“Buck’s Quest” THE GAME

MAJOR PROJECT

SUBMITTED TO THE UNIVERSITY INSTITUTE OF INFORMATION TECHNOLOGY, HIMACHAL PRADESH UNIVERSITY IN PARTIAL FULFILMENT OF THE REQUIREMENT’S FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

(2015-2019)

Supervised By:
Er. Akshay Bhardwaj
(Assistant Professor)
UNIVERSITY INSTITUTE OF INFORMATION TECHNOLOGY

Submitted By:
SURESH
Roll no: 63541
B.Tech-IT 8th Sem

UNIVERSITY INSTITUTE OF INFORMATION TECHNOLOGY

HIMACHAL PRADESH UNIVERSITY SUMMER HILL, SHIMLA 171005

JUNE, 2019

DECLARATION

I **Suresh,** student of **University Institute of Information Technology affiliated to HPU, Summer Hill, Shimla (H.P)** declare that this project report on **"Buck's Quest"** for the completion of degree B.Tech (IT) on a game is entirely made by me on the game **"Buck's Quest"** which is also completely planned, designed and deployed by me having Roll Number 63541, worked under supervision of my project guide **Er. Akshay Bhardwaj (Asst. Professor)**. I solely responsible for the entire development of the game and creation of project report on it.

Suresh (B. Tech - IT)
8th Sem - 63541

CERTIFICATE

This is to certify that this project report on **"Buck's Quest"** game is submitted by **Suresh** Roll Number 63541, who carried out the project work under my supervision. I approve this project for submission of the Bachelor of Technology in the Computer Science & Engineering and Information Technology, University Institute of Information Technology affiliated to HPU, Summer Hill, Shimla (H.P). 171005

Er. Akshay Bhardwaj
Asst. Professor (Project Guide)

ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed Project Coordinator Er. Akshay Bhardwaj **(Asst. Professor) UIIT, HPU Summer Hill,** for their valuable guidance, encouragement, and help for completing this work. Their useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ahead. At the end, I would like to express my sincere thanks to all my friends and others who helped me directly or indirectly during this project work.

Table of Contents

List of Figures

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

“Buck’s Quest” is a 2D Game Created in “Unity®” the game Engine. It is a puzzle and platform game something like **“Limbo”, “That Level Again”** etc. This game is created to explore the various concepts of game development, various concepts like GDD (Game Design Document), life cycle of game development, programming perspectives like scripting the game and logical design of games, graphics development for games, logic development, game level design.

2D video games refer to action happening on a 2D plane and typically are either side-scrolling or vertically-scrolling. What's more, the characters and environments are usually rendered in 2D. 3D video games refer to characters and environments rendered in 3D. Action and movements have depth

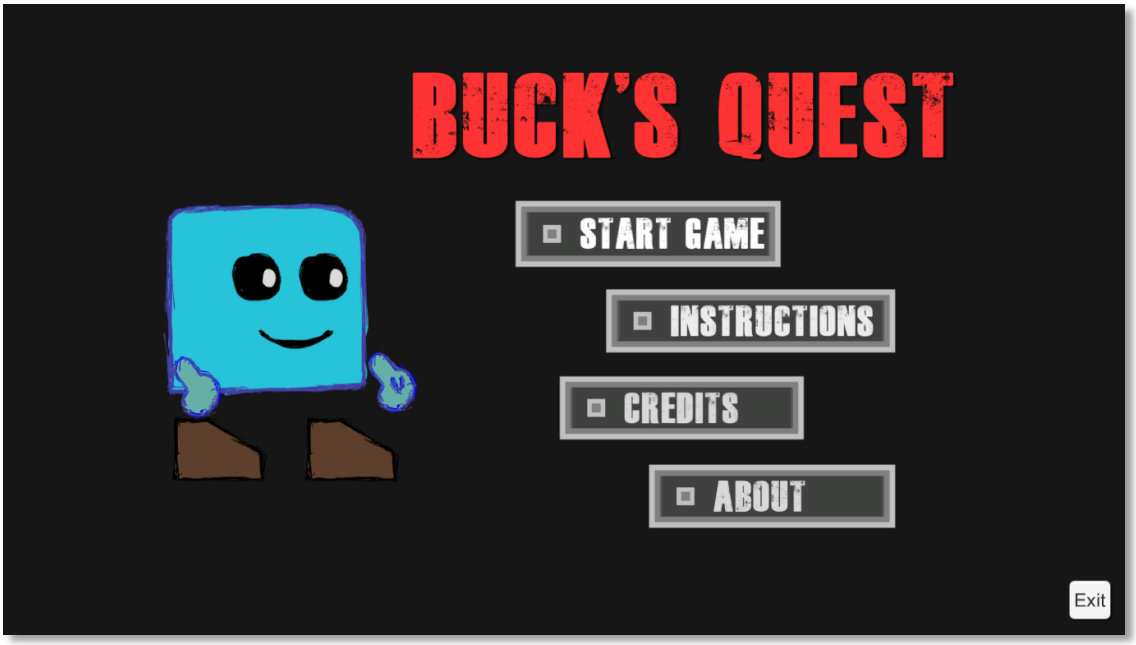


Fig1. Screenshot showing welcome screen.

1.2 VIDEO GAME DEVELOPMENT

Game development is quite an interesting process. Game development teaches all various wide range of topics like story-telling, game programming, project management, team management, risk management, resources management etc. **Video game development** is the process of creating a video game. The effort is undertaken by a developer, ranging from a single person to an international team dispersed across the globe. Development of traditional commercial PC and console games is normally funded by a publisher, and can take several years to reach completion. Indie games usually take less time and money and can be produced by individuals and smaller developers. The independent game industry has been on the rise, facilitated by the growth of new online distribution systems such as Steam and Uplay, as well as the mobile game market for Android and iOS devices.

The first video games, developed in the 1960s, were non-commercial. They required mainframe computers to run and were not available to the general public. Commercial game development began in the '70s with the advent of first-generation video game consoles and early home computers like the Apple I. At that time, owing to low costs and low capabilities of computers, a lone programmer could develop a full and complete game. However, in the late '80s and '90s, ever-increasing computer processing power and heightened expectations from gamers made it difficult for a single person to produce a mainstream console or PC game.

Games are produced through the software development process. Games are developed as a creative outlet and to generate profit. Development is normally funded by a publisher. Well-made games bring profit more readily. However, it is important to estimate a game's financial requirements, such as development costs of individual features. Failing to provide clear implications of game's expectations may result in exceeding allocated budget. In fact, the majority of commercial games do not produce profit. Most developers cannot afford changing development schedule and require estimating their capabilities with available resources before production.

1.3 VIDEO GAME DEVELOPER

A video game developer is a software developer that specializes in video game development – the process and related disciplines of creating video games. A game developer can range from one person who undertakes all tasks to a large business with employee responsibilities split between individual disciplines, such as programming, design, art, testing, etc. Most game development companies have video game publisher financial and usually marketing support. Self-funded developers are known as independent or indie developers and usually make indie games. A developer may specialize in a certain video game console (such as Nintendo's Nintendo Switch, Microsoft's Xbox One, Sony's PlayStation 4), or may develop for a number of systems (including personal computers and mobile devices). Video-game developers specialize in certain types of games (such as role-playing video games or first-person shooters). Some focus on porting games from one system to another, or translating games from one language to another. Less commonly, some do software-development work in addition to games.

Indie Game Developers

Independents are software developers which are not owned by (or dependent on) a single publisher. Some of these developers self-publish their games, relying on the Internet and word of mouth for publicity. Without the large marketing budgets of mainstream publishers, their products may receive less recognition than those of larger publishers such as Sony, Microsoft or Nintendo. With the advent of digital distribution of inexpensive games on game consoles, it is now possible for indie game developers to forge agreements with console manufacturers for broad distribution of their games.

