ATMIYAUNIVERSITY

RAJKOT

A

Report On

DOOM

Under subject of

PROJECT

B. TECH Semester– VIII

(Computer Engineering)

Submitted by:

1.	Pushpak Gohil	190002067
2.	Purvit Dhaduk	190002026

Prof. Krina Masharu

(Faculty Guide)

Prof. Tosal M. Bhalodia

(Head of the Department)

Academic Year

(2022-23)

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this project entitled "**DOOM"** submitted towards completion of project in **7th Semester** of B.Tech. (Computer Engineering) is an authentic record of our original work carried out under the guidance of "**Prof. Krina Masharu".**

We have not submitted the matter embodied in this project for the award of any other degree.

Semester:7th

Place: Rajkot

Signature:
Pushpak Gohil (190002035)
Purvit Dhaduk (190002026)

ATMIYA

UNIVERSITYRAJKOT

<u>CERTIFICATE</u>

Date:

This is to certify that the “**DOOM**” has been carried out by **Pushpak Gohil** and **Purvit Dhaduk** under my guidance in fulfillment of the subject Project in COMPUTER ENGINEERING(7thSemester) of Atmiya University, Rajkot during the academic year 2022-23.

Prof. Krina Masharu	Prof.Tosal M.Bhalodia
(Project Guide)	**(Head of the Department)**

ACKNOWLEDGEMENT

We have taken many efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to Prof. Janak Maru for their guidance and constant supervision as well as for providing necessary information regarding the Major Project titled **"DOOM".** We would like to express our gratitude towards staff members of Computer Engineering Department, Atmiya University for their kind co-operation and encouragement which helped us in completion of this project.

We even thank and appreciate to our colleague in developing the project and people who have willingly helped us out with their abilities.

Pushpak Gohil (190002035)
Purvit Dhaduk (190002067)

ABSTRACT

DOOM is an adventure game for windows computer. You must tackle all theobstacles to level up your powers. There are amazing levels which can check your gaming skills too. You can play this game and enjoy it.

INDEX

Sr. No.	TITLES			Page No.
	Acknowledgement			**I**
	Abstract			**II**
	Index			**III**
	List of Figures			**VI**
	List of Tables			**VII**
1.	**Introduction**			**1**
	1.1	Introduction		1
	1.2	Purpose		1
	1.3	Scope		1
	1.4	Feasibility study		2
		1.4.1	Operational Feasibility	2
		1.4.2	Technical Feasibility	2
		1.4.3	Economical Feasibility	3
	1.5	Organization of the report		3
		1.5.1	Introduction	3
		1.5.2	Software Requirements Specification	3
		1.5.3	Design &Planning	3
		1.5.4	Results and Discussion	3
		1.5.5	Summary and conclusion	3
2.	**Software Requirements Specification**			**4**
	2.1	Hardware Requirement		4
	2.2	Software Requirement		4
3.	**Design &Planning**			5
	3.1	Software Development Life Cycle Model		5
		3.1.1	Waterfall Model	5
	3.2	General overview		5
	3.3	DFD(data flow diagram)		6
	3.4	Use case Diagram		8
	3.5	Class Diagram		9
	3.6	Input / Output Interface		10

4.	**Implementation Details**				**12**
	4.1	Front End			12
		4.1.1	Unity3D		12
	4.2	Back End			12
		4.2.1	C#		12
5	**Testing and Implementation**				**13**
	5.1	Unit Testing			13
		5.1.1	Introduction		13
		5.1.2	Benefits		13
	5.2	Integration Testing			14
		5.2.1	Purpose		14
			5.2.1.1	Big Bang	15
			5.2.1.2	Top-down And Bottom-up	15
	5.3	Software Verification and Validation			16
		5.3.1	Introduction		16
		5.3.2	Classification Of Methods		17
		5.3.3	Test Cases		17
			5.3.3.1	Test Suit	17
	5.4	Black-Box Testing			17
		5.4.1	Test Procedures		17
		5.4.2	Test Cases		18
	5.5	White-Box Testing			18
		5.5.1	Levels		18
		5.5.2	Procedures		19
		5.5.3	Advantages		19
		5.5.4	Disadvantages		19
	5.6	System Testing			20
6	**Limitations**				21
7	**Conclusion**				22

LIST OF FIGURES

Figure No.	Table Title		Page No.
3.3	Data Flow Diagram		6
	3.3.1	Level-0	6
	3.3.2	Level-1	6
	3.3.3	Level-2	7
3.4	Use case Diagram		8
3.5	Class Diagram		9
3.6	Input/ Output Interface		10
	3.6.1	Interface	10
	3.6.2	Main Menu	10
	3.6.3	Enemy View	11
	3.6.4	Game Map	11

LISTOFTABLES

Table No.	Table Title	Page No.
2.1.1	**Hardware Requirements**	**4**
2.2.1	**Software Requirements**	**4**
5.3.3.1	**Admin Login Test**	**17**

INTODUCTION

1.1 Introduction

DOOM is a windows application for the gaming. In this project, we use C# and Unity3D.The entire project mainly consists of 3 modules, which are

- Host module
- Client module

It is a windows-based application which acts as a communication bridge between the gamers. This application maintains a centralized repository of all information related to game.

