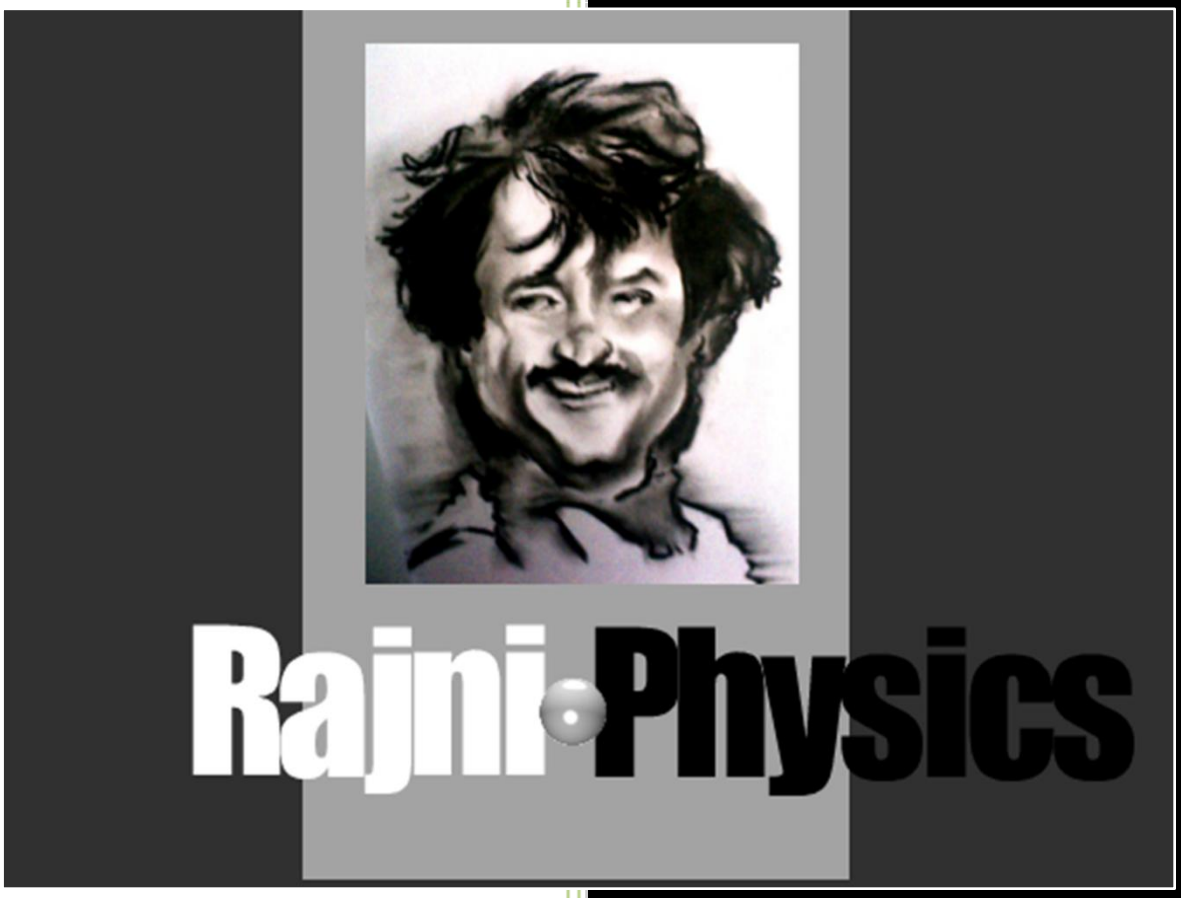
A XNA Based Game



NIDHI JAIN

GOKUL MANTRI

SARIM ZAIDI

BE CMPN 2010-11

RAJNI PHYSICS

Submitted by

Nidhi Jain 07-618

Gokul Mantri 07-632

Sarim Zaidi 07-660

UNDER THE GUIDANCE OF

Prof. Avinash Shrivas

Department of Computer Engineering

Vidyalankar Institute of Technology

Wadala(E), Mumbai 400037

University of Mumbai

2010-2011

CERTIFICATE

This is to certify that

Nidhi Jain 07-618

Gokul Mantri 07-632

Sarim Zaidi 07-660

Have successfully submitted the project report titled

R A J N I P H Y S I C S

A project report submitted to University of Mumbai in fulfillment of Degree course in Computer **Engineering.**

Leading to Bachelor's Degree in Engineering.

2010 – 2011

Under the guidance of Prof. Avinash Shrivas

Signature of Guide Head of Department

Examiner 1 Principal Examiner 2

ACKNOWLEDGMENT

We have immense pleasure in presenting our project titled **“RAJNI PHYSICS”**. We have strived hard to make this project report a success. Along with this we would like to thank many of our advisors.

We would like to take this opportunity to thank those who have provided us with exhaustive guidance and tremendous inspiration throughout the project. We are heartily indebted to and feel a sense of gratitude towards Prof. Avinash Shrivas, our project guide, for giving us valuable guidance, information and also for providing us with his valuable support in every phase of project development.

We would also like to thank our Head of Department, Prof. Varsha Bhosale, for her continuous support and guidance.

We thank all college staff for their sincere support whenever we approached them. We would definitely like to thank our college, Vidyalankar Institute of Technology.

INDEX

Serial No	Topic	Page No.
1.	Project Overview	1
2.	Introduction and Motivation	2
2.1	Theory Behind the Project Concept	4
2.2	Problem statement	5
2.3	Need for the project	6
3.	Analysis and Design	8
3.1	Analysis	8
3.1.1	Process Model Used	8
3.1.2	Preliminary Survey	10
3.1.3	Feasibility Study	13
3.1.4	Cost Analysis	14
3.1.5	Project Time Line Chart	17
3.1.6	Task Distribution	18
3.2	Design	19
3.2.1	UML Diagrams	19
3.3	Technologies Used	24
3.3.1	Hardware and Software	24
3.3.2	Introduction to Programming	28
4.	Implementation Details	36
5.	Test Cases	60
6.	Conclusion	61
7.	Future Work	62
	Appendix I ,II,III	64
	Bibliography	71

LIST OF TABLES AND FIGURES

Serial No	Topic	Page No.
1.	“The Legend Of Vraz” Screen-Shots	12
2.	Cost of project	14
3.	FP estimation	15
4	Time Line Chart	17
5.	Task distribution	18
6.	Use Case Diagram	19
7.	Class Diagram	20
8.	Activity Diagram	21
9.	Component Diagram	22
10.	Sequence Diagram	23
11.	XNA Framework	31
12.	XNA Code Workflow	34
13.	Agile Process Model	39
14.	Iterative Agile Process Model	40
15.	Sprite Sheet	45
16.	OOP Structure of Rajni Physics	55
17.	Screen Shots of Rajni Physics	64

ABSTRACT

Recent years have seen the rise of computer users with PCs, laptops and the pocket-sized computing devices like smartphones, gaming consoles. Worldwide there are over 4 Billion computer users roughly half of the world's population. This pervasiveness of computer devices has created a substantial market for software applications and games that continues to grow extremely rapidly.

Game development industry is currently at its pinnacle. A lot of focus has been shifted to developing enthralling and entertaining gaming products. Game construction encapsulates four tiers, namely graphics, concept, coding and distribution.

One of the premier reasons for picking game development as a final year project is because it involves application of all engineering concepts say physics, programming etc. Also, the motivation for opting for this project was to learn how software architecture practices and processes can improve the final product in an inspiring and practical way. To develop an end user application like game is quite challenging.

Rajni Physics' is a casual game. A casual game is a video game or online game targeted at or used by a mass audience of casual gamers. Casual games can have any type of gameplay, and fit in any genre. XNA framework has been used for the development of the project.

1. Project Overview

The game development substantiates the idea of imparting creativity to technology; we chose to develop a game as we wanted to explore the creative side of coding.

The project was taken with the intention of digging into the heavy programming strategies that was left unexplored during our engineering studies.

We wanted to develop something that has a universal appeal and is not restricted to a particular user group. The game we ought to develop has no age or professional barriers and also tests the analytical and puzzle solving capabilities of a gamer. A game is something which has an entertainment factor and Games are developed as a creative outlet and to generate profit. Well-made games bring profit more readily. However, it is important to estimate game's financial requirements, such as development costs of individual features. Often game projects made with "heart and soul" turn no profit. Failing to provide clear implications of game's expectations may result in exceeding allocated budget.

The development of online community has provided impetus to innovative game developers who can upload their game online to get ratings for the same, we want to deploy our game so that it meets the Microsoft creators club online community standards, so that the numbers of testers are super numerous and belong to plethora of perspective. Hence we want to develop a game using Microsoft developed XNA game framework.

2. Introduction and Motivation

Due to the great distance between design and implementation worlds, different skills are necessary to create a game system. To solve this problem, a lot of strategies for game development, trying to increase the abstraction level necessary for the game production, were proposed. In this way, a lot of game engines, game frameworks and others, in most cases without any compatibility or reuse issues between them, were developed. This required a new generative programming approach, able to increase the production of a digital game by the integration of different game development artifacts, following a system family strategy focused on variable and common aspects of a computer game. As a result, high level abstractions of games, based on a common language, can be used to configure Meta programming transformations during the game production, providing a great compatibility level between game domain and game implementation artifacts.

We have chosen XNA framework to support the game development. The XNA Framework is based on the native implementation of .NET Compact Framework 3.1 for Xbox 360 development and .NET Framework 3.1 on Windows. It includes an extensive set of class libraries, specific to game development, to promote maximum code reuse across target platforms. The framework runs on a version of the Common Language Runtime that is optimized for gaming to provide a managed execution environment. The runtime is available for Windows XP, Windows Vista, Windows 7, and Xbox 360. The XNA Framework encapsulates low-level technological details involved in coding a game, making sure that the framework itself takes care of the difference between platforms when games are ported from one compatible platform to another, and thereby allowing game developers to focus more on the content and gaming experience. The XNA Framework provides support for both 2D and 3D game creation and allows use of the Xbox 360 controllers and vibrations.

As the development of a video game can be a very time consuming process, and independent games are often distributed for free, the question arises why someone would engage in such an endeavor when they could apply their software engineering skills to something which would give a reliable financial return on their investment, such as employment in the video game industry. The reasons vary per individual, but here are a few:

- Personal Gratification

Creating a video game is a challenge, much like playing a video game, and can be very rewarding to complete.

- Developing Engineering Skills

Solving problems in the development of video games will help in solving other problems in engineering.

- Community

Involvement in independent video game communities will generally entail the development of video games.

- Prototyping

Creating a small independent video game can help a developer explore a space where an innovative idea might be further developed into a commercial game. Publishing these games to the community can give valuable feedback.

2.1 Literature Survey

The gaming industry has come a long way in a short period of time, since its humble beginnings more than thirty years ago. Gone are the days when people were thrilled to see a square white block and two rectangular paddles on the screen. Today, exploring three-dimensional worlds in high resolution with surround sound is a common gaming experience. Game development is the latest buzzword. The use of new peripherals, programmed for a wider array of mobile platforms, will develop the new games which will in many respects differ greatly from past. If one wants to secure a place in the game development future, the games should have more fun element and cross the barrier of gender and age limitations.

At the instigation of game development the methodology employed was quite different from what is used now. Earlier, it began with development in C++ without any packages or APIs. It was very disoriented way of programming which didn't lead to creation of high quality games. Initially games were just a sequence of actions without any graphics, getting outcome of each action at every point in the game. Later, the strategy changed and along with amazing graphics, audio was also included in the game. The process went on to become more and more complex and as the recent trends suggest using object oriented methods utilizing the functionalities provided by a game engine or a framework, is the most productive way of game development.

The XNA framework was used for the implementation of Rajni Physics. The XNA framework was selected since it allows developers to create games that can run on both a PC and a game console – the Xbox 360. XNA Game Studio is a Software Development Kit (SDK) with a set of "prebuilt program components" that can be used as part of other programs. Using XNA Game Studio and the underlying language C#.NET (the official language of XNA), video games can be developed for Windows. XNA also allows one to write games for the Xbox 360.

A lot of dissertation has been done on game development. There are many magazines, books, e-books, reports available regarding the process of design and development of game.

2.2 Problem Statement

The project is to develop a win32 based game for Windows platform. Gaming application interestingness depends on the design, story line and realistic gaming experience which are provided by compelling audio and amazing graphics. The Game should be realistic and interactive so that user is engulfed into the whole playing experience.

Making a key decision in game programming is which, if any, APIs and libraries to use. Today, there are numerous libraries available which take care of key tasks of game programming. Some libraries can handle sound processing, input, and graphics rendering. Some can even handle some AI tasks such as path finding. There are many game engines that handle most of the tasks of game programming and only require coding game logic and it is very crucial to select one properly. Using a proper framework will simplify the development effort as we would be work at a higher abstract level rather than at the lowest level, which is quite tedious.

Plot of the game revolves around a lead character .The game would have various levels and interesting modes of play .The character is inspired from a very famous personality of Indian cinema, and also involves a new kind of ‘physics’ characteristic to this personality. The game will have numerous stages involving a conundrum, which should be solved by the gamer in a very unique way.

The game concept revolves around ‘laws of Rajnikanth hence our game is aptly called as ‘RAJNI PHYSICS’.

2.3 Need For the Project

‘Rajni Physics’ is a casual game.

A casual game is a video game or online game targeted at or used by a mass audience of casual gamers. Casual games can have any type of gameplay, and fit in any genre. They are typically distinguished by their simple rules and lack of commitment required in contrast to more complex hardcore games. They require no long-term time commitment or special skills to play, and there are comparatively low production and distribution costs for the developers. Casual games typically are played on a personal computer online in web browsers, although they now are starting to become popular on game consoles and mobile phones, too. Casual gaming demographics also vary greatly from those of traditional computer games.

Some very peculiar points about our game:

- Allows gameplay in short bursts.
- The ability to quickly reach a final stage, or continuous play with no need to save the game.
- Different difficulty levels.

"Casual gamer" is a loosely defined term used to describe a type of video game player whose time or interest in playing games is limited compared with a hard-core gamer. Casual gamers can conceivably consist of any people who show more than a passing interest in video games, therefore it is difficult to categorize them as a group. For this reason, games which attempt to appeal to the casual player tend to strive for simple rules and ease of game play, the goal being to present a pick-up-and-play experience that people from almost any age group or skill level could enjoy. This increases the domain of players who can play our game as we are keeping the playing intensities really simple.

There is no precise classification of casual genres in the modern gaming industry. That can be explained by the easy ideas that form the basis for each game as well as a great amount of genre mixes existing in this field. But if we

were to classify the game it would fall under genre of Arcade - Action game and Puzzle solving.

