NEEVENGINE: USING XNA GAME STUDIO FOR SERIOUS GAME DESIGN AND DEVELOPMENT

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Abhishek Sood
Spring 2012
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Abhishek Sood:

NeevEngine: Using XNA Game Studio for Serious Game Design and Development

Dr. Kris Stewart, Chair
Department of Computer Science

Dr. Khaled Morsi
Department of Mechanical Engineering

Prof. Alan Riggins
Department of Computer Science

Approval Date
Copyright © 2012

by

Abhishek Sood

All Rights Reserved
DEDICATION

Dedicated to my parents, Mr. Rajiv Sood and Mrs. Shyama Sood, who always encouraged and supported me in all my efforts.
Experience is the best teacher.
ABSTRACT OF THE THESIS

NeevEngine: Using XNA Game Studio for Serious Game Design and Development

by
Abhishek Sood
Master of Science in Computer Science
San Diego State University, 2012

This paper describes a modular framework for development and rapid prototyping of serious games using XNA framework for Xbox 360 game programming. NeevEngine, a game engine was designed and developed on top of XNA Game Studio using Visual Studio 2008 Express. NeevEngine consists of an internal Graphics Engine, UI framework, Input framework and Quiz Api which form the core components of the engine. The modular design of the engine provides extensible possibilities for further enhancements. NeevEngine was used for development of Mat Isles, a mechanical laboratory simulation where the player goes through the steps involved in Tensile Testing Experiment. The game engine provides support for 3D room rendering, animations, presentation of educational content and presentation of quizzes. All the quiz data is stored in a MySql database with the help of PHP webservice which can be later retrieved. This allows the game writer and content developers to modify and include new content material to further enhance the educational value of the game. Results show that XNA can be a viable platform for creation and development of educational games, minimizing core development work needed to manage art/sound assets and development of basic techniques for 3D game programming. With the help of NeevEngine this process is further streamlined and enhanced thereby enabling the development team to focus on content generation and presentation.
# TABLE OF CONTENTS

ABSTRACT .......................................................................................................................... v
LIST OF TABLES .................................................................................................................. vii
LIST OF FIGURES ................................................................................................................ vii
ACKNOWLEDGEMENTS ...................................................................................................... ix

1 INTRODUCTION ............................................................................................................... 6
   1.1 What’s so serious about these games? ........................................................................... 6
   1.2 XNA Framework ......................................................................................................... 7
   1.3 Mat Isles .................................................................................................................... 7

2 BACKGROUND RESEARCH ......................................................................................... 9
   2.1 Classifications of Games ........................................................................................... 9
      2.1.1 Platforms ........................................................................................................... 9
      2.1.2 Genres ............................................................................................................. 11
      2.1.3 Demographics .................................................................................................. 11
   2.2 Supplementing Education through Games ................................................................. 13
   2.3 Existing Serious Games ............................................................................................ 14
   2.4 XNA and Game Development .................................................................................. 15

3 GAME ENGINE DESIGN ............................................................................................... 19
   3.1 Need for a Game Engine ............................................................................................ 19
   3.2 Requirements ............................................................................................................ 20
   3.3 NeevEngine Architecture Design .............................................................................. 20
   3.4 Initialization and Window Management .................................................................... 23
   3.5 Game Screen Management ....................................................................................... 24
   3.6 Developing a Graphics Engine .................................................................................. 26
   3.6 Capturing the User Input ........................................................................................... 28
   3.7 User Interface ........................................................................................................... 29
   3.8 Model Sanity – Testing 3D Graphic Content .............................................................. 31

4 USE CASE: MAT ISLES ................................................................................................. 34
   4.1 Backend support ....................................................................................................... 34
   4.2 Game Text generation ............................................................................................... 36
LIST OF FIGURES

Figure 1. IntelliGym - Basketball Edition [24].................................................................15
Figure 2. Technology Architecture for a XNA Game .........................................................16
Figure 3. Software Abstraction Levels in a Game ...............................................................19
Figure 4. NeevEngine Architecture Block Diagram..........................................................21
Figure 5. Class diagram for Game Window Management ....................................................23
Figure 6. Class diagram for Game Screen Management .....................................................25
Figure 7. Class diagram for 2D graphics routines ...............................................................27
Figure 8. Class diagram for 3D graphics routines ...............................................................28
Figure 9. Class diagram for Input Management .................................................................29
Figure 10. Various UIWidget controls available in NeevEngine .......................................31
Figure 11. Model Sanity - Open .fbx file to view with NeevEngine Graphic Routines .........32
Figure 12. Model Sanity - Displaying model after it has been compiled into XNA understandable binary file.................................................................33
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to all the people who have guided and encouraged me during my thesis research.

First and foremost, I would like to thank my project advisor and thesis chair, Dr. Kris Stewart for giving me the opportunity to work on this thesis project. I am grateful for her continuous guidance, support, and motivation throughout the project.

Special thanks go to Dr. Khaled Morsi, for accepting me as his research assistant on Mat Isles project. My thesis has immensely benefited from his project as a use case.

I am grateful to Professor Alan Riggins for being on my committee and for my first programming course at SDSU under him. He also graded me on my first project ever that was game for which I am thankful.

I would also like to thank Professor Mark Siprut for his excellent guidance on the creative side of a very interesting field.

I would like to thank the Mat Isles team together with whom NeevEngine was deployed in real world use case scenario. Thanks go to Claudia Faulk, Mark Thompson and Christina Bertang for their hard work and support during development of Mat Isles.

Thanks go to my parents for providing me a way to fall in love with computers, programming and technology in general.
CHAPTER 1

INTRODUCTION

Simulations and contextually relevant games are reproductions of essential features of a real life situation, presenting the user with visual feedback to enforce learning and deeper understanding of specific subject matter. Many education fields are now experimenting to introduce, in some form or other, engaging real world educational or serious games (henceforth referred as games) to increase collaboration and student learning through experimentation. As with most other software engineering fields, game development benefits immensely from well thought-out reusable, modular code. In this thesis document, I take the opportunity to present the design and development of NeevEngine on top of XNA Game Studio 3.1. NeevEngine is used for development of a mechanical laboratory simulation, Mat Isles. This document explores the impact of NeevEngine on development of serious games.

1.1 What’s so serious about these games?

Even before the advent of computer simulations, Clark Abt discussed the idea and used the term in his 1970 book *Serious Game* published by Viking Press [1]. According to the book *Serious Game* is defined as:

Reduced to its formal essence, a game is an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context. A more conventional definition would say that a game is a context with rules among adversaries trying to win objectives. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement.

While this is mostly true for time period before the advent of person computer and interactive games, Journal of Simulation and Gaming (S&G) defines *Serious Game* as [2]:

…game that makes use of computer technology and advanced video graphics and that is used for the purposes of learning and training, as researched and talked about in countless conferences, publications, institutes, and websites.

