**KonQr - 3D**

**Mini Project Batch Number 13A06**

Mini-project report submitted in partial fulfillment of the requirement for the award of the degree of B.Tech in Information Technology.

**BY**

Name of the candidate(s) Reg.No:

1. Shikha Sharma 13251A1250

2. U.N.Jyothi Reddy 13251A1256

3. Sreya V.S 13251A1257

4. K.Venkata Mithravinda 13251A1258



**Department of IT**

**G. Narayanamma Institute of Technology and Science**

**(for Women)**

Autonomous of JNTUH, Accredited by NAAC of UGC

An ISO 9001-2008 Certified Institution

**Shaikpet, Hyderabad-08**

**July, 2015**

(ii)

**CERTIFICATE**

This is to certify that the Mini-project report entitled “KonQr 3D” is a bonafide work done by

Name of the candidate(s) Reg.No:

1. Shikha Sharma 13251A1250

2. U.N.Jyothi Reddy 13251A1256

3. Sreya V.S 13251A1257

4. K.Venkata Mithravinda 13251A1258

under the guidance of Asst.Prof.M.Deepthi during 14 December 2015 to 4 April 2016, in partial fulfillment for the award of degree in B.Tech in Information Technology, from G.Narayanamma Institute of Technology and Science, Autonomous of JNTUH .

Internal Guide Head of the Department

Asst. Prof. M.Deepthi Dr.I.RaviPrakash Reddy

(iii)

**CERTIFICATE**

This is to certify that the Min-project report entitled **“KONQR - 3D”** that is being submitted by

Name of the candidate(s) Reg.No:

1. Shikha Sharma 13251A1250

2. U.N.Jyothi Reddy 13251A1256

3. Sreya V.S 13251A1257

4. K.Venkata Mithravinda 13251A1258

in partial fulfillment for the award of B.Tech in Information Technology to the G.Narayanamma Institute of Technology and Science is a record of bonafide work carried out by her at our organization/institution.

**Signature of Head/Director**

**of Organization/Institution**

**Name and Designation**

(iv)

**ACKNOWLEDGEMENT**

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all along the completion of project work.

We owe our profound gratitude to our project guide Asst.Prof.M.Deepthi, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good outcome.

We would not forget to remember Dr.I.RaviPrakash Reddy, Head of the Department, Information Technology, for his unlisted encouragement and more over for his timely support and guidance till the completion of our project work.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Department of Information Technology especially Ms. Sravanthi and Ms.Venkateswaramma, which helped us in successfully completing our project work.

**(v)**

**Abstract**

KonQr 3D, a uniplayer game comprises multiple 3D models. The player equipped with weapons has to cross multiple levels one by one in order to finish the game. There will be various enemies and one fighter battling to conquer the premises and win the treasure. The addition of external buttons to emulate keys will be used to interact with the game software. The characters will be animated and supported by unique parameters for each, in addition to relevant audio support.

The game comprises of a player, in the form of a girl and is equipped with a gun in order to attack the enemies. The player health variation depends on the ability to combat the enemies and number of times the player is attacked.

Concepts of 3-dimensional geometry, gravity and Physics have been used to achieve highly realistic look and feel. The game shall be attempted to support multiple platforms and shall be user friendly, omitting age and compatibility constraints respectively.

(vi)

**Table of contents**

1. Introduction

1.1 Purpose of Project

1.2 Literature Survey/Background of project

1.3 Existing System

1.4 Proposed System

1. Scope

2.1 Project Scope

2.2 Out of Scope

1. Analysis/Requirement Specifications

3.1 User Interface

3.2 Software Interface

3.3 Hardware Interface

3.4 Functional Requirements

3.5 Non-Functional Requirements

1. Design

4.1 High level conceptual solution Architecture

4.2 UML Diagrams or E-R diagrams (need for database projects)

4.2.1 Class Diagram

4.2.2 Sequence Diagram

4.2.3 Collaboration Diagram

4.2.4 Activity Diagram

4.2.5 Usecase Diagram

4.2.6 State Chart Diagram

1. Implementation

5.1 Module Descriptions

5.2 Programs/Algorithms

1. Testing

6.1 Introduction & Types of Testing used (Specific to your project)

6.2 Test Cases

1. Snapshots or Output Screens
2. Conclusion & Future Scope (Further enhancements)
3. Acronyms and Business terms

1

**1. Introduction**

**1.1 Purpose:**

KonQr 3D has been designed to enhance computation and assessment capability of the player. Using the capability to calculate the target’s distance and trigger the riffle, the player can injure the target. Thus helping enhancement of forecasting withstand and strategizing abilities.

**1.2 Background of the problem:**

Most of the high-end games are expensive and very few are multi-platform. Most games include long and herculean tasks to reach the destination and are generally single-platform.

**1.3 Proposed system:**

KonQr 3D, a uniplayer game comprises multiple 3D models. The player equipped

with weapons has to cross multiple levels one by one in order to finish the game. There

will be multiple enemies and one fighter battling to conquer the premises. The addition of

external buttons to emulate keys will be used to interact with the game software. The

characters will be animated and supported by unique parameters for each, in addition to relevant audio support.Concepts of computer graphics shall also be used to achieve highly realistic look and feel. The game shall be attempted to support multiple platforms and shall be user friendly, omitting age and compatibility constraints respectively.

2

**2. Scope**

**Project scope:**

* Score card: A continuous score monitoring
* Game history: Information about tools used will be specified
* Tools and tips: Details of the controls to be used for the player’s motion are mentioned accompanied with user’s guide.

**3. Analysis/Requirement Specifications**

1. **User interface:**

The user interface will be primarily consistent on desktops or laptops using windows or linux operating systems, webplayer and secondarily on Android phones.

**b. Hardware interface:**

As the game does not demand internet connection, a basic assembled system is required.

**Software interfaces :**

Languages : c# and JavaScript.

Software or tech : Unity Game Engine, Mono develop, Microsoft visual Studio 2015 .

**Unity Game engine:**

**Unity** is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles,mobile devices and websites. With an emphasis on portability, the engine targets the following APIs: Direct3D on Windows and Xbox 360; OpenGL on Mac and Windows; OpenGL ES on Android and iOS; and proprietary APIs on video game consoles. Unity allows specification of texture compression and resolution settings for each platform the game engine supports, 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. Unity's graphics engine's platform diversity can provide a shader with multiple variants and a declarative fallback specification, allowing Unity to detect the best variant for the current video hardware; and if none are compatible, fall back to an alternative shader that may sacrifice features for performance.

Unity is notable for its ability to target games to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles.Supported platforms include Android, Apple TV, BlackBerry 10, iOS, Linux, Nintendo 3DS line, OS X, PlayStation 3, PlayStation 4, PlayStation Vita, Unity Web Player (including Facebook[), Wii, Wii U, Windows Phone 8, Windows, Xbox 360, and Xbox One.

Unity Web Player is a browser plugin that is supported in Windows and OS X only.

**Unity editor**:

A UNIQUELY EXTENDIBLE TOOL

Unity’s carefully designed APIs make it easy to extend the Unity engine in more ways than expected.

The editor can be extended, or chosen from over 1700 free and paid extensions on the Unity Asset Store.

CREATE HIGHLY-OPTIMIZED CONTENT

Unity’s tools are used to deliver reliable performance, smooth framerate and superb player experience across target platforms. The games created perform better at runtime, reduce graphics bottlenecks and take control of asset loading.

AMAZING VISUAL FIDELITY, RENDERING POWER AND AMBIENCE

The game looks just how one envisaged it with Real-time Global Illumination and physically-based shader. From luminous day, to the gaudy glow of neon signs at night; from sunshafts, to dimly lit midnight streets and shadowy tunnels – an evocative dynamic game to enthrall players on any platform is obtained.

RAPID ITERATION

Unity’s Play Mode is an incredibly powerful development tool for rapid iterative editing. Press Play and instantly one is inside the game, playing and previewing how it will look in its platform-specific final build. Pause it, and alter values, assets, scripts and other properties, and instantly see the results. And, one can step through your game frame by frame for easy debugging.

**MONODEVELOP:**

**MonoDevelop** is an open source integrated development environment for Linux, OS X,and Windows. Its primary focus is development of projects that use Mono and .NET frameworks. MonoDevelop integrates features similar to those of [NetBeans](https://en.wikipedia.org/wiki/NetBeans) and Microsoft Visual Studio, such as automatic code completion, source control, a [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUI) and [Web designer](https://en.wikipedia.org/wiki/HTML_editor). MonoDevelop integrates a [Gtk#](https://en.wikipedia.org/wiki/Gtk_Sharp) GUI designer called [Stetic](https://en.wikipedia.org/wiki/MonoDevelop#Stetic).[[7]](https://en.wikipedia.org/wiki/MonoDevelop#cite_note-Stetic-7) It supports [Boo](https://en.wikipedia.org/wiki/Boo_(programming_language)), [C](https://en.wikipedia.org/wiki/C_(programming_language)), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [CIL](https://en.wikipedia.org/wiki/Common_Intermediate_Language), [D](https://en.wikipedia.org/wiki/D_(programming_language)), [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)),[Oxygene](https://en.wikipedia.org/wiki/Oxygene_(programming_language)), [Vala](https://en.wikipedia.org/wiki/Vala_(programming_language)), and [Visual Basic.NET](https://en.wikipedia.org/wiki/Visual_Basic.NET).

A customized version of MonoDevelop ships with [Unity](https://en.wikipedia.org/wiki/Unity_(game_engine)), the game engine by [Unity Technologies](https://en.wikipedia.org/wiki/Unity_Technologies).

MonoDevelop can be used on [Windows](https://en.wikipedia.org/wiki/Windows), [OS X](https://en.wikipedia.org/wiki/OS_X) and [Linux](https://en.wikipedia.org/wiki/Linux). The first two have been officially supported since version 2.2. [Xamarin](https://en.wikipedia.org/wiki/Xamarin) offers a rebranded version of MonoDevelop 4.0 as Xamarin Studio which now uses platform-specific code in various places to enhance the look and feel. While Mono provides a package for [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)) 10 running on[SPARC](https://en.wikipedia.org/wiki/SPARC), MonoDevelop packages for [OpenSolaris](https://en.wikipedia.org/wiki/OpenSolaris) are only provided by groups from the OpenSolaris community. MonoDevelop on [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD) is likewise supported only by the FreeBSD community.

# MonoDevelop

MonoDevelop is the Integrated Development Environment (IDE) supplied with Unity. An IDE combines the familiar operation of a text editor with additional features for debugging and other project management tasks. The text editor will not be covered here since it is fairly intuitive, but the integration of the editor and debugger with Unity are described below.

## Setting Up MonoDevelop

MonoDevelop is installed by default with Unity, although there is the option to exclude it from the installation on Windows. You should check that MonoDevelop is set as the external script editor in the Preferences (menu: Unity > Preferences and then select the External Tools panel). With this option enabled, Unity will launch MonoDevelop and use it as the default editor for all script files.

## Setting Up the Debugger

To enable MonoDevelop’s source level debugging (see below for details) you should firstly check that the Editor Attaching option is enabled in the Preferences on the External Tools panel. Then, you should synchronize the Unity project with the MonoDevelop project (menu: Assets > Open C# Project). Also, make sure that the Development Build and Script Debugging options are enabled in the Build Settings for your target platform (menu: File > Build Settings). If you are building a webplayer then you should additionally check that the development release channel setting is enabled on the player’s context menu in the browser (right click on Windows or cmd-click on Mac OSX).

**MICROSOFT VISUALSTUDIO**

**icrosoft Visual Studio** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) from [Microsoft](https://en.wikipedia.org/wiki/Microsoft). It is used to develop [computer programs](https://en.wikipedia.org/wiki/Computer_program) for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), as well as [web sites](https://en.wikipedia.org/wiki/Web_site), [web applications](https://en.wikipedia.org/wiki/Web_application) and [web services](https://en.wikipedia.org/wiki/Web_service). Visual Studio uses Microsoft software development platforms such as [Windows API](https://en.wikipedia.org/wiki/Windows_API), [Windows Forms](https://en.wikipedia.org/wiki/Windows_Forms), [Windows Presentation Foundation](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation), [Windows Store](https://en.wikipedia.org/wiki/Windows_Store) and [Microsoft Silverlight](https://en.wikipedia.org/wiki/Microsoft_Silverlight). It can produce both [native code](https://en.wikipedia.org/wiki/Native_code) and [managed code](https://en.wikipedia.org/wiki/Managed_code).

