

INTRODUCTION

Developing software applications is a time consuming process. During the last years, several software development methodologies, often known as ***agile software development***, have become widely used by software developers to address this issue. Many different development methodologies can be more or less good, depending of the task and application type.

One of the software development methodologies is the *evolutionary software method*, which as the name hints, takes on an evolutionary approach to the problem, and allows the project to evolve through different stages of the project. Our case study will show how well this evolutionary approach worked on our project where we choose to develop a **2D Graphic** computer game. Some requirements for the computer game were given from the beginning, such as:

2D Graphics - 2D computer graphics is the computer-based generation of digital images mostly from two dimensional models (such as 2D geometric models, text, and digital images) and by techniques specific to them. The word may stand for the branch of computer science that comprises such techniques or for the models themselves.

Impressive result – The game result must impress whoever plays the game. It should last long, and make the players come back and play it over and over again.

Graphical effects – To achieve an impressive result, we would need to add modern graphical effects, such as real-time rendered soft shadows, motion blur, and ambient occlusion.

Working with these requirements, we decide to use **UNITY 3D** as our platform to develop our 2D game with.

SCOPE OF PROJECT

The scope of the project is to develop a 2-dimensional Computer game. The system shall use the Key Event library under the Unity Engine to detect when the keys are been pressed on the keyboard to control the animated character in the game (HEN). The project will be based on creating an adventure game with the goal in mind of being fun. Listed below are the scopes that I will be covering in the

Development of this game

I. Single Player

II. PC based

III. 2-dimensional platform

IV. Triple level

V. Adventure-based

VI 2D platform for GUI and menu systems

VII Testing for 0 bugs in game

VIII. Written in C# using the Unity 3d.

Technology used

- **Unity 3d**
- **C#**

The report is aimed at people who are beginning with learning Unity and possess at least a basic knowledge of the Unity 3D game engine. It does not discuss the whole process of creating the game.

The project is not developed with ways of generating revenue. Using the game for recreating and learning purposes is more suitable for it.

Unity 3d

Unity is a cross-platform real-time engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as an OS X-exclusive game engine. As of 2018, the engine has been extended to support 27 platforms. The engine can be used to create both three-dimensional and two-dimensional games as well as simulations for its many platforms.

#Overview

Unity gives users the ability to create games and interactive experiences in both 2D and 3D. The engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality. Prior to C# being the primary programming language used for the engine, it previously supported Boo, which was removed in the Unity 5 release, and a version of JavaScript called Unity Script, which was deprecated in August 2017 after the release of Unity 2017.1 in favor of C#.

#Scripting language

C#

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language –

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

SOFTWARE DEVELOPMENT METHODOLOGY

The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines is called *software engineering*.

Software engineering is the discipline whose aim is:

1. Production of quality software
2. Software that is delivered on time
3. Cost within the budget
4. Satisfies all requirements.

Software process is the way in which we produce the software. Apart from hiring smart, knowledgeable engineers and buying the latest development tools, effective software development process is also needed, so that engineers can systematically use the best technical and managerial practices to successfully complete their projects.

A **software life cycle** is the series of identifiable stages that a software product undergoes during its lifetime .A software lifecycle model is a descriptive and diagrammatic representation of the software life cycle .A life cycle model represents all the activities required to make a software product transit through its lifecycle phases .It also captures the order in which these activities are to be taken.

Life Cycle Models

There are various life cycle models to improve the software processes.

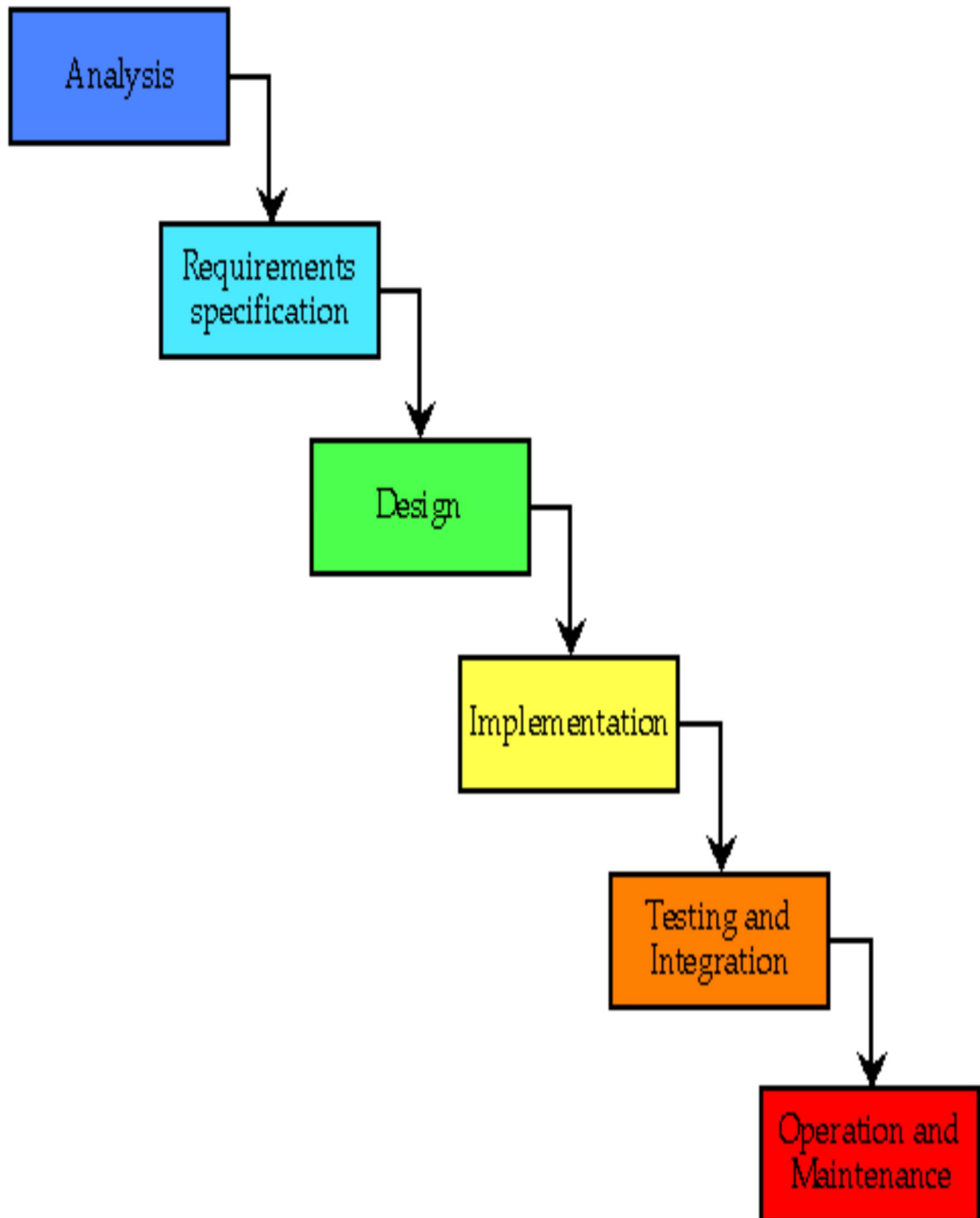
- WATERFALL MODEL
- PROTOTYPE MODEL
- ITERATIVE ENHANCEMENT MODEL
- EVOLUTIONARY MODEL
- SPIRAL MODEL

In the project, **Waterfall model** is followed.