With the advent of computer graphics and visualizations the use of games is no longer limited just to entertainment. Researchers are using engagement and collaborative values of
games to enhance education and learning systems with relatively high degrees of success [3, 4]. With the support of peer-like agent, the game’s competitive environment can help users to concentrate on brainstorming tasks and aid them in collaborating and developing problem solving abilities [5].

1.2 XNA Framework

XNA originally came out of Xbox New Architecture development. Instead of being released under the Xbox name, the Xbox 360 was released, and XNA became defined as "XNAs Not Acronymed" [6]. XNA Game Studio 3.1 is a programming environment developed by Microsoft used for game development on the Xbox 360 console, and Windows-based computers. XNA in collection of managed libraries based on Microsoft .Net framework 2.0 [7].

XNA was first announced in GDC (annually biggest Game Developers Conference) in 2004 by Microsoft and was first released at Gamefest Conference (a new game developer conference by Microsoft) in 2006 as XNA Game Studio Express beta 1. A multitude of game hobbyists and game programming beginners downloaded the beta and wrote many small 2D games with the help of “Space Wars” starter kit included with the framework. Programming a 3D game was still hard and required a lot of effort to work with the beta framework. Microsoft gradually released another beta in November of 2006 before the final release of XNA Game Studio Express in December 2006, which included the content pipeline and many advanced features that still form most of the core concepts of XNA till date [6, 7].

XNA is completely free and allows development on Windows and Xbox 360 platforms. But to run your game on Xbox 360 you have to join the “Creators Club” for an annual fee of $99. This makes XNA quiet lucrative as development tool for game development by small teams. As of writing of this thesis “Creators Club” has been renamed to “App Hub” and XNA Game Studio 4.0 was released with support for Windows Phone 7. The App Hub subscription is available for $99 per year and includes support for publishing your game for Xbox 360 and Windows Phone 7[8].

1.3 Mat Isles

Mat Isles is an educational game based on three core experiments performed in Mechanical Engineering Lab. These experiments include Tensile Testing, Impact Testing and
exploration of 3-D lattice structure of a metal. Mat Isles is being developed under the guidance of a panel of three San Diego State University professors, Dr. Khaled Morsi, Dr. Kris Stewart and Professor Mark Siprut. The project is funded through NSF grant DUE #0837162. [20]

The project uses Neev Engine, which is built on top of XNA, to simulate the laboratory experience for Tensile and Impact testing of different structural metals such as copper, aluminum and iron. This application provides a safer, controlled environment for the Material Science students to explore the experiments in depth on their computers. The game is not a replacement for real world lab experiments but is used as a supplement to familiarize the students with safety and experiment procedures in an real world lab.

The game includes a 3D world consisting of lab room, mechanical equipment and safety equipment replicating the ones found in real world lab. The students start a simulation as they take their first steps into the lab and simulation gently guides them through the actual safety and experiment procedure. The simulation also includes various quizzes to evaluate the student responses as they go through the experiments. This supplements and enforces lab education in a similar way as in real world.
CHAPTER 2

BACKGROUND RESEARCH

2.1 Classifications of Games

Video Games can be classified based on various metrics. Some of the more commonly used classifications are based on the platforms the game is available on; theme and game play focus of the game; demographics the game audience represents.

2.1.1 PLATFORMS

The term "platform" refers to hardware or software architecture or a combination thereof that serves as a foundation or base for game development. The term originally dealt with only hardware, and it may still refer to only a CPU model or computer family. The term “system” is sometimes used interchangeably with “platform”. Some of the well-known gaming platforms/systems are:

2.1.1.1 Personal Computer

“PC Games” refer to the games playable on personal computers connected to a high resolution monitor. One of the earliest games developed for computers was *SapceWars*, developed by Steve Russell for PDP-1 [9]. A number of text based games were made by hobbyists in later years. With the advent of graphics and visualization technologies, pc games have evolved tremendously both in content size and game play sophistication. The player typically uses a keyboard and/or a mouse to control the game. Many of the highly successful games are available as PC only titles including but not limited to World of Warcraft, Half Life, Diablo etc.

2.1.1.2 Consoles

A video game console is a modified computer that is specifically designed to play video games. The console is generally connected to a home television and the user controls the game through gamepads or analog stick controllers. In 1972 Magnavox released Magnavox Odyssey, the first home video game console which could be connected to a TV
set to play games. Today the video game console market is dominated by three main systems: Microsoft’s Xbox 360, Sony’s Playstation 3 and Nintendo’s Wii

2.1.1.3 Arcade machines

Arcade game is a coin-operated entertainment machine which is installed in public businesses such as restaurants and video arcades. The arcade machines operated on 25cent coins which gave the player a fixed amount of lives once the coin was inserted. Atari’s Pong, released in 1972 was the first successful arcade machine [9]. Earlier arcade machines consisted of electronic circuitry and a video display that is housed in a wooden box. The controllers were mounted on the wooden box which the player used to play the game. The machine would only allow the player to play once a 25cent coin was inserted. With advent of video game consoles and computer games the arcade video games suffered great financial loses and decline due to the high cost incurred on repetitive replays of game.

2.1.1.4 Handheld Gaming Devices

Handheld gaming devices consist of a small display, controllers and speakers all coupled together in a lightweight, portable electronic device. Unlike video game consoles, the display, speakers and controllers are part of the same unit. They generally come with only a specific pre-installed game and allow the user to play the game anywhere and at any time. Some of the newer handheld devices like Nintendo DS and Sony Playstation Portable have introduced color LED displays and allow new game download over wireless networks.

2.1.1.5 Mobile Phones, PDAs

Mobile phones and PDAs are not generally developed as gaming devices but they do offer capabilities to play games with the added benefit of being portable. In 1997, Nokia released first mobile game, Snake, for Nokia 6110 device [10]. Snake was hugely popular and has since been regarded as classic. With the rise of mobile OS platforms like Apple iOS, Google Android OS, and Microsoft Windows Mobile 7, the mobile OS developers themselves have launched digital download storefronts that can be run on the devices using the OS or from software used on PCs. These storefronts (Apple's iOS App Store, Android’s Market Place and Microsoft’s App Hub) act as centralized digital download services from which a variety of entertainment media and software can be downloaded, including games.
2.1.2 GENRES

Video games are also classified into various genres based on the gameplay, art style and content. Since the genres are based on the content, with the introductions of new games genres have evolved and modified as well. Some games even fall into multiple genres and have given rise to entirely new genres.

The games that are generally play from first person perspective where the player observes and interacts with the world as though he is an actual person standing in the world are called “FPS” games. FPS stands for First Person Shooters and generally includes games such as Halo, Half Life etc. Some games focus on driving representing the player as a vehicle such as car, bike or truck. A few games simulate a real world experience such as flying a plain or controlling and building a village or resort. They fall under simulation and strategy genres respectively. Some genres represent combinations of others, such as massively multiplayer online role-playing games, or, more commonly, MMORPGs. It is also common to see higher level genre terms that are collective in nature across all other genres such as with action, music/rhythm or horror-themed video games.