Visual Studio includes a [code editor](https://en.wikipedia.org/wiki/Code_editor) supporting [IntelliSense](https://en.wikipedia.org/wiki/IntelliSense) (the [code completion](https://en.wikipedia.org/wiki/Code_completion) component) as well as [code refactoring](https://en.wikipedia.org/wiki/Code_refactoring). The integrated [debugger](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building [GUI](https://en.wikipedia.org/wiki/GUI) applications, [web designer](https://en.wikipedia.org/wiki/Web_designer), [class](https://en.wikipedia.org/wiki/Class_(computing)) designer, and [database schema](https://en.wikipedia.org/wiki/Database_schema) designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for [source-control](https://en.wikipedia.org/wiki/Source_control) systems (like [Subversion](https://en.wikipedia.org/wiki/Subversion_(software))) and adding new toolsets like editors and visual designers for [domain-specific languages](https://en.wikipedia.org/wiki/Domain-specific_language) or toolsets for other aspects of the [software development lifecycle](https://en.wikipedia.org/wiki/Software_development_lifecycle) (like the [Team Foundation Server](https://en.wikipedia.org/wiki/Team_Foundation_Server) client: Team Explorer).

Visual Studio supports different [programming languages](https://en.wikipedia.org/wiki/Programming_language) and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include [C](https://en.wikipedia.org/wiki/C_(programming_language)),[[6]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-6) [C++](https://en.wikipedia.org/wiki/C%2B%2B)and [C++/CLI](https://en.wikipedia.org/wiki/C%2B%2B/CLI) (via [Visual C++](https://en.wikipedia.org/wiki/Visual_C%2B%2B)), [VB.NET](https://en.wikipedia.org/wiki/VB.NET) (via [Visual Basic .NET](https://en.wikipedia.org/wiki/Visual_Basic_.NET)), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) (via [Visual C#](https://en.wikipedia.org/wiki/Visual_C_Sharp)), and [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language)) (as of Visual Studio 2010[[7]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-7)). Support for other languages such as [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Node.js](https://en.wikipedia.org/wiki/Node.js), and [M](https://en.wikipedia.org/wiki/MUMPS) among others is available via language services installed separately. It also supports [XML](https://en.wikipedia.org/wiki/XML)/[XSLT](https://en.wikipedia.org/wiki/XSLT), [HTML](https://en.wikipedia.org/wiki/HTML)/[XHTML](https://en.wikipedia.org/wiki/XHTML), [JavaScript](https://en.wikipedia.org/wiki/JavaScript) and [CSS](https://en.wikipedia.org/wiki/Cascading_Style_Sheets). Java (and J#) were supported in the past.

Before Visual Studio 2015, commercial versions of Visual Studio were available for free to students via Microsoft's[DreamSpark](https://en.wikipedia.org/wiki/DreamSpark) program, when only commercial versions supported plugins.[[8]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-8) Starting with Visual Studio 2015, Microsoft provides Community editions, which support plugins, at no cost to all users.

### **Code editor**

Like any other [IDE](https://en.wikipedia.org/wiki/Integrated_Development_Environment), it includes a [code editor](https://en.wikipedia.org/wiki/Code_editor) that supports [syntax highlighting](https://en.wikipedia.org/wiki/Syntax_highlighting) and [code completion](https://en.wikipedia.org/wiki/Autocomplete) using [IntelliSense](https://en.wikipedia.org/wiki/IntelliSense) for [variables](https://en.wikipedia.org/wiki/Variable_(programming)), [functions](https://en.wikipedia.org/wiki/Subroutine), [methods](https://en.wikipedia.org/wiki/Method_(computer_science)), [loops](https://en.wikipedia.org/wiki/Program_loops) and [LINQ](https://en.wikipedia.org/wiki/LINQ) queries. IntelliSense is supported for the included languages, as well as for [XML](https://en.wikipedia.org/wiki/XML) and for [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) and [JavaScript](https://en.wikipedia.org/wiki/JavaScript) when developing web sites and [web applications](https://en.wikipedia.org/wiki/Web_application). Autocomplete suggestions appear in a [modeless](https://en.wikipedia.org/wiki/Modeless) [list box](https://en.wikipedia.org/wiki/List_box) over the code editor window, in proximity of the editing [cursor](https://en.wikipedia.org/wiki/Cursor_(computing)). In Visual Studio 2008 onwards, it can be made temporarily semi-transparent to see the code obstructed by it. The code editor is used for all supported languages.

The Visual Studio code editor also supports setting bookmarks in code for quick navigation. Other navigational aids include [collapsing code blocks](https://en.wikipedia.org/wiki/Code_folding) and [incremental search](https://en.wikipedia.org/wiki/Incremental_search), in addition to normal text search and [regex](https://en.wikipedia.org/wiki/Regular_expression) search. The code editor also includes a multi-item [clipboard](https://en.wikipedia.org/wiki/Clipboard) and a task list. The code editor supports code snippets, which are saved templates for repetitive code and can be inserted into code and customized for the project being worked on. A management tool for code snippets is built in as well. These tools are surfaced as floating windows which can be set to automatically hide when unused or docked to the side of the screen. The Visual Studio code editor also supports [code refactoring](https://en.wikipedia.org/wiki/Code_refactoring) including parameter reordering, variable and method renaming, [interface](https://en.wikipedia.org/wiki/Interface_(computing)) extraction and encapsulation of class members inside properties, among others.

Visual Studio features background compilation (also called incremental compilation). As code is being written, Visual Studio compiles it in the background in order to provide feedback about syntax and compilation errors, which are flagged with a red wavy underline. Warnings are marked with a green underline. Background compilation does not generate executable code, since it requires a different compiler than the one used to generate executable code.[[25]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-25) Background compilation was initially introduced with [Microsoft Visual Basic](https://en.wikipedia.org/wiki/Microsoft_Visual_Basic) but has now been expanded for all included languages.

### **Debugger**

Visual Studio includes a [debugger](https://en.wikipedia.org/wiki/Debugger) that works both as a source-level debugger and as a machine-level debugger. It works with both [managed code](https://en.wikipedia.org/wiki/Managed_code) as well as [native code](https://en.wikipedia.org/wiki/Native_code) and can be used for debugging applications written in any language supported by Visual Studio. In addition, it can also attach to running processes and monitor and debug those processes. If source code for the running process is available, it displays the code as it is being run. If source code is not available, it can show the [disassembly](https://en.wikipedia.org/wiki/Disassembly). The Visual Studio debugger can also create [memory dumps](https://en.wikipedia.org/wiki/Memory_dump) as well as load them later for debugging. Multi-threaded programs are also supported. The debugger can be configured to be launched when an application running outside the Visual Studio environment crashes.

The debugger allows setting [breakpoints](https://en.wikipedia.org/wiki/Breakpoint) (which allow execution to be stopped temporarily at a certain position) and watches (which monitor the values of variables as the execution progresses). Breakpoints can be conditional, meaning they get triggered when the condition is met. Code can be [stepped over](https://en.wikipedia.org/wiki/Program_animation), i.e., run one line (of source code) at a time. It can either *step into* functions to debug inside it, or *step over* it, i.e., the execution of the function body isn't available for manual inspection. The debugger supports*Edit and Continue*, i.e., it allows code to be edited as it is being debugged. When debugging, if the mouse pointer hovers over any variable, its current value is displayed in a tooltip ("data tooltips"), where it can also be modified if desired. During coding, the Visual Studio debugger lets certain functions be invoked manually from the Immediate tool window. The parameters to the method are supplied at the Immediate window.

### Designer

Visual Studio includes a host of visual designers to aid in the development of applications. These tools include:

**Windows Forms Designer**

The Windows Forms designer is used to build [GUI](https://en.wikipedia.org/wiki/GUI) applications using [Windows Forms](https://en.wikipedia.org/wiki/WinForms). Layout can be controlled by housing the controls inside other containers or locking them to the side of the form. Controls that display data (like textbox, list box, grid view, etc.) can be [bound](https://en.wikipedia.org/wiki/Data_binding) to data sources like [databases](https://en.wikipedia.org/wiki/Database) or [queries](https://en.wikipedia.org/wiki/LINQ). Data-bound controls can be created by dragging items from the Data Sources window onto a design surface.[[31]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-31) The UI is linked with code using an [event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming) model. The designer generates either [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) or [VB.NET](https://en.wikipedia.org/wiki/VB.NET) code for the application.

**WPF Designer**

The WPF designer, codenamed *Cider*,[[32]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-32) was introduced with Visual Studio 2008. Like the Windows Forms designer it supports the drag and drop metaphor. It is used to author [user interfaces](https://en.wikipedia.org/wiki/User_interface) targeting [Windows Presentation Foundation](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation). It supports all WPF functionality including [data binding](https://en.wikipedia.org/wiki/Data_binding) and automatic layout management. It generates[XAML](https://en.wikipedia.org/wiki/XAML) code for the UI. The generated [XAML](https://en.wikipedia.org/wiki/XAML) file is compatible with [Microsoft Expression Design](https://en.wikipedia.org/wiki/Microsoft_Expression_Design), the designer-oriented product. The XAML code is linked with code using a[code-behind](https://en.wikipedia.org/wiki/Code-behind) model.

**Web designer/development**

Visual Studio also includes a web-site editor and designer that allows web pages to be authored by dragging and dropping widgets. It is used for developing [ASP.NET](https://en.wikipedia.org/wiki/ASP.NET)applications and supports [HTML](https://en.wikipedia.org/wiki/HTML), [CSS](https://en.wikipedia.org/wiki/CSS) and [JavaScript](https://en.wikipedia.org/wiki/JavaScript). It uses a [code-behind](https://en.wikipedia.org/wiki/Code-behind) model to link with ASP.NET code. From Visual Studio 2008 onwards, the layout engine used by the web designer is shared with [Microsoft Expression Web](https://en.wikipedia.org/wiki/Microsoft_Expression_Web). There is also [ASP.NET MVC](https://en.wikipedia.org/wiki/ASP.NET_MVC) support for [MVC](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) technology as a separate download[[33]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-33) and [ASP.NET Dynamic Data](https://en.wikipedia.org/wiki/ASP.NET_Dynamic_Data)project available from Microsoft.[[34]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-34)

**Class designer**

The Class Designer is used to author and edit the classes (including its members and their access) using [UML](https://en.wikipedia.org/wiki/Unified_Modeling_Language) modeling. The Class Designer can generate [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) and [VB.NET](https://en.wikipedia.org/wiki/VB.NET)code outlines for the classes and methods. It can also generate class diagrams from hand-written classes.

**Data designer**

The data designer can be used to graphically edit [database schemas](https://en.wikipedia.org/wiki/Database_schema), including typed tables, primary and foreign keys and constraints. It can also be used to design queries from the graphical view.

**Mapping designer**

From Visual Studio 2008 onwards, the mapping designer is used by [LINQ to SQL](https://en.wikipedia.org/wiki/Language_Integrated_Query) to design the [mapping](https://en.wikipedia.org/wiki/Object-Relational_mapping) between [database schemas](https://en.wikipedia.org/wiki/Database_schema) and the [classes](https://en.wikipedia.org/wiki/Class_(computing)) that encapsulate the data. The new solution from ORM approach, [ADO.NET Entity Framework](https://en.wikipedia.org/wiki/ADO.NET_Entity_Framework), replaces and improves the old technology.

### Other tools**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Microsoft_Visual_Studio&action=edit&section=6)**]**

**Open Tabs Browser**

The open tabs browser is used to list all open tabs and to switch between them. It is invoked using CTRL+TAB.

**Properties Editor**

The *Properties Editor* tool is used to edit properties in a GUI pane inside Visual Studio. It lists all available properties (both read-only and those which can be set) for all objects including [classes](https://en.wikipedia.org/wiki/Class_(computing)), forms, web pages and other items.

**Object Browser**

The *Object Browser* is a [namespace](https://en.wikipedia.org/wiki/Namespace) and [class library](https://en.wikipedia.org/wiki/Class_library) browser for [Microsoft .NET](https://en.wikipedia.org/wiki/Microsoft_.NET). It can be used to browse the namespaces (which are arranged hierarchically) in [managed](https://en.wikipedia.org/wiki/Managed_code)[assemblies](https://en.wikipedia.org/wiki/Assembly_(CLI)). The hierarchy may or may not reflect the organization in the file system.

**Solution Explorer**

In Visual Studio parlance, a solution is a set of code files and other resources that are used to build an application. The files in a solution are arranged hierarchically, which might or might not reflect the organization in the file system. The *Solution Explorer* is used to manage and browse the files in a solution.