One of the most intriguing facts about the gaming industry is the nature of the product and the time of presence of product in market. ‘Rajni Physics’ is a very unique game, relating the story with that of a legendary actor ‘Rajnikanth’ helps people relate with the game. Unlike other super heroes in fictions, ‘Rajnikanth’ is the living legend .As of now there is no game based on ‘Rajnikanth’ in the market which makes it one of its kinds.

3. Analysis and Design

3.1 Analysis

3.1.1 Process Model

The significant decision at the initial stage of development is to select a proper game engine or game framework for further development. Today, there are numerous libraries available which take care of key tasks of game programming. Some libraries can handle sound processing, input, and graphics rendering. Some can even handle some AI tasks such as path finding. There are many game engines that handle most of the tasks of game programming and only require coding game logic and it is very crucial to select one properly.

One of the essential modules of game development is movement of character on input from external device and performing stunts like jump, climb, run and other antics. However, the movement should not be free and there has to be constraints on movement, which brings collision detection into action. Collision occurs when the character moves across the play area and collides with a physical object. This is essential to give the game a realistic feel by projecting lifelike environment.

Graphics is the heart and soul of the game. Graphics include the characters and backgrounds, termed as Sprites. The sprite plan depends on the design, the storyline of the whole game. The sprite sheets need to have a transparent background to allow overlapping of layers without depicting the presence of layers of graphics. Development of graphics can be done with any image editor like Adobe Photoshop, Corel draw and others. All the graphics developed needs to be the refined to create a meaningful animation. Animation and audio are major components of a game which make it interesting and exciting. Animation tools like Gif animator, Pivot are simple to use and efficient. Audio engine like Miles is the best available solution for sound development.

After all the initial deployment and development comes the core part of programming that involves physics and artificial intelligence. No self-respecting action game can get without a physics game engine. By means of game physics we mean classical mechanics: the laws that govern how the larger objects move under the influence of laws of gravity and other forces. The objects are made to be felt as solid things with mass, inertia, bounce and buoyancy. Game artificial intelligence is all about making computer characters do things in a human way and to make a simulated environment look like real setup.

3.1.2 Preliminary Survey

Currently, there is a wide range of development being done using XNA. A lot of help in terms of logic of development and pieces of implementation are available online. The nature of help is varied yet certain aspects are still to be explored and answered. Keeping that in mind, we tried to study a game with almost the same features as we desired.

"**The Legend of Vraz**", a 2D action packed adventure arcade game for Windows PC. It is also one of the earliest Microsoft XNA games developed in India and one of the first games in India from an Independent developer. It is also the first game ever made on Indian Miniature painting style.
The game story involves young prince Vraz who is in love with Princess Avi. And to win her hand in marriage he agrees to undertake the five tasks set out for him by Avi's father, the Maharaja of Kund. The five tasks that Vraz must complete are to collect the biggest and brightest red rose, a diamond-studded heart, a traditional horse and a sword; and to earn 100,000 coins for the wedding. However, the Vizier of Kund has other plans. He's in love with Avi himself, and realizes that marrying her is his ticket to becoming the future king of Kund, and perhaps the eventual ruler of the entire region. He is in love with Avi and realizes that if he can marry Avi, he will be the future king of Kund and may be one day of the entire region.

They took the approach of a controller which controls the rendering and hiding of menu screen and passing controls to respective screen. The only game component added was the controller which when required to display a screen, would load its contents on run time and would display the screen. This reduced a lot of memory overload as only the visible contents were loaded in game and others are not. This also made code very simple as in, from only a single class – the class of the controller we could manage all the game screens.

The use of character animation has allowed to create multiple animation of the same character and save them into the file for any third part application such as this game to read and act accordingly. This allowed reusing the same

animation for multiple actions and reducing the size of the total content that was shipped with the game.

Out of the many possible collision algorithms they have used one of the shortest and simplest methods.

Artificial Intelligence is one of the important factors that make the game looks like a real game and the gameplay looks very interactive. It implies that the game is engaging the player and makes them think. First of all the game ensured that characters move on the path assign to them and do the patrolling. This means that the character moves on the platform and when they are on the platform they just stand still and play the appropriate animation. In the other case we have to take appropriate decision when the player character comes in to the visibility of the enemy characters, they take appropriate action such as changing the direction, attack the player or dodge if the player is attacking and some other scenarios like this. This is a part of the AI. But in this game there are different types of enemies and they have different physic so game can take a generic decision. Example: the langur baby can jump as it is light in weight whereas the soldier can't as he has heavy Armour on his body so we have taken decision based on this kind of situations. Even to make the game more realistic, different expressions like happy, sad, angry, die and so on have been included.

Figure 1: “The Legend Of Vraz” Screen-Shots

3.1.3 Feasibility Study

India is fast emerging as a key one-stop destination for game development with research firm AC Nielsen estimating the Indian gaming market to hit $50 million in 2005 and it is expected to be almost $ 500 million by 2010. Game development today is a $20 billion industry in US. India has more than a finger in this lucrative pie. It has emerged as a key one-stop destination for game development.

3.1.3.1 Technology and system feasibility

After thoroughly assessing the outline design of game development in terms of Input, Processes, Output, Fields, Programs, and Procedures, the amount of data stored would be moderate, the frequency of updates would be minimum and as per the general trend in market the game can sustain in market provided that it is of remarkable quality.

A basic level of hardware is required in the development phase of the project. The tools used in development of this game are quite well known and seem to exist in market for long duration. The team's skill rate is quite good and is proficient to complete the project on time.

3.1.3.2 Economic feasibility

The cost involved in development of game is moderate. The main sector of spending is purchasing legal software. With basic, available hardware the project can be developed.

Operational cost involves purchasing of a gaming console in order to deploy and test the game on other systems.

Considering the minimal nature of investment as supportive to large amount of benefits after development, it is a decent choice to opt for this development field.

3.1.3.3 Schedule feasibility

Considering our expertise in the field of game development, we would require eight months to completely design and deliver the end product –"Rajni Physics" and would definitely satisfy the mandatory deadlines.

3.1.4 Cost Analysis

Developing a game for a Windows based system using XNA framework doesn't incur substantial investment.

Item	Type	Cost
Visual C# Express Edition 2008	Software	Free
XNA Game Studio 3.1	Software	Free
Pivot , Stick figure generator	Software	Free
Audacity, Sound editor	Software	Free
Aivary, Image editor	Software	Free
XACT, Sound utility	Software	Free
Abode Creative Suite, demo	Software	Free
Internet facility	Add on	Variable
Microsoft XNA unleashed, Chad Carter	Book	Rs.600/-

Table 1. Cost of Project

Utilizing free wares available the project cost has been cut down tremendously. The Microsoft products can also be freely availed by registering in DreamSpark program.

Information Domain Value	Count		Weighting factor				Result
			Simple	Average	Complex		
External Input(EI)	1	X	3	4	6	=	6
External Output(EO)	2	X	4	5	7	=	14
External Inquiries(EQ)	0	X	3	4	6	=	0
Internal Logical File(ILF)	1	X	7	10	15	=	15
External Interface Files(EIF)	0	X	5	7	10	=	0

Table 2. FP Estimation

Total count: <u>35</u>

To compute functional points (FP), the following relationship is used:

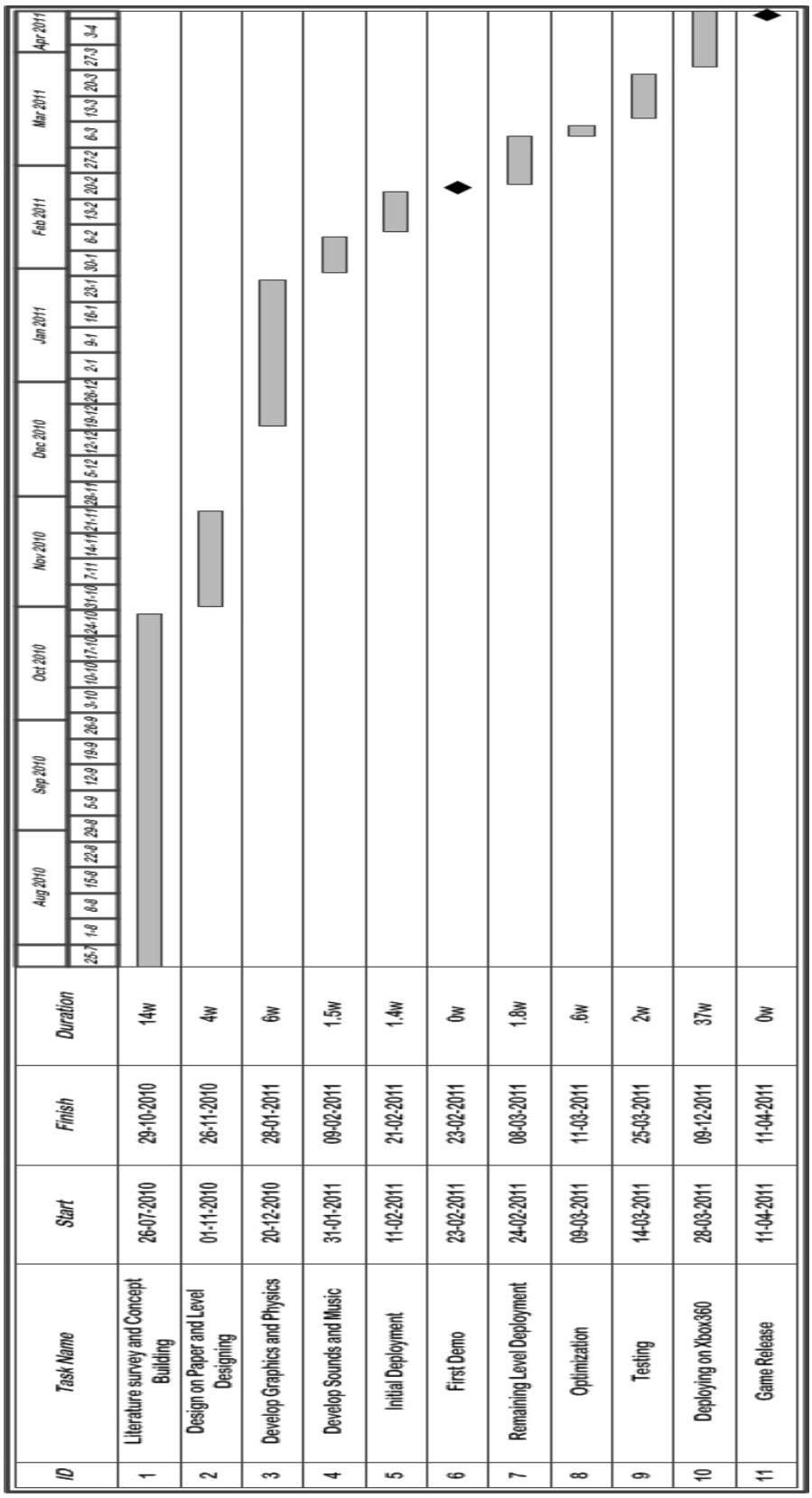
FP = Total count X [0.65 +.01 X Σ (F_i)] = **32.2 ppm**

No.	Factors	Value
1	Does the system require reliable backup and recovery?	2
2	Are the specialized data communications required to transfer information to or from the application?	1
3	Are these distributed processing functions?	0
4	Is performance critical?	4
5	Will the system run in an existing, heavily utilized operational environment?	3
6	Does the system require online data entry?	0
7	Does the online data entry require the input transaction to be built over multiple screens or operations?	1
8	Are the ILFs updated online?	0
9	Are the Inputs, outputs, files or inquiries complex?	3
10	Is the internal processing complex?	3
11	Is the code designed to be reusable?	4
12	Are the conversion and installation included in the design?	2
13	Is the system designed for multiple installations in different organizations?	4
14	Is the application designed to facilitate change and ease of use by the user?	0

Table 3. FP Estimation

$\Sigma (F_I) = \underline{27}$

3.1.5 Project Time Line Chart



ID	Task Name	Start	Finish	Duration
1	Literature survey and Concept Building	26-07-2010	29-10-2010	14w
2	Design on Paper and Level Designing	01-11-2010	26-11-2010	4w
3	Develop Graphics and Physics	20-12-2010	28-01-2011	6w
4	Develop Sounds and Music	31-01-2011	09-02-2011	1.5w
5	Initial Deployment	11-02-2011	21-02-2011	1.4w
6	First Demo	23-02-2011	23-02-2011	0w
7	Remaining Level Deployment	24-02-2011	08-03-2011	1.8w
8	Optimization	09-03-2011	11-03-2011	.6w
9	Testing	14-03-2011	25-03-2011	2w
10	Deploying on Xbox360	28-03-2011	09-12-2011	37w
11	Game Release	11-04-2011	11-04-2011	0w

Figure 2. Time Line Chart

3.1.6 Task Distribution

Task	Details	Duration (hours)
Getting Started with Architecture / framework (NJ, GM ,SZ)	• Study of C# • Study of XNA 2.0, 3.0 ,3.1 • Installing of all SDKs	30 50 5
Story line (NJ, GM, SZ)	• Lead Character • The objective of game • Dialogues • Level Designs	5 5 5 15
Design of Sprites and boards and animations (NJ, GM, SZ)	• Backgrounds • Characters Design • Animations • Sprite Sheets	10 15 20 20
Audio creation (GM)	• Converting • XACT project	10 30
Initialization and Deployment (NJ, GM, SZ)	• Loading of components • Game Logic • Base classes • Debugging	10 20 40 10
Movement and collision detection (NJ, GM, SZ)	• Detecting the instances • Utilizing the features • Physics	20 30 20
Advanced concepts (NJ, SZ)	• Optimization • Game state determination • Heads Up Display • Level implementations	20 30 10 40
Documentation (NJ, GM, SZ)	• Project Report • IEEE paper • Presentation	30 10 10