2.1.3 DEMOGRAPHICS

Core games

In general, discussion about video gaming in both the press and politics revolves around titles found in the core games classification; historically, consisting of video games developed for play on personal computers, dedicated video game consoles or handheld game consoles.

Core games are generally defined by their intensity, depth of play or scale of production involved in their creation and can include games across a wide spectrum of genres. For example the Bit.Trip series for WiiWare, the Fallout series for PC and console or LittleBigPlanet for the PS3, all fall within the core games classification. Core games are sometimes considered demanding in their gameplay and typically require more time to explore or master the game.
Casual games

Casual games are highly accessible games developed with easy of accessibility as their main strength. They are simple to understand and have a smaller, less complicated rule set. Casual games frequently support the ability to jump in and out of play on demand, giving the player the option to save their progress and resume it later at any point in time. Some of earlier examples of casual games, that existed even before the term was coined, include Solitaire and Minesweeper, which used to ship with many versions of windows.

Casual games often make use of online portals and website such as Popcap, Facebook etc. to reach a wider audience. Initially available mostly for personal computers, these games have quickly spread to cellphones, PDAs, and even on-line consoles download systems such as XBOX live, Playstation Network and WiiWare. In recent years, one of the most famous companies that has become synonymous with the term Casual Gaming and has grown to unforeseen proportions is Zynga, which launched Zynga Poker on Facebook and MySpace websites in 2007 [15]. Since then, Zynga has grown into a social/casual gaming powerhouse with games such as City Ville, Farmville and Mafia Wars. As of November 2011, Zynga has more than 200 million monthly users on Facebook with City Ville having over 54 million monthly users alone.

Serious games

Serious games focus on conveying information or a learning experience to the player. Educational software, such as language learning or touch typing, does not typically fall under this category. The primary distinction depends on the target audience and primary knowledge represented in the game. Serious games are games generally made for reasons beyond simple entertainment and as with the core and casual games may include works from any given genre, although some such as exercise games, educational games, or propaganda games may have a higher representation in this group due to their subject matter. These games are typically designed to be played by professionals as part of a specific job or for skill set improvement. They can also be created to convey social-political awareness on a specific subject.

Serious games have been used successfully in many researches to supplement the education in course work. There are several well-documented cases where serious games
have helped with computer education [17], divergent thinking [18] and study of human behavior [19]. The use of serious game to a serious game for treatment of cockroach phobia in adult human is particularly interesting [19]. The power of serious games as an education supplement tool has been proven time and again.

**Educational games**

Educational games focus on teaching specific study area to students. These include spelling and counting games for kids or subject matter expertise on medical or technical materials for adults. Some educational games don’t have a specific audience instead they provide relevant information on real world topics such as driving laws. On September 23, 2009, U.S. President Barack Obama launched a campaign called "Educate to Innovate" aimed at improving the technological, mathematical, scientific and engineering abilities of American students. The campaign states that it wants to harvest the power of video games for education [11]. This campaign has created new opportunities in video game realm and has contributed to competitions such as STEM National Video Game Competition and the Imagine Cup [12][13]. Both of these examples are events that bring to focus the power of video games as a relevant media for important current issues in this case the latter being education and spread of knowledge. www.NobelPrize.org uses games to entice the user to learn about information pertaining to the Nobel Prize achievements while engaging in a fun to play video game [14].

### 2.2 Supplementing Education through Games

Numerous systematizations of learning objectives can be found in literature. Depending on the theoretical approach, these are more strongly oriented toward the external environment for which learning should prepare or toward learning psychological concepts that come from different forms of knowledge and skills.

Almost by definition, any initiative that mixes videogames and education can be considered as game-based learning. Initiatives range from the introduction of AAA commercial games in educational processes to the application of slightly interactive multimedia wrappers around traditional educational content. Thus, educational game design is a broad subject that groups very different approaches and methodologies.
Within that broadness, authors like Prensky (2001) state that an effective educational game design must achieve a balance between fun and educational value [16].

### 2.3 Existing Serious Games

Microsoft Flight simulator is one of the longest running serious games, first published with the same name in 1982 under the same name. The game has been used in US military to train air pilots. Other virtual reality games for training ground forces such as police and fire fighter have been in constant use by the United States military. Games such as Delta Force 2: Land warrior and Steal Beasts are now used to train real soldiers. The games focus not only the complicated equipment that soldiers use but also on team work and co-ordination [21]. Second Life is another example of using virtual world to further educational and business interest [22]. Many of the top universities, including SDSU, have an online presence in Second Life to help reach and connect more students.

A few other examples of serious games that stand out are:

1. **A Force More Powerful**: Inspired by the York-Zimmerman documentary of the same name, this video game is designed to teach waging of a conflict using non-violent methods and is intended for use by activists and leaders of nonviolent resistance and opposition movements. The graphic style of the game closely follows that of other top down city building games such as Sim City. The game, developed by Breakaway Games Ltd., was showcased at Serious Game Summit in 2005 where it received critical acclaim. [23]

2. **DARWARS Ambush! Convoy Simulator**: A PC based networked multiplayer simulation, DARWARS Ambush! Was designed to create low-cost experimental training systems. Developed as part of DARPA's DARWARS project, this video game is generally restricted to US Military and Defense personal. Game’s most important innovation was the user-author ability, giving soldiers to create their own scenarios in matter of hours or days without a contractor between them and the system.

3. **IntelliGym**: Available by the name of Basketball IntelliGym, the video game consists of series of computer based cognitive simulators designed to increase cognitive performance of athletes. The game focuses on rapid decision-making, pattern
recognition, anticipation and perception. Coaches have reported significant improvement in player performance of trainees, as shown by their statistical measures. [24]

![IntelliGym - Basketball Edition](image)

**Figure 1. IntelliGym - Basketball Edition [24]**

There are numerous other examples of serious games, such as Microsoft Flight Simulator X (airplane flying simulation), Steel beasts (tank simulation), Floodsim (flood prevention simulation) etc., available as part of research projects or training simulations, being used in trainings and education to varying degrees of success.

### 2.4 XNA and Game Development

Microsoft provides XNA framework specifically for video game development on PC, Windows Phone 7 and Xbox 360. XNA framework is not a Game Engine; it does not contain code for collision detection, physics simulation or other modular systems found in a typical game engine. XNA framework comes as a part of XNA Game Studio, which is a plugin that gets installed into Visual Studio IDE. Typically XNA framework games are written in C#, but there is support for other languages with loss of some functionality.

XNA framework is based on native implementation of .Net Compact Framework 2.0 for Xbox 360 and .Net Framework 2.0 for Windows. It provides extensive set of class libraries, to enable maximum reuse of boilerplate game code across target platforms. XNA games are run on Common Language Runtime that is optimized for gaming to provide
maximum performance in managed environment. Since the CLR is available on multiple platforms (Windows, Xbox, Windows Phone 7), the game code can be run on any of these with minor modifications.

