VectorForce: A Video Game Project

Mark Chapman, David Turner, Arturo Concepcion, Aldo Lewis
School of Computer Science and Engineering
California State University, San Bernardino
San Bernardino, CA 92407

ABSTRACT

In an era where videogames are beginning to surpass the film industry's gross product, an increasing number of students find themselves gravitating to a gaming field that continues to grow even in a recession. It was no surprise that including a game design degree helped to bolster our own school's declining computer science enrollment. However, it was not enough to simply provide theories on game design in order to both prepare students for potential video game industry jobs or other software development careers. Very early in the curriculum design process, it became apparent that we needed to create an actual game in order to solidify students' understanding of game design principles. While there is a swath of scholarly work being performed on game design theory and specialized technical areas, there is little work about how one actually proceeds in creating an entire game, much less how to structure a curriculum that will actually allow instructors to teach students to create a game. The intent of this paper is to provide a heuristic of how one might organize a team of students to create a videogame, and includes a discussion of the logistics required for such an undertaking. The task is accomplished by outlining the creation of VectorForce, the first game ever created at CSUSB. Game production began in the summer of 2007 and ended when the game was published in the Microsoft Xbox Marketplace on 23 Sep 2009.

Categories and Subject Descriptors

K.3.2 [**Computer and Education**]: Computer and Information Science Education- *computer science education, curriculum.*

General Terms

Design, Human Factors, Theory

Keywords

Game Curriculum, Game Design, Game Development

1. INTRODUCTION

While fears of outsourcing and a devalued hi-tech American job market drive students to major in non-computer science professions, a number of university departments across the country have started to experiment with using video game creation and design to attract freshmen students. Some departments use small video games to outline computational concepts [1, 2, 3, 5, 6]. Such projects demonstrated the fact that students were interested in creating tangible programs and demonstrative products rather than abstract computational concepts. In fact, these researchers found that they were able to attract students who would otherwise not be interested in a computer science education. In some cases a game design program was even more alluring than an in-demand IT degree. Though the School of Computer Science and Engineering at CSUSB already offers an IT degree called the B.A. degree in Computer Systems, the understanding that computer games appeal to undergraduates prompted the creation of a game design option within this degree program.

Designed in the Summer of 2007, VectorForce is an adaptation of the traditional top-down shooter genre exemplified by games such as Raiden, Ikaruga, Gradius, and R-type. Initially dubbed CyberShot, production of VectorForce began with the leadership of two computer science faculty members at CSUSB who allowed students to follow their passion for game design. While the two faculty members provided their knowledge of computational concepts and organizational structures, they handed the creative reins to students. With a small team of a dozen students, VectorForce gained quick momentum because it appealed to student interests and allowed them to learn at their own pace, encouraging them to define their own stakes in the project.

To make the endeavor even more concrete, we decided on a development route that would allow us to not only create the game, but also publish it. Microsoft's XNA creator's club provides independent creators with a well-documented object-oriented framework that permits the quick creation of a polished game for PCs and the Xbox game console. The XNA framework provides an interface that greatly simplifies the loading and use of art assets such as 3D models, images, music and sound effects. Game code under the XNA framework is written in C#, which is a much easier language to use than C++, the traditional language used in video game creation. With XNA, students could focus more attention on learning general concepts necessary for creating a game without being bogged down by extensive low-level programming, while still maintaining a clear goal. The creation of the game was not just an exercise; it was a project that would be available to the public through distribution using Microsoft's Xbox Live Marketplace. The students would be creating the project for a large audience.

The impetus for designing a real game was carried into game design classes where students were given the option to conceptualize their own games and could also see the development of a more polished product. Thus, classes such as Game Design and Game Programming dealt equally with theory and application, which provided students with a clear connection between the two fields. The game design class centered on the design process of game creation including choosing a genre, documentation, art creation, art loading, and level design. The game programming class focused on the creation of programming objects using C#, Lua, OpenGL, XML, etc., as well as using any other tools provided by the XNA framework. Having a working game was vital to teaching both classes. Of course, the road to a working game was long and filled with a number of potential detours. The remainder of the paper focuses on the obstacles we encountered and the rationale behind our choices.