Table 4. Task Distribution

3.2 Design

3.2.1 UML Diagrams

3.2.1.1 Use Case Diagram

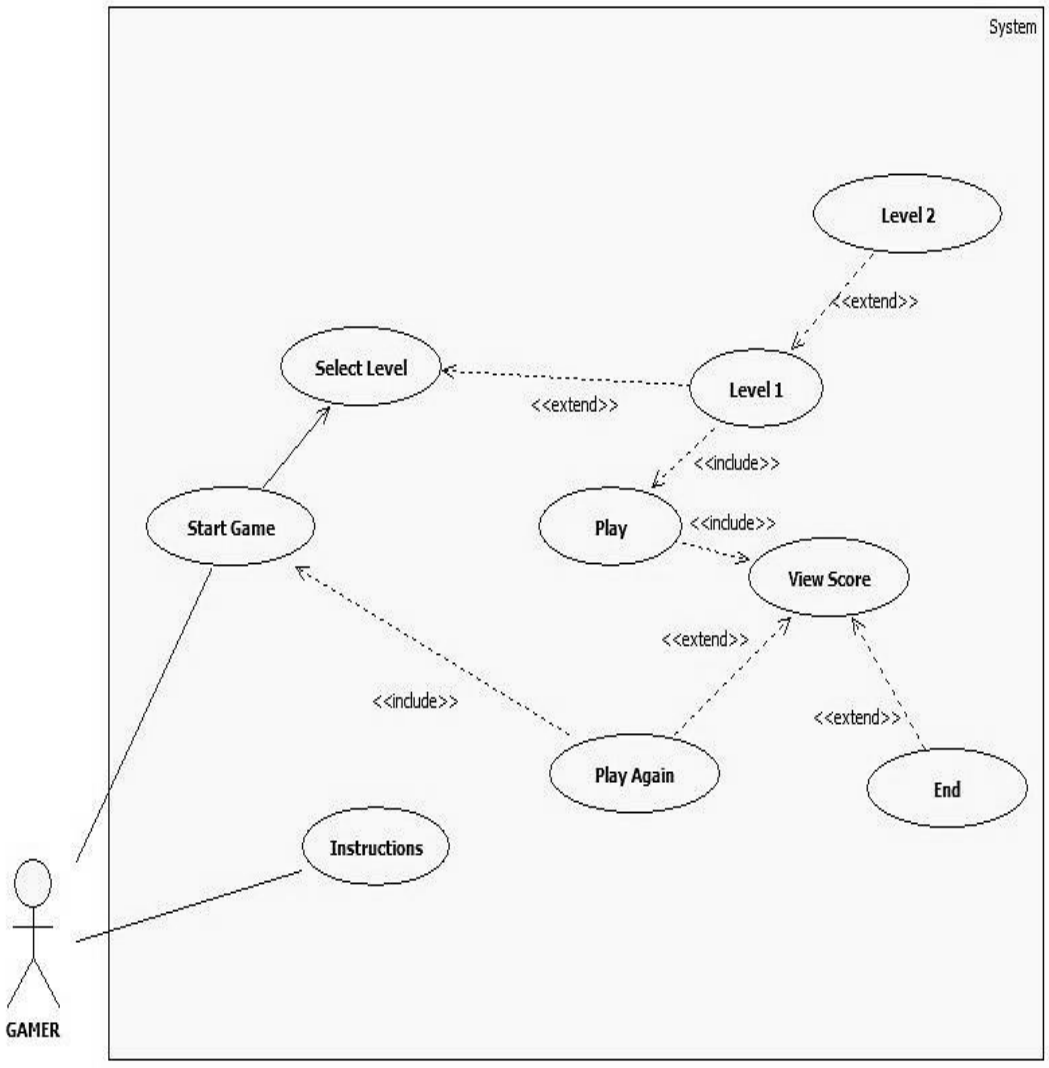


Figure 3. Use Case Diagram

3.2.1.2 Class Diagram

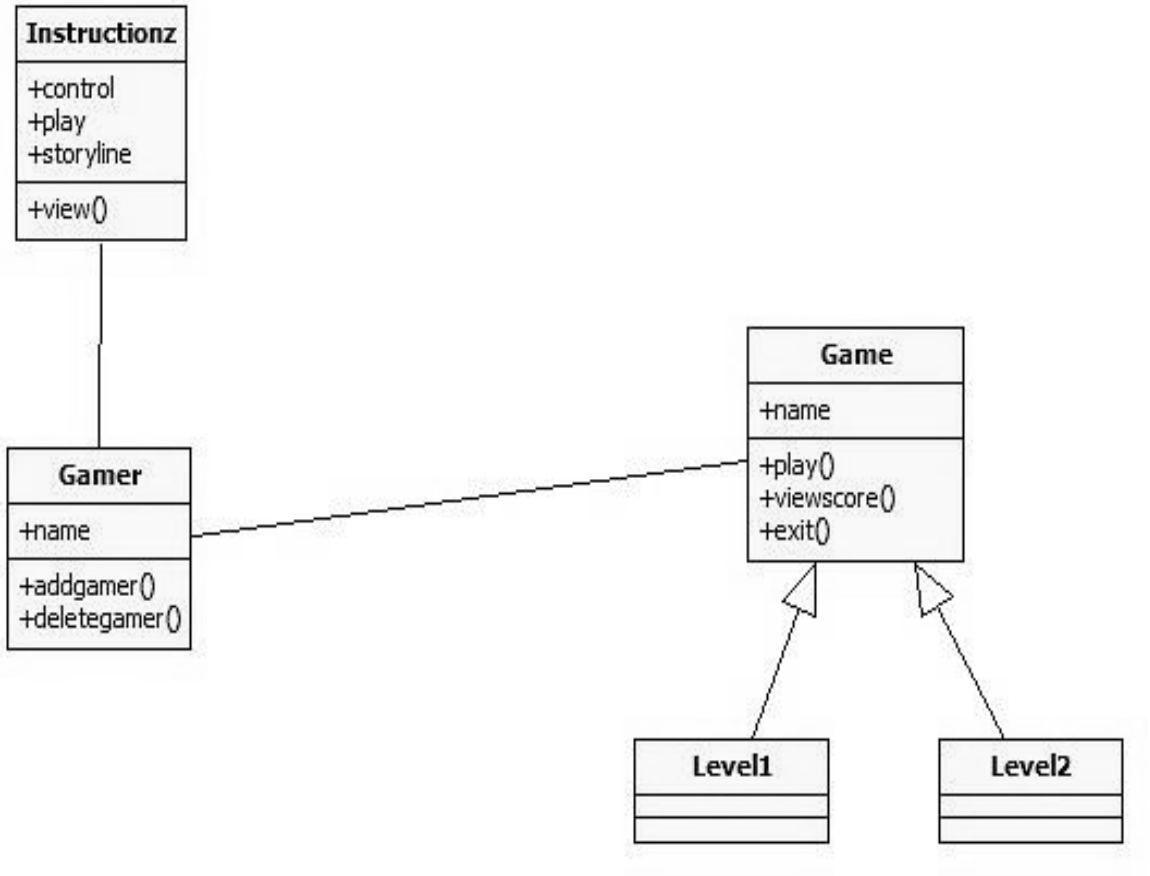


Figure 4. Class Diagram

3.2.1.3 Activity

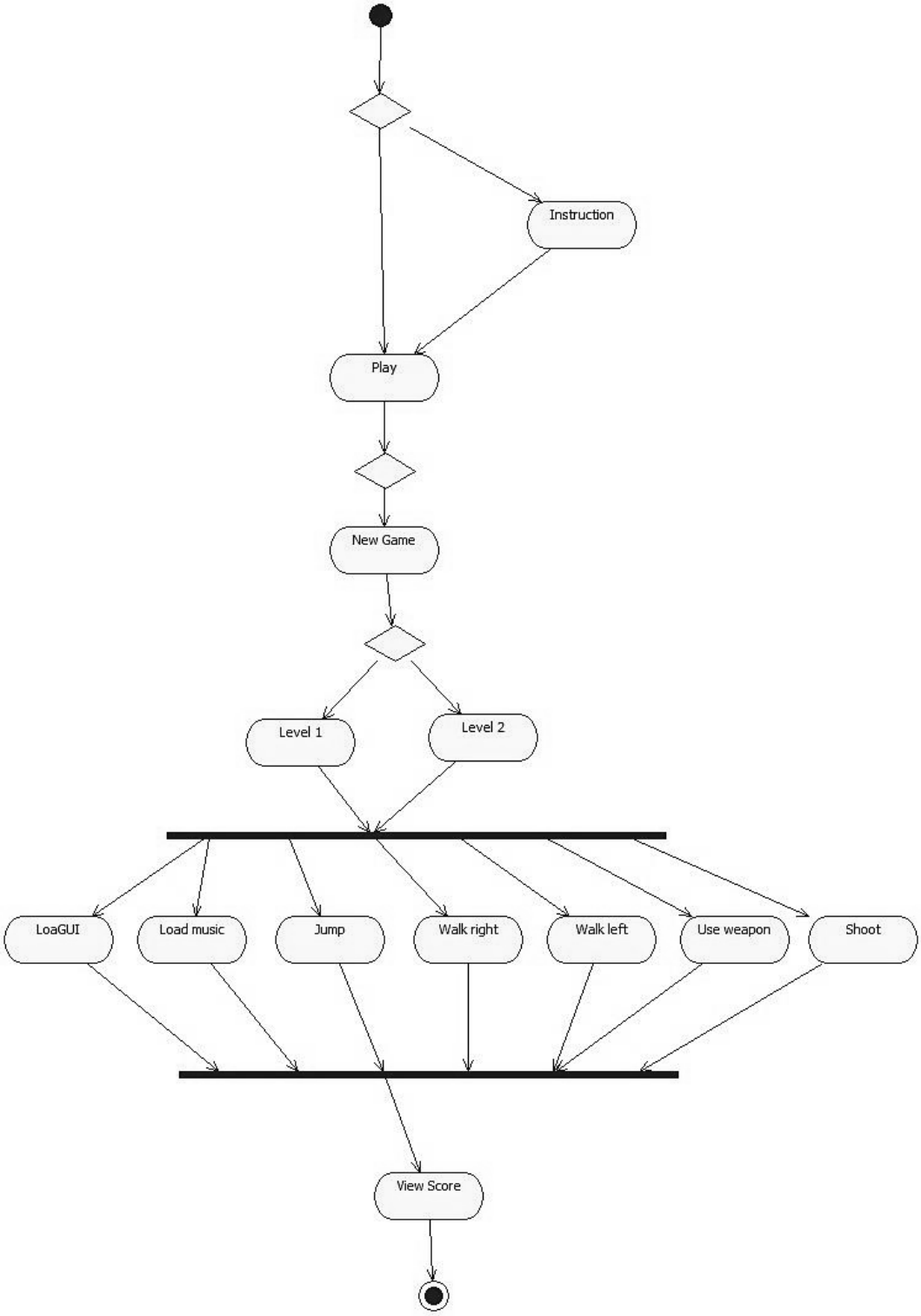


Figure 5. Activity Diagram

3.2.1.3 Component Diagram

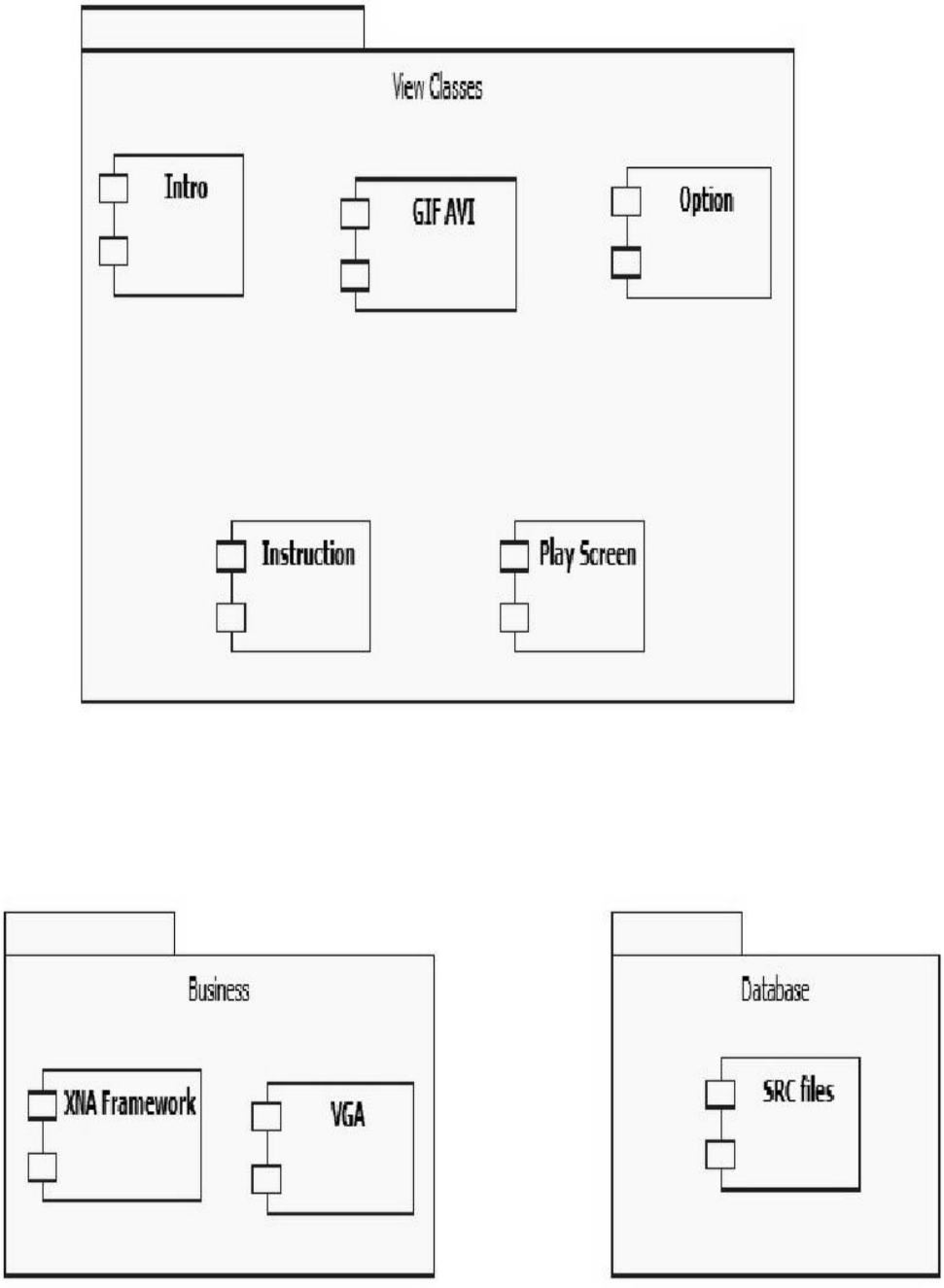


Figure 6. Component Diagram

3.2.1.5 Sequence Diagram

Scenario 1: Playing the game

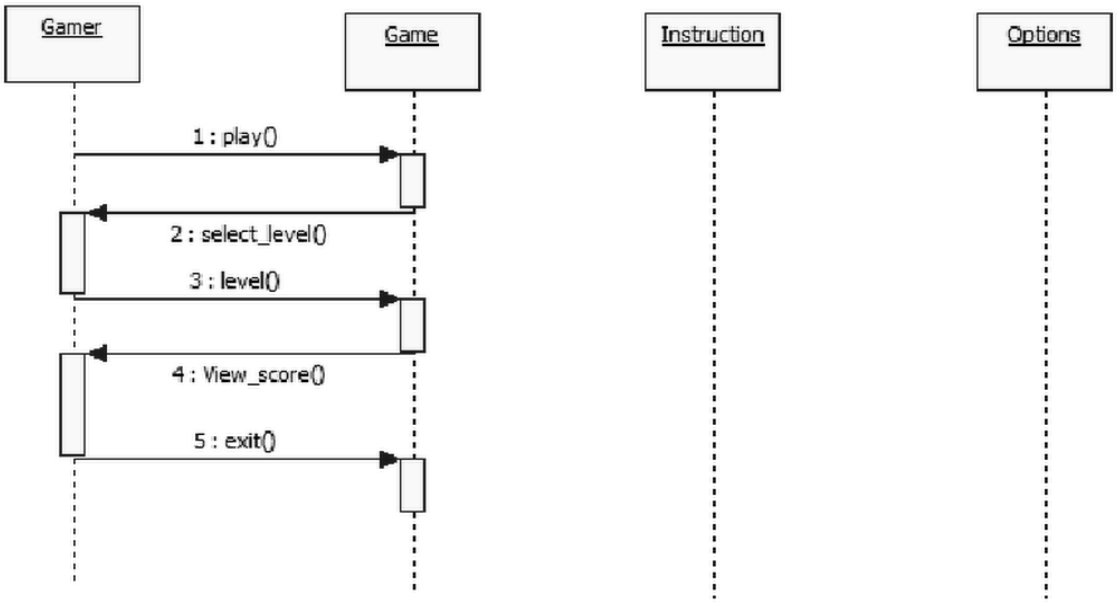
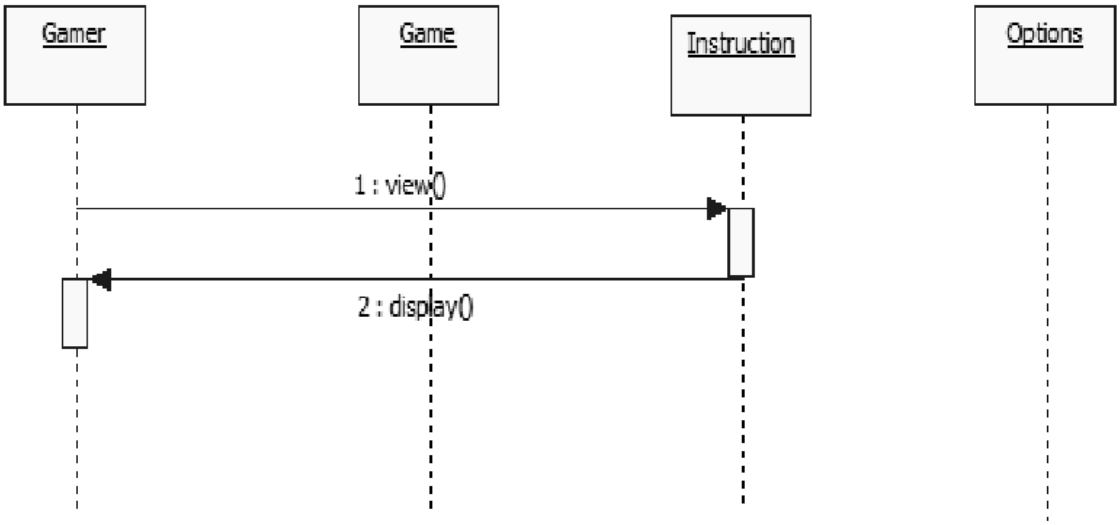


Figure 7. Sequence Diagram

Scenario 2: View Instructions



3.3 Technologies Used

3.3.1 Hardware and Software Requirements

3.3.1.1 Hardware

3.3.1.1.1 A standalone machine with following requirements

- Windows XP, Vista
- 2.xGhz processor
- DirectX 9.0c or higher
- 512MB RAM
- MS-compatible mouse
- MS-compatible Keyboard
- Head Phones

3.3.1.2 Software

3.3.1.2.1 Visual Studio C# Express Edition 2008 SP1

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight. Microsoft Visual C# is Microsoft's implementation of the C# specification, included in the Microsoft Visual Studio suite of products. C# is a multi-paradigm programming language encompassing imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

3.3.1.2.2 XNA frame work 3.1 in .NET

XNA Build is a set of game asset pipeline management tools, which help by defining, maintaining, debugging, and optimizing the game asset pipeline of individual game development efforts. A game asset pipeline describes the process by which game content, such as textures and 3D models, are modified to a form suitable for use by the gaming engine. XNA Build helps identify the pipeline dependencies, and also provides API access to enable further processing of the dependency data. The dependency data can be analysed to help reduce the size of a game by finding content that is not actually used.

3.3.1.2.3 Adobe PhotoShop CS4

Photoshop is used for media editing, animation, and authoring. The Photoshop Document stores an image with support for most imaging options available in Photoshop. These include layers with masks, colour spaces, ICC profiles, transparency, text, alpha channels and spot colours, clipping paths, and duotone settings. This is in contrast to many other file formats, like .GIF that restrict content to provide streamlined, predictable functionality. The output file format can be exported to and from Adobe Illustrator, Adobe

Premiere Pro, and After Effects, to make professional graphics and provide non-linear editing and special effects services, such as backgrounds, textures, and so on, for game. Photoshop is a pixel-based image editor, unlike programs which are vector-based image editors.

3.3.1.2.4. Pivot Stick Figure Animator 3.1

Pivot Stick figure Animator allows creating stick figure animations easily and without any professional artistic skills. It can move the sections of the sick figure and easily create a chain of animation frames that can be previewed as you go. The software allows use of more than one stick-figure in the animation, and even creates new stick figures using an easy to use visual editor that lets you assemble objects out of lines and circles. In addition, enables to optionally set animation size, speed and more. The result can be saved as animated GIF file.

3.3.1.2.5 Adobe Flash CS4

Flash manipulates vector and raster graphics to provide animation of text, drawings, and still images. It supports bidirectional streaming of audio and video, and it can capture user input via mouse, keyboard, microphone, and camera. Flash contains an Object-oriented language called Action Script, which can be used to load images on a particular response by user.

3.3.1.2.6 VLC player

VLC media player is a free and open source media player and multimedia framework written by the VideoLAN project. VLC is a portable multimedia player, encoder, and streamer supporting many audio and video codecs and file formats as well as DVDs, VCDs, and various streaming protocols. It is able to stream over networks and to transcode multimedia files and save them into various formats.

3.3.1.2.7 Audacity