**Team Explorer**

*Team Explorer* is used to integrate the capabilities of [Team Foundation Server](https://en.wikipedia.org/wiki/Team_Foundation_Server), the [Revision Control System](https://en.wikipedia.org/wiki/Revision_Control_System) into the IDE (and the basis for Microsoft's [CodePlex](https://en.wikipedia.org/wiki/CodePlex) hosting environment for open source projects). In addition to source control it provides the ability to view and manage individual work items (including bugs, tasks and other documents) and to browse [TFS](https://en.wikipedia.org/wiki/Team_Foundation_Server) statistics. It is included as part of a TFS install and is also available as a download for Visual Studio separately. Team Explorer is also available as a stand-alone environment solely to access TFS services.

**Data Explorer**

*Data Explorer* is used to manage databases on [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) instances. It allows creation and alteration of database tables (either by issuing [T-SQL](https://en.wikipedia.org/wiki/T-SQL) commands or by using the Data designer). It can also be used to create [queries](https://en.wikipedia.org/wiki/Database_query) and [stored procedures](https://en.wikipedia.org/wiki/Stored_procedure), with the latter in either [T-SQL](https://en.wikipedia.org/wiki/T-SQL) or in [managed code](https://en.wikipedia.org/wiki/Managed_code) via [SQL CLR](https://en.wikipedia.org/wiki/SQL_CLR). Debugging and[IntelliSense](https://en.wikipedia.org/wiki/IntelliSense) support is available as well.

**Server Explorer**

The *Server Explorer* tool is used to manage database connections on an accessible computer. It is also used to browse running [Windows Services](https://en.wikipedia.org/wiki/Windows_Service), performance counters,[Windows Event Log](https://en.wikipedia.org/wiki/Windows_Event_Log) and [message queues](https://en.wikipedia.org/wiki/Message_queue) and use them as a datasource.

**Text Generation Framework**

Visual Studio includes a full text generation framework called [T4](https://en.wikipedia.org/wiki/Text_Template_Transformation_Toolkit) which enables Visual Studio to generate text files from templates either in the IDE or via code.

**ASP.NET Web Site Administration Tool**

The [ASP.NET Web Site Administration Tool](https://en.wikipedia.org/wiki/ASP.NET_Web_Site_Administration_Tool) allows for the configuration of [ASP.NET](https://en.wikipedia.org/wiki/ASP.NET) websites.

**Visual Studio Tools for Office**

[Visual Studio Tools for Office](https://en.wikipedia.org/wiki/Visual_Studio_Tools_for_Office) is a SDK and an add-in for Visual Studio that includes tools for developing for the [Microsoft Office](https://en.wikipedia.org/wiki/Microsoft_Office) suite. Previously (for Visual Studio .NET 2003 and Visual Studio 2005) it was a separate SKU that supported only [Visual C#](https://en.wikipedia.org/wiki/Visual_C_Sharp) and [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic_.NET) languages or was included in the Team Suite. With Visual Studio 2008, it is no longer a separate SKU but is included with Professional and higher editions. A separate runtime is required when deploying VSTO solutions.

### **Extensibility**

Visual Studio allows developers to write extensions for Visual Studio to extend its capabilities. These extensions "plug into" Visual Studio and extend its functionality. Extensions come in the form of *macros*, [*add-ins*](https://en.wikipedia.org/wiki/List_of_Microsoft_Visual_Studio_Add-ins), and *packages*. Macros represent repeatable tasks and actions that developers can record programmatically for saving, replaying, and distributing. Macros, however, cannot implement new commands or create tool windows. They are written using [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) and are not compiled.[[11]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-extpage1-11) Add-Ins provide access to the Visual Studio object model and can interact with the IDE tools. Add-Ins can be used to implement new functionality and can add new tool windows. Add-Ins are plugged into the IDE via [COM](https://en.wikipedia.org/wiki/Component_Object_Model) and can be created in any COM-compliant languages.[[11]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-extpage1-11) Packages are created using the Visual Studio [SDK](https://en.wikipedia.org/wiki/Software_Development_Kit) and provide the highest level of extensibility. They can create designers and other tools, as well as integrate other programming languages. The Visual Studio SDK provides unmanaged [APIs](https://en.wikipedia.org/wiki/API) as well as a managed API to accomplish these tasks. However, the managed API isn't as comprehensive as the unmanaged one.[[11]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-extpage1-11) Extensions are supported in the Standard (and higher) versions of Visual Studio 2005. [Express Editions](https://en.wikipedia.org/wiki/Visual_Studio_Express) do not support hosting extensions.

Visual Studio 2008 introduced the **Visual Studio Shell** that allows for development of a customized version of the IDE. The Visual Studio Shell defines a set of VSPackages that provide the functionality required in any IDE. On top of that, other packages can be added to customize the installation. The Isolated mode of the shell creates a new AppId where the packages are installed. These are to be started with a different executable. It is aimed for development of custom development environments, either for a specific language or a specific scenario. The Integrated mode installs the packages into the AppId of the Professional/Standard/Team System editions, so that the tools integrate into these editions.[[18]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-codefocus-18) The Visual Studio Shell is available as a free download.

After the release of Visual Studio 2008, Microsoft created the Visual Studio Gallery. It serves as the central location for posting information about extensions to Visual Studio. Community developers as well as commercial developers can upload information about their extensions to Visual Studio .NET 2002 through Visual Studio 2010. Users of the site can rate and review the extensions to help assess the quality of extensions being posted. RSS feeds to notify users on updates to the site and tagging features are also planned.[[40]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-40)

## Supported products[[edit](https://en.wikipedia.org/w/index.php?title=Microsoft_Visual_Studio&action=edit&section=8)]

[**Microsoft Visual C++**](https://en.wikipedia.org/wiki/Visual_C%2B%2B)

Microsoft Visual C++ is Microsoft's implementation of the [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B) [compiler](https://en.wikipedia.org/wiki/Compiler) and associated languages-services and specific tools for integration with the Visual Studio IDE. It can compile either in C mode or C++ mode. For C, it follows the [1990 version of the ISO C](https://en.wikipedia.org/wiki/C90_(C_version)) standard with parts of [C99](https://en.wikipedia.org/wiki/C99) specification along with MS-specific additions in the form of libraries.[[41]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-41) For C++, it follows the [ANSI C++](https://en.wikipedia.org/wiki/ANSI_C%2B%2B) specification along with a few [C++11](https://en.wikipedia.org/wiki/C%2B%2B11) features.[[42]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-42) It also supports the [C++/CLI](https://en.wikipedia.org/wiki/C%2B%2B/CLI) specification to write [managed code](https://en.wikipedia.org/wiki/Managed_code), as well as mixed-mode code (a mix of [native](https://en.wikipedia.org/wiki/Native_code) and [managed code](https://en.wikipedia.org/wiki/Managed_code)). Microsoft positions Visual C++ for development in native code or in code that contains both native as well as managed components. Visual C++ supports [COM](https://en.wikipedia.org/wiki/Component_Object_Model) as well as the [MFC](https://en.wikipedia.org/wiki/Microsoft_Foundation_Classes) library. For MFC development, it provides a set of wizards for creating and customizing MFC [boilerplate code](https://en.wikipedia.org/wiki/Boilerplate_code), and creating GUI applications using MFC. Visual C++ can also use the Visual Studio forms designer to design UI graphically. Visual C++ can also be used with the [Windows API](https://en.wikipedia.org/wiki/Windows_API). It also supports the use of *intrinsic functions*,[[43]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-43) which are functions recognized by the compiler itself and not implemented as a library. Intrinsic functions are used to expose the[SSE](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions) instruction set of modern CPUs. Visual C++ also includes the [OpenMP](https://en.wikipedia.org/wiki/OpenMP) (version 2.0) specification.[[44]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-44)

[**Microsoft Visual C#**](https://en.wikipedia.org/wiki/Visual_C_Sharp)

Microsoft Visual C#, Microsoft's implementation of the [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) language, targets the .NET Framework, along with the language services that lets the Visual Studio IDE support C# projects. While the language services are a part of Visual Studio, the compiler is available separately as a part of the .NET Framework. The Visual C# 2008, 2010 and 2012 compilers support versions 3.0, 4.0 and 5.0 of the C# language specifications, respectively. Visual C# supports the Visual Studio Class designer, Forms designer, and Data designer among others.[[45]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-45)

[**Microsoft Visual Basic**](https://en.wikipedia.org/wiki/Visual_Basic_.NET)

Microsoft Visual Basic is Microsoft's implementation of the [VB.NET](https://en.wikipedia.org/wiki/VB.NET) language and associated tools and language services. It was introduced with Visual Studio .NET (2002). Microsoft has positioned Visual Basic for [Rapid Application Development](https://en.wikipedia.org/wiki/Rapid_Application_Development).[[46]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-46)[[47]](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-47) Visual Basic can be used to author both console applications as well as GUI applications. Like Visual C#, Visual Basic also supports the Visual Studio Class designer, Forms designer, and Data designer among others. Like C#, the VB.NET compiler is also available as a part of .NET Framework, but the language services that let VB.NET projects be developed with Visual Studio, are available as a part of the latter.

**Microsoft Visual Web Developer**

Microsoft Visual Web Developer is used to create web sites, [web applications](https://en.wikipedia.org/wiki/Web_application) and [web services](https://en.wikipedia.org/wiki/Web_service) using ASP.NET. Either [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) or [VB.NET](https://en.wikipedia.org/wiki/VB.NET) languages can be used. Visual Web Developer can use the Visual Studio Web Designer to graphically design web page layouts.

Team Foundation Server is intended for collaborative [software development](https://en.wikipedia.org/wiki/Software_development) projects and acts as the server-side backend providing [source control](https://en.wikipedia.org/wiki/Source_control), data collection, [reporting](https://en.wikipedia.org/wiki/Report), and project-tracking functionality. It also includes the *Team Explorer*, the client tool for [TFS](https://en.wikipedia.org/wiki/Team_Foundation_Server) services, which is integrated inside Visual Studio Team System.

**d. Functional Requirements**

* Player must move without time lag.
* The control should trigger immediate movement in the game.
* Smooth transition from one level to another.
* The product must contain appropriate score recorder.
* Game should have more than one level of challenges.
* Game should have at least one player character**.**

**e. Non Functional Requirements**

The product shall take initial time to load, considering the high end elements in the same. It shall take little amount of initial download time and run time. The product load speed shall depend on the user’s system configuration and system’s space.

Game specific:

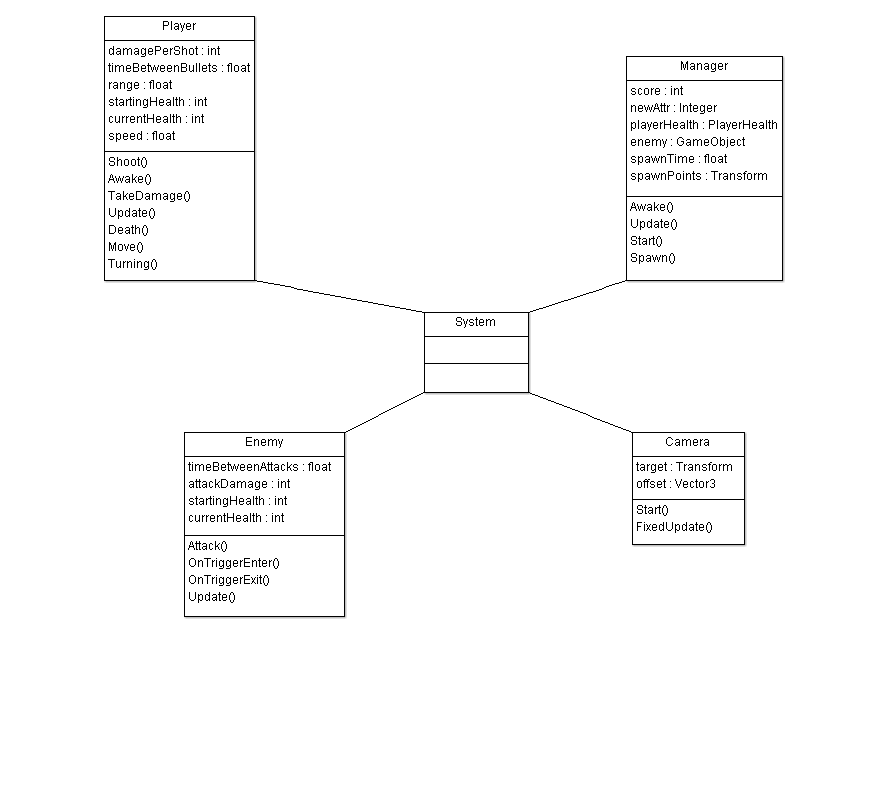
* Pause functionality shall be made available.
* Speed: Nominal and reasonable game speed and less buffer time.
* The game shall contain realistic 3D character.

