AN ALGORITHMIC AND SOFTWARE ENGINEERING BASED APPROACH TO ROBUST VIDEO GAME DESIGN

Hollie Boudreaux[1], Ashok Kumar[2], and Jim Etheredge[2]

[1]Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, Louisiana, USA

hmb7226@louisiana.edu

[2]Department of Computer Science, University of Louisiana at Lafayette, Lafayette, Louisiana, USA

axk1769@louisiana.edu, jne1390@louisiana.edu

ABSTRACT

Design and development of a large video game is a challenging software engineering and project management problem. Thus, it is a non-trivial task. This paper describes the design and development of a game, entitled N-STAL, which embodies nearly every aspect of game design and implementation, including researching, crafting of assets, selection and usage of proper tools, software development, testing, and team play. Key software engineering principles are followed throughout the design, development, and validation stages. Successful execution of such design and development in an academic setting inspires team-based learning in students. It challenges them to meet project deadlines, prepares them for life-long learning, and helps them understand some of the challenges that are faced with modeling, simulation, and user interfacing.

KEYWORDS

Game Design, Game Loop, Collision Detection, Collision Avoidance, Software Development

1. INTRODUCTION

The design and development of a large video game is a challenging software engineering problem and thus a non-trivial task. To make the venture successful requires taking a systematic and methodical approach to issues such as identification of story and characters, design of art including sound and artistic assets, game loop algorithm efficiency, and robust design and implementation of the software that follows the key principles of software engineering. In addition, building a co-operative, possibly cross-disciplinary team, and ensuring that the team functions smoothly, adheres to the design principles, and meets the project deadline is of critical importance. Further, ensuring that the end product is fault-proof, user-friendly, intuitive, and entertaining is the ultimate milestone. The proposed work is done with these themes as its central objective. Inclusion of students, utilizing their talent, and guiding them to build their software engineering and game development skills is accomplished during the course of this project. The work is done as a part of the most advanced video game design project in the senior year.

This paper discusses the design and development of a game entitled N-STAL which was designed and developed as a part of an advanced course project, in an undergraduate course, in the Computer Science Department of the University of Louisiana at Lafayette. Six computer science students, four artists, and one musician were involved in the project. This paper presents the work largely from the design and development perspectives of software engineering. The

DOI : 10.5121/ijsea.2011.2303

rest of the paper is organized as follows. Game specification is described in section 2; time line for the project is described in section 3; proposed design and implementation is described in section 4; related works are discussed in section 5, and conclusion is provided in section 6, followed by references.

2. GAME SPECIFICATION

2.1. Story

In N-STAL, the user plays the role of an explorer sent to a strange planet to search it for mineral deposits and new types of plants. Upon arriving at the planet, however, he discovers that the planet once held intelligent life – buildings lie in ruin, and a large mechanical robot sits dormant. In addition, the planet is inhabited by a mysterious robot security system (named Boss). Many smaller robots are still functional and performing their primary function: protecting the planet from intruders. The player must choose to avoid or combat these robots while exploring. The explorer is equipped with only a hoverbike and a simple weapon and must search the planet in order to find a way to escape.

The team that created N-STAL wanted to make a game that was modeled after and inspired by games that featured more exploration and less outright combat or enemy encounters, such as Shadow of the Colossus [1]. The intent was to focus on one or two large encounters that would be fantastic, as opposed to having the player combat many smaller enemies that would be less interesting or exciting to fight. The team felt that the player would be more impressed and challenged by something that was physically imposing. Video of the game being played can be found on the UL game concentration's YouTube channel [2].

2.2. Sound and Art Specification

Sound assets were developed by the musician available to the team. These assets include sounds for the robots shooting and dying, the player character shooting and being hit, and different types of background music.

A group of four artists were available to the team. They were tasked with creating assets such as models for the character, the bike, and enemies. The user interface and the game environment were also created by the artists.

Figure 1. The player character and hover bike

Their main character of N-STAL was modeled not after a soldier, as is typical in such games, but an explorer. Inspired by Steam Bout, a game created the previous year by the students at our university, a psychological profile of the main character was created and passed to the artist team. Figure 1 shows the main character and his hover bike. The story and game path were created in parallel to ensure that there were no conflicts. In addition to the main character, the artists devoted much of their time to the primary "boss" model, a large, quadruped robot. The boss model, which can be seen in Figure 2, is so large that parts of it are often obscured by fog.