Audacity is a free software, cross-platform digital audio editor and recording application.

- Importing and exporting of WAV, AIFF, MP3, Ogg Vorbis, and all file formats supported by libsndfile library. Support for Free Lossless Audio Codec (FLAC) and additional formats such as WMA, AAC, AMR and AC3 via the optional FFmpeg library.
- Recording and playing back sounds
- Editing via Cut, Copy and Paste (with unlimited levels of Undo).
- Multitrack mixing

3.3.2 Introduction to Programming Language

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create traditional Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Visual C# 2010 provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on version 4.0 of the C# language and version 4.0 of the .NET Framework.

C# syntax is highly expressive, yet it is also simple and easy to learn. The curly-brace syntax of C# will be instantly recognizable to anyone familiar with C, C++ or Java. Developers who know any of these languages are typically able to begin to work productively in C# within a very short time. C# syntax simplifies many of the complexities of C++ and provides powerful features such as nullable value types, enumerations, delegates, lambda expressions and direct memory access, which are not found in Java. C# supports generic methods and types, which provide increased type safety and performance, and iterators, which enable implementers of collection classes to define custom iteration behaviors that are simple to use by client code. Language-Integrated Query (LINQ) expressions make the strongly-typed query a first-class language construct.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# makes it easy to develop software components through several innovative language constructs, including the following:

- Encapsulated method signatures called delegates, which enable type-safe event notifications.
- Properties, which serve as accessors for private member variables.
- Attributes, which provide declarative metadata about types at run time.
- Inline XML documentation comments.

Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.

If you have to interact with other Windows software such as COM objects or native Win32 DLLs, you can do this in C# through a process called "Interop." Interop enables C# programs to do almost anything that a native C++ application can do. C# even supports pointers and the concept of "unsafe" code for those cases in which direct memory access is absolutely critical.

The C# build process is simple compared to C and C++ and more flexible than in Java. There are no separate header files, and no requirement that methods and types be declared in a particular order.

C# programs can consist of one or more files. Each file can contain zero or more namespaces. A namespace can contain types such as classes, structs, interfaces, enumerations, and delegates, in addition to other namespaces. The following is the skeleton of a C# program that contains all of these elements.

```
// A skeleton of a C# program
using System;
namespace YourNamespace
{
class YourClass
{
}

struct YourStruct
{
}

interface IYourInterface
{
}

delegate int YourDelegate();

enum YourEnum
{
}
namespace YourNestedNamespace
{
struct YourStruct
{
}
}
class YourMainClass
{
static void Main(string[] args)
{
//Your program starts here...
}
}
}
```

Code Snippet 1.

XNA Game Studio is an integrated development environment designed to make it easier to develop games for Microsoft Windows, Xbox 360 platforms, and Zune devices. XNA Game Studio extends supported versions of Microsoft Visual Studio tools to support the XNA Framework. The XNA Framework is a managed-code class library that contains functionality targeted specifically to game development tasks. In addition, XNA Game Studio includes tools for incorporating graphical and audio content into your game.

The XNA Framework is designed to be similar to the .NET Framework in terms of its design patterns and idioms. With XNA Game Studio, you are able to incorporate functionality in your game from both the XNA Framework and the .NET Framework. Use the XNA Framework for game-specific tasks such as graphics rendering and managing input, and use the .NET Framework for more general programming tasks.