WATERFALL MODEL

The **Waterfall Model** was first Process **Model** to be introduced. It is very simple to understand and use. In a **Waterfall model**, each phase must be completed before the next phase can begin and there is no overlapping in the phases. **Waterfall model** is the earliest SDLC approach that was used for software development.

In “The Waterfall” approach, the whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.



WATERFALL MODEL

This model contains 6 phases:

O Feasibility study

The feasibility study activity involves the analysis of the problem and collection of the relevant information relating to the product. The main aim of the feasibility study is to determine whether it would be financially and technically feasible to develop the product.

O Requirement analysis and specification

The goal of this phase is to understand the exact requirements of the customer and to document them properly (SRS).

O Design

The goal of this phase is to transform the requirement specification into a structure that is suitable for implementation in some programming language.

O Implementation and unit testing

During this phase the design is implemented. Initially small modules are tested in isolation from rest of the software product.

O Integration and system testing

In this all the modules are integrated and then tested altogether.

O Operation and maintenance

Release of software inaugurates the operation and life c.

SRS

#INTRODUCTION

This covers the requirements specification of our game "Climber", It includes the specification of this documentation with general description, specific requirements, and analysis of models. It also includes changes management of this requirement specification in case of any change.

In this section the documentation of this report is specified. It specifies the document convention, document scope and also provides a suggestion for the readers of the document.

#Purpose of this Chapter

This Software Requirements Specification (SRS) part is intended to give a complete overview of our Project the game "Ghost in the Town" including the action flow, initial user interface and story therein. The SRS document details all features upon which we have currently decided with reference to the manner and importance of their implementation.

#Document Conventions

This document will freely interchange the pronoun "we" with the team's acronym. As the development team is responsible for the SRS document, no ambiguity arises from its usage.

There is a clear distinction, however, between the use of the words "player/gamer" and "character" The "player" is the human being interacting with the game in the real world, while the "character" is the in-game player avatar being manipulated.

#Intended Audience and Reading Suggestions

The SRS document also gives project managers a way to ensure the game's adherence to our original vision.

Although the document may be read from front to back for a complete understanding of the project, it was written in sections and hence can be read as such. For an overview of the document and the project itself, see Overall Description.

For a detailed description of the game play elements and their interaction with the character, read System Features. Readers interested in the game-play interface and navigation between different front-end menus should go through External Interface Requirements.

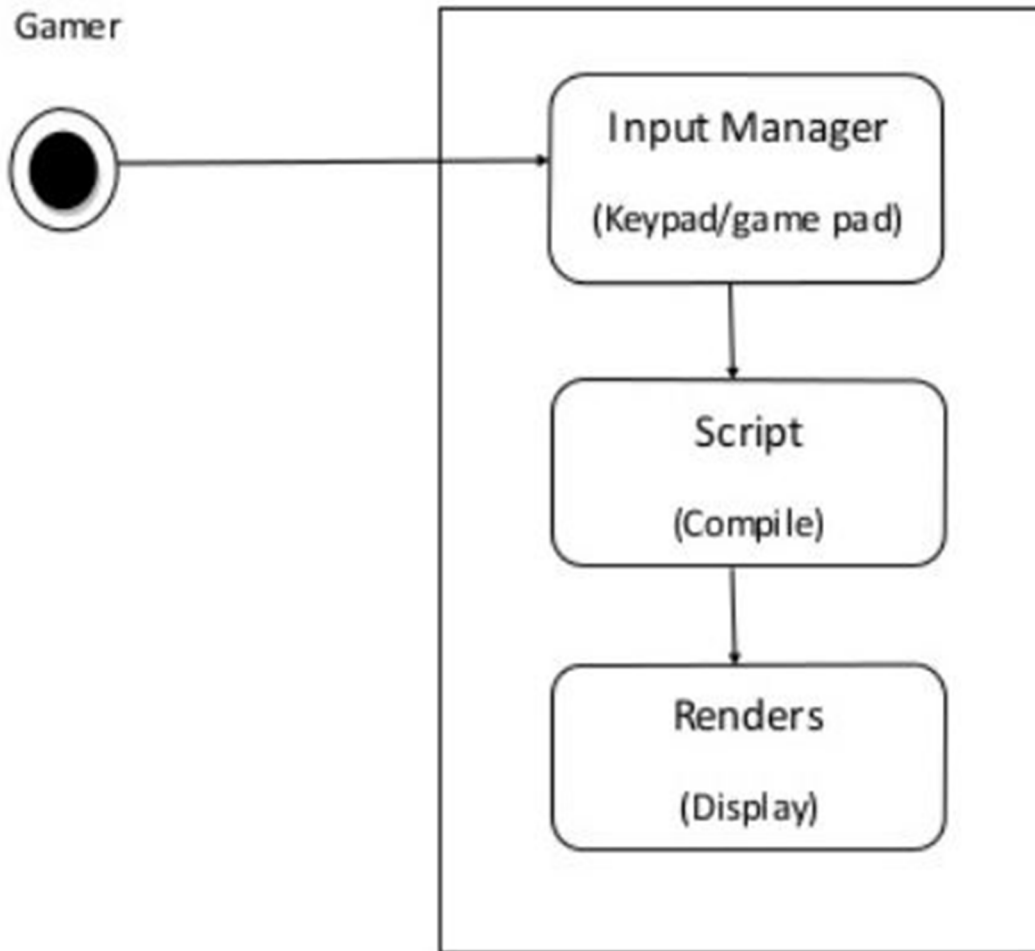
Technical standards to which the team will hold the project are laid out in Other Nonfunctional Requirements. The development schedule, meanwhile, will be maintained in the Key Milestones.

#Scope of the Document

The Software Requirements Specification (SRS) describes the functional and nonfunctional requirement for the project. As we said before the purpose of this project is to provide virtual image for the combination of both structured and unstructured information of our project 'Climber'.

Project 'Climber' was conceived by 4 of our team members as having an anticipated development cycle greater than the length of the semester. The team wishes to carry on the project until its completion. The game will continue to grow until we feel it satisfactory for open source distribution.

#System Environment



Gamer can interact with system by giving input (press key to start game) to the system. System gives those input to script, if any change occur (if the value is changed) this object send to renders to display the things (a character can change its place).

#Quality Function Deployment of "Climber"

Quality Function Deployment is a technique that translates the needs of the customer into technical requirements for software or game; it concentrates on maximizing customer satisfaction from the Software engineering process .With respect to our project the following requirements are identified by an OFD.

- Normal Requirements
- Expected Requirements
- Exciting requirements

Specific Requirements

This section covers the project external requirements of our game and also indicates the user characteristics for this project.

External Interface Requirements of the Game

User Interfaces

Every game must have a menu so it can be user friendly enough and gamers can easily fulfill their need. Menu is also an important thing while creating the SRS document section.

Hardware Interfaces

"Climber" is a computer gaming application designed specifically for the PC platform and is functional on all windows. Gaming application data is stored locally on the game engine elements. "Climber" has been developed for Windows and all subsequent releases.

Graphics library and a 3 dimensional graphics library based on OpenGL ES 2.0 specifications as well as hardware orientation, scaling, pixel format conversion and accelerated 3D graphics.

User Characteristics for the System

There is only one user at a time in this software and the user interacts with the game (system) in different manner. So, Gamer is the only one who communicates with the system through playing the game. And this gamer can be any person. The primary requirement is that, the gamer must read the playing procedure provided by us (developers).

Analysis Model of Our Game Project

This section describes the Software Requirements Specification of our project by analyzing the proper models of requirement engineering.