This game provides online service to the users. By using this application, the user can playgame anywhere with their friends.

Users need to download this application. They can simply open the game and play easily.

1.2. PURPOSE

User can play game from anywhere with their friends and enjoy it.

1.3. SCOPE

(1.) The entire project mainly consists of 2 modules, which are

- Host module
- User module

(2.) Host has the authority to create and delete the room.

(3.) Host Dashboard: In this section, host can start the game.

(4.) User Dashboard: In this section, user can see game settings.

1.4. FEASIBILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it’s worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully. There are 3 parts in feasibility study.

1) Operational Feasibility
2) Technical Feasibility
3) Economical Feasibility

1.4.1 OPERATIONAL FEASIBILITY

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes.To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, produceibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

1.4.2 TECHNICAL FEASIBILITY

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. The application is the fact that it has been developed on windows XP platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible .The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

1.4.2 ECONOMICAL FEASIBILTY

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

1.5 ORGANISATION OF REPORT

1.5.1 INTRODUCTION

This section includes the overall view of the project i.e. the basic problem definition and the general overview of the problem which describes the problem in layman terms. It also specifies the software used and the proposed solution strategy.

1.5.2 SOFTWARE REQUIREMENTS SPECIFICATION

This section includes the Software and hardware requirements for the smooth running of the application.

1.5.3 DESIGN & PLANNING

This section consists of the Software Development Life Cycle model.It also contains technical diagrams like the Data Flow Diagram and the Entity Relationship diagram..

1.5.4 RESULTS AND DISCUSSION

This section has screenshots of all the implementation i.e. user interface and their description.

1.5.5 SUMMARY AND CONCLUSION

This section has screenshots of all the implementation i.e. user interface and their description.

CHAPTER 2 :
SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Hardware Requirements

Table 2.1.1 Hardware Requirements

Number	Description
1	PC with 120 GB or more Hard disk.
2	PC with 2 GB RAM
3	PC with Pentium 1 and Above.

2.2 Software Requirements

Table 2.2.1 Software Requirements

Number	Description	Type
1	Operating System	Any Windows
2	Language	C#
3	Database	PUN (Photon Unity Network)
4	IDE	Visual Code
5	Editor	Unity3D

CHAPTER 3
DESIGN & PLANNING

3.1 Software Development Life Cycle Model

3.1.1 WATERFALL MODEL

The waterfall model was selected as the SDLC model due to the following reasons:

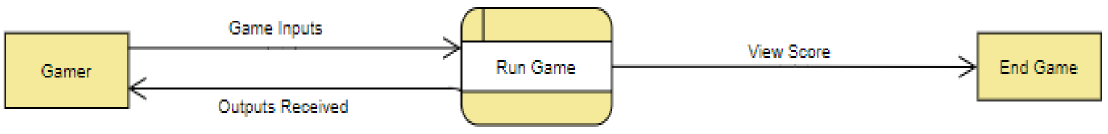
Requirements were very well documented, clear and fixed. Technology was adequately understood.Simple and easy to understand and use. There were no ambiguous requirements.
Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a reviewprocess. Clearly defined stages. Well understood milestones easy to arrange tasks.

3.2 GENERAL OVERVIEW

Health Advisor is powerful, flexible, and easy to use and is designed and developed to deliver real conceivable benefits to hospitals. More importantly it is backed by reliable and dependable support. Health Advisor is custom built to meet the specific requirement of the mid and large size hospitals acrossthe globe.It has a counter that counts calories which will be accessed by users. It contains modules like booking appointment, managing reports and medical history, patient search, etc.It contains a section named as contact us queries where you can ask for any doubts related to your appointment booking or rescheduling.