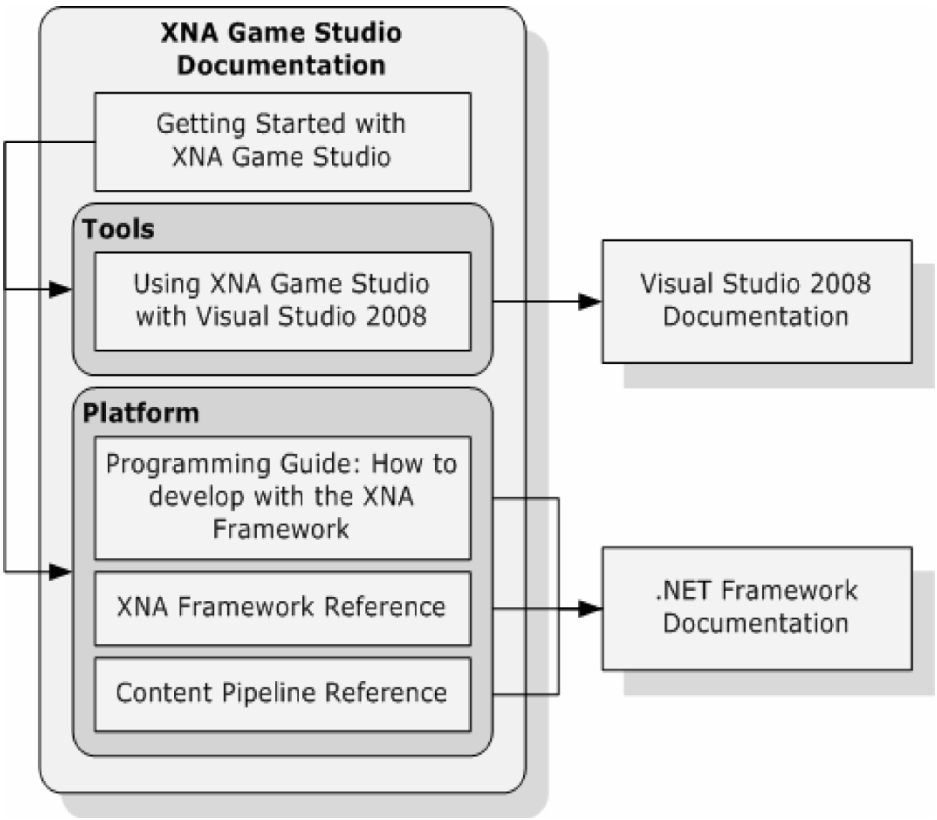


Figure 8. XNA Framework

```
# a skeleton code of a XNA project (game.cs)
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to
before starting to run.
        /// This is where it can query for any required services and
load any non-graphic
        /// related content.  Calling base.Initialize will enumerate
through any components
        /// and initialize them as well.
        /// </summary>

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place
to load
        /// all of your content.
        /// </summary>

        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw
textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
        }
```

```
        /// <summary>
        /// UnloadContent will be called once per game and is the
place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing
audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>

        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
                this.Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }
        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

Code Snippet 2.

The first class-level variable is of the type GraphicsDeviceManager. This is a very important object because it provides the developer, with a way to access the graphics device on your PC, Xbox 360, or Zune. The GraphicsDeviceManager object has a property called GraphicsDevice that represents the actual graphics device on your machine. Because that graphics device object acts as a conduit between your XNAgame and the graphics card on the machine (or more accurately, the Graphics Processing Unit, or GPU, on the graphics card), everything you do on the screen in your XNAgames will run through this object.

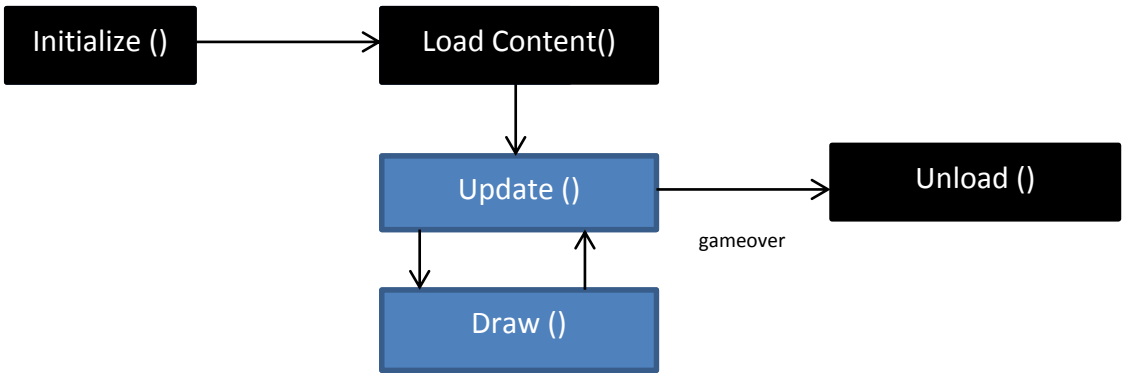


Figure 9. XNA Code Workflow

The second variable is an instance of the SpriteBatch class. This is the core object you'll be using to draw sprites. In computer graphics terms, a sprite is defined as a 2D or 3D image that is integrated into a larger scene. 2D games are made by drawing multiple sprites in a scene (player sprites, enemy sprites, background sprites, etc.). The Initialize method is used to initialize variables and other objects associated with your Game1 object. Your graphics device object will be instantiated at this point and can be used in the Initialize method to help you initialize other objects that depend on its settings. You'll use this method to initialize score values and other such items in future chapters in this book. The LoadContent method is called after the Initialize method, as well as any time the graphics content of the game needs to be reloaded (e.g., if the graphics device is reset due to the player changing the display settings, or something like that). The LoadContent method is where you will load all graphics and other content required by your game, including

images, models, sounds, and so on. Again, as your current project doesn't really do anything exciting, there isn't much happening in this method.

After the LoadContent method finishes, the Game1 object will enter into something known as a game loop. Almost all games use some form of game loop, whether or not they are written in XNA. This is one area where game development differs from typical application development, and for some developers it can take a bit of getting used to.

Essentially, a game loop consists of a series of methods that are called over and over until the game ends. In XNA, the game loop consists of only two methods: Update and Draw. One can think of the game loop in these terms: all logic that affects the actual game play will be done in the Update or the Draw method. The Draw method is typically used, surprisingly enough, to draw things. You should try to do as little as possible in the Draw method other than draw your scene. Everything else needed to run your game (which eventually will involve moving objects, checking for collisions, updating scores, hecking for end-game logic, etc.) should take place in the Update method.

4. Implementation Details

4.1 Story Line

Somewhere in a random location, a Legend, Rajni with no fear is trapped by the enemy of humanity, Adi. He is the one who defines his own laws. All he has with him is not his weapons but the power to run his imagination in any dimension. Adi is out to destroy the world but only one who can destroy him is our saviour, Rajni. Adi has let out his guards to fight Rajni.

At every level of the game, Rajni fights these guards with his unique and out-of-box thinking. There are various obstacles as well for Rajni to tame. He can win over Adi using his physics "RAJNI PHYSICS."

Level 1

Rajni enters an old 3-storey building full of enemy guards and obstacles. On ground floor he faces constantly fired cannon balls, which he can evade by jumping over them, consecutive fire balls will kill him, and then he takes the ladder to 1st floor. Now, he has an enemy to fight with and again a ladder to move onto next floor.

Now, enemy starts running so Rajni has to get catch the enemy before he runs away in order to get the gun. There are assisting power-ups like speed up and speed down to be aware of. Here level one ends.

Level 2

Rajni is trapped inside a graveyard where lots of enemy’s are present. Each floor has two enemies walking on the floor randomly. Now, Rajni has to kill them using the notion of rebounce, by hitting at angles on the wall. He has got limited bullets as well as limited time. Beware even Rajni can die if there is a miscalculation in the re-bounce logic.

Level 3*

Rajni has fallen into a trap, a virtual whirlpool, where a fan in the duct is operating at full speed throwing off objects (clockwise way) and Rajni is floating around (anticlockwise way) .Rajni has to shoot at the object and

destroy them before he is smashed. The whole idea is to make shooting tricky by two separate directional rotating feel.

Level 4*

Rajni finds himself in a maze and the girl of his life, Saee is trapped by a giant, all Rajni has is a limited number of bullets. Once they get exhausted he will get a hunter in his hand, so he has to get some more powers like –

- Sound power play (Ability to kill enemies with sound even before enemy's bullet reaches Rajni, this sound will work only in close proximity.)
- Rajnikantha Turbo shield (He will get a shield which he can place so that enemy's bullet rebounces and kills enemy)
- Coconut-Shoconut (Finally he reaches the Giant enemy and has to be of his size to fight with him, so he drink Coconut-Shoconut to grow in size.)

And finally with all might he saves Saee.

(* These levels are a part of our future work)

4.2 Methodology

Game development is a software development process, as a video game is software with art, audio, and gameplay. Formal software development methods are often snubbed. Games with poor development methodology are likely to run over budget and time estimates, as well as contain a large amount of bugs. Planning is important for individual and group projects alike. Maintenance is mostly non-existent when you make a game, especially when talking about console games. There are exceptions to this like patching and MMOs but for the most part, it's not there. The market is extremely transient by nature. Nothing stays the same in the videogame sector. User expectations keep changing, the definition of fun is the hardest thing to catch, and hardware (especially graphics hardware) is evolving at an alarming rate. There is no magic formula for fun. It cannot be planned, it cannot be predicted. It has to come through trial and error; there simply is no other way. And this is the main problem with the current studies in software engineering. It just doesn't take this "fun factor" issue into account.

So as a first step, we need to create a **game development life cycle**. There are many ways to go about it and there are things that a team would do and other wouldn't do. So first, things need to be chalked out and then recreated with simpler terms (for example function/non-function design documents can be put into a requirements phase, and story can be put into a creative thinking phase. Simple and concise statements should do). So, the game cycle developed is:

- Initial idea developed
- Functional requirements documented
- Story/script created
- Non-function requirements documented
- Research on what'll be needed
- Technical documentation created
- Licensing of required technology
- Coding , asset creation and asset integration
- Quality assurance

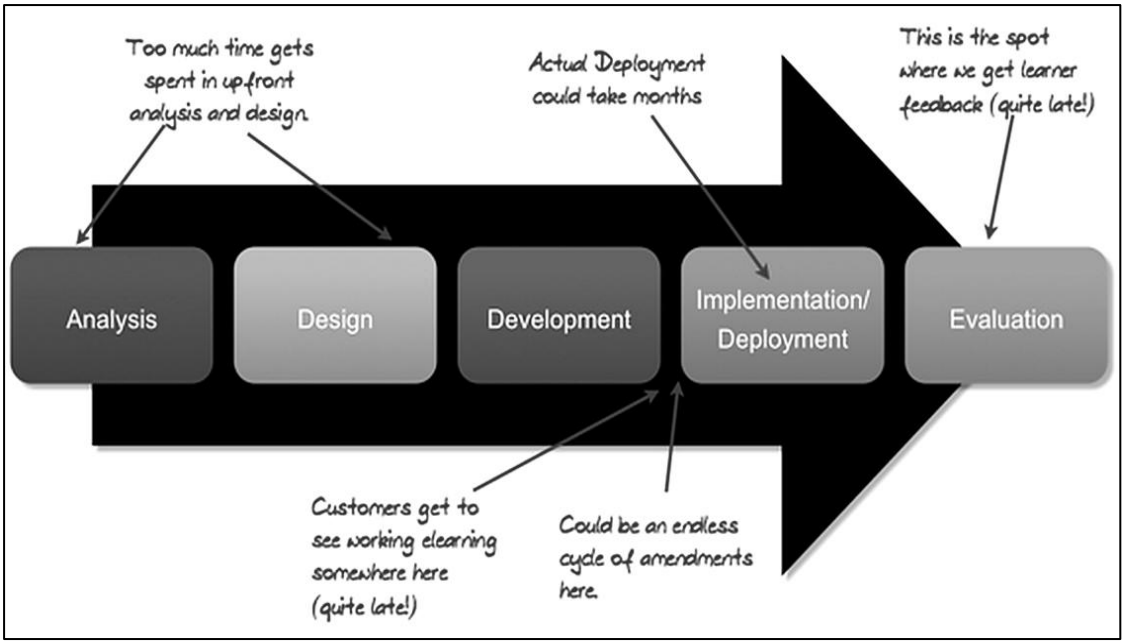


Figure 10. Agile Process Model

The other suitable and one of the more successful methods used is **agile development**. It is based on iterative prototyping, a subset of software prototyping. Agile development depends on feedback and refinement of game's iterations with gradually increasing feature set. This method is effective because most projects do not start with a clear requirement outline. There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

Agile methods break tasks into small increments with minimal planning, and do not directly involve long-term planning. Iterations are short time frames (time boxes) that typically last from one to four weeks. Each iteration involves a team working through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly.

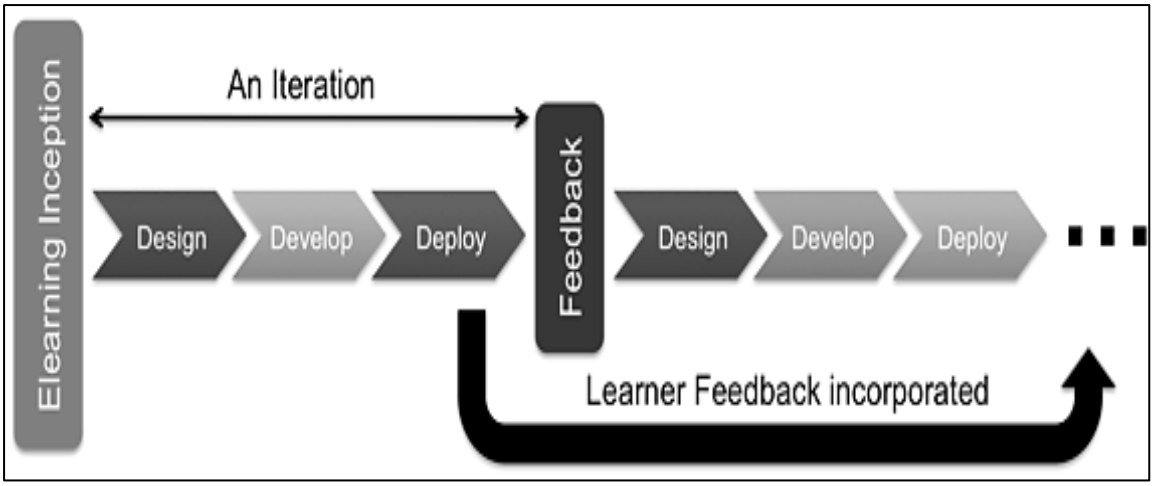


Figure 11. Iterative Agile Process Model

4.3 Code Level Design

4.3.1 To Begin:

To begin with game development, we have to decide upon certain requirements and key ingredients. A story line is very essential for game development. As mentioned above our story line is based on Indian Legend ‘Rajnikanth’.

Choosing upon the game engine or the framework is of utter importance. The coding language depends upon the framework or game engine that is selected. There are many game engines available in market which handles game logic, graphics and audio.

Graphics and sound development can be done using third party tools. So there are many options available.

We choose upon XNA game studio 3.1 which is based on C#. Before XNA 3.1 is installed, C# Express edition 2008 SP1 needs to be installed properly and only then XNA 3.1 should be installed.

Once you create a new XNA game project and simply debug it a blank screen with blue color is the observed along with certain line of code in the program.cs and game.cs files.

4.3.2 Graphics:

A side-scrolling game or side-scroller is a video game in which the viewpoint is taken from the side, and the onscreen characters generally move from the left side of the screen to the right. Games of this type make use of scrolling computer display technology, and sometimes parallax scrolling to suggest added depth.

The screen follows the player character such that the player character is always positioned near the center of the screen. In other games the position of the screen will change according to the player character's movement, such that the player character is off-center and more space is shown in front of the character than behind. Sometimes, the screen will scroll not only forward in the speed and direction of the player character's movement, but also backwards to previously visited parts of a stage. In other games or stages, the screen will only scroll forwards, not backwards, so that once a stage has been passed it can no longer be visited. Another way would be that screen stays stationary and the characters move right-left.

Graphics development for a game can be done by either vector graphics or by adding 2D / 3D images.

2D / 3D images can be created using Photoshop and Maya. Generally speaking there are many image editors and creators available to choose from. Individual components of the background can be created and loaded at different positions while coding the game.

```
GraphicsDeviceManager graphics;
SpriteBatch spriteBatch;
graphics = new GraphicsDeviceManager(this);
graphics.PreferredBackBufferHeight = 600;
graphics.PreferredBackBufferWidth = 800;
Content.RootDirectory = "Content";
spriteBatch = new SpriteBatch(GraphicsDevice);
……
const string broken_wall0 = "Images/broken_wall0";
const string ladder1 = "Images/ladder1";
…….
broken_wall0Texture = theContent.Load<Texture2D>(broken_wall0);
ladder1Texture = theContent.Load<Texture2D>(ladder1);
…….
Texture2D broken_wall0Texture;
Texture2D ladder1Texture;
……..
public Texture2D mbroken_wall0
        {
            get { return broken_wall0Texture; }
        }
  public Texture2D mladder1
        {
            get { return ladder1Texture; }
        }
…….
spriteBatch.Begin();

spriteBatch.Draw(mTextureManager.mbroken_wall0, new Rectangle(410,
250, 400, 200), Color.White);

spriteBatch.Draw(mTextureManager.mladder1, new Rectangle(578, 453,
59, 102), Color.White);

spriteBatch.End();
```

Code Snippet 3.

This piece of code creates a GraphicsDeviceManager which helps to handle all the complexities realeted to graphics.

With the Begin and End calls from the SpriteBatch object, XNA is telling the graphics device that it's going to send it a sprite (or a 2D image). The graphics device will be receiving large amounts of data throughout an XNA game, and the data will be in different formats and types. Whenever you it can process it correctly. Therefore, one can't just call SpriteBatch.Draw anytime you want; you first need to let the graphics card know that sprite data is being sent by calling SpriteBatch.Begin.

4.3.3 Animation:

Computer animation is the process used for generating animated images by using computer graphics. The more general term computer generated imagery encompasses both static scenes and dynamic images, while computer animation only refers to moving images produced by exploiting the persistence of vision to make a series of images look animated. Given that images last for about one twenty-fifth of a second on the retina fast image replacement creates the illusion of movement.

To create the illusion of movement, an image is displayed on the computer screen and repeatedly replaced by a new image that is similar to the previous image, but advanced slightly in the time domain (usually at a rate of 24 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures. For 2D figure animations, separate objects (illustrations) and separate transparent layers are used, with or without a virtual skeleton. Then the limbs, eyes, mouth, clothes, etc. of the figure are moved by the animator on key frames. The differences in appearance between key frames are automatically calculated by the computer in a process known as tweening or morphing. Finally, the animation is rendered.

In, XNA we use the concept of a single sprite-sheet with various different pieces of motion such that when they are run in a loop, it seems like a real motion (this is somewhat like flipbook used for cartoon animation).