Scenario Based Model

This Model depicts how the user interacts with the system and the specific sequence of activities that occur as the software is used.

Data Model

If software requirements include the need to create, extend or interface with database or if complex data structures must be constructed and manipulated, the software team may choose to create a data model as part of overall requirements modeling. Although our game has many data objects, it does not have any data storage. All the objects and their related data are handled by the game engine. So the developers need not think about data storage. For this reason, data model is redundant for this game project.

Behavioral Model

The Behavioral indicates how software will respond to external events or stimuli. There are two ways to show these responses. One is state diagram and the other is sequence. Usually state diagram can be made in two ways, one is creating a state diagram for each class and the other is to create a state diagram for the whole system. As we don't have any class, for this is not an object oriented game, we have followed the later one. We used the modules of the use case scenario to create the state diagram and to lessen complexity we have divided the state diagram into two diagrams. On the other hand, for the sequence diagram, we have created separate a sequence diagram for all the use cases when necessary.

Use case diagram



Climber

System

Level 0



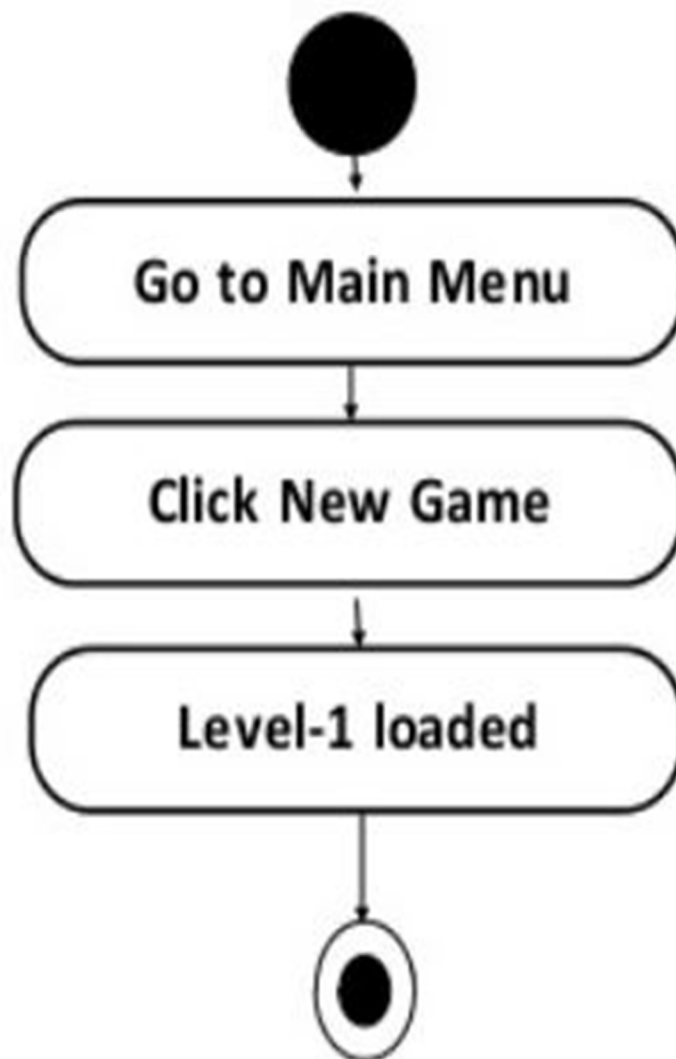
New game

Score

Exit

Action

Activity Diagram



Requirement Change Management of Our System

The developers intend to release a complete and fully functional game that follows the guidelines mentioned in the SRS document. However, since the product will be released for multiple Android platforms i.e. the different phones running the Android system), updates will likely be required. These updates would consist of any bug fixes that are necessary, compatibility patches for all of the current phones that support the Android System, and expansions of the content. If the players find any issues or has any comments they would be able to contact the developers through the official support email address which is

Lakhitolani@gmail.com

Bugs and Glitches

The players would be able to contact the developers through the support email system. This is where they would present any bugs or glitches they have detected and if they have any beliefs that the game is not functioning properly. General concerns or comments would also need to be submitted here.



Game Overview

“CLIMBER”

‘Climber’ is the name of our game which is based on the cross level games. In this project we have used Unity 3d platform for development and c# as a scripting language.

In this game there are various levels which are developed in order of difficulties for the player. After crossing the level, Player is allowed to play the next level.

Description of Game:-

This is the single player game in which player have to start from the first level and player has his image in the game in form of a HEN .Player has to grab the egg’s embedded in the screen, the points are given on taking each egg, The compulsory part of game is that player has to take final trophy to open the gate which is the door of next difficult level.

We have used brick’s, water as an obstacle for the player to increase the difficulty level.

No of levels: - 3

Some images.....

Hen-



Egg-



Trophy-

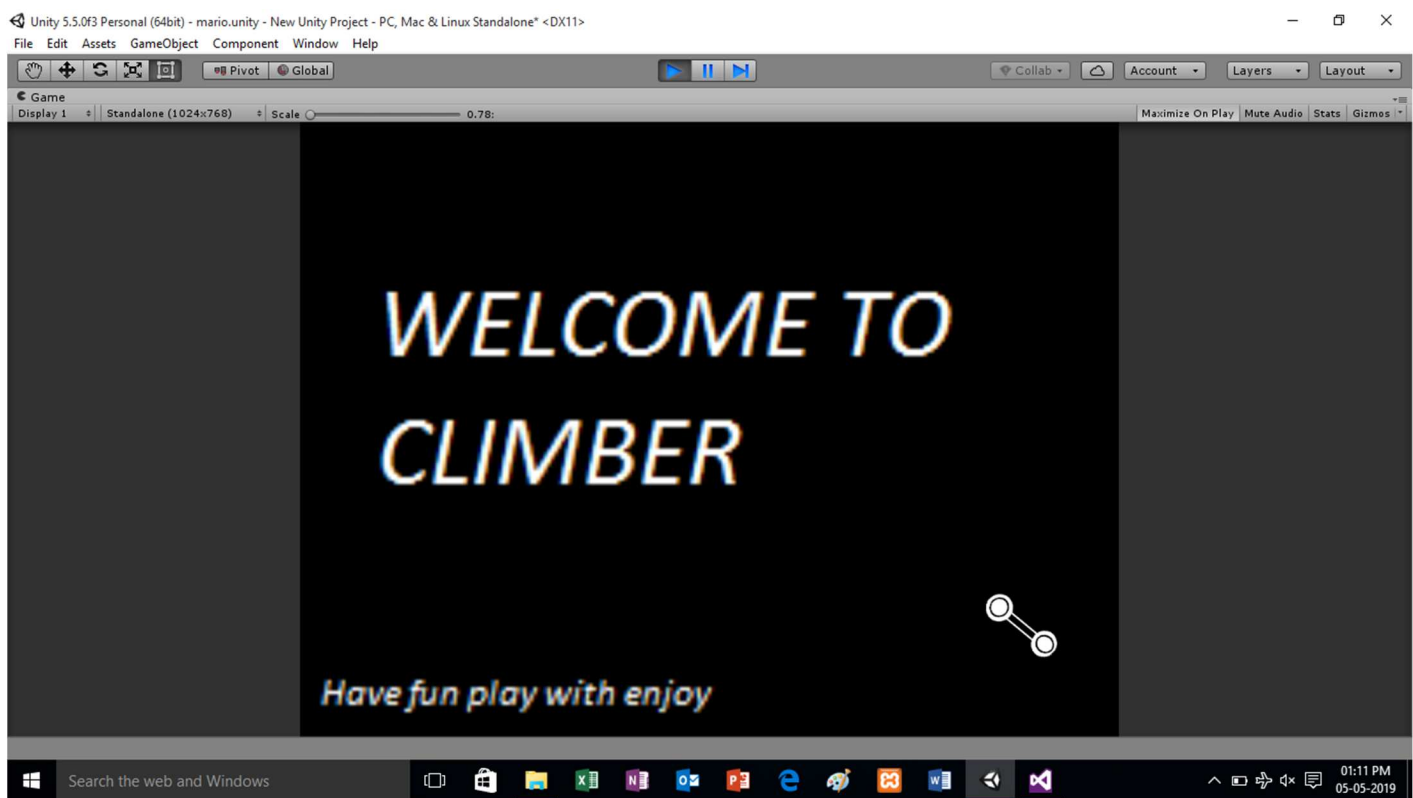


Gate-




Graphics and Visuals of the game

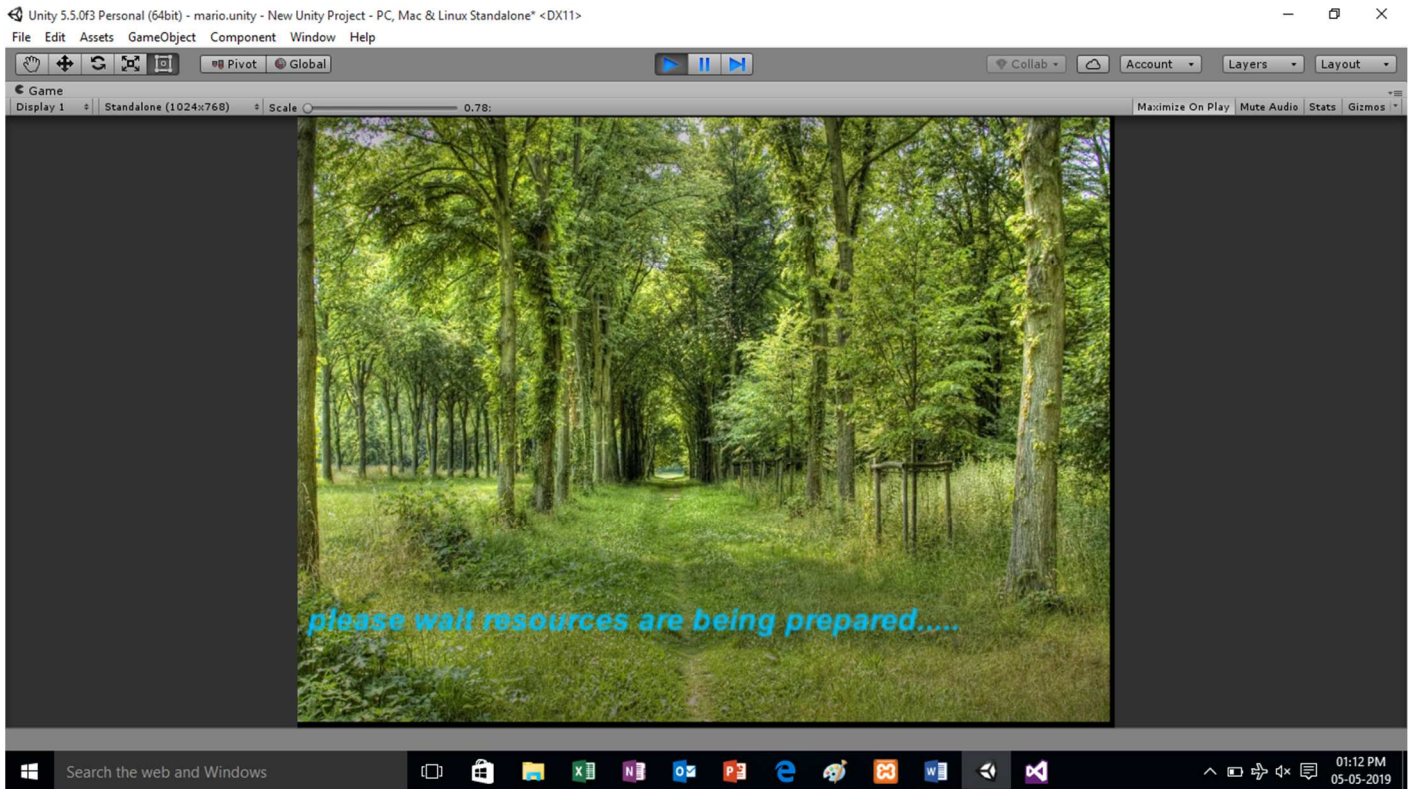
1. Welcome window



This welcome window welcomes the player to the developed game which is based on, Hen hunting for the eggs with our motto **“Have fun play with enjoy”**.

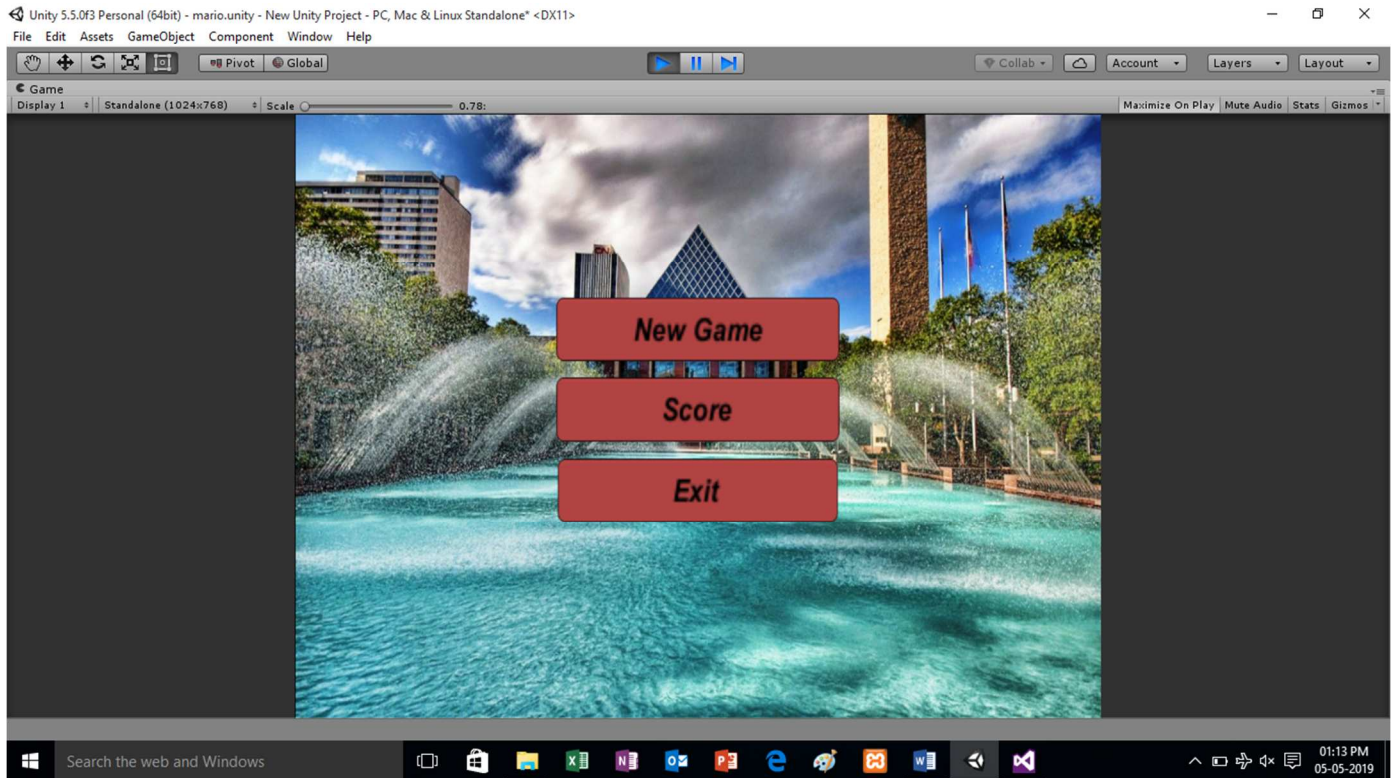
Here  this sign show that game is loading.

2.Loading window



This window gives the message to the user that u have to wait till the resources are being prepared for the proper execution of the developed game.

3.Start window



This screen allows player to choose one option among three.

1.New Game

By clicking on this button player enters to the level 1 of the developed game.

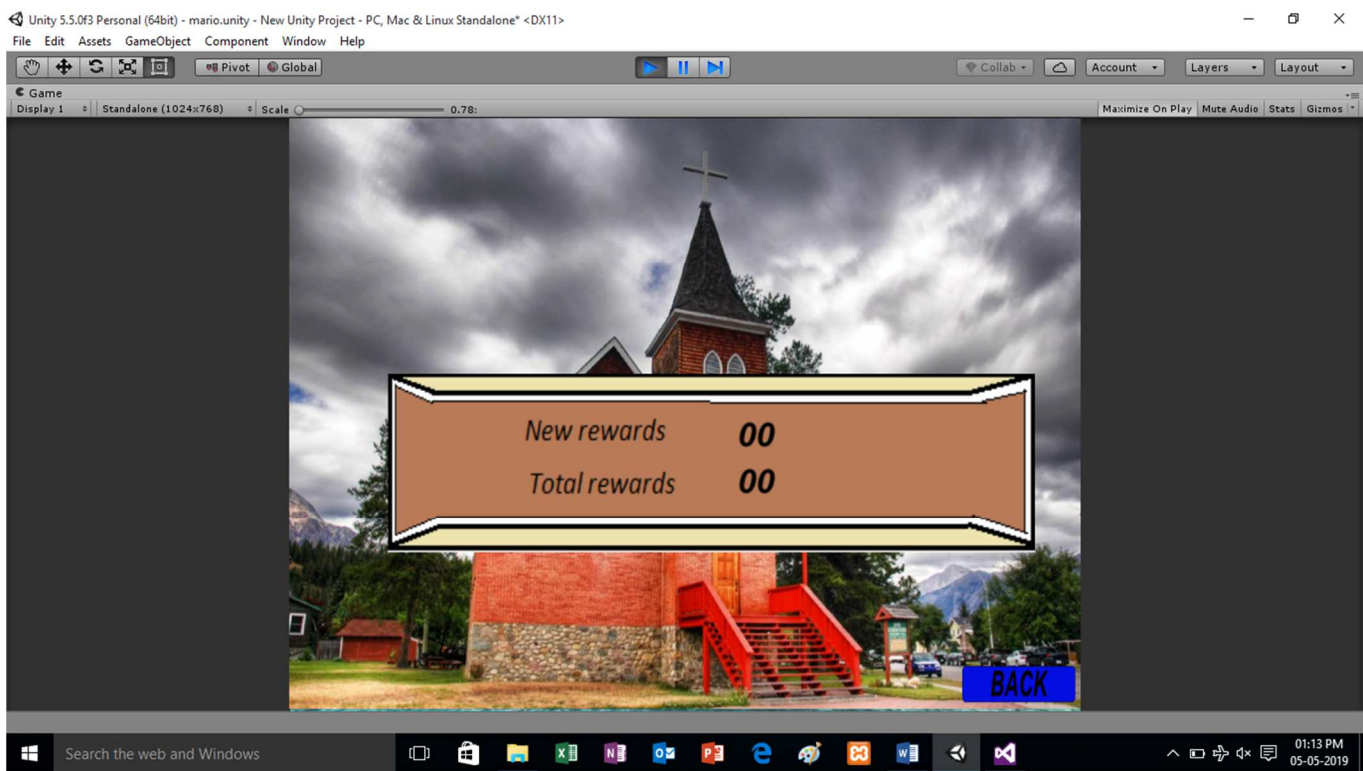
2.Score

By clicking this button player gets the scored no which scored previously.

3.Exit

By pressing this button player quits the game and the running application is terminated.

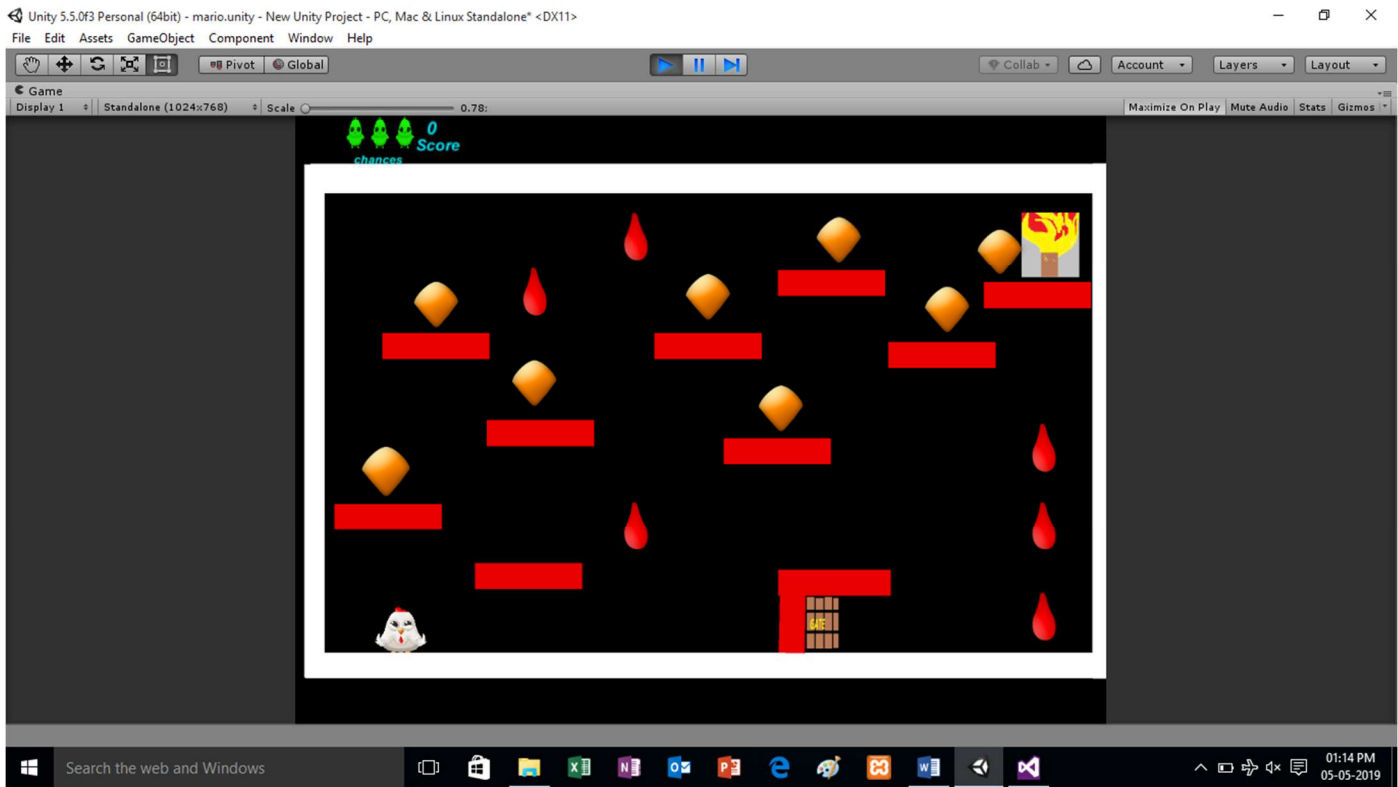
4. Score window



This screen shows the New score and the Total score (New + Previous), scored by the player.

In, addition to that there is also a back button which takes player back to the Start Menu.

5. Level 1

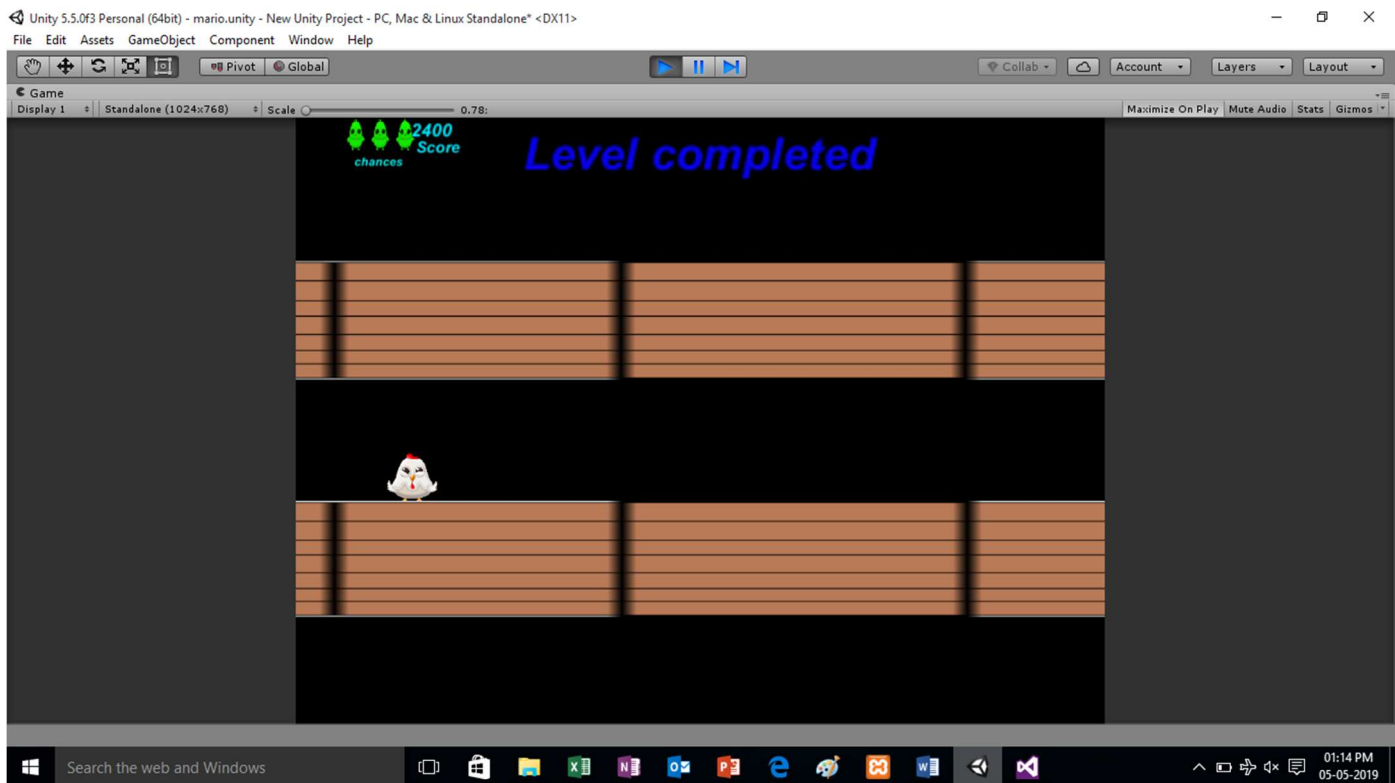


This window shows the level 1 of the developed game, in this level Hen (player) has to take eggs, on getting each egg, points are given to the player.

Finally player has to take trophy to open the gate which takes player to the next level.

Red color blocks are the bricks which act as obstacles for the player to increase difficulty.

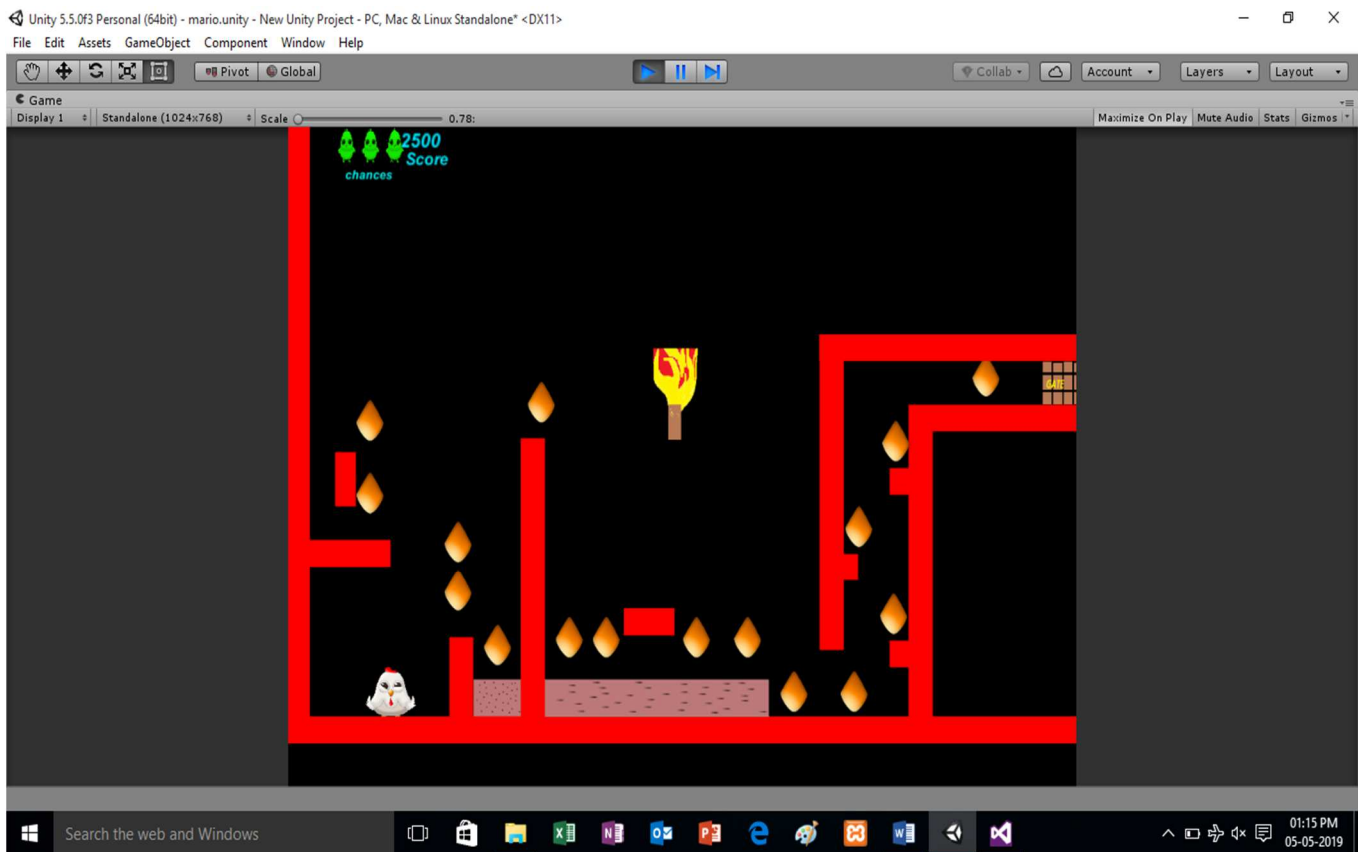
6. Level completion window



When player has successfully completed the level this window appears which depicts the message “Level completed” and takes to the next window with floating hen.

Hen floats from left to right.

7. Level 2

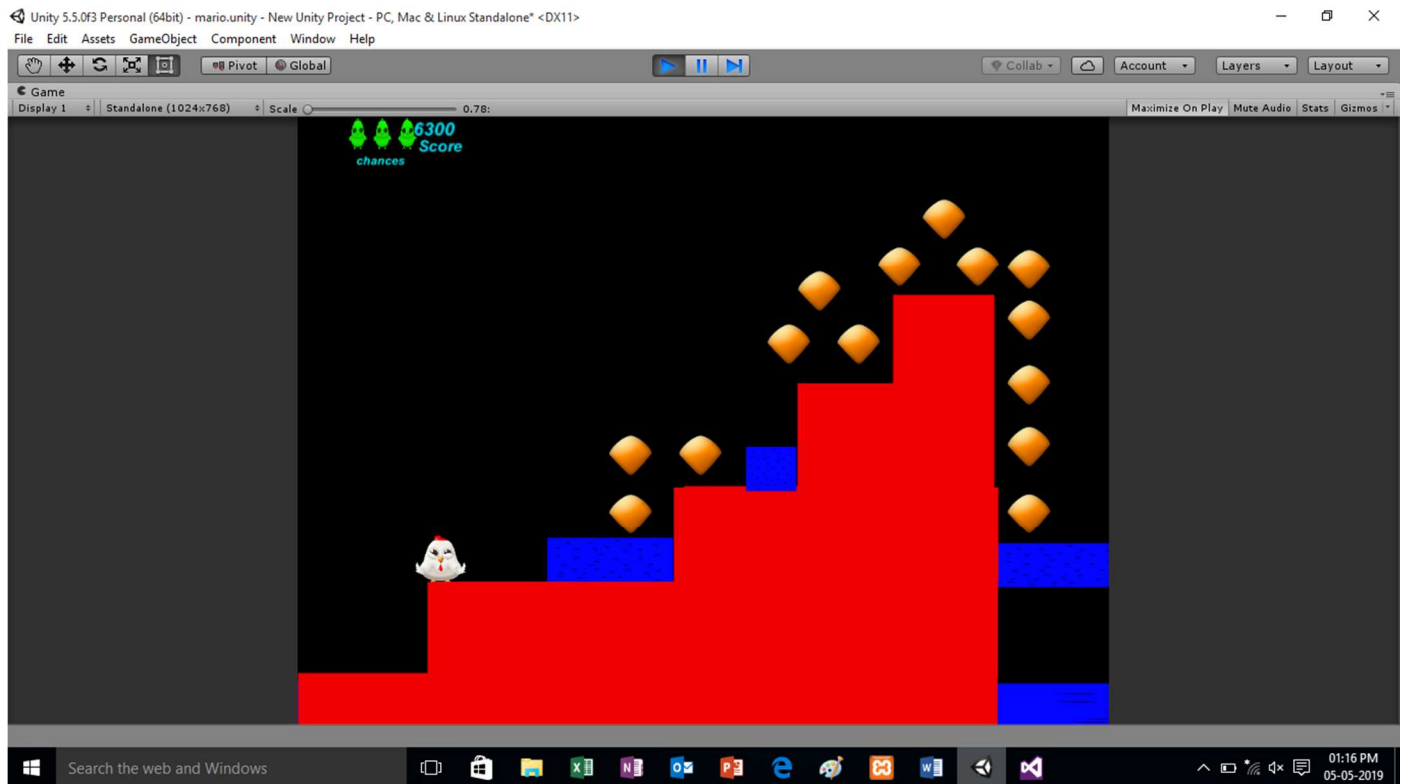


This window shows the level 2 of the developed game, in this level Hen (player) has to take eggs, on getting each egg, points are given to the player.

Finally player has to take trophy to open the gate which takes player to the next level.

Red color blocks are the bricks which act as obstacles for the player to increase difficulty.

8. Level 3



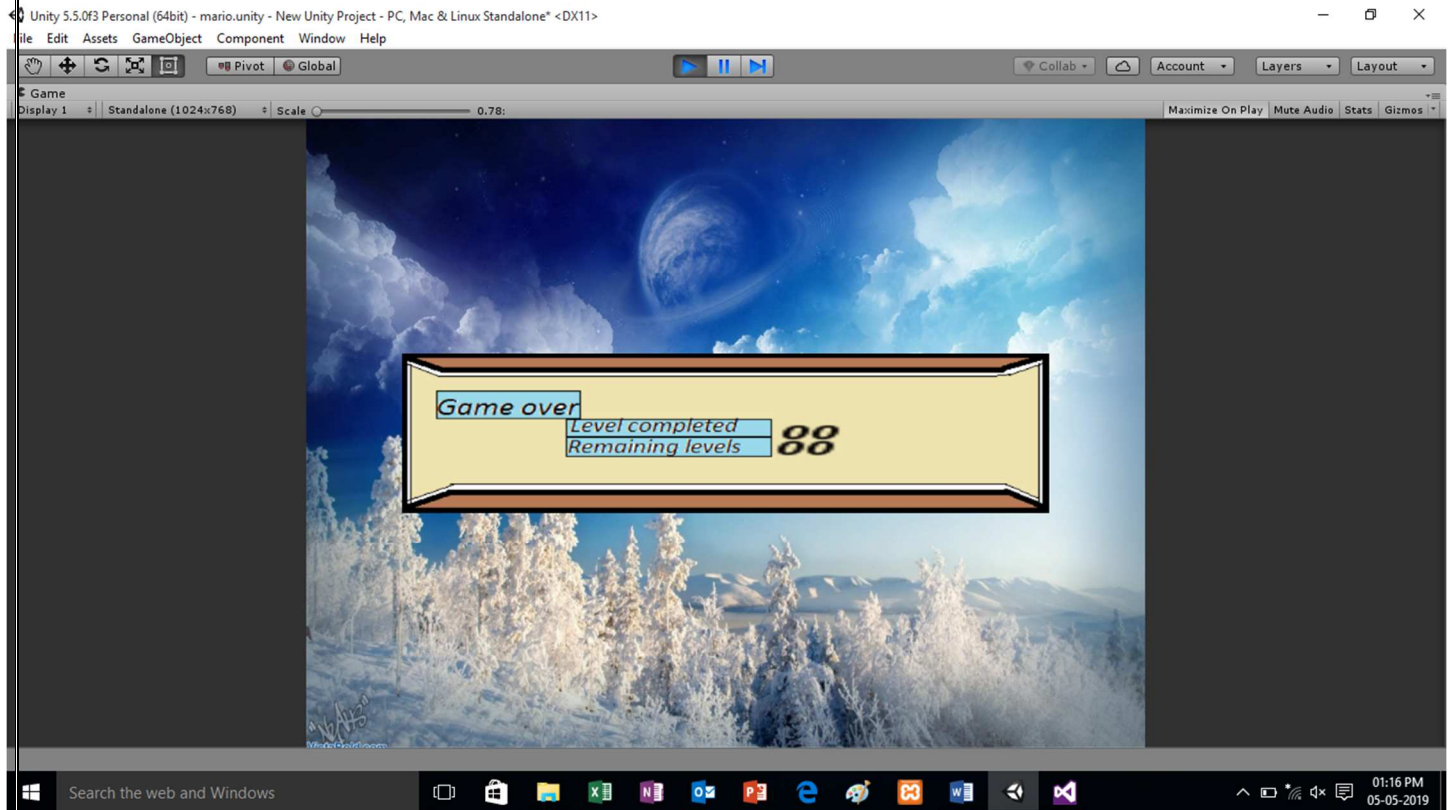
This window shows the level 3 of the developed game, in this level Hen (player) has to take eggs, on getting each egg, points are given to the player.

Finally player has to take trophy to open the gate which takes player to the next level.

Red color blocks are the bricks which act as obstacles for the player to increase difficulty.

Blue blocks are water if hen falls on them player dies.

9. End screen



This is the final window which shows the score.

Feasibility study

Depending on the results of initial investigation, the survey is expanded to a more detailed feasibility study. Feasibility study is a test of system proposal according to its work ability, impact on the organization ability to meet user needs, and effective use of resources.

The three major areas under the feasibility study of project are:

1. Technical Feasibility
2. Operational Feasibility
3. Economic Feasibility
4. Social Feasibility

3.1 Technical Feasibility

It is a measure of the practicality of a specific technical solution and the availability of the technical expertise and resources.

3.2 Operational Feasibility

The system will be used if it is developed well then be resistance for users that undetermined.

3.3 Economical Feasibility

It looks at the financial aspects of the project; it determines whether the management has resources and budget to invest in the proposed system and the estimate time for the cost to be incurred.

3.4 Social Feasibility

The assessment of social feasibility will be done alongside technical feasibility. Each of the alternative technical solutions that emerge must be evaluated for its social implication.

TESTING

Testing is a process of executing a program with the intent of finding an error. Testing is a crucial element of software quality assurance and presents ultimate review of specification, design and coding.

System Testing is an important phase. Testing represents an interesting anomaly for the software. Thus a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

A good test case is one that has a high probability of finding an as undiscovered error. A successful test is one that uncovers an as undiscovered error.

Testing Objectives:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a probability of finding an as yet undiscovered error.
3. A successful test is one that uncovers an undiscovered error.

Testing Principles:

1. All tests should be traceable to end user requirements.
2. Tests should be planned long before testing begins
3. Testing should begin on a small scale and progress towards testing in large.
4. Exhaustive testing is not possible.

The primary objective for test case design is to derive a set of tests that has the highest likelihood for uncovering defects in software. To accomplish this objective two different categories of test case design techniques are used. They are

- **White box testing.**
- **Black box testing.**

White-box testing:

White-box testing focus on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed.

Black-box testing:

Black box testing is designed to validate functional requirements without regard to the internal workings of a program. Black box testing mainly focuses on the information domain of the software, deriving test cases by partitioning input and output in a manner that provides thorough test coverage. Incorrect and missing functions, interface errors, errors in data structures, error in functional logic are the errors falling in this category.

Testing strategies:

A strategy for software testing must accommodate low-level tests that are necessary to verify that all small source code segments has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

Testing fundamentals:

Testing is a process of executing program with the intent of finding error. A good test case is one that has high probability of finding an undiscovered error. If testing is conducted successfully it uncovers the errors in the software. Testing cannot show the absence of defects, it can only show that software defects present.

Testing Information flow:

Information flow for testing flow's the pattern. Two class of input provided to test the process. The software configuration includes a software requirements specification, a design specification and source code.

Test configuration includes test plan and test cases and test tools. Tests are conducted and all the results are evaluated. That is test results are compared with expected results. When erroneous data are uncovered, an error is implied and debugging commences.

Unit testing:

Unit testing is essential for the verification of the code produced during the coding phase and hence the goal is to test the internal logic of the modules. Using the detailed design description as a guide, important paths are tested to uncover errors with in the boundary of the modules. These tests were carried out during the programming stage itself. All units of Unity3d successfully tested.

Integration testing:

Integration testing focuses on unit tested modules and build the program structure that is dictated by the design phase.

System testing:

System testing tests the integration of each module in the system. It also tests to find discrepancies between the system and its original objective, current specification and system documentation. The primary concern is the compatibility of individual modules. Entire system is working properly or not will be tested here, and specified path ODBC connection will correct or not, and giving output or not are tested here these verifications and validations are done by giving input values to the system and by comparing with expected output. Top-down testing is implementing here.

Acceptance Testing:

This testing is done to verify the readiness of the system for the implementation. Acceptance testing begins when the system is complete. Its purpose is to provide the end user with the confidence that the system is ready for use. It involves planning and execution of functional tests, performance tests and stress tests in order to demonstrate that the implemented system satisfies its requirements.

Tools to special importance during acceptance testing include:

Test coverage Analyzer – records the control paths followed for each test case.

Timing Analyzer – also called a profiler, reports the time spent in various regions of the code are areas to concentrate on to improve system performance.

Test Cases:

Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed.

Using White-Box testing methods, the software engineer can drive test cases that-

- Guarantee that logical decisions on their true and false sides.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structure to assure their validity.

The test case specification for system testing has to be submitted for review before system testing commences.

Maintenance and environment

As the number of computer based systems, grievie libraries of computer software began to expand. In house developed projects produced tones of thousand soft program source statements. Software products purchased from the outside added hundreds of thousands of new statements. A dark cloud appeared on the horizon. All of these programs, all of those source statements-had to be corrected when false were detected, modified as user requirements changed, or adapted to new hardware that was purchased. These activities were collectively called software Maintenance.

The maintenance phase focuses on change that is associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of changes are encountered during the maintenance phase.

1. Correction
2. Adaptation
3. Enhancement
4. Prevention

Correction-Even with the best quality assurance activities are lightly that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.

Maintenance is a set of software Engineering activities that occur after software has been delivered to the customer and put into operation. Software configuration management is a set of tracking and control activities that began when a software project begins and terminates only when the

software is taken out of the operation.

We may define maintenance by describing four activities that are undertaken after a program is released for use:

- a. Corrective Maintenance
- b. Adaptive Maintenance
- c. Perfective Maintenance or Enhancement
- d. Preventive Maintenance or reengineering

Only about 20 percent of all maintenance work are spent "fixing mistakes". The remaining 80 percent are spent adapting existing systems to changes in their external environment, making enhancements requested by users, and reengineering an application for use.

ADAPTATION-Over time, the original environment (E>G., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate change to its external environment.

ENHANCEMENT- As software is used; the customer/user will recognize additional functions that will provide benefit. Perceptive maintenance extends the software beyond its original function requirements.

PREVENTION-Computer software deteriorates due to change, and because of this, preventive maintenance, often called software re-engineering, and must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced. Software configuration management (SCM) is an umbrella activity that is applied throughout the software process. SCM activities are developed to prevent.