1.4 GAME PROGRAMMING

Game programming, a subset of game development, is the software development of video games. Game programming requires substantial skill in software engineering and computer programming in a given language, as well as specialization in one or more of the following areas: simulation, computer graphics, artificial intelligence,

physics, audio programming, and input. For massively multiplayer online games(MMOG),knowledge of additional areas such as network programming and database programming is requisite. Though often engaged in by professional game programmers, some may program games as a hobby.

Though the programmer's main job is not to develop the game design, the programmers often contribute to the design, as do game artists. The game designer will solicit input from both the producer and the art and programming lead for ideas and strategies for the game design. Often individuals in non-lead positions also contribute, such as copywriters and other programmers and artists. Programmers often closely follow the game design document. As the game development progresses, the design document changes as programming limitations and new capabilities are discovered and exploited.

CHAPTER-02

PLANNING AND ANALYSIS

2.1 COMING UP WITH IDEAS

Since I have already deigned a game before which was also a platformer game i.e. 2D game and specific genre was endless runner game something **Subway Surfer, Temple Run, Sonic Dash, Super Mario Run** etc. Now, for next game I want to explore new genre which is also very interesting to play games and design the games as well which is **Puzzle.** Since the game should be extremely simple to design but it must have some level of challenge as well to explore the various aspects of game design being my second game. So, I skimmed through some diverse games gameplay and then I came to a conclusion that I shall design a Puzzle game which is quite interesting to design and play as we grasp the basics of game design Some best examples of **Puzzle** games are: **Limbo, That Level Again, Inside** etc.

There are a bunch of genres out there. Some have been around for a long time, and for good reason. Think about some of your favourites. Whether it's RTS, FPS, or a racing genre, you have options!

Playdead's Limbo Game

Limbo is a puzzle-platform video game developed by independent studio Playdead. The game was released in July 2010 on Xbox Live Arcade, and has since been ported to several other systems, including the PlayStation 3 and Microsoft Windows. Limbo is a 2D side-scroller, incorporating a physics system that governs environmental objects and the player character. The player guides an unnamed boy through dangerous environments and traps as he searches for his sister. The developer built the game's puzzles expecting the player to fail before finding the correct solution. Playdead called the style of play "trial and death", and used gruesome imagery for the boy's deaths to steer the player from unworkable solutions. The game is presented in black-and-white tones, using lighting, film grain effects and minimal ambient sounds to create an eerie atmosphere often associated with the horror genre.

Fig2. Screenshot of an amazing puzzle game “Limbo”.

Playdead’s Inside Game

Inside (stylized as INSIDE) is a puzzle-platformer adventure game developed and published by Playdead in 2016 for PlayStation 4, Xbox One and Microsoft Windows. The game was released for iOS in December 2017. The player controls a boy in a dystopic world, solving environmental puzzles and avoiding death. It is the spiritual successor to Playdead's 2010 game Limbo, and features similar 2.5D gameplay.

Fig3. Screenshot of another amazing puzzle game “INSIDE”.

2.2 Planning the Game level and Gameplay

2.2.1 Welcome Screen

When the game file is opened following screen shall be the welcome screen. Instruction on the screen will tell the player to start the screen. For example, let's say "Press spacebar to start the game".

Splash Screen

A splash screen is a graphical control element consisting of a window containing an image, a logo, and the current version of the software. A splash screen usually appears while a game or program is launching. A splash page is an introduction page on a website. A splash screen may cover the entire screen or web page; or may simply be a rectangle near the center of the screen or page. The splash screens of operating systems and some applications that expect to be run in full screen usually cover the entire screen.

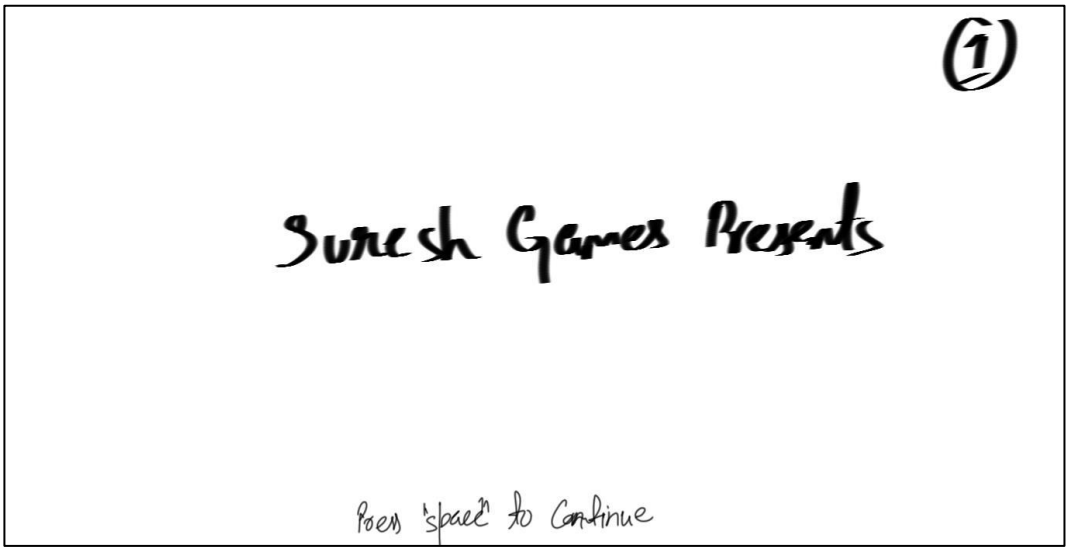


Fig4. Welcome screen of the game.

2.2.2 Start Screen of the Game

As the user presses the space to continue, he reaches the start screen where game character is revealed and in idle animation loop to have some life in the start screen. Start screen contains the Instructions for the user about how to play and what are the controls.

A start screen creates the first impression for a user and should be eye catching. Start screen can be planned and designed inside of Adobe Photoshop and reuse some elements from the scene background to create the high-fidelity assets for the start screen in Illustrator.

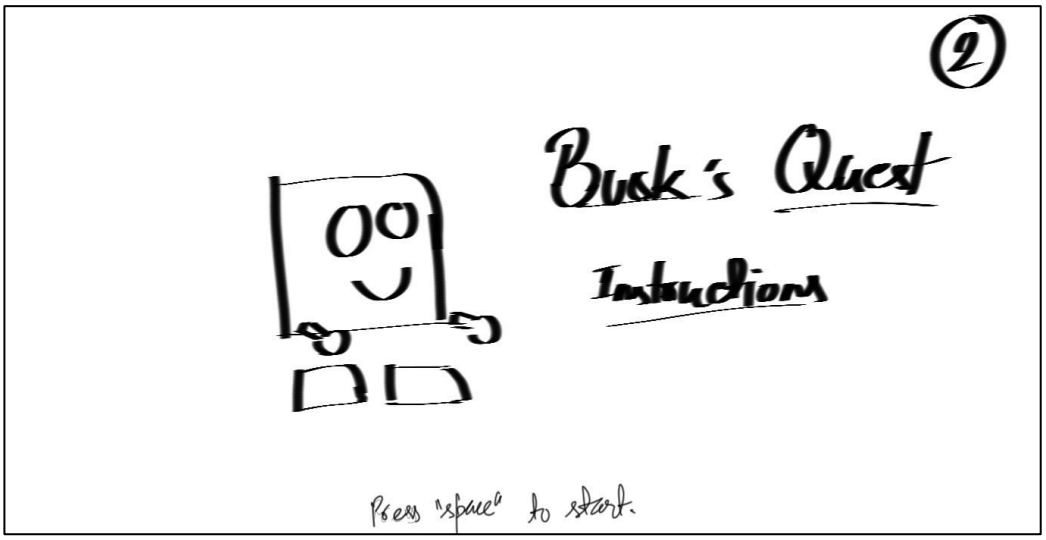


Fig5. Game Start Screen.