```
if ((new Rectangle(130, 315, 39, 41).Intersects(new
Rectangle((int)this.enemyPosition().X, (int)this.enemyPosition().Y,
frameSize.X, frameSize.Y))) && enemyrun == false)
            {
                currentEnemyState = EnemyState.walkingRight;
                sheetSize = new Point(2, 2);
                millisecondsPerFrame = 50;
                enemyTexture = Texture.menemy_walkinr;
                directionEnemy.X = 1;
                directionEnemy.Y = 0;
                speedEnemy = new Vector2(ENEMY_SPEED, ENEMY_SPEED);
                return;
            }
```

Code Snippet 4.

Here, this piece of code is simply checking for position of enemy so that at appropriate instance it can run the walk right animation using the sprite sheet. The millisecondsPerFrame is an important characteristic of animation and the value specified has a significant impact on final outcome. The sheetSize and speedEnemy are used to control the animation.

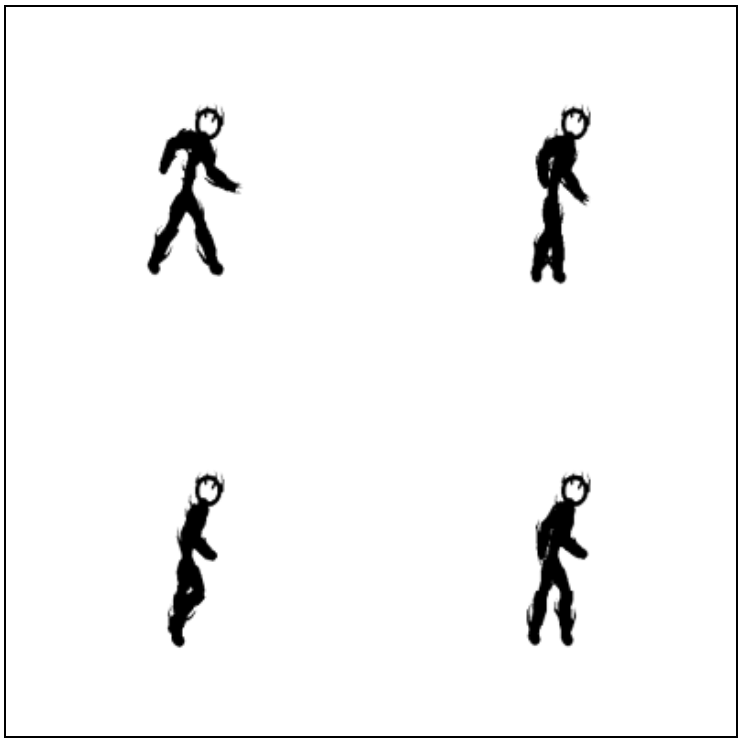


Figure 12. Enemy walking towards right side (sprite-sheet)

4.3.4 Input:

A game controller is a device used with games or entertainment systems used to control a playable character or object, or otherwise provide input in a computer game. A controller is typically connected to a game console or computer by means of a wire, cord or nowadays, by means of wireless connection. Controllers which have been classified as game controllers are keyboards, mice, game pads, joysticks, etc.

The main function of a game controller is to govern the movement/actions of a playable body/object or otherwise influence the events in a video or computer game. The keyboard and mouse are typical input devices for a personal computer and are currently the main game controllers for computer games.

In Rajni Physics, we are only using the keyboard as the main input device. Using XNA, it is very easy to read in the current state of your keyboard. The correct libraries, Microsoft.Xna.Framework.Input, are already linked to by default when XNA project is started, so we can immediately move on to the code that reads in the keyboard input. The Keyboard class has a static method called GetState that retrieves the current state of the keyboard in the form of a KeyboardState structure.

```
Private bool IsSpacePressed()
        {
            if ((mCurrentKeyboardState.IsKeyDown(Keys.Space) == true
&& mPreviousKeyboardState.IsKeyDown(Keys.Space) == false))
            {
                return true;
            }
            return false;
        }
```

Code Snippet 5.

This particular piece of code is simply checking if SpaceBar is pressed, if yes then returns True or else False. The returned value can then be used later to trigger certain actions.

4.3.5 Physics:

Computer animation physics or game physics involves the introduction of the laws of physics into a simulation or game engine, particularly in 3D computer graphics, for the purpose of making the effects appear more real to the observer. Typically, simulation physics is only a close approximation to real physics, and computation is performed using discrete values.

A common aspect of computer games that model some type of conflict is the explosion. Early computer games used the simple expediency of repeating the same explosion in each circumstance. However, in the real world an explosion can vary depending on the terrain, altitude of the explosion, and the type of solid bodies being impacted. Depending on the processing power available, the effects of the explosion can be modeled as the split and shattered components propelled by the expanding gas. This is modeled by means of a particle system simulation. A particle system model allows a variety of other physical phenomena to be simulated, including smoke, moving water, precipitation, and so forth. The individual particles within the system are modeled using the other elements of the physics simulation rules, with the limitation that the number of particles that can be simulated is restricted by the computing power of the hardware. Thus explosions may need to be modeled as a small set of large particles, rather than the more accurate huge number of fine particles.

Physics give a realistic feel to the game. As a gamer interacts with the objects in game the response of the game should be apt so that it makes things seem real. In our game, we have used physics at various instances like while running, walking, jumping (motion physics) and when Rajni shoots at the enemy. All the motion physics have to be dealt while designing the graphics. The real time game scene complexities are handled by code.

```
    if (positionBullet.X < 0 || positionBullet.X > 790 ||/*3rd flr rt
corner*/ (new Rectangle(350, 200, 1, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height)))||/*1st flr rt corner*/ (new
Rectangle(350, 400, 1, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) || /*2nd flr vertical*/(new
Rectangle(450, 200, 10, 40).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) ||/*2nd flr left edge*/ (new
Rectangle(450, 300, 1, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) ||/*3rd flr vertical*/ (new
Rectangle(340, 100, 10, 60).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))))
                {

                    mSpeed.X = -mSpeed.X;
                    positionBullet.X += mSpeed.X;

                }
                if (positionBullet.Y < 0 || positionBullet.Y > 590
||/*1st flr*/ (new Rectangle(0, 400, 350, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) ||/*3rd flr*/ (new
Rectangle(0, 200, 350, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) ||/*2nd flr*/ (new
Rectangle(450, 300, 350, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))) || (new Rectangle(50, 100,
300, 10).Intersects(new Rectangle((int)positionBullet.X,
(int)positionBullet.Y, bulletImage.Width, bulletImage.Height))) ||
(new Rectangle(450, 200, 350, 10).Intersects(new
Rectangle((int)positionBullet.X, (int)positionBullet.Y,
bulletImage.Width, bulletImage.Height))))
                {
                    mSpeed.Y = -mSpeed.Y;
                    positionBullet.X += mSpeed.X;

                }
```

Code Snippet 6.

Taking into consideration, a scenario of firing the bullet. The following code simply checks if the bullet hits the wall and if yes then increase the X or Y values of vector Speed and also modifies position vector

4.3.6 Collision:

Collision detection typically refers to the computational problem of detecting the intersection of two or more objects.

In addition to determining whether two objects have collided, collision detection systems may also calculate time of impact, and report a contact manifold (the set of intersecting points).Collision response deals with simulating what happens when a collision is detected. Solving collision detection problems requires extensive use of concepts from linear algebra and computational geometry.

Physical simulators usually function one of two ways, where the collision is detected a posteriori (after the collision occurs) or a priori (before the collision occurs). In addition to the a posteriori and a priori distinction, almost all modern collision detection algorithms are broken into a hierarchy of algorithms. Often the terms “discrete” and “continuous” are used rather than a posteriori and a priori.

In the a posteriori case, we advance the physical simulation by a small time step, then check if any objects are intersecting, or are somehow so close to each other that we deem them to be intersecting. At each simulation step, a list of all intersecting bodies is created, and the positions and trajectories of these objects are somehow “fixed” to account for the collision. We say that this method is a posteriori because we typically miss the actual instant of collision, and only catch the collision after it has actually happened.

In the a priori methods, we write a collision detection algorithm which will be able to predict very precisely the trajectories of the physical bodies. The instants of collision are calculated with high precision, and the physical bodies never actually interpenetrate. We call this a priori because we calculate the instants of collision before we update the configuration of the physical bodies.

The main benefits of the a posteriori methods are as follows. In this case, the collision detection algorithm need not be aware of the myriad physical variables; a simple list of physical bodies is fed to the algorithm, and the program returns a list of intersecting bodies. The collision detection algorithm

doesn’t need to understand friction, elastic collisions, or worse, nonelastic collisions and deformable bodies. In addition, the a posteriori algorithms are in effect one dimension simpler than the a priori algorithms. Indeed, an a priori algorithm must deal with the time variable, which is absent from the a posteriori problem.

On the other hand, a posteriori algorithms cause problems in the “fixing” step, where intersections (which aren’t physically correct) need to be corrected. Moreover, if the discrete step is not related to objects relative speed, the collision could go undetected, resulting in an object which passes through another, if fast enough.

The benefits of the a priori algorithms are increased fidelity and stability. It is difficult (but not completely impossible) to separate the physical simulation from the collision detection algorithm. However, in all but the simplest cases, the problem of determining ahead of time when two bodies will collide (given some initial data) has no closed form solution—a numerical root finder is usually involved.

Some objects are in resting contact, that is, in collision, but neither bouncing off, nor interpenetrating, such as a vase resting on a table. In all cases, resting contact requires special treatment: If two objects collide (a posteriori) or slide (a priori) and their relative motion is below a threshold, friction becomes stiction and both objects are arranged in the same branch of the scene graph.

We have used the posteriori concept for collision detection, were particular event is triggered on intersecting rectangles.

```
            ball1 = new Rectangle((int)ballPosition1().X,
(int)ballPosition1().Y, 22, 22);
            ball2 = new Rectangle((int)ballPosition2().X,
(int)ballPosition2().Y, 22, 22);
            ball3 = new Rectangle((int)ballPosition3().X,
(int)ballPosition3().Y, 22, 22);
            ball4 = new Rectangle((int)ballPosition4().X,
(int)ballPosition4().Y, 22, 22);
            ball5 = new Rectangle((int)ballPosition5().X,
(int)ballPosition5().Y, 22, 22);
            ball6 = new Rectangle((int)ballPosition6().X,
(int)ballPosition6().Y, 22, 22);


           if (rajni2Rectangle.Intersects(ball1) ||
rajni2Rectangle.Intersects(ball2) ||
rajni2Rectangle.Intersects(ball3) ||
                    rajni2Rectangle.Intersects(ball4) ||
rajni2Rectangle.Intersects(ball5) ||
rajni2Rectangle.Intersects(ball6))
            {
                mCurrentHealth -= 1;

            }
            if (rajni1Rectangle.Intersects(thorn1) ||
rajni1Rectangle.Intersects(thorn2) ||
rajni1Rectangle.Intersects(thorn3))
            {
                mCurrentHealth -= 1;
            }
```

Code Snippet 7.

Here, the rectangles are created around the fireballs and thorns, which when intersects with the rectangle of the character Rajni the health is decreased.

4.3.7 AI:

Game artificial intelligence refers to techniques used in computer and video games to produce the illusion of intelligence in the behavior of non-player characters (NPCs). The techniques used typically draw upon existing methods from the field of artificial intelligence (AI). However, the term game AI is often used to refer to a broad set of algorithms that also include techniques from control theory, robotics, computer graphics and computer science in general.

Since game AI is centered on appearance of intelligence and good gameplay, its approach is very different from that of traditional AI; workarounds and cheats are acceptable and, in many cases, the computer abilities must be toned down to give human players a sense of fairness.

One of the more positive and efficient features found in modern day video game AI is the ability to hunt. AI originally reacted in a very black and white manner. If the player was in a specific area then the AI would react in either a complete offensive manner or be entirely defensive. In recent years, the idea of “hunting” has been introduced; in this ‘hunting’ state the AI will look for realistic markers, such as sounds made by the character or footprints they may have left behind. These developments ultimately allow for a more complex form of play. With this feature, the player can actually consider how to approach or avoid an enemy.

Another development in recent game AI has been the development of “survival instinct”. In-game computers can recognize different objects in an environment and determine whether it is beneficial or detrimental to its survival. Like a user, the AI can “look” for cover in a firefight before taking actions that would leave it otherwise vulnerable, such as reloading a weapon or throwing a grenade. There can be set markers that tell it when to react in a certain way. An example could be if the AI notices it is out of bullets, it will find a cover object and hide behind it until it has reloaded. Actions like these make the AI seem more human.