The framework provides low-level encapsulated code that takes care of details involved in porting the game code across different platforms. This helps the developers to concentrate on the game design, gameplay coding and art content production. XNA framework also integrates with Cross-platform Audio Creation Tool (XACT), which takes care of audio content creation. XNA Framework Content Pipeline is a part of XNA framework that deals with import and management of 2D and 3D art assets used in the game. This simplifies the process of importing 2D or 3D art from content creation programs (such as Photoshop, Maya etc.) and managing them together in a XNA project. XDK Extensions, formerly known as XNA Game Studio Professional, is an add-on to XNA Game Studio and requires the Microsoft Xbox Development kit. Both are available to only licensed Xbox developers. XDK Extensions include additional managed APIs that provide access to features reserved for licensed developers, such as Achievements and Leaderboards.

![Figure 2. Technology Architecture for a XNA Game](image)
A few core classes that are available in XNA and utilized most heavily in any XNA project are [25]:

1. **Game Class**: Game class provides basic graphics device initialization, game logic, rendering code, and a game loop. This can be considered as the entry point into the game code. The LoadContent method is Game class is responsible for loading the art assets and fonts used by the game. The game logic is contained in the Update method and the rendering logic is contained in the Draw method. The game loop consists of repeatedly calling the Update and Draw methods after a fixed amount of time is passed (in case of fixed frame rate) or one after the other as soon as possible (in case the frame rate is not limited). The Run method in Game is responsible for initializing the game, begin running the game loop, and start processing events for the game.

2. **GameComponent/DrawableGameComponent Class**: Game components provide a modular way of adding functionality to a game. A game component is created by deriving the new component either from the GameComponent class, or, if the component loads and draws graphics content, from the DrawableGameComponent class. Game logic and rendering code is added to the game component by overriding Update, Draw and Initialize methods. A game component is registered with a game by adding it to Game.Components member that is a collection of GameComponent objects. A registered component will have its draw, update, and initialize methods called from the Initialize, Update, and Draw methods in Game class.

3. **GameTime Class**: Each time the Update or Draw methods are called they are given a GameTime object as calling parameter. The GameTime object contains a snapshot of the elapsed time since game has started both as in game time and real time. This information can be used to time animations, actions and simulate timed behavior on screen.

4. **GraphicDeviceManager Class**: Graphics device manager handles the configuration and management of the graphic device. Graphic device is the low-level configuration that holds the information relevant to the rendering of data onto GPU memory, maintains the state and dictates how the data is presented on the screen.

5. **GameServiceContainer Class**: Game services reduce the coupling between essential game objects and objects which need to interact with them. Services work through a
mediator—in this case, Services member variable in Game class. Service providers register with Game.Services, and service consumers request services from Game.Services. This arrangement allows an object that requires a service to request the service without knowing the name of the service provider.
CHAPTER 3

GAME ENGINE DESIGN

3.1 Need for a Game Engine

Games consist of a lot of boilerplate code that interacts with low level graphics, audio and input hardware. Frameworks, like XNA, make this interaction less painful by abstracting the low-level hardware code and supporting different platforms. There is still a need to define a generic architect for modular design of game code and keeping the game play code separate from graphics, input and game state management code. This is where a game engine comes into play.

![Software Abstraction Levels in a Game](image)

Figure 3. Software Abstraction Levels in a Game

A game engine contains essential parts of the game, those required for development of any game in a specific genre. Game engine consist of code that is responsible for the import of assets form artist, parsing the data files, manipulation of the art assets to construct a comprehensible world. It also provides a generic framework to handle user input, game window management, game state management, a generic game loop to update and draw the game, graphics routines to optimize the drawing of 2D and 3D graphics, managing audio
content and basic user interface controls to streamline the game development. Separating the
game logic and game art from game engine code makes it immensely helpful for game
designers/artist/programmers to focus on gameplay rather than being worried about low-level
engine code.

This division of game code into game engine and game play modules also enables the
same game engine code to work with multiple game projects. This increases code reusability,
code testability and leads to cleaner coding practices.

3.2 Requirements

Since NeevEngine was specifically developed to support Mat Isles development,
some of the requirements that were outlined during the design phase were:

1. Managing the 3D and 2D art assets used in the game. Optimizing the drawing
   routines to keep the game performance above 60 frames per second on base system.
2. Animating 3D models to replicate experiment procedure followed in real world
   laboratories.
3. Managing the game state and flow of information on the screen. Provide the player
   with the ability to review previously covered education material.
4. Provide support for player input through keyboard and mouse.
5. Provide UI controls that make up the game and are used to present educational
   material.
6. 3D camera management for drawing of 3D graphics and give the player a feeling of
   immersion in a 3D lab from first person point of view.

3.3 NeevEngine Architecture Design

NeevEngine consists of 4 main components: main game class, game components to
manage functionalities of different modules, game state management and graphic routines to
draw sprites and 3D graphics. These components manage different aspects of game and work
together to support game logic written by developers and manage art content created by
artists.

Main game class, NeevGame, extends Game class from XNA framework. It contains
the game initializing code and defines a basic game loop. It adds all the required components
such as InputManager, GameScreenManager and CameraManager for maintaining and
updating the state of these modules. WinFormGame wraps the NeevGame object and adds the capability to draw the game on a Windows framework panel. This panel is then attached to a Windows Form thereby enabling use of XNA with basic Windows GUI components. This lets the developer create game tools, such as Level Editor, that rely on same graphic routines as used in the NeevGame.

Figure 4. NeevEngine Architecture Block Diagram

NeevGame initializes and adds different game components that are responsible for defining various behaviors attached the game. Each game component is responsible for a specific functionality provided by the engine. Most of them are also added as services in NeevGame.Services object for access through any part of the game. InputManager component is responsible for querying and maintaining the user input received through keyboard or mouse. GameScreenManager is responsible for maintaining a stack of GameScreen objects that contain modular code for different screens in the game.

CameraManager is responsible for maintaining the camera state in 3D game world. There are 3 different types of cameras available in NeevEngine: Orbit camera (which rotates, zooms and pans around a fixed origin), LookAt camera (which defines an arbitrary eye point
looking towards an arbitrary target point) and Quaternion camera (which can be rotated and translated in the game world). For the 3D screens focusing on the experiments Orbit camera mode is used and for the Lab screen, in which the user has to navigate through the lab, Quaternion camera is used to give the illusion of an FPS game.

GameConsole manages a debug console that can be accessed by pressing tilde (~) key and is available in Debug build by default. NeevGame defines various helper log methods which can be used to log variables and helpful debug info while the game is running. These log messages are then printed to the GameConsole, giving developer another powerful tool to debug any game logic bugs they may encounter during the development.

