University of Aden
Faculty of Engineering
Department of Computer Science and Engineering

WAR GAME

By

Yaqoob Saif Mahyoob Ameer

Bander Ameen Qassem

Mohammed Nasser Alhuraibi

Mnder Abdulhabeeb Kasheh

Basem Abdulqawi Abdulla

Saber Abdulla Ahmed

A thesis submitted in partial fulfillment for the degree of
BACHELOR OF SCIENCE in
COMPUTER SCIENCE & ENGINEERING

Under the guidance of

Dr.Lutfi Moh'd Khanbari

Batch 16

October 2018

DECLARATION

We hereby declare that the work which is being submitted in this project titled:

WAR GAME

In partial the fulfillment of the requirements for the award of degree of Bachelor of science in Computer Science and Engineering submitted in Faculty of Engineering under the supervision of **Dr.Lutfi Moh'd Khanbari**, is our own. The project has not been accepted for any degree and is not concurrently submitted for award or other degree.

Student Name :Yaqoob Saif Mahyoob Ameer	Roll Number : 135564
Student Name :Bander Ameen Qassem	Roll Number : 135518
Student Name : Mohammed Nasser Alhuraibi	Roll Number : 135051
Student Name : Mnder Abdulhabeeb Kasheh	Roll Number : 125593
Student Name :Basem Abdulqawi Abdulla	Roll Number: 135028
Student Name :Saber Abdulla Ahmed	Roll Number : 135018


CERTIFICATE

This is to certify that the project titled

WAR GAME

Has Been Successfully Completed

By

YAQOOB SAIF MAHYOOB AMEER	135564
BANDER AMEEN QASSEM	135518
MOHAMMED NASSER ALHURAIBI	135051
MNDER ABDULHABEEB KASHEH	125593
BASEM ABDULQAWI ABDULLA	135028
SABER ABDULLA AHMED	135018

Under my supervision and is adequate in terms of scope and quality for the award of the degree of Bachelor of Science in Computer Science and Engineering

Supervisor,

Dr. Lutfi Moh'd Khanbary

Department of Computer Science & Engineering
Faculty of Engineering, University of Aden

Data : ……

Acknowledgment

First we would like to thank Allah the greatest for the continuous and surplus support each time with us.

We would also like to express our gratitude to all of our support and support during the graduation project, starting with the endless thanks to our supervisor.

Dr.Lutfi Moh'd Khanbary , who made every effort to encourage us to do a great job of providing our group with valuable information and advice to be the best at a time, we would also like to thank

Eng / Reem Ali Salah Alardi,

In addition, we would like to express our thanks to all faculty members in the Department of Computer Science and Engineering.

Also we are very grateful to our families, friends for their support throughout the production of this project.


ABSTRACT

Over the years the games industry is rising, using new technologies and reaching new target audiences. In the scenario of the games development, sometimes professionals from several areas work in the same product with large and exhaustive documentation.

War game is similar to any other game in which a party competes with the computer or two parties against some to win the game's outcome and can be conceived as a map where the forms or symbols representing the military units of the parties are attacked according to specific rules, rules and procedures based on knowledge and experience that limits the game.

In our project will be playing with the computer where the game in more than one level and in each level there are many obstacles and difficulties faced by the player such as fighters or warplanes where the player must overcome them to move to the level after which each level is harder than the level before him where the player Use cars, warplanes and various weapons.

The purpose of this project is to entertain and conduct experiments to improve the quality of humanitarian decisions in a particular case through measurement, implementation and management based on scientific rules.

Contents:

INTRODUCTION

1.1 Introduction to video games

History of video games:

The history of video games goes back to what it was in the early 1950s, when academic computer scientists began designing simple games and simulations as part of their research. Video games were not popular until the 1970s and 1980s, when video games and video games were introduced using joysticks, buttons, and other controls, along with graphics on computer screens and home computer games for the general public. Since the 1980s, video games have become a popular form of entertainment and part of modern popular culture in most parts of the world. Early arcade video games were developed from 1972-1978. During the 1970s, the first generation of home keyboards appeared, including the popular Pong game and various "clones". The 1970s were also the era of central computer games. It was the golden age of arcade video games from 1978 to 1982. Videos with large animated cartoon machines decorated with coin were popular in shopping malls and popular, affordable home keyboards such as the Atari 2600 and intellivision enable people to play games on their home TVs. During the 1980s, gaming devices, online games and portable LCD games appeared. This era of video game crash was influenced by 1983.

From 1976 to 1992, the second generation of video boards emerged.

The third generation of keyboards, which were 8-bit modules, appeared from 1983 to 1995. The fourth generation of keyboards, which were 16-bit models, appeared from 1987 to 1999. The 1990s saw the return and downfall of the hallways, the transition to three video games Dimensional, improved handheld games, and computer games. It was the fifth generation of keyboards, which were 32 and 64-bit units, from 1993 to 2006. During this era, mobile phone games appeared. During the first decade of the twenty-first century, the sixth generation of keyboards (1998-2013) emerged. During this period, online games and mobile games have become major aspects of game culture. The seventh generation of

consoles was from 2005 to 2012.

This era marked huge development budgets for some games, with some cinematic graphics. The Wii console launched a best-selling, which the user can control the game actions with the real life movement of the controller.

In 2013, the eighth generation of keyboards, including Nintendo Wii U, Nintendo 3DS, Microsoft Xbox One, Sony PlayStation 4 and PlayStation Vita appeared. Bessie games have played a large share of the market in Asia and Europe for decades and continue to grow due to digital distribution. Since the development and use of large-scale consumers of smart phones, mobile games have been a key factor in the games, because they can reach people who were not interested in games, who cannot afford or support custom devices, such as video games

1.1.1 Developing video games:

Game development is a software development process which results in a game. A computer game is a game that involves human interaction with a user interface to generate visual feedback on a video device and profit for developer. Development is performed by one developer or by group of developers.

A development team includes the following roles:

A game designer is a person who creates the game design. Game design includes game play, the rules and structure of a game. A game designer has to have a clear understanding of the game concept and provide the game with narrative content for development team. In larger projects, there are often separate designers for various parts of the game, such as game mechanics, user interface, characters, dialogue, etc. The designer should have good imagination, logical thinking and understanding of the software part of the project. A game artist is a person who creates game art. Art may be 2D oriented or 3D oriented.

2D artists may produce concept art, sprites, textures, environmental backdrops or terrain images, and user interface. 3D artists may produce 3D models, animation and 3D environment. Artists sometimes occupy both roles. A game programmer is a software engineer who implements game codebases and related tools. An individual programming task may include physics, AI, managing of graphical content, integration of sounds, implementation of various games rules, UI, input processing, network communications, etc.

The basic pieces really haven't changed that much since the birth of the Atari 2600.

1.2 Introduction to graphic design:

1.2.1 What is a Graphic Designer?

Graphics design is the art of planning and presenting ideas and experiences using visual forms and text. Graphic design should work to convey a meaning, information or idea, to act as a person, and the goal of graphic design is to facilitate people's lives. The designer makes it simple and simple. Sweetest. Apart from learning programs for design, it is important to be familiar with graphic design in general, such as knowing the difference between point, line and shapes. Many design programs have used two programs for the work of this project:

Adobe Photoshop and Adobe Illustrator.

1.2.2 What does a Graphic Designer do?

Graphic designers combine art and technology to communicate ideas through images and the layout of web screens and printed pages. They may use a variety of design elements to achieve artistic or decorative effects. They develop the overall layout and production design for advertisements, brochures, magazines, and corporate reports. Graphic designers work with both text and images. They often select the type, font, size, colour, and line length of headlines, headings, and text. Graphic designers also decide how images and text will go together on a page or screen, including how much space each will have. When using text in layouts, they collaborate closely with writers who choose the words and decide whether the words will be put into paragraphs, lists, or tables.

1.2.3 What is some good advice for graphic design students?

It's important to love the whole design process, from beginning to end. Make sure to keep updated with any changes in the industry; it moves at a fast pace. Having said that, don't blindly follow trends leaving great ideas behind. Doing design for the sake of design is not the way to go. Sometimes you'll become overly invested in a design you're working on, so much so that you become defensive if it's criticized. Don't get overly sensitive about criticism - keep your mind open to suggestions. Just because someone doesn't like your work doesn't mean you are a bad designer.

Invest in your portfolio, and be selective in what goes in it. Focus on your best pieces, and make them as impressive as you can make them. Employers will often ask you to explain what's in your portfolio and why it's in there, so make sure you can explain why you included each piece.

1.3 Research Background

The game of war has many uses, some used in reaching solutions and decisions to solve the dilemmas (tasks). Or training on the stature in certain roles at specific times to achieve the best results. Some use them to evaluate and develop military plans at various levels, and to evaluate weapons and equipment under certain circumstances.

It seems from the first sense that the term war is modern but in fact old in its idea, its application and its uses. The idea of a war goes back a long time BC. The Chinese strategic thinker (Sun Itzu) exercised it on a special board with colored stones representing the forces of both parties and each party aims to encircle the other party and spread the use of such games in several countries such as: India, Greece, Egypt and others. With the technological development and the emergence of computer technology, and especially in the seventies, the computer began to enter the field of war games and thus reduced the play of manual warfare. Using the computer to play the war, this jump was regarded as the beginning (the Golden Age) to play the war.

1.4 Problem Statement

There are many problems that face us in this project, which need to look more so we can solve such as control the movement of the model or stop the movement and how to add different weapons to the enemy, also have a problem in the code, and there is a basic problem is that video games take a lot of RAM and have an impact on the speed of the computer.

1.5 Objectives

The game aims to achieve the following objectives:

1- Training (decision-making) at various levels.
2- Planning (evaluation plans) at various levels.
3- Entertainment (to gain experience - compete - and develop people skills).

4- Understanding and gaining experience in the field of 2D and 3D graphics, animation design and object modeling, as well as experience in the field of television, advertising designs and films.

1.6 Scope of research:

War game could help in different civil fields as:

1. Civil fields
   - Graphic Designer.
   - Helping people improve decisions.
2. Military fields
   - Help them in military plans.

1.7 Thesis Organization

- **Chapter 1:** Introduction
- **Chapter 2:** Literature Review
- **Chapter 3:** Methodology
- **Chapter 4:** Results and Discussion
- **Chapter 5:** Conclusion and Future work

LITERATURE REVIEW

2.1 Research From Previous Project:

Of course, we are not here to study any previous project and was a development of our previous project, which we put in (summer application) from here started the idea from our heads that we are working on the development of our previous project and thank god we completed this project and here we will put a scenario for this project all games start from the idea, the idea is the following: we have a place where there are enemies and they have tanks and anti-aircraft how to restore the region from the grip of enemies thus the game consists of three levels each level harder than the first in terms of the number of enemies and now come to explain what is happening in the stage number one phase one is a mountainous area and the enemy zone is located on the player's platform. The first thing is that the player goes to take the weapon from the commander or the gunman and then take a helicopter and then it goes to a certain point but if i run away the player this particular point in the helicopter will launch anti-aircraft missiles from the enemy side he has to land at the specified point and take the car and go towards the enemies are eliminated and then move phase two, which consists of a city means we have two cities in which the enemy is the same as the first scenario but here it is more difficult in terms of many promising. The player does not have a helicopter but has a car and moves to the other city the enemy is eliminated and moved to a

third level which is a rural area and the scenario is the same scenario but here the enemy has helicopters and this is the most difficult stage

Fig. (2.1) player taking in plan

2.1.1 Design of Educational Game:

The research of educational games (computer and video games) can be traced back to the middle of 1950s, when attention was mainly paid to the design, development and Application of commercial video games [1]. Till 1980s, some researchers began to Concentrate on the educational potential of video games, studying how it motivated Players' intrinsic motivation and trying to apply it into instruction. At the end of 1990s, With the development of computer technology and network, the research was not only Restricted to video games, but extended to the field of electronic games and large-scale Online games. In the late 1990s, the design, development and application of educational Games were gradually concerned in the field of instruction and game industry. We began our review with an exhaustive search of electronic databases. This search Led us to several meta-analysis and reviews including those conducted by kebritchi,m.et Al. And kowit rapeepisarn et al. Etc. On designing theories and practice in educational Games. From this initial search, we began a more extensive review using basic search Procedures and standard criteria for a

comprehensive literature search. This process Included library searches (both electronic and manual) in educational databases such as Eric, psycinfo, and social sciences index using search terms including educational

Game design, computer game design, video game design, and so on. The design is a key to the balance between educational potential and playfulness in Educational games. This article comments the history of educational game design Thought, design theories, design approaches, as well as design and development models

2.1.2 History of Video Games

The history of video games is another great take on the history of video games. The story begins with the research labs of the 1940s and tells the groundbreaking success of the Wii, providing a unique look at gaming's past. From the twentieth century to the end of the eighties, and the most famous of these games space invaders, super Mario, duck hunt and so on. In 1952 the "Douglas" by designing the first graphics games for the computer at all, she was working on edsac vacuum-tube computer. Whose screen contained a cathode ray tube. In 1958, William hainbootham designed the first video game ever, a tennis game for two, then came 1962, Steve Russell invented the game space war this game designed on a mainframe. Then in 1967, Ralf bear designed the first game which was played on television (semi-Atari) and was called chase.

2.1.3 Video Games as Virtual Worlds for Learning

The first step towards understanding how video games can (and we argue, will) transform education is changing the widely shared perspective that games are "mere entertainment." More than a multi-billion-dollar industry, more than a compelling toy for both children and adults, more than a route to computer literacy, video games are important because they let people participate in new worlds. They let players think, talk, and act—they let players *inhabit*—roles otherwise inaccessible to them. A 16-year-old in Korea playing *lineage* can become an international financier, trading raw materials, buying and selling goods in different parts of the virtual world, and speculating on currencies. A *deus ex* player can experience life as a government special agent, where the lines between state-sponsored violence and terrorism are called into question.

These rich virtual worlds are what make games such powerful contexts for learning. In Game worlds, learning no longer means confronting words and symbols separated from the things those words and symbols are about in the first place. The inverse square law of gravity is.

No longer something understood solely through an equation; students can gain virtual Experience walking on worlds with smaller mass than the earth, or plan manned space flights that require understanding the changing effects of gravitational forces in different parts of the solar system. In virtual worlds, learners experience the concrete realities that words and symbols describe. Through such experiences, across multiple contexts, learners can understand complex concepts without losing the connection between abstract ideas and the real problems they can be used to solve. In other words, the virtual worlds of games are powerful because they make it possible to develop situated understanding.

Although the stereotype of the gamer is a lone teenager seated in front of a computer, Game play is also a thoroughly social phenomenon. The clearest examples are massively Multiplayer online games: games where thousands of players are simultaneously online at any given time, participating in virtual worlds with their own economies, political systems, and cultures. But careful study shows that most games—from console action games to pc strategy games—have robust game playing communities. Whereas schools largely sequester students From one another and from the outside world, games bring players together, competitively and cooperatively, into the virtual world of the game and the social community of game players. In schools, students largely work alone with school-sanctioned materials; avid gamers seek out news sites, read and write faqs, participate in discussion forums, and most importantly, become critical consumers of information. Classroom work rarely has an impact outside of the classroom; its only real audience is the teacher. Game players, in contrast, develop reputations in online communities, cultivate audiences as writers through discussion forums, and occasionally even take up careers as professional gamers, traders of online commodities1, or game modders and designers. The virtual worlds of games are powerful, in other words, because playing games means developing a set of effective social practices.

By participating in these social practices, game players have an opportunity to explore New identities. In one well-publicized case, a heated political contest erupted for the president

2.1.4 Games and Gamers

Researchers and commentators have attempted to understand the lure of computer games. This has been characterized as a combination of fantasy, challenge and curiosity, and a level of engagement described as 'flow' where players become oblivious to distractions. Concern has been expressed that this leads to a neglect of other activities, often assumed to be automatically more worthy. Other authors see games play as inherently valuable, leading to a development of a range of skills and competences that may transfer to other social and work-related uses of digital Technologies. The debate around violence and gaming is as yet unresolved. There are two perspectives; that games increase aggression or that games provide a release for pent-up aggression. In all likelihood both are legitimate conclusions, and the outcomes varies with game and player**.** Gender is a common subject of games related research, focusing mainly on the image of females within games, or the role of gender in influencing games play. There Are few clear outcomes, but the proportion of gamers who are female seems to be growing and this may be related to the increase in social gaming through on and offline multiplayer options.

2.2 Theoretical Background

Design is about making decisions in many fields design require engineering skill; proto-types help designers select the best solution. This section describes prototyping used for game design.

2.2.1 Software Prototyping

Prototype refers to incomplete versions of the software program. A prototype typically simulates only a few aspects of, and may be completely different from, the final prod-uct. Prototyping is a process of creating model for future applications in order to deter-mine the validity of the application, its functionality and, in general, the concept of the application.

2.2.2 Game Development

This section describes the basic game development concepts used in the present study. Game development is a very broad discipline and the majority of terms cannot be covered in a single paper.

2.2.3 Finite State Machine

The finite state machine is used by game developers to implement any behavior, for example the behavior of some units or complex objects.

Fsm consists of four main elements:

- states that determine the behaviour
- the state transitions which define the transition from one state to another
- rules and conditions that must be met for the transition
- input states obtained externally or internally, according to the rules that may lead to a transition state

2.2.4 Game Resources

Any game consists of many parts, but among them there are two main ones. First is game resources, and the second is the code.

Game resources are the graphic, music and other assets that are used to decoration of the game. Graphic files are used to store images.

2.2.5 Genre and Audience

Information about the game genre and target audience is very important, as it allows understanding the features of the game, especially the future promotion of the finished product. Target group represented by users who may be interested in the game.

2.3 Concerns Around Computer Games Play

2.3.1 Games Displacing Other Activities

Games displacing other activities the impression of much of the mass media, and some research, is often of a population of children playing video games to the exclusion of all other activities. The implication often being that time traditionally dedicated to ‘better activities’ such as social play and physical recreation, is spent on anti-social, physically unhealthy and mentally numbing Game playing, possibly leading to addictive patterns of use. Though research is often oriented towards

'snapshot' results as opposed to tracking game-players over longer periods of time, longitudinal research does not show a long-term significant commitment to games over Other activities.

2.3.1 The Effects of Violence on Game Players

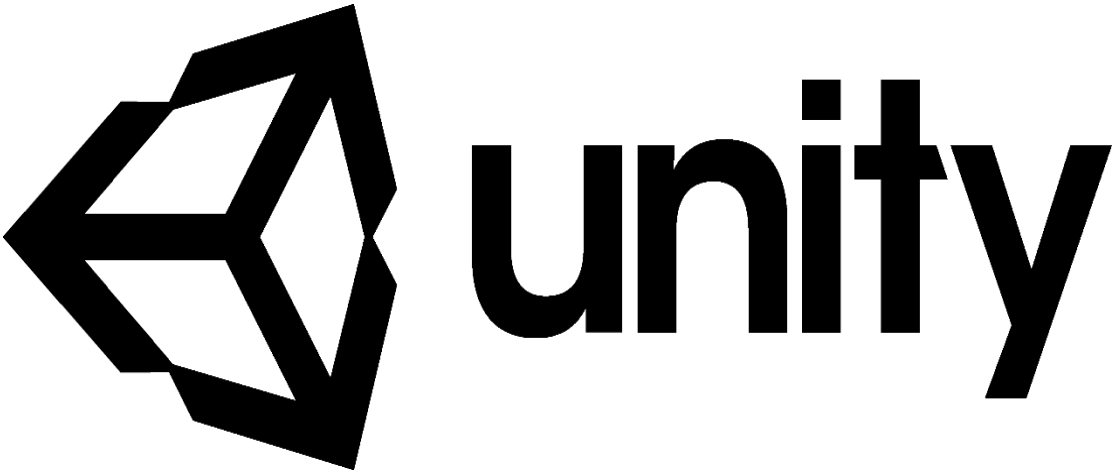
The issue of violence within games is an important one, as it crucially affects the acceptability of digital games in general and especially in the educational sector. Views on violence and game players are often polarized and entrenched, even amongst academic researchers.

METHODOLOGY

3.1 Tools used in our project

3.1.1.1 Unity3d

Is a game engine or game authoring tool that enables creative people to build video games. With Unity, we can build video games faster and easier than ever before. The unit allows you to create interactive games running on all platforms such as Windows and Linux, as well as smart phones with different systems. You can develop it using several languages including C #, JavaScript.



Fig,3.1 unity logo

3.1.1.2 What is Unity?

1. **Game engine** – system designed to help create video games

- Easier & Faster

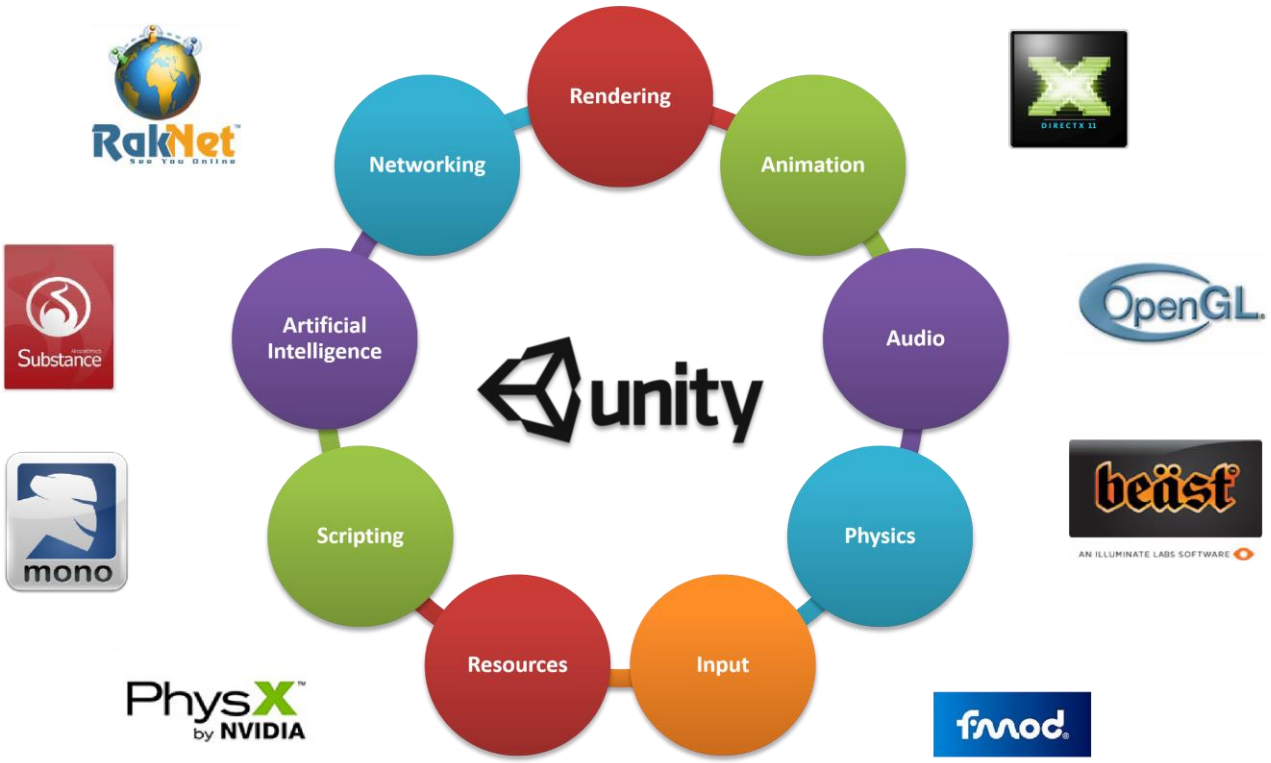
2. **Visual editor** – see changes in real-time