We specifically outline the path taken until VectorForce was accepted onto the Xbox Live Marketplace on September 23rd, 2009—the game may be found at http://cse.csusb.edu/games/vf/. The paper starts with a discussion of initial game development considerations such as choosing a game genre and finding a niche within that genre market, discovering what resources are available for developing the game, and researching what tools can enable the creation of the game. We then elaborate on design considerations that needed to be revisited after production began. In particular, reclassifying gameplay elements for more logical and innovative game mechanics, deciding game length logistics, and examining art assets to verify art style cohesion and quality. Finally, no matter how well a game is designed, there will always be a need for fine-tuning the finished product. We therefore discuss the internal and external playtesting phase of our development, where we polished the game before wide release.

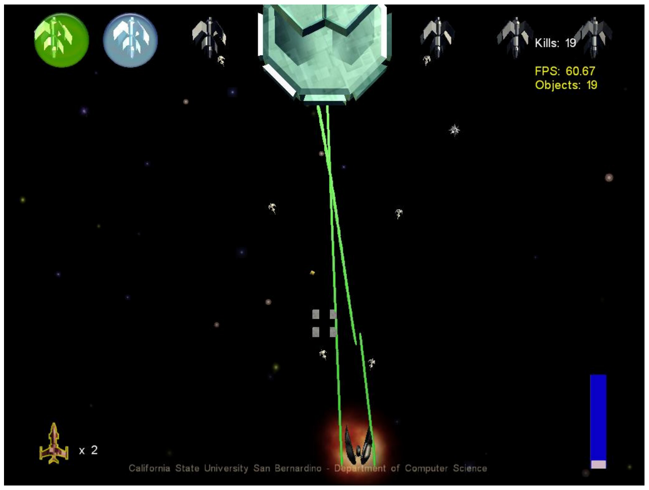
2. INITIAL GAME DESIGN

When first starting to design a game, several factors must be considered. A genre must be chosen. With VectorForce, we determined the genre through knowing what type of gameplay we wanted. We wanted simple, action-heavy gameplay. We also wanted a game without a lot of complexity. A gimmick must also be chosen. Try to determine one or two things the game will do differently than others in the same genre and market. Finally, we wanted a game that was simple to pick up and play and didn't require a large time investment. These three factors led us towards the Shooter genre with a space setting. From there, we chose our top-down camera perspective, which defined our game in the sub-genre "Top-Down Shooter". We also had to consider our limitations. We were working primarily with a student-based development team and their interests usually dictated what part of the game they wished to work on. We also had to consider the tools at our disposal, which we could use to craft the game.

2.1 Choosing the Gimmick

Because the Shooter genre is a classic genre with an overabundance of titles, we had to determine how VectorForce was to distinguish itself. We chose two gameplay elements to do this. The first was incorporating two different primary weapons for the player to use. This idea stemmed from the initial choice of using the Xbox 360 controller to control the game. The controller has two triggers, and felt we could put them to good use. We extended from this the idea of using the weapons in combination with each other to produce interesting destructive effects that were too powerful to use as primary weapons and that were more visually impressive than the primary weapons. From here, we decided upon three different primary weapons, which could combine into nine different combo weapons. Figure 1 provides an example of one of the early combo weapons in action.

Our second gimmick was derived from a multitude of other games and genre conventions. We termed this our "Superweapon". This was a weapon that could be used when the player had destroyed enough enemies to fill up a bar meter. When used, the superweapon would help protect the player and devastate any enemies on the playing field. We felt that the superweapon would be a good periodical reward for a player that was performing well, would help the player escape any tight spots, and would provide the added boon that giant lasers and explosions are always a bonus for players.


Kills: 19
FPS: 60.67
Objects: 19
x 2
California State University San Bernardino - Department of Computer Science
Figure 1. An early combo weapon created for Vector Force

To add variety to the game, we decided to allow the player to choose from three ships, each with their own superweapon. The game was quickly becoming about choices, so we felt this was a good direction to follow. Initially, only the superweapon and player model were the differing factors between player ships. This changed during later design. The player ships we designed were the Red Sparrow, the Black Scarab, and the White Hornet. We gave the Red Sparrow a multi-directional missile release for its superweapon. The Black Scarab was aptly named for its buzzsaw shield. The White Hornet was granted a giant laser that was unrivaled in terms of destructive power. At the end of the first iteration, these weapons were not balanced against the enemies that were in the game. We still felt they were fun, rewarding, and that the themes fit the game.