2.2.3 Level_01 Design and gameplay planning

This is level_01 design and gameplay planning. Player has to come out of this prison kind of thing to start his/her journey to more interesting levels. Player is surrounded by cave tiles all are identical and immovable but one of them can be pushed to make the path. Tile on which exit is written is the cave tile which player has to find and push to get out of this place and move to next level. It's kind of hit and trial approach to play the game and it creates the interest in the player to challenge himself to get to next level and see the but kind of approach s/he can use to complete the game levels.

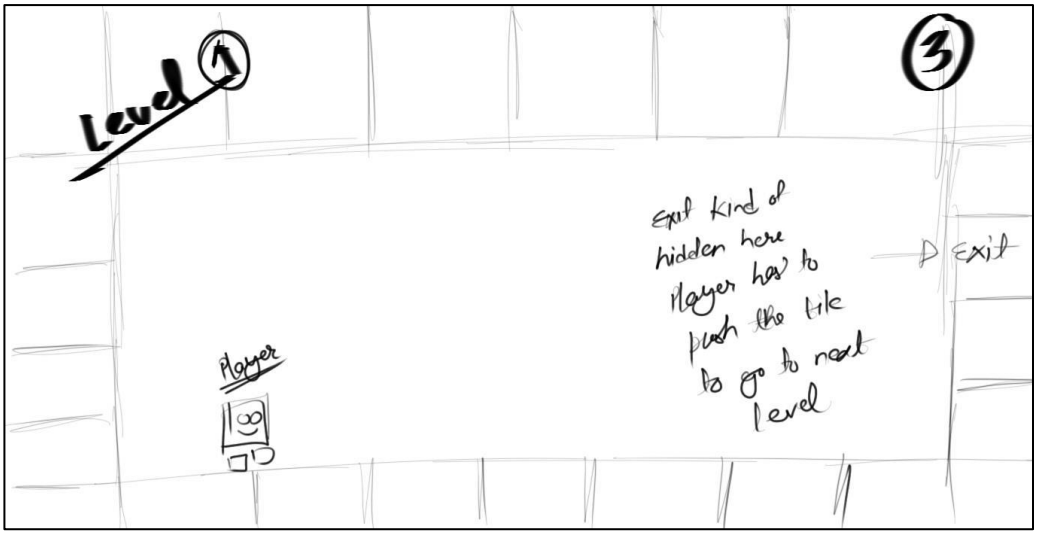


Fig6. Gameplay planning and level design level_01.

2.2.4 Level_02 Design and gameplay planning

In this level, player shall be pushing the cave tile which he was pushing in the previous level i.e. level_01. Here also we have identical tiles which are immovable but one tile can be pushed or pulled to clear the path. Tile given below with cross lines in it should be moved or grabbed to make the path towards exit.

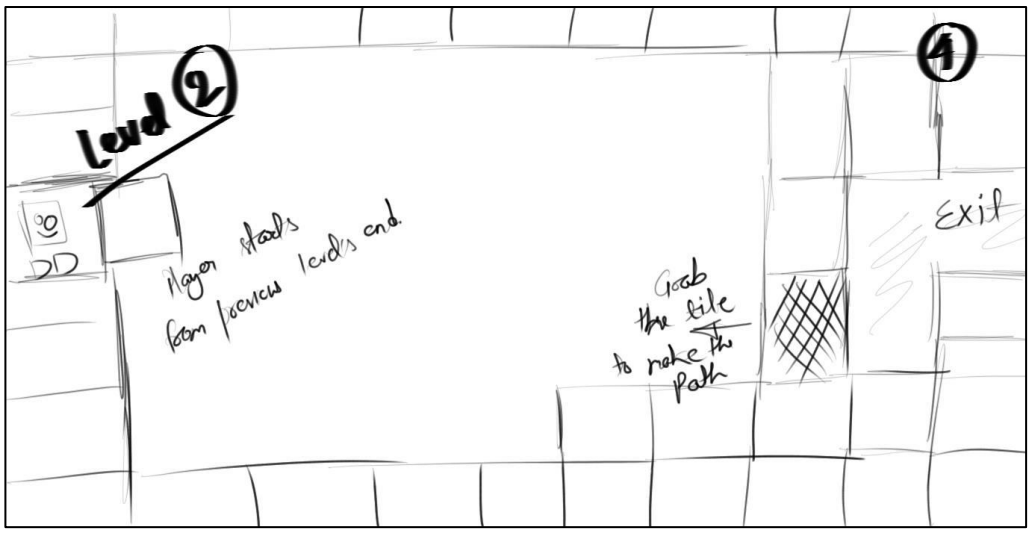


Fig7. Gameplay planning and level design level_02.

2.2.5 Level_03 Design and gameplay planning

Level_03 has some interesting challenge for player; s/he might think that path ahead is empty and s/he can go to next level without any challenge but there is invisible wall to the right of screen then player might think that he has to pull up the tile in the floor to make the path but it's very heavy that player will fail to make the path and now s/he might feel very challenged to complete the level. Then, he will start searching for the path and as I said by hit and trial method s/he shall start looking for path and eventually player will find the path to go to next level. As given in the figure below player has to reach the exit point where there is now wall and is exactly the path to next level.

Fig8. Gameplay planning and level design level_03

2.2.6 Level_04 Design and gameplay planning

Level_04 is pretty simple according to my opinion that player has to grab a few tiles to make the path. But still it has some level of challenge to player think about the approach or actions s/he has to do to complete the level. Since, I have seen in my prototype which I gave to my few friends to play they find it interesting to play this level is not simple as it appeared to be. It really takes time to player to complete this level which is indirectly identical to level_02.

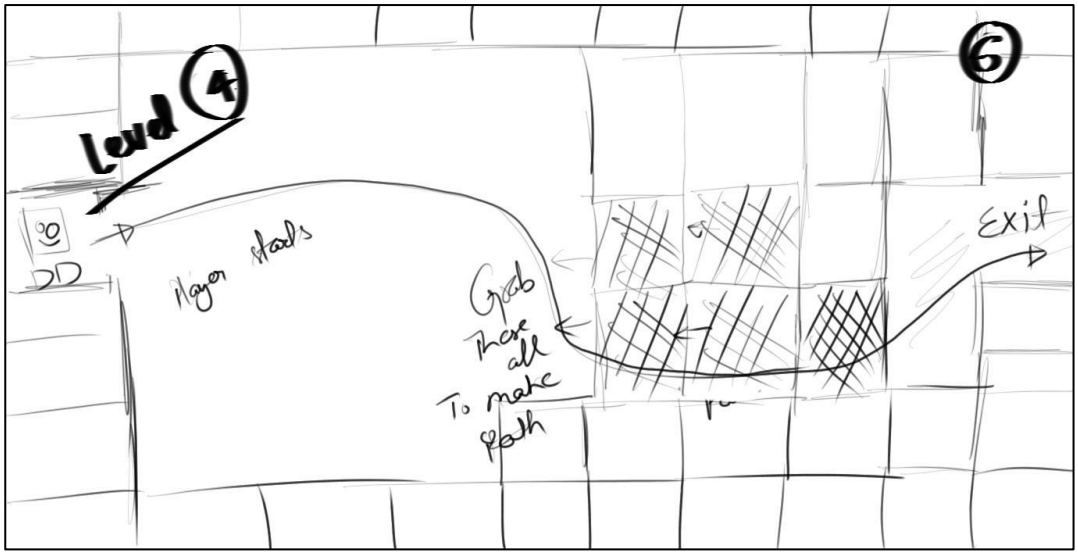


Fig9. Gameplay planning and level design level_04.

2.2.7 Level_05 Design and gameplay planning

Level_05 is almost 99% identical to level_03 in terms of design and look but different approach or actions are needed to complete the level. What might be the steps of a player to complete the level as I have seen in my prototype gameplay. First step player do is go to the exit point of level_03 but since there is wall in right side of screen completely. Player has to come back and pull the tile below in the ground.