- Interactive & Rapid prototyping

3. **Component-based** – functionality built out of smaller pieces

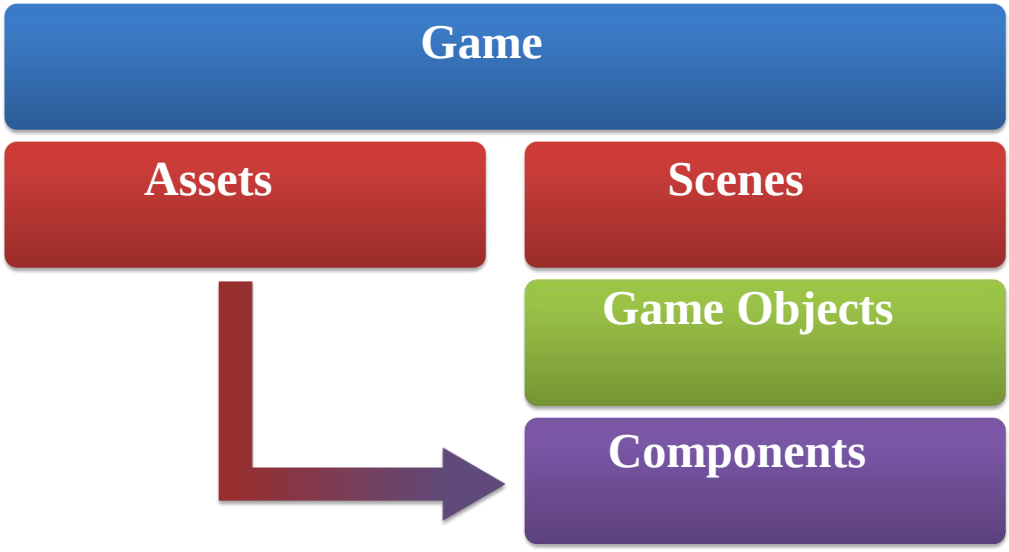
- Modular & Extensible

3.1.1.3 What can Unity do for you?

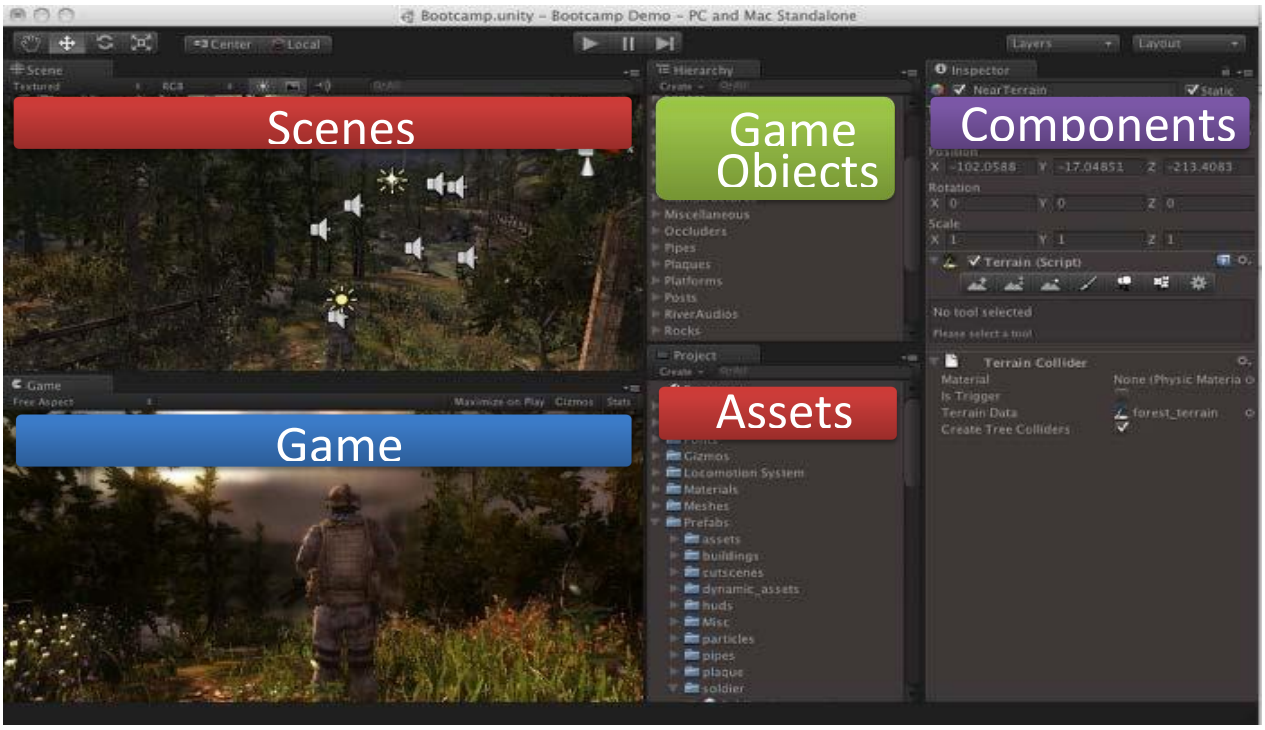


Fig,3.2 unity do

3.1.1.4 Architecture



3.1.1.5 Editor Interface

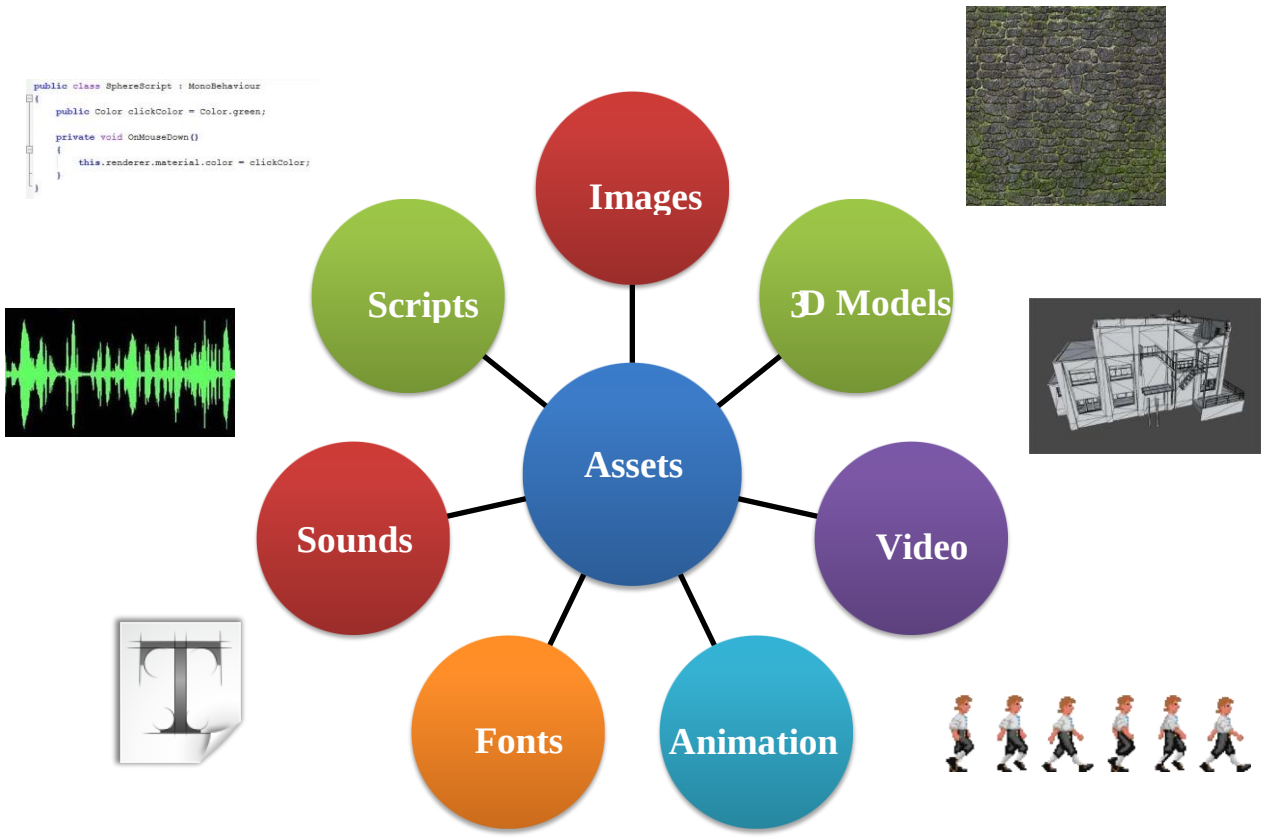


Fig,3.3 unity interface

3.1.1.6 Assets

Since Unity 3, the Asset Store has been similar to Apple's/Android's app stores, except, instead of apps, you can buy prebuilt assets that can be imported directly into your project. We will be using this in our project; but, before we do so, we will need to have an account, that can be created using the following steps:

1. To open the **Asset Store** via Unity, we can go to **Window | Asset Store** from the top toolbar.
2. Next, go to the top-right corner of the panel and click on the **Create Account** button. Fill out your information and click on **Create account** at the bottom of the screen.
3. You should receive an activation e-mail shortly after you submit the form.
   Open it and click on the activation link provided to verify your account.

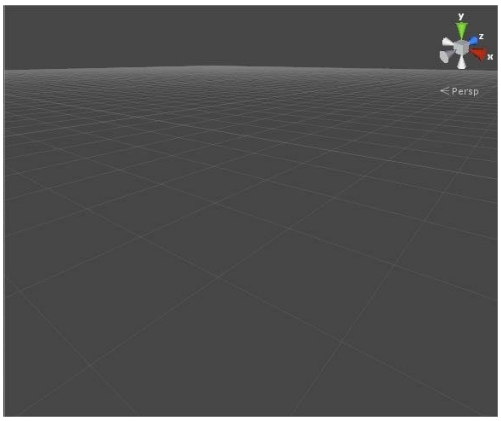


Fig,3.4 Assets Continer

3.1.1.7 Scene

- Unity games are divided into scenes
- Scenes are empty spaces...
- ...that can be filled with game objects

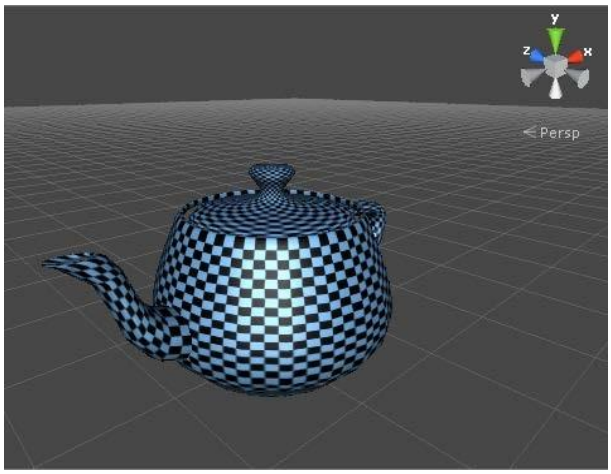
Fig,3.5 Game



Fig,3.6 Scene

3.1.1.8) Game Object

- Everything inside a scene is a game object
- Game objects also start out empty and do nothing...
- ...but by adding components to them they can become anything!



3.1.1.9 Component

- Each component adds a piece of functionality to the game object
- The combination of all components defines what the game object is

Other Components

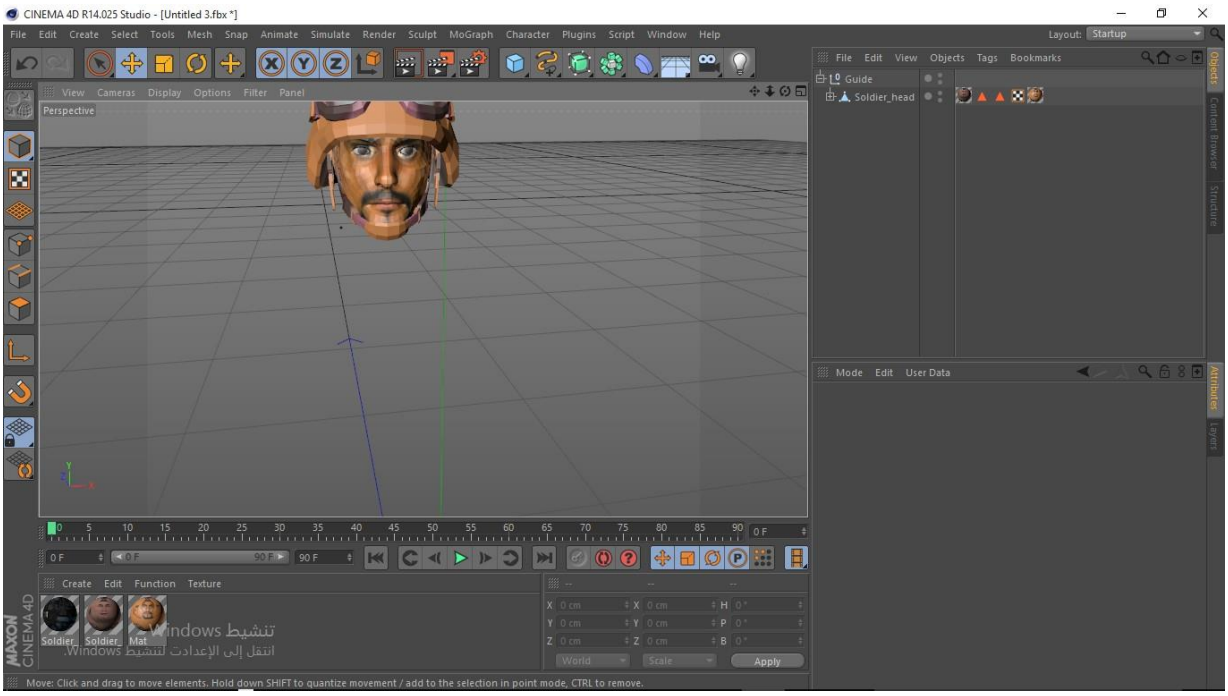
- Light
- Camera
- Text mesh
- Audio listener & source
- Particle system
- Skybox
- …and many more.

3.1.2 Cinema4d:

Is a three-dimensional graphics program with advanced features produced by Maxon Computer German and Cinema 4D can be modeling, moving, lighting, cladding and output. The program uses in the design of graphics and games, as well as architectural design, architecture, decoration and design of animation films. The Cinema4d program provides the tools and possibilities for 3D design so that we can create any design using the Modeling Tools in the program and create modifications to 3D objects and then output the final work. Without the need for any additional or external program.

Fig,3.7 cinema 4d logo

No matter if you're just checking CINEMA 4D out or if you already own your own copy of CINEMA 4D or one of its complete XL or Studio Bundle packages, you already know about the incredible things CINEMA 4D can do. We have been working very closely with our customers for several years now in order to satisfy their needs and wishes. This has lead to the creation and introduction of new functionality, according to their needs.



Fig,3.8 cinema 4d interface

3.1.2.1 Installing the application

Follow these steps to install the application onto your data carrier:

- Place the installation DVD into your computer's DVD drive. The DVD's symbol should appear in your Explorer (Windows) / Finder (Mac) window. If this symbol does not appear, open the DVD manually in the Explorer / Finder.
- Double-click on the "Setup(.exe)" file.

CINEMA 4D, BodyPaint 3D or NET Render will now be installed. Which components will be installed depends on the software package you purchased.
This information is included in the registration information you received.

Windows

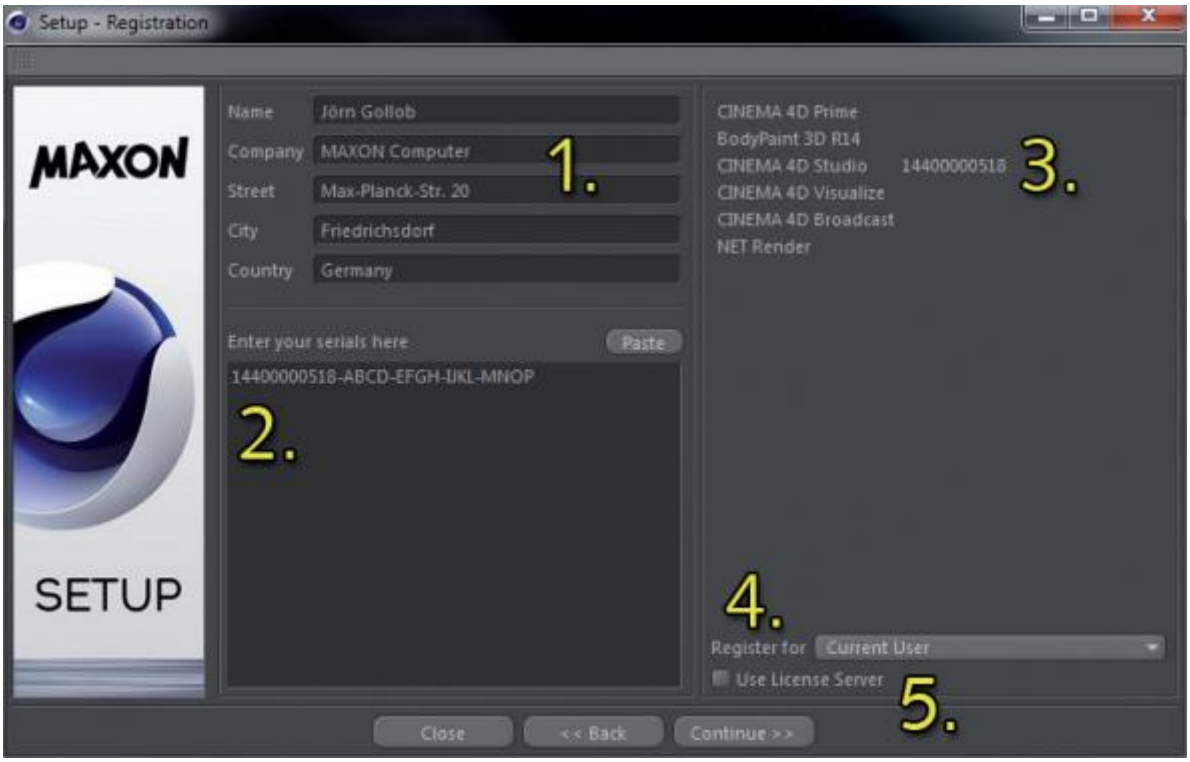
First you will be asked to select the language in which the installation should take place. After selecting the desired language, click on "OK". The minimum system requirements for installing CINEMA 4D R14 onto a Windows system are:

- Windows XP (32/64-bit), Vista (32/64-bit), Windows 7 (32/64-bit).
- For Windows XP an installed Internet Explorer 7.
- For Windows 7 operating systems, Internet Explorer 8 or higher.
- 1024 of free MB RAM.
- AMD or Intel (compatible) processor with SSE2 support.
- An OpenGL-capable graphics card (NVIDIA or AMD/ATI).
- A screen (monitor) resolution of 1280 x 800 with a minimum of 24-bit color depth.
- 7 GB of free hard drive space.

Note: If the if SSE2 support does not exist, an error message will appear and the installation process must be terminated.
In the next installation step, an informational text file will be displayed. This file contains installation-specific information. Please read this file carefully because it may contain information regarding changes to this installation document, among others.

Once you have read this file, click on “Continue”.

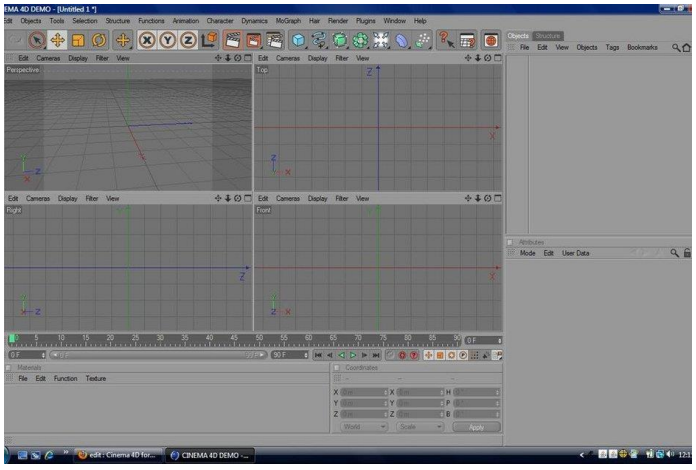
You will be brought to the registration page, in which you will be required to enter your personal information and the serial number included in your registration information.



Fig,3.9 cinema 4d setup

3.1.2.2 Modeling :

Cinema 4D is a professional 3D modeling and animation software application. This tutorial is designed for those who have little or no experience in 3D modeling using Cinema 4D. If you do not already have Cinema 4D installed on your computer.



Fig,3.10 cinema 4d interface modeling

In our Project, we will model a face for student Bandar. We will create the cube using object primitives with the fillet option. The dots will be created using sphere primitives, arrays to create copies of objects, and booleans to delete pieces out of object

Fig,3.11 face modeling

3.1.2.3 Animation :

CINEMA 4D Visualize includes two animation objects that let you create an animation that spans a single day or an entire year. All you have to do is define the attributes accordingly and start the animation. The (CINEMA 4D) global Z-axis represents the North.

To make aligning and illuminating your Project easier, the new Sky object displays a compass rose at the Project's center. This lets you easily align your Project and check the orientation of the sun.

The following properties can be edited in the Attribute Manager.

- **Basic**

  "Start" time or date
  "End" time or date
  "Cloud Movement" (defines the speed of the cloud movement)• "Clouds" (turns clouds on or off)

- **Advanced**

  Switch between simple and physical sky models ("Physical Sky")
  The intensity of the surrounding environment's light ("Sky Intensity")
  The intensity of the sunlight ("Sun Intensity")
  On/off switch for lens flares ("Lens flares")

On/off switch for atmospheric effects (“Physical Atmosphere”)
The rendered atmosphere’s density (“Atmosphere Density”)
Adaption of the atmosphere’s size to the Project size (“Atmosphere Scaling”)

The length of the animation correlates directly to the length of the overall animation, as is the case with the camera animation. The overall length of the animation can be defined in the “Edit / Project Settings..” menu.

Fig,3.12 Animation time line

3.1.2.4 Motion Layer System

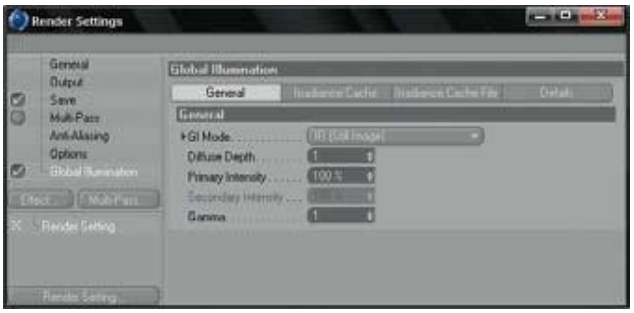
Motion Layers are mixed in the Timeline. So-called Motion Sources are created using animation data that can then be mixed in the Timeline using Motion Clips. It is important that the Motion Clips themselves contain no animation data. The Motion Clips reference the already created animation data. This makes it possible for multiple Motion Clips to reference a single source (non-destructive animation) and deleting a Clip will not result in the source (original) animation being deleted. The Motion Layer System takes the place of the Motion Mixer functionality, familiar to many users from Release 9. This new and more powerful system lets you save Motion Sources that you have created as presets, which can be used to create your own preset library.

3.1.2.5 Animation Layer System

This system works roughly similar to Posemixer or the Morph tag. Individual animations are displayed in the Attribute Manager and are organized there as superimposed layers. For example: You assign the character you want to animate a Motion System tag (can be assigned to the top object in the hierarchy. All keyframes of all Child objects will be taken into account by the Motion System tag when the animation is recorded) and add an animation layer. Now record the first keyframes for your character’s motion and add a new layer at any time during the process. The currently active layer will contain any keyframes created while it is active. If a new layer is created, all subsequently created keyframes will lie on this layer.

3.1.2.6 New Render Settings

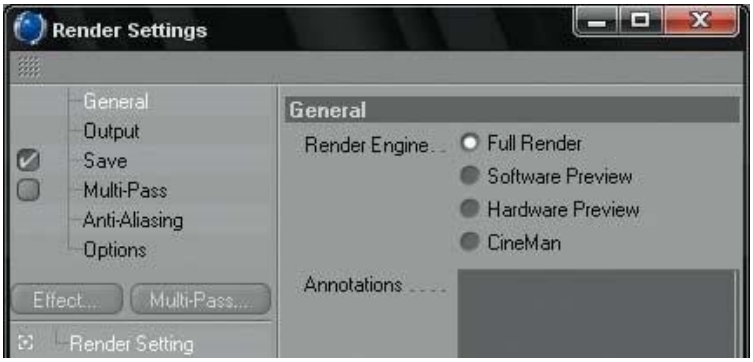
The Render Settings have been completely reorganized in order to give you a better overview of the numerous Render Settings. Render Settings can now be saved as presets that can be grouped, commented and renamed. This lets you better organize large projects and easily switch from one Render Setting preset to another. The Render Settings menu has been given a new look and offers a better overview of the available options.



Fig,3.13 Redering setting

3.1.2.7 New Global Illumination (GI)

The GI render engine has been completely reworked and operates entirely differently in R11 – for even more realistic rendering! The new GI boasts noise reduction and flicker-free rendering. The interface has been redesigned to make it easy to use for both professional and novice users alike.



3.1.3 Adobe Illustrator.

Illustrator is a vector-based imaging program. Unlike Photoshop, which deals in pixels (raster images), this one deals in lines and algorithms for various shapes. It functions by generating curved paths connected by modifiable anchor points. These anchors, with their handles, are ultimately editable and never "leave" the structure of the file.

3.1.3.1 What are vector graphics?

Computer graphics fall into two main categories -- vector graphics and bitmap images. Understanding the difference between the two helps you create, edit, and import artwork.

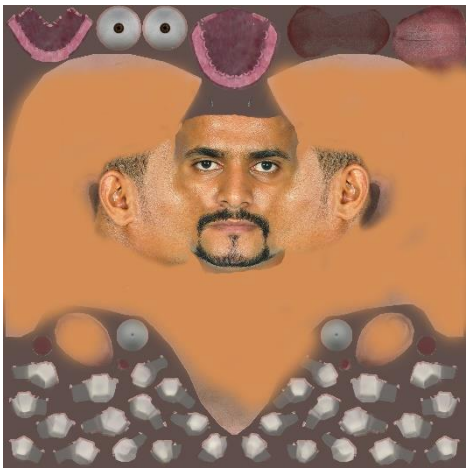
In Illustrator, the type of graphic image can have important effects on your workflow. For example, some file formats only support bitmap images and others only vector graphics. Graphic image types are particularly important when importing or exporting graphic images to and from Illustrator. For example, linked bitmap images cannot be edited in Illustrator. Graphic formats also affect how commands and filters can be applied to images; some filters in Illustrator will only work with bitmap images

3.1.4 Adobe Photoshop.

Fig,3.14 photoshop logo

Adobe Photoshop is the industry-standard tool for digital imaging, which makes Photoshop expertise a valuable commodity in the workplace. Learning Photoshop is also a good way to learn imaging concepts. Concepts you learn from working with Photoshop apply to other imaging tools as well (including Photoshop Elements, which has a very similar user interface and features).

In our project we used this tool to create Texture Material for 3d face.



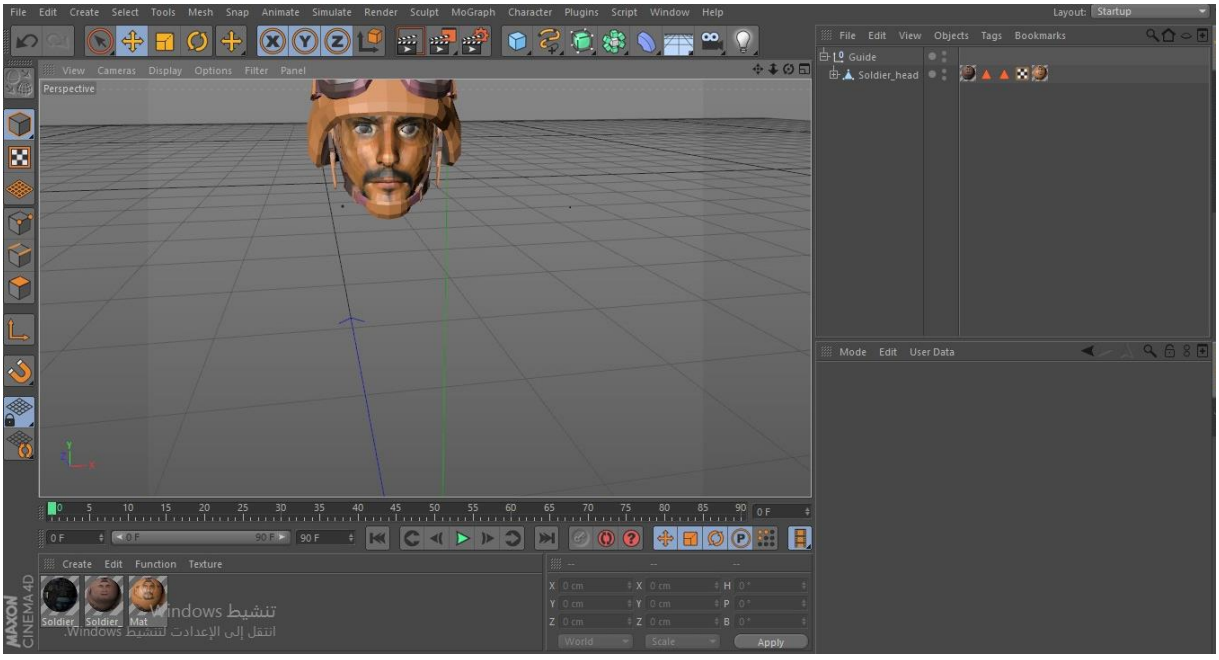
Fig,3.15 Texture Material

3.2 Design

3.2.1 Design and Creating Player

The first Step we import the player modeling from C4D.