2.2 Developing Under Limitations

VectorForce was a project that was primarily student-driven, with professors guiding the students in the technical areas. Because of this, there was a very strong drive to see the game through to completion and an unrivaled passion throughout the development process. There were, of course, drawbacks to using a primarily student worker base. Student schedules limited the time they could spend working on the project. The enthusiasm students generated and the knowledge gained while working on a game project counterbalanced this, as well as the fact that students generally possessed a better grasp of current games from first-hand experience.

2.3 Disciplines

Video game development requires a mix of different disciplines, such as design, programming, and art. In a student-driven project, each student will naturally take an interest in one of these disciplines and will go to great lengths to learn it. Our approach with VectorForce was to allow students to determine which area of the project they wanted to contribute to. In our experience, many students gravitated towards programming, with fewer students interested in any type of game design, and only a few interested in art. Those students who were interested in art usually had a background in 3D modeling.

2.4 Tools of the Trade

VectorForce's graphical assets were built using three different modeling programs. The models were created and animated with Autodesk 3D Studio Max, four of which can be seen in figure 2. Several models were created by dedicated 3D modeling students, while others were contributed by amateur modelers in the CSCI 440 (Game Design) class. To texture the models and create the 2D graphical assets, we used Adobe Photoshop and the GNU Image Manipulation Program GIMP. Again, students of varying levels of ability created these assets as they were needed.

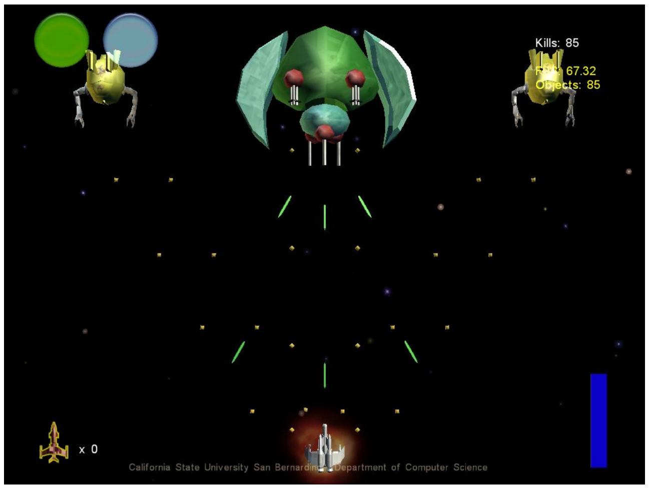

Kills: 85
67.32
Objects: 85
x 0
California State University San Bernardino Department of Computer Science

Figure 2. The original boss created for Cybershot

To program the game, we used Microsoft's Visual Studio programming environment. This naturally interfaced with Microsoft's XNA tools and libraries, which allowed us to create the game for the Xbox 360.

3. FINAL GAME DESIGN

As VectorForce grew through the contributions of students of the game design course (CSCI 440) and also from students outside the course yet dedicated to the project, revisions to the initial game design started and new goals had to be determined. Around this time, Microsoft had just announced its Indie Games movement through Xbox Live Arcade. With the recent news, we determined our final goal: to be published under the Indie Games label. To accomplish this, we went back and redesigned several of the weapons, added new features, determined how many levels we wanted in the game, and polished the game in preparation for releasing the game to the public.

3.1 Refining the Weapons

In the initial design of VectorForce, many of the combination weapons created shared similar properties, such as enemy tracking and projectile type. We decided to use these properties to define each of the primary weapons. We determined four different properties, which were Bullet, Missile, Laser, and Magnet. This increased our primary weapon count to four and our combination weapon count to 16. Once we created this design model, we followed through by matching existing combo weapons to their primary weapons. We kept the weapons that fit the new model. The ones that did not were redesigned to maintain the spirit of the weapon but merge with the properties of its primary weapons. Finally, we polished the weapons until they worked effectively and produced satisfying visual and destructive effects. The effect of these design considerations can be seen in figure 3.