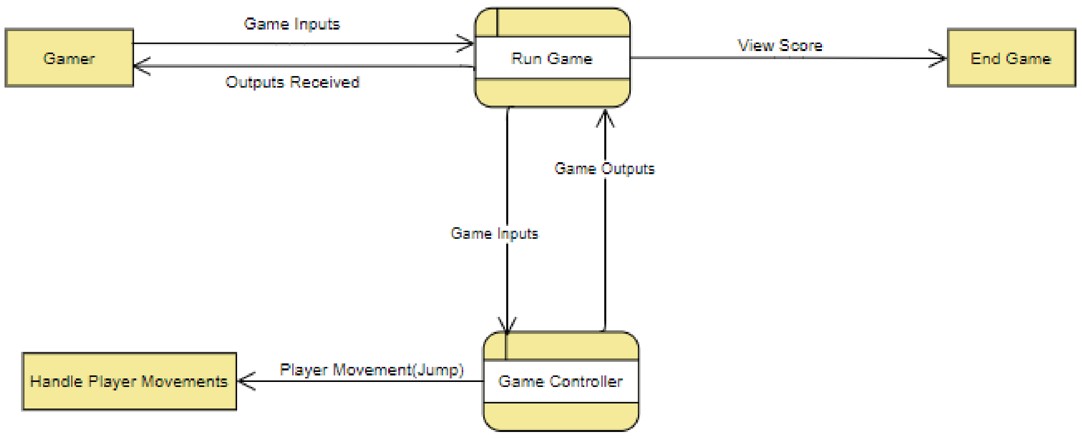
3.3 DATA FLOW DIAGRAM

3.3.1 Level 0 DFD



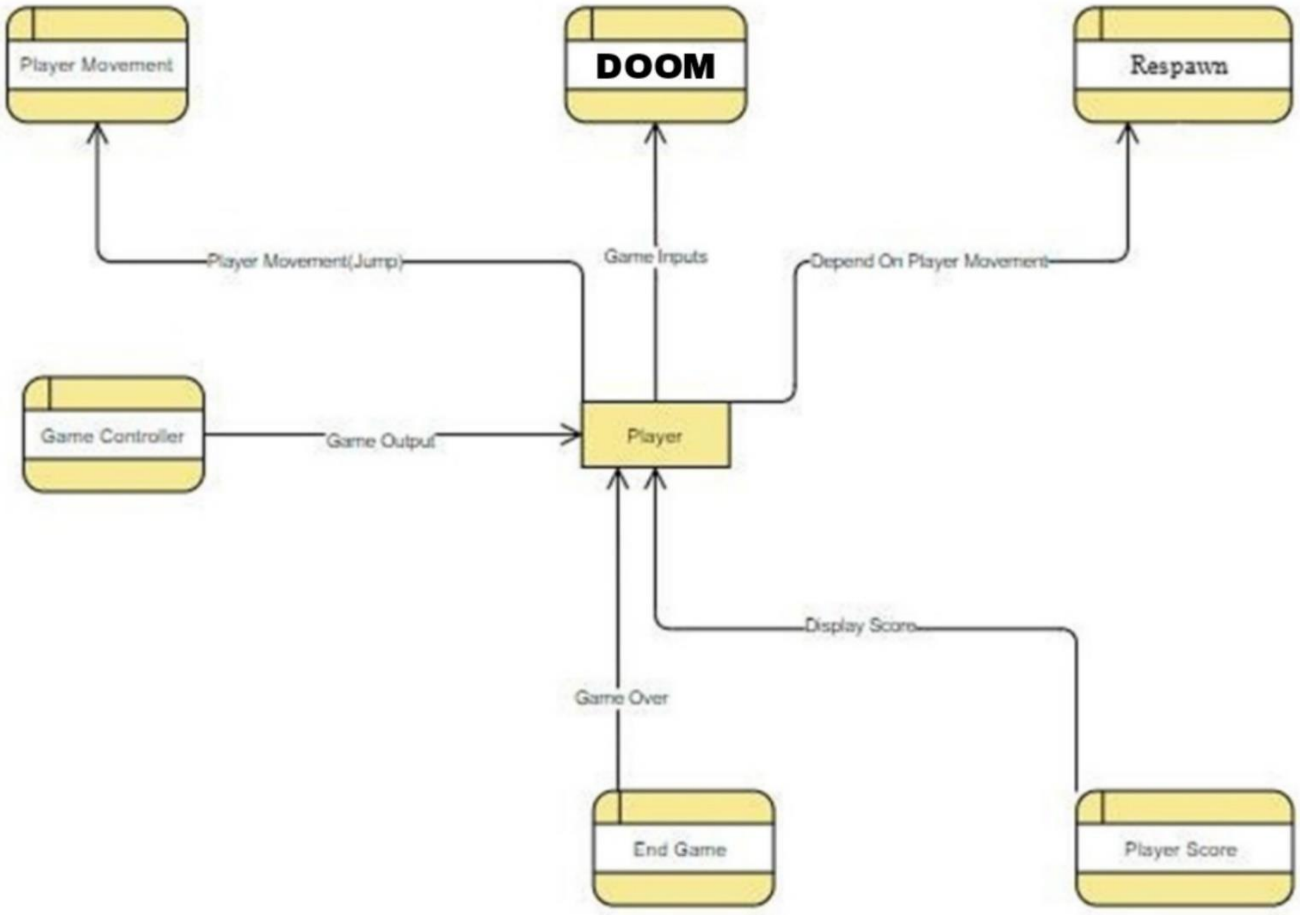
Level 0

3.3.2 Level 1 DFD



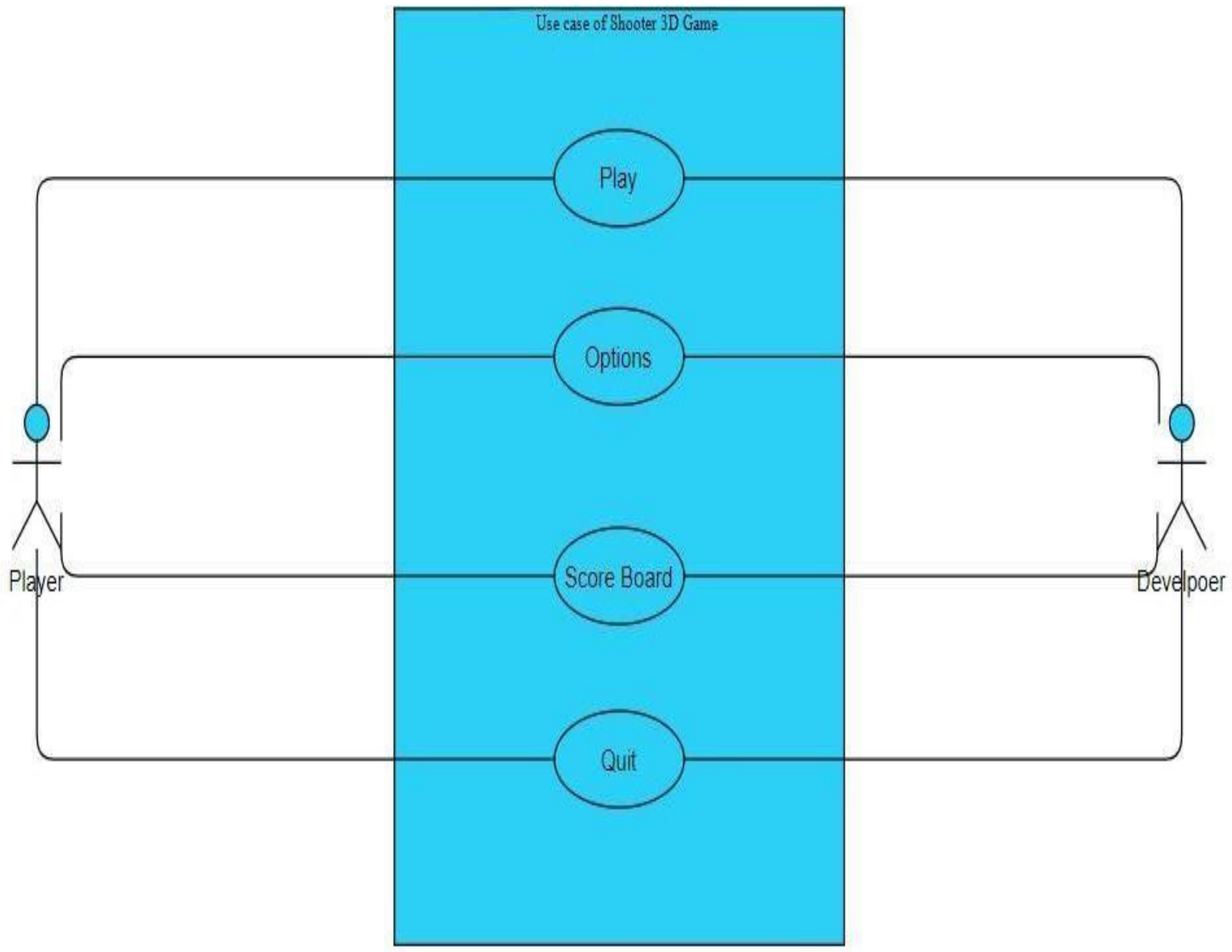
Level 1

3.3.3 Level 2 DFD

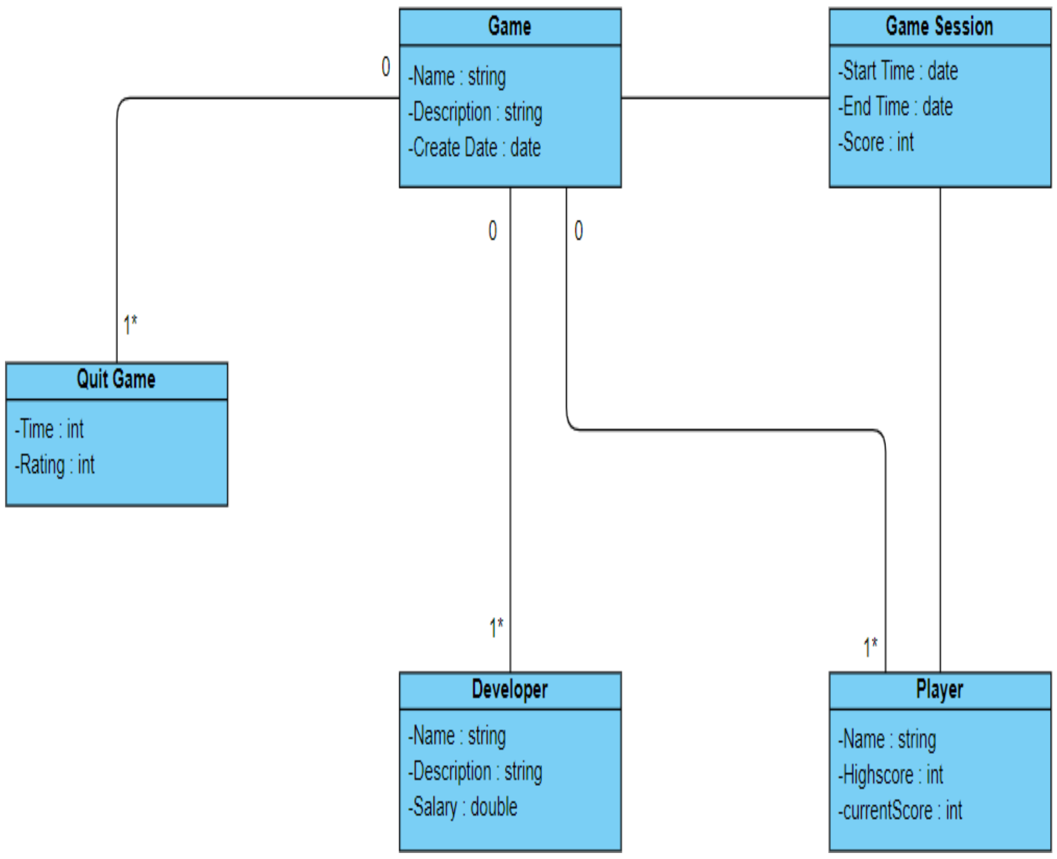


Level 2

3.4 USE CASE DIAGRAM



3.5 CLASS DIAGRAM



3.6 Input /Output Interface

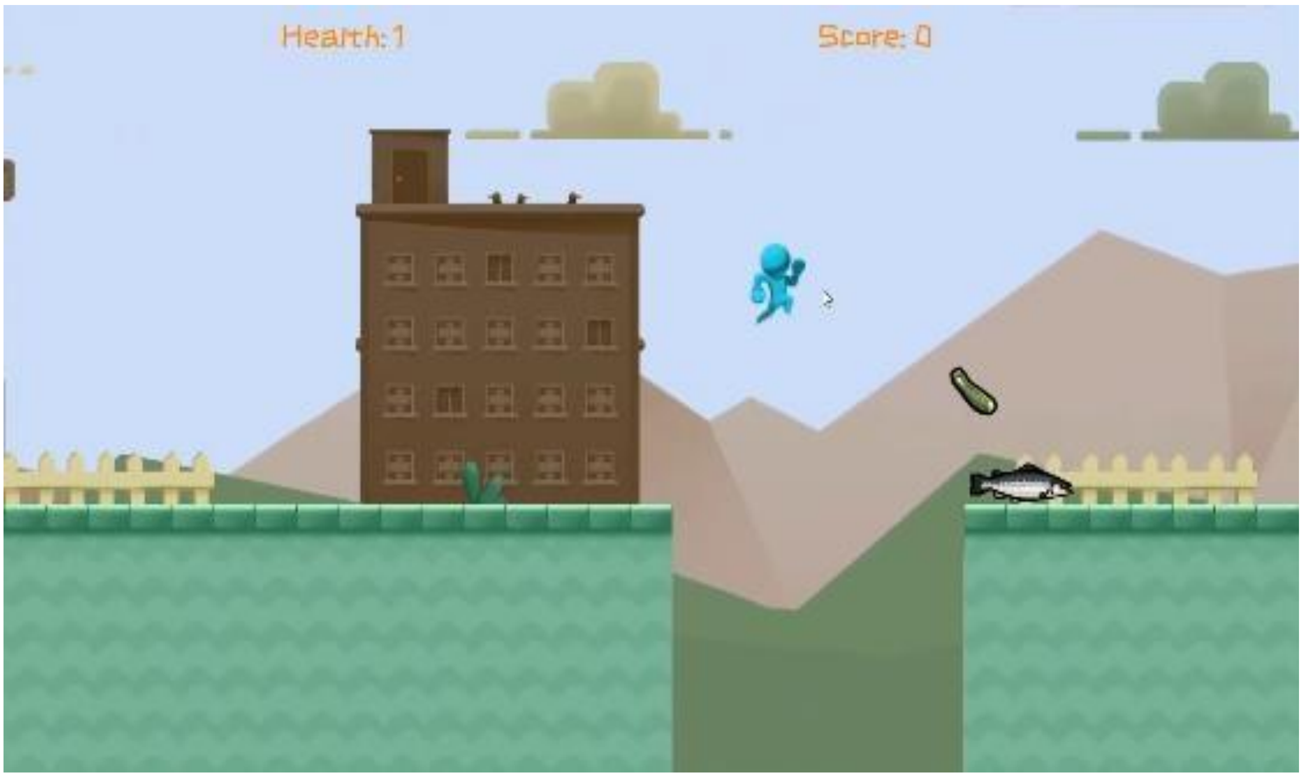


Fig.3.6.1 Interface

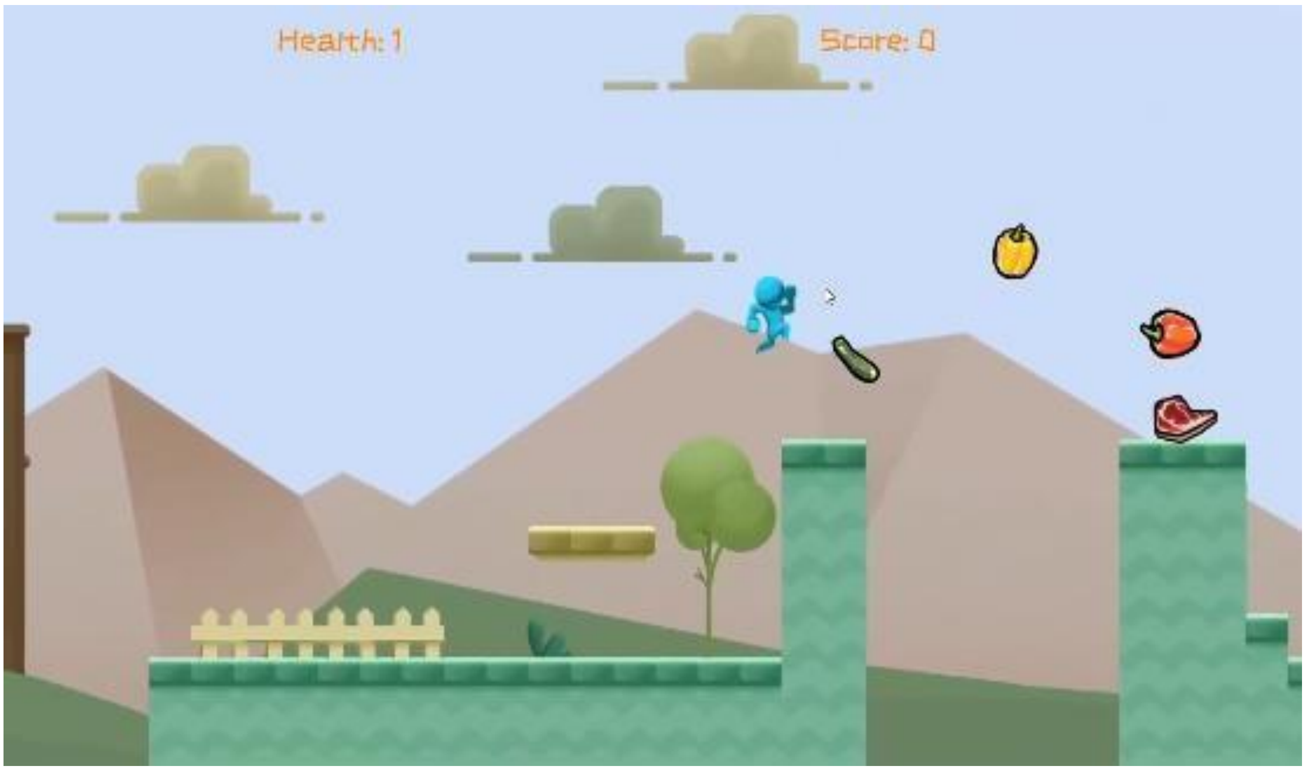


Fig.3.6.2 Main Menu

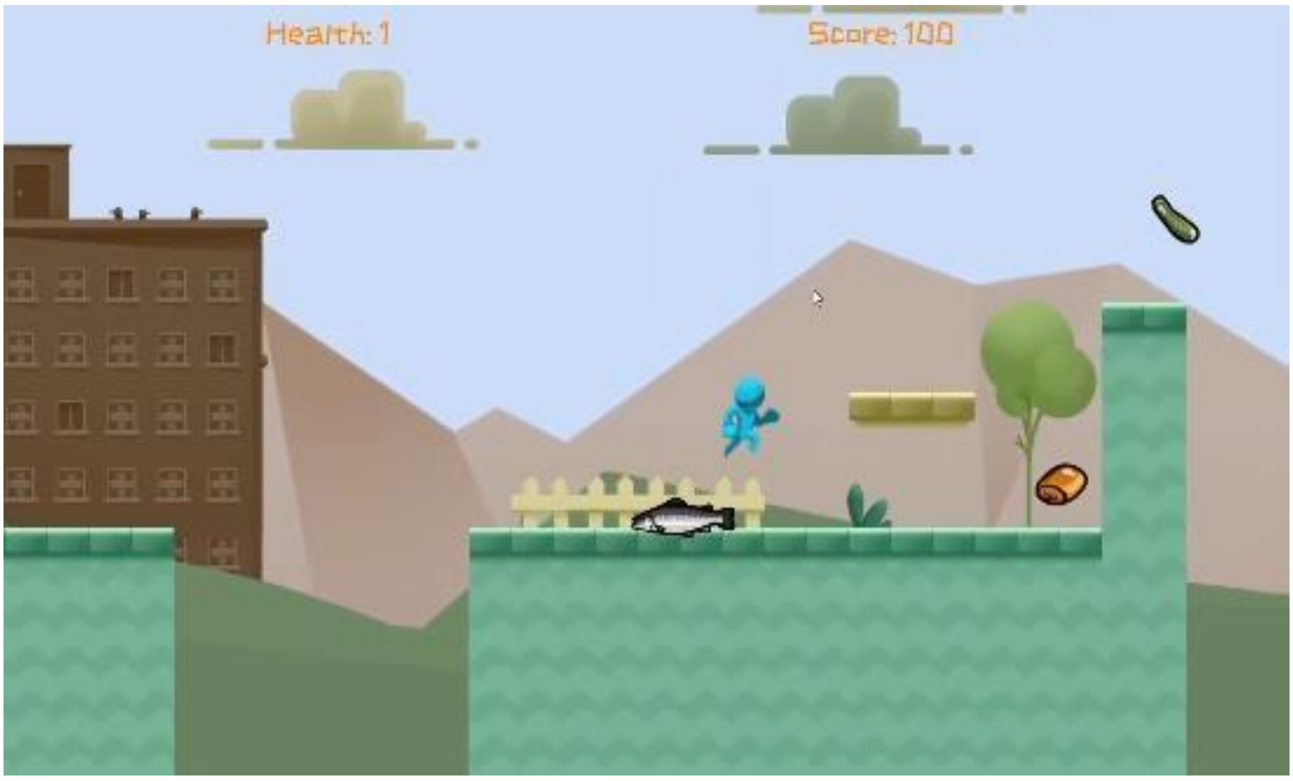


Fig.3.6.3 Enemy View

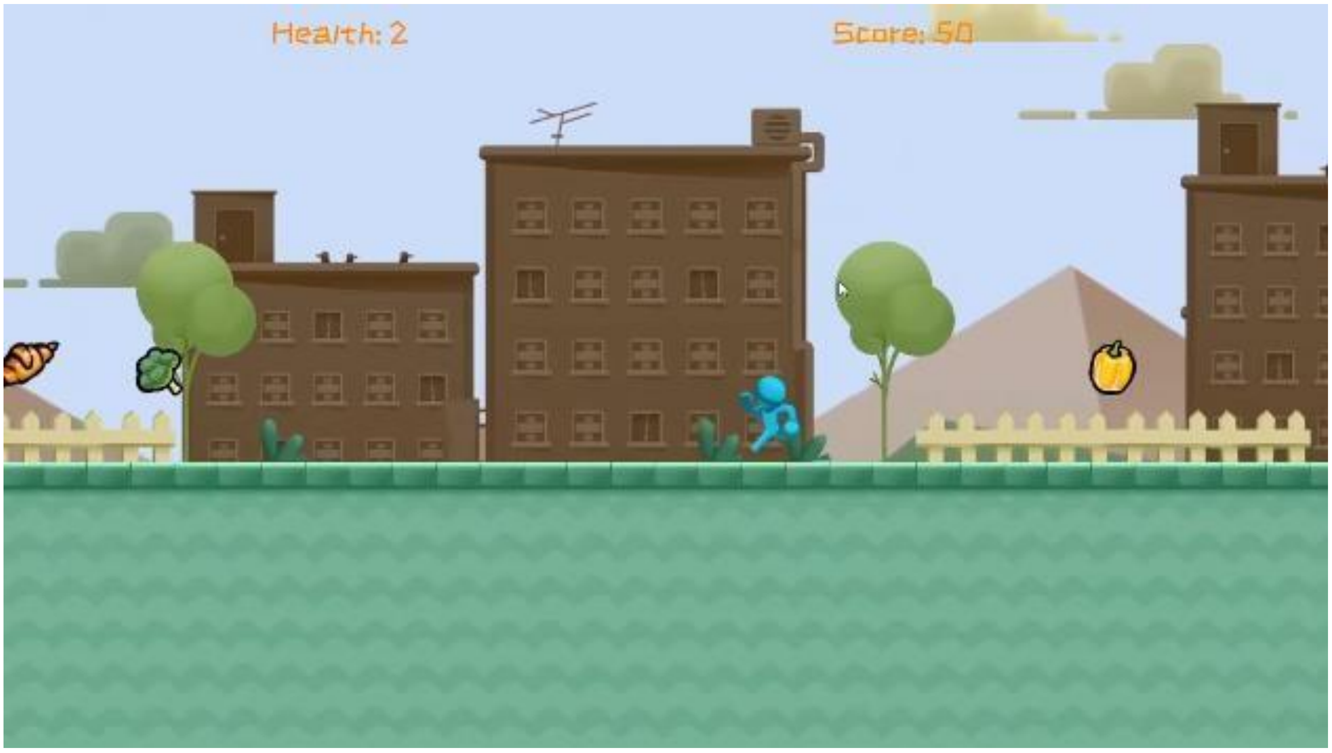