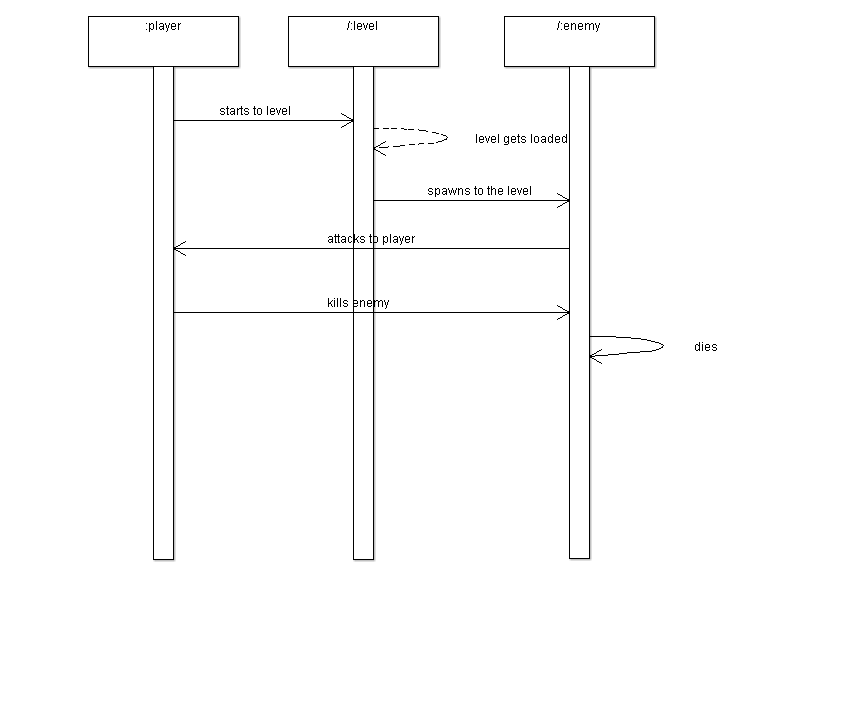
**4. Design**

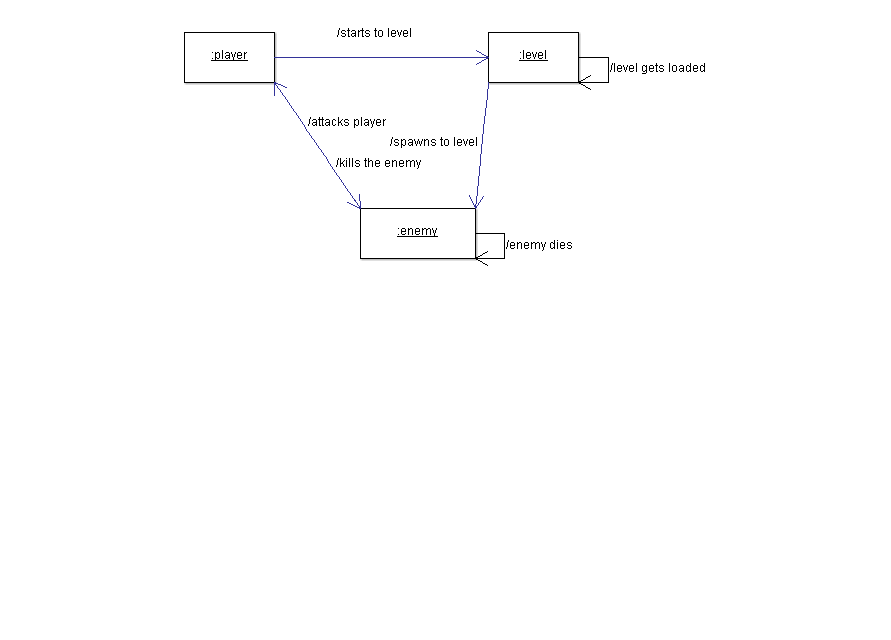
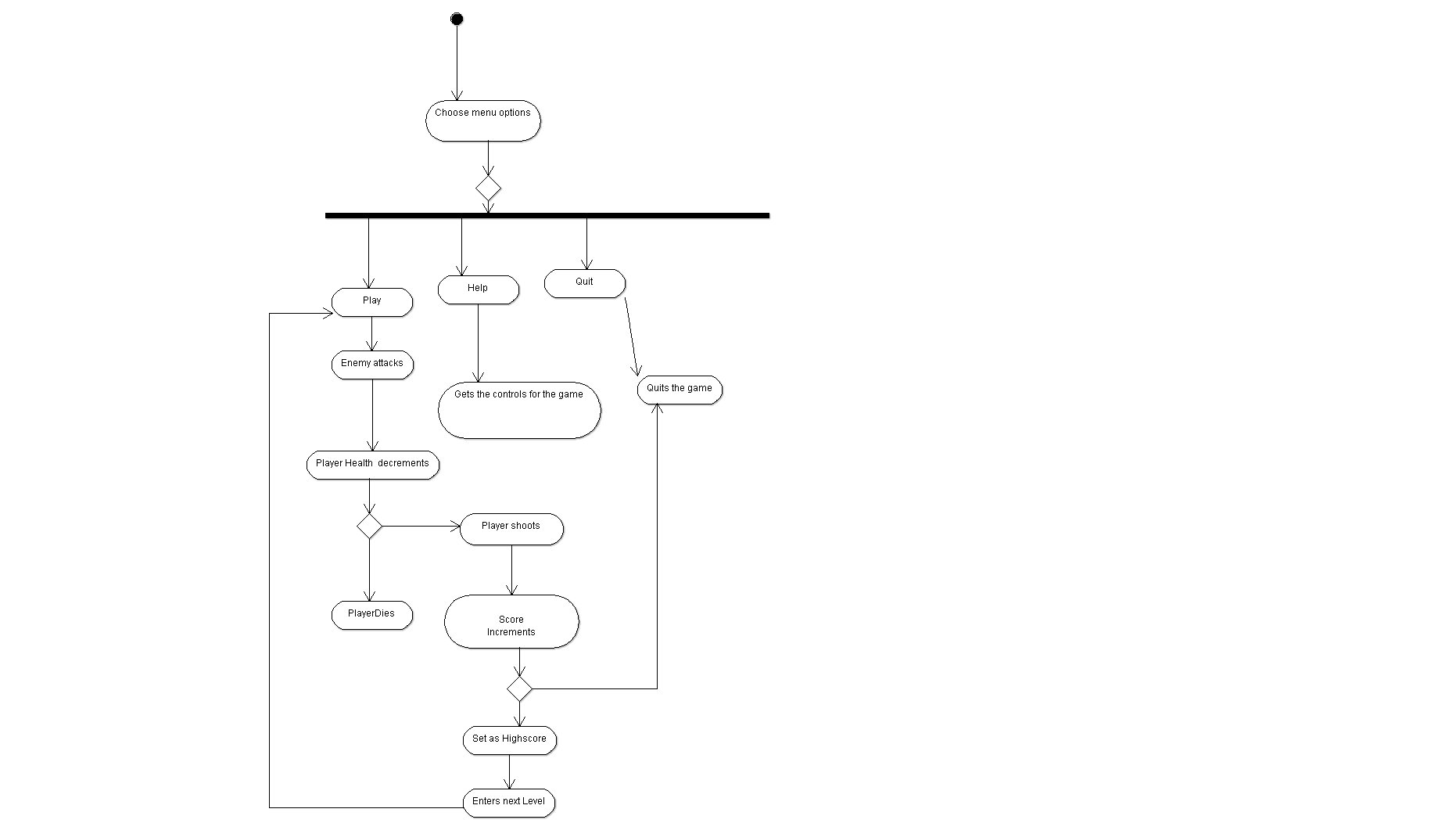
4.1 High level conceptual solution Architecture

4.2 UML Diagrams:

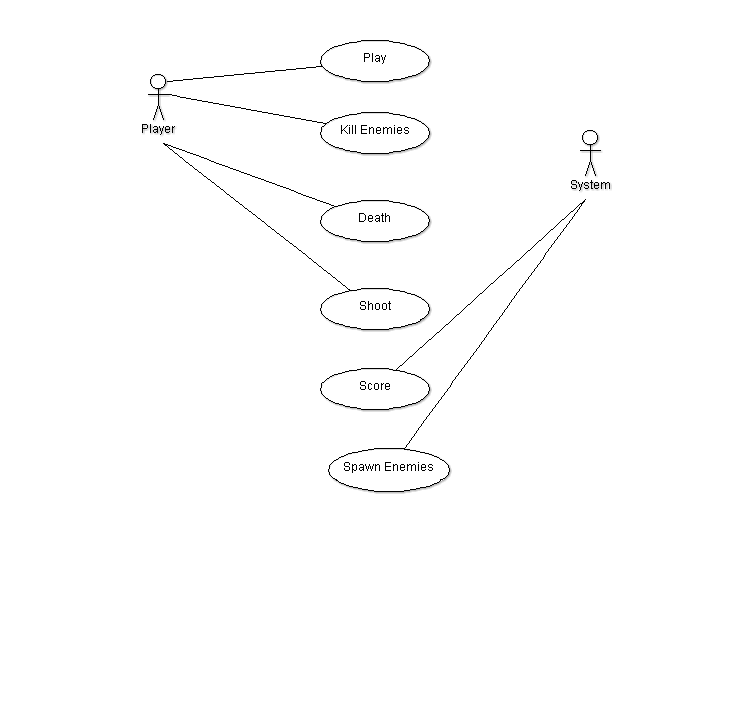
4.2.1 Class diagram



4.2.2 Sequence Diagram

4.2.3 Collaboration Diagram4.2.3 Activity diagram

4.2.4 Use case Diagram



5. IMPLEMENTATIONS

5.1 MODULE DESCRIPTIONS

**MODULES:**

# **Canvas**

The Canvas is the area that all UI elements should be inside. The Canvas is a Game Object with a Canvas component on it, and all UI elements must be children of such a Canvas.

Creating a new UI element, such as an Image using the menu GameObject > UI > Image, automatically creates a Canvas, if there isn’t already a Canvas in the scene. The UI element is created as a child to this Canvas.

The Canvas area is shown as a rectangle in the Scene View. This makes it easy to position UI elements without needing to have the Game View visible at all times.

Canvas uses the EventSystem object to help the Messaging System.

## Draw order of elements

UI elements in the Canvas are drawn in the same order they appear in the Hierarchy. The first child is drawn first, the second child next, and so on. If two UI elements overlap, the later one will appear on top of the earlier one.

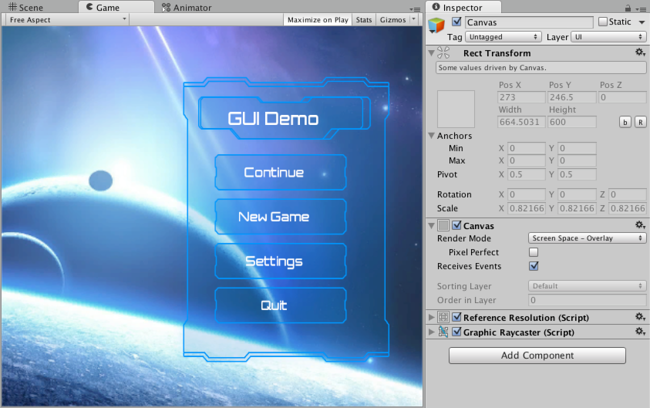
To change which element appear on top of other elements, simply reorder the elements in the Hierarchy by dragging them. The order can also be controlled from scripting by using these methods on the Transform component: SetAsFirstSibling, SetAsLastSibling, and SetSiblingIndex.

## Render Modes

The Canvas has a Render Mode setting which can be used to make it render in screen space or world space.

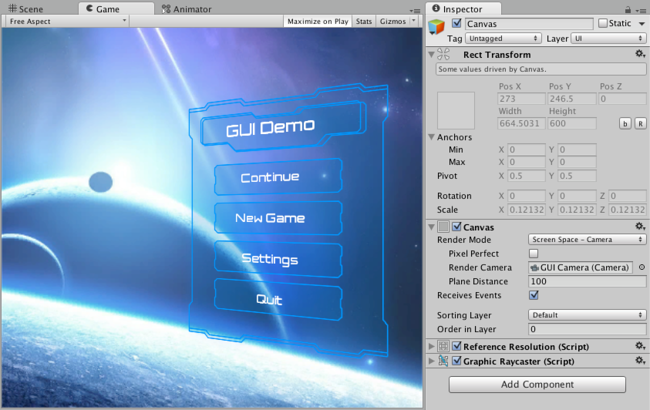
### Screen Space - Overlay

This render mode places UI elements on the screen rendered on top of the scene. If the screen is resized or changes resolution, the Canvas will automatically change size to match this.