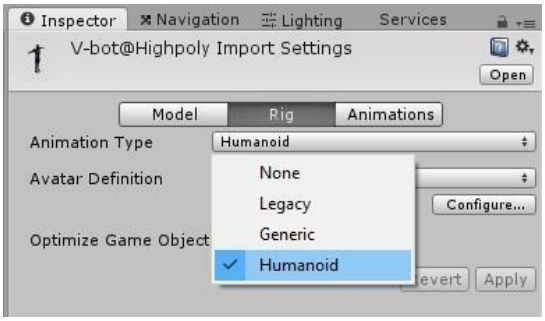
In our project to modeling the face character for student Bandar



Fig,3.16 student face

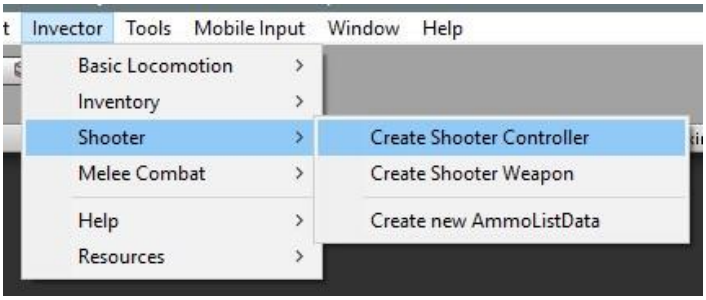
a) Creating A New Shooter Controller

Make sure that your fbx character is set up as **Humanoid**



Fig,3.17 humanoid

To setup a new character, go to the tab *Invector* > *Shooter* > *Create Shooter Controller*

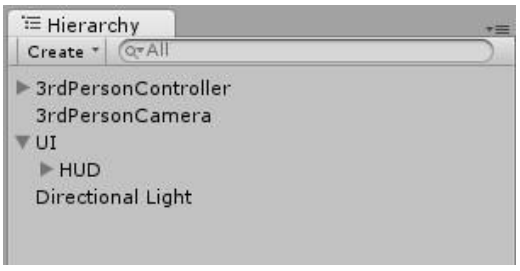


Fig,3.17 Create **shooter**

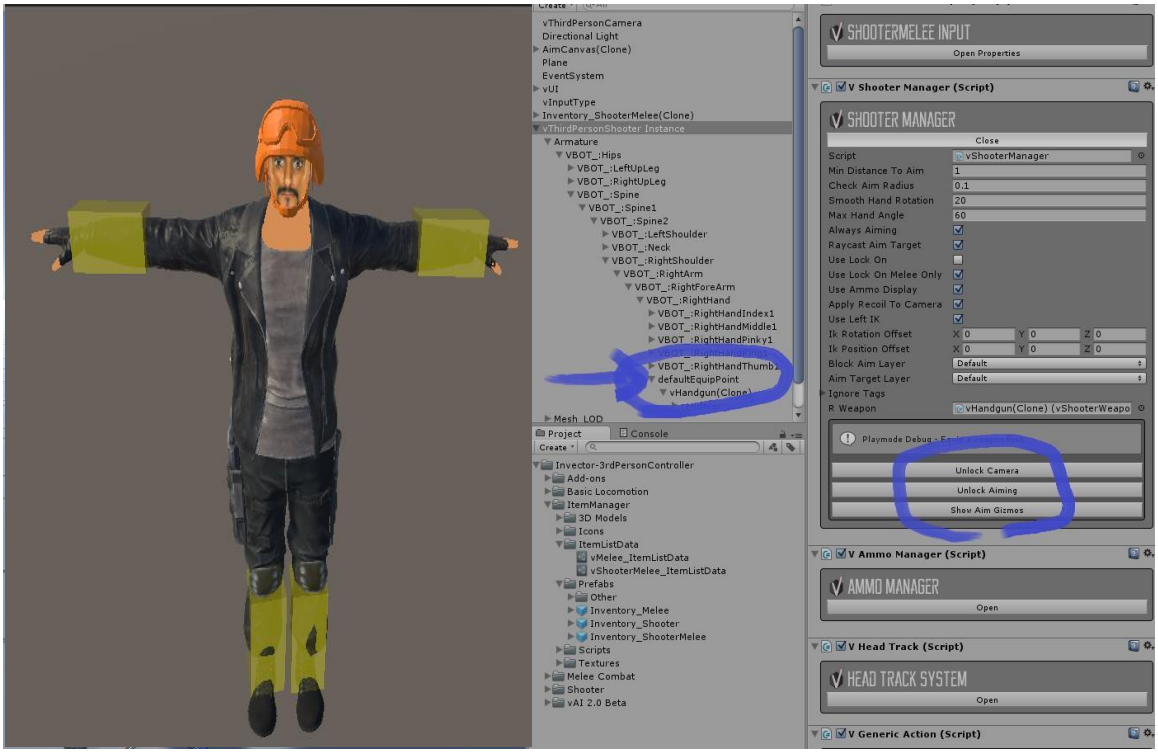
Make sure your Character is **Fully Rigged** and set up the FBX as a **Humanoid**, then assign the FBX to the field “FBX Model” and click on the button “Create”.

Ps* Make sure to select the **Invector_ShooterOnly or Invector_ShooterMelee** if you want to use both shooter and melee as your Animator Controller, you can find the file at the folder: Shooter > Animator, or just click on the little circle icon.

The **Character Creator** window will take care of all the hard work automatically and set up components such as capsule collider, layers, tags, rigibody, etc... It will create the **ThirdPersonController**, **ThirdPersonCamera** and a UI Canvas with a **HUD** to display health, stamina and other information’s.



b) How to Align a Shooter Weapon AND lefthand IK

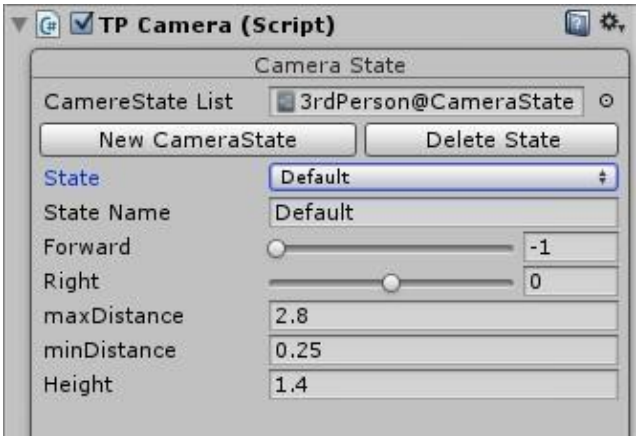


- try add a weapon into the inventory, hit play, equip the weapon and aim (you will see that the character will aim down)
- go to the inspector and find the **ShooterManager**, hit the button **LockCamera** and **LockAiming**

- to fix the position of the weapon and arms, change the transform of the **defaultEquipPoint** and not the weapon itself or the renderer inside, start by rotating the defaultEquipPoint so the IK can work to aim forward, then position the weapon into the hand.
- copy the transform component of the defaultEquipPoint and paste the values after exiting PlayMode, now all the weapons should be align correctly.
- Now to adjust the **LeftHandIK Rotation and Position OffSet** go to the **ShooterManager** and insert values until you get the results you need (each character rig will use different values). Don't forget to copy and paste this values after exiting PlayMode.

c) Creating A New Camera State

In the Third Person Camera you can create new CameraStates to manage different values, states like “Default”, “Aiming”, “Crouch”, to set up new camera position, distance, height, etc.



Fig,3.18 Camera

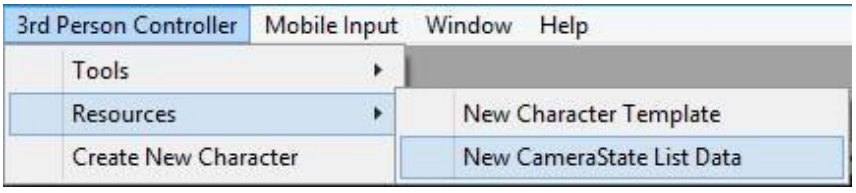
Then just change the CameraState on the method ControlCameraState() on the script TP_Motor.

Example:

if(aiming) tpCamera.ChangeState *("Aim", true);*

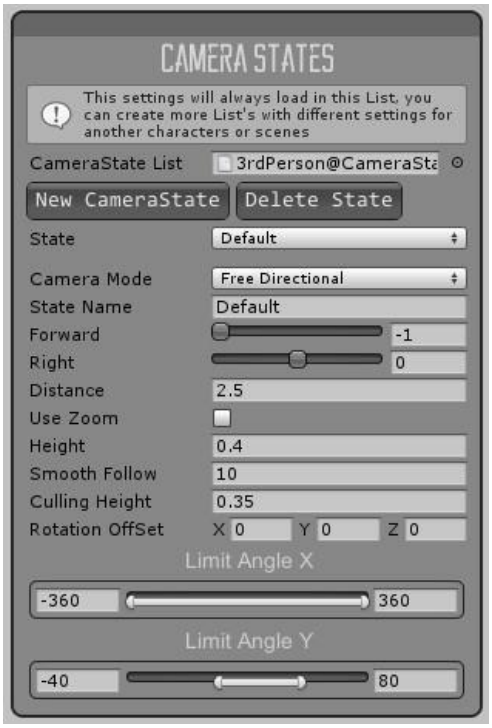
The first string value is the State Name that you created on the Camera Inspector, the second value is a bool, leave it true if you want a smooth transition to this state or false if not.

If you have more than one character and want to use different States, you can create a new **CameraState List Data** here (pic below) and assign on the CameraState List field on TP Camera Inspector.



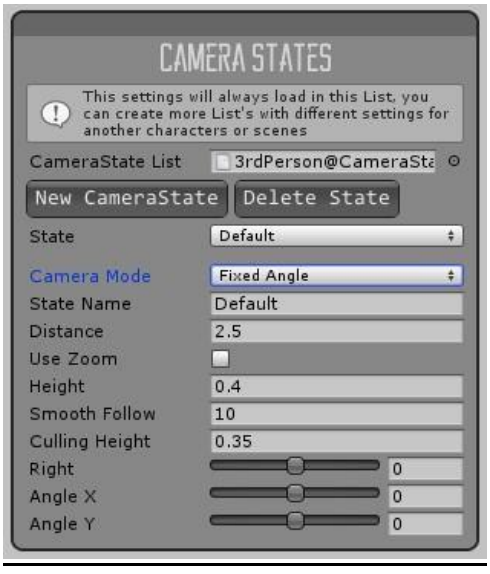
d) CameraMode – Free Directional

This CameraMode offer a free directional – orbital around the character, with a lot of options to customize and make over the shoulders, or above the character, zoom (mouse only) etc…



e) CameraMode – Fixed Angle

This is a feature to use for Isometric or Topdown games, you can set up a fixed rotation for the camera and make games like Diablo or MGS 1.

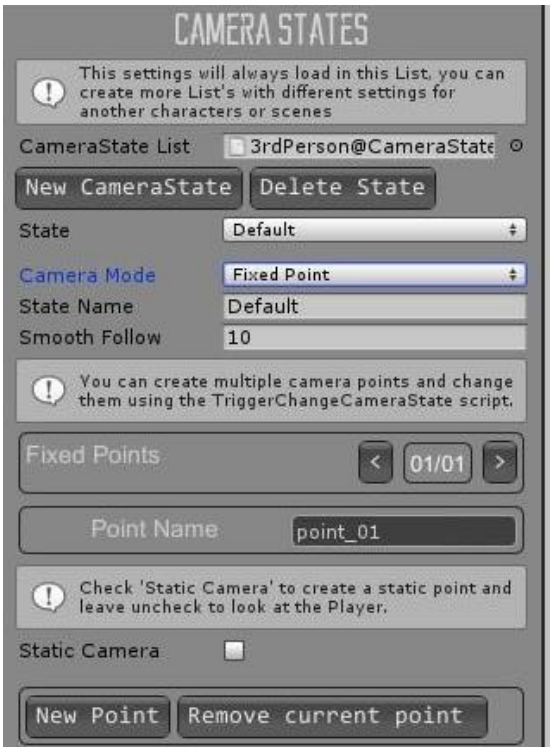


Fig,3.19 Camera state

f) CameraMode – Fixed Point

Fixed Points are states that you can create to use the Camera as a CCTV mode (Oldschool Resident Evil series), this state will follow the character by default or you can check Static Camera to make it fixed.

You can also create multiple points and change with the **TriggerChangeCameraState** that has an option for smooth transition between points or not. *always leave a safe-space between triggers



3.2.1 Design and Creating AI

All living organisms share NEURAL SYSTEMS that enable them to interact with and interact with the surrounding environment and help them control the vital processes needed to sustain the life of these organisms. Neural systems vary from one organism to another where they are simple to install and normal to work in.

Primary organisms with simple cytoskeletal structure, complex structure and the nature of work in higher organisms such as humans.

It is the nervous system of man The most complex neurological systems at all, which are concentrated mostly in the human brain, which characterizes the nature of work that led to superiority of the human other creatures in the ability to understand and recognize the forms and symbols and learning and talking and remembering and recognition and control of the precise

motor and so many qualities and capabilities that cannot Any other object other than a human being has access to it.

He also knows human intelligence

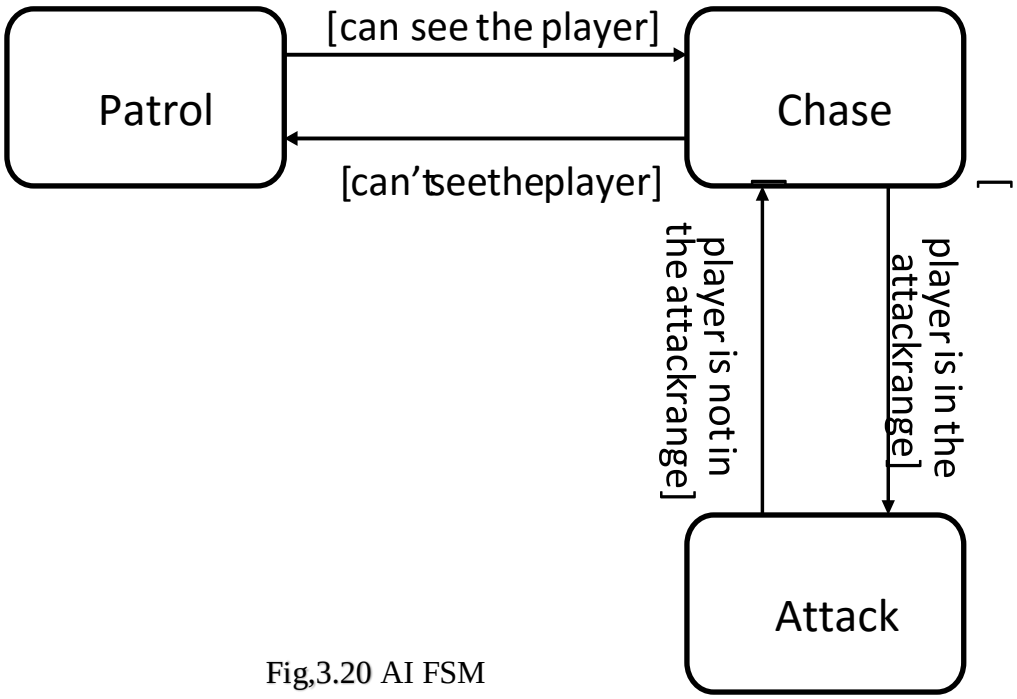
(Human Intelligence) is the ability and skill to develop and find solutions to problems (using the symbols SYMBOLS) and the different problem search methods (KNOWLEDGE) and the ability to use EXPERTIES in deriving new information and knowledge that lead to solutions to problems in a specific field, the level of intelligence varies from person to person and human intelligence is responsible for the development and creativity in the growth of different civilizations.

In view of the importance of human intelligence, man has always been searching for the nature of this intelligence and how it can be measured and set the steps to simulate its methods in the form of software using computers. The study of human intelligence has long been confined to psychologists, but rapid progress in all branches of science in the latter half of the century has led to the contribution and cohesion of many sciences (physiology, biology, mathematics, physics, computers, philosophy and linguistics) Human intelligence and development, researchers have hoped to transfer the methods of innate intelligence and experience acquired by humans

Artificial intelligence is the machine's ability to simulate the human mind and how it works, such as its ability to think, discover and benefit from past experiences. Since the computer's development in the mid-20th century, it has been discovered that computers can perform tasks that are more complex than we thought, Complex sports, in addition to his ability to play chess skillfully, but despite the many advantages of the speed of processing and high storage capacity, but now there is no program can meet the flexibility of the human mind, especially as related to the tasks that involve on the other hand, there are some applications that have been able to match the level of performance of experts and professionals to perform specific tasks, and limited applications that artificial intelligence was able to do is medical diagnosis, computer search engines and ability to recognize the voice and handwriting.

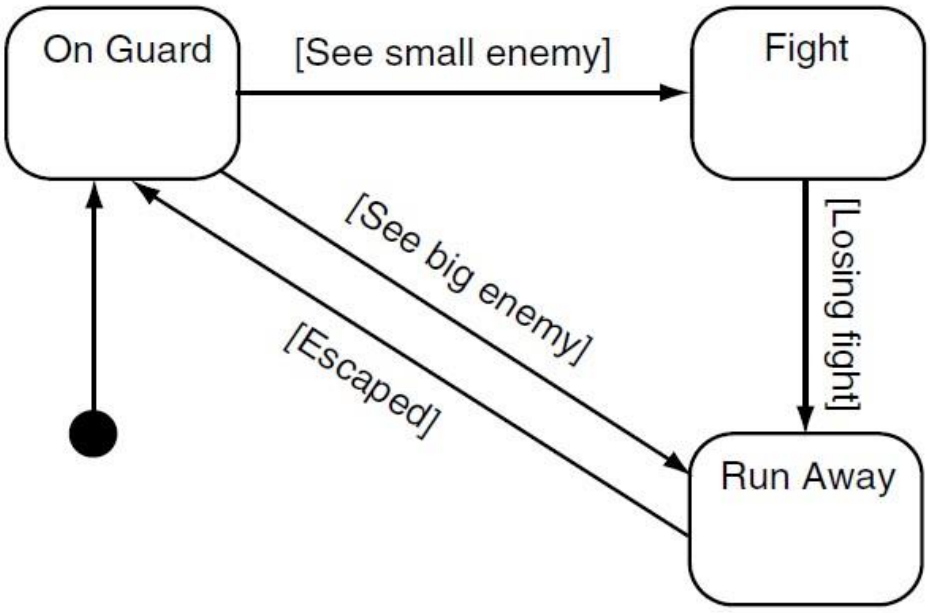
a) FiniteStateMachines

- Usually, game characters have a limited set of possible behaviors. They carry on doing the same thing until some event or influence makes them change.
  - Example: a guard will stand at its post until it notices the player, then it will switch into attack mode, taking cover and firing.
- State machines are the technique most often used for this kind of decision making process in games.
- What is a state machine?
- Actions or behaviors are associated with each state.
- Each transition leads from one state to another, and each has a set of associated conditions.
- When the conditions of a transition are met, then the character changes state to the transition's target state.
- Each character is controlled by one state machine and they have a current state.



Fig,3.20 AI FSM

b) Unity – Implementation

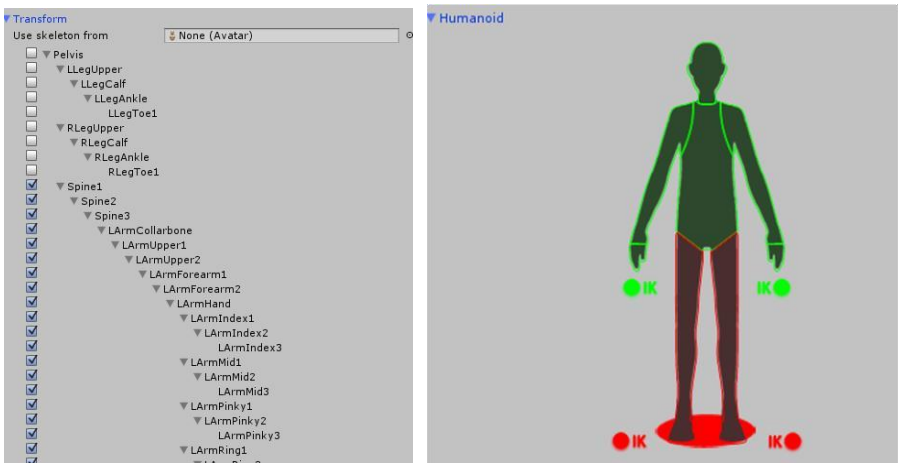


c) Initial Preparation

1.Your model must be fully modelled, rigged, and animated.

2.You must have created a ragdoll object for your model. The easiest way to do this is usually with the unity ragdoll wizard. The Unity documentation has some pages to help you get through this process.

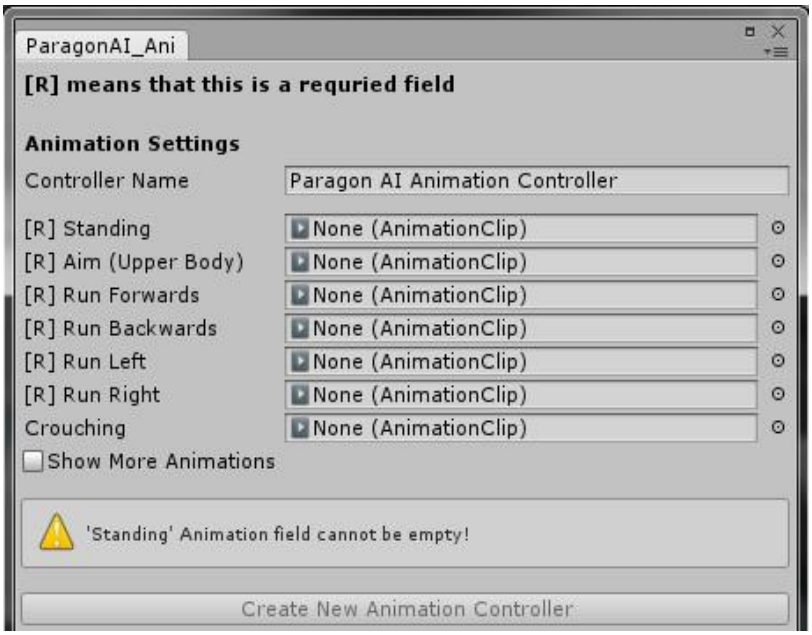
3.You must have created a mask for your model that excludes the legs. This is so that we can play movement animations on the lower body while simultaneously allowing the upper body to play animations such as reloading or firing. It will also allow our agent to aim their gun towards the target. Examples for both generic and humanoid rigs are as follows.



4.You must create an "eye" object and make it an object within the ragdoll's hierarchy. The forward axis (represented by the blue arrow in the scene view in local space) must be facing forwards. If you don't get it right though, you can always change it later.

d) Creating. The Animation Controller

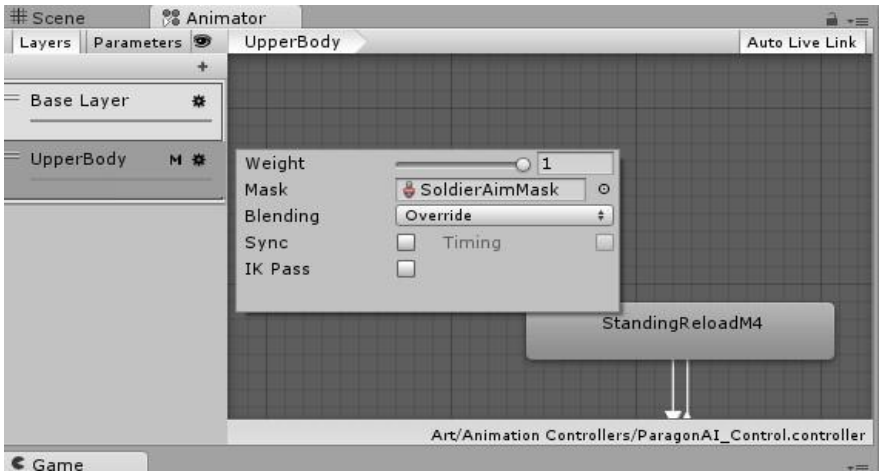
1.Ntavigate to Assets > Create > Tactical AI Animation Controller. This should bring up the following window:



Fig,3.21 Animation controller

2. Select the appropriate animations and give the controller an appropriate name. Then, click on *Create New Animation Controller*.

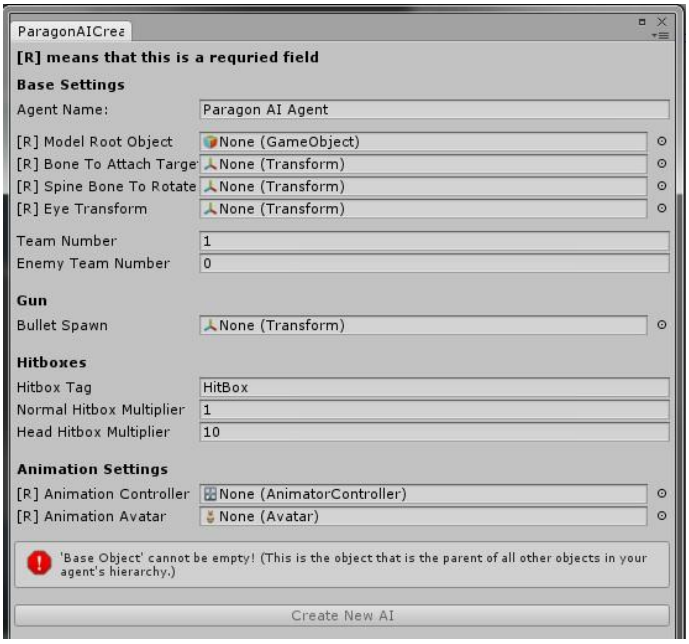
3. Finally open your controller and go to the second (lower) layer. Click on the gear. Set its mask to the one we created above and set its weight to 1. Make sure the blending mode is set to "override."



Fig,3.22 **Animator**

e) Creating The Agent

1. Navigate to *GameObject > 3DObject > Tactical AI Agent.* This should bring up a window similar to the following:



2. Fill out the appropriate fields.

f) Explanations for each field are as follows:

Agent Name: The name you want to assign to your agent.

Model Root Object: The upper-most object in your ragdoll's hierarchy.

Bone to Attach Target to: The transform at which other agents will aim at when targeting this agent.

Spine Bone to Rotate: The bone your agent will rotate in order to aim their weapon at their target

Eye Transform: The transform that will be used to determine where your agent performs line of sight checks. The blue arrow.

Team Number: A number that will identify which team this agent is a part of.

Enemy Team Number: One of the numbers that identify which team this agent will be fighting. If you only want them to fire on the player, the default value of 0 should be fine, unless you have changed the team of the player. If you want the agent to fight more than 1 team, you can manually add this later.

Bullet Spawn: The transform at which your bullets will be spawned at

Bullet Object: The prefab that is your enemy's bullet

Bullet Sound: The audio clip that will be played every time your agent fires.

Hitbox tag: The tag that each of your hitboxes will use. Every collider on the object will be transformed into a hitbox.

Normal Hitbox Multiplier: The factor by which incoming damage will be multiplied, by most hitboxes, by default. You can change the value of any individual hitbox later.

Head Hitbox Multiplier: The factor by which incoming damage will be multiplied by the head hitbox

Animation Controller: The animation controller this agent will use.

Animation Avatar: The avatar this agent will use.

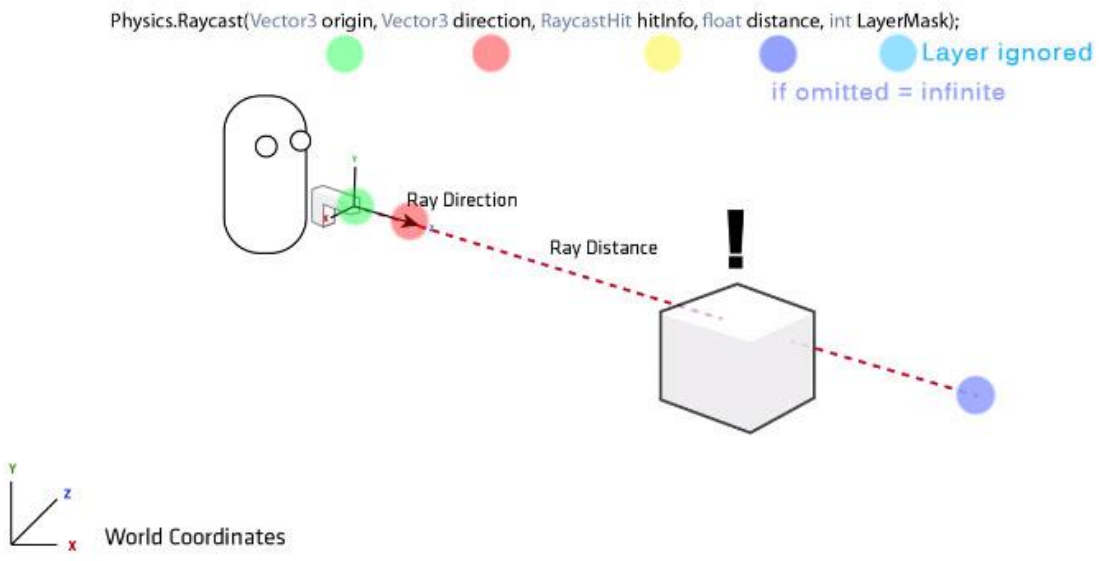
When you've filled out the appropriate fields, click *Create New AI*. Make sure to save it as a prefab when you are done.

g) Adding the Ai Controller

In order for Tactical AI to work, you'll need an AI Controller in your scene. Simply create an empty Game Object, and add the Controller Script. Once there, change the layer mask variable to include all the layers which your level parts belong to. For best results, the level parts should be on a different layer than your characters or agents.

i) How the enemy sees the player or any game object?