Figure 2. The boss robot.

Though an increased focus on story was created for the game, some parts of the story were curtailed due to time constraints in favor of perfecting the gameplay. Features successfully incorporated into the final product are: the force fields, final boss, a slight exploration aspect, and the core weapon and vehicle ideas. Even though some of the other features had to be abandoned due to time constraints, the finished product followed the original proposal very closely.

2.3. Software Engineering Principles

A key specification for the development team was to develop the software using robust software engineering principles, such as reliability, correctness, reusability, and verifiability.

3. Project Development Timeline

3.1. Initial Requirements

The first week of class, the programmers are split into two teams and given the list of game requirements. This list includes, but is not limited to:

- The software must have a title screen and ending screen.
- The software must have at least one completed level.
- The content must not exceed a PG-13/T rating.

3.2. Game Proposal Document

By the end of the second week, the first draft of the game proposal document is due. In this document, the team describes their idea and the features they intend to integrate into the software. The features list is divided into two groups: a minimum list of features that will be in the game, and a larger list of those features that will be implemented if time permits.

3.3. Ogre Tutorials

While the teams are creating their game idea, the programmers are required to go through the tutorials on the Ogre website to become familiar with the graphics engine.

3.4. Development Period

After the game proposal is approved, the development period begins. During this time, the teams communicate with the artists and musicians about art and music assets, develop code, conduct group meetings, and participate in weekly status reports with the professor.

3.5. Project Demo

The final demonstration is held as a seminar open to the public. Each team presents their final product and discusses the game concept, architecture of the software, and the decisions made.

4. DESIGN, IMPLEMENTATION, AND VALIDATION OF THE PROPOSED GAME

This section discusses the design and development of the software, including the tools used, such as Ogre.

4.1. Game States

The types of states used in the game include, menu, play, pause, credits, game information, win, and dead. Each state is described below. A diagram showing the connections between states is shown in Figure 3.

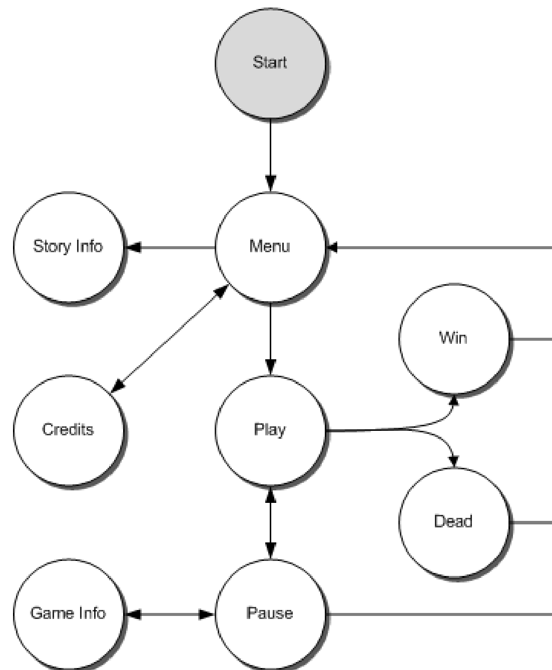

Figure 3. Connection between game states.

- The **play** state is the state in which the player can move the main character around and explore the game world.
- The **credits** state displays information about the team members.

- The **menu** state is the main menu that appears upon starting the game. The credits state can be reached from here, as well as the play and game information states.
- The **pause** state can only be reached from the play state. Being in this state gives the player the option of continuing game play, via returning to the play state, or exiting the game.
- The **story information** state displays the story and game controls. This state can be reached from the main menu.
- The **game information** state displays the game controls. It can only be reached from the pause state.
- The **win** state is reached when the player defeats the boss robot. It shows a congratulatory message and allows the player to return to the menu state.
- The **dead** state is reached when the player is defeated. A conciliatory message is displayed and the player is allowed to return to the menu state.

4.2. Managers

Managers were used to control all objects in the game. Each type of manager was implemented as a separate class. Regular enemies all shared a generic enemy manager, while objects that only have one instance, such as the boss and main character, had a separate game manager.

Each manager class contains the following functions:

- **Initialize** – This functions creates the appropriate game objects and initializes any necessary parameters.
- **Update** – An object's animation is updated. If the object is firing a weapon, that is also handled here.
- **Movement** – Moves the object in the direction it is currently facing.