GameCursor manages the mouse presentation inside a NeevGame. It provides the ability to update the graphics used in place of standard windows mouse as well as restricting the mouse movement to center of the screen when Quaternion camera is in use thereby preventing it to run out of the XNA game screen.

FPSCounter measure the game performance every tick and displays this information on the upper left corner when the game is running in Debug mode. FPS in this scenario refers to Frame per Second, a unit of measuring how many frames are drawn on screen per second. For the animations to look smooth, the game should be running at 30 frames per second though it is recommended to aim for 60 frames per second which is the common refresh frequency of most computer monitors.

The game screen management is achieved through the use of GameScreen class. Subclasses of GameScreen class together with GameScreenManager enables the developers to write modular code for different game screen in the game, making it cleaner and easier to maintain. The UIWidget and its subclasses provide developers some standard Windows GUI controls which can be used to create the user interface of the game. Together, these to concepts make the task of creating a UI much simpler and rapid.

At the heart of NeevEngine are the graphic routines that manage the 2D, 3D and font art created by artists. These routines are also responsible for drawing the UI, 3D graphics and text in the NeevGame. Since the Draw method is being called almost 60 times per second, any delay in drawing graphics on screen cause the game to lag and breaks the user immersion in the game. Therefore, it is crucial that these graphic drawing routines be blazingly fast. NeevEngine also wraps Texture (NeevTexture), Model (NeevModel) and SpriteFont
(NeevFont) objects which extend their functionalities and enables them to be used with NeevEngine graphic routines.

### 3.4 Initialization and Window Management

XNA framework provides the Game class that takes care of the low level initialization and preparation of the system and also provides helpful events, such as activation and deactivation of game itself. It is recommended that, the programmer extend Game class and over write appropriate methods to add game play functionality.

![Figure 5. Class diagram for Game Window Management](image-url)
The NeevGame class, in NeevEngine namespace, extends Game class and contains the code for initializing the game engine by loading the resources, creating the graphic context, adding relevant services, updating and rendering the game. The NeevEngine class fires up sub-systems to manage game screens (GameScreenManager), catch the input (InputManager), manage the camera properties (CameraManager) and manage the rendering of 2D and 3D graphics (UIRenderer and Renderer3D).

Keeping in mind the philosophy of inheritance adopted by XNA framework, NeevEngine class was designed to be inherited and extended by the developer. Some of the important methods to override in child class are BeginRun (to designate the starting GameScreen), Update (to update any gameplay that is not supported by NeevEngine services) and Draw (to draw any component not handled by NeevEngine).

In addition to NeevGame class, which provides the initialization code for an XNA game, NeevEngine namespace also contains WinFormGame class. WinFormGame inherits from Windows.Frame class and was designed to run on Windows platform only. WinFormGame is a convenience wrapper around NeevEngine and adds Windows initialization, management and rendering code around the XNA Game class. The main benefit of WinFormGame is that it provides base code to create game authoring and content management tools such as Model Sanity. The tools show the user exactly how the content would look inside the game, as the graphics engine powering both the game and tools is same.

### 3.5 Game Screen Management

To keep the game play and game menu code modular, NeevEngine uses the concept of Game Screen management. The functionality of each menu screen and game play screen is wrapped inside a subclass of GameScreen class. The GameScreen class provides base code for input handling, UI components management and UI rendering. There are two subclasses of GameScreen class, which are available for special purposes. These are DialogScreen and AnimationScreen. DialogScreen provides boilerplate code to show a dialog box with text on screen. It also has support for getting user input through choice of buttons. AnimationScreen provides boilerplate code to add 3D-animations in a 3D screen. It contains a list of Animation
objects which are run as specified in the subclass. This screen was extensively used for showing 3D animations in Mat Isles.

Figure 6. Class diagram for Game Screen Management

The GameScreenManager class is responsible for managing different GameScreen objects in the system. It maintains a stack of GameScreen objects and provides methods to maintain for addition, popping and replacing the top GameScreen in this stack. Only the top
level GameScreen is updated and rendered during each update-draw loop and this is handled by the GameScreenManager class.

This division of game into smaller screens helps in keeping the code modular and is beneficial in describing a flow from one game screen to other. The functionality of one game screen is de-coupled from the functionality of subsequent game screens.

### 3.6 Developing a Graphics Engine

XNA provides support to render fonts, 2D and 3D graphics through SpriteFont, Texture2D, Texture3D and Model classes. The 2D art such as font and 2D graphics are generally referred to as Sprites. The SpriteBatch class in Graphics package in XNA is responsible for drawing a group of sprites to be drawn with same rendering settings. A Model object is a collection of ModelMesh, which holds the 3D geometry data in the model. Each ModelMesh further consists of a collection of Effect objects that contain the effects and techniques used for drawing a particular Mesh.

NeevEngine wraps all fonts, 2D textures and 3D textures in NeevFont, NeevTexture and NeevModel classes respectively. The engine handles the initialization and rendering of all the art assets. The UIRenderer is responsible for drawing majority of 2D art and font that make the UI of the game. All the font and 2D sprites to be rendered at the end of the Update method are stored in a collection by UIRenderer and are drawn on screen in Draw method. In addition to sprites, UIRenderer includes an instance of LineManager, which is responsible for drawing 3D and 2D lines. LineManager is mostly used in Debug mode to print out debug information and to render the game console.

During each call to the Update method in GameScreen class, the GameScreen object registers all the sprites, fonts and lines to render to UIRenderer class. This collection of sprites is updated each tick and then drawn in during the Draw method of the NeevEngine class. The 2D graphics are drawn in order of which they are inserted. This preserves the Z factor for each sprite and gives an illusion of depth. The 2D graphics can consist of text or images making up most of the game UI.
Similarly, Renderer3D class maintains a collection of 3D graphics. There are two rendering queues in Renderer3D. The 3D graphics are in the default queue are drawn before any of the UI is rendered. The 3D graphics in post-rendering queue are drawn after the UI has been laid out. This functionality helps in creating UIWidget, which can draw 3D graphics inside them aiding the developers and designer in creating more complex user interface.

Figure 7. Class diagram for 2D graphics routines
3.6 Capturing the User Input

NeevEngine supports input through keyboard and mouse. The recognition of user inputs is event based. XNA provides two core structures, which contain the data about keyboard and mouse at any given point in time. These are KeyboardState and MouseState. KeyboardState structure contains the list of all the Keys that are pressed during the call to GetState method of Keyboard class in XNA framework. MouseState structure contains the x and y co-ordinates of Mouse pointer in addition to the state (Pressed/Released) of each MouseButton.

The InputManager class, which is a subclass of GameComponent, is responsible for querying the keyboard and mouse state every frame. It stores the current and previous states of keyboard and mouse buttons and contains helper functions to get key press and mouse movement events. An object of InputManager is added to NeevEngine.Services object and
used throughout the engine to query the state of a particular key or mouse button as well as the mouse pointer co-ordinates.