Our agent to detect the player using line of sight, which is literally a line in the form of a raycast. A raycast is a feature in Unity that allows you to determine which objects are intersected by a line cast from a point toward a given direction. While this is a fairly efficient to handle visual detection in a simple way, it doesn't accurately model the way vision works for most entities. An alternative to using line of sight is using a cone-shaped field of vision



Fig,3.23 Raycast

ii)script raycasting

```
RaycastHit hit;
// get the forward vector of the player's camera
Vector3 fwd = transform.TransformDirection(Vector3.forward);
//create a ray using fwd vector as direction and a max size of 20.0f
//hit is the out parameter if something is detected
if (Physics.Raycast (transform.position, fwd,out hit, 20.0F)) {
//if there something in our front, check if it's the monolith
if(hit.collider.gameObject.name.Equals("monolith"))
print(hit.collider.gameObject.name);
}
```

iii) Custom Behaviours

Sometimes, you'll want your agents to do something that is not covered by the default behaviors. To do this, you'll be able to create your own behaviors by extending the **TacticalAI.CustomAIBehaviour** class. By default, it should look something like the following:

```
using UnityEngine;
using System.Collections;

namespace ParagonAI{
public class CustomAIBehaviourTemplate : ParagonAI.CustomAIBehaviour{


    public override void Initiate()
        {
            base.Initiate();
        }

    public override void AICycle()
        {

        }

    public override void EachFrame()
        {

        }

    public override void OnEndBehaviour()
        {

        }
}
}
```

iv) Inherited methods

Initiate() : Automatically called when the behavior begins. If you want all references to be automatically set, you need to call base.Initiate(); in this method.

AICycle() : Automatically called every AI cycle. The frequency can be modified in the **BaseScript**.

EachFrame() : Automatically called every frame.

OnEndBehaviour() : Automatically called when the AI changes its behavior, or dies.

ApplyBehaviour(): Call this method to make the **BaseScript** use this behaviour. The **BaseScript** in question must be attached to the same object as the behavior.

KillBehaviour(): Call this method to destroy this behavior. Automatically calls **OnEndBehaviour().**

Inherited variables

targetVector: This automatically provides the BaseScript with a destination to move to.

behaveLevel: Whether this is an Idle or a Combat Behavior.

baseScript: This behavior's **BaseScript**

gunScript: This behavior's **GunScript.**

soundScript: This behavior's **SoundScript**.

rotateToAimGunScript: This behavior's **RotateToAimGunScript .**

animationScript: This behavior's **AnimationScript**.

coverFinderScript: This behavior's **CoverFinderScript**.

myTransform: The transform of the agent this behavior is attached to.

agent: This agent's NavMeshAgent component.

layerMask: The layermask used by this agent for line of sight purposes .

3.2.2) How to make an agent use a custom behaviour?

To make an agent use a behavior via the editor:

1. Add your custom behavior component to the same object that holds the baseScript of your agent.
2. In the inspector for your custom behavior, set behaveLevel to either Idle or Combat, depending on what you want it to do.
3. In the inspector for the baseScript:

   If your behavior is an Idle Behavior, set the myIdleBehaviour variable to "*Custom*".

   If your behavior is a Combat Behavior, set the myAIType variable to "*Custom*".

To change an agent's idle or combat behavior to a custom one during runtime:

1. Add the custom behavior component to the same object that holds the baseScript of your agent. (this can be done during runtime with another script or in the editor).

You can use the following method during runtime:

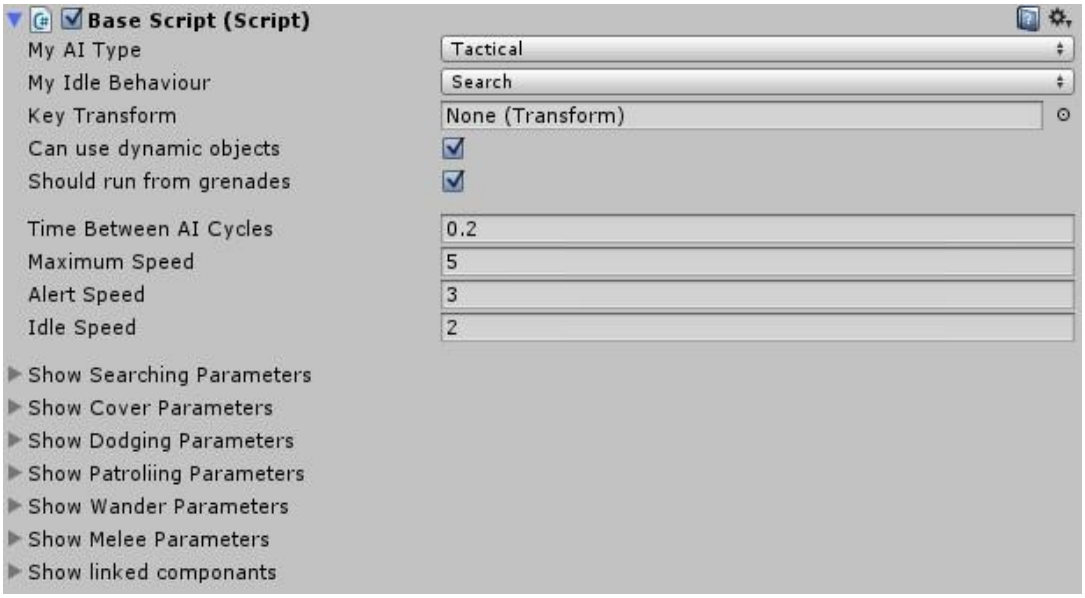
2. During runtime, set the behavior's behaveLevel to either Idle or Combat.
3. Call the **ApplyBehaviour()** method on your behavior component.

3.2.3) AN Agent's Scripts

Note that some scripts use custom inspectors. As such, they may not update in real-time during gameplay. You will have to set the inspector to run in debug mode in order to see the variable's real-time values.

a) Base script

This script is the heart of the system; it holds everything else together. It's primarily used to change



b) Variables

My AI Type: Determines the AI's behavior in combat.

My Idle Behavior: Determines the AI's behavior when not engaged in combat.

Key Transform: A transform used by a number of behaviors.

Can use dynamic objects: Whether or not this agent can use dynamic objects.

Time between AI Cycles: The time in between AICycle calls on behaviors. A low number will give higher quality behaviors, but a higher number will offer better performance.

Maximum Speed: The maximum speed the agent can move at. This speed will be reached in active combat.

Alert Speed: The speed the agent will move at when they are aware of an enemy, but don't know their exact position.

Idle Speed: The speed the agent will move at when they are completely unaware of an enemy.

Radius to call off search: How close the agent needs to get to the target position, before setting its target position to the agent's current position.

Time between cover safety checks: How often the agent will check to make sure their current cover location is safe.

Maximum time in cover: The maximum time the agent will spend in cover before choosing a new location.

Minimum time in cover: The minimum time the agent will spend in cover before choosing a new location. The exact time will be a randomly chosen number between the minimum time and maximum time. The agent may leave early due to external factors, such as the location not being deemed safe anymore.

Min Skirmish Dist: The minimum distance the agent will attempt to be from the target when using the Skirmish behaviour

Max Skirmish Dist: The maximum distance the agent will attempt to be from the target when using the Skirmish behavior .

Maximum time in cover: The maximum time the agent will spend in cover before choosing a new location.

Minimum time in cover: The minimum time the agent will spend in cover before choosing a new location. The exact time will be a randomly chosen number between the minimum time and maximum time. The agent may leave early due to external factors, such as the location not being deemed safe anymore.

Min Skirmish Dist: The minimum distance the agent will attempt to be from the target when using the Skirmish behaviour .

Max Skirmish Dist: The maximum distance the agent will attempt to be from the target when using the Skirmish behavior.

Can Cross Behind Target: Enabling this will make a Skirmishing agent take a more circular path around an enemy rather than possibly going dangerously close to them. Disabling this can be good for open areas where you want the agents to give the target a wide berth. However, disabling this will hinder them when they try and flank enemies in tight spaces or behind cover.

Max Time To Wait At Each Skirmish Pos: The maximum time each agent will spend at each new skirmishing location before seeking a new one.

Should dodge: Whether the agent should try and dodge **Dodging Speed**: The speed the agent will move at when dodging.

Dodging time: How long each dodge will take.

Dodging Clear Height: When trying to dodge, the agent will check to see if the direction of their strafe is clear.

This variable determines how far off the ground obstructions can be without preventing a dodge from occurring.

Time between dodges: The minimum time between dodges.

Minimum distance from target to dodge: The closest distance the agent can be to their target and still dodge,

Close enough to patrol node distance: How close the agent has to get to a patrol node before proceeding to the next one.

Patrol Nodes: An array that contains the agents patrol nodes. The agent will start at the first node on the list and work their way down. At the end, they will cycle back to the first node. Patrol nodes are represented by transforms. It's recommended you use empty game objects for this purpose.

Show patrol path: Whether or not the patrol path should be visualized in the editor.

Wander Diameter: The diameter of the area in which a random wander destination can be chosen. The center is the agent's current position, or their key transform's position, if the agent has one.

Dist to choose next wander point: How close the agent must get to the current wander destination to choose another one.

Can melee: Whether or not this agent can perform melee attacks.

Melee damage: How much damage is inflicted by each melee attack.

Time between melees: How long an agent must wait between melee attacks.

Melee range: How close an agent must be to a target in order to perform a melee attack.

Time until damage: How long after the animation starts the damage should be dealt. This will allow you to sync up the damage with the actual strike in the animation.

Gun Script: This agent's **GunScript**.

Audio Script: This agent's **AudioScript**.

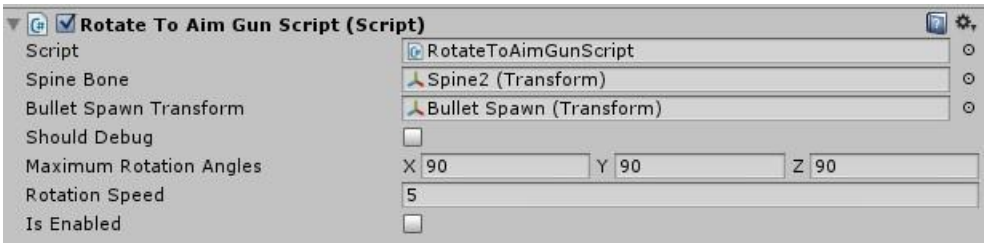
Head Look Script: This agent's **RotateToAimGunScript**.

Animation Script: This agent's **AnimationScript**.

Cover Finder Script: This agent's **CoverFinderScript**.

c) Rotate to aim gun script

This script rotates a bone in the agent's spine so that they aim their gun at the target.



Variables

Spine Bone: The bone the AI should rotate in order to aim their gun towards the target.

Maximum Rotation Angles: How far the agent can rotate their spine

Bullet Spawn Transform: The transform the AI should orient towards their target.

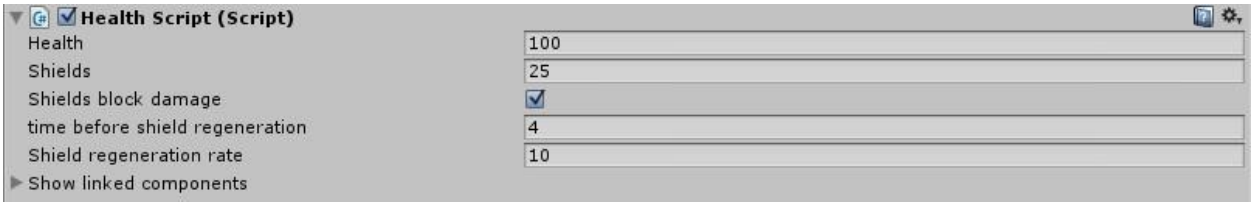
Should Debug: Whether or not the Scene Window should visualize debug information. The blue line points directly from the gun spawn to the target. The red line shows the gun's actual forwards vector. Note that, while the two should get close, they will likely never perfectly line up.

Rotation Speed: How fast the agent can rotate towards their target.

Is Enabled: Whether or not the agent should rotate so that the gun aims towards the target.

d) Health script

This script controls everything health related.



Variables

Health: How much health the agent has. When this reaches zero, the agent will die.

Shields: A second layer of health. When this reaches zero, the agent will behave more defensively until they regenerate. For example, they will not move out of cover to fire at enemies.

Shields block damage: Whether or not the shields must be at zero before damaging health. If true, damage will be applied to the shields until they are down, and then damage will be applied

to health. If false, damage will be applied to both the shields and the health at the same time. The only purpose the shields will serve is to determine when the agent should act more defensibly.

Time before shields regeneration: How long the agent must wait after taking a hit before regenerating. The time will reset every time the agent is hit.

Shield regeneration rate: How many units of shields per second the agent will regenerate.

My AI Base Script: This agent's **BaseScript**.

My Target Script: This agent's **TargetScript**.

Rigidbodies: An array containing the rigidbodies that are part of this agent's ragdoll.

Colliders to Enable: An array containing the colliders that should be enabled on this agent's death. Good for gun models and other objects that shouldn't be hit by bullets but should use physics upon death.

Sound Script: This agent's **SoundScript**.

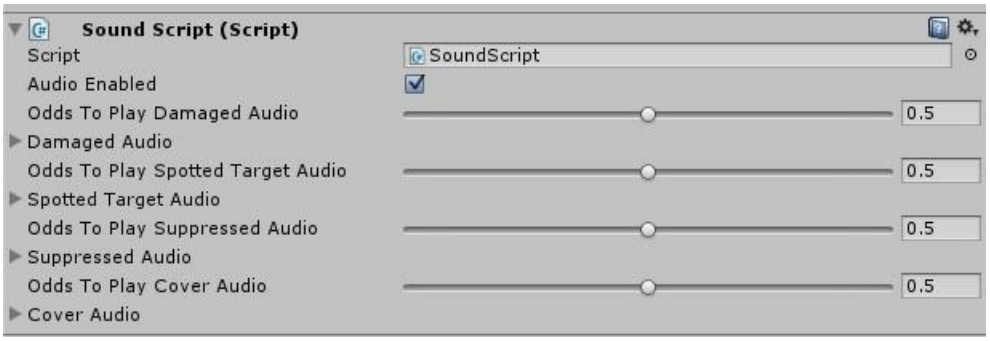
Rotate To Aim Gun Script: This agent's **RotateToAimGunScript**.

Animator: This agent's animator.

GunScript: This agent's **Gunscript**.

e) Sound script

This script will let you play various sounds during several events during an AI's life cycle.



Variables

Audio Enabled: Whether or not this script can play audio.

Odds to Play Damaged Audio: Odds of the agent playing any damaged audio clip when they receive damage.

Damaged Audio: A list damaged audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to Play Spotted Audio: Odds of the agent playing any spotted audio clip when they see a target.

Spotted Audio: A list spotted audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to **Play Suppressed Audio**: Odds of the agent playing any suppressed audio clip when their shields reach zero.

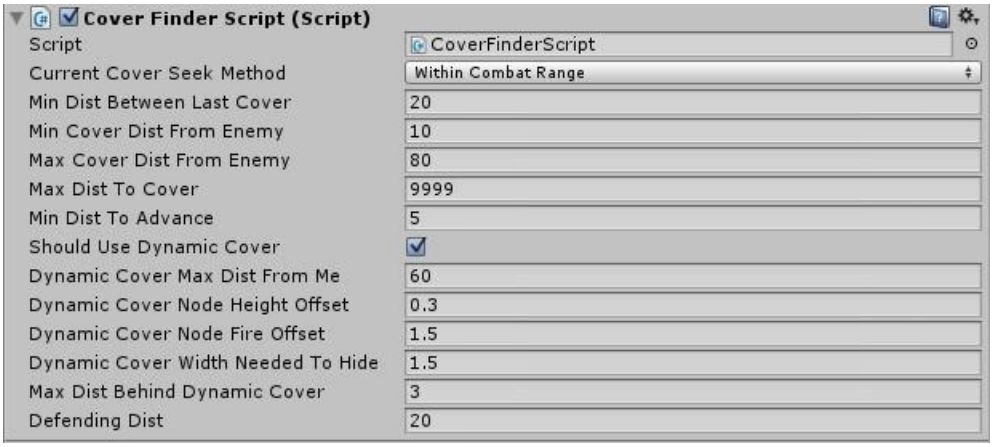
Suppressed Audio: A list suppressed audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to Play Cover Audio: Odds of the agent playing any cover audio clip when they are using cover.

Cover Audio: A list cover audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Cover finder script

This script is used to find suitable cover locations.



Variables

Current Cover Seek Method: The method that will be used to find cover. There are three options to choose from.

> *Random Cover:* The agent will pick a completely random piece of valid cover. If this agent's baseScript has a keyTransform, the agent will attempt to find cover that is closer to it than this agent's defendingDist.

Within Combat Range: The agent will pick the closest piece of valid cover to them that is within their designated combat range. If this agent's baseScript has a keyTransform, the agent will attempt to find cover that is closer to it than this agent's defendingDist.

Advance Towards Target: The agent will generally move in the direction of the target, taking cover along the way. If this agent's baseScript has a keyTransform, the agent will attempt to advance towards this position. This method DOES NOT work with dynamic cover.

Min Dist Between Last Cover: The minimum distance an agent must move after leaving a cover location. Only used with the "*WithinCombatRange*" cover seeking method.

Min Cover Dist From Enemy: The minimum distance cover mist be between the agent and their target. Only used with the "*WithinCombatRange*" cover seeking method.

Max Cover Dist From Enemy: The maximum distance cover mist be between the agent and their target. Only used with the "*Within Combat Range*" and "*Advance Towards Target*" cover seeking methods.

Max Dist To Cover: The maximum distance cover can be from the agent. Only used with the "*WithinCombatRange*" cover seeking method.

Dynamic Cover Selection: The method that will be used to dynamic cover. There are two options to choose from.

Navmesh Scan: The agent will use the navmesh to find cover. This can have poor performance on large maps, but will usually have better results.

Raycasts: Better suited for larger maps, but agents will usually only use the cover they find directly around them, they won't use the map as effectively as other methods.

Should Use Dynamic Cover: Whether not this agent can dynamically find cover.

Dynamic Cover Max Dist From Me: How far this agent can travel to find navmesh cover. Generally a shorter distance than **MaxDistToCover**.

Dynamic Cover Node Height Offset: How high off the ground line of sight checks for cover safety should be performed. This should be roughly as high as your character's head when crouching.

Dynamic Cover Node Fire Offset: How high off the ground line of sight checks for cover firing should be performed. This should be roughly as high as your character's weapon when standing.

Dynamic Cover Width Needed To Hide: When using tall cover, how far to the side the agent must move to be safe. Should be a little wider than the width of your agent

Max Dist Behind Dynamic Cover: How far behind a line-of-sight blocking object your agent can be at most.

Defending Dist: How close to the agent's Key Transform your agent should take cover when using the "*Within Combat Range*" combat mode or dynamic cover.

Angle Between Casts: Increasing this value improves performance but makes it less likely that the agent will find cover.

Max Ray Dist: The maximum cover finding ray length.

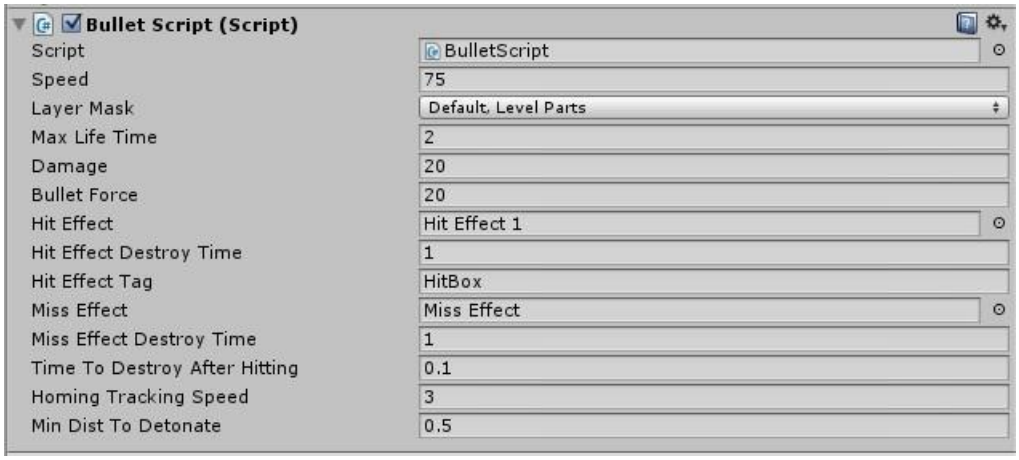
Ray Cast Height Off Ground: How high from the ground the cover checking rays could be.

Dist From Wall To Be: The minimum distance an agent will try and by from the wall they are taking cover from. If agents using raycast cover are not crouching, try increasing this value.

3.2.4 Damage Scripts

a) Bullet script

This script is used for AI bullets and missiles.



Outgoing methods

Damage(float damage): Called on the object the bullet hits.

Variables

Speed: How fast the bullet will move.

Layer Mask: The layers of the object this bullet can hit.

Max Life Time: The time until the bullet will automatically destroy itself.

Damage: How much damage this bullet will do.

Bullet Force: How much force this bullet will apply to rigidbodies upon hitting them.

Hit Effect: The prefab that will be created when it hits the object marked by the hit effect tag.

Hit Effect Destroy Time: How long until the hit effect is automatically destroyed.

Hit Effect Tag: The tag that marks objects which the bullet will create a hit effect upon impact.

Miss Effect: The prefab that will be created when the bullet hits anything without the hit effect tag.

Miss Effect Destroy Time: How long until the miss effect will automatically be destroyed.

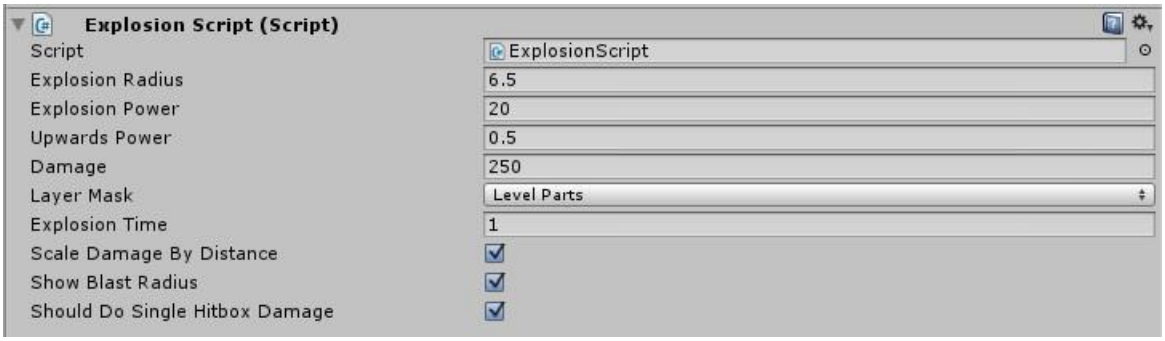
Time to Destroy after Hitting: How long the bullet will exist after hitting an object. Good for making effects such as bullet trails linger after impact.

Homing Tracking Speed: If this bullet's gunscript is designated as firing a missile, this determines how quickly the bullet will rotate to move towards its target.

Min dist to Detonate: If this bullet's gunscript is designated as firing a missile, this variable determines how close to the target the bullet must be to instantiate its hitEffect and destroy itself.

b) Explosion script

This applies damage and force for explosions.



Outgoing methods

SingleHitBoxDamage(float damage): Called on all colliders caught in the explosion.

Damage(float damage): Called on all colliders caught in the explosion.

Variables

Explosion Radius: The maximum distance objects can be from the explosion and still take damage.

Explosion Power: How much force should be applied to objects caught within the blast.

Upwards Power: How much upwards force should be applied to objects caught within the blast.

Damage: How much damage the explosion does.

Layer Mask: Layers of objects that can block the explosion.

Explosion Time: How long the explosion should exist before being automatically destroyed.

Scale Damage by Distance: Whether or not the amount of damage done to a target should scale by distance.

Show Blast Radius: Whether or not the blast radius should be illustrated in the scene view.

Should Do Single Hitbox Damage: If true, the method SingleHitBoxDamage will be called on targets. If false, Damage will be called. If your health script does not have the method SingleHitBoxDamage, this should be false.

c) Explosive barrel script

This script is used to make objects that explode after taking enough damage.

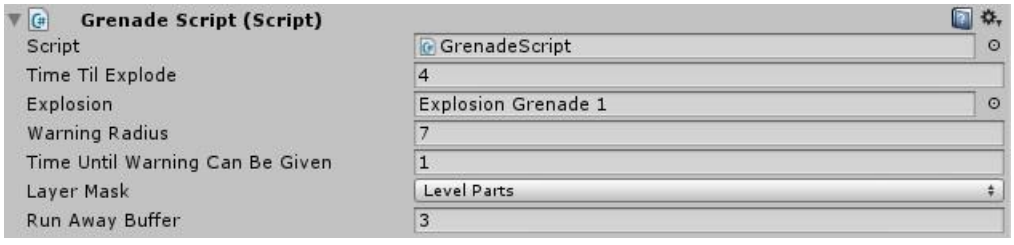
Variables

Health: How much damage the object can take before exploding

Explosion: Prefab that contains this object's explosion.

Grenade script

This script is used for grenades



Variables

Time Til Explode: How long after the grenade is created it takes to explode.

Explosion: The explosion prefab created by this grenade.

Warning Radius: How far away agents can be and be warned of the grenade. Warned agents will attempt to get out of range pf the grenade.

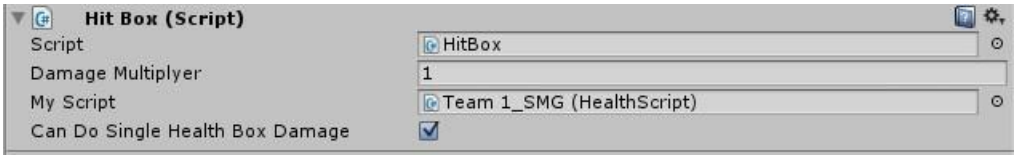
Time Until Warning Can Be Given: How long after the grenade is created until a warning can be given. After the time is up, a warning will be given when it first hits an object.

Layer mask: Layers of objects that can block warning.

Run Away Buffer: Warned agents will be told to run away a distance of the warning radius + the run away buffer.

Hitbox

This script is used to transfer damage and apply it to a health script



Incoming methods

Damage(float damage): Applies damage to the object's **HealthScript**, after multiplying it by the damageMultiplier.

SingleHitBoxDamage(float damage): Applies damage to the object's **HealthScript**, after multiplying it by the damageMultiplier. Only one instance of SingleHitBoxDamage will have an effect per frame. This is used to stop explosions from applying damage multiple times to the same agent across multiple hitboxes.

Variables

Damage Multiplier: How much incoming damage should be multiplied by before being sent to the health script.

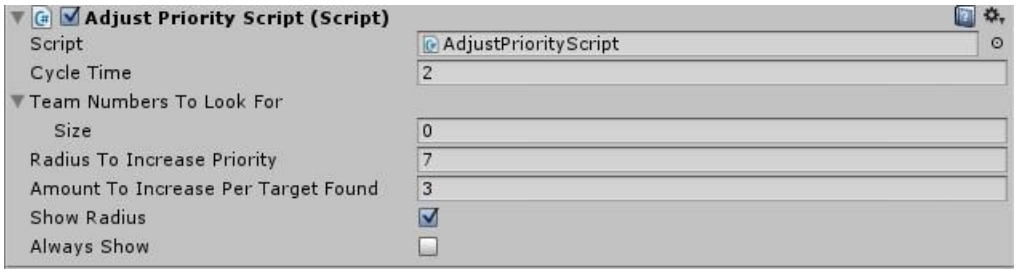
My Script: This agent's health script. This variable can be any MonoBehaviour that has a Damage(float damage) method.