4.2.1. Enemy Manager

When initialized, the enemy manager creates a vector of enemies. For each enemy, the following operations are performed during the update cycle if the current state requires that enemies be updated:

- Check collisions between the enemy and other objects. This only checks for collisions; the processing of collisions happens in a later step. Pseudocode for checking collisions between enemies and the main character is shown in Figure 4.
- Allow gravity to pull the enemy towards the terrain if the enemy is not on the ground. This is necessary if the enemy walks off a ledge in pursuit of the player.
- Determine if the enemy can detect the player. This is accomplished by creating an Ogre sphere object around the enemy and using a built-in function that checks if the sphere intersects with the player. If the player is within the sphere the enemy enters aggressive mode.
- If the enemy is in aggressive mode, it will attempt to move towards the player and attack. If the enemy is not aggressive then it does nothing.

4.2.2. Boss Manager

There is only one instance of the boss in the game. When the game begins, the boss is inactive, mostly underground, and protected by a shield. When the shield is disabled by the player, the boss slowly rises accompanied by a sound effect and particle system until it is completely above

ground. The boss then begins to move and attack the player until either it or the player is defeated. Regardless of where the player is, the upper part of the boss will face towards the player. Pseudocode for the boss update function is shown in Figure 5.

```
Collision Detection: Checking enemies against the main character
Create an Ogre sphere object around the main character just large enough to contain it
For each enemy in the enemy list:
    Create an Ogre sphere around the enemy just large enough to contain it
    Use the built-in intersection function to determine if the two spheres intersect
    If there is an intersection:
        Create a vector between the enemy and main character
        Move the enemy in the opposite direction of the vector
```

Figure 4. Pseudocode for the collision detection function

```
Boss Manager Update Function
If the boss is dead
    Play death animation
Else if the boss has been activated
    If the boss has not completely risen
        Play rumble sound
        Display smoke particle system
        Move the boss up
        If the boss is completely above ground
            StartFight = true
    If startFight = true
        Play boss fight music
        Move the boss along the preset path
        Turn the upper section to face the player
        Attack the player if in range
```

Figure 5. Pseudocode for the boss manager update function

4.3. Game Loop

The game loop always starts with processing the current state of the keys. In this step, several buttons are polled that will cause a change in the game state if they are being pressed. For example, pressing “Esc” or “P” while in the play state will switch to the pause state. Next, the current game state is checked. The steps that are taken at this point are dependent upon the current game state. An example of the game loop when in the play state is shown in Figure 6 below. Some of the actions shown in the figure, such as updating sounds, will occur regardless of the current state.

If the game is in the play state, sounds, such as a weapon firing and the background music, are updated first. Next, the camera is updated based on the player's movement. Collisions between the player and other game objects are checked next. If the player is colliding with any object, he is pushed backwards over the next several frames until he is no longer colliding with anything. Finally, all game objects are updated via their manager.

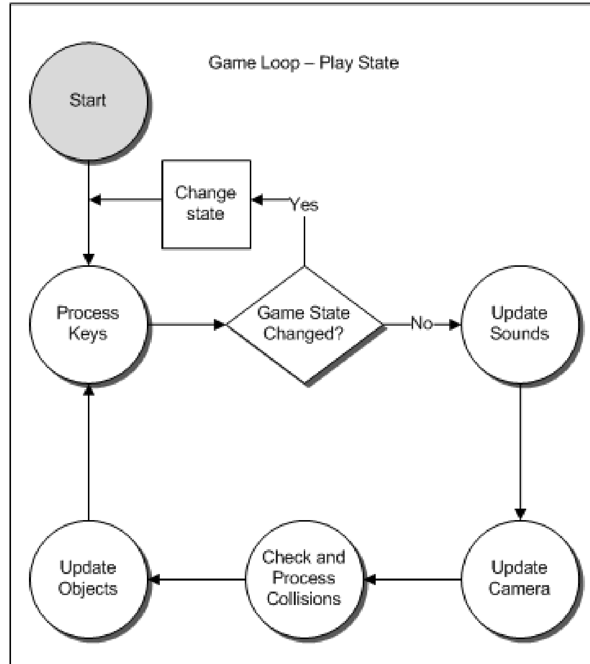

Figure 6. Flowchart of the game loop while in the play state

4.4. Implementation

The software was implemented using Ogre, CEGUI, and irrKlang. Screenshots of the implemented software are shown in Figures 7 and 8. Selected code segments from the boss manager and enemy manager are shown in Figures 9 and 10, respectively.