```
If (rajniPosition().Y < 350 && c==1)
            {
                enemyrun = true;
                Cue enemyRunCue;
                enemyRunCue = msoundBank.GetCue("aiyo amma");
                enemyRunCue.Play();
                currentEnemyState = EnemyState.walkingLeft;
                c = 2;
            }
```

Code Snippet 8.

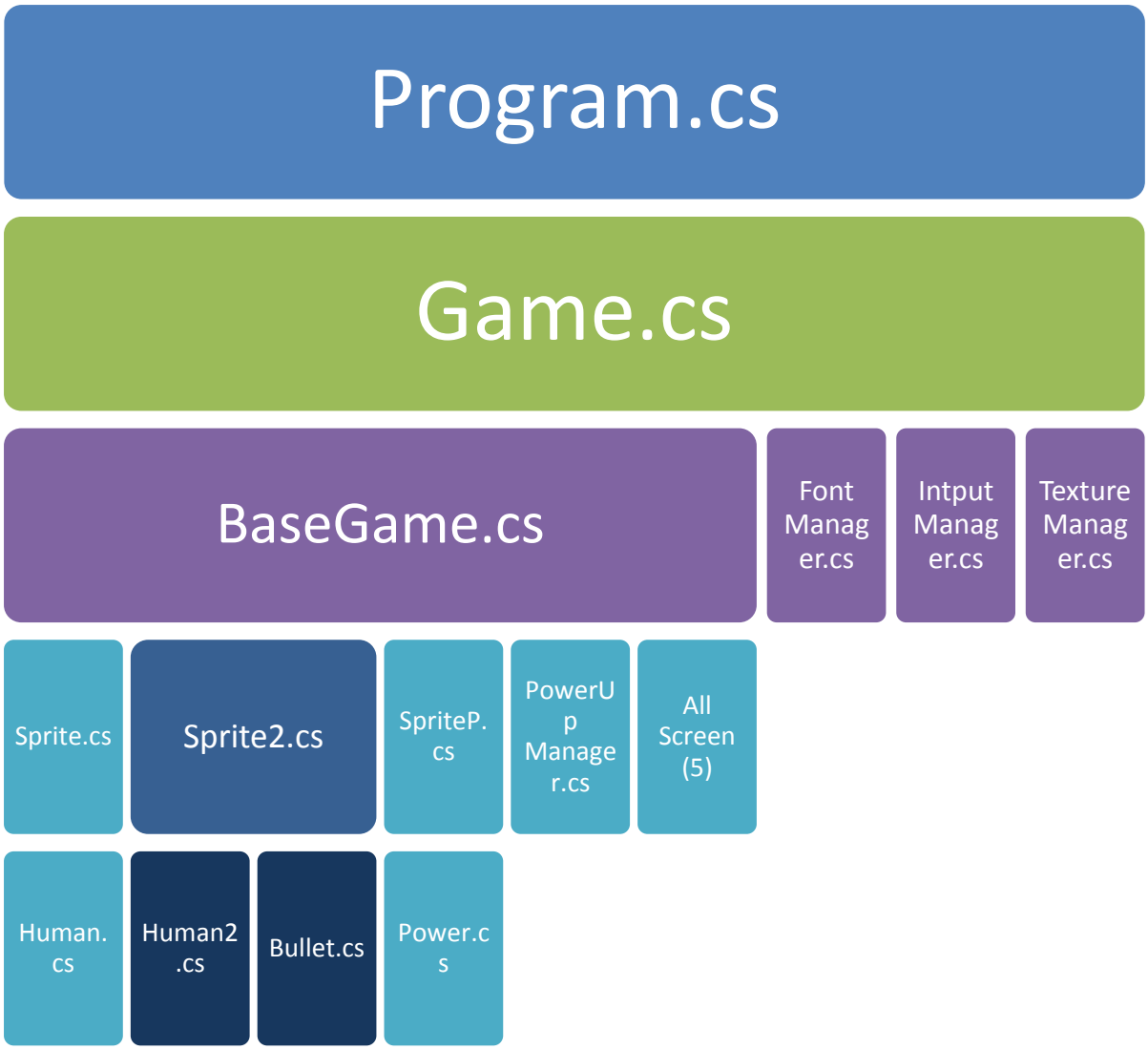
Implementing an evasion algorithm is the opposite of the chase algorithm: detect the position of the player and move the evading object in the opposite direction. This particular piece of code make enemy run once the character Rajni climbs the stairs and reaches the first floor. So now, when enemy starts running it makes decisions like to climb the stairs and run towards the exit on felling the presence of 'Rajnikath', which is an Artificial Intelligence.

4.3.8 Object Oriented Programming form:

Games software has its roots in a "cottage" industry, ignoring formal methodologies, instead leaving the programmers to find homespun solutions to the technical problems faced. The picture has now changed: the scale of the problems faced by programmers means that more methodical techniques must be applied to game development to prevent projects spiraling out of control, both in terms of technical complexity and cost. The best way to keep things simple in its own complexity is to use Object Oriented Form. This allows reuse of code from one level to other, thereby saving space as well as size and at the same time makes it much simple to keep the things under control while programming.

A solid design is just as important as accurate code. Time spent on design in a game development project is absolutely essential to speed up development, increase maintainability, and improve performance. The use of a solid class hierarchy reduces duplicate code and improves maintainability of the system as a whole.

There are in all 19 coded classes in our project. The details of the same and a basic structure of the classes is shown below.



All Screens

- TitleScreen.cs
- IntroScreen1.cs
- HelpScreen.cs
- LevelScreen1.cs
- LevelScreen2.cs
- KeyScreen.cs

Figure 13. OOP Structure of Rajni Physics

4.3.9 Audio:

Video game music is any of the musical pieces or soundtracks and background musics found in video games. It can range from a primitive synthesizer tune to an orchestral piece, usually such that the older the game, the simpler the music. In recent times, many games have had complex soundtracks similar to those of movies, and sometimes even interactive soundtracks which change in order to create an appropriate atmosphere, based on what the player does. It is also much more common for video game soundtracks to be commercially sold or even be performed in concerts that focus on video game music.

With the XNAFramework 3.0, there are a couple of different ways to implement sound. In XNA, developers use a tool called the Microsoft Cross-Platform Audio Creation Tool (XACT) exclusively for audio. Using XACT, developers can create compilations of sound files that are processed by the content pipeline and implemented using the XNAFramework's sound API.

The XACT method of implementing audio offers features beyond those available using the simplified sound API, but as a tradeoff, it is a more complicated method of implementing audio. When dealing with XACT, the actual sound files themselves are not added to the project in Visual Studio.

The other way is to process the .wav files and adding them directly to the XNA project, but this has a disadvantage as you cannot adjust the features of sound like volume variation, infinite loop, pitch variations directly to the sounds, which was possible using XACT. It lacks any of the design-time sound development options available when using XACT, the sound API in XNA 3.0 gets the job done.

Rajni Physics is using the XACT tool for handling the complexities related to audio.

```
AudioEngine audioEngine;
WaveBank waveBk;
SoundBank soundBk;
Cue trackCue;

……

audioEngine = new AudioEngine(@"Content\Audio\RajniS.xgs");
waveBk = new WaveBank(audioEngine,@"Content\Audio\Wave Bank.xwb");
soundBk = new SoundBank(audioEngine,@"Content\Audio\Sound Bank.xsb");

trackCue = soundBank.GetCue("Dhagit dhum");
trackCue.Play();
```

Code Snippet 9.

This particular piece of code loads the project file created using the XACT framework. An audio engine is created which synchronizes all the sounds. The wave bank and sound bank will correspond to the wave and sound bank sections of your XACT file. The Cue object is used to pull out individual cues from the sound bank to play those cues.

4.3.10 Game State Method

Throughout the life of any game, the game will go through different states. The various states in a game are important to detect whether the player has won the game or is still playing it or is dead. This is somewhat the concept of finite state machine (FSM).

State is also important to keep a track of the characters in the game, whether a particular figure is walking right or walking left or if it is jumping. When the game is within a stage only a certain actions will be valid and only certain events will be triggered. At every instance the code keeps checking for the current state of the game or the character.

```
enum GameState
        {
            Running,
            Congratulations,
            GameOverH,
            GameOverE,
            Paused
        }
GameState mCurrentState = GameState.Running;
……..

  switch (mCurrentState)
            {
                case GameState.Running:
                    {
                        //Decrease the time remaining in the game
                        mGameTimeRemaining -=
theGameTime.ElapsedGameTime;
                        //do all checks and collisions detection here
                        mHuman.Update(theGameTime);
                        mPowerUps.Update(theGameTime, mHuman, ref
mGameTimeRemaining);
                        CheckForGameOver();
                        CheckForCongratulations();
                        CheckForPause();

                        break;
                    }

                case GameState.Paused:
                    {
                        CheckForPause();
                        break;
                    }

                case GameState.Congratulations:
                     {
                         break;
                     }

                 case GameState.GameOverH:
                     {
                         CheckForRestart();
                         break;
                     }