Can Do Single Health Box Damage: Prevents a target with more than one hitbox from taking damage more than once from an explosion. Requires your health script to have Coroutine SingleHitBoxDamage(float damage).

d) Helper Scripts

Adjust priority script

This script adjusts the priority of a **TargetScript** when other targets of a given team are within a certain radius.



Variables

Cycle Time: How often this script will update its **TargetScript**'s priority.

Team Numbers to Look For: Which teams should increase the priority. If this is left blank, it will automatically match the team of its **TargetScript**.

Radius to Increase Priority: How close other targets must be in order to increase the priotity.

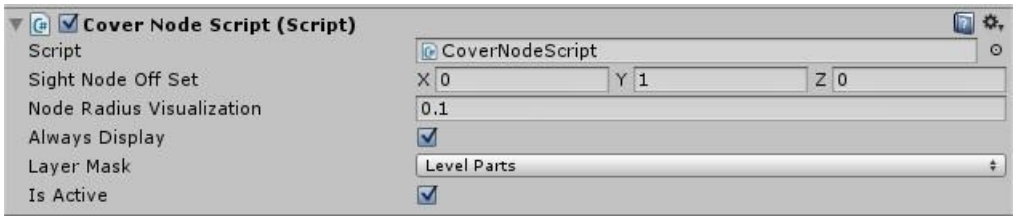
Amount to Increase Per Target Found: How much the **TargetScript**'s priority increases for every target found within the given radius.

Show Radius: Whether the radius should be illustrated in the scene view.

Always Show: Whether or not the radius should always be shown in the scene view, or only when there are targets in range.

Cover node script

This script marks game objects as cover positions.



Methods

ValidCoverCheck(Vector3 targetPosition): Checks whether this node is active and is a valid cover location.

Variables

Sight Node Offset: How much the sight node position is offset by the Cover Node. The position is used for line of sight checks to make sure the agent can return fire from this node.

Node radius visualization: The size of the spheres in the scene view that are used to represent the nodes position.

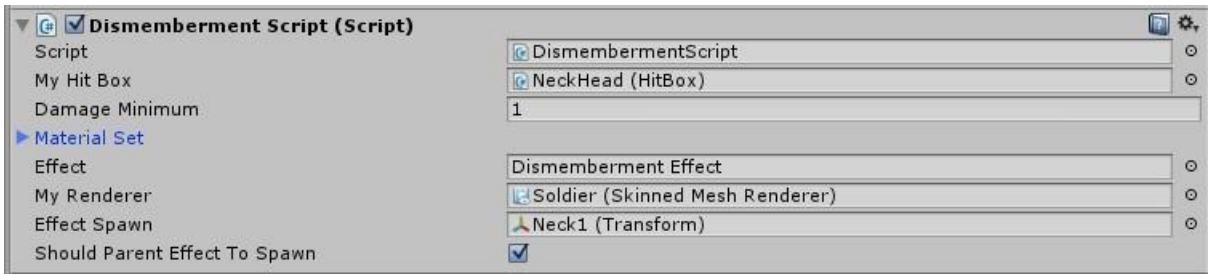
Always Display: If false, the node visualization will only appear when the object is selected.

Layer Mask: Layers of objects that block sight lines.

Is Active: Whether or not this node can be selected.

Dismemberment script

A sample script used to demonstrate the **BaseScript**'s OnAIDeath call. This script can be used to simulate the effect of a part of the AI's body being removed if the AI is killed by enough damage to a specific hitbox.



Variables

My Hit Box: The Hitbox that will be checked. The script will only trigger if the "killing blow" was applied to this hitbox.

Damage Minimum: How much damage must have been done to the hitbox to trigger the dismemberment.

Material Set: What your mesh renderer's material array will look like after this script triggers. This array should be made identical to the renderer's default array, except that the removed areas should be replaced with a 100% transparent material.

Effect: The prefab that will be spawned upon dismemberment. This is usually a particle effect- for example, a shower of sparks if a robot's head is destroyed.

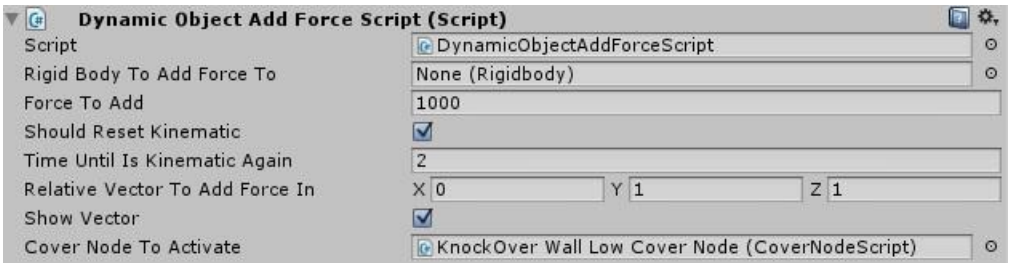
My Renderer: The AI's mesh renderer.

Effect Spawn: The transform at which the effect will be spawned.

Should Parent Effect to Spawn: Whether or not the effect should be made a child of the Effect Spawn.

Dynamic object add force script

A sample script used in conjunction with the dynamic object system. It applies a force to an object in a given vector.



Methods

IEnumerator UseDynamicObject(): Applies the force to the object, enables the cover node, and resets the rigidbody back to kinematic after the specified amount of time has passed.

Variables

Rigidbody to Add Force to: The rigidbody which this script affects.

Force To Add: the magnitude of the force that should be applied to the rigidbody.

Should Reset Kinematic: Whether or not the pushed object should become kinematic again after the force is applied.

Time until is Kinematic again: The amount of time after the force is applied that the Rigidbody should wait before becoming kinematic again.

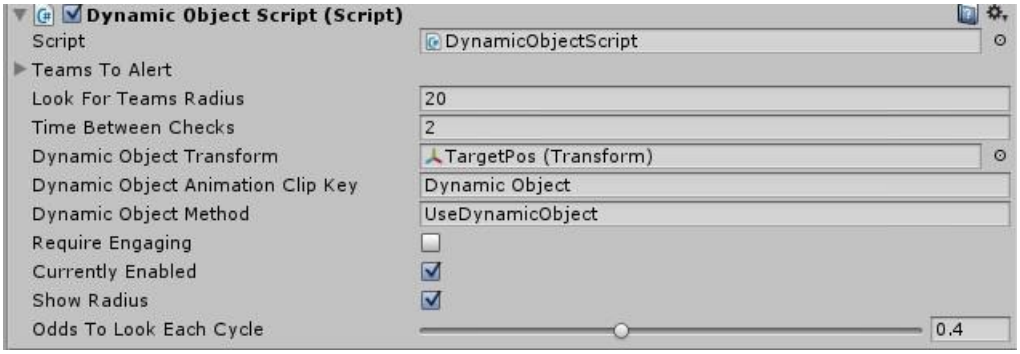
Relative Vector to Add Force in: A vector indicating the direction the force will be applied.

Show Vector: Whether or not the direction of the push vector should be shown in the scene view.

Cover Node to Activate: If you want a cover node to become active after the force is applied, it should be placed here.

Dynamic object script

A sample script used in conjunction with the dynamic object system. The will attempt to get agents within a given radius to use a dynamic object



Variables

Teams to Alert: A list containing the team numbers that this script should affect.

Look for teams radius: The maximum distance the agent can be from this object and still be able to use the dynamic object.

Time between checks: How often this script should check for valid agents

Dynamic Object Transform: The transform of the dynamic object.

Dynamic Object Animation Clip Key: The animation trigger that will be called on the agent when it uses the dynamic object.

If you have multiple controllers for different agents that will be using this object, they must share the same trigger name. You do not need a trigger if you don't want an animation to be played.

Dynamic Object Method: The method that will be called on the dynamic object.

Require Engaging: Whether or not agents must be engaging to use this dynamic object.

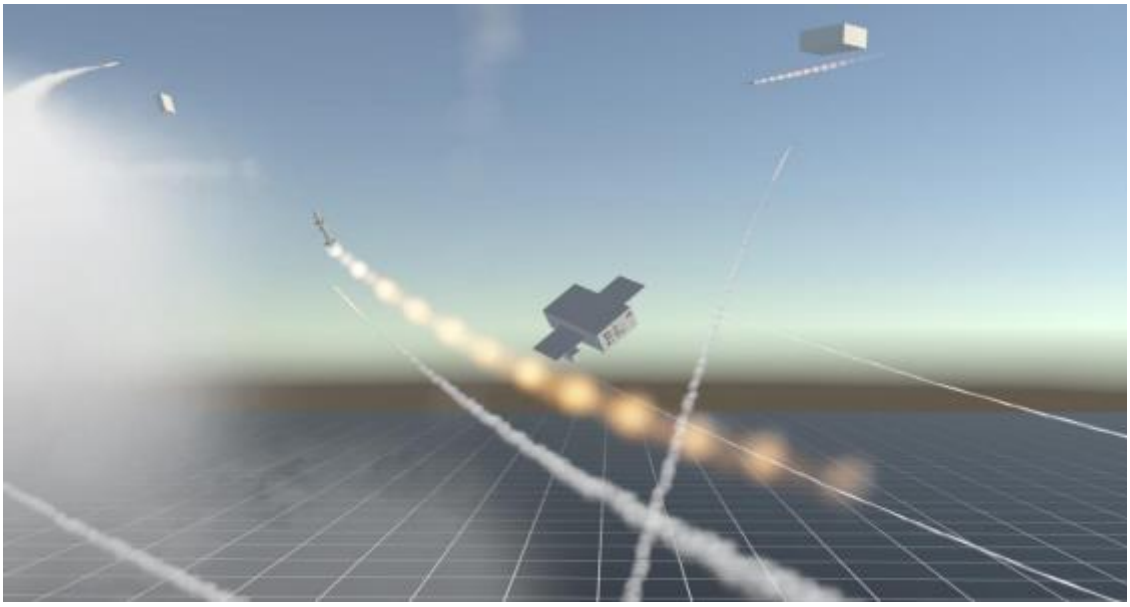
Currently Enabled: Whether or not this object is currently looking for agents to use it.

Show Radius: Whether or not the radius to look for teams should be visualized in the scene view.

Odds to Look Each Cycle: The odds the script will try and select an agent to use the dynamic object each cycle.

Time to Wait: How long the agent should wait at the dynamic object after using it. Should be approximately the length of the animation.

3.2.5 Ace Arcade Missiles



a) Description

Ace Arcade Missiles is a complete code and effects package for creating arcade style missiles.

• Detailed documentation for reference

• Two kinds of guidance: Pure pursuit and Lead

• Support for both Update and FixedUpdate based projects

• Many tweakable missile parameters such as seeker cone, motor acceleration, and turn rate

• Missiles can be set with a delayed activation to allow dropping before firing

• Effects manager that handles both ribbon trails and particle system trails, as well as explosions

• Spawned effects (trails and explosions) automatically clean themselves up

• Audio is handled automatically when assigned an audio clip with adjustable parameters

• Two types of missile launchers: Pod and Hardpoint

• Hardpoints can carry missiles externally, while pods launch from user defined tubes with reloadable magazines

• Pack includes example effects for particle and ribbon trails, basic sound effects, and explosions

b) Major Component Details

This section details the specific settings of each parameter on the user facing included components and anything worth noting to ensure proper operation.

AA Missile

The core of the asset pack, this controls both missile flight and guidance. Missiles can be fired manually if they are already in the scene by calling the "Launch" function on them, but they were primarily designed to be fired from launchers.

c) Required Components

Rigidbody: A missile is required to have a rigidbody, but in most cases it can be ignored as the missile will override certain Rigidbody parameters on start.

This is used for collision detection, and requires that your target have a collider of some kind. If the missile starts to travel fast enough that it clips through targets, you might want to experiment with setting the collision detection to Continuous.

The mass, drag, and angularDrag of the Rigidbody are only used when the missile is dropping.

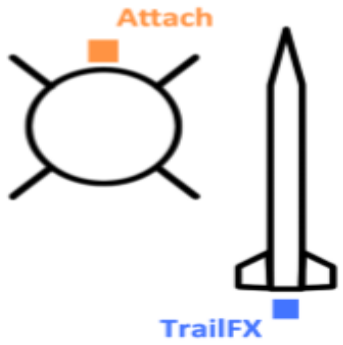
Capsule Collider: Missiles require a capsule collider. It's best to adjust the size and orientation of this yourself to match the missile, as the automatically created one will be default to a 1 meter sphere. Because the capsule collider covers the entire missile, it's best to not have any other colliders on the missile game object.

d) Missile Parameters

Movement Update Cycle: Run missile movement code in fixed update versus update. If you notice jittery movement by the missile during play, try changing this. Update is for projects where all movement is handled through the Update loop. Fixed Update is for physics based projects that rely on rigid bodies, forces, and prominently feature updates through the FixedUpdate loop.

Target Update Cycle: Runs guidance code in the update versus fixed update. Same as above, but for the target's movement. If set incorrectly, lead guidance will likely be thrown off and jitter.

Own Ship: This is typically assigned by the launcher and only needs to be assigned if manually launching a missile already in the scene. Set this to the launching game object (airplane, missile turret, etc.) to prevent the missile from colliding with the launching object. Only necessary to set



Fig,3.24 messel

Attach Point: This is a Transform that the missiles uses as reference for where to attach to a hardpoint launcher. Not necessary for rocket launchers, but should be assigned regardless for good practice. If not assigned manually through the Inspector, you can alternatively create a GameObject called “Attach” and the missile will automatically use that. If neither is done, the missile will use its origin as an attach point.

Guidance Type: Either pure pursuit or lead. Pure pursuit will fly the missile directly towards the target. This is simpler and less expensive, but less effective. Lead on the other hand will have the missile intercept the target, reaching it faster and while being much more difficult to avoid.

Seeker Cone: How far off-boresight the missile can track a target. A wider angle means the missile is more likely to keep lock. If a target maneuvers outside of the seeker code, the missile will go dumb and fly straight until it either hits something or times out.

A missile won’t re-acquire its target if the target is lost. A larger seeker cone also allows the missile to make more dramatic maneuvers when leading the target as it’ll be able to make bigger turns while still keeping the target in sight.

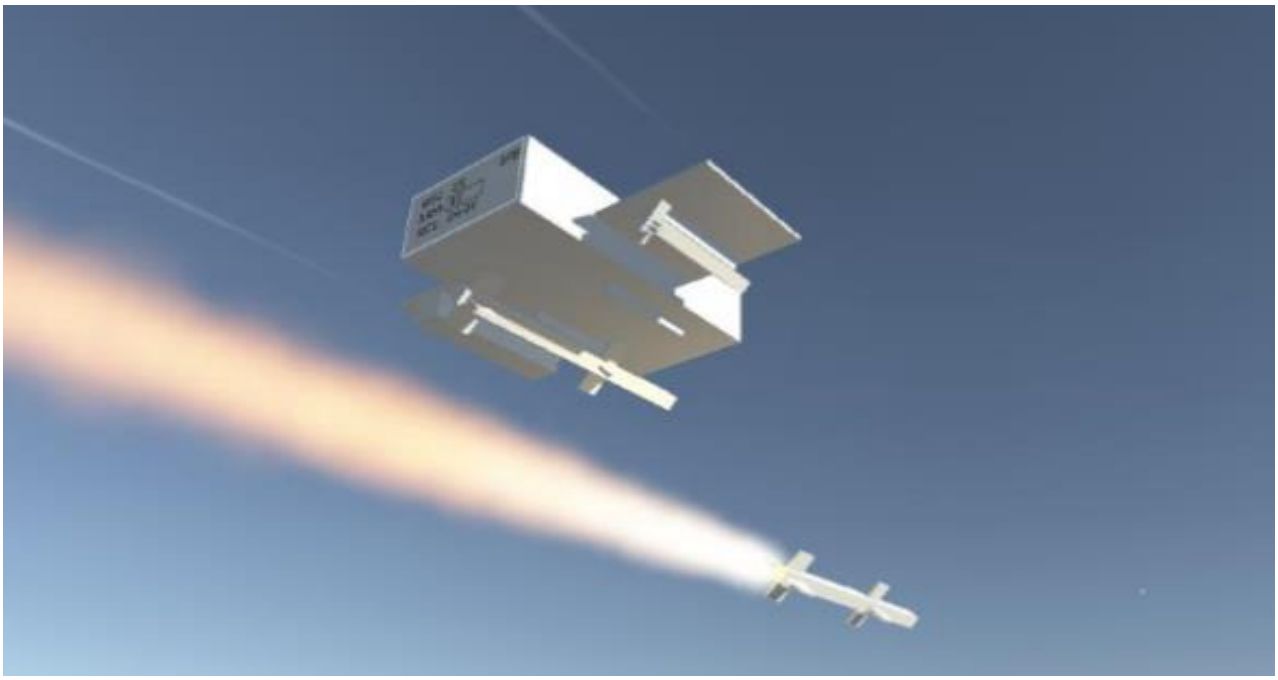
Seeker Distance: How far the missile can track a target. If the target falls outside this range, the missile will go dumb and fly straight until it either hits something or times out.

Override Initial Speed: When true, initial speed will be taken from either the velocity passed into the Launch function, or from the forward velocity of the missile after a drop launch if a drop delay is used. This is useful for missiles that you want to inherit their start speed from their launchers.

Initial Speed: How fast the missile is when it launches. Note that this does not inherit any velocity that the missile might have had before launch.

Motor Lifetime: When nonzero, allows the missile to accelerate for the time specified. Once the motor lifetime is up, the missile will continue on at whatever speed it was when the motor finished. Use this for missiles that you want to accelerate over time.

Acceleration: How much speed per second the missile gains while the motor is active.



One of the two launchers that can be used to fire missiles. This one is designed for external carry and display of missile. The launcher will spawn missiles of the assigned prefab at given launch points. After a missile fires, a new missile will appear after a given reload time. To fire a missile from a launcher, simply call the Launch function on it.

AA Hardpoint and AA Pod both inherit from abstract class AA Launcher, so most parameters are shared between them.

e) Hardpoint Parameters

Own Ship: Assign this to prevent the launcher and missile from colliding with whatever launched/owns it.

Missile Prefab to Launch: The missile game object that will be instantiated by the launcher. While this parameter can be assigned to a missile already on the scene (and thus instantiating a copy of something already in the scene) this can cause issues with duplication of the trail effect. Only use missile prefabs.

Missile Count: Ammo count of the launcher. When it reaches 0, the launcher won't spawn a new missile on the Launch point.

Fire Delay: How long it takes for the missile on the launcher to respawn.

Launch Points: Array of Transforms used as reference positions for the missile to launch. If unassigned, list will be auto populated with child Game Objects that follow the naming scheme "Hp1", "Hp2", "Hp3", etc. If none are found, a single launch point will automatically be created at the launcher's center.

Mixer Group: Associates the generated audio source with the specified mixer group.

Fire Clip: Sound that plays when the hardpoint is fired. This will automatically create an AudioSource with the following values. Note that pitch is a random value between 0.8 and 1.5. This is to create variety in the fire sound for repeated firings.

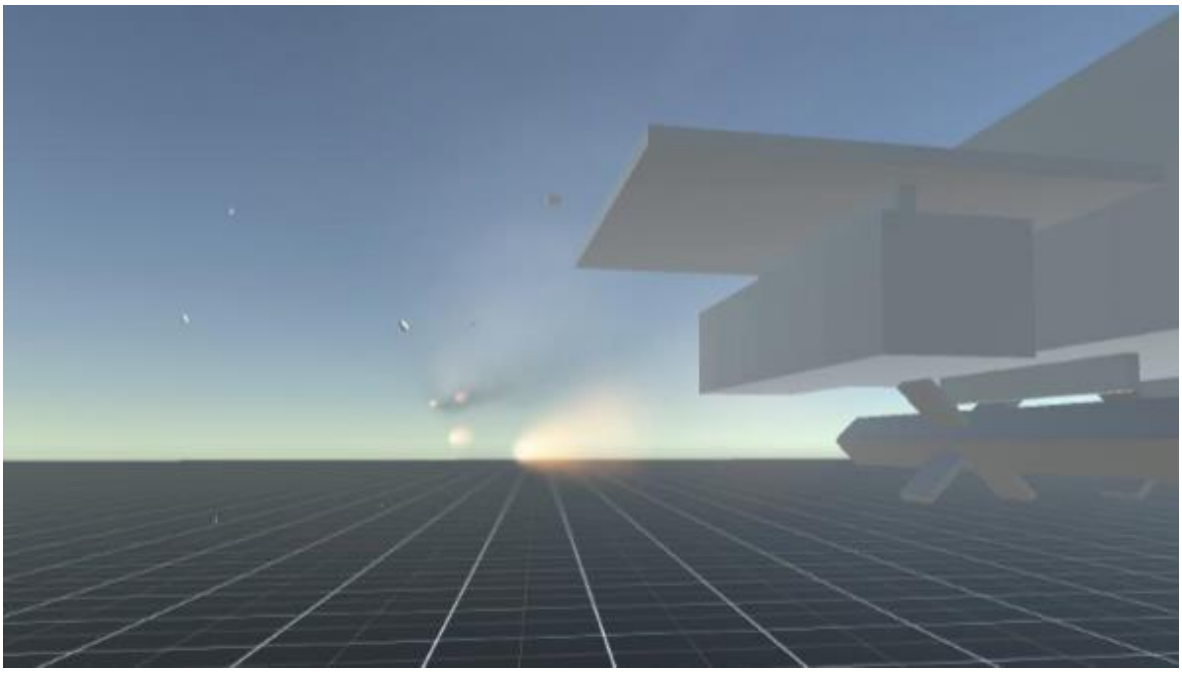
- Output: Taken from Mixer Group
- Mute : False
- Bypass Effects: False
- Bypass Listener Effects: False
- Play On Awake: True
- Loop: False
- Priority: 128
- Volume: Taken from Fire Volume
- Pitch: Random value between 0.8 and 1.5
- Stereo Pan: 0
- Spatial Blend: 1
- Reverb Zone Mix: 1
- Doppler Level: 0

• Spread: 0
• Volume Rolloff: Logarithmic Rolloff
• Min Distance: Taken from Fire Min Distance
• Max Distance: Taken from Fire Max Distance

f) AA POD

The AA Pod simulates a rocket pod style launcher. Rather than the missile physically being placed on a hardpoint, they are dynamically spawned as needed. Pods take advantage of multiple launch points by cycling through each point on each successive firing to simulate different rocket tubes. Pods are unique in having “magazines” with their own size, count, and reload times. Magazine reloading is triggered automatically when all missiles are fired from the pod. Reloading can be called manually on the pod, but all missiles currently in the pod will be lost.

The AA Pod shares most of the same parameters as the AA Hardpoint. Listed below are options specific to the AA Pod.



g) AA Pod Parameters

Dispersion Angle: Randomized angle at which the missile comes out of the pod at. The higher the number, the more spread out the missiles will be.

Missile Count: For the AA Pod, this doubles as how many missiles can be fired before a reload is required.

Magazine Count: Number of times the pod can additionally reload after all missiles have been fired from the first salvo or a manual reload is triggered.

Magazine Reload Time: Time to reload a missile pod magazine. All missiles must be depleted to start a reload. Alternatively, you can manually call the "ReloadPod" function.

Minor Component Details

This section briefly describes the smaller components used by the Ace Arcade Missile Asset Pack to assist other things such as particle effects.

AA Enable Light On Effect Play

On the included trail particle system prefabs, these turn a light on when the effect is playing. It's expected to be used on a child Game Object of the trail effect. It must be specified if the parent Game Object is a TrailRenderer or PartcleSystem.

AA Jitter Direction

Used on particle systems to jitter the emitter in a smooth way. This allows for smoke trails that jitter in direction rather than being perfectly uniform. It approximates the look of minute course adjustments from the missile and variable wind along the path of the missile.

AA Remove Effect

Very basic script that removes a particle system once it's finished emitting. This is used to clean up trails after the missile has impacted, the particle system has been detached from the missile, and it stops emitting new particles.

AA Scrolling Trail

Used on TrailRenderer based missile trails to scroll the texture.

There are many publicly accessible functions in the scripts.

AA Missile

public void Launch(Transform newTarget)

Launch the missile at the given target. If no target is given, the missile will dumbfire. If the missile has a drop delay, use the Launch function with inherited velocity for the correct drop behavior.

public void Launch (Transform newTarget, Vector3 inheritedVelocity)

Launch the missile at the given target with an inherited velocity for correct drop behavior. If no target is given, the missile will dumbfire. It's recommended to use this function in general as it will work for missiles with and without drop delays.

AA Missile Effects

public void Explode ()

Trigger the explosion effect and automatically detach any spawned trail effects.

AA Hardpoint

public override void Launch (Transform target)

Launches a spawned missile at the given target if possible. If no target is given, the missile will fire without guidance.

public override void Launch (Transform target, Vector3 velocity)

Launches a spawned missile at the given target. If no target is given, the missile will fire without guidance. Velocity is used to give a missile with a drop delay an initial velocity. Typical use case would be passing in the velocity of the launching platform.

public override void ResetLauncher ()

If the missile prefab has changed, calling this function will delete the current loaded missiles and replace them with the new ones. Resets the ammo count of the launcher as well.

AA POD

public override void Launch (Transform target)

Launches a spawned missile at the given target if possible. If no target is given, the missile will fire without guidance.

public override void Launch (Transform target, Vector3 velocity)

Launches a spawned missile at the given target. If no target is given, the missile will fire without guidance. Velocity is used to give a missile with a drop delay an initial velocity. Typical use case would be passing in the velocity of the launching platform.

public override void ResetLauncher()

Used to reset the ammo on a rocket pod. Intended to be used when the missile prefab is changed. Also resets the launcher's cooldown, magazine reload cooldown, missile count, and magazine count.

• 3.3 Implementation:

i) Components and Method:

For the purpose of War Game will need to download this package from assets store . this package to help us to create player in our project.