Figure 7. Aggressive enemies moving towards the player.

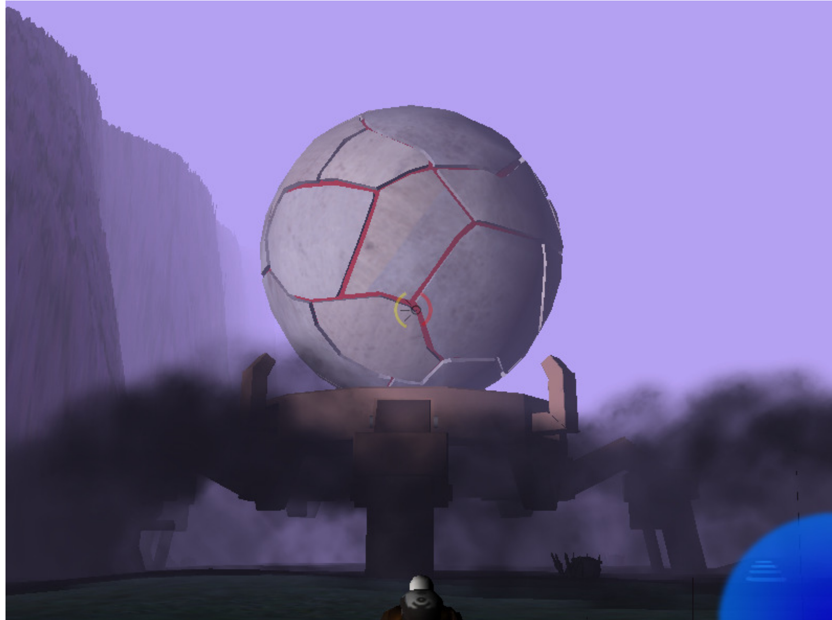

Figure 8. Activated boss rising from underground

```
void update(const FrameEvent& evt, MainChar *Robert)
{
...

if(active)
  {
  Vector3 loc = bRoot->getPosition();

  if (loc.y < 0)
   {
   if (! engine->isCurrentlyPlaying("../../media/sounds/bossrumble.wav") ) engine-
>play2D("../../media/sounds/bossrumble.wav", false);
   mSceneMgr->destroyParticleSystem("ForceFieldEffect");
   mSceneMgr->getParticleSystem("bossdirt")->setVisible(true);
   bRoot->setPosition(loc + Vector3(0,20 * evt.timeSinceLastFrame,0));
   }
  else if (startFight)
   {
   if (engine->isCurrentlyPlaying("../../media/sounds/bossrumble.wav") ) engine->stopAllSounds();
   if (! engine->isCurrentlyPlaying("../../media/sounds/boss.wav") ) engine->play2D("../../media/sounds/boss.wav", false);
   mSceneMgr->getParticleSystem("bossdirt")->removeAllEmitters();
   movement(evt.timeSinceLastFrame, Robert);
   lineOfSight();
   attacks(evt.timeSinceLastFrame);
   LookAt();

   ...
  }
```

Figure 9: Sample code from the boss update function.

4.4.1. irrKlang

irrKlang [3] is a high level, cross-platform sound engine and audio library. It is usable in C++ and all .NET languages and is free for non-commercial use.

```
void update(Real timeSinceLastFrame, MainChar *Robert)
 {
 isCollide = false;
 while(!eList.empty())
  {
  ptr = eList.back();
  if(ptr->alive)
   {
   ptr->Collide = false;
   collision(ptr, timeSinceLastFrame, Robert);
   gravity(ptr);
   lineOfSight(ptr);
   if(!ptr->Collide) movement(ptr, timeSinceLastFrame);
   attacks(ptr, timeSinceLastFrame);
   }
  death(ptr, Robert);
  tList.push_back(ptr);
  eList.pop_back();
  ptr->animState->addTime(timeSinceLastFrame);
  }//while
 mproj.update(timeSinceLastFrame, Robert);
 eList = tList;
 tList.clear();
 }//update
```

Figure 10. The enemy update function

4.4.2. CEGUI

Crazy Eddie's GUI System (CEGUI) [4] is a free library that provides windowing and widgets. The library is object oriented, written in C++, and is multi-platform. CEGUI provides renderers for OpenGL and DirectX, as well as graphics engines such as Ogre. It uses XML scripts to describe the user interface layout and objects. The functionality of these objects is determined by the application code.