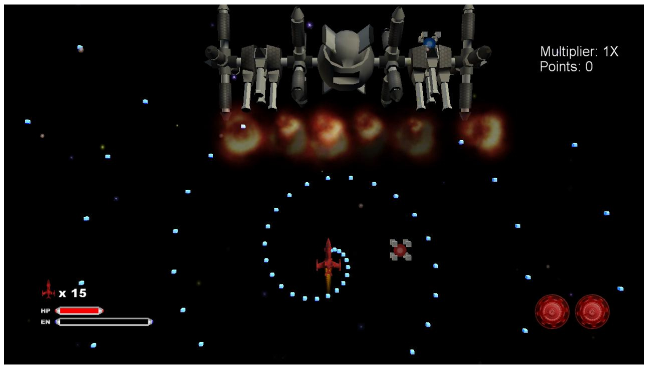


Figure 3. A new combination weapon created during the second iteration of VectorForce

One decision we made early in the gameplay refinement process was to retain the superweapon's thematic similarity with their early counterparts. By extension, we maintained the player-controlled ship's appearance, only retexturing them to be seen more easily against the black backgrounds of the levels. We increased the power of the Red Sparrow's missile pod by making it fire several times in a spiral pattern. We toned down the Black Scarab's buzzsaw shield by lowering the damage it could inflict and by shortening its duration. Finally, we narrowed down the giant laser the White Hornet used, but allowed it to destroy projectiles, providing the ship with a small defensive boost when it activated its superweapon.

3.2 Level Design

With our new goal of releasing the game, we needed to determine how long the game was going to be. Games of the shooter genre are not known for their length, so we decided that five levels would be appropriate for the gameplay we were offering and the enemy types we had created. Each level was designed by hand, with enemies appearing in "waves". Waves were a combination of enemy types with pre-scripted behaviors that appeared at time intervals in each level. This allowed us to create complex enemy spawns that could each present the player with a particular challenge. Finally, to add some replay value to the game, we created a "Challenge" level. This level is an infinite level, where the only goal is to earn as many points as possible and survive for as long as possible. All enemy waves could be spawned in this level, in any order and combination, as well as completely random boss enemy spawns.

Not only did we have to determine what our levels would look and play like, but we had to estimate how they would end. We came to the conclusion that we needed to create five different bosses to end each level, each with different and unique behaviors. We started this process by asking our modelers to create large enemies that they thought were interesting and that had plenty of weaponry. Each time we received a boss model, we reviewed what it could possibly do. We then created firing patterns and chose projectiles that punctuated the boss' uniqueness. To give an example, we first received the Mohawk, which eventually became the boss of the second level in the game. This boss can be seen in figure 4. The model was wide, with symmetrical armaments consisting of one large cannon and two smaller cannons. Our designers took this model and analyzed what types of attacks they could give it. They experimented with rapid shooting patterns, asymmetrical movement, and other attack types. They finally decided on using a back-and-forth movement pattern, with a constant stream of bullets that tracked the player with downward laser fire. Intermittently, the boss would stop firing for a few seconds, then unleash two very large sustained lasers which could potentially box a player between them. One final element we added was battle damage. When the boss endured enough damage, one cannon was set aflame and its firing pattern changed slightly. One large laser would not fire, but the other continued to fire normally. When both cannons were damaged, the speed of the firing pattern and movement were increased until the boss was eventually defeated. Though the development team liked this pattern changing element, we regret that we were not able to create more bosses with this element.

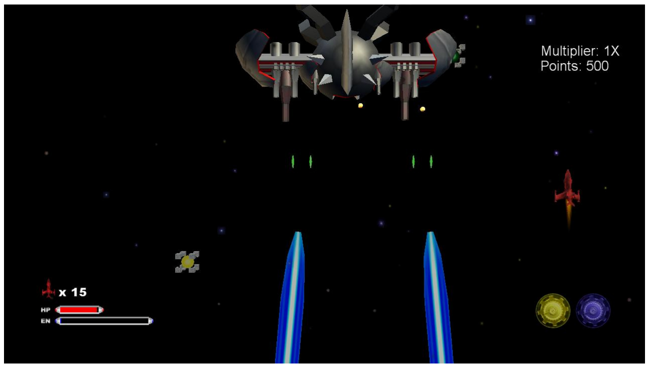


Figure 4. The Mohawk, VectorForce's second boss

3.3 Tools, Revisited

New goals required new tools to be used and existing knowledge of previous tools to be refined. Any model created within 3D Studio Max was scrutinized. Several models were polished and retextured, with a couple others being scrapped and recreated. In addition to 3D Studio Max, one student used IronCAD to create several impressive enemies.

