electing to Major in CS</th>
<th>% Recruited to Major in CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring’16</td>
<td>3.38/4</td>
<td>3.22/4</td>
<td>5</td>
<td>26.3%</td>
</tr>
<tr>
<td>Fall’16</td>
<td>3.23/4</td>
<td>3.24/4</td>
<td>3</td>
<td>21.4%</td>
</tr>
</tbody>
</table>

It is instructive to compare some statistics between the two course offerings. During the Spring’16 semester, only 1 person withdrew from the course, while during the Fall’16 semester, 4 students dropped the class. This is almost certainly due to the fact that the initial course offering was more selective and required the Department Chair’s permission to enroll. The Fall semester mix of students was more typical of a regular non-majors CS core course, which usually includes some students with weaker analytical problem solving skills. The fact that the Fall’16 course assessment average is weaker than the Spring’16 can also be explained in similar terms, since a more select group of students will typically have a more favorable view of technical course. Notice, however, that the Fall’16 Game Development offering is on-par with the Fall semester departmental average, which includes all CS courses, populated with CS majors, who typically enjoy taking CS courses. Most telling, the percentage of students, who elected to major in Computer Science after taking the Game Development course was similar, despite the less selective enrollment nature of the Fall’16 course offering. As indicated in section 2, this statistic is an important measure of success for the course. During the Spring’16 semester, five of the twenty students enrolled in the course (5 of 16 if not counting the seniors) opted to major in Computer Science immediately after completion of the course. In informal conversation, 3 other students indicated that, while still undecided, they may consider majoring in Computer Science based on what they learned in the Game Development course. During the Fall’16 semester, 3 students elected to pursue a CS major, while 4 others indicated that they are considering a CS major, a CS minor, or a double-major. These CS major enrollment results far exceed anything achieved in the regular non-majors CS literacy course: On average less than 4-6 students from all 20+ sections of the basic literacy course choose to major in Computer Science every semester. The non-majors Game Development course produced a similar number from a single section. This clearly demonstrates the potential of game development to engage and inspire students to pursue a Computer Science degree.

5 LESSONS LEARNED

Our previous experience teaching project-based courses such as Mobile Applications Development, Web Development, and the upper-level Game Development course has demonstrated that students are engaged much more effectively when given the opportunity to be creative and pursue their own ideas through projects involving strong visual/auditory elements. However, before the initial offering of the new non-majors Game Development course, there were strong reservations with regard to the success of teaching a fairly complicated programming-oriented material to students with no prior exposure to any software development: We fully expected to not be able to cover all topics on the syllabus – in fact, some topics were listed as "time permitting". We also expected a large drop-out rate, and poor student evaluations at the end of the semester.

The course, however, turned out better than we had hoped: The instructor managed to cover more material than anticipated, including Computer Science concepts far beyond the introductory nature of
the course - classes and objects, the singleton pattern, finite state machines, and some aspects of computability theory. These were assessed through questions on two tests and the final exam, and the results clearly indicate that a large portion of the students gained a fair grasp of these advanced issues.

Even more impressive was what students managed to accomplish in terms of final projects. Before the initial offering of the course, we expected that many students may not be able to complete a fully functional, multi-level game. However, at the end of both course offerings, all student teams presented completed projects, which, while not as intricate and complex as those of CS upperclassmen, fully complied with all specifications, and in many cases exceeded the project requirements and implemented features that students had researched independently. All colleagues and guests who came to the final project presentations at the end of the semester were truly impressed by the level of student creativity and the breadth of student vision and imagination.

The course has a strong interdisciplinary element, emphasizing fundamental skills from disciplines as diverse as Creative Writing, Psychology, Sociology, and Marketing. Students indicated that they find this particularly useful. In conversations with the instructor, several students indicated that the course changed their perceived notion of Computer Science as a “geeky” discipline, and made them aware of how integrated Computer Science is with their particular area of study and with other disciplines.

Students also appreciated the diversity paper assignment. While some students found the assignment pointless, most indicated that it was a time-consuming but worthwhile effort. Indeed, many student papers revealed an unexpected level of maturity, depth of argument, and passion for the selected topic.

From the instructor’s point of view, while the course has fulfilled most of its objectives, there are issues that need to be resolved going forward:

- The programming concepts should be reinforced with more homework. As the tests and final exams indicated, students gained a good understanding of completed code, and were able to correctly explain the behavior of fairly complicated program segments. However, when asked to produce their own code, many experienced difficulties.