4.4.3. Ogre

The students used the Object-oriented Graphics Rendering Engine (Ogre) [5] to develop their game. Ogre is a free, open source graphics engine. It is not a game engine, although many development teams have produced commercial titles with it. Ogre is written and maintained by a small core group of programmers and contributed to by a growing online community which also maintains a thorough wiki [6].

A main draw of Ogre is that it can be combined with external libraries for things such as sound, particle systems, and physics, to create a game engine customized to the requirements of a project. These libraries are not included by default, or even required to be used, but Ogre provides a simple interface for integrating these libraries into an application. This use of additional external libraries is good practice for real production situations because graphics or game engines will rarely have all the required functionality built in.

After the initial adjustment period, the team found it easy to use. A general complaint was that they did not have enough time to get used to Ogre before needing to use it, as understanding a graphics engine is not an easy task. Unfortunately, the team only has a sixteen week semester to develop their game, and as a result the team members are forced to learn Ogre along the way.

4.5. Validation

The software consisted of a large code base. This section describes how the game was tested and software engineering principles were adhered to.

- Correctness – The minimum requirements for the game were specified by the professor at the beginning of the semester. At the final demonstration, all the required elements were checked for and points were deducted from the final grade for anything not present.
- Reliability – The intended functionality of the software is to allow the user to explore the game world, gathering weapons and turning off the generators to release the boss. The player then has to defeat the boss in order to win the game. The reliability of the software was tested by having many students play through it multiple times.
- Usability – When the game is executed, there is an option on the main menu screen to view the game controls. N-STAL uses a control scheme that is standard for similar games, making it easier for new users to become accustomed to using the software. A screenshot of the game information screen is shown in Figure 11.
- Efficiency – The software uses models with low polygon counts which render quickly due to the low number of triangles required. High resolution textures are applied to the models to give them the appearance of a model that uses many more polygons without the performance impact.
- Portability – N-STAL was primarily developed on Windows XP using Visual Studio 2008. However, once the program is compiled, and an executable file is created, the game may be played on any Windows XP system with an adequate video card. This is accomplished by packaging all the necessary .dll files in the same folder as the .exe file and including all the required resources. In addition, Ogre is multi-platform. The code could be recompiled on Linux or MacOS and the software would work identically on those operating systems. Provided that the .dll files are the same on Windows Vista and/or Windows 7, the software should also be compatible with those operating systems without recompiling.
- Reusability – The software was not built with reusability in mind. However, the skeleton of the software consists of simple state transitions that could ported to another system and easily modified. Other elements, such as event triggers, would not be reusable except in very similar software.
- Robustness – Range checks were performed on variables to ensure that, for instance, the player character could not walk faster than allowed. Angle checks were also performed to stop the hoverbike from moving up steep cliffs. However, if a speed boost was used, that would allow the bike to move up some cliffs that it would otherwise not be able to ascend.

4.6 Team Organization and Management

The students elected a team leader, who was in charge of keeping the project on schedule. Due to the size of the software being developed, the team was required to use version control. Subversion (SVN) [7], an open source command line version control system, was used. In addition, the team was required to use Trac [8], which provides a wiki and ticket system for tracking of bugs, tasks, and more.

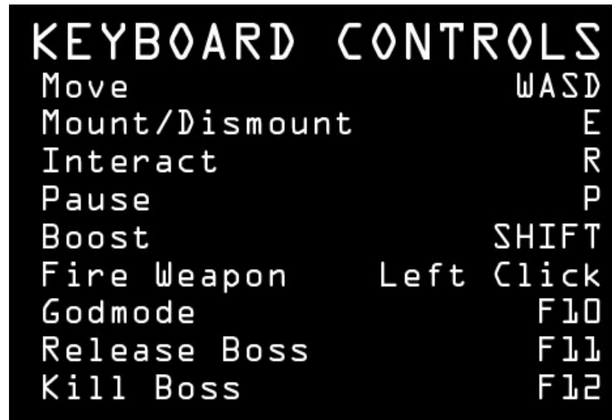

Figure 11. Screenshot of the game controls.

The use of SVN and Trac also emphasizes the team cooperation aspect of the course and fosters a collaborative environment. When students are not assigned tasks or given any direction on what part of the code to work on, check-in conflicts, where multiple programmers are attempting to change the same part of the code simultaneously, will occur often. Using Trac to coordinate which student is working on what section of the code base reduces these conflicts.