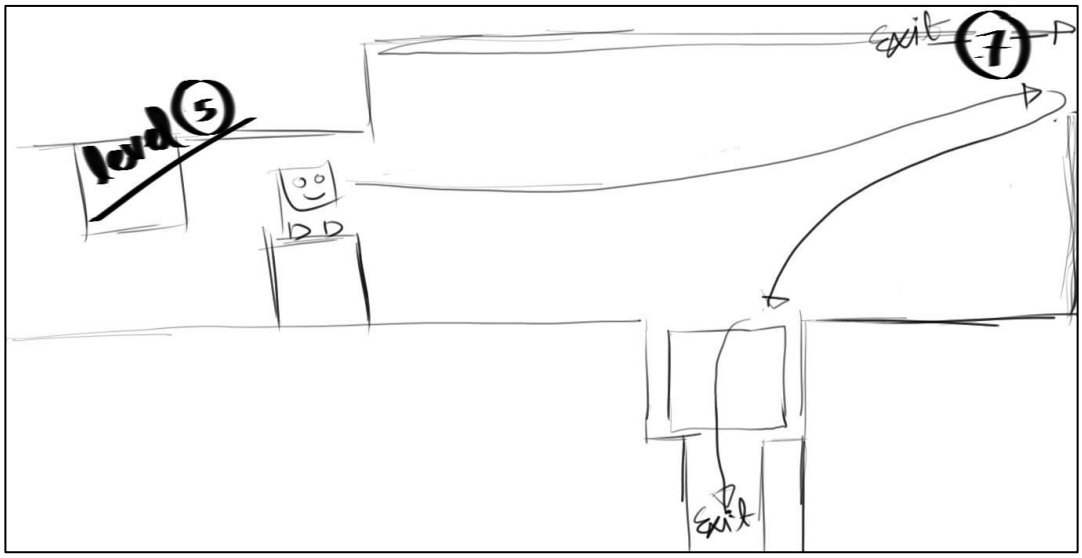


Fig10. Gameplay planning and level design level_05.

2.2.8 Ending and Restarting

As the player reaches the actual exit in the floor in level_05 player has finished the game and directed to the following scene which is indicating “The End” of game. If player wants to replay the game again he has to press spacebar as given in the diagram below and instructions in it.

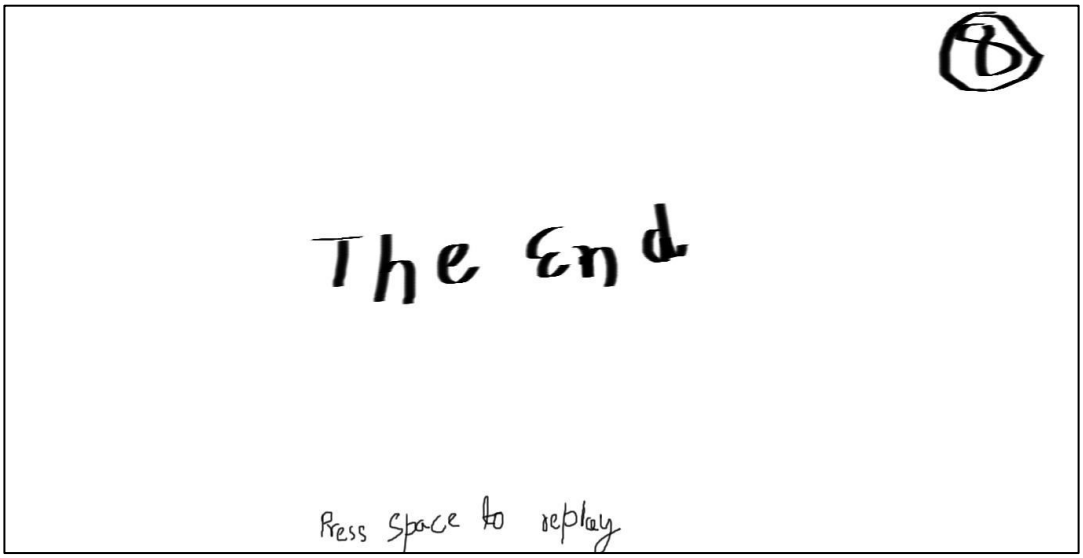


Fig11. Ending and restarting the game.

CHAPTER-03

INTRODUCTION TO "UNITY"

3.1 "Unity" the Game Engine

According to Wikipedia: -

"Unity is a cross-platform game 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 25 platforms."

Unity is quite diverse game engine. It supports three scripting languages: **JavaScript**, **C#** and **Bool**. Unity allows the game developers to develop the 2D as well as 3D games for all kinds of devices ranging from Regular devices to **VR-Devices**. Now it supports around 25 platforms like **Android**, **WindowsPC, MacOSX, iOS, Blackberry, PlayStation, Xbox** etc.

Core of the game designing in unity are **GameObjects**, **Prefabs**, UI Elements etc. Well everything inside the scene is a **GameObject** and any number of scripts can be attached to a single GameObject to perform various kinds of functions. A **Camera, UI Element etc** everything is a GameObject.

"C# for Unity" is a scripting language which has largest userbase among all the three scripting languages. Almost all the scripts extend the **MonoBehaviour**. But all the scripts attached to a GameObject should extend the MonoBehaviour class. MonoBehaviour is a class in namespace **UnityEngine**.

The engine can be used to create three-dimensional, two-dimensional, virtual reality, and augmented reality games, as well as simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering and construction.

Unity gives users the ability to create games and experiences in both 2D and 3D, and 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 UnityScript, which was deprecated in August 2017 after the release of Unity 2017.1 in favor of C#.

Unity gives users the ability to create games and experiences in both 2D and 3D, and 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 with the release of Unity 5, and a version of JavaScript called UnityScript, which was deprecated in August 2017, after the release of Unity 2017.1, in favor of C#.

Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports,[29] and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

As of 2018, Unity has been used to create approximately half of the new mobile games on the market and 60 percent of augmented reality and virtual reality content.