- One aspect of Javascript, which was difficult to integrate naturally into the course was loops. The basic Unity game loop, which reloads the active scene on every frame takes care of much of the necessity of using loops. It is, therefore, difficult to come up with a convincing example of the use of loops in a simple game. Of course, loops can be introduced separately from the game development topics, but a better approach, which will be adopted during the next course offering, is to use loops to instantiate large numbers of objects (scene props, enemies, etc.) from prefabs. Students will also be required to use loops in their projects. This will guarantee at least some level of familiarity with this critical concept.

- One important issue that surfaces in the course (as in many others courses) is that of plagiarism. When faced with a challenge, students often search for a ready-made solution online, taking code indiscriminately from discussion forums, and using it without understanding or acknowledgement. When confronted, students often fail to understand the inappropriateness of their actions. Thus, it is imperative that the instructor clearly spells out at the beginning of the semester what is permitted and what is considered plagiarism. The disclosure of all resources used in student projects must be mandated and enforced for each submission.

6 CONCLUSION

In this paper we presented our experience in designing and teaching a new Game Development course for non-CS majors. The course is a part of the College Core, and has attracted significant interest among incoming students as well as upperclassmen looking for an alternative to the traditional Computer Science literacy course. Based on the success of the initial offering of the course and the continued high enrollment, we believe that the course will prove to be an exciting and useful alternative for students to satisfy their College Core STEM requirement, and will attract new students to the Computer Science department.
REFERENCES


APPENDIX: STUDENT SURVEY RESPONSES

Question: Liked best about course:

<table>
<thead>
<tr>
<th>Semester</th>
<th>Student Responses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring'16</td>
<td>I enjoyed this course due to the fact it was a video game course. The subject is interesting and it portrayed video games in a better light. I now know the steps to make a video game and wish to continue my career in it.</td>
</tr>
<tr>
<td>Spring'16</td>
<td>Being able to learn hands on with the unity system</td>
</tr>
<tr>
<td>Spring'16</td>
<td>In this course I liked that you could create a game and you could create and make your own terrain and scenery</td>
</tr>
<tr>
<td>Spring'16</td>
<td>Teacher asks nothing more than what is expected, course is unique and interesting, 10/10 would suggest to others</td>
</tr>
<tr>
<td>Spring'16</td>
<td>I liked the fact that we had the opportunity to sort of create a little company and create a small game</td>
</tr>
<tr>
<td>Spring'16</td>
<td>It was great!</td>
</tr>
<tr>
<td>Fall'16</td>
<td>Creating games using a software called Unity 3d</td>
</tr>
<tr>
<td>Fall'16</td>
<td>I liked the whole creative aspect of the class, like creating game terrains</td>
</tr>
<tr>
<td>Fall'16</td>
<td>Making of the game.</td>
</tr>
<tr>
<td>Fall'16</td>
<td>It is an extremely challenging course. It’s very interesting but it’s also very complicated especially with the time frame we have.</td>
</tr>
<tr>
<td>Fall'16</td>
<td>Slightly learned how to code</td>
</tr>
</tbody>
</table>

Question: Suggested Changes:

<table>
<thead>
<tr>
<th>Semester</th>
<th>Student Responses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring'16</td>
<td>to make this class easier as this class is a core class, so there will be kids taking the course who won’t major in gaming. Therefore it should be less challenging</td>
</tr>
<tr>
<td>Spring'16</td>
<td>I think the projects should have been better spaced out into smaller chunks and additional smaller assignments to reinforce the lessons</td>
</tr>
<tr>
<td>Spring'16</td>
<td>In place of exams, 5 question quiz every week at the start of class to make sure students are aware of things going on in past classes</td>
</tr>
<tr>
<td>Spring'16</td>
<td>A little more guidance on our projects</td>
</tr>
<tr>
<td>Fall'16</td>
<td>Just to slow down a bit. Sometimes, the class moves too quickly and many students get confused</td>
</tr>
<tr>
<td>Fall’16</td>
<td>go over coding a little bit more</td>
</tr>
<tr>
<td>Fall’16</td>
<td>simpler assignments, because certain assignments that are assigned dealing with code and other similar aspects are really difficult especially if you haven’t had any prior experience. I suggest that the coding aspect be applied to a higher level computer class.</td>
</tr>
<tr>
<td>Fall’16</td>
<td>Nothing much really.</td>
</tr>
<tr>
<td>Fall’16</td>
<td>Less small projects in between and more help/Guidance with the final project. Also stressing the importance of team work is going to need to be more significant with future classes.</td>
</tr>
<tr>
<td>Fall’16</td>
<td>Elaborate more on lectures. Don’t sidetrack yourself. Stay on point and people should be fine. Don’t expect people to know what they’re doing when doing it for the first time. Other than that, good class.</td>
</tr>
</tbody>
</table>