UI in screen space overlay canvas

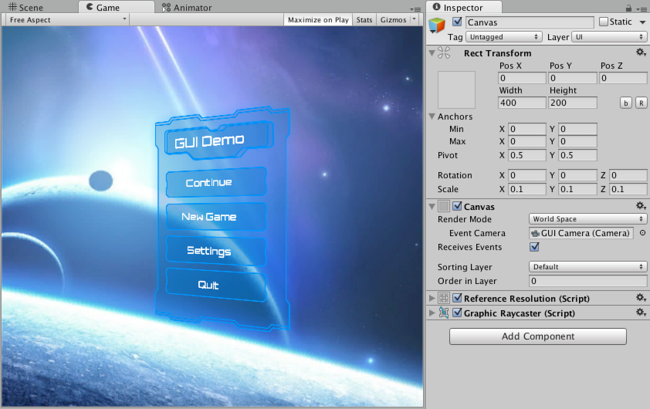
### Screen Space - Camera

This is similar to Screen Space - Overlay, but in this render mode, the Canvas is placed a given distance in front of a specified Camera. The UI elements are rendered by this camera, which means that the Camera settings affect the appearance of the UI. If the Camera is set to Perspective, the UI elements will be rendered with perspective, and the amount of perspective distortion can be controlled by the CameraField of View. If the screen is resized or changes resolution, or the camera frustrum changes, the Canvas will automatically change size to match as well.

UI in screen space camera canvas

### World Space

In this render mode, the Canvas will behave as any other object in the scene. The size of the Canvas can be set manually using its Rect Transform, and UI elements will render in front of or behind other objects in the scene based on 3D placement. This is useful for UIs that are meant to be a part of the world. This is also known as a “diegetic interface”.

UI in world space canvas

# Audio

A game would be incomplete without some kind of audio, be it background music or sound effects. Unity’s audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied. Unity can also record audio from any available microphone on a user’s machine for use during gameplay or for storage and transmission.

To simulate the effects of position, Unity requires sounds to originate from Audio Sources attached to objects. The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera. Unity can then simulate the effects of a source’s distance and position from the listener object and play them to the user accordingly. The relative speed of the source and listener objects can also be used to simulate the Doppler Effect for added realism.

Unity can’t calculate echoes purely from scene geometry but you can simulate them by adding Audio Filters to objects. For example, you could apply the Echo filter to a sound that is supposed to be coming from inside a cave. In situations where objects can move in and out of a place with a strong echo, you can add a Reverb Zone to the scene. For example, your game might involve cars driving through a tunnel. If you place a reverb zone inside the tunnel then the cars’ engine sounds will start to echo as they enter and the echo will die down as they emerge from the other side.

The Unity Audio Mixer allows you to mix various audio sources, apply effects to them, and perform mastering.

The manual pages for [Audio Source](http://docs.unity3d.com/Manual/class-AudioSource.html), [Audio Listener](http://docs.unity3d.com/Manual/class-AudioListener.html), [Audio Mixer](http://docs.unity3d.com/Manual/class-AudioMixer.html), the [audio effects](http://docs.unity3d.com/Manual/class-AudioEffect.html) and [Reverb Zones](http://docs.unity3d.com/Manual/class-AudioReverbZone.html) give more information about the many options and parameters available for getting effects just right.

## Working with Audio Assets

Unity can import audio files in **AIFF**, **WAV**, **MP3** and **Ogg** formats in the same way as other assets, simply by dragging the files into the Project panel. Importing an audio file creates an Audio Clip which can then be dragged to an Audio Source or used from a script. The Audio Clip reference page has more details about the import options available for audio files.

For music, Unity also supports tracker modules, which use short audio samples as “instruments” that are then arranged to play tunes. Tracker modules can be imported from **.xm**, **.mod**, **.it**, and **.s3m** files but are otherwise used in much the same way as ordinary audio clips.

## Audio Recording

Unity can access the computer’s microphones from a script and create Audio Clips by direct recording. The Microphone class provides a straightforward API to find available microphones, query their capabilities and start and end a recording session. The script reference page for [Microphone](http://docs.unity3d.com/ScriptReference/Microphone.html) has further information and code samples for audio recording.

# Audio Files

As with Meshes or Textures, the workflow for Audio File assets is designed to be smooth and trouble free. Unity can import almost every common file format but there are a few details that are useful to be aware of when working with Audio Files.

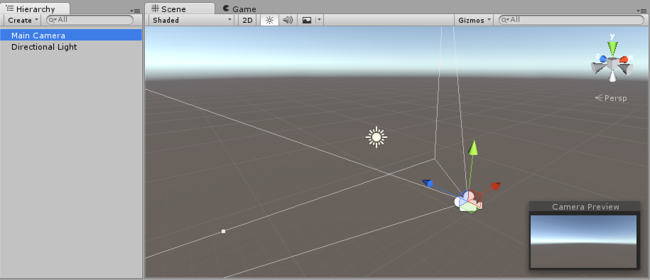
Since Unity 5.0 audio data is separated from the actual AudioClips. The AudioClips merely refer to the files containing the audio data and there are various combinations of options in the AudioClip importer that determine how the clips are loaded at runtime. This means that you have great flexibility for deciding which audio assets should be kept in memory at all times (because you may not be able to predict how often or how fast they will be playing, i.e. footsteps, weapons and impacts), while other assets may be loaded on demand or gradually as the player progresses through the level (speech, background music, ambience loops etc).

When audio is encoded in Unity the main options for how it is stored on disk is either PCM, ADPCM or Compressed. Additionally there are a few platform-specific formats, but they work in similar ways. Unity supports most common formats for importing audio (see the list below) and will import an audio file when it is added to the project. The default mode is Compressed, where the audio data is compressed with either Vorbis/MP3 for standalone and mobile platforms, or HEVAG/XMA for PS Vita / XBox One.

**SCENES:**

# Scenes

Scenes contain the objects of your game. They can be used to create a main menu, individual levels, and anything else. Think of each unique Scene file as a unique level. In each Scene, you will place your environments, obstacles, and decorations, essentially designing and building your game in pieces.

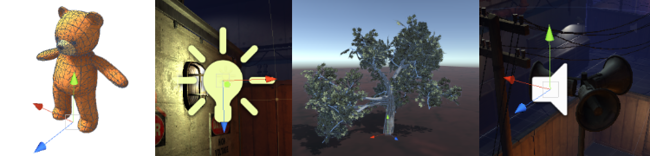
A new empty scene, with the default 3D objects - a camera and a directional light.

# GameObjects

The GameObject is the most important type of object in Unity. It is very important to understand what a GameObject is, and how it can be used.

## What are GameObjects?

Every object in your game is a GameObject. However, GameObjects don’t do anything on their own. They need special properties before they can become a character, an environment, or a special effect. But each of these objects do very different things. If every object is a GameObject, how do we differentiate an interactive power-up object from a static room? What makes these GameObjects different from each other?

Four different Game Objects, an animated character, a light, a tree and an audio source

The answer to this question is that GameObjects are containers. They can hold the different pieces that are required to make up a character, a light, a tree, a sound, or whatever else you would like to build. So to really understand GameObjects, you have to understand these pieces which are called **Components**.

Depending on what kind of object you want to create, you will add different combinations of Components to the GameObject. Think of a GameObject as an empty cooking pot, and Components as different ingredients that make up your recipe of gameplay. Unity has lots of different built-in component types, and you can also make your own Components using Scripts.

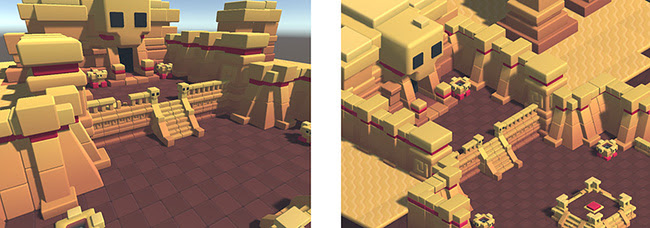
# Cameras

Just as cameras are used in films to display the story to the audience, Cameras in Unity are used to display the game world to the player. You will always have at least one camera in a scene, but you can have more than one. Multiple cameras can give you a two-player splitscreen or create advanced custom effects. You can animate cameras, or control them with physics. Practically anything you can imagine is possible with cameras, and you can use typical or unique cameras to fit your game’s style.

A Unity scene is created by arranging and moving objects in a three-dimensional space. Since the viewer’s screen is two-dimensional, there needs to be a way to capture a view and “flatten” it for display. This is accomplished using Cameras.

A camera is an object that defines a view in scene space. The object’s position defines the viewpoint, while the forward (Z) and upward (Y) axes of the object define the view direction and the top of the screen, respectively. The [Camera component](http://docs.unity3d.com/Manual/class-Camera.html) also defines the size and shape of the region that falls within the view. With these parameters set up, the camera can display what it currently “sees” to the screen. As the camera object moves and rotates, the displayed view will also move and rotate accordingly.

## Perspective and orthographic cameras

The same scene shown in perspective mode (left) and orthographic mode (right)

A camera in the real world, or indeed a human eye, sees the world in a way that makes objects look smaller the farther they are from the point of view. This well-known perspective effect is widely used in art and computer graphics and is important for creating a realistic scene. Naturally, Unity supports perspective cameras, but for some purposes, you want to render the view without this effect. For example, you might want to create a map or information display that is not supposed to appear exactly like a real-world object. A camera that does not diminish the size of objects with distance is referred to as orthographic and Unity cameras also have an option for this. The perspective and orthographic modes of viewing a scene are known as camera projections. (scene above from [*BITGEM*](https://www.assetstore.unity3d.com/en/#!/publisher/1299))

## The shape of the viewed region

Both perspective and orthographic cameras have a limit on how far they can “see” from their current position. The limit is defined by a plane that is perpendicular to the camera’s forward (Z) direction. This is known as the far clipping plane since objects at a greater distance from the camera are “clipped” (ie, excluded from rendering). There is also a corresponding near clipping plane close to the camera - the viewable range of distance is that between the two planes.

Without perspective, objects appear the same size regardless of their distance. This means that the viewing volume of an orthographic camera is defined by a rectangular box extending between the two clipping planes.

When perspective is used, objects appear to diminish in size as the distance from camera increases. This means that the width and height of the viewable part of the scene grows with increasing distance. The viewing volume of a perspective camera, then, is not a box but a pyramidal shape with the apex at the camera’s position and the base at the far clipping plane. The shape is not exactly a pyramid, however, because the top is cut off by the near clipping plane; this kind of truncated pyramid shape is known as a frustum. Since its height is not constant, the frustum is defined by the ratio of its width to its height (known as the aspect ratio) and the angle between the top and bottom at the apex (known as the field of view of FOV). See the page about [understanding the view frustum](http://docs.unity3d.com/Manual/UnderstandingFrustum.html) for a more detailed explanation.

## The background to the camera view

For indoor scenes, the camera may always be completely inside some object representing the interior of a building, cave or other structure. When the action takes place outdoors, however, there will be many empty areas in between objects that are filled with nothing at all; these background areas typically represent the sky, space or the murky depths of an underwater scene.

A camera can’t leave the background completely undecided and so it must fill in the empty space with something. The simplest option is to clear the background to a flat color before rendering the scene on top of it. You can set this color using the camera’s Background property, either from the inspector or from a script. A more sophisticated approach that works well with outdoor scenes is to use a [Skybox](http://docs.unity3d.com/Manual/class-Skybox.html). As its name suggests, a skybox behaves like a “box” lined with images of a sky. The camera is effectively placed at the center of this box and can see the sky from all directions. The camera sees a different area of sky as it rotates but it never moves from the center (so the camera cannot get “closer” to the sky). The skybox is rendered behind all objects in the scene and so it represents a view at infinite distance. The most common usage is to represent the sky in a standard outdoor scene but the box actually surrounds the camera completely, even underneath. This means that you can use a skybox to represent parts of the scene (eg, rolling plains that stretch beyond the horizon) or the all-round view of a scene in space or underwater.

# The Animator Controller Asset

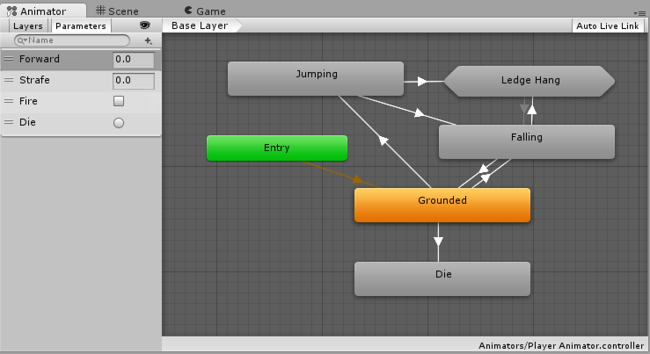
When you have an animation clips ready to use, you need to use an Animator Controller to bring them together. An Animator Controller asset is created within Unity and allows you to maintain a set of animations for a character or object.