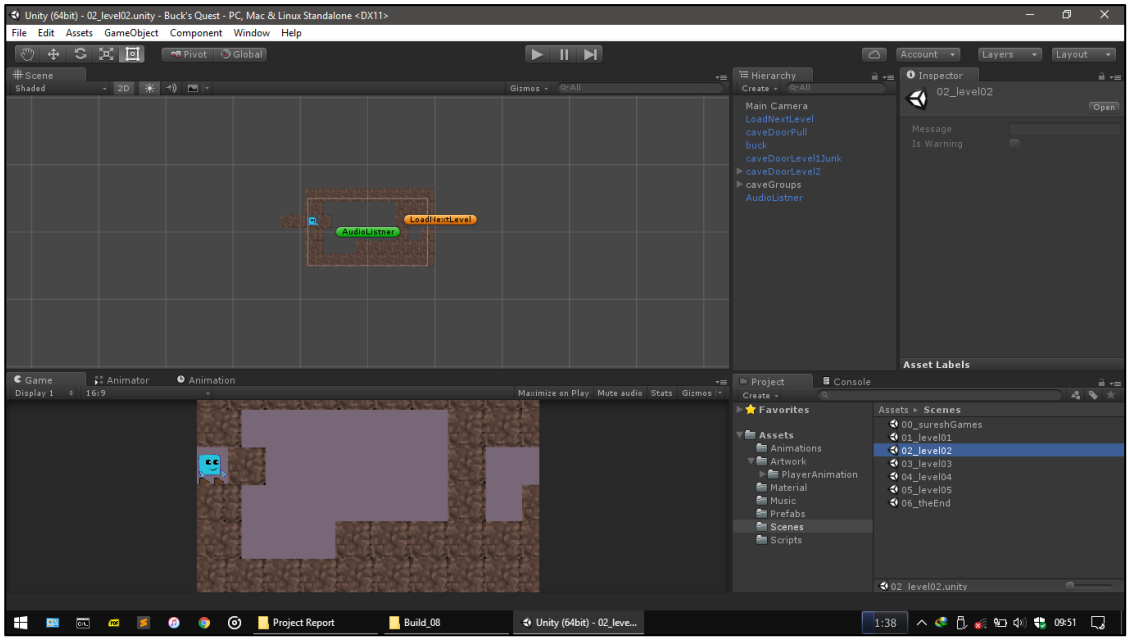


Fig12. Unity Game Engine Interface

3.2 Unity Scripting Environment

Although Unity uses an implementation of the standard Mono runtime for scripting, it still has its own practices and techniques for accessing the engine from **scripts**. The behavior of **GameObjects** is controlled by the **Components** that are attached to them. Unity allows us to create our own Components using **scripts**. These allow us to trigger game events, modify Component properties over time and respond to user input in any way we like. Unity supports the C# programming language natively. C# (pronounced C-sharp) is an industry-standard language similar to Java or C++. In addition to this, many other .NET languages can be used with Unity if they can compile a compatible DLL. A script makes its connection with the internal workings of Unity by implementing a class which derives from the built-in class called MonoBehaviour. We can think of a class as a kind of blueprint for creating a new Component type that can be attached to GameObjects. Each time we attach a script component to a GameObject, it creates a new instance of the object defined by the blueprint. The name of the class is taken from the name we supplied when the file was created. The class name and file name must be the same to enable the script component to be attached to a GameObject.

The main things to note, however, are the two functions defined inside the class. The **Update** function is the place to put code that will handle the frame update for the GameObject. This might include movement, triggering actions and responding to user input, basically anything that needs to be handled over time during gameplay. To enable the Update function to do its work, it is often useful to be able to set up variables, read preferences and make connections with other GameObjects before any game action takes place. The **Start** function will be called by Unity before gameplay begins (ie, before the Update function is called for the first time) and is an ideal place to do any initialization.

CHAPTER-04

GAME DESIGN

4.1 GDD (Game Design Document)

According to Wikipedia: -

"A game design document (often abbreviated GDD) is a highly descriptive living software design document of the design for a video game. A GDD is created and edited by the development team and it is primarily used in the video game industry to organize efforts within a development team. The document is created by the development team as result of collaboration between their designers, artists and programmers as a guiding vision which is used throughout the game development process. When a game is commissioned by a game publisher to the development team, the document must be created by the development team and it is often attached to the agreement between publisher and developer; the developer has to adhere to the GDD during game development process."

Structure of My **GDD** is given below: -

4.1 Name of the Game – "Buck's Quest".

4.2 Creating the Prototype.

4.3 Theme/Genre – Puzzle/2D Game.

4.4 Core Gameplay mechanics brief.

4.5 Project Scope.

- 4.5.1 Hardware.
- 4.5.2 Software Requirement

4.6 Assets needed.

- 4.6.1 Textures.
- 4.6.2 Sprites.
- 4.6.3 Sprite-sheets/Animation.

4.7 Code/Programs.

4.8 Building the Game.

My entire game development is organized on this GDD (Game Design Document). It works as a blue print or roadmap for game development.

A detailed **GDD** is very useful and is being used by all the big studios which help them to set the budget, finish line for various phases during game Development.

The document is created by the development team as result of collaboration between their designers, artists and programmers as a guiding vision which is used throughout the game development process. When a game is commissioned by a game publisher to the development team, the document must be created by the development team and it is often attached to the agreement between publisher and developer; the developer has to adhere to the GDD during game development process.

The purpose of a game design document is to unambiguously describe the game's selling points, target audience, gameplay, art, level design, story, characters, UI, assets, etc. In short, every game part requiring development should be included by the developer in enough detail for the respective developers to implement the said part. The document is purposely sectioned and divided in a way that game developers can refer to and maintain the relevant parts.

Life Cycle: -

Game developers may produce the game design document in the pre-production stage of game development—prior to or after a pitch. Before a pitch, the document may be conceptual and incomplete. Once the project has been approved, the document is expanded by the developer to a level where it can successfully guide the development team. Because of the dynamic environment of game development, the document is often changed, revised and expanded as development progresses and changes in scope and direction are explored. As such, a game design document is often referred to as a living document, that is, a piece of work which is continuously improved upon throughout the implementation of the project, sometimes as often as daily. A document may start off with only the basic concept outlines and become a complete, detailed list of every game aspect by the end of the project.

4.2 Creating the prototype

After having the basic idea of GDD in hands in hard copy, first thing one game developer must do is to create the prototype and reach as many as possible trusted players to test the game. Do they like idea and game mechanics? If yes, then analyze what is the rating and ratio of liking? If no, then host some questionnaire, survey and whatever else is necessary to improve the liking of the game we must do. Because if we make a game which nobody likes then it's not worth working in such projects. A prototype is an early sample, model, or release of a product built to test a concept or process or to act as a thing to be replicated or learned from. It is a term used in a variety of contexts, including semantics, design, electronics, and software programming. A prototype is generally used to evaluate a new design to enhance precision by system analysts and users. Prototyping serves to provide specifications for a real, working system rather than a theoretical one. In some design workflow models, creating a prototype (a process sometimes called materialization) is the step between the formalization and the evaluation of an idea.

4.3 Theme/Genre

"Theme/ Genre – 2D Game/Platformer, Puzzle."

Genre: -

Puzzle video games make up a unique genre of video games that emphasize puzzle solving. The types of puzzles can test many problem-solving skills including logic, pattern recognition, sequence solving, and word completion. The player may have unlimited time or infinite attempts to solve a puzzle, or there may be a time limit, or simpler puzzles may be made difficult by having to complete them in real time, as in Tetris.

The genre is very broad, but it generally involves some level of abstraction and may make use of colours, shapes, numbers, physics, or complex rules. Unlike many video

games, puzzle video games often do make use of "lives" that challenge a player by limiting the number of tries. In puzzle video games, players often try for a high score or to progress to the next level by getting to a certain place or achieving some criteria.

Theme: -

2D games are also called Platform game. The word platform also describes that something held on a platform. Here the player can run, jump, shoot, collect powers on a platform. It is a video game genre; 2D games are become very old. But some developers still like to play 2D games to get the innovative idea, because we learn everything from past. That's why they want to play 2D games to make their game more interesting and get idea to give some new feature in their game. Mostly the characters of 2D games are cartoonish and unrealistic. We can't give a realistic feel in our 2D character. But in 3D it is possible because of depth. By using depth, we can make a character which would look like a real. Mostly platform games are based on some levels, if the player can kill all enemies or cross a certain part which s/he has to cross (like-Mario) then only the player can move to the next level. In next level may be there would be more enemies which the player has to kill. As per my knowledge the era of platform games started in the early 1980's and the 3D games started in mid of 1990. There is some confusion that which is the first 2D games.

Frogs is an arcade game which released in 1978, this is the first game where the character can jump on the screen, making the genre's earliest ancestor. Space Panic, which is also arcade game and released in 1980, is sometimes credited as the first platform game. Donkey Kong is an arcade game created by Nintendo and released in July 1981, was the first game that allowed players to jump over elements which are there and across gaps, making it the first true platformer. There are some versions of Donkey Kong. The next version of Donkey Kong is Donkey Kong Jr. which is also become a famous game. Donkey Kong introduces Mario. The third version of Donkey Kong was not become so famous but it succeeds by Mario Bros, which is a platform game and it has an extra feature of multiplayer, where two players can play simultaneously. By using the same rule in future many gaming companies made multiplayer games. Pitfall is a video game released by Activision in 1982 for Atari 2600.