Fig.3.6.4 Game Map

CHAPTER 4

IMPLEMENTATION DETAILS

In this Section we will do Analysis of Technologies to use for implementing the project.

4.1 FRONT END

4.1.1 Unity 3d:

- Unity is, simply put, the world's most popular game engine. It packs a ton of features together and is flexible enough to make almost any game you can imagine.
- With unrivalled cross-platform features, Unity is popular with both hobby developers and AAA studios. It's been used to create games like Pokémon Go, Heath stone, Rim world, Cuphead, and plenty more.
- While3Dis inthename, Unity3D also packs toolsfor2D game development.
- Programmers love It because of the C# scripting API and built-in Visual Studio integration. Unity also offers JavaScript as a scripting language and Mono Develop as an IDE to those who want an alternative to Visual Studio.
- But artists love it as well since it comes with **powerful animation tools** that make it simple to create your own 3D cut scenes or build 2D animations from scratch. Nearly anything can be animated in Unity.

4.2 BACK END

4.2.1 C#:

- C# is a strongly typed object-oriented programming language. C# is open source, simple, modern, flexible, and versatile. In this article, let's learn what C# is, what C# can do, and how C# is different than C++ and other programming languages.
- A programming language on computer science is a language that isused to write software programs.
- C# is a programming language developed and launched by Microsoft in 2001. C# is a simple, modern, and object-oriented language that provides modern day developers flexibility and features to build software that will not only work today but will be applicable for years in the future.

CHAPTER 5

TESTING AND IMPLEMENTATION

The term implementation has different meanings ranging from the conversation of a basic application to a complete replacement of a computer system. The procedures, however, are virtually the same. Implementation includes all those activities that take place to convert from old system to new. The new system may be totally new replacing an existing manual or automated system or it may be major modification to an existing system. The method of implementation and time scale to be adopted is found out initially. Proper implementation is essential to provide a reliable system to meet organization requirement.