An Animator Controller Asset in the Project Folder

Animator Controller assets are created from the Assets menu, or from the Create menu in the Project window.

In most situations, it is normal to have multiple animations and switch between them when certain game conditions occur. For example, you could switch from a walk animation to a jump whenever the spacebar is pressed. However even if you just have a single animation clip you still need to place it into an animator controller to use it on a Game Object.

The controller manages the various animation states and the transitions between them using a so-called State Machine, which could be thought of as a kind of flow-chart, or a simple program written in a visual programming language within Unity. More information about state machines can be found [here](http://docs.unity3d.com/Manual/AnimationStateMachines.html). The structure of the Animator Controller can be created, viewed and modified in the Animator Window.

A simple Animator Controller

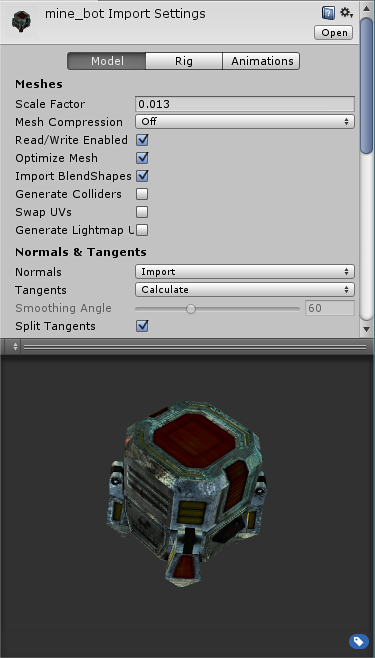
The animator controller is finally applied to an object by attaching an Animator component that references them.

Player and other models:

Model files that are placed in the Assets folder in your Unity project are automatically imported and stored as Unity assets.

A model file may contain a 3D model, such as a character, a building, or a piece of furniture. The model is imported as multiple assets. In the Project view the main imported object is a Model Prefab. Usually there are also up to several Mesh objects that are referenced by the Model Prefab.

A model file may also contain animation data which can be used to animate this model or other models. The animation data is imported as one or more Animation Clips.



A [Mesh Filter](http://docs.unity3d.com/Manual/class-MeshFilter.html) together with [Mesh Renderer](http://docs.unity3d.com/Manual/class-MeshRenderer.html) makes the model appear on screen.

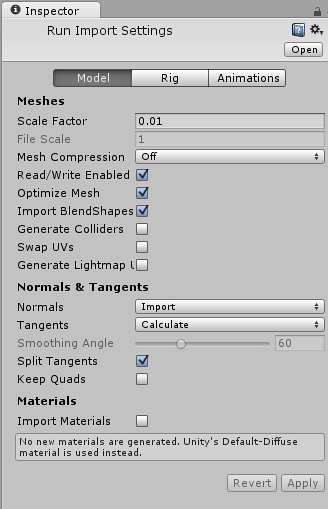
# Import settings for Meshes

The Import Settings for a model file will be displayed in the Model tab of the FBX importer inspector when the model is selected. These affect the **mesh**, it’s **normals** and imported **materials**. Settings are applied per asset on disk so if you need assets with different settings make (and rename accordingly) a duplicate file.

Although defaults can suffice initially, it is worth studying the settings glossary below, as they can determine what you wish to do with the game object.

Some general adjustments to be made for example might be:

* Scale - this scale factor is used for compensating difference in units between Unity and 3d modeling tool - it rescales whole file. Normally you can simply set it to 1. Note that Unity’s Physics Engine is scaled as 1 unit equals 1 metre. It is important that if you want to have correct physical behaviour you should have the model correctly scaled in the original modeling application. If this cannot be done, or you do not have control over the modification of the mesh, the scale of the model can be adjusted here.
* Generate colliders - this will generate a collison mesh to allow your model to collide with other objects.
* Material Naming and Search - this will help you automatically setup your materials and locate textures



FBX Importer Inspector: Model tab

**HEALTH:**

**Health** is an [attribute](https://en.wikipedia.org/wiki/Attribute_(role-playing_games)) assigned to entities within a [role-playing](https://en.wikipedia.org/wiki/Role-playing_game) or [video game](https://en.wikipedia.org/wiki/Video_game) that indicates its state in combat.[[1]](https://en.wikipedia.org/wiki/Health_(gaming)#cite_note-Basics-1) Health is usually measured in **hit points** or **health points**, often shortened as **HP**. When the HP of a [player character](https://en.wikipedia.org/wiki/Player_character) reaches zero, the player may lose a [life](https://en.wikipedia.org/wiki/Life_(gaming)) or their character might become incapacitated or die. When the HP of an [enemy](https://en.wikipedia.org/wiki/Mob_(gaming)) reaches zero, the player might be rewarded in some way.

Any entity within a game could have a health value, including the player character, [non-player characters](https://en.wikipedia.org/wiki/Non-player_character) and objects. Indestructible entities have no diminishable health value.

Health might be displayed as a numeric value, such as "50/100". Here, the first number indicates the current amount of HP an entity has and the second number indicates the entity's maximum HP. In video games, health can also be displayed graphically, such as with a bar that empties itself when an entity loses health (a **health bar**), icons that are "chipped away" from, or in more novel ways.

**RIGID BODIES:**

Rigidbodies allow your GameObjects to act under control of the physics engine. This opens the gateway to realistic collisions, varied types of joints, and other very cool behaviors. Manipulating your GameObjects by adding forces to a Rigidbody creates a very different feel and look than adjusting the Transform Component directly. Generally, you shouldn’t manipulate the Rigidbody and the Transform of the same GameObject - only one or the other.

The biggest difference between manipulating the Transform versus the Rigidbody is the use of forces. Rigidbodies can receive forces and torque, but Transforms cannot. Transforms can be translated and rotated, but this is not the same as using physics. You’ll notice the distinct difference when you try it for yourself. Adding forces/torque to the Rigidbody will actually change the object’s position and rotation of the Transform component. This is why you should only be using one or the other. Changing the Transform while using physics could cause problems with collisions and other calculations.

Rigidbodies must be explicitly added to your GameObject before they will be affected by the physics engine. You can add a Rigidbody to your selected object from Components->Physics->Rigidbody in the menu. Now your object is physics-ready; it will fall under gravity and can receive forces via scripting, but you may need to add a Collider or a Joint to get it to behave exactly how you want.

### Parenting

When an object is under physics control, it moves semi-independently of the way its transform parents move. If you move any parents, they will pull the Rigidbody child along with them. However, the Rigidbodies will still fall down due to gravity and react to collision events.

### Scripting

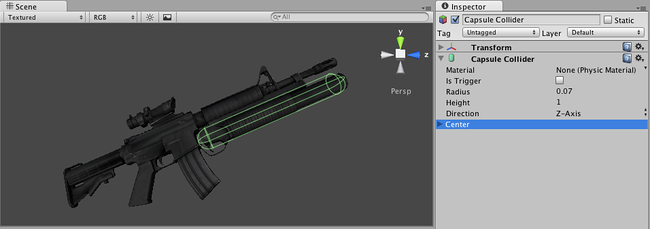
To control your Rigidbodies, you will primarily use scripts to add forces or torque. You do this by calling [AddForce()](http://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html) and [AddTorque()](http://docs.unity3d.com/ScriptReference/Rigidbody.AddTorque.html) on the object’s Rigidbody. Remember that you shouldn’t be directly altering the object’s Transform when you are using physics.

### Animation

For some situations, mainly creating ragdoll effects, it is neccessary to switch control of the object between animations and physics. For this purpose Rigidbodies can be marked [isKinematic](http://docs.unity3d.com/ScriptReference/Rigidbody-isKinematic.html). While the Rigidbody is marked isKinematic, it will not be affected by collisions, forces, or any other part of the physics system. This means that you will have to control the object by manipulating the [Transform](http://docs.unity3d.com/Manual/class-Transform.html) component directly. Kinematic Rigidbodies will affect other objects, but they themselves will not be affected by physics. For example, Joints which are attached to Kinematic objects will constrain any other Rigidbodies attached to them and Kinematic Rigidbodies will affect other Rigidbodies through collisions.

### Colliders

Colliders are another kind of component that must be added alongside the Rigidbody in order to allow collisions to occur. If two Rigidbodies bump into each other, the physics engine will not calculate a collision unless both objects also have a Collider attached. Collider-less Rigidbodies will simply pass through each other during physics simulation.

Colliders define the physical boundaries of a Rigidbody

LIGHTS

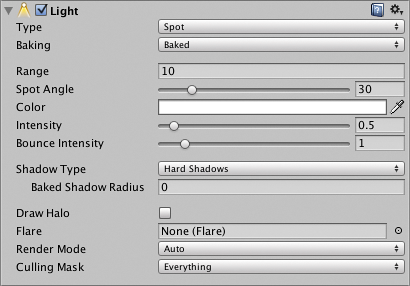
# Lights

Lights are an essential part of every scene. While meshes and textures define the shape and look of a scene, lights define the color and mood of your 3D environment. You’ll likely work with more than one light in each scene. Making them work together requires a little practice but the results can be quite amazing.

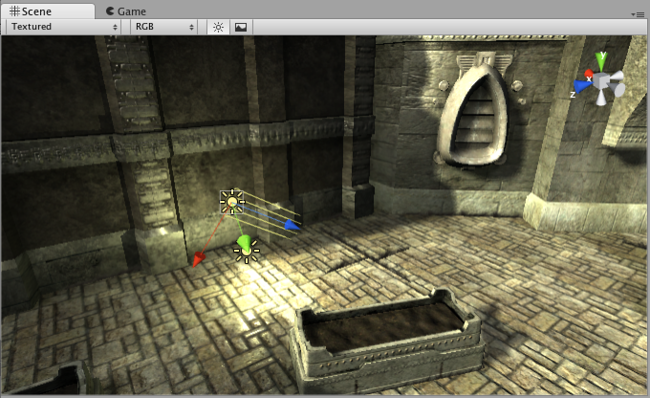
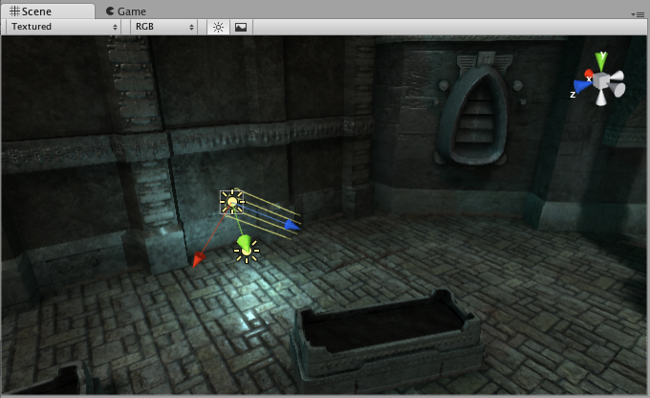
A simple, two-light setup

Lights can be added to your scene from the GameObject->Light menu. You will choose the light format that you want from the sub-menu that appears. Once a light has been added, you can manipulate it like any other GameObject. Additionally, you can add a Light Component to any selected GameObject by using Component->Rendering->Light.

There are many different options within the Light Component in the Inspector.

Light Component properties in the Inspector

By simply changing the Color of a light, you can give a whole different mood to the scene.

Bright, sunny lightsDark, medieval lightsSpooky night lights

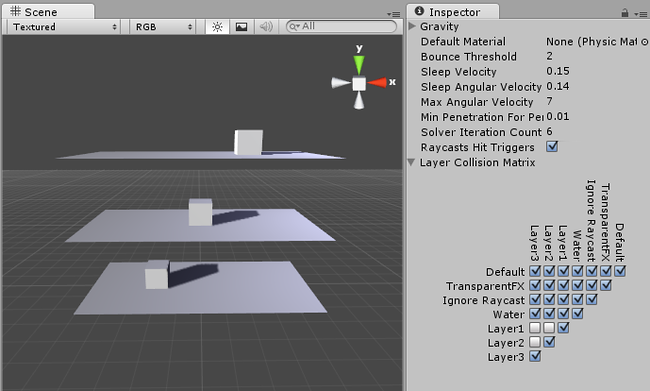
## Rendering paths

Unity supports different Rendering Paths. These paths affect mainly Lights and Shadows, so choosing the correct rendering path depending on your game requirements can improve your project’s performance.

**Layers:**

Layers are most commonly used by Cameras to render only a part of the scene, and by Lights to illuminate only parts of the scene. But they can also be used by raycasting to selectively ignore colliders or to create [collisions](http://docs.unity3d.com/Manual/LayerBasedCollision.html).