5. RELATED WORK

Bayliss and Bierre [9] describe a newly designed curriculum named Game Design and Development (GD&D) at Rochester Institute of Technology. The curriculum does have some overlap with the traditional Computer Science and Information Technology curriculums, however, the entire focus is on game design and development. The GD&D curriculum aims to appeal to a broader range of students, with possibly more female students than in traditional CS and IT curriculums. The GD&D curriculum provides three innovative initial programming courses for GD&D students via usage of Wii-controlled games and a supplied code base. It has been observed that GD&D students go on to take the regular course in C++ programming and do well in terms of performance as well as retention. Furthermore, upon completing the curriculum, the majority of GD&D students wished to become software developers in the gaming industry (44%) or game designers (25%). In other words, the curriculum inspired most GD&D students to become software developers and designers.

The work by Bidarra, et al. in [10] describes a course on game projects, taught in the second year, at Delfts University of Technology with a focus on game development in large teams using students from different disciplines. The proposed course's learning outcomes include demonstrating proficiency in applying media and programming techniques within the context of computer games, striving for a balance between the effectiveness of a programming technique and the desired quality of a game effect; describing the main modules of a game engine and purposefully using their functionality, deepening object-oriented programming skills while building a complex and large software system; an. developing and contrasting teamwork skills within the context of a realistic interdisciplinary team. The survey results reported in the paper indicate that the students were highly motivated upon completing the course and were largely happy with the projects.

Brown, Lee, and Alejandre [11] introduced a game design project in the department of computer science at Drexel University with an emphasis on developing team development skills in addition to soft skills. The students developed math games for middle school students with constant feedback from the target audience as the games were being developed. A team consisted of students from computer science, education, and digital media. The course entailed

not only planning, design, and development, but also demonstrating the product to the end user and taking their feedback into account for further improving the product.

Kessler, van Langeveld, and Altizer [12] describe a new interdisciplinary program at the University of Utah, entitled Entertainment Arts and Engineering (EAE). This program combines students from the School of Computing and the Division of Film Studies in order to teach both video game development and computer animation. The focus of the program is on shared classes where students from both Computer Science and Fine Arts study together and cooperate on game and animation projects. The program concludes with a yearlong capstone course in which the students work together to make a video game or animated short from scratch. Reports indicate that the program attracts more students and better equips them for a career in video games and animation.

Kanode and Haddad [13] highlight the major challenges encountered in game development, namely, creation and management of diverse assets (e.g., 3D models, textures, sound, music, dialog, and video), project scope (e.g., millions of lines of code that is developed by several hundred programmers who are located in different geographic locations; management of features; meeting timelines for development), game publishing (i.e., finding and getting the financial backing of a game publisher), team organization, and use of third party technology. Further, they emphasize that software engineering principles can be used to improve the process of game design, manage the project scope as well as to manage the teams better.

Ryoo, Fonseca, and Janzen [14] propose a problem-based learning curriculum centered on game development to deliver basic object-oriented programming concepts in an interactive and engaging manner. Their proposed approach consists of inception phase, elaboration phase, construction phase, and transition phase.

Baker, Navarro, and van der Hoek [15] propose an educational card game that simulates the software engineering process and is designed to teach those process issues that are not sufficiently highlighted by lectures and projects. Their proposed game was found to be favorable by their students and also it was reported to be moderately successful in teaching software engineering principles.

Claypool and Claypool [16] propose an endeavor to enhance interest and retention in an existing software engineering curriculum through the use of game-based projects which allow the students to actively participate in different phases of software lifecycle, exposes them to project and team management issues as well as the entire gamut of game design.

In [17], Ampatzoglou and Stamelos note that games present several characteristics that differentiate their development from classical software development and conduct a literature survey on research concerning software engineering for computer games. As a conclusion of their study, the authors propose employment of more elaborate empirical methods, i.e. controlled experiments and case studies in game software engineering research, which have not been extensively used in the past.

6. CONCLUSIONS

The design and development of the N-STAL game represents the embodiment of the objectives of the senior level gaming course offered as a core course in the concentration. Team members included programmers, artists and musicians working over an entire semester in as real a development environment as possible. While the final version of the game includes many of the features planned at the beginning, time constraints prevented the implementation of all features planned. The course has evolved over time from small teams to larger teams that include artists and musicians. The utilization of game object managers, game state transitions, and psychological profiling for character design significantly improved the game development process in terms of both development team management and feature set implementation.