5.1 UNIT TESTING

5.1.1 Introduction

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object- oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testersduring the development process. It forms the basis for component testing. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

5.1.2 Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, itaffords several benefits.

1) Find problems early: Unit testing finds problems early in the development cycle. In test-driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

2) Facilitates Change: Unit testing allows the programmer to refactor code or upgrade system libraries later, and make sure the module still works correctly (e.g., in regression testing). The procedureis to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

3) Simplifies Integration: Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

4) Documentation: Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API).Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

5.2 INTEGRATION TESTING

Integration testing (sometimes called integration and testing, abbreviated I&T) is thephase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan tothose aggregates, and delivers as its output the integrated system ready for system testing.

5.2.1 Purpose

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed totest whether all the components within assemblages interact correctly, for example across procedure callsor process activations, and this is done after testing individual modules, i.e., unit testing. The overall ideais a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross- dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatic complexity of the software and software architecture, reusability of modules and life-cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns [2] are: collaboration integration, backbone integration, layer integration, client- server integration, distributed services integration and high-frequency integration.

5.2.1.1 Big Bang

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing. A type of big-bang integration testing is called "usage model testing" which can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use. Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment. For integration testing, Usage Model testing can bemore efficient and provides better test coverage than traditional focused functional integration testing. Tobe more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

5.2.1.2 Top-down and Bottom-up

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level components. The process is repeated until the component at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpfulonly when all or most of the modules of the same development level are ready. This method also helps todetermine the levels of software developed and makes it easier to report testing progress in the form of a percentage. Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module. Sandwich testing is an approach to combine top-down testing with bottom-up testing.

5.3 SOFTWARE VERIFICATION AND VALIDATION

5.3.1 Introduction

In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

Validation: Are we building the right product?
Verification: Are we building the product, right?
According to the Capability Maturity Model (CMMI-SW v1.1)

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that"you built it right". Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfill its intended use.

From Testing Perspective

Fault – wrong or missing function in the code.
Failure – the manifestation of a fault during execution.
Malfunction – according to its specification the system does not meet its specified functionality
Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required. Within the modeling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

M&S Verification is the process of determining that a • computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.
M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).

5.3.2 Classification of Methods

In mission-critical software systems, where flawless performance is absolutely necessary, formal methods may be used to ensure the correct operation of a system. However, often for non-mission- critical software systems, formal methods prove to be very costly, and an alternative method of softwareV&V must be sought out. In such cases, syntactic methods are often used.

5.3.3 Test Cases

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Othermethods, such as reviews, may be used early in the life cycle to provide for software validation.

5.3.3.1 Test Suit

Admin login test:

Test Case	Test Data	Test Result	Test Report
Blank Username	Username	Invalid	Fill required detail
Invalid Username	Username: ADMIN	Invalid	Username Incorrect
Invalid Password	Password: user	Invalid	Password Incorrect
Valid Username and Password	Username: admin Password: admin	Valid	Login

Table 5.3.3.1.1 Admin Login Test

5.4 Black-Box Testing

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher-level testing but can also dominate unit testing as well.

5.4.1 Test Procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

5.4.2 Test Cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

5.5 White-Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester choosesinputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT). White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

5.5.1 Levels

1) Unit testing: White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.

2) Integration testing: White box testing at this level is written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behavior in an open environment through the use of white box testing for any interactions of interfaces that are known to the programmer.

3) Regression testing: White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

5.5.2 Procedures

White-box testing's basic procedures involves the tester having a deep level of understanding of the source code being tested. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analyzed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white box testing to layout all of the basic information.
Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
Output involves preparing final report that encompasses all of the above preparations and results.

5.5.3 Advantages

White-box testing is one of the two biggest testing methodologies used today. It has several major advantages:

Side effects of having the knowledge of the source code are beneficial to thorough testing. Optimization f code by revealing hidden errors and being able to remove these possible defects. Gives the programmer introspection because developers carefully describe any new implementation. Provides traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.
White box testing gives clear, engineering-based, rules for when to stop testing.

5.5.4 Disadvantages

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:

White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high level of knowledge due to the complexity of the level of testing that needs to be done.
On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.
The tests focus on the software as it exists, and missing functionality may not be discovered.

5.6 SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all of the "integrated" softwarecomponents that have passed integration testing and also the software system itself integrated withany applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system asa whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

CHAPTER 6

LIMITATIONS

Though we tried our best in developing this game but as limitations are more parts of any game so are of our game. Some limitations of DOOM are: -

- This game is in developing phase so there are limited levels.
- Avatar is not available, we will make sure to give you some amazing customize avatar.
- We will add powers so the player can shoot the enemy

CHAPTER 7

CONCLUSION

DOOM is a adventure game, which is created for fun enjoy by passing unique levels. To play this game you just have to download it and play it.

REFERENCE

Assets of the game are taken from Unity assets store:

- https://www.youtube.com/watch?v=ME8mHCvRymA&t=535s
- https://www.youtube.com/watch?v=IvGfss4fWEY