Layer-Based collision detection is a way to make Game Objects collide with another specific Game Objects that are tied up to specific layers.

Objects Colliding with their own layer.

In the image above you can see 6 GameObjects, (3 planes, 3 cubes) and the “Collision Matrix” to the right that states which Objects can collide with which layer.

**5.2 PROGRAMS/ALGORITHMS**

CAMERA FOLLOW:

using UnityEngine;

using System.Collections;

public class CameraFollow : MonoBehaviour {

    public Transform target;

    public float smoothing = 5f;

    Vector3 offset;

    void Start()

    {

        offset = transform.position - target.position;

    }

    void FixedUpdate()

    {

        Vector3 targetCamPosition = target.position + offset;

        transform.position = Vector3.Lerp(transform.position, targetCamPosition, smoothing \* Time.deltaTime); // Move camera smoothly

    }

}

CAMERA MOVE:

var rSpeed = 3.0;

var mSpeed = 20.0;

var X = 0.0;

var Y = 0.0;

function Update (){

    X += Input.GetAxis("Mouse X")\*rSpeed;

    Y += Input.GetAxis("Mouse Y")\*rSpeed;

    transform.localRotation = Quaternion.AngleAxis(X, Vector3.up);

    transform.localRotation \*= Quaternion.AngleAxis(Y, Vector3.left);

    transform.position += transform.forward\*mSpeed\*Input.GetAxis("Vertical")\*Time.deltaTime;

    transform.position += transform.right\*mSpeed\*Input.GetAxis("Horizontal")\*Time.deltaTime;

}

PLAYER MOVEMENT:

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

    public float speed = 6f;

    Vector3 movement;

    private Animator anim;

    Rigidbody playerRigidbody;

    int floorMask;

    float camRayLength = 100f;

    void Awake()

    {

        floorMask = LayerMask.GetMask("Floor");

        anim = GetComponent<Animator>();

        playerRigidbody = GetComponent<Rigidbody>();

    }

    void FixedUpdate()

    {

        float h = Input.GetAxisRaw("Horizontal");

        float v = Input.GetAxisRaw("Vertical");

        Move(h, v);

        Turning();

        Animating(h, v);

    }

PLAYER SHOOTING:

using UnityEngine;

public class PlayerShooting : MonoBehaviour

{

    public int damagePerShot = 20;

    public float timeBetweenBullets = 0.15f;

    public float range = 100f;

    Animator anim;

    float timer;

    Ray shootRay;

    RaycastHit shootHit;

    int shootableMask;

    ParticleSystem gunParticles;

    LineRenderer gunLine;

    AudioSource gunAudio;

    Light gunLight;

    float effectsDisplayTime = 0.2f;

    void Awake ()

    {

        shootableMask = LayerMask.GetMask ("Shootable");

        gunParticles = GetComponent<ParticleSystem> ();

        gunLine = GetComponent <LineRenderer> ();

        gunAudio = GetComponent<AudioSource> ();

        gunLight = GetComponent<Light> ();

    }

    void Update ()

    {

        timer += Time.deltaTime;

if((Input.GetButton ("Fire1")||Input.GetKey(KeyCode.Space)) && timer >= timeBetweenBullets && Time.timeScale != 0)

        {

            Shoot ();

        }

        if(timer >= timeBetweenBullets \* effectsDisplayTime)

        {

            DisableEffects ();

        }

    }

    public void DisableEffects ()

    {

        gunLine.enabled = false;

        gunLight.enabled = false;

    }

    void Shoot ()

    {

        timer = 0f;

        gunAudio.Play ();

        gunLight.enabled = true;

        gunParticles.Stop ();

        gunParticles.Play ();

        gunLine.enabled = true;

        gunLine.SetPosition (0, transform.position);

        shootRay.origin = transform.position;

        shootRay.direction = transform.forward;

        if(Physics.Raycast (shootRay, out shootHit, range, shootableMask))

        {

            EnemyHealth enemyHealth = shootHit.collider.GetComponent <EnemyHealth> ();

            if(enemyHealth != null)

            {

                enemyHealth.TakeDamage (damagePerShot, shootHit.point);

            }

            gunLine.SetPosition (1, shootHit.point);

        }

        else

        {

            gunLine.SetPosition (1, shootRay.origin + shootRay.direction \* range);

        }

    }

}

PLAYER HEALTH:

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using UnityEngine.SceneManagement;

public class PlayerHealth : MonoBehaviour

{

    public int startingHealth = 100;

    public int currentHealth;

    public Slider healthSlider;

    public Image damageImage;

    public AudioClip deathClip;

    public float flashSpeed = 5f;

    public Color flashColour = new Color(1f, 0f, 0f, 0.1f);

    Animator anim;

    AudioSource playerAudio;

    PlayerMovement playerMovement;

    PlayerShooting playerShooting;

    bool isDead;

    bool damaged;

    void Awake()

    {

        anim = GetComponent<Animator>();

        playerAudio = GetComponent<AudioSource>();

        playerMovement = GetComponent<PlayerMovement>();

        playerShooting = GetComponentInChildren<PlayerShooting>();

        currentHealth = startingHealth;

    }

    void Update()

    {

        if (damaged)

        {

            damageImage.color = flashColour;

        }

        else

        {

            damageImage.color = Color.Lerp(damageImage.color, Color.clear, flashSpeed \* Time.deltaTime);

        }

        damaged = false;

    }

    public void TakeDamage(int amount)

    {

        damaged = true;

        currentHealth -= amount;

        healthSlider.value = currentHealth;

        playerAudio.Play();

        if (currentHealth <= 0 && !isDead)

        {

            Death();

        }

    }

    void Death()

    {

        isDead = true;

        playerShooting.DisableEffects();

        anim.SetTrigger("Die");

        playerAudio.clip = deathClip;

        playerAudio.Play();

        playerMovement.enabled = false;

        playerShooting.enabled = false;

    }

}

ENEMY MOVEMENT:

using UnityEngine;

using System.Collections;

public class EnemyMovement : MonoBehaviour

{

    Transform player;

    PlayerHealth playerHealth;

    EnemyHealth enemyHealth;

    NavMeshAgent nav;

    void Awake()

    {

        player = GameObject.FindGameObjectWithTag("Player").transform;

        playerHealth = player.GetComponent<PlayerHealth>();

        enemyHealth = GetComponent<EnemyHealth>();

        nav = GetComponent<NavMeshAgent>();

    }

    void Update()

    {

        if (enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)

        {

            nav.SetDestination(player.position);

        }

        else

        {

            nav.enabled = false;

        }

    }

}

ENEMY ATTACK:

using UnityEngine;

using System.Collections;

public class EnemyAttack : MonoBehaviour

{

    public float timeBetweenAttacks = 0.5f;

    public int attackDamage = 10;

    Animator anim;

    GameObject player;

    PlayerHealth playerHealth;

    EnemyHealth enemyHealth;

    bool playerInRange;

    float timer;

    void Awake ()

    {

        player = GameObject.FindGameObjectWithTag ("Player");

        playerHealth = player.GetComponent <PlayerHealth> ();

        enemyHealth = GetComponent<EnemyHealth>();

        anim = GetComponent <Animator> ();

    }

    void OnTriggerEnter (Collider other)

    {

        if(other.gameObject == player)

        {

            playerInRange = true;

        }

    }

    void OnTriggerExit (Collider other)

    {

        if(other.gameObject == player)

        {

            playerInRange = false;

        }

    }

    void Update ()

    {

        timer += Time.deltaTime;

        if(timer >= timeBetweenAttacks && playerInRange && enemyHealth.currentHealth > 0)

        {

            //anim.SetTrigger("Attacking");

            Attack ();

        }

        if(playerHealth.currentHealth <= 0)

        {

            anim.SetTrigger ("PlayerDead");

        }

    }

    void Attack ()

    {

        timer = 0f;

        if(playerHealth.currentHealth > 0)

        {

            playerHealth.TakeDamage (attackDamage);

        }

    }

}

ENEMY HEALTH:

using UnityEngine;

public class EnemyHealth : MonoBehaviour

{

    public int startingHealth = 100;

    public int currentHealth;

    public float sinkSpeed = 2.5f;

    public int scoreValue = 10;

    public AudioClip deathClip;

    Animator anim;

    AudioSource enemyAudio;

    ParticleSystem hitParticles;

    CapsuleCollider capsuleCollider;

    bool isDead;

    bool isSinking;

    void Awake ()

    {

        anim = GetComponent <Animator> ();

        enemyAudio = GetComponent <AudioSource> ();

        hitParticles = GetComponentInChildren <ParticleSystem> ();

        capsuleCollider = GetComponent <CapsuleCollider> ();

        currentHealth = startingHealth;

    }

    void Update ()

    {

        if(isSinking)

        {

            transform.Translate (-Vector3.up \* sinkSpeed \* Time.deltaTime);

        }

    }

    public void TakeDamage (int amount, Vector3 hitPoint)

    {

        if(isDead)

            return;

        enemyAudio.Play ();

        currentHealth -= amount;

        hitParticles.transform.position = hitPoint;

        hitParticles.Play();

        if(currentHealth <= 0)

        {

            Death ();

        }

    }

    void Death ()

    {

        isDead = true;

        capsuleCollider.isTrigger = true;

        anim.SetTrigger ("Dead");

        enemyAudio.clip = deathClip;

        enemyAudio.Play ();

    }

    public void StartSinking ()

    {

        GetComponent <NavMeshAgent> ().enabled = false;

        GetComponent <Rigidbody> ().isKinematic = true;

        isSinking = true;

        ScoreManager.score += scoreValue;

        Destroy (gameObject, 2f);

    }

}

GAME MANAGER:

using System.Collections;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class GameManager : MonoBehaviour

{

    public PlayerHealth playerHealth;

    public float levelStartDelay = 2f;

    public float restartDelay = 5f;

    public float levelOneMaxScore = 50;

    public float levelTwoMaxScore = 100;

    public float levelThreeMaxScore = 150;

    public Text levelText;

    public int level = 1;

    public GameObject levelImage;

    private bool doingSetUp;

    Animator anim;

    private float restartTimer = 0;

    private float levelStartTimer = 0;

    void Awake()

    {

        anim = GetComponent<Animator>();

    }

    private void OnLevelWasLoaded(int index)

    {

        InitGame();

    }

    void InitGame()

    {

        doingSetUp = true;

        // levelImage = GameObject.Find("LevelImage");

        // levelText = GameObject.Find("LevelText").GetComponent<Text>();

        levelText.text = "Level 0" + level;

        levelImage.SetActive(true);

        Invoke("HideLevelImage", levelStartDelay);

    }

    private void HideLevelImage()

    {

        levelImage.SetActive(false);

        doingSetUp = false;

    }

    void Update()

    {

        SaveScores();

        if (playerHealth.currentHealth <= 0)

        {

            PlayerPrefs.DeleteKey("Score");

            anim.SetTrigger("GameOver");

            restartTimer += Time.deltaTime;

            if(restartTimer >= restartDelay)

            {

                SceneManager.LoadScene(0);

            }

        }

        if (doingSetUp)

        {

            return;

        }

        if (ScoreManager.score >= levelOneMaxScore)

        {

            if (SceneManager.GetActiveScene().name == "Level 01")

            {

                levelStartTimer += Time.deltaTime;

                if(levelStartTimer >= levelStartDelay)

                {

                    StartCoroutine(LevelChange());

                    SceneManager.LoadScene("Level 02");

                }

            }

        }

        if (ScoreManager.score >= levelTwoMaxScore)

        {

            if (SceneManager.GetActiveScene().name == "Level 02")

            {

                levelStartTimer += Time.deltaTime;

                if (levelStartTimer >= levelStartDelay)

                {

                    StartCoroutine(LevelChange());

                    SceneManager.LoadScene("Level 03");

                }

            }

        }

        if(ScoreManager.score >= levelThreeMaxScore)

        {

            // Player wins

            SceneManager.LoadScene(4);

        }

    }

    void SaveScores()

    {

        PlayerPrefs.SetInt("HighScore", HighScoreManager.highScore);

        PlayerPrefs.SetInt("Score", ScoreManager.score);

        if (PlayerPrefs.GetInt("Score") > PlayerPrefs.GetInt("HighScore"))

        {

            HighScoreManager.highScore = ScoreManager.score;

            PlayerPrefs.SetInt("HighScore", HighScoreManager.highScore);

        }

    }

    IEnumerator LevelChange()

    {

        float fadeTime = GameObject.Find("HUDCanvas").GetComponent<Fading>().BeginFade(1);

        yield return new WaitForSeconds(fadeTime);