One piece of the video game puzzle we were missing in early game design was quality music and sound effects. One brave student elected to fulfill the task. Using Apple's Garageband software, he singlehandedly created every piece of background music in the game. In addition, he helped provide the sound effects used in the game.

3.4 Documentation

When we proceeded with VectorForce, changes started happening daily. A weapon's damage or effect would be altered, enemies would be given a different pattern to follow, or the way something worked would be changed. Most, if not all of the changes we made to VectorForce through the final development period were not properly documented. This was a mistake.

The original game design document was never updated to reflect the changes we made during this time. Time was a big factor in the failure to document the changes we were making. At the time, we had no method for easily documenting changes and we did not try to rectify that. Because changes were happening daily, it would have been impossible for one person on the team to effectively gather a list of changes made during that day and incorporate it into a document. Another factor in the lack of formal documentation was that no one wanted to do it. Everyone on the project had other commitments besides the project. If a hasty change was made, all of us hoped that an e-mail was sent out detailing the change or the person at least notified the other team members that a change had taken place. The final factor in this was lack of experience in creating game design documentation. We did not know how some changes should be documented and that frustrated us.

Instead of formal documentation, we used quickly created lists of items that needed to be completed. If some element of the programming needed to be finished, it was written down on a piece of paper and, usually, it was addressed by the end of the day. If a game element had to be redesigned, time would be set aside to discuss it--also usually done within the span of a day. The final product resulting from these discussions was, again, a few notes jotted down on a piece of paper or a whiteboard and sent to the person who would be performing the changes.

We believe the biggest change in the project that never received any documentation was the addition of the Gatling Cannon, a weapon built into the basic controls of the ship, which could destroy enemy projectiles. The weapon was conceived and implemented by one person and the rest of the team was notified after the fact. Thankfully, this change was not destructive and did indeed help define the gameplay feel for the second half of VectorForce's development, but the fact still remains that it was never fully discussed before implementation and was never in the original design for the game.

It was also difficult to track assigned tasks. We all had our general areas of expertise on the game, but many times, two people would be asked to work on the same piece of the game without knowing that someone else was working on the same piece. This lack of organization created overlap that usually left some work unused. With proper documentation, this may have not been the case.

It was only by having a small, closely-knit team that we were able to effectively push forward with development without much hindrance. Our advice is to find time for documentation. Find a way for all of the members on the development team to record their changes and have it be easily accessible to everyone on the team. This helps trace completed and continuing assignments, and how design choices should be implemented.

4. PLAYTESTING

During the late stages of development for VectorForce, we subjected the game to two types of playtesting: internal and external. Playtesting is the true test of separating gamemakers from their product. To do this, we had to step back and objectively look at each piece of the game. When we determined areas we needed to improve, we invested more effort to make it better.

4.1 Internal Testing

Internal testing consisted of handling bugs and crashes, as well as gameplay balancing. Here, we tried to break the game in every way possible. Many times, we succeeded. Each time a bug or crash was found, we identified what caused it and subsequently fixed it. Each time a weapon or enemy malfunctioned, we fixed its behaviors until we were satisfied with it. Our programming experience lent itself to finding and fixing conventional programming bugs, and helped stabilize the game through its many changes. Our eye for detail helped determine when a gameplay element was not working as intended.

We also considered and improved gameplay elements that did not stand the test of multiple playthroughs. Some weapons were created with too narrow of a scope of function and had to be expanded and tweaked to make them more effective. Here, nothing was completely redesigned. Usually it meant adding more projectiles to a weapon, changing its rate of fire, changing its damage, or changing how it acquired targets. Mostly, the work consisted of balancing the power of the weapon and not changing the weapons themselves. Another area was enemy wave design and cohesion. If a particular section of a level proved to be deadly to everyone playing through it, we restructured that part of the level. This meant either going into the waves used in that part of the level and making them less dense or figuring out a better combination of waves to use. This was aimed at lessening the frustration level of the player through changing or lessening the difficulty of that section of a level. Overall, our internal testing prepared us for the game's next phase of testing.