Each GameScreen object has an object of ActionMap class. The ActionMap class stores a binding of a particular key with a particular delegate object known as InputEventHandler, for a particular event type known as InputEventType. This provides an event based handling to the user input. During each call to the Update method in GameScreen class, the ActionMap object checks for different events (Pressed/Released/Down/Repeat) for the keys in the collection and if it detects the appropriate event the corresponding InputEventHandler runs. ActionMap class contains Bind method, which can be used to bind events and handlers together.

### 3.7 User Interface

As stated previously, most of the UI rendering is taken care of UIRenderer class. NeevEngine provide a base class, called UIWidget, which stores relevant information about the size and location as well as basic mouse behavior code for sub-classing. These UIWidget
provide out of the box functionality for most UI elements. Some of the most used UIWidget subclasses defined in NeevEngine are:

1. **Label**: A label allows a single line of text to be displayed. The label resizes automatically to make the text visible. A background texture can be applied to the label. If the text is omitted the label resizes to show the background image.

2. **TextureButton**: TextureButton comprises of two 2D textures, one for normal state and one for highlighted state. The TextureButton recognizes the mouse events such as MouseEnter, MouseExit and MouseClicked. Event handler can be added to these events to perform tasks relevant to the game.

3. **TextureTextButton**: Subclass of TextureButton, TextureTextButton provides functionality to store a NeevFont and single line of text.

4. **TextureTextBox**: TextureTextBox provides a text input field to get alpha-numeric input from the player. The 2D texture is used as background. The player can type in information such as their name.

5. **RadioButton**: Provides a standard radio button with the ability to set 2D graphics for selected/not-selected state. More than one RadioButton can be combined using a RadioButtonGroup providing in interface for exclusive choice.

6. **CheckBox**: Similar to RadioButton, a CheckBox provides inclusive choice selection.

7. **ScrollPane**: Provides customizable scroll bars, which appear when the content inside the scroll pane exceeds the available size of the scroll pane.

8. **TextPane**: TextPane provides the ability to display multiline text. The TextPane can be added to ScrollPane to provide scrollable behavior if multiline text does not fit the intended size of TextPane.
These UIWidget can be added to any GameScreen object, which maintains a collection of UIWidget objects. The GameScreen is responsible for updating and drawing the UIWidget collection in screen each tick. This provides developers and designers out of box functionality for many common types of UI controls without having to maintain their state or data. They are also free from writing the drawing code for any of these as they can simply enable/disable or hide/unhide these controls. To create more complex UI controls, UIWidget class can be sub-classed and used by the developers, already containing the basic functionality for updating and drawing of control.

### 3.8 Model Sanity – Testing 3D Graphic Content

During the development and creation of 3D art assets; it is crucial for the designers and artist to look at the models when they are imported through the XNA graphics pipeline. This also establishes a content flow from 3D graphic software, such as Maya, to XNA. During the initial phase of development the process for exporting 3D graphics and included textures from Maya into XNA was unclear and therefore NeevEngine includes a tester tool called Model Sanity.
Model Sanity project contains only one major class called ViewModelScreen, which uses the UIWidget provided by NeevEngine to create a simple interface to test 3D models. The camera is set to orbit mode and provides a 360-degree view of the loaded model. When the Model Sanity program is run, the user can load a model by pressing Ctrl-O key. The program then asks the user for the model file to be loaded. After selecting the .fbx model file, it is built into XNA usable binary format using the ContentBuilder object. ContentBuilder class in NeevEngine.Graphics name space wraps the functionality provided by BuildEngine class in .Net Framework.

Model Sanity is build using the NeevEngine and uses the same graphic routines to display 3D graphics. During initial phase of Mat Isles development it proved to be instrumental in providing quick feedback to artist and establishing a content flow process from Maya to XNA.

![Figure 11. Model Sanity - Open .fbx file to view with NeevEngine Graphic Routines](image)
Figure 12. Model Sanity - Displaying model after it has been compiled into XNA understandable binary file
CHAPTER 4

USE CASE: MAT ISLES

4.1 Game Concept

The focus of Mat Isles is to design a game that simulates a Mechanical Engineering Laboratory delivering crucial lab experience on the basics of materials strength and elasticity as explain in a real world Material Science Laboratory. Due to high number of students enrolling in Material Science course, non-mechanical engineering students often miss out on this important practical experience. Mat Isles is an attempt to gamify the educational content in this lab and make it more accessible to engineering and non-engineering students alike.

Mat Isles is a NSF funded project (grant DUE #0837162) and was conceived by Dr. Khaled Morsi at San Diego State University, Mechanical Engineering department. Dr. Kris Stewart is the Co-PI in this project responsible for the game software design. Other members of team include Professor Mark Siprut of Arts Department, who is in charge of managing the student team responsible for the game art and Dr.? Marilee Bresciani, who is responsible for student learning assessment achieved through Mat Isles. Mat Isles was developed by a team of graduate and under-graduate students studying at San Diego State University.

The project plan for Mat Isles was broken in two phases. Phase 1 objectives were to develop a set of 3 experiment simulations designed to be single player experiences. The game would include Tensile Testing experiment, Impact Testing experiment and 3D simulation of crystal lattice to explore the cell structure of materials. Phase 2 of the game concentrates on delivering a multi-player experience which would help students work as a team in a simulated lab with enhanced graphics and animations.

This document is not a comprehensive report on the Mat Isles project. It merely points out the programming design and technologies used for realization of this project.

4.2 Backend support

Since Mat Isles has a requirement to collect student responses in form of quizzes, it was critical to have a database system where each game play scores, quiz answers and player information was stored. This information would later be retrieved by the student learning
assessment team and various statistical measures would be performed on it. For tracking purposes each student was assigned a Mat Isles ID and password which he or she would use to play the game. The game consists of quizzes which test the players on the knowledge obtained from the game and help them get more familiarized with the practical knowledge.

While the game was running all this data was being collected by the game. After completing the experiments this data is recorded in a database running on Rohan web server provided by SDSU IT department. XML RPC was chosen as the mode of communication between the Game and the PHP webserver connecting to MySQL database on Rohan.

Rohan is an Academic Computing WWW server maintained by San Diego State University for use in courses and academic projects. The server provides support for most open source web technologies. For Mat Isles game a separate project account was created on Rohan with MySQL, PHP and SVN support.

MySQL is a database management system that is open source, highly secure and easy to use. Since Rohan already provides a MySQL instance it was chosen to hold the Mat Isles related information such as student logins and scores.

PHP (“Hypertext preprocessor”) is a scripting language used extensively for web development to generate dynamic pages. PHP is a server side language that can communicate with MySQL server and add dynamic content to HTML pages. With help the help of “Incutio XML-RPC Library for PHP” a webservice interface was created in PHP on top of MySQL instance. This XML-RPC interface was used for communication between MySQL and Mat Isles.