4.4 Game Mechanics Brief

According to Wikipedia: -

"**Game mechanics** are methods invoked by agents designed for interaction with the game state, thus providing gameplay. All games use mechanics; however, theories and styles differ as to their ultimate importance to the game. In general, the process and study of game design, or ludology, are efforts to come up with game mechanics that allow for people playing a game to have an engaging, but not necessarily fun, experience."

The interaction of various game mechanics in a game determines the complexity and level of player interaction in the game, and in conjunction with the game's environment and resources determine game balance. Some forms of game mechanics have been used in games for centuries, while others are relatively new, having been invented within the past decade.

Game Mechanics in "Buck's Quest": -

1. It uses hit and trial approach to finish the game.
2. Player has any number of chances and almost no restriction to move around in the game level.
3. There is no time limit to complete the game. Player can stay in the same level for infinite time.
4. There is only one thing to reward the player which kind of emotional side which is amazing feeling to complete the level. There is no other scoring system to reward the player. I'm thinking about number scoring system for future release of the game.

4.5 H/S Requirements

4.5.1 Hardware Requirement

S. No.	NAME	HARDWARE
1	Processor Speed	1.6Ghz Minimum
2	RAM	4GB RAM
3	Hard Disk Capacity	10GB
4	Mouse (Recommended)	3 Button Mouse

4.5.2 Software Required

1. Adobe Photoshop CC
Purpose – To Design the various Textures needed for the game.

2. Adobe Illustrator CC
Purpose – To Design the various Sprites.

3. Adobe Animate CC
Purpose – To Create the character animation and other animations.

4. Unity 3D
Purpose – To Compile the Whole game.

5. Mono Develop
Purpose – To write the **C#** scripts for various requirements in the game.

4.6 Assets Needed

4.6.1 Textures

A texture is Simply an image file. Generally, in **PNG, JPEG, Gif** formats but mostly developers like to use **PNG** file format images. To show the different shapes for transparency is used. Since, **JPEG** do not support alpha i.e. transparency therefore rarely used. And **GIF** is mainly developed for **web** therefore it is also avoided.

Textures bring your Meshes, Particles, and interfaces to life! They are image or movie files that we lay over or wrap around GameObjects. As they are so important, they have a lot of properties.

The shaders you use for your objects put specific requirements on which textures you need, but the basic principle is that you can put any image file inside your project. If it meets the size requirements (specified below), it will get imported and optimized for game use. This extends to multi-layer Photoshop or TIFF files - they are flattened on import, so there is no size penalty for your game. Note that this flattening happens internally to Unity, and is optional, so you can continue to save and import your PSD files with layers intact. The PSD file is not flattened, in other words.

Following texture are created in Adobe Photoshop CC and Adobe Illustrator CC.

1. **Cave Piece**

This is the only texture is needed for the game. I used only one texture image and it reduced hell a lot of development headache of the game.

Fig13. Cave tile for level design.

4.6.2 Sprites

Sprite is a computer graphics term for a two-dimensional bitmap that is integrated into a larger scene. Originally sprites referred to independent objects that are composited together, by hardware, with other elements such as a background. The composition occurs as each scan line is prepared for the video output device, such as a CRT, without involvement of the main CPU and without the need for a full-screen frame buffer. Sprites can be positioned or altered by setting attributes used during the hardware composition process. Examples of systems with hardware sprites include the Atari 8-bit family, Commodore 64, Amiga, Nintendo Entertainment System, Sega Genesis, and many coin-operated arcade machines of the 1980s. Sprite hardware varies in how many sprites are supported, how many can be displayed on a single scan line (which is often a lower number), the dimensions and colours of each sprite, and special effects such as scaling or reporting pixel-precise overlap.

Use of the term *sprite* has expanded to refer to any two-dimensional bitmap used as part of a graphics display, even if drawn into a frame buffer (by either software or a GPU) instead of being composited on-the-fly at display time.

4.6.3 Animations

A sprite sheet is a bitmap image file that contains several smaller graphics in a tiled grid arrangement. By compiling several graphics into a single file, you enable Animate and other applications to use the graphics while only needing to load a single file. This loading efficiency can be helpful in situations such as game development where performance is especially important.

We can create a sprite sheet from a selection of any combination of movie clips, button symbols, graphic symbols, or bitmaps. You can select items in the Library panel or on the Stage, but not both. Each bitmap and each frame of the selected symbols appear as a separate graphic in the sprite sheet. If you export from the Stage, any transforms (scaling, skewing, and so on) you have applied to the symbol instance are preserved in the image output.

Following Animations are created using Adobe Photoshop, Adobe Animate CC.

1. **Main Character Animation Sheet1 – Idle and Jump Animation**

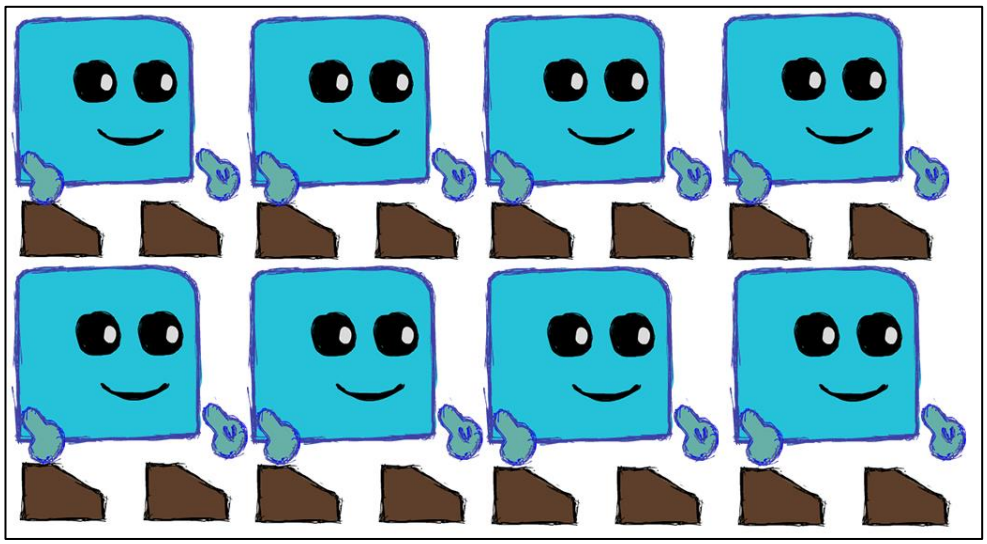


Fig14. Buck Idle Animation sheet.