1) Invector

https://assetstore.unity.com/packages/templates/systems/third-person-controller-shooter-template-84583

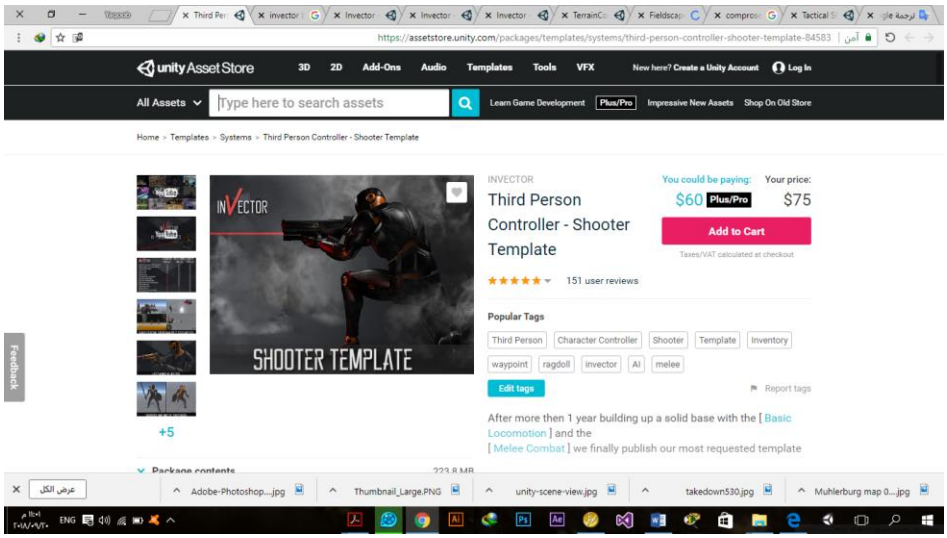
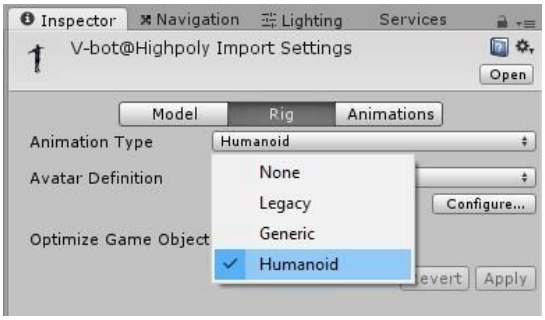
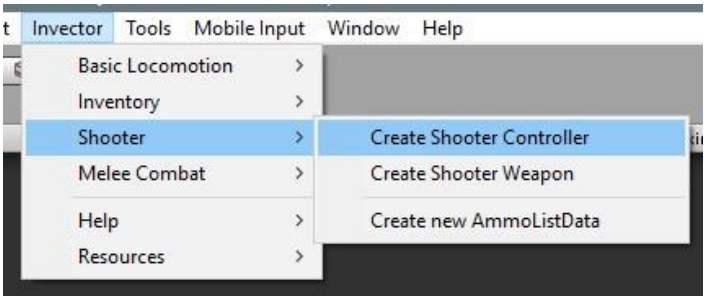


Fig:3.25 Assets Store for Invector package

Make sure that your Fbx character is set up as **Humanoid**



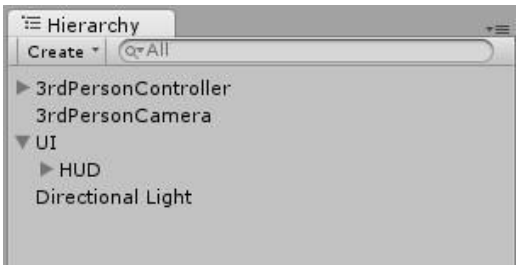
To setup a new character, go to the tab *Invector > Shooter > Create Shooter Controller*



Make sure your Character is **Fully Rigged** and set up the FBX as a **Humanoid**, then assign the FBX to the field “FBX Model” and click on the button “Create”.

Ps* Make sure to select the **Invector_ShooterOnly or Invector_ShooterMelee** if you want to use both shooter and melee as your Animator Controller, you can find the file at the folder: Shooter > Animator, or just click on the little circle icon.

The **Character Creator** window will take care of all the hard work automatically and set up components such as capsule collider, layers, tags, rigibody, etc... It will create the **ThirdPersonController**, **ThirdPersonCamera** and a UI Canvas with a **HUD** to display health, stamina and other information’s.



ii) Ways to control the player by the user.

1 Turn Right (D)

2. Turn left (A)

3. UP (W)

4. Down (S)

5. Car Enter (E)

6. Exit car (ENTER)

7-Speed car (C)

8. Take the weapon (E)

9. Toolbar sniper (right mouse button + Z)

10. Movement of somersault (Q)

11. Crosh (C)

12. Change weapon (left arrow)13 Reload Weapon (R)

14 Exit game (ESC)

Fig,3.26 Pick up weapon

2 TerrainComposer2

TerrainComposer2 is a powerful node based **multi-terrain** tile generator. TC2 makes use of the latest GPU technology to give you instant real-time results, which makes creating terrains faster and more easy than ever before. TC2 its folder like layer system and workflow is similar to that of Photoshop, which makes it possible to have full control and make quick changes any time during the workflow.

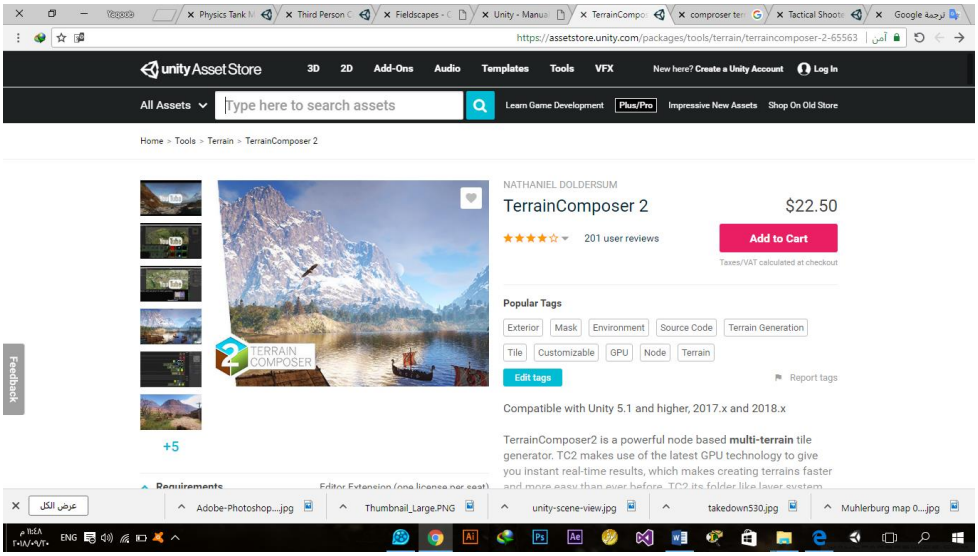


Fig.3.27 **Asset**

- ## 3.3.1 GUI

Graphical interfaces are graphical, graphical, window displays or more that include tools and components that allow the user to perform effective and attractive tasks within the. From these components: menus, toolbar, push button, selection button, slider, etc. Through interfaces, you can write and read data, as well as connect more than one interface and share data between them, in addition to the possibility of displaying this data in the form of graphs and tables, in our project we are download this package from Assets Store

3.3.2) Starting GUI creation

All the GUI creation and editing is done in Game view in WYSYWIG. The interface you create will look exactly as you see it while editing.

All user interface consists of Elements – small interface portions with individual functions: button, checkbox, window, etc. Unlike scripted Unity GUI, all elements of Instant GUI are separate game objects, and that gives the ability to quickly find, sort, and edit elements using Hierarchy and Game view. Elements form a hierarchy structure: if an element contains other elements (for example like window contains buttons) all the containing elements should it's be children. All of the elements are parented to root object with Instant GUI script (named Instant GUI as default).

To start creating GUI the root element should be created. Select Game Object → Create Instant GUI → New Instant GUI from a menu bar. A game object with "Instant GUI" name will be created. Then the default interface Style Set should be assigned. Style Set defines interface visual appearance: textures, colors, fonts, etc. Style sets are saved as an asset files. Drag "Glow. asset" from Demo folder to the root element's Style/Style Set slot. This will make all the new elements use Glow style set as the default.

The next step is to create the element of your choice – for example main window: Select Game Object → Create Instant GUI → Window from a menu bar.

Switch to the Game view and select the element that was just created in a hierarchy view. An editor frame with controls will appear around element. Dragging rectangular controls (anchors) attached to the sides of the element change it's relative position, and circle controls at the corners change element's pixel offset.

Element uses relative (in percent) and per-pixel positioning simultaneously. Firstly element is placed using relative position in a space of parent element, and then pixel offset is applied. For example, if left and right relative anchors set to left side of a parent, the element will be bound to the left side of a parent and element will have a constant size. If left anchor is at the left side of a parent, and the right anchor is on the right the element's size will be shrinking and expanding with the size change of a parent. If both anchors set to 50% between left and right element with a constant size will be fixed to the center of parent element. The pixel offset will offset the element after it was placed relatively.

Dragging relative or offset controls with **Control** key pressed will move these controls with a grid snap.

Dragging with **Shift** pressed will move control in one dimension only.

You can move element changing it's offset by clicking inside it's rectangle and dragging it to the desired place.

Using **arrow keys** with selected element will move it in relevant direction.

Clicking on another element will select it.

Alt-clicking on element will activate it. For example alt-clicking on button will press it the way it was pressed in game. Please note that it works only with element activation, other actions such as typing text in InputText or scrolling Slider will work only in playmode.

Press **L** to lock selection. When selection is locked no element could be selected by clicking it, but you can select element from hierarchy view. Press **H** to toggle editor controls on or off.

In Our project ,the first step creating the start main menu show in figure .

We set up four Buttons is called Level Select , Rest ,about and Quit.

When you press the Select Level button, the list of levels will appear show in figure 5.

Fig,3.28 Main menu

Figure 3.29 : Select Level

Now we will program the transition button to the first level when the user presses the first level.

Create new script C# .We take the script over the button.

```
using UnityEngine.SceneManagement;

public class GoTo_Level1 : MonoBehaviour {
    public InstantGuiButton jj;
       // Use this for initialization
       void Start () {

       }

       // Update is called once per frame
       void Update () {
        if (jj.activated) {

            SceneManager.LoadScene("mmm");
        }
       }
}
```

First thing we defined a variable of type `InstantGuiButton` then we did a working condition if the user pressed the button is going to the scene, `SceneManager.LoadScene("mmm");`

So with all the buttons.

When we start level 1 and press ESCAP Button the main menu show up.

Show the figure 6.

When we press Setting Button the Setting menu show up show in figure 7.

Fig,3.30 Loading level

3.3.3) Pause Main menu:

We create the pause menu it's contains Buttons ,Resume ,Setting ,Restart Level, next level and quit .

- Resume Button : Resume level.
- Setting Button :control the sound Background for level and skybox you can change from two light and night .
- Restart Button :restart level.
- Next level Button :got to the level two when you killed all enemy .
- Quit Button :Exit from game.
- Help Button Instruction for User
- About Button. about the game

Fig,3.31 Pause menu

When you press settinge Button You can control the sound Background and change your sound as you like and you can change Sky if you went.

Fig,3.32 Setting menu

3.3.4) MinMape.

In our project use this package to make Map.

Figure :3.33 Map

How to use:

- import the plugin package into your project
- drag the MiniMap prefab to your canvas
- set the player transform
- add the Map Marker component to the objects you want to be shown on the map

3.3.5) Description of the map options:

Player transform*:* is transform of the object that is the midpoint of the map.

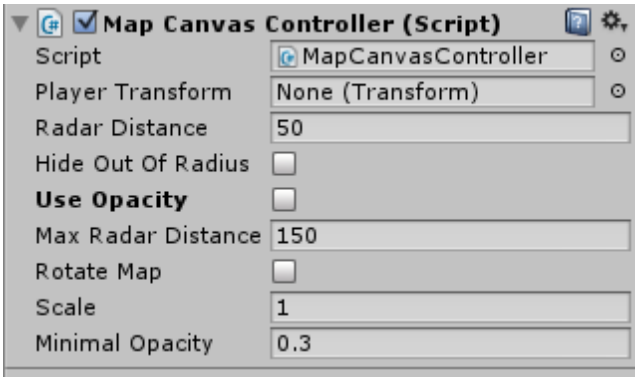


Figure :10 Map properties

Radar distance if the object is within this distance it will be show on the map. If objects are farther than this distance - they will be always on the border of the map or now shown at all, depending on the ***Hide Out Of Radius*** value.

Objects that are farther than ***Radar Distance*** but are within ***Max Radar Distance*** will be shown on the border of the map if ***Hide Out Of Radius*** is false. If ***Use Opacity*** is true, those objects will become more transparent while getting closer to ***Max Radar Distance***.

If ***Rotate Map*** is enabled, the map will rotate and the player arrow will point straight forward, otherwise the map will be fixed and the arrow will show the direction.

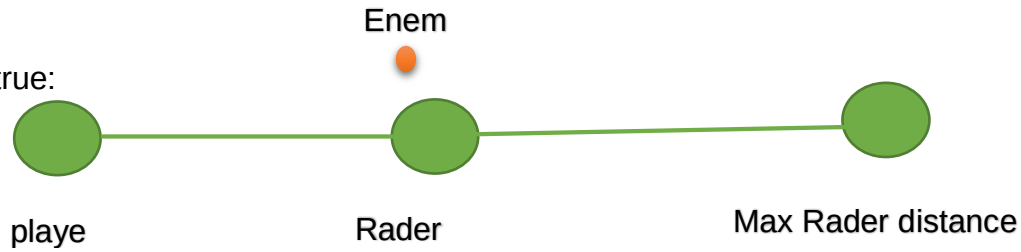
If ***Rotate Map*** is enabled, the map will rotate and the player arrow will point straight forward, otherwise the map will be fixed and the arrow will show the direction.

If ***Rotate Map*** is enabled, the map will rotate and the player arrow will point straight forward, otherwise the map will be fixed and the arrow will show the direction.
Scale is used to scale ***Radar Distance*** and Max ***Radar Distance***.

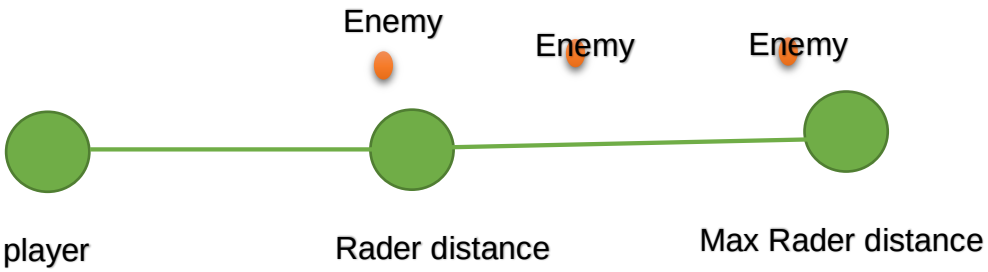
Minimal Opacity - minimal opacity for the markers that are farther than radar distance.

Next pictures will show how the marker will be shown on the map depending on the options .

Hide out of radius = true:

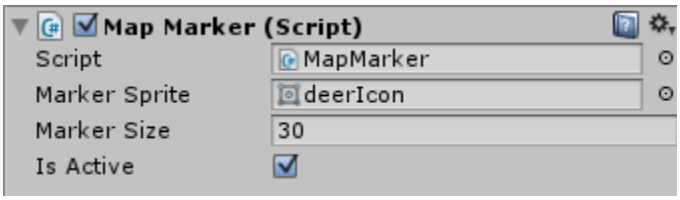


Hide out of radius = false, use opacity = false:



3.3.6 Map Marker options:

Marker Sprite – with this image the object will be marked on the map. Sample sprites are included in the package.
Marker Size – width and height of the marker.
Is Active – if false – the object won't be shown on the map show figure: 11.



4.1 Video Game

i) Result

Game development is a software development process which results in a game. A computer game is a game that involves human interaction with a user interface to generate visual feedback on a video device and profit for developer. Development is performed by one developer or by group of developers.

ii) Discussion

A game designer is a person who creates the game design. Game design includes game play, the rules and structure of a game. A game designer has to have a clear understanding of the game concept and provide the game with narrative content for development team. In larger projects, there are often separate designers for various parts of the game, such as game mechanics, user interface, characters, dialogue, etc. The designer should have good imagination, logical thinking and understanding of the software part of the project. A game artist is a person who creates game art. Art may be 2D oriented or 3D oriented.

2D artists may produce concept art, sprites, textures, environmental backdrops or terrain images, and user interface. 3D artists may produce 3D models, animation and 3D environment. Artists sometimes occupy both roles. A game programmer is a software engineer who implements game codebases and related tools. An individual programming task may include physics, AI, managing of graphical content, integration of sounds, implementation of various games rules, UI, input processing, network communications, etc.

4.2 Creating Exterior Environments

i) Result

In the Figure below , you learned how to create areas quickly, making use of Prototype. Now, this is a great tool when you're trying to create something man-made, such as houses, streets, and space stations. But for more organic things, such as hills, this can lead to issues. In this chapter, we are going to explore ways to add more organic-feeling areas to our level.

This project will be split into a number of tasks.

ii) Discussion

Before we start, we need to have a project created that already has UTPC and Prototype installed. If you do not have these already.

The first step is to create a new Unity project. Now add a new scene (File > New Scene). The new scene has been created with two game objects, the Main Camera and a Directional Light (Unity 5 will automatically add a skybox into the scene).

Now, in order to add a terrain game object, go to the GameObject menu, and select 3D Object > Terrain. This will add a flat plane (called Terrain) into your scene. This plane is the mesh that we are going to sculpt in order to create your level terrain

If you start with a non TC Scene, after opening the TC window you first need to create a terrain. The Terrain Area GameObject will be automatically selected and it has the terrain as a child and in its Inspector window you need to choose a project path to save your terrain, give it a name and click the ‘Create’ button. Make sure the path is within your Unity project. At the ‘Terrain Setup’ part you can easily setup your Unity terrain, this is a quicker way then editing the Unity terrain with its terrain script by itself. This will also work with multi terrain tiles (included in Asset Store version). The splat and grass textures in their tabs are already assigned and you can put your own textures there.

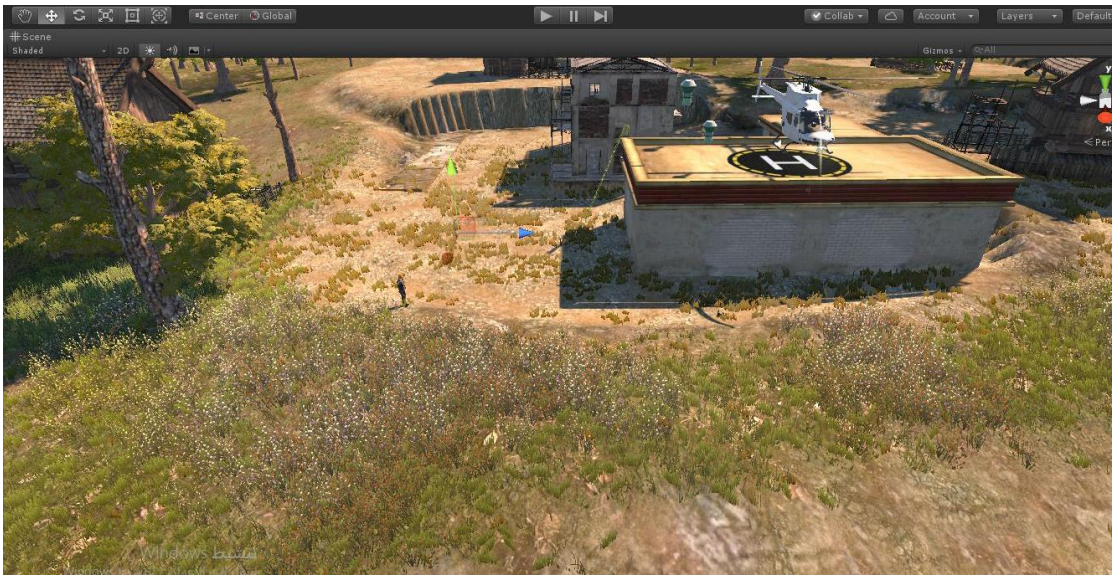


Fig.4.1. CREATING EXTERIOR ENVIRONMENTS

4.3 Creating and Design Player

i) Result

we import the player modeling from C4D.

In our project to modeling the face character for student Bandar.

Import player Modeling from C4d.

we Import Animation from C4d but it is difficult make in C4D we use the website www.mixmo.com we download Animation from it after download animation we make Animator in unity3d.

ii) Discussion

You can view and set up character behavior from the Animator Controller view (Menu: Window > Animator Controller).

The various ways an Animator Controller can be created:

- From the Project View by selecting ‘Create > Animator Controller’.
- By right-clicking in the Project View and selecting ‘Create > Animator Controller’.
- From the Assets menu by selecting ‘Assets > Create > Animator Controller’.

During Play Mode, the Animator pans the view so that the current state being played is always visible. The Animator Controller respects the independent zoom factors of the Base Layer and Sub-State Machine, and the window pans automatically to ensure visibility of the active state or states.

To modify the zoom during Play Mode, follow these steps:

- Enable Auto Live Link in the Animator Controller window
- Click the Play button to enter Play Mode
- Click Pause
- In the Animator Controller, select the state or states you want to zoom into
- Press the F key to zoom into the selection
- Click the Play button again to resume Play Mode

Fig.4.2. CREATING AND DESIGN PLAYER

4.4 Creating Your Own AI Agent

i) Result

All living organisms share NEURAL SYSTEMS that enable them to interact with and interact with the surrounding environment and help them control the vital processes

needed to sustain the life of these organisms. Neural systems vary from one organism to another where they are simple to install and normal to work in.

Primary organisms with simple cytoskeletal structure, complex structure and the nature of work in higher organisms such as humans.

It is the nervous system of man The most complex neurological systems at all, which are concentrated mostly in the human brain, which characterizes the nature of work that led to superiority of the human other creatures in the ability to understand and recognize the forms and symbols and learning and talking and remembering and recognition and control of the precise motor and so many qualities and capabilities that cannot Any other object other than a human being has access to it.

ii) Discussion

1.Your model must be fully modelled, rigged, and animated.

2.You must have created a ragdoll object for your model. The easiest way to do this is usually with the unity ragdoll wizard. The Unity documentation has some pages to help you get through this process.

3.You must have created a mask for your model that excludes the legs. This is so that we can play movement animations on the lower body while simultaneously allowing the upper body to play animations such as reloading or firing. It will also allow our agent to aim their gun towards the target. Examples for both generic and humanoid rigs are as follows.

4.You must create an “eye” object and make it an object within the ragdoll’s hierarchy. The forward axis (represented by the blue arrow in the scene view in local space) must be facing forwards. If you don’t get it right though, you can always change it later.

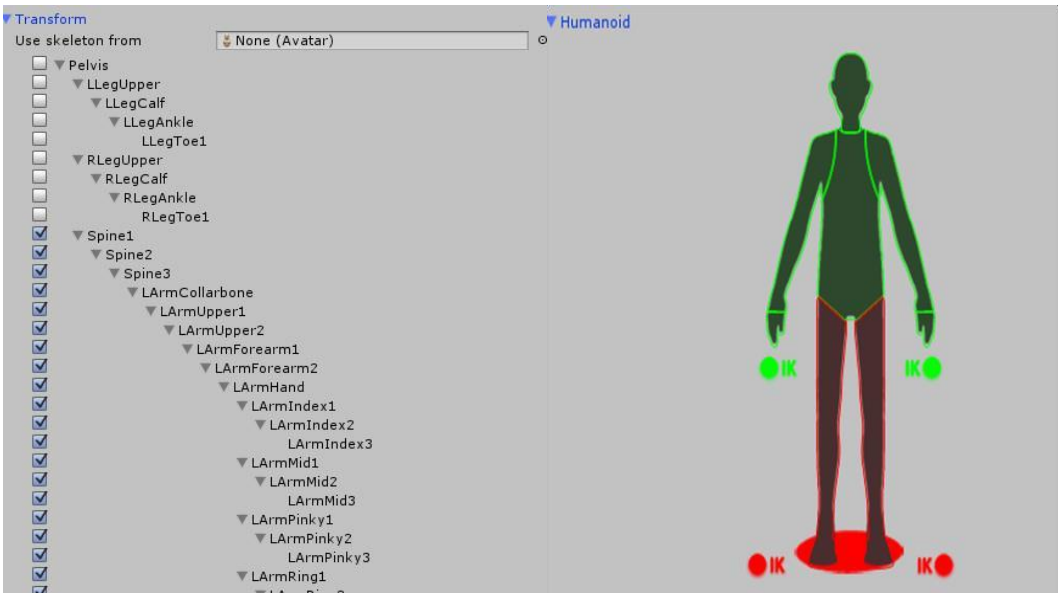


Fig.4.3. CREATING YOUR OWN AI AGENT

4.5 GUI Design

i) Result

Graphical interfaces are graphical, graphical, window displays or more that include tools and components that allow the user to perform effective and attractive tasks within the. From these components: menus, toolbar, push button, selection button, slider, etc. Through interfaces, you can write and read data, as well as connect more than one interface and share data between them, in addition to the possibility of displaying this data in the form of graphs and tables, in our project we are download this package from Assets Store

ii) Discussion

- In the below graphical interface, when you select the data from the drop-down list, and then choose the style of the graphic you want by clicking the push button, this button calls the child that draws the selected data on axes.

- Most graphical interfaces wait for users to affect the components in the interface, and then respond to each event that is activated Each component in the interface and even the interface itself is associated with one or more user

- written routines known as callbacks, called call back because it calls or asks the something.

- Any action done by the user, such as clicking the push button, selecting an item from the list, entering text or numbers, or any event that he performs, will be the reason for the execution of a call.

Fig.4.4 GUI DESIGN

The gui then responds to these events.

- When you build the graphical interface, you will add calls telling the object what to do when responding to an event.

This type of programming is often known as event-driven programming.

For example: Click the button (event).

- In event-driven programming, the invocation is performed asynchronously It is triggered by external events that affect the program.

In the unity 3d graphical interfaces, most events are the result of user interaction with the interface, and the interface can respond to other types of events.

CONCLUSION AND FUTURE WORK

5.1 Finalizing Our Project

i) Conclusion:

Once you finish your game project, it's important to take the time to get your projects out in the correct way.

Here is the outline of our tasks:

- Building the game in Unity
- Building an installer for Windows

Prerequisites

Before you start, you need to have your project completed just the way that you want it to be when you send it out into the world.

ii) Future Work:

Games are at the forefront of another important technical field; it's user engagement. Expanding play is a top priority for game designers. One way to attract people is to enhance their personal relationship with the game. Many of the most popular games today, from golf simulators to Sims, allow you to make your own avatar. This creates an emotional connection between the player and the game. The QVIT, an online clothing store, allows shoppers to make personal clothing measurements, betting that seeing yourself as a dresser will make you more likely to buy them. Some methods are used to make you continue to play directly from behavioral psychology. In fact, many gaming companies employ specialists in psychology to devise strategies for participation. Think about the last game you played. The first stages are very easy, winning and rewarding, both in the form of weapons upgrades or virtual funds within the game. But the longer you play, the more difficult it will be to move to the next level. This is called the period or percentage of the reward. If the periods are too long, frustrate the player and leave the game.

Although very short, the player feels that the game is very easy and does not merit his time. The improvement of reward periods is derived from the work of PF Skinner and other behavioral scientists.

5.2 Building the game in Unity

i) Conclusion:

There are many times during development you may want to see how your game will appear if you build it outside the editor. It can give you a sense of accomplishment; I know, I felt that way the first time I pushed a build to a console development kit.

No matter what platform we wish to create out game for in order to build it we need to go to the **Build Settings**.

ii) Future Work:

In order to access **Build Settings**, we will need to go to **File** | **Build Settings** from the top menu. Once you're ready, select **Platform** from the bottom-left menu. The Unity logo will show the one you're currently compiling for. We're going to compile for Windows now, so if it is currently not set to **PC, Mac & Linux Standalone**, select it and press the **Switch Platform** button.Once you have all of this set up, press the **Build** button. It will ask you for a name and a location to place the game. I'm going to name it **WareGame** and put it in an **Export** folder located in the same directory as the **Assets** and **Library** folders. Afterward, hit **Save**. While building for Windows, you should get something like the preceding screenshot. We have the executable, but we also have a **Data** folder that contains all of the assets for our application (right now called **WareGame_Data**). You must include the **Data** folder with your game or it will not run. This is troublesome; but later on in this chapter, will create an installer that will put it on a computer without any hassle.

5.3 Building an installer for Windows

iii) Conclusion:

Now that we know what happens by default, let's take some time to customize the project to make it look as nice as possible. **PlayerSettings** is where we can define different parameters for each platform that we want to put the game onto.

To open **PlayerSettings**, you can either click on the **PlayerSettings** button from the **Build Settings** menu or go to **Edit** | **Project Settings** | **Player**. **PlayerSettings** is actually shown in **Inspector**. There are some key properties at the top that are cross-platform, which means that they will apply to all platforms (or rather will be the defaults you can override later)

From the **Project** tab, create a new folder within the **MyGame** folder and name it **Sprites**.

Now, in the **Example Code** folder, you'll find a **cursor_hand** image. Drag and drop it to the **Assets/MyGame/Sprites** folder of the **Project** browser. Once there, select the image and change **Texture Type** to **Cursor**.