Since NeevEngine and Mat Isles is written in C# it was easy to find an open source library for making XML RPC calls to PHP webserver. The technical team ended up using XML-RPC .Net, created and maintained by Charles Cook, to add this functionality to Mat Isles code. The file MegameXmlProxy defines a one to one mapping for remote procedures available from the PHP web service. The GameStats class is responsible for collecting the player data during game play. After the simulation is finished the game data is sent to MySql instance running on Rohan through rpc calls to PHP xml rpc server.
4.3 Game Text generation

Text is one of the major content of any game and Mat Isles is no exception. Being an educational game meant even more text content which was often changed and revised. To decouple the inclusion of text data in the game code and provide the content owner (in this case Dr. Morsi) a simpler tool to change and add text frequently, technical team came up with an elegant solution to store the text data in a Google Docs Spreadsheet. The idea is to store all the text data in a centrally accessible place which also provides an easy to use interface for text modification. Each text string was assigned a unique ID which helped in identifying the string in game code and made the game code maintenance easier.

The technical team also wrote a String Generator utility to access the specific Google Spreadsheet and download the text content as an XML file. String Generator is developed on the code base provided by Google API for Google Docs access written in C#. String generator also generated a StringConstants.cs file which includes the string ids for access in game code.

The main benefit achieved through this decoupling of string data from game code is that it made adding, maintaining and changing the text displayed by game much easier. The content owners are given the power to define new text and modify it as they see fit, while the game programmers didn’t have to worry about new text changes in the code. All that is needed is to fire up String Generator and get an updated copy of text content for game.
Figure 13. String Generator - Login to Google Account
Since Mat Isles needs to track individual performance of each player for assessment, each student can be assigned a unique Mat Isles username. The players after receiving the User Name from professor can register themselves through the beginning screens of Mat Isles. During registration the player is prompted to choose a password for future login. The player can login into the Mat Isles after that and begin the game. During game play all his responses are recorded in the GameStats.cs class. When the simulation ended the data collected in GameStats.cs was sent to the MySql database through PHP xml rpc client running on Rohan webserver.

Login screen is a typical example of GameScreen object being used in live project. The screen is made up of UIWidget objects provided by NeevEngine.

**Figure 14. String Generator - The text content for Mat Isles**

<table>
<thead>
<tr>
<th>id</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TT_WELCOME</td>
</tr>
<tr>
<td>2</td>
<td>TT_SPEECHREF</td>
</tr>
<tr>
<td>3</td>
<td>TT_INTERFACE</td>
</tr>
<tr>
<td>4</td>
<td>TT_TASKLIST</td>
</tr>
<tr>
<td>5</td>
<td>TT_TIMER</td>
</tr>
<tr>
<td>6</td>
<td>TT_QUIZ_SUMMARY</td>
</tr>
<tr>
<td>7</td>
<td>TT_LA8_OBJECTIVES</td>
</tr>
<tr>
<td>8</td>
<td>TT_LA8_OBJECTIVES_TEXT</td>
</tr>
<tr>
<td>9</td>
<td>TT_LA8_OBJECTIVES_INFO1</td>
</tr>
<tr>
<td>10</td>
<td>TT_LA8_OBJECTIVES_INFO2</td>
</tr>
<tr>
<td>11</td>
<td>TT_LA8_OBJECTIVES_INFO3</td>
</tr>
<tr>
<td>12</td>
<td>Lab Protocol: page 4</td>
</tr>
<tr>
<td>13</td>
<td>Quiz Question 2</td>
</tr>
<tr>
<td>14</td>
<td>Quiz Question 3</td>
</tr>
<tr>
<td>15</td>
<td>Quiz Question 4</td>
</tr>
<tr>
<td>16</td>
<td>Quiz Question 5</td>
</tr>
<tr>
<td>17</td>
<td>Quiz Question 6</td>
</tr>
<tr>
<td>18</td>
<td>Quiz Question 7</td>
</tr>
<tr>
<td>19</td>
<td>Quiz Question 8</td>
</tr>
<tr>
<td>20</td>
<td>Quiz Question 9</td>
</tr>
<tr>
<td>21</td>
<td>Quiz Question 10</td>
</tr>
<tr>
<td>22</td>
<td>ASTN Question 2</td>
</tr>
<tr>
<td>23</td>
<td>ASTN Question 3</td>
</tr>
<tr>
<td>24</td>
<td>ASTN Question 4</td>
</tr>
<tr>
<td>25</td>
<td>ASTN Question 5</td>
</tr>
<tr>
<td>26</td>
<td>ASTN Question 6</td>
</tr>
<tr>
<td>27</td>
<td>Lab Safety Information roompg17</td>
</tr>
<tr>
<td>28</td>
<td>SAFETY QUIZ QUESTION 1 A</td>
</tr>
<tr>
<td>29</td>
<td>SAFETY QUIZ QUESTION 2 A</td>
</tr>
<tr>
<td>30</td>
<td>SAFETY QUIZ QUESTION 3 A</td>
</tr>
</tbody>
</table>

**4.4 Login and Register Screens**
4.5 Quiz Screens

The management of quiz text data is handled in the similar way as game text. Since the quizzes contain question and answers, they are not stored on Google Docs; instead another tool is used to manage the quiz XML files directly. Quiz XML Generator is also written in C# and uses the Quiz classes defined in Mat Isles project. Quiz XML Generator provides simplified UI to create new quizzes or modify existing ones. Each quiz consists of a multiple questions and each question can have multiple answers. The correct answers are also marked and stored in XML file. This XML file is later loaded by Mat Isles code and is used to display it on the screen during game play.

Similar to String Generator, Quiz XML Generator tool simplified the process of quiz content creation and maintenance. The tool is simple enough and usable by quiz content owner (again Dr, Morsi) and the game programmer is only concerned with the display code for each quiz.
### Figure 16. Quiz XML Generator - Modifying a quiz xml file

<table>
<thead>
<tr>
<th>Question 1: TT_PRO_1</th>
<th>Which specimen dimensions do you initially need to measure?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Diameter</td>
<td>Delete</td>
</tr>
<tr>
<td>b. Width and thickness</td>
<td>Delete</td>
</tr>
<tr>
<td>c. Width and height</td>
<td>Delete</td>
</tr>
<tr>
<td>d. None of the above</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question 2: TT_PRO_2</th>
<th>Which steel specimen will you be discarding?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. The one that weighs the most</td>
<td>Delete</td>
</tr>
<tr>
<td>b. The one that weighs the least</td>
<td>Delete</td>
</tr>
<tr>
<td>c. The one that conform to ASTM standards</td>
<td>Delete</td>
</tr>
<tr>
<td>d. The one that do not conform to ASTM Standards</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question 3: TT_PRO_3</th>
<th>How far apart are the two indents that you will initially place in the center of the specimen?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 0.5 inches</td>
<td>Delete</td>
</tr>
<tr>
<td>b. 1 inch</td>
<td>Delete</td>
</tr>
<tr>
<td>c. 1.5 inches</td>
<td>Delete</td>
</tr>
<tr>
<td>d. None of the above</td>
<td>Delete</td>
</tr>
</tbody>
</table>
4.6 Animation Screens showing the Experiment Steps

To simulate various steps involved in Tensile Testing experiments, AnimationScreen class was extended which showed the animation of experiment being performed. Since the practical knowledge is best described through visual representation, major effort is done to model the 3D graphics and create animations that resemble closely to real life scenario. Figure 18 depicts the games showing the steps to hammer a notch in the test specimen. This is a crucial step to be performed before the test specimen is attached to the Universal Testing Machine. Figure 20 shows the extension meter being attached to the specimen just before the experiment begins.