Since the development methodologies used can never hope to overcome the inherent time constraints the end result will always be a "partial" game implementation. Future offerings of the course will address this problem by incorporating the continuation of development projects across multiple semesters with different teams and possibly adding creative writing students to the mix. This will allow for the development of more complete games and give students a more complete exposure to the realities of large game development projects.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Colin Baker, Christopher Hebert, Christopher Hoback, Philip Lanclos, Devin Rooney, and Philip Spear for their work in creating N-STAL.

REFERENCES

[1] Shadow of the Colossus. Official American Site: http://us.playstation.com/games-and-media/games/shadow-of-the-colossus-ps2.html

[2] N-STAL gameplay Video. http://www.youtube.com/watch?v=brvzoolAMRE

[3] irrKlang. http://www.ambiera.com/irrklang/

[4] CEGUI. http://www.cegui.org.uk/wiki/index.php/Main_Page

[5] Ogre Website. http://ogre3d.org/, 2009.

[6] Ogre Wiki. http://www.ogre3d.org/wiki/index.php/Main_Page, 2009.

[7] Subversion. http://subversion.tigris.org/

[8] Trac. http://trac.edgewall.org/

[9] Jessica D. Bayliss and Kevin Bierre. Game design and development students: who are they? In GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education, pages 6-10, New York, NY, USA, 2008. ACM.

[10] Rafael Bidarra, Jerke Boers, Jeroen Dobbe, and Remco Huijser. Bringing a pioneer games project to the next level. In GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education, pages 11-15, New York, NY, USA, 2008. ACM.

[11] Quincy Brown, Frank Lee, and Suzanne Alejandre. Emphasizing soft skills and team development in an educational digital game design course. In FDG '09: Proceedings of the 4th International Conference on Foundations of Digital Games, pages 240-247, New York, NY, USA, 2009. ACM.

[12] Robert Kessler, Mark van Langeveld, and Roger Altizer. Entertainment arts and engineering(or how to fast track a new interdisciplinary program). In SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education, pages 539-543, New York, NY, USA, 2009. ACM.

[13] C.M. Kanode, and H. Haddad, "Software Engineering Challenges in Game Development", Sixth International Conference on Information Technology: New Generations, ITNG '09, pages 260 – 265, 2009.

[14] J. Ryoo, F. Fonseca, and D. Janzen, "Teaching Object-Oriented Software Engineering throug. Problem-Based Learning in the Context of Game Design", 21st Conference on Software Engineering Education and Training, CSEET '08, pages 137-144, 2008

[15] A. Baker, A., E. Navarro, E., and A. van der Hoek, "An experimental card game for teaching software engineering", 16th Conference on Software Engineering Education and Training, CSEE&T 2003, pages 216-223, 2003

[16] K. Claypool and M. Claypool, "Teaching Software Engineering Through Game Design", Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, June 2005.

[17] A. Ampatzoglou and I. Stamelos, "Software Engineering Research for Computer Games: A Systematic Review", Information and Software Technology, Vol. 52, Issue 9, September 2010.

Authors

Hollie Boudreaux received the B.S. degree (summa cum laude) in computer science from Nicholls State University, Thibodaux, LA in 2004, and the M.S. degree in computer science in 2007 from the University of Louisiana at Lafayette, Lafayette, LA, where she is currently working toward the Ph.D. Degree. Her research interests include computer graphics, video game design and development, artificial intelligence, and multiagent systems.

Dr. Ashok Kumar is an Assistant Professor in the Department of Computer Science at the University of Louisiana at Lafayette. Dr. Kumar obtained his Ph.D. in 1999 and worked for four years in industry before joining academia full time. He has over fifty publications in refereed journals, conferences, and book chapters. He has served on the program committees of several conferences.

fifty publications in refereed journals, conferences, and book chapters. He has served on the program committees of several conferences.

Dr. Jim Etheredge received the M.S. degree in computer science from the University of Southwestern Louisiana in 1986 and the Ph.D. in computer science from the University of Southwestern Louisiana in 1989. He is currently an associate professor of computer science at the University of Louisiana at Lafayette, Lafayette, LA and the coordinator for the Video Game Design and Development concentration of the undergraduate computer science curriculum. His research and teaching interests include video game design and development, artificial intelligence, multiagent game systems, and database management systems.