Note that, while the image I created works, you're more than welcome to create your own cursor and put it here to suit your particular game.

iv) Future Work:

On your computer, in the **Example Code** folder, move the **ConfigBanner** image into the **Assets/MyGame** folder. Select the object and under **Texture Type**, change it to **Editor GUI and Legacy GUI** and then click on **Apply**.

Then, under the **PlayerSettings** section in **Inspector**, click on the **Splash Image** section to open the **Config Dialog Banner** property in which you should set the newly imported image.

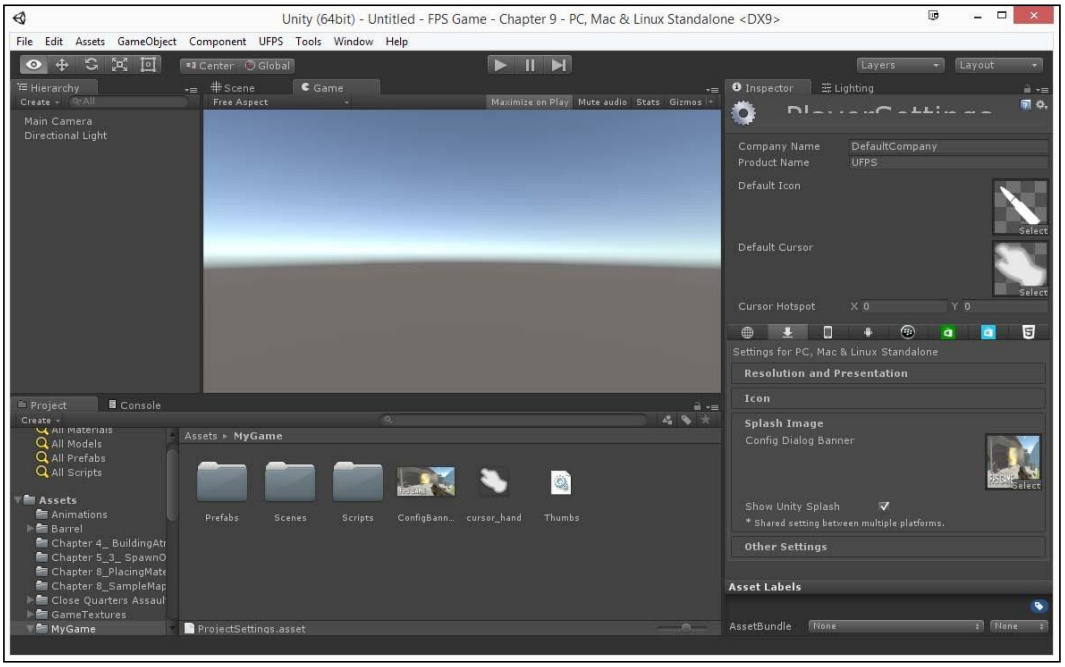


Fig:5.1. Dialog Banner

If you want to create a **Config Dialog Banner** property of your own, make sure that you make the image 432 x 200 pixels in size or smaller.

Next, you'll need to decide whether you want **Display Resolution Dialog** to be displayed or not. If you want to keep it hidden, skip this step, since it's disabled by one of the assets we've imported. Otherwise, open the **Resolution and Presentation** section and, under **Standalone Player Options**, set **Display Resolution Dialog** to **Enabled**.

5.4 Appendix:

War games can help people understand the graphics. They can also see war zones and how homes are destroyed because of them

1. Take advantage of the graphics found in the game.
2. Reclaiming them is a plan to eliminate the enemy.
3. Explain to people what is going on after the war.
4. In addition to the benefit of the leadership and the officers of how to eliminate the

5.5 Game specifications:

1. It should be 3D type and not 2D type.

2. it is thired the controller processing not first the controller processing.

3. Use the Cinema 4D platform and not the Max or Mayan.

4. The minimum spec is not a hard and fast rule of the engine but something that you find through testing. The best way to set the min spec is to actually run the game on various older machines and see how it holds up. You can use that testing to narrow in what you consider to be a reasonable min spec for your game. The recommended spec is probably similar to the machine you developed on, or something that consistently performs well in your testing.

5.6 Game requirements:

1. Minimum requirements: 2.5 GHz Intel Core i3 processor, 2 GB RAM, at least 2 GB of free hard drive space.
2. There are multiple ways to get minimum specs for your game, some easier than others.

3. Example- Minimum requirements: DX10 + with shader model 4.0 capabilities.

4. For stuff like OS, graphics card, and CPU compatibility, that info can be accessed easily.For example, if you are making a game with Unity, the system requirements can be found here, on Unity's official requirements page.In fact, it even provides in-depth details for platforms, such as iOS, Android, WebGL, and Windows Store Apps.Just look under "For running Unity games" to find the correct info.Other engines should have similar pages too.

5. Graphics card
   Nvidia GeForce GTX 1080.

6. For game development, you should have at least one computer available that's capable of handling the highest graphics level you plan to implement. If you're doing a simple 2D game, you might not need a GPU at all.

   On the other hand, if you plan on doing AAA-level photorealistic 3D graphics, you'll probably want a pair of GTX 1080 Ti cards so you can test your game at ultra settings. (You should also have a potato of a laptop on hand to test low settings.)

5.7 Code the TacticalAI

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Invector.CharacterController;

/**
This script finds cover based on an agent's position, target position, and a key position in the level
  First tries to find static cover, but if none can be found, dynamic cover is used
 * */

namespace TacticalAI
{
   public class CoverFinderScript : MonoBehaviour
   {
      public vThirdPersonController dd;
     public static int a;

      //Cover Seek Methods

      public CoverSeekMethods currentCoverSeekMethod =
CoverSeekMethods.WithinCombatRange;
      public enum CoverSeekMethods
      {
         RandomCover = 0, WithinCombatRange = 1, AdvanceTowardsTarget = 2,
      }

      public enum DynamicCoverMethods
      {
         NavmeshScan = 0, Raycasts = 1,
      }

public static void BandarKill()
```

```
{
    a = a + 1;
    print("sssssssssssss");
}
void Update()
{

    if (a == 5)
    {


        dd.maxHealth = 0f;
        a = 0;
    }
}
```

5.7. 1 //Distance Stuff

```
private Vector3 lastCoverPos;
public float minDistBetweenLastCover = 20;
private float minDistBetweenLastCoverSquared = 10;
public float minCoverDistFromEnemy = 10;
public float maxCoverDistFromEnemy = 50;

//Use squared values for cheaper distance check via Vector3.SqrMagnitude

private float maxCoverDistSqrd;
private float minCoverDistSqrd;
public float maxDistToCover = 9999;

public float minDistToAdvance = 5;

private TacticalAI.CoverNodeScript[] coverNodeScripts;

//Dynamic cover

public DynamicCoverMethods dynamicCoverSelectionMode =
DynamicCoverMethods.Raycasts;
public bool shouldUseDynamicCover = true;
public bool useFirstDynamicCoverFound = true;
public float dynamicCoverMaxDistFromMe = 15f;

//private var dynamicCoverMaxDistFromMeSqr : float;

public float dynamicCoverNodeHeightOffset = 0.3f;
public float dynamicCoverNodeFireOffset = 1.5f;
public float dynamicCoverWidthNeededToHide = 1.5f;
```

```
public float maxDistBehindDynamicCover = 5;

public bool useOnlyStaticCover = true;

public float defendingDist = 20;
private float defendingDistSquared = 20;
private Transform myTransform;

[HideInInspector]
public LayerMask layerMask;

Vector3[] verts;

TacticalAI.NavmeshInterface navI;


// Use this for initialization

void Start()
{
   navI = gameObject.GetComponent<TacticalAI.BaseScript>().navI;
   if(useOnlyStaticCover && shouldUseDynamicCover){
            verts = navI.GetNavmeshVertices();
   }
   myTransform = transform;

   //dynamicCoverMaxDistFromMeSqr =
dynamicCoverMaxDistFromMe*dynamicCoverMaxDistFromMe;

   maxCoverDistSqrd = maxCoverDistFromEnemy * maxCoverDistFromEnemy;
   minCoverDistSqrd = minCoverDistFromEnemy * minCoverDistFromEnemy;
   layerMask = TacticalAI.ControllerScript.currentController.GetLayerMask();
   defendingDistSquared = defendingDist * defendingDist;
   minDistToAdvance = minDistToAdvance * minDistToAdvance;
   if (TacticalAI.ControllerScript.currentController != null)
   {
      coverNodeScripts = TacticalAI.ControllerScript.currentController.GetCovers();
   }
   else
   {
      Debug.LogWarning("No Controller has been detected!  An AIControllerScript is
required for the AI to work!  Please create a new gameObject and attach the Paragon AI
ControllerScript to it!");
   }
}
```

```
        //Utility Stuff

        public void ResetLastCoverPos()
        {
            lastCoverPos = new Vector3(100000, 100000, 100000);
        }

        public TacticalAI.CoverData FindCover(Transform targetTransform)
        {
            return FindStaticCover(targetTransform, null);
        }

        public TacticalAI.CoverData FindCover(Transform targetTransform, Transform
transformToDefend)
        {
            return FindStaticCover(targetTransform, transformToDefend);
        }

        // Static Cover

        TacticalAI.CoverData FindStaticCover(Transform targetTransform, Transform
transformToDefend)
        {
            if (targetTransform && myTransform)
            {
                Vector3 targetTransformPos = targetTransform.position;

                //Closest Covers

                if (currentCoverSeekMethod == CoverSeekMethods.WithinCombatRange)
                {
                    return FindCoverWithinCombatRange(targetTransformPos, transformToDefend);
                }
                //Advance Towards Cover

                else if (currentCoverSeekMethod == CoverSeekMethods.AdvanceTowardsTarget)
                {
                    return FindCoverWithinCombatRange(targetTransformPos, transformToDefend);
                }

                //Random Cover

                else
                {
                    return FindRandomCover(targetTransformPos, transformToDefend);
                }
```

```
        }
        TacticalAI.CoverData bsData = new TacticalAI.CoverData();
        return bsData;
    }


    //Agent will try and find cover that is within a given range of the target.

    TacticalAI.CoverData FindCoverWithinCombatRange(Vector3 targetTransformPos,
Transform transformToDefend)
    {
        int i = 0;
        Vector3 myPos = myTransform.position;
        TacticalAI.CoverNodeScript currentCoverNodeScript = null;
        float closestDistSquared = maxDistToCover;
        float nodeCheckingNowDistSquared;
        float distToTargetSquared;


        //We will take cover outside of the desired range if we can't find any within.

        bool foundCoverWithinAcceptableRange = false;
        for (i = 0; i < coverNodeScripts.Length; i++)
        {
            //Check if the node we are checking is occupied and within acceptable distances to key
points

            if (!coverNodeScripts[i].isOccupied() &&
Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() - lastCoverPos) >
minDistBetweenLastCoverSquared &&
                (!transformToDefend || Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() -
transformToDefend.position) < defendingDistSquared))
            {
                distToTargetSquared = Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() -
targetTransformPos);
                nodeCheckingNowDistSquared = Vector3.SqrMagnitude(myPos -
coverNodeScripts[i].GetPosition());

                //Check for line of sight

                if (coverNodeScripts[i].ValidCoverCheck(targetTransformPos))
                {
                    //Prefer nodes within othe agent's combat range
```

```
                if (minCoverDistSqrd < distToTargetSquared && maxCoverDistSqrd >
distToTargetSquared)
                {
                    if (!foundCoverWithinAcceptableRange || (nodeCheckingNowDistSquared <
closestDistSquared))
                    {
                        closestDistSquared = nodeCheckingNowDistSquared;
                        currentCoverNodeScript = coverNodeScripts[i];
                        foundCoverWithinAcceptableRange = true;
                    }
                }
                //Check if this is the closest so far

                else if (!foundCoverWithinAcceptableRange && nodeCheckingNowDistSquared
< closestDistSquared)
                {
                    closestDistSquared = nodeCheckingNowDistSquared;
                    currentCoverNodeScript = coverNodeScripts[i];
                }
            }
        }
    }


    //pass the data to the script that asked for cover

    if (currentCoverNodeScript != null)
    {
        lastCoverPos = currentCoverNodeScript.GetPosition();
        return new TacticalAI.CoverData(true, currentCoverNodeScript.GetPosition(),
currentCoverNodeScript.GetSightNodePosition(), false, currentCoverNodeScript);
    }

    //Only bother with dynamic cover if we need it

    if (shouldUseDynamicCover && !ControllerScript.pMode)
    {
        return FindDynamicCover(targetTransformPos, transformToDefend);
    }

    return new TacticalAI.CoverData();
}

//The agent will find cover that is closer to their target each time they change cover locations
```

```
    TacticalAI.CoverData FindAdvancingCover(Vector3 targetTransformPos, Transform
transformToDefend)
    {
        int i = 0;
        Vector3 myPos = myTransform.position;
        TacticalAI.CoverNodeScript currentCoverNodeScript = null;



  //Will find closest cover that is nearer than the last one we have if possible.
        //If not, we'll move to the target.

        Vector3 posToAdvanceTo;

        if (transformToDefend)
            posToAdvanceTo = transformToDefend.position;
        else
            posToAdvanceTo = targetTransformPos;

        float distBetweenMeAndTarget = Vector3.SqrMagnitude(myPos - posToAdvanceTo) -
minDistToAdvance;
        float closestDistBetweenMeAndCover = distBetweenMeAndTarget;

        for (i = 0; i < coverNodeScripts.Length; i++)
        {
            if (!coverNodeScripts[i].isOccupied())
            {
                float sqrDistBetweenNodeAndTargetPos =
Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() - posToAdvanceTo);

                //Check if we'll be closer to target than we stand now

                if (sqrDistBetweenNodeAndTargetPos < distBetweenMeAndTarget)
                {
                    //Check if this node is closest to us

                    if (Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() - myPos) <
closestDistBetweenMeAndCover)
                    {
                        //Check if node is safe

                        if (coverNodeScripts[i].ValidCoverCheck(targetTransformPos))
                        {
                            closestDistBetweenMeAndCover = sqrDistBetweenNodeAndTargetPos;
                            currentCoverNodeScript = coverNodeScripts[i];
                        }
```

```
                }
            }
        }
    }
    if (currentCoverNodeScript != null)
    {
        lastCoverPos = currentCoverNodeScript.GetPosition();
        return new TacticalAI.CoverData(true, currentCoverNodeScript.GetPosition(),
currentCoverNodeScript.GetSightNodePosition(), false, currentCoverNodeScript);
    }

    //Dynamic advancing cover is NOT supported

    return new TacticalAI.CoverData();
}

//The agent will use a random peice of cover that satisfies the line of sight requirements.

TacticalAI.CoverData FindRandomCover(Vector3 targetTransformPos, Transform
transformToDefend)
{
    int i = 0;
    TacticalAI.CoverNodeScript currentCoverNodeScript = null;
    List<TacticalAI.CoverNodeScript> availableCoverNodeScripts = new
List<TacticalAI.CoverNodeScript>();

    //Fill a list with potential nodes

    for (i = 0; i < coverNodeScripts.Length; i++)
    {
        if (!coverNodeScripts[i].isOccupied())
        {
            if (coverNodeScripts[i].ValidCoverCheck(targetTransformPos) &&
(!transformToDefend || Vector3.SqrMagnitude(coverNodeScripts[i].GetPosition() -
transformToDefend.position) < defendingDistSquared))
            {
                availableCoverNodeScripts.Add(coverNodeScripts[i]);
            }
        }
    }

    if (availableCoverNodeScripts.Count > 0)
    {
        //Pick a random node
```

```
            currentCoverNodeScript = availableCoverNodeScripts[Random.Range(0,
availableCoverNodeScripts.Count)];
            lastCoverPos = currentCoverNodeScript.GetPosition();

            return new TacticalAI.CoverData(true, currentCoverNodeScript.GetPosition(),
currentCoverNodeScript.GetSightNodePosition(), false, currentCoverNodeScript);
        }



  //Only bother with dynamic cover if we need it

        if (shouldUseDynamicCover && !ControllerScript.pMode)
        {
            return FindDynamicCover(targetTransformPos, transformToDefend);
        }

        return new TacticalAI.CoverData();
    }

    //Dynamic Cover

    TacticalAI.CoverData FindDynamicCover(Vector3 targetTransformPos, Transform
transformToDefend)
    {
        if(dynamicCoverSelectionMode == DynamicCoverMethods.Raycasts)
        {
            return FindRaycastDynamicCover(targetTransformPos, transformToDefend);
        }


        if(!useOnlyStaticCover){
                verts = navI.GetNavmeshVertices();
        }
        Vector3 myPos;
        if (!transformToDefend)
            myPos = myTransform.position;
        else
            myPos = transformToDefend.position;
        myPos.y += dynamicCoverNodeHeightOffset;
        var hideOffset = dynamicCoverNodeFireOffset - dynamicCoverNodeHeightOffset;

        //int nodesFound = 0;

        Vector3 hidingPosCheckingNow;
        int x = 0;
```

```
        //int y;

        float currDistTarget;

        Vector3 coverHidePos = Vector3.zero;
        Vector3 coverFirePos = Vector3.zero;

        float closestDistToMeSoFarSqr = dynamicCoverMaxDistFromMe *
dynamicCoverMaxDistFromMe;
        float distBetweenMeAndCoverNow;

        bool shouldCont = true;

        //Use each vertex on the navmesh as a potential "firing position"
        //Then test whether we can hide from enemy fire by either crouching or moving off to the
side (distance to move is hideOffset)
        //If we can see the enemy from the firing position and not see them from the hiding
position, then it is a valid cover spot.

        for (int i = 0; i < verts.Length; i++)
        {
           //random value to make sure we don't take the same cover every time

           if (Random.value > 0.5 && Vector3.SqrMagnitude(verts[i] - myPos) >
minDistBetweenLastCover)
           {
              currDistTarget = Vector3.SqrMagnitude(verts[i] - targetTransformPos);
              distBetweenMeAndCoverNow = Vector3.SqrMagnitude(verts[i] - myPos);

              if (distBetweenMeAndCoverNow < closestDistToMeSoFarSqr && currDistTarget >
minCoverDistSqrd && currDistTarget < maxCoverDistSqrd)
              {
                 verts[i].y += dynamicCoverNodeFireOffset;

                 //If we can fire from here

                 if (!Physics.Linecast(verts[i], targetTransformPos, layerMask))
                 {
                    verts[i].y -= hideOffset;

                    //Debug.Break();
```

```
                if (Physics.Raycast(verts[i], targetTransformPos - verts[i],
maxDistBehindDynamicCover, layerMask) &&
!TacticalAI.ControllerScript.currentController.isDynamicCoverSpotCurrentlyUsed(verts[i]))
                {
                    shouldCont = true;

                    //If chest high wall

                    hidingPosCheckingNow = verts[i];


  //Check to make sure we have clear LoS between the firing position and the move position.

                    if (!Physics.Linecast(targetTransformPos, verts[i], layerMask) &&
Physics.Linecast(hidingPosCheckingNow, targetTransformPos, layerMask))
                    {
                        closestDistToMeSoFarSqr = distBetweenMeAndCoverNow;
                        coverHidePos = hidingPosCheckingNow;
                        coverFirePos = verts[i];
                        shouldCont = false;
                        if (useFirstDynamicCoverFound)
                        {
                            break;
                        }
                    }

                    //Check for side cover

                    if (shouldCont)
                    {
                        for (x = -1; x <= 1; x += 2)
                        {
                            hidingPosCheckingNow = verts[i] + myTransform.right * x *
dynamicCoverWidthNeededToHide;
                            //If we're safe
                            if (!Physics.Linecast(hidingPosCheckingNow, verts[i], layerMask) &&
Physics.Linecast(hidingPosCheckingNow, targetTransformPos, layerMask))
                            {

   //lastCoverPos = hidingPosCheckingNow;
  //return new TacticalAI.CoverData(true, hidingPosCheckingNow, verts[i], true, null);

                                closestDistToMeSoFarSqr = distBetweenMeAndCoverNow;
                                coverHidePos = hidingPosCheckingNow;
                                coverFirePos = verts[i];
```

```
                        shouldCont = false;
                        if (useFirstDynamicCoverFound)
                        {
                            break;
                        }
                    }
```

```
 //Taken out for performance

/*  hidingPosCheckingNow = verts[i] + myTransform.forward * x *
dynamicCoverWidthNeededToHide;

                        //If we're safe
                        if (shouldCont && !Physics.Linecast(hidingPosCheckingNow, verts[i],
layerMask) && Physics.Linecast(hidingPosCheckingNow, targetTransformPos, layerMask))
                        {
                            //return new TacticalAI.CoverData(true, hidingPosCheckingNow,
verts[i], true, null);
                            closestDistToMeSoFarSqr = distBetweenMeAndCoverNow;
                            coverHidePos = hidingPosCheckingNow;
                            coverFirePos = verts[i];
                            if (useFirstDynamicCoverFound)
                            {
                                i = verts.Length;
                            }
                        } */
                    }
                }

            }
        }
    }
}
if (coverHidePos != Vector3.zero)
{
    lastCoverPos = coverHidePos;
    return new TacticalAI.CoverData(true, coverHidePos, coverFirePos, true, null);
}

return new TacticalAI.CoverData();
```

```
    }

    public int angleBetweenCasts = 10;
    public float maxRayDist = 1000f;
    public float rayCastHeightOffGround = 0.1f;
    public float distFromWallToBe = 1;

    TacticalAI.CoverData FindRaycastDynamicCover(Vector3 targetTransformPos, Transform
transformToDefend)
    {
        if(!transformToDefend)
        {
            transformToDefend = myTransform;
        }

        int i = 0;
        Vector3 currentCastPos = transformToDefend.position;
        Vector3 currentCastVector = Vector3.Normalize(targetTransformPos -
transformToDefend.position);
       currentCastVector = Quaternion.Euler(0, -90, 0) * currentCastVector;

        Vector3 hidingPosCheckingNow;
        Vector3 coverFirePos;
        RaycastHit hit = new RaycastHit();

        if(Physics.Raycast(transformToDefend.position, Vector3.down, out hit, maxRayDist,
layerMask))
        {
            currentCastPos = hit.point;
        }

        currentCastPos.y += rayCastHeightOffGround;

        //Debug.Break();
        //Fill a list with potential nodes

        for (i = 0; i < 360f; i += angleBetweenCasts)
        {
            if (Physics.Raycast(currentCastPos, currentCastVector, out hit, maxRayDist,
layerMask))
            {
                if
(!TacticalAI.ControllerScript.currentController.isDynamicCoverSpotCurrentlyUsed(hit.point))
                {
                    hidingPosCheckingNow = hit.point;
                    hidingPosCheckingNow -= currentCastVector * distFromWallToBe;
```

```
                //
                //Debug.DrawLine(currentCastPos, hidingPosCheckingNow, Color.yellow);
                //

                if (Physics.Linecast(hidingPosCheckingNow, targetTransformPos, layerMask))
                {

                    coverFirePos = hidingPosCheckingNow + new Vector3(0,
dynamicCoverNodeFireOffset, 0);



                    //Debug.DrawLine(hidingPosCheckingNow, coverFirePos, Color.blue);
                    //

                    if (!Physics.Linecast(coverFirePos, targetTransformPos, layerMask))
                    {

                        //
                        //Debug.DrawLine(coverFirePos, targetTransformPos, Color.green);
                        //

                        lastCoverPos = hidingPosCheckingNow;
                        return new TacticalAI.CoverData(true, hidingPosCheckingNow,
coverFirePos, true, null);
                    }
                }
            }
        }

        currentCastVector = Quaternion.Euler(0, angleBetweenCasts, 0) * currentCastVector;
    }



    return new TacticalAI.CoverData();
  }

 }
}

namespace TacticalAI
{
 public class CoverData
 {
  public bool foundCover = false;
```

```
        public Vector3 hidingPosition;
        public Vector3 firingPosition;
        public bool isDynamicCover;
        public TacticalAI.CoverNodeScript coverNodeScript;

        public CoverData(bool f, Vector3 hp, Vector3 fP, bool d, TacticalAI.CoverNodeScript cns)
        {
            foundCover = f;
            hidingPosition = hp;
            firingPosition = fP;
            isDynamicCover = d;
            coverNodeScript = cns;
        }

        public CoverData()
        {
            foundCover = false;
        }
    }
}
using UnityEngine;
using System.Collections;

/**  This script takes damage from various body parts, multiplies it, and passes it onto the
Health Script.
 * */

namespace TacticalAI
{
    public class HitBox : MonoBehaviour
    {
        public float damageMultiplyer = 1;
        private Vector3 addForceVector;

        private Rigidbody myRigidBody;
        public TacticalAI.HealthScript myScript;
        public bool canDoSingleHealthBoxDamage = true;
        public static int countt;

        public int countEn;
        [HideInInspector]
        public float damageTakenThisFrame = 0;

        //public bool storeDamage = false;

        void Awake()
```

```
{
    myRigidBody = gameObject.GetComponent<Rigidbody>();
}

public void ApplyDamage(float damage)
{
    Damage(damage);
}

public void ApplyDamage(float damage, float force, Vector3 dir)
{
    Damage(damage, force, dir);
}
void OnCollisionEnter(Collision other)
{
    // Destroy(this);

    if (other.gameObject.tag == "sd")
    {
        //hud = vHUDController.instance;
        //hud.ShowDamageSprite();
        // bandar(15f);sbyte=
        //  Destroy

        // dd.band();
        //countt = countt + 1;

        //if (countt == 2) {
        myScript.band();
        // countt = 0;
        // myScript.band();
        // countEn = countEn + 1;
    }
    // SetCountText();
    //mn(10f);
    // Destroy(gameObject,1f);

    if (this == null)
        return;

    //dd.Destroy(gameObject);
    //print("rrrrrrrrBANDA");
    //numberkillenemy++;


}
```

```
    public void Damage(float damage)
    {
        if (myScript)
        {
  //Use the multiplier to take differing amounts of damage depending on where the AI is hit

            damage = damage * damageMultiplyer;

  //Store the amount of damage taken for the dismemberment sript

            StartCoroutine("StoreDamageTakenRecently", damage);

            if (myScript)
                myScript.Damage(damage);
        }
    }

    public void Damage(float damage, float force, Vector3 dir)
    {
        if (myScript)
        {
  //Use the multiplier to take differing amounts of damage depending on where the AI is hit

            damage = damage * damageMultiplyer;

            StartCoroutine(AddForceVector(force*dir));

            //Store the amount of damage taken for the dismemberment sript

            StartCoroutine("StoreDamageTakenRecently", damage);

            if (myScript)
                myScript.Damage(damage);
        }
    }
   //Use for explosives

    public void SingleHitBoxDamage(float damage)
    {
    //We don't do the damage multiplier here because this is used for explosions, and we  don't
want to leave it up to RNG which hitbox is used first

        if (myScript)
        {
   //StartCoroutine("StoreDamageTakenRecently", damage);
```

```
            if (canDoSingleHealthBoxDamage)
                StartCoroutine(myScript.SingleHitBoxDamage(damage));
            else
                myScript.Damage(damage);
        }
    }

    public IEnumerator StoreDamageTakenRecently(float d)
    {


  //Only store the damage this frame.  That way only a single strong damage source will trigger the dismemberment script.

        damageTakenThisFrame += d;
        yield return 0;
        damageTakenThisFrame -= d;
    }

  //Do I even need this?