4.2 External (Community) Testing

As a part of submitting a game to the Xbox Live Marketplace, an Indie Game must go through a community review process in which other game developers download and play the game. In this process, the reviewers rate for game stability and give feedback on gameplay.

As mentioned before, we had experience related to keeping a program stable. What we did not have experience with was keeping the game stable on the XBox 360 console. During this process, we had to investigate the 360's architecture in order to solve many issues that arose during community review. The biggest issue to tackle was garbage collection. The .NET garbage collector on the XBox 360 is not as good as the one for Windows. As a result, the game would frequently stall during garbage collection phases. We solved the problem by caching and re-using objects created within each level of game play. Another problem was the n^2 time complexity of our collision detection system, which slowed the frame rate when many projectiles were present at the same time on the screen. We solved this problem by re-implementing the collision system using a binary space partition, thus effectively reducing time complexity to n * log(n). We also improved the frame rate by performing collision detection and rendering concurrently in the separate threads. Another significant problem was the variety of TV screen sizes and resolutions; we had to figure out ways for the game to properly display in as many resolutions as possible.

Other issues we had to tackle concerned difficulty and balancing. Many times, we received feedback indicating that the game was too difficult. Sometimes, it was an issue of player firepower. Other times, enemy formations and boss behaviors were too difficult to overcome and would trap a player who was not perfectly prepared. The biggest issue we faced was the issue of damage indication, both in terms of enemies taking damage and the player taking damage. Here, we learned that if a player does not immediately notice that they are taking damage, they will not take measures to avoid damage. Additionally, if a player does not know or feel that the enemies are taking damage, they will get frustrated with how long it takes to dispatch an enemy regardless of time spent.

In the end, it took seven submissions for review before our game was stable enough and technically sound enough before we were accepted onto the Xbox Live Marketplace.

5. CONCLUSION

Designing a game is an iterative process that requires planning and dedication. There is no question that a game elicits student participation unlike any other task, and that the creation of one is vital to a meaningful videogame development curriculum. It allows students to not only see what tools are available to them, but also how certain technologies work and how a team is structured in order to create a final product.

More importantly, implementing a game allows students not only the pride of accomplishment, but also a concrete education. Throughout the entire game creation process, we were able to solidify and test our understanding of game design. We learned the importance of choosing a genre and understanding its conventions in order to determine how to find a niche. It was also important to analyze what resources were available and how limitations affected game design. While a large company with hundreds of employees might be able to produce any game, a small college collaboration will usually need creativity in dealing with limited resources. Luckily, our lack of personnel led us to reach out to other departments on campus, allowing us to see that there are other aspects to game design besides programming, such as adhering to a coherent art design.

Another vital aspect of game design was balancing our game mechanics after production began. We discovered that the initial weapon design had evolved as we added more weapons, so we needed to find a new model to group the weapons' functionalities. Though the process partly taught us how to organize mechanics better from the beginning, the more important lesson we learned is that not everything can be planned ahead of time and that some aspects have to be refined during actual game implementation. This applied equally to the length of the game; it was not until we knew how the game played that we were able to determine how long the levels in the game should be and what challenges we could provide the player. We also reached the understanding that we needed models that were already built before we could start creating refined and polished products.

Even when we believed we were finished with the game, we still needed to learn how to deal with constructive criticism. The game was intended for a wider audience, so receiving feedback from people with no stakes in the game was important. The most important element community feedback can provide is an ongoing dialogue of effective game design elements. Anyone planning to build a game may have preconceived notions of what constitutes a good game, and some ideas about how to build the ideal game, but it is not until the game is built that a person can truly tell whether the product they built correlates to game design ideals.