                 case GameState.GameOverE:
                     {
                         CheckForRestart();
                         break;
                     }
            }
```

Code Snippet 10.

In this piece of code, we first create certain states of the game and then later using the switch case try to update the states as the code runs.

5. Test Cases

Sr No.	Test Case	Expected Result	Observed Result
1.	Step into the Level 1 on Pressing Enter.	Starts the game.	Level 1 is loaded.
2.	Character Jumps on Pressing Space Bar.	Jump up to a certain height and then fallback to a position at ground without going beyond the floor space.	The character takes a leap and then return to a position on the ground with collision playing their roles properly.
3.	Walk left and right.	Move the character with proper walking animation while walking in any direction.	The character moves smoothly to left and right directions.
4.	Effect of Power Up.	The characters speed should increase or decrease based on the powerUp he uses.	Significant Change in speed of character is observed after using the power up.
5.	End of Level 1	The level should end and a sound should be played after Rajni catches the enemy.	A cry of victory is heard as Rajni catches enemy and Level 2 is started.
6.	Fire the Bullet.	The bullet fired by Rajni should travel according to the laws of motion and should become ineffective after some seconds. The bullet should kill enemies if it strikes them on rebound	Audio is played on firing of bullet which eventually runs out of power after some seconds, killing the enemies in its path. Laws of motion are followed as bullet is rebounded of walls.

6. Conclusion

We have successfully completed the entire deployment and development of ‘Rajni Physics’, a game based on XNA game studio 3.1 utilizing the features of OOPM. The game has compelling audio and attractive graphics. The story line is of great importance hence a lot of thought has been put it after the story.

We have been successful in implementing all the components of a basic Game. They are.

- Collision detection
- Artificial Intelligence
- Animation
- Sound
- Screen states
- State management(using *enum*)
- Physics

Certain features of game are:

- Simple yet attractive graphics
- Audio at every instance and every event
- Heads Up Display (Health Bar)
- Automated enemies
- Player movement and combat actions

Thus, though our Game has less Game time due to the reduction in game levels, we have been successful in implementing all the game components successfully.

7. Future work

‘‘Rajni Physics’ as the name suggests has been developed with a sense that it relates to people, so that people like to play the game by thinking about superstar ’Rajnikanth’.
All the basic and crucial components of XNA game development have been covered in this project; however there are certain potential areas where it could be enhanced.

1. Graphics
Graphics is something that can cause a game to be a hit even if its coding is not extremely good. We have developed graphics using ‘Pivot Stick generator’, this is the section where utilizing more powerful image editor like ‘Autodesk Maya’ and ‘Photoshop’ could come into picture. Animating the characters and displaying more intriguing graphics helps a lot in the overall look and feel of the game. Use of animating software’s like ‘Flash’ can be used for better and more realistic animations, bigger game development projects use tablet P.C’s with pen gesture for creating realistic figures , which are drawn by an artist in the game development team.
Moreover our game use’s 2D graphics which can be enhanced by using 3D graphics, thus bringing the user close to ‘Rajni’.

2. Level Design
We have used static level contrary to the moving background of many known games. Creating moving background asks for an artist who can generate good piece of art that can be put into a loop to generate a moving background.

3. Collision
Collision detection is a very important concept in any game development. The accuracy of the collision is of upmost importance to make a good and interesting game. Out of many collision detection methods we have used ‘intersecting rectangles’ as the basic collision method as it suited our game

the best. Methods like per-pixel collision can be used for more accurate collision in case of very random-shaped objects.

4. Artificial Intelligence

We have tried to include the basic of artificial intelligence, thus it calls for a lot of work in this field as AI is a very huge topic, creating databases and rules for learning can be applied to have a better AI in your game.

5. Sound

We have added sounds at various possible events happening in the game and have created good background scores too, though there can be an option to change the background score according to the users choice.

6. Levels and Settings

Providing options to the user to choose from various levels and an option of saving the game state to be loaded later can be provided, settings for the volume of the music, the brightness, the difficulty level could be provided.

APPENDIX

I. Screen Shots

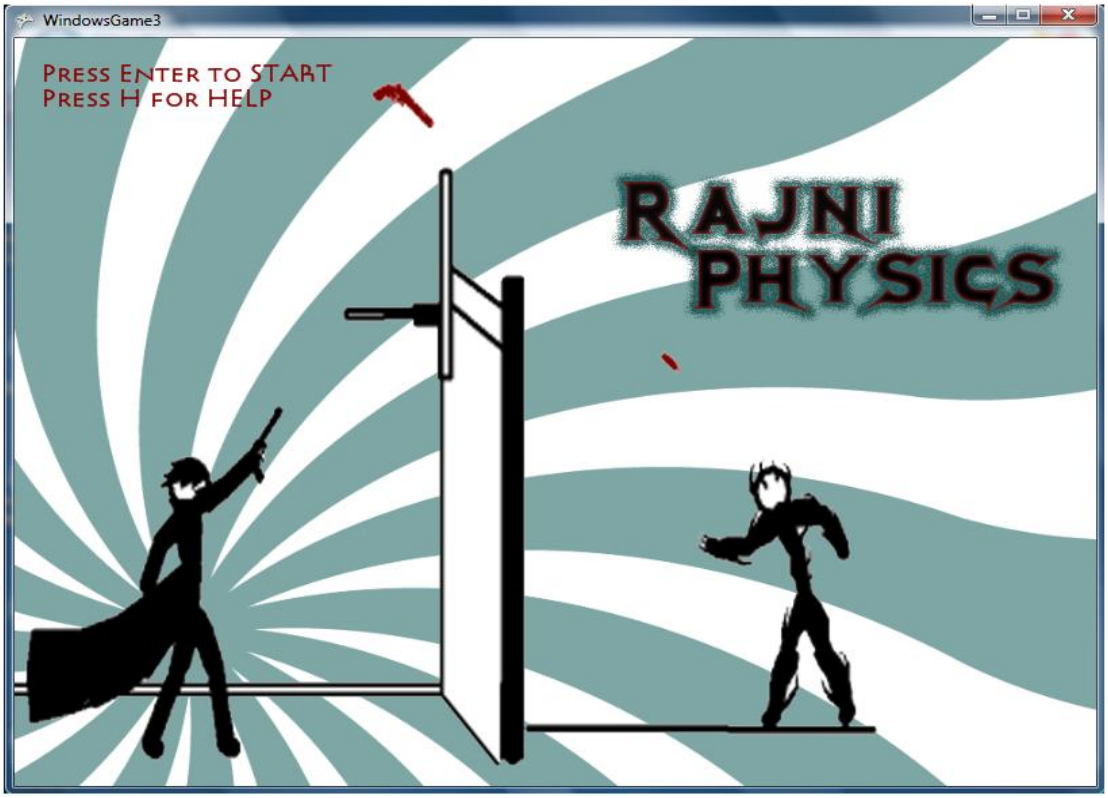


Figure 14. Main Screen

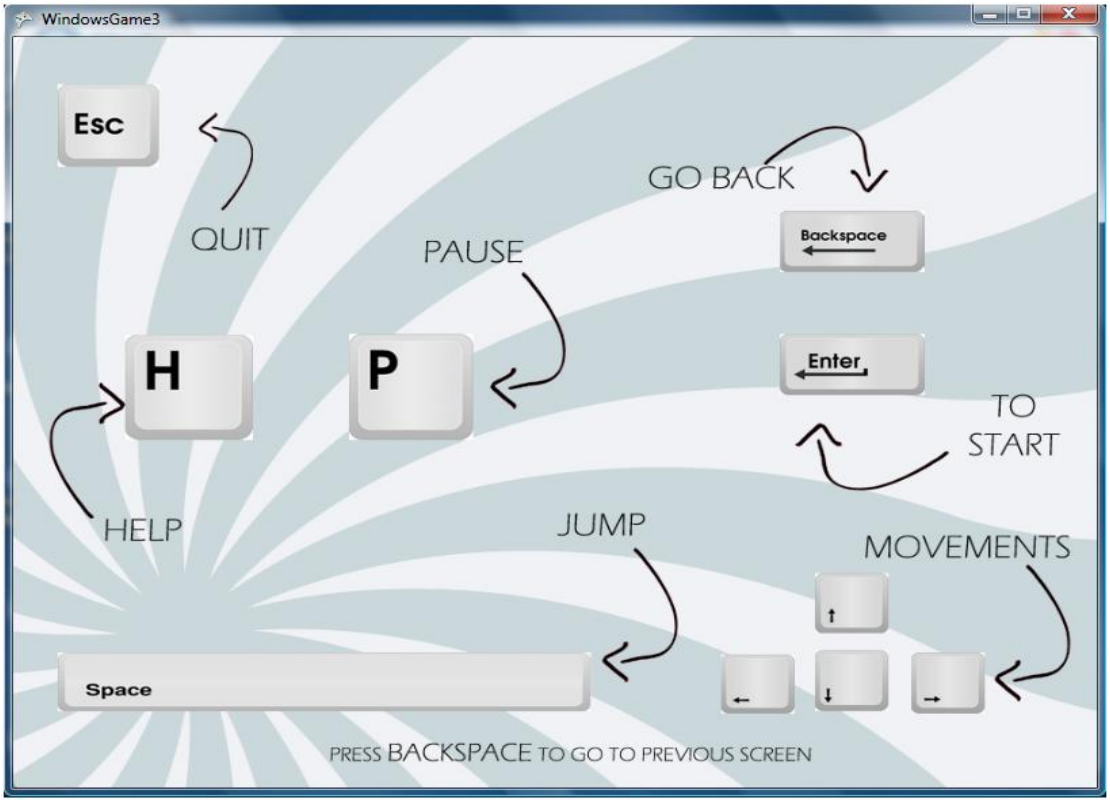


Figure 15. Controls

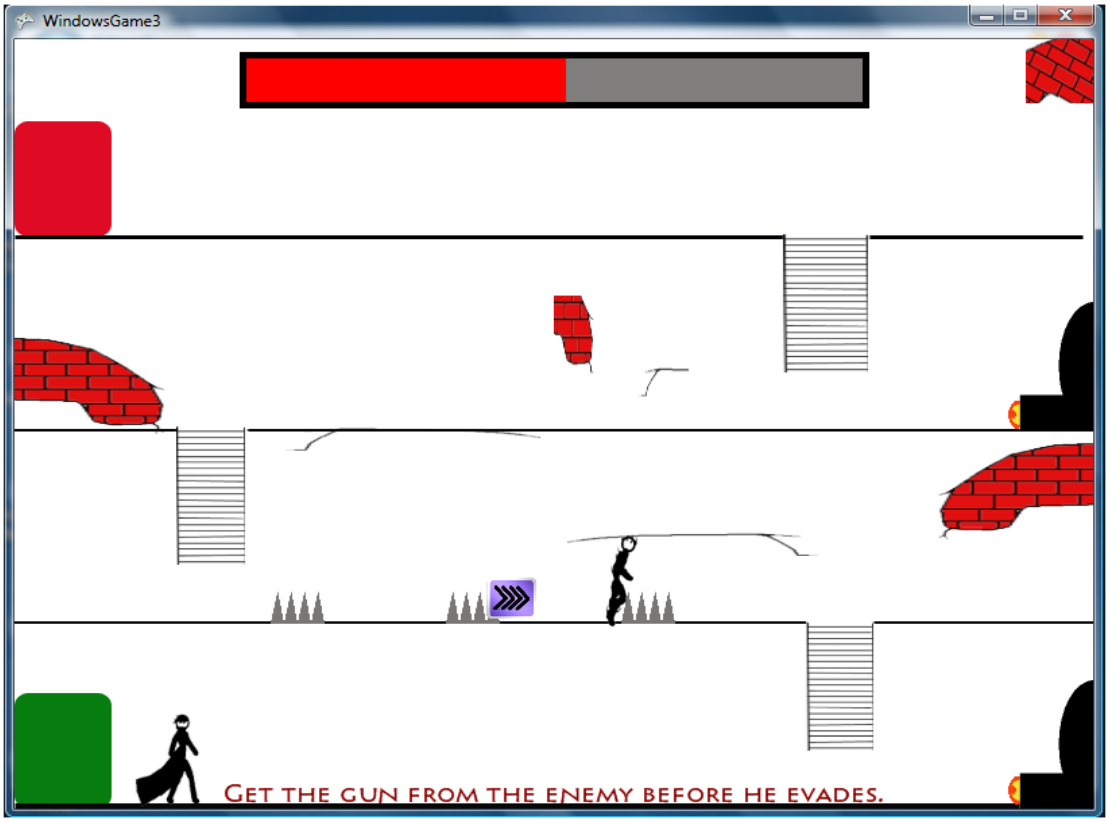


Figure 16. Level 1

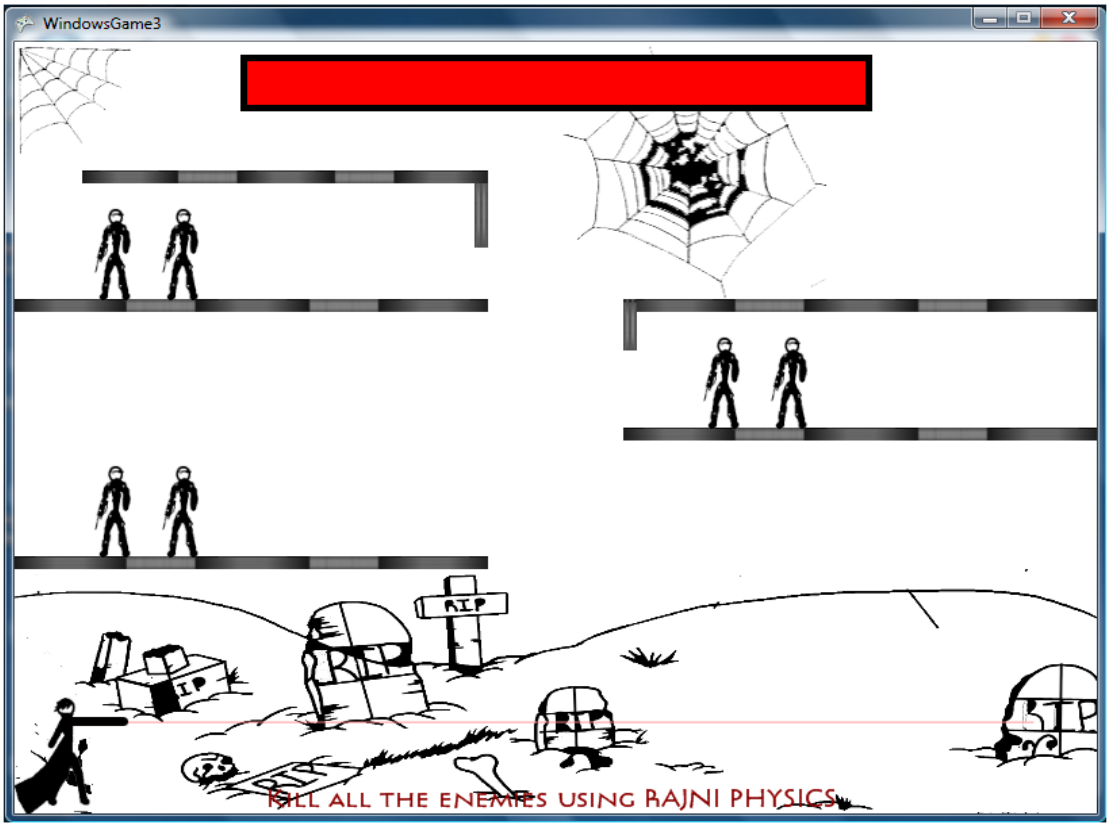


Figure 17. Level 2

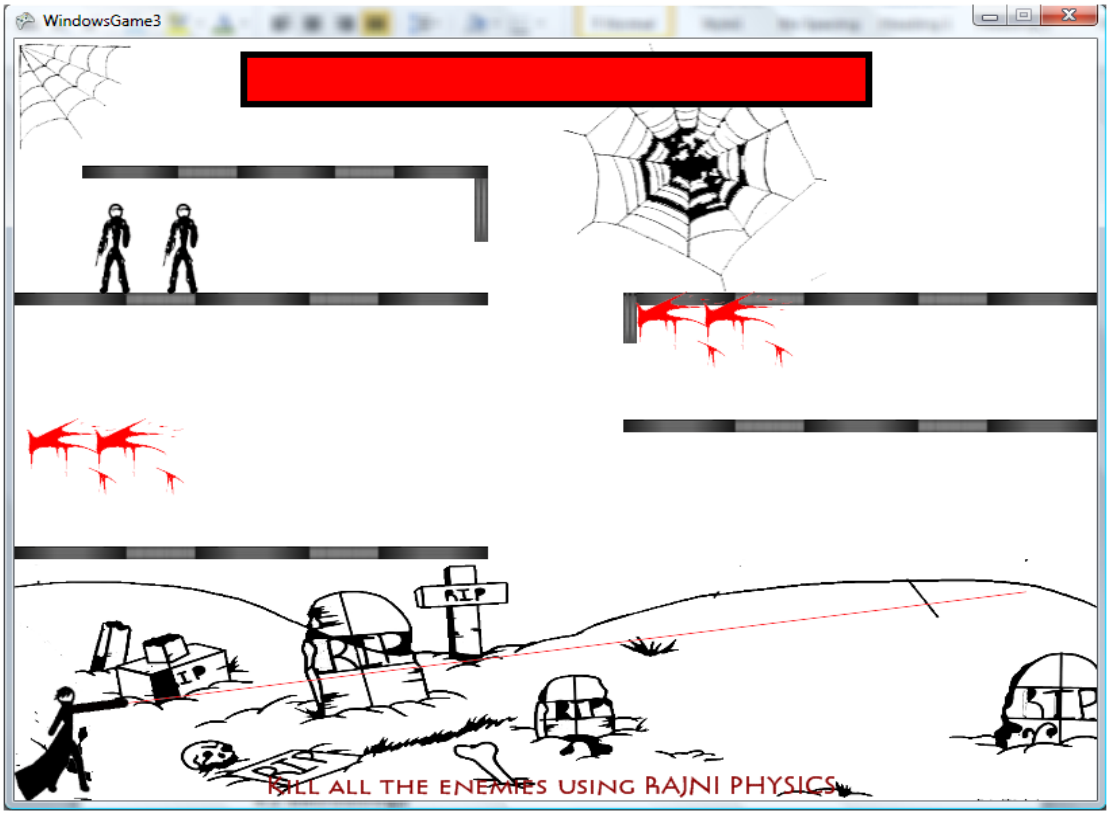


Figure 18. Level 2

II.User Manuals

Once we open the main screen of Rajni Physics, there are two options either

- Start game (press Enter)
- Help screen (press H)

Helpscreen provides all the details about the game and also the controls.

After the game is started, use help of following controls to play the game

Key	Action
Arrow keys	Direction
Space Bar	Jump
Enter	Restart / Next Level
A	Shoot
Escape	Quit

Table a. Key board Controls

Currently, game consists of two levels which can be played one after the other.

III. Classes & External Libraries

The XNA Framework class library is a library of classes, interfaces, and value types that are included in XNA Game Studio. This library provides access to XNA Framework functionality and is designed to be the foundation on which XNA Game Studio applications, components, and controls are built.

A .Microsoft.Xna.Framework

Provides commonly needed game classes such as timers and game loops.

Name	Description
Game	Provides basic graphics device initialization, game logic, and rendering code.
GameTime	Snapshot of the game timing state expressed in values that can be used by variable-step (real time) or fixed-step (game time) games.
GraphicsDeviceManager	Handles the configuration and management of the graphics device.
Vector2	Defines a vector with two components.
Vector3	Defines a vector with three components.

B. Microsoft.Xna.Framework.Audio

Contains low-level application programming interface (API) methods that can load and manipulate XACT-created project and content files to play audio.

Name	Description
AudioEngine	Represents the audio engine. Applications use the methods of the audio engine to instantiate and manipulate core audio objects.
Cue	Defines methods for managing the playback of sounds.
SoundBank	Represents a sound bank, which is a collection of cues.
WaveBank	Represents a wave bank, which is a collection of wave files.

C. Microsoft.Xna.Framework.Content

Contains the run-time components of the Content Pipeline.

D. Microsoft.Xna.Framework.Design

Provides a unified way of converting types of values to other types.

E. Microsoft.Xna.Framework.GamerServices

Contains classes that implement various services related to gamers. These services communicate directly with the gamer, the gamer's data, or otherwise reflect choices the gamer makes. Gamer services include input device and profile data APIs.

F. Microsoft.Xna.Framework.Graphics

Contains low-level application programming interface (API) methods that take advantage of hardware acceleration capabilities to display 3D objects.

Name	Description
GraphicsDevice	Performs primitive-based rendering, creates resources, handles system-level variables, adjusts gamma ramp levels, and creates shaders.
Texture2D	Represents a 2D grid of texels.
SpriteBatch	Enables a group of sprites to be drawn using the same settings.
SpriteFont	Represents a font texture.

G. Microsoft.Xna.Framework.Graphics.PackedVector

Represents data types with components that are not multiples of 8 bits.

H. Microsoft.Xna.Framework.Media

Contains classes to enumerate, play, and view songs, albums, playlists, and pictures.

I. Microsoft.Xna.Framework.Net

Contains classes that implement support for Xbox LIVE, multiplayer, and networking for XNA Framework games.

J. Microsoft.Xna.Framework.Storage

Contains classes that allow reading and writing of files.

K. Microsoft.Xna.Framework.Input

Contains classes to receive input from keyboard, mouse, and Xbox 360 Controller devices.

Name	Description
Keyboard	Allows retrieval of keystrokes from a keyboard input device.
KeyboardState	Represents a state of keystrokes recorded by a keyboard input device.

BIBLIOGRAPHY

[1] Computer Industry Almanac Inc., www.c-i-a.com

[2] Chad Carter , 'Microsoft XNA unleashed', 2008

[3] Aaron Reed 'Learning XNA 3.0', 2008

[4] Riemer Grootjans XNA 3.0 'Game Programming Recipes', 2009

[5] Jesse Liberty, Donald Xie 'Programming C# 3.0 ', 2008

[6] Herbert Schildt 'C#: a beginner's guide', 2008

[7] Computer Industry Almanac Inc., www.c-i-a.com

[8] XNA creators club, www.xna-creators.com

[9] XNA App Hub, create.msdn.com

[10] Riemers XNA tutorial www.riemers.net

[11] Johan Omark's XNA tutorials ,www.xnatutotrial.com

[12] The Future of Game Development Trends ,www.cigame.com/future-game-development-trends.html

[13] Game developer, UBM TechWeb

[14] Games, Kappa Publishing Group, Inc

[15] Rasha Morsi, Chad Richards and Mona Rizvi “Work in Progress – BINX: A 3D XNA Educational Game for Engineering Education”, Norfolk State University, 2010

[16] Lorenzo Cantoni and Nadzeya Kalbaska “The Waiter Game - Structure and Development of a Hospitality Training Game”, Lugano, Switzerland, 2010.

[17] Zatungames Inc., www.zatungames.com

[18] Casual Games, http://en.wikipedia.org/wiki/Casual_games

[19] Alan Thorn, ‘Game Engine Design and Implementation’, 2011

[20] Perry, Sherrod,’Essential XNA Game Studio 2.0 Programming’, 2008

[21] Ian Millington, ‘Game physics engine development’, 2007

[22] John Funge ,’Artificial Intelligence for Games’, Ian Millington, 2009