    public IEnumerator AddForceVector(Vector3 fv)
    {
        yield return null;
        if (myRigidBody)
            {
                myRigidBody.AddForce(fv, ForceMode.Impulse);
            }
    }
  }
}
```

5.8 How to Align a Shooter Weapon AND left-hand IK

i) Programming Code:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

namespace Invector.ItemManager
{
    [vClassHeader("Weapon Holder Manager", "Create a new empty object inside a bone and add the vWeaponHolder script")]
    1 reference
    public class vWeaponHolderManager : vMonoBehaviour
    {
        public vWeaponHolder[] holders = new vWeaponHolder[0];
        [HideInInspector]
        public bool inEquip;
        [HideInInspector]
        public vItemManager itemManager;

        public Dictionary<string, List<vWeaponHolder>> holderAreas = new Dictionary<string, List<vWeaponHolder>>();

        0 references
        void OnDrawGizmosSelected()
        {
            holders = GetComponentsInChildren<vWeaponHolder>(true);
        }

        0 references
        void Start()
        {
            itemManager = GetComponent<vItemManager>();
            if (itemManager)
            {
                itemManager.onEquipItem.AddListener(EquipWeapon);
                itemManager.onUnequipItem.AddListener(UnequipWeapon);

                holders = GetComponentsInChildren<vWeaponHolder>(true);
                if (holders != null)
                {
                    foreach (vWeaponHolder holder in holders)
                    {
                        if (!holderAreas.ContainsKey(holder.equipPointName))
                        {
                            holderAreas.Add(holder.equipPointName, new List<vWeaponHolder>());
                            holderAreas[holder.equipPointName].Add(holder);
```

Fig.(a): How to Align a Shooter Weapon AND left-hand IK

```
                {
                    holderAreas.Add(holder.equipPointName, new List<vWeaponHolder>());
                    holderAreas[holder.equipPointName].Add(holder);
                }
                else
                {
                    holderAreas[holder.equipPointName].Add(holder);
                }

                holder.SetActiveHolder(false);
                holder.SetActiveWeapon(false);
            }
        }
    }
}
```

```
1 reference
public void EquipWeapon(vEquipArea equipArea, vItem item)
{
    var slotsInArea = equipArea.ValidSlots;
    if (slotsInArea != null && slotsInArea.Count > 0 && holderAreas.ContainsKey(equipArea.equipPointName))
    {
        for (int i = 0; i < slotsInArea.Count; i++)
        {
            if (slotsInArea[i].item != null)
            {
                var holder = holderAreas[equipArea.equipPointName].Find(h => slotsInArea[i].item.id == h.itemID &&
                (equipArea.currentEquipedItem != item && equipArea.currentEquipedItem != slotsInArea[i]));

                if (holder)
                {
                    holder.SetActiveHolder(true);
                    holder.SetActiveWeapon(true);
                }
            }
            if (equipArea.currentEquipedItem != null)
            {
                var holder = holderAreas[equipArea.equipPointName].Find(h => h.itemID == equipArea.currentEquipedItem.id);
                if (holder)
                {
```

Fig. :(b) How to Align a Shooter Weapon AND left-hand IK

```
                StartCoroutine(ActiveHolder(holder, false, (itemManager.inventory != null && itemManager.inventory.isOpen)));
            }
        }
    }
}
```

```
reference
ublic void UnequipWeapon(vEquipArea equipArea, vItem item)

    if (holders.Length == 0 || item == null) return;

    if ((itemManager.inventory != null) && holderAreas.ContainsKey(equipArea.equipPointName))
    {
        var holder = holderAreas[equipArea.equipPointName].Find(h => item.id == h.itemID);
        if (holder)
        {
            var containsItem = equipArea.ValidSlots.Find(slot => slot.item == item) != null;
            holder.SetActiveHolder(containsItem);
            holder.SetActiveWeapon(containsItem);
        }
    }
```

```
loat timer;
reference
Enumerator ActiveHolder(vWeaponHolder holder, bool activeWeapon, bool immediat = false)

    if (!immediat)
        inEquip = true;

    timer = holder.equipDelayTime;
    holder.SetActiveHolder(true);

    if (!activeWeapon)
        holder.SetActiveWeapon(true);

    while (timer > 0 && !immediat)
    {
        yield return new WaitForEndOfFrame();
        timer -= Time.deltaTime;
```

Fig. (c): How to Align a Shooter Weapon AND left-hand IK

```
float timer;
1 reference
IEnumerator ActiveHolder(vWeaponHolder holder, bool activeWeapon, bool immediat = false)
{
    if (!immediat)
        inEquip = true;

    timer = holder.equipDelayTime;
    holder.SetActiveHolder(true);

    if (!activeWeapon)
        holder.SetActiveWeapon(true);

    while (timer > 0 && !immediat)
    {
        yield return new WaitForEndOfFrame();
        timer -= Time.deltaTime;
    }

    holder.SetActiveWeapon(activeWeapon);
    inEquip = false;
}
```

Fig.(d): How to Align a Shooter Weapon AND left-hand IK

5.9 Creating A New Shooter Controller

i) Programming Code:

```
ameByBandar                                    Invector.CharacterController.vThirdPersonController
using UnityEngine;
using System.Collections;

namespace Invector.CharacterController
{
    [vClassHeader("Third Person Controller")]
    36 references
    public class vThirdPersonController : vThirdPersonAnimator
    {
        #region Variables

        public static vThirdPersonController instance;

        #endregion

        0 references
        protected virtual void Awake()
        {
            StartCoroutine(UpdateRaycast()); // limit raycasts calls for better performance
        }

        0 references
        protected virtual void Start()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
                this.gameObject.name = gameObject.name + " Instance";
            }
            else
            {
                Destroy(this.gameObject);
                return;
            }

            #if !UNITY_EDITOR
                Cursor.visible = false;
```

Fig.(a): Creating A New Shooter Controller

```
/// <param name="referenceTransform"></param>
2 references
public virtual void RotateWithAnotherTransform(Transform referenceTransform)
{
    var newRotation = new Vector3(transform.eulerAngles.x, referenceTransform.eulerAngles.y, transform.eulerAngles.z);
    transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.Euler(newRotation), strafeRotationSpeed * Time.fixedDeltaTime);
    targetRotation = transform.rotation;
}

#endregion

#region Check Action Triggers

/// <summary>
/// Call this in OnTriggerEnter or OnTriggerStay to check if enter in triggerActions
/// </summary>
/// <param name="other">collider trigger</param>
1 reference
public virtual void CheckTriggers(Collider other)
{
    try
    {
        CheckForAutoCrouch(other);
    }
    catch (UnityException e)
    {
        Debug.LogWarning(e.Message);
    }
}

/// <summary>
/// Call this in OnTriggerExit to check if exit of triggerActions
/// </summary>
/// <param name="other"></param>
1 reference
public virtual void CheckTriggerExit(Collider other)
{
    AutoCrouchExit(other);
```

Fig.(b): Creating A New Shooter Controller

```
#region Locomotion Actions

2 references
public virtual void Sprint(bool value)
{
    if(value)
    {
        if (currentStamina > 0 && input.sqrMagnitude > 0.1f)
        {
            if (isGrounded && !isCrouching)
                isSprinting = !isSprinting;
        }
    }
    else if (currentStamina <= 0 || input.sqrMagnitude < 0.1f || actions || isStrafing && !strafeWalkByDefault && (direction >= 0.5 || direction <= -0.5 || speed <= 
    {
        isSprinting = false;
    }
}

1 reference
public virtual void Crouch()
{
    if (isGrounded && !actions)
    {
        if (isCrouching && CanExitCrouch())
            isCrouching = false;
        else
            isCrouching = true;
    }
}

3 references
public virtual void Strafe()
{
    isStrafing = !isStrafing;
}

1 reference
public virtual void Jump()
{
    if (customAction) return;
```


Fig.(c): A Creating New Shooter Controller

```
public virtual void CheckTriggerExit(Collider other)
{
    AutoCrouchExit(other);
}

#region Update Raycasts

1 reference
protected IEnumerator UpdateRaycast()
{
    while (true)
    {
        yield return new WaitForEndOfFrame();

        AutoCrouch();
        StopMove();
    }
}

#endregion

#region Crouch Methods

1 reference
protected virtual void AutoCrouch()
{
    if (autoCrouch)
        isCrouching = true;

    if (autoCrouch && !inCrouchArea && CanExitCrouch())
    {
        autoCrouch = false;
        isCrouching = false;
    }
}

2 references
protected virtual bool CanExitCrouch()
{
    // radius of SphereCast
    float radius = _capsuleCollider.radius * 0.9f;
```


Fig.(d): Creating A New Shooter Controller

5.10 Creating A New Camera State

i) Programming Code: fig.:

```
1 reference
public virtual void Jump()
{
    if (customAction) return;

    // know if has enough stamina to make this action
    bool staminaConditions = currentStamina > jumpStamina;
    // conditions to do this action
    bool jumpConditions = !isCrouching && isGrounded && !actions && staminaConditions && !isJumping;
    // return if jumpCondigions is false
    if (!jumpConditions) return;
    // trigger jump behaviour
    jumpCounter = jumpTimer;
    isJumping = true;
    // trigger jump animations
    if (input.sqrMagnitude < 0.1f)
        animator.CrossFadeInFixedTime("Jump", 0.1f);
    else
        animator.CrossFadeInFixedTime("JumpMove", .2f);
    // reduce stamina
    ReduceStamina(jumpStamina, false);
    currentStaminaRecoveryDelay = 1f;
}
```

```
1 reference
public virtual void Roll()
{
    if (animator.IsInTransition(0)) return;

    bool staminaCondition = currentStamina > rollStamina;
    // can roll even if it's on a quickturn or quickstop animation
    bool actionsRoll = !actions || (actions && (quickStop));
    // general conditions to roll
    bool rollConditions = (input != Vector2.zero || speed > 0.25f) && actionsRoll && isGrounded && staminaCondition && !isJumping;

    if (!rollConditions || isRolling) return;

    animator.SetTrigger("ResetState");
    animator.CrossFadeInFixedTime("Roll", 0.1f);
    ReduceStamina(rollStamina, false);
    currentStaminaRecoveryDelay = 2f;
```

Fig.(a): Creating A New Camera State

```
references
ublic class vThirdPersonCamera : MonoBehaviour

  private static vThirdPersonCamera _instance;
  3 references
  public static vThirdPersonCamera instance
  {
      get
      {
          if (_instance == null)
          {
              _instance = GameObject.FindObjectOfType<vThirdPersonCamera>();

              //Tell unity not to destroy this object when loading a new scene!
              //DontDestroyOnLoad(_instance.gameObject);
          }

          return _instance;
      }
  }
```

```
egion inspector properties
  public Transform target;

  [Tooltip("Lerp speed between Camera States")]
  public float smoothBetweenState = 3f;
  public float smoothCameraRotation = 8f;
  public float scrollSpeed = 10f;

  [Tooltip("What layer will be culled")]
  public LayerMask cullingLayer = 1 << 0;
  [Tooltip("Change this value If the camera pass through the wall")]
  public float clipPlaneMargin;
  public float checkHeightRadius;
  public bool showGizmos;
  public bool startUsingTargetRotation = true;
  public bool startSmooth = true;
  [Tooltip("Debug purposes, lock the camera behind the character for better align the states")]
  public bool lockCamera;
```

Fig.(b): Creating A New Camera State

```
#region hide properties
    [HideInInspector]
    public int indexList, indexLookPoint;
    [HideInInspector]
    public float offSetPlayerPivot;
    [HideInInspector]
    public float distance = 5f;
    [HideInInspector]
    public string currentStateName;
    [HideInInspector]
    public Transform currentTarget;
    [HideInInspector]
    public vThirdPersonCameraState currentState;
    [HideInInspector]
    public vThirdPersonCameraListData CameraStateList;
    [HideInInspector]
    public Transform lockTarget;
    [HideInInspector]
    public Vector2 movementSpeed;
    [HideInInspector]
    public vThirdPersonCameraState lerpState;
    private Transform targetLookAt;
    private Vector3 currentTargetPos;
    private Vector3 lookPoint;
    private Vector3 current_cPos;
    private Vector3 desired_cPos;
    private Quaternion lookTargetAdjust;
    private Camera _camera;
    private float mouseY = 0f;
    private float mouseX = 0f;
    private float currentHeight;
    private float currentZoom;
    private float cullingHeight;
    private float cullingDistance;
    private float switchRight, currentSwitchRight;
    private bool useSmooth;
    private bool isNewTarget;
```

Fig:(c) Creating A New Camera State

```
yBandar                    vThirdPersonCamera                    CullingRayCast(Vector3 f
{
    Init();
}

4 references
public void Init()
{
    if (target == null)
        return;

    _camera = GetComponent<Camera>();
    currentTarget = target;
    currentTargetPos = new Vector3(currentTarget.position.x, currentTarget.position.y + offSetPlayerPivot, currentTarget.position.z);
    targetLookAt = new GameObject("targetLookAt").transform;
    targetLookAt.position = currentTarget.position;
    targetLookAt.hideFlags = HideFlags.HideInHierarchy;
    if(startUsingTargetRotation) targetLookAt.rotation = currentTarget.rotation;
    // initialize the first camera state
    mouseY = currentTarget.eulerAngles.x;
    mouseX = currentTarget.eulerAngles.y;
    switchRight = 1f;
    currentSwitchRight = 1f;
    ChangeState("Default", false);
    currentZoom = currentState.defaultDistance;
    distance = currentState.defaultDistance;
    currentHeight = currentState.height;
    if(startSmooth) useSmooth = true;
}

0 references
void FixedUpdate()
{
    if (target == null || targetLookAt == null || currentState == null || lerpState == null) return;

    switch (currentState.cameraMode)
    {
        case TPCameraMode.FreeDirectional:
            CameraMovement();
            break;
        case TPCameraMode.FixedAngle:
            CameraMovement();
```

Fig:(d) Creating A New Camera State

5.11 Design and Creating AI

i) Programming Code:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

/*
//Selects what behaviours to use, and defines the parameters of the default behaviours
//Also handles melee
*/
namespace TacticalAI
{
    17 references
    public class BaseScript : MonoBehaviour
    {
        //Base Stuff
        //Scripts
        public TacticalAI.GunScript gunScript;
        public TacticalAI.SoundScript audioScript;
        public TacticalAI.RotateToAimGunScript headLookScript;
        public TacticalAI.AnimationScript animationScript;
        public TacticalAI.CoverFinderScript coverFinderScript;

        public Transform targetTransform;

        private LayerMask layerMask;

        //NavMeshAgent agent;
        public NavmeshInterface navmeshInterfaceClass = null;
        public NavmeshInterface navI;
        Transform myTransform;

        TacticalAI.TargetScript myTargetScript;

        //Speed and navigation stuff
        public float origAgentStoppingDist;

        public float sprintSpeed = 6.0f;
        public float runSpeed = 5.0f;
        public float alertSpeed = 4.0f;
        public float idleSpeed = 3.0f;
```

Fig.(a): Design and Creating AI

```
//Melee
public bool canMelee = false;
public float meleeDamage = 100;
public float timeBetweenMelees = 2.0f;
public float meleeRange = 2.0f;
public float timeUntilMeleeDamageIsDealt = 0.2f;
[HideInInspector]
public bool isMeleeing = false;

public bool canSprint = false;
public float distFromTargetToSprint = 25f;

5 references
public enum AIType
{
    Berserker = 0, Tactical = 1, Custom = 2, Skirmish = 3,
}

6 references
public enum IdleBehaviour
{
    Search = 1, Patrol = 2, Wander = 3, MoveToTransform = 4, Custom = 5,
}

30 references
public enum AIBehaviour
{
    GoToMoveTarget = 0, Cover = 1, Search = 2, Dodging = 3, Patrolling = 4, Wander = 5, UseDynamicObject = 6, InvestigateSound = 7, Custom = 8, MoveToTransform = 9, RunFr
}


public float minSkirmishDistFromTarget = 7f;
public float maxSkirmishDistFromTarget = 20f;
public bool canCrossBehindTarget = true;
public float maxTimeToWaitAtEachSkirmishPoint = 3f;

// Use this for initialization
0 references
void Awake()
```

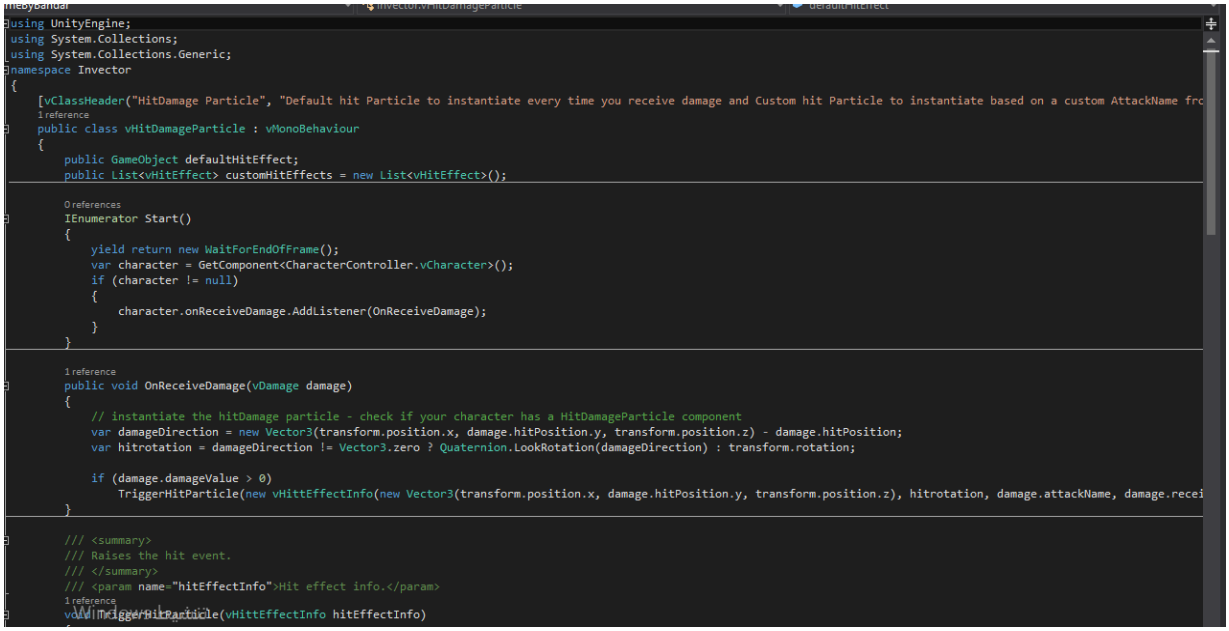
Fig.(b): Design and Creating AI

```
void Start()
{
    GetDefaultBehaviours();
    if (TacticalAI.ControllerScript.currentController != null)
    {
        layerMask = TacticalAI.ControllerScript.currentController.GetLayerMask();
    }
    else
    {
        this.enabled = false;
    }
    //StartCoroutine(WaitBetweenDodges());
    if (!ControllerScript.pMode)
    {
        StartCoroutine("AICycle");
    }
    else
    {
        StartCoroutine("PerformanceAICycle");
    }

}

float perfCycleTime = 2;

0 references
IEnumerator PerformanceAICycle()
{
    while (this.enabled)
    {
        if (currentBehaviour != null)
        {
            currentBehaviour.AICycle();
        }

        if (targetTransform && engaging && !usingDynamicObject && !isMeleeing && !inParkour && isDodging)
        {
```

Fig.(c): Design and Creating AI

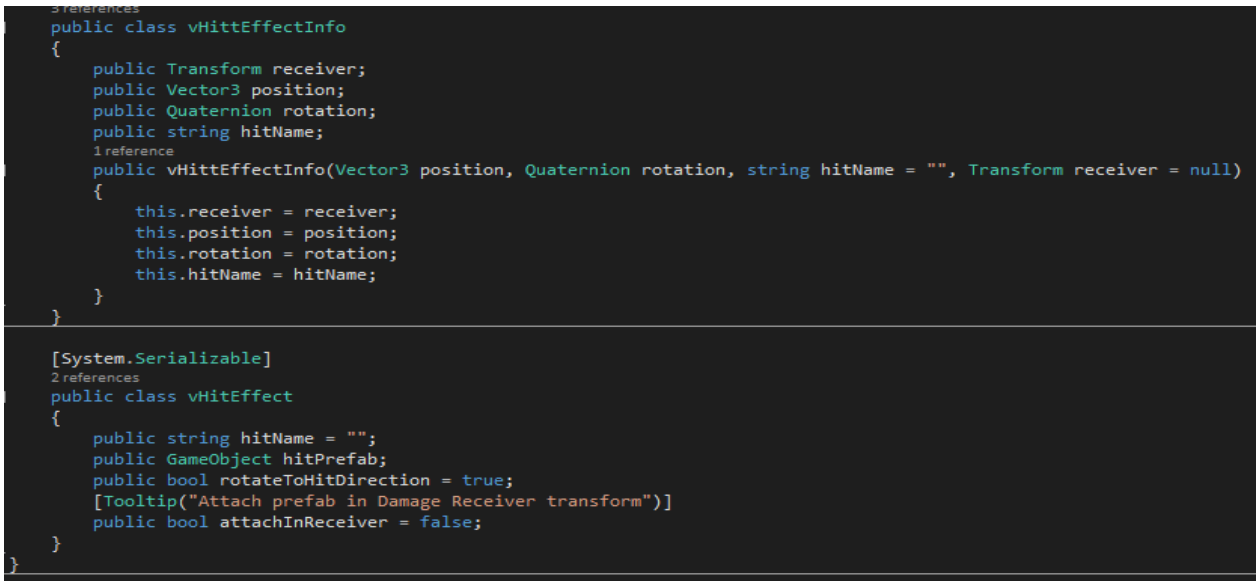
5.12 Damage Scripts

i) Programming Code



```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
namespace Invector
{
    [vClassHeader("HitDamage Particle", "Default hit Particle to instantiate every time you receive damage and Custom hit Particle to instantiate based on a custom AttackName fro
    1 reference
    public class vHitDamageParticle : vMonoBehaviour
    {
        public GameObject defaultHitEffect;
        public List<vHitEffect> customHitEffects = new List<vHitEffect>();

        0 references
        IEnumerator Start()
        {
            yield return new WaitForEndOfFrame();
            var character = GetComponent<CharacterController.vCharacter>();
            if (character != null)
            {
                character.onReceiveDamage.AddListener(OnReceiveDamage);
            }
        }

        1 reference
        public void OnReceiveDamage(vDamage damage)
        {
            // instantiate the hitDamage particle - check if your character has a HitDamageParticle component
            var damageDirection = new Vector3(transform.position.x, damage.hitPosition.y, transform.position.z) - damage.hitPosition;
            var hitrotation = damageDirection != Vector3.zero ? Quaternion.LookRotation(damageDirection) : transform.rotation;

            if (damage.damageValue > 0)
                TriggerHitParticle(new vHittEffectInfo(new Vector3(transform.position.x, damage.hitPosition.y, transform.position.z), hitrotation, damage.attackName, damage.recei
        }

        /// <summary>
        /// Raises the hit event.
        /// </summary>
        /// <param name="hitEffectInfo">Hit effect info.</param>
        1 reference
        void TriggerHitParticle(vHittEffectInfo hitEffectInfo)
```

Fig.(a): Damage Scripts



```
3 references
public class vHittEffectInfo
{
    public Transform receiver;
    public Vector3 position;
    public Quaternion rotation;
    public string hitName;
    1 reference
    public vHittEffectInfo(Vector3 position, Quaternion rotation, string hitName = "", Transform receiver = null)
    {
        this.receiver = receiver;
        this.position = position;
        this.rotation = rotation;
        this.hitName = hitName;
    }
}

[System.Serializable]
2 references
public class vHitEffect
{
    public string hitName = "";
    public GameObject hitPrefab;
    public bool rotateToHitDirection = true;
    [Tooltip("Attach prefab in Damage Receiver transform")]
    public bool attachInReceiver = false;
}
}
```

Fig.(b): Damage Scripts

```
3 references
public class vHittEffectInfo
{
    public Transform receiver;
    public Vector3 position;
    public Quaternion rotation;
    public string hitName;
    1 reference
    public vHittEffectInfo(Vector3 position, Quaternion rotation, string hitName = "", Transform receiver = null)
    {
        this.receiver = receiver;
        this.position = position;
        this.rotation = rotation;
        this.hitName = hitName;
    }
}

[System.Serializable]
2 references
public class vHitEffect
{
    public string hitName = "";
    public GameObject hitPrefab;
    public bool rotateToHitDirection = true;
    [Tooltip("Attach prefab in Damage Receiver transform")]
    public bool attachInReceiver = false;
}
}
```

Fig.(c): Damage Scripts

5.13 Ace Arcade Missiles

i) Programming Code:

```
ameByBandar                    GuidanceType                    Pursuit
using UnityEngine;
using System.Collections;

3 references
public enum GuidanceType
{
    Pursuit,
    Lead
}

10 references
public enum UpdateType
{
    FixedUpdate,
    Update
}

[RequireComponent(typeof(Rigidbody))]
[RequireComponent(typeof(CapsuleCollider))]
[RequireComponent(typeof(AAMissileEffects))]
13 references
public class AAMissile : MonoBehaviour
{
    new Transform transform;
    new Rigidbody rigidbody;
    new CapsuleCollider collider;

    [Header("General Parameters:")]
    [Tooltip("Run movement code in fixed update versus update.\n\nIf you notice jittery movement, try changing this. Fixed Update is typically used for rigidbody based projects."
    public UpdateType movementUpdateCycle = UpdateType.Update;

    [Tooltip("Run guidance code in fixed update versus update.\n\nIf your target has a rigidbody and moved through physics, set this to Fixed.")]
    public UpdateType targetUpdateCycle = UpdateType.Update;

    [Tooltip("Transform of the target. Typically assigned by launcher that shot the missile, but can be manually assigned for a missile already in the scene. If null on launch, t
    public Transform target;

    [Tooltip("Launching object. Typically assigned by the launcher and only needs to be assigned if manually launching a missile already in the scene. When assigned, this will pr
    public Transform ownShip;

    [Tooltip("Position where this missile attaches to hardpoint style launchers. If not assigned, this will automatically search for a GameObject named \"Attach\". If no such Gam
    public Transform attachPoint;
```

Fig.(a): Ace Arcade Missiles

```
[Tooltip("Position where this missile attaches to hardpoint style launchers. If not assigned, this will automatically search for a GameObject named \"Attach\". If no such Gam
public Transform attachPoint;

[Header("Missile parameters:")]

[Tooltip("Pursuit flies directly towards the target. Lead will fly ahead to intercept, making it significantly more difficult to dodge.")]
public GuidanceType guidanceType = GuidanceType.Pursuit;

[Tooltip("How far off boresight the missile can see the target. Also restricts how far the missile can lead.")]
public float seekerCone = 45.0f;

[Tooltip("How far off boresight the missile can see the target. Also restricts how far the missile can lead.")]
public float seekerRange = 5000.0f;

[Tooltip("When true, initial speed will be taken from either the velocity passed into the Launch function, or from the forward velocity of the missile after a drop launch if
public bool overrideInitialSpeed = false;

[Tooltip("Velocity that the missile has immediately on ignition.")]
public float initialSpeed = 0.0f;

[Tooltip("How long the missile will accelerate. After this, the missile maintains a constant speed.")]
public float motorLifetime = 3.0f;

[Tooltip("How much speed per second the missile will gain after launch.")]
public float acceleration = 15.0f;

[Tooltip("How many degrees per second the missile can turn.")]
public float turnRate = 45.0f;

[Tooltip("After this time, the missile will self-destruct. Timer starts on launch, not motor activation.")]
public float timeToLive = 15.0f;

[Header("Drop options:")]

[Tooltip("Velocity (in local space) at which the missile will be ejected from its launch point.")]
public Vector3 ejectVelocity = Vector3.zero;
```


Fig.(b): Ace Arcade Missiles

```
private Vector3 targetPosLastFrame;
private Quaternion guidedRotation;

// Used to prevent lead markers from getting huge when missiles are very slow.
private const float MINIMUM_GUIDE_SPEED = 1.0f;

0 references
public bool MissileLaunched { get { return isLaunched; } }
2 references
public bool MotorActive { get { return motorActive; } }

0 references
private void Awake()
{
    transform = GetComponent<Transform>();