We are currently embarking on a new, more ambitious project –an MORPG (Multiplayer Online Role Playing game) named Mythic, which resembles the recent hit-game Borderlands. We started development on the new game with a number of learned lessons. We were familiar with how a game is structured, what tools are available for development, and how to bring various facets together. There was already an understanding that in a more ambitious game we would need a bigger team: more programmers, more artists, and even a few writers. It was also apparent that we would need the same type of structure that had helped us work as a team, as well as a consistent way for all of us to communicate. We also started a new project with the learned lesson that we would need to choose a genre, develop balanced game weapons, enforce consistent level design, and even be prepared for the eventual step of game testing.

Along the way, we are also learning new lessons. We are learning that more complex games require more complicated programming methods and tools, so we have now shifted to the powerful Unreal Engine for level design, while we are also making a long-term investment in our own graphics engine. There is also the fact that while we might have required a few static 3-D ship models for VectorForce, we now require animated humans, complete with sophisticated humanoid meshes and rigged skeletons. Game design is also different: whereas VectorForce prompted the player along with the promise of higher scores and more challenging levels, our new game incorporates story and requires us to provide compelling quests and characters that the player will care about. Our combat system is not the simple shoot-and-dodge mechanism of VectorForce; we now incorporate combo attacks, damage mitigation, close range and long-range attacks, and enemy stun time. All of these new gameplay aspects require individual attention, so we need a bigger staff.

We have started to reach out to our local high schools, community colleges, and other departments at our university to gather talent in the creation of our new game. Wherever we go, we find students excited about creating a game and in learning the tools behind it. Luckily, we have managed to receive funding for our program under an NSF CPATH grant intended to help computer science departments across the country maintain high computer science degree enrollments. The intent is to teach other disciplines the value of Computational Thinking and the role of technology in America's future. We can already see the influence that our department has on other institutions and the way that willing students learn about technology. A number of students already begin to see the advantages that a computer science degree can have in their future, and they see how tangible such an education is. The creation of a video game is not just a diversion. It is a means to attract students and explain fundamental computational concepts to them.

6. ACKNOWLEDGEMENTS
The VectorForce team would like to thank all the students who, directly or indirectly, helped contribute to VectorForce's development and success. We would also like to extend thanks to everyone who gave us feedback in all stages of development.

7. REFERENCES
[1] Bayliss, J. D. and Bierre, K. 2008. Game design and development students: who are they?. In *Proceedings of the 3rd international Conference on Game Development in Computer Science Education* (Miami, Florida, February 27 - March 03, 2008). GDCSE '08. ACM, New York, NY, 6-10. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1463673.1463675

[2] Bayliss, J. D. and Strout, S. 2006. Games as a "flavor" of CS1. *SIGCSE Bull.* 38, 1 (Mar. 2006), 500-504. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1124706.1121498

[3] Cliburn, D. C. and Miller, S. 2008. Games, stories, or something more traditional: the types of assignments college students prefer. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 138-142. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1352135.1352184

[4] Gestwicki, P. and Sun, F. 2007. On games, patterns, and design. In *Proceedings of the 2007 Symposium on Science of Design* (Arcata, California, March 22 - 24, 2007). SoD '07, vol. 364. ACM, New York, NY, 17-18. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1496630.1496640

[5] Rankin, Y., Gooch, A., and Gooch, B. 2008. The impact of game design on students' interest in CS. In *Proceedings of the 3rd international Conference on Game Development in Computer Science Education* (Miami, Florida, February 27 - March 03, 2008). GDCSE '08. ACM, New York, NY, 31-35. DOI= http://doi.acm.org/10.1145/1463673.1463680

[6] Whitehead, J. 2008. Introduction to game design in the large classroom. In *Proceedings of the 3rd international Conference on Game Development in Computer Science Education* (Miami, Florida, February 27 - March 03, 2008). GDCSE '08. ACM, New York, NY, 61-65. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1463673.1463686

[7] Zagal, J. P., Ladd, A., and Johnson, T. 2009. Characterizing and understanding game reviews. In *Proceedings of the 4th international Conference on Foundations of Digital Games* (Orlando, Florida, April 26 - 30, 2009). FDG '09. ACM, New York, NY, 215-222. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1536513.1536553

[8] Zyda, M. 2009. Computer science in the conceptual age. *Commun. ACM* 52, 12 (Dec. 2009), 66-72. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1610252.1610272