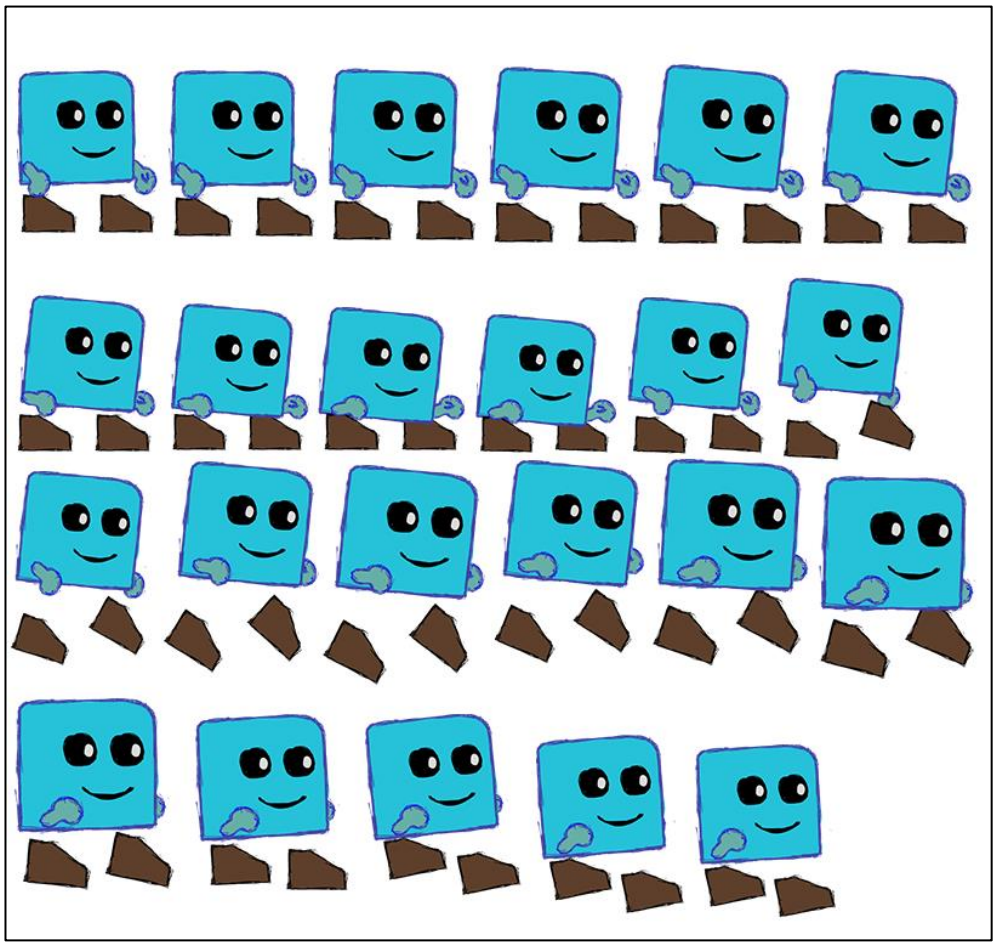


Fig15. Buck Jump Animation sheet.

Spritesheets have been used in games for a long time. Classic games such as **Legend of Zelda: A Link to the Past** and even modern games like **Cut the Rope** have used them.

A **sprite** is a single graphic image that is incorporated into a larger scene so that it appears to be part of the scene. Sprites are a popular way to create large, complex scenes as you can manipulate each sprite separately from the rest of the scene. This allows for greater control over how the scene is rendered, as well as over how the players can interact with the scene.

It is not **uncommon** for games to have tens to **hundreds of sprites**. Loading each of these as an individual image would **consume** a lot of **memory** and **processing** power. To help manage sprites and **avoid** using so **many images**, many games use **spritesheets**.

4.6.4 Logo Design

A logo plays a vital role in deployment of a game. So, coming up with a logo which resonates the game mechanics is very important. Since **"Buck's Quest"** is a puzzle game and presence of the character of the game in the logo is extremely important because a character of the game is core of the game and plays role of interface between play and player. It is very important that a logo should contain the core of game mechanics in it. Therefore, putting the name and character in the logo can be important thing to do and this is what I did to "Buck's Quest" logo.

Fig16. Game Logo

4.7 Programming the scripts

The behaviour of **GameObjects** is controlled by the **Components** that are attached to them. Although Unity's built-in Components can be very versatile, you will soon find you need to go beyond what they can provide to implement your own gameplay features. Unity allows you to create your own Components using **scripts**. These allow you to trigger game events, modify Component properties over time and respond to user input in any way you like.

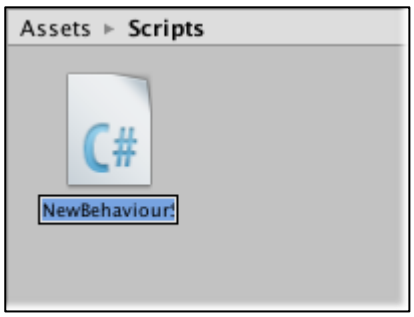
Unity supports the C# programming language natively. C# (pronounced C-sharp) is an industry-standard language similar to Java or C++.

In addition to this, many other .NET languages can be used with Unity if they can compile a compatible DLL.

Creating the Scripts: -

Unlike most other assets, scripts are usually created within Unity directly. You can create a new script from the Create menu at the top left of the Project panel or by selecting **Assets > Create > C# Script** from the main menu.

The new script will be created in whichever folder you have selected in the Project panel. The new script file's name will be selected, prompting you to enter a new name.



Anatomy of a C# Script file: -

When you double-click a script Asset in Unity, it will be opened in a text editor. By default, Unity will use Visual Studio, but you can select any editor you like from the **External Tools** panel in Unity's preferences (go to **Unity** > **Preferences**).

```
using UnityEngine;
using System.Collections;
public class MainPlayer : MonoBehaviour {
    // Use this for initialization
    void Start () {

    }
    // Update is called once per frame
    void Update () {

    }
}
```

Following Scripts were written in C# for all the GameObjects in the Game: -

1. **BuckControls.cs: -**

   This script is used to take the input from user and then apply the locomotion in game. If user press left/right arrow then s/he shall move left/right. If user presses the left/right arrow keys and shift then player shall run. If player wants to grab something then s/he shall press the ctrl key with arrow keys.

```
using UnityEngine;
using System.Collections;

public class BuckControls : MonoBehaviour {

     Animator animator;
     Rigidbody2D body2D;
     SpriteRenderer render2D;

     public float walkSpeed = 5;
     public float runSpeed  = 10;
     public float jumpSpeed = 20;

     // Use this for initialization
     void Start () {
          animator = GetComponent<Animator> ();
          body2D = GetComponent<Rigidbody2D> ();
          render2D = GetComponent<SpriteRenderer> ();
     }
```

```
      // Update is called once per frame
      void Update () {
            float speedX = 0f;
            float speedY = 0f;
//=================================================================
            //Buck vanilla walk
//=================================================================
            if (Input.GetKey ("right") && !Input.GetKey
(KeyCode.RightControl)) {
                  speedX = walkSpeed;
                  render2D.flipX = false;
                  animator.SetInteger ("AnimState", 1);
            } else if (Input.GetKey ("left")&& !Input.GetKey
(KeyCode.RightControl)) {
                  speedX = -walkSpeed;
                  render2D.flipX = true;
                  animator.SetInteger ("AnimState", 1);
            }else
                  animator.SetInteger ("AnimState", 0);
//=================================================================
            //Buck Jump
//=================================================================
            if (Input.GetKey ("up")) {
                  body2D.gravityScale = 20;
                  speedY = jumpSpeed;
                  animator.SetInteger ("AnimState", 3);
            }
//=================================================================
            //Buck Run(shift key)
//=================================================================
            if (Input.GetKey ("right") && Input.GetKey
(KeyCode.RightShift)) {
                  speedX = runSpeed;
                  render2D.flipX = false;
                  animator.SetInteger ("AnimState", 2);
            } else if (Input.GetKey ("left")  && Input.GetKey
(KeyCode.RightShift)) {
                  speedX = -runSpeed;
                  render2D.flipX = true;
                  animator.SetInteger ("AnimState", 2);
            }
//=================================================================
            //Buck CTRL walk
//=================================================================

//=========================**************************=======
            if (render2D.flipX == false && Input.GetKey
(KeyCode.RightControl) && Input.GetKey ("right")) {
                  //render2D.flipX = false;
                  speedX = walkSpeed;
                  animator.SetInteger ("AnimState", 4);
            }
            else if (render2D.flipX == false && Input.GetKey
(KeyCode.RightControl) && Input.GetKey ("left")) {
                  //render2D.flipX = false;
```

```
                speedX = -walkSpeed;
                animator.SetInteger ("AnimState", 5);
          }
          else if (render2D.flipX == true && Input.GetKey
(KeyCode.RightControl) && Input.GetKey ("right")) {
                //render2D.flipX = true;
                speedX = walkSpeed;
                animator.SetInteger ("AnimState", 5);
          }
          else if (render2D.flipX == true && Input.GetKey
(KeyCode.RightControl) && Input.GetKey ("left")) {
                //render2D.flipX = true;
                speedX = -walkSpeed;
                animator.SetInteger ("AnimState", 4);
          }
//=========================**************************=======

//=================================================================
          //Assgining the locomotion
//=================================================================
          body2D.velocity = new Vector2 (speedX, speedY);
//=================================================================
     }
}
```

2. LoadNextScene.cs: -

When game start loading welcome screen then this script helps us to control the loading of different panels and after all main purpose is to load the next scene i.e. level_01. Different panels can only be accessed or skipped after they finish loading completely. Two timers are used to calculate the time so that next panels or scenes can loaded. First timer is running on update( ) method while second timer uses a coroutine method because we want the second timer to run after the successful loading of second panel.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LoadNextScene : MonoBehaviour {


     public string nextSceneName;
     public GameObject panel_01, panel_02;
     public float panelWaitTime = 14.0f;
     public float sceneWaitTime = 8.0f;
     public float timer1 = 0.0f;
     public float timer2 = 0.0f;

     private bool loadLock;
     private bool lockPanelLoad;
```

```
	// Use this for initialization
	void Start () {
		panel_01.SetActive (true);
		panel_02.SetActive (false);
	}

	// Update is called once per frame
	void Update () {
		timer1 += Time.deltaTime;

		if (panelWaitTime < timer1 && !lockPanelLoad) {
			if (Input.GetKeyDown ("space")) {
				panel_01.SetActive (false);
				lockPanelLoad = true;
				panel_02.SetActive (true);
				StartCoroutine ("Timer2");

			}
		}

		if (sceneWaitTime < timer2 && lockPanelLoad) {
			if (Input.GetKeyDown ("space") && !loadLock) {
				LoadScene ();
			}
		}
	}

	IEnumerator Timer2(){
		for (;;) {
			timer2 += Time.deltaTime;
			yield return null;
		}
	}

	void LoadScene(){
		loadLock = true;
		SceneManager.LoadScene (nextSceneName);
	}
}
```

3. GrabDoor.cs: -

This script is used to grab the cave tiles to make the clear path.

```
using UnityEngine;
using System.Collections;

public class GrabDoor : MonoBehaviour {

	public float walkSpeed = 1;

	void OnTriggerStay2D(Collider2D target){
		if (target.gameObject.tag == "Player") {
```

```
                  if (Input.GetKey (KeyCode.RightControl) &&
Input.GetKey ("left")) {
                        this.transform.Translate (-walkSpeed, 0f,
0f, Space.Self);
                  }
            }
      }

}
```

4. **LoadNextLevel.cs: -**

   This script is used to load the next level scenes when the player riches the collider to enter into new level.

```
using UnityEngine;
using System.Collections;

public class GrabDoor : MonoBehaviour {

      public float walkSpeed = 1;

      void OnTriggerStay2D(Collider2D target){
            if (target.gameObject.tag == "Player") {
                  if (Input.GetKey (KeyCode.RightControl) &&
Input.GetKey ("left")) {
                        this.transform.Translate (-walkSpeed, 0f,
0f, Space.Self);
                  }
            }
      }

}
```

CHAPTER-05

DEPLOYMENT

5.1 Building the Game Package

Building the game for deployment is the final step of game development phase. Whole game development fails if we fail to deploy the game to the audience/users as many as possible. Unity supports wide range of platforms for which game packages can be built. A game can be built for following platforms: -

Windows, Android, iOS, MacOSX, Window Store, Windows Phone, Xbox, Blackberry, Linux, Tizen, HTML5, Samsung TV, Apple TV, PlayStation, Facebook etc.

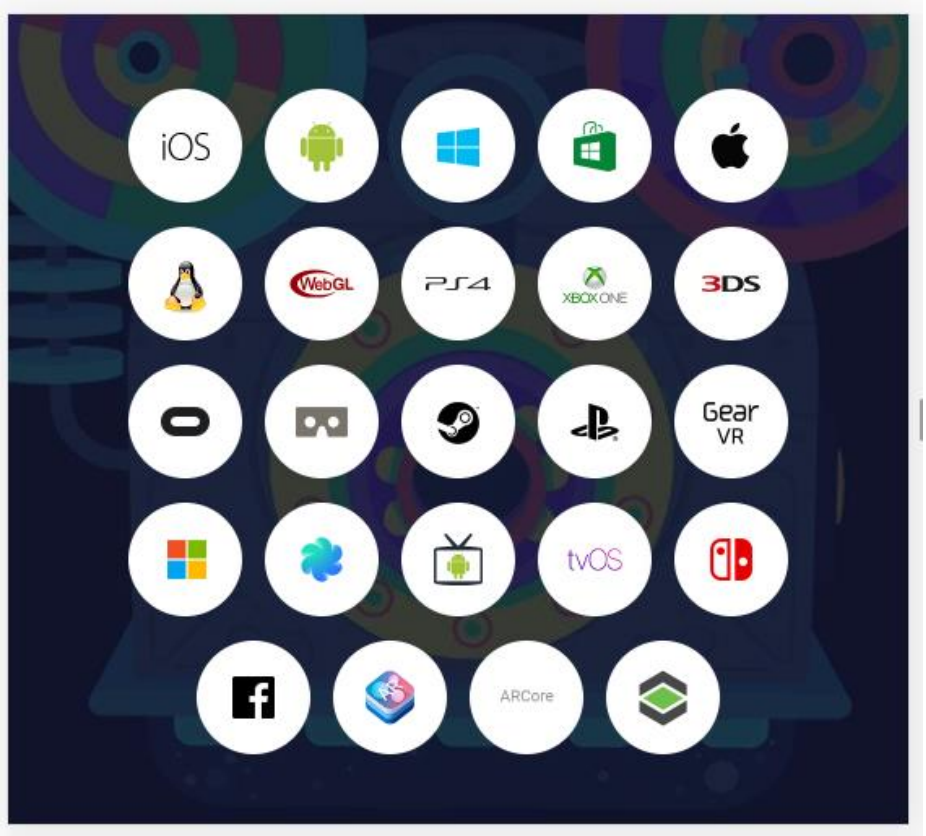


Fig17. List of the platforms for which games can be deployed.

In General, we can say unity supports all five major platforms: -

1. Mobile devices.
2. VR and AR Devices.
3. Desktop devices.
4. Console Devices.
5. Web.

After building the game particular package should be uploaded to regarding store so that package is accessible to various users in the internet.

Following is the list for various platforms: -

Sr.	**Platform**	**Address**
01.	Android	Playstore i.e. http://play.google.com
02.	iOS	AppStore i.e. https://www.apple.com/in/ios/app-store/
03.	Windows Store	Microsoft Store i.e. https://www.microsoft.com/en-in/store/apps/windows

I have uploaded my game to http://itch.io as a zip file. Game is released only for one **WindowsPC** and can be downloaded from following link from my "**itch.io**" profile: -

Download here: -

https://shhotu010.itch.io/bucks-quest

Watch gameplay here: -

https://www.youtube.com/watch?v=IlDl7SmMhPA

REFERENCES

Following is the list of references to refer to various topics of my report: -

1. https://en.wikipedia.org/wiki/Video_game_development
2. https://en.wikipedia.org/wiki/Video_game_developer
3. https://en.wikipedia.org/wiki/Game_programming
4. https://playdead.com/games/limbo/
5. https://playdead.com/games/inside/
6.https://play.google.com/store/apps/details?id=ru.iamtagir.game.android&hl
7. https://unity.com/
8. https://docs.unity3d.com/530/Documentation/ScriptReference/index.html
9. https://en.wikipedia.org/wiki/Game_design_document
10. https://en.wikipedia.org/wiki/Prototype
11. https://en.wikipedia.org/wiki/Platform_game
12. https://en.wikipedia.org/wiki/Game_mechanics
13. https://www.adobe.com/in/products/animate.html
14. https://www.adobe.com/in/products/photoshop.html
15. https://www.adobe.com/in/products/illustrator.html
16. https://unity3d.com/unity/features/multiplatform