    }

}

using UnityEngine;

public class EnemyManager : MonoBehaviour

{

    public PlayerHealth playerHealth;

    public GameObject enemy;

    public float spawnTime = 3f;

    public Transform[] spawnPoints;

    ENEMY MANAGER:

    void Start ()

    {

        InvokeRepeating ("Spawn", spawnTime, spawnTime);

    }

    void Spawn ()

    {

        if(playerHealth.currentHealth <= 0f)

        {

            return;

        }

        int spawnPointIndex = Random.Range (0, spawnPoints.Length);

        Instantiate (enemy, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation);

    }

}

SCORE MANAGER:

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using UnityEngine.SceneManagement;

public class ScoreManager : MonoBehaviour

{

    public static int score;

    Text text;

    void Awake ()

    {

        text = GetComponent <Text> ();

        score = 0;

    }

    void Start()

    {

        if (SceneManager.GetActiveScene().name == "Level 01")

        {

            score = 0;

        }

        else

        {

            score = PlayerPrefs.GetInt("Score");

        }

    }

    void Update ()

    {

        text.text = "Score: " + score;

    }

}

MENU:

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

using System.Collections;

public class MenuScript : MonoBehaviour {

    public Canvas quitMenu;

    public Canvas controlsCanvas;

    public Button startButton;

    public Button quitButton;

    public Button controlsButton;

// Use this for initialization

void Start () {

        quitMenu = quitMenu.GetComponent<Canvas> ();

        controlsCanvas = controlsCanvas.GetComponent<Canvas>();

        startButton = startButton.GetComponent<Button>();

        quitButton = quitButton.GetComponent<Button>();

        controlsButton = controlsButton.GetComponent<Button>();

        quitMenu.enabled = false;

        controlsCanvas.enabled = false;

    }

public void ExitPress()

    {

        quitMenu.enabled = true;

        controlsCanvas.enabled = false;

        startButton.enabled = false;

        quitButton.enabled = false;

        controlsButton.enabled = false;

    }

    public void NoPress()

    {

        quitMenu.enabled = false;

        controlsCanvas.enabled = false;

        startButton.enabled = true;

        quitButton.enabled = true;

        controlsButton.enabled = true;

    }

    public void ControlsPress()

    {

        quitMenu.enabled = false;

        controlsCanvas.enabled = true;

        startButton.enabled = false;

        quitButton.enabled = false;

        controlsButton.enabled = false;

    }

using UnityEngine;

[RequireComponent(typeof(NavMeshAgent), typeof(Animator))]

public class Move : MonoBehaviour

{

    [SerializeField, HideInInspector]

    NavMeshAgent agent;

    [SerializeField, HideInInspector]

    Animator animator;

    void Reset()

    {

        agent = GetComponent<NavMeshAgent>();

        animator = GetComponent<Animator>();

    }

    void Update()

    {

        animator.SetFloat("Speed", agent.velocity.sqrMagnitude);

    }

}

    public void StartLevel()

    {

        SceneManager.LoadScene("Level 01");

    }

    public void QuitGame()

    {

        Application.Quit();

    }

}

6. TESTING

6.1 INTRODUCTION&TYPES OF TESTING USED

Game testing, a subset of game development, is a software testing process for quality control of video games. The primary function of game testing is the discovery and documentation of software defects.

Quality assurance is a critical component in game development, though the video game industry does not have a standard methodology. Instead developers and publishers have their own methods.

Testing methodology:

Testing starts as soon as first code is written and increases as the game progresses towards completion.

Early in the game development process focus is on daily feedback for new code. A good bug-reporting system may help the programmers work efficiently. Tester feedback may determine final decisions of exclusion or inclusion of final features.

The game is play-tested and testers note any uncovered errors. These may range from bugs to art glitches to logic errors and level bugs. Testing requires creative gameplay to discover often subtle bugs. Some bugs are easy to document, but many require detailed description so a developer can replicate or find the bug. Testers implement concurrency control to avoid logging bugs multiple times.

Most companies rank bugs according to an estimate of their severity:

A bugs are critical bugs that prevent the game from being shipped, for example, they may crash the game.

B bugs are essential problems that require attention; however, the game may still be playable. Multiple B bugs are equally severe to an A bug.

C bugs are small and obscure problems, often in form of recommendation rather than bugs.

A typical bug report progression of testing process is seen below:

**Identification**: Incorrect program behaviour is analysed and identified as a bug.

**Reporting**: The bug is reported to the developers using a defect tracking system. The circumstances of the bug and steps to reproduce are included in the report. Developers may request additional documentation such as a real-time video of the bug's manifestation.

**Analysis**. The developer responsible for the bug, such as an artist, programmer or game designer checks the malfunction. This is outside the scope of game tester duties, although inconsistencies in the report may require more information or evidence from the tester.

**Verification.** After the developer fixes the issue, the tester verifies that the bug no longer occurs. Not all bugs are addressed by the developer, for example, some bugs may be claimed as features (expressed as "NAB" or "not a bug"), and may also be "waived" (given permission to be ignored) by producers, game designers, or even lead testers, according to company policy.

**Types of testing**:

**Functionality testing** is most commonly associated with the phrase "game testing", as it entails playing the game in some form. Functionality testing does not require extensive technical knowledge. Functionality testers look for general problems within the game itself or its user interface, such as stability issues, game mechanic issues, and game asset integrity.

**Compliance testing** is the reason for the existence of game testing labs.[clarification needed] First-party licensors for console platforms have strict technical requirements titles licensed for their platforms. Even a single violation in submission for license approval may have the game rejected, possibly incurring additional costs in further testing and resubmission. In addition, the delay may cause the title to miss an important launch window, potentially costing the publisher even larger sums of money.

**Compatibility testing** is normally required for PC titles, nearing the end of development as much of the compatibility depends on the final build of the game. Often two rounds of compatibility tests are done - early in beta to allow time for issue resolution, and late in beta or during release candidate.

Compatibility testing ensures that the game runs on different configurations of hardware and software.

The hardware encompasses brands of different manufacturers and assorted input peripherals such as gamepads and joysticks.

The testers also evaluate performance and results are used for game's advertised minimum system requirements. Compatibility or performance issues may be either fixed by the developer or, in case of legacy hardware and software, support may be dropped.

**Soak testing** in the context of video games, involves leaving the game running for prolonged periods time in various modes of operation, such as idling, paused, or at the title screen. This testing requires no user interaction beyond initial setup, and is usually managed by lead testers. Automated tools may be used for simulating repetitive actions, such mouse clicks. Soaking can detect memory leaks or rounding errors that manifest only over time. Soak tests are one of the compliance requirements.

**Beta testing** is done during beta stage of development. Often this refers to the first publicly available version of a game. Public betas are effective because thousands of fans may find bugs that the developer's testers did not.

**Regression testing** is performed once a bug has been fixed by the programmers. QA checks to see whether the bug is still there and then runs similar tests to see whether the fix broke something else. That second stage is often called "halo testing"; it involves testing all around a bug, looking for other bugs.

**Load testing** tests the limits of a system, such as the number of players ,the number of sprites active on the screen, or the number of threads running in a particular program.

**Multiplayer testing** may involve separate multiplayer QA team if the game has significant multiplayer portions. This testing is more common with PC games. The testers ensure that all connectivity methods (modem, LAN, Internet) are working. This allows single player and multiplayer testing to occur in parallel.

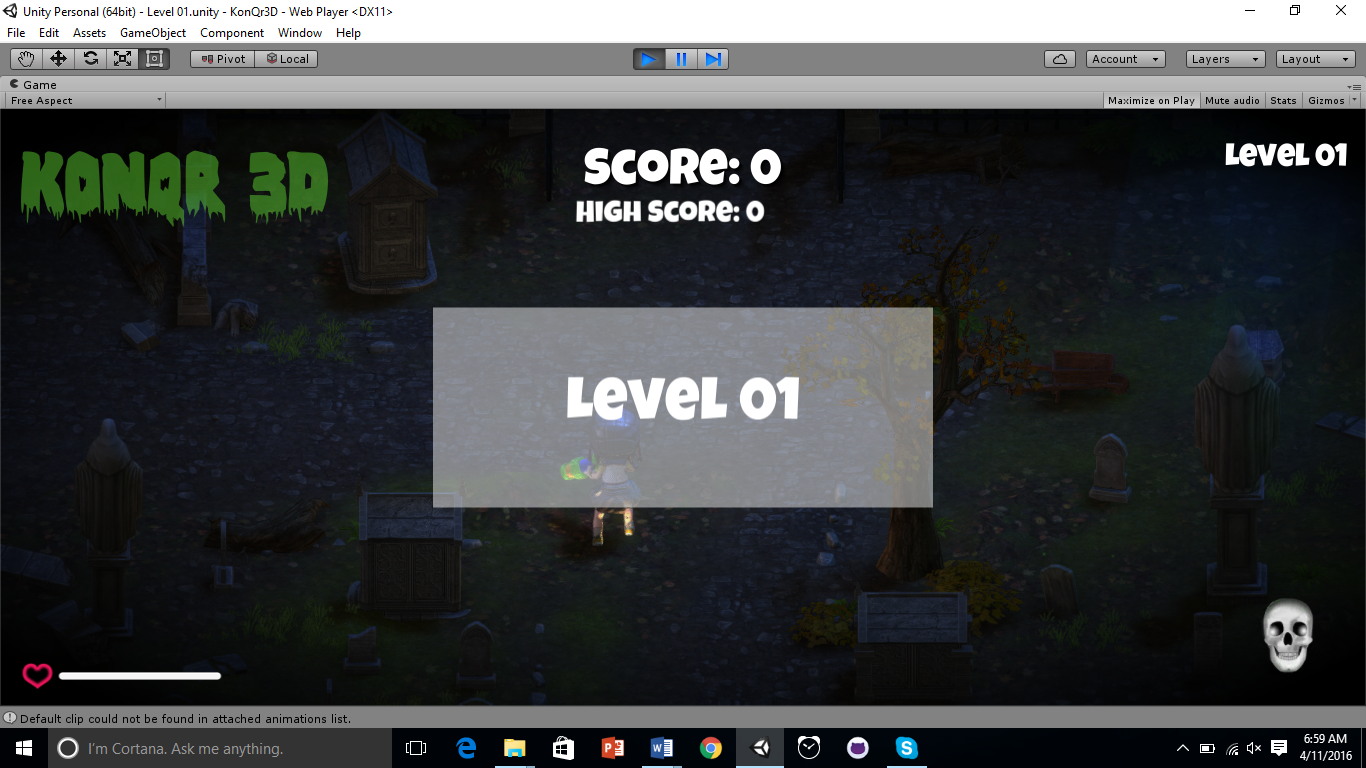
**Mobile game testing** is mainly done manually or as automated during the development. Mobile game testing typically includes all above testing types. Popular mobile game platforms are Android and iOS. Mobile game test automation with image recognition features can be found in Testdroid.

6.2 TEST CASES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case Id #1 | | Test Case Description - | | | |
| S# | Prerequisites | | S# | Test Data Requirement | |
| 1  2  3 | User’s system should have a graphics card  Audio output (speakers)  System with graphics card | | 1  2  3 | Check for animation, movement of character, graphics, Zoom In/Out (all gestures) etc  ON/OFF Device sound and check  Input and output devices (Keyboard,mouse,screen and speakers) | |
| Test Condition | | | | | |
| Playing KONQr 3D | | | | | |
| Step # | Step Details | Expected Results | | Actual Results | Pass/Fail/Not Executed/Suspended |
| 1 | User selects ‘help’ in start screen | Navigate to help page | | Views controls to be used | Pass |
| 2 | Selects ‘play’ button in start screen | Navigates to level 01 | | Navigates to level01 | Pass |
| 3 | Selects ‘pause’ | Dynamic state(play) | | In stationary state(pause) | Fail |
|  |  |  | |  |  |

Table 6.2.1 Test cases for playing a game

7. SNAPSHOTS





8.FUTURE SCOPE(FURTHER ENHANCEMENTS) AND CONCLUSIONS

Through this game, the user can multitask and cross each level. The look and feel of the game makes the player learn the art of defending and combating enemies.

Yet, there can be further enhancements in the game:

* Dynamic start screen
* Death animation
* Various player and environment selection option
* Weapon variation

9.BIBLIOGRAPHY

9.1 e-Books and PDFs

1. “Beginning 3D Game development with Unity” by SueBlackman

9.2 WEBSITES

1 . https://unity3d.com/

2 . https://www.assetstore.unity3d.com/en/

3. https://www.youtube.com/watch?v=tqPKRxFFzYc

4. www.stackoverflow.com

10.ACRONYMS AND BUSINESS TERMS

|  |  |
| --- | --- |
| **Acronym** | **Full form** |
| API | Application Program Interface |
| AI | Artificial Intelligence |