NeevEngine also provided support to play a video file which is being used in Figure 19 to show the specimen being elongated as the tensile strength test is being performed on it. The player can pause, resume or rewind the video as he sees fit.

Inclusion of these animations and video in an education game helps the player in familiarizing and understanding the theoretical concepts better.
Figure 18. Animation screen showing the step for making a notch in test specimen

Figure 19. 2D video showing the tensile testing experiment being performed
Figure 20. Animation screen depicting the proper steps to attach and detach test specimen from Universal Testing Machine

4.7 Gauging the Student Performance

Since all the data from students game play is being stored in a MySQL database, it technical team created a website for score reporting and student id management. The site home page shows the Top 5 score of all time and most recent 5 scores added to the database. Notice that the scores are all set to Zero. This is due to the fact that a metric was still being developed at the time of writing this document.

The website includes an interface for the administrator to login and manage student accounts. A list of registered students is shown from where the admin can remove any student or reset there game password. Clicking on the student ID shows the score history for that specific student giving a basic idea of his or her performance over multiple game plays.
Figure 21. Mat Isles website - Homepage - showing the top 5 and recent 5 scores

Figure 22. Mat Isles website - Student account management
Figure 23. Mat Isles website - Scores for a particular student
CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The objective of this research was to build a flexible and extensible Game Engine for use in development of Serious Games. I used XNA Game Studio 3.1 as the base platform which provides extensive libraries for game development on top of Microsoft .Net framework 2.0. A modular architecture has been proposed and implemented that shortens the game development time and helps in rapid prototyping of games on Windows and Xbox 360.

The use case shows that NeevEngine helped in cost mitigation, change management and rapid development of Mat Isles. Model Sanity helped the artists generate and test art assets by streamlining the process of content inclusion from Maya 2011 to XNA 3.1. It was instrumental in testing and rapid development of art assets. The modular design of the engine made it easy to produce prototypes and incorporate changes quickly into the existing code base. The graphic engine provided enhanced 3D visuals and an accurate representation of mechanical laboratory machines in Mat Isles. It also added the support for animations to explain steps involved in the experiment.

The UI module helped in development of a consistent user interface, used for presenting the overview of steps involved in the Tensile Testing experiment developed for Mat Isles. Once the education material was displayed the Quiz component presented the player with a number of related questions about the content. All the responses were recorded in a MySql database on Rohan webserver which may help in future development of the game and justify the inclusion of additional educational material.

XNA proved to be a good choice as game development platform and with the help of NeevEngine the overall process was streamlined. The results were consistent with previous research findings. NeevEngine proved to be instrumental in development of Mat Isles.
5.2 Future Work

I have tried to accomplish a lot in this project and NeevEngine turned out to be quiet satisfactory for its first iteration. None the less, there are a few major areas that need more work and can be improved upon. Some of them are highlighted here:

1. Currently the engine does not support ambient and point lights. Lights and shadows can provide more realism and enhance the immersion in game. Graphics Engine may be improved by incorporating lighting and shadow techniques.

2. NeevEngine does not support the development of games on Xbox 360 or Windows Phone 7. Xbox 360 support can be added by removing external C++ libraries currently being used to provide Server connection support. Windows Phone 7 game development is supported with XNA 4.0 but NeevEngine uses XNA 3.1. XNA 4.0 has deprecated a lot of functionality from 3.1 and streamlined the game development even further. By making NeevEngine compatible to XNA 4.0 it can be used for Windows Phone 7 game development as well.

3. Currently when the player switches from one game screen to another no transition effects are supported. This breaks player immersion and can be improved upon by modifying GameScreenManager class.

4. Even though Model Sanity proved to be quiet helpful, one can write a Level Editor tool that uses NeevEngine as base. This tool would simplify the content creation process even further. Similarly, a UI Designer tool can also be written, which can help the artists create the game UI directly through NeevEngine. Both these tools can output XML/binary files which can be read by NeevEngine after a fresh run, thereby eliminating the need to build the whole project again for minor UI changes.

5. InputManager can be extended further to incorporate support for Gamepads and Kinect. The games developed with those devices in mind can turn out to be even more fun while preserving the educational content.

6. Support for Mesh Bone animations can be added to graphic engine. This will let the artists create animations in 3D software, such as Maya, which can then be utilized by the game. Custom content importers and processors can be written to import other 3D content files besides .fbx files.
7. Inclusion of a Physics Engine to simulate real time weight and motion for 3D objects can enable NeevEngine to be used in more dynamic real world serious games. There are some open source physics engines such as Box2D, Farseer Physics Engine and Physics 2D .net that can be integrated with NeevEngine to provide physics simulation.
REFERENCES


20. Award Abstract #0837162 Accessibility of Materials Laboratory Experience for Engineering Undergraduates


22. Using Second Life As A Business-To-Business Tool

23. Taking a look at A Force More Powerful

24. Basketball Intelligym computer game improves on-court basketball skills

ABSTRACT OF THE THESIS

NeevEngine: Using XNA Game Studio for Serious Game Design and Development

by
Abhishek Sood
Master of Science in Computer Science
San Diego State University, 2012

This paper describes a modular framework for development and rapid prototyping of serious games using XNA framework for Xbox 360 game programing. NeevEngine, a game engine was designed and developed on top of XNA Game Studio using Visual Studio 2008 Express. NeevEngine consists of an internal Graphics Engine, UI framework, Input framework and Quiz Api which form the core components of the engine. The modular design of the engine provides extensible possibilities for further enhancements. NeevEngine was used for development of Mat Isles, a mechanical laboratory simulation where the player goes through the steps involved in Tensile Testing Experiment. The game engine provides support for 3D room rendering, animations, presentation of educational content and presentation of quizzes. All the quiz data is stored in a MySql database with the help of PHP webservice which can be later retrieved. This allows the game writer and content developers to modify and include new content material to further enhance the educational value of the game. Results show that XNA can be a viable platform for creation and development of educational games, minimizing core development work needed to manage art/sound assets and development of basic techniques for 3D game programming. With the help of NeevEngine this process is further streamlined and enhanced thereby enabling the development team to focus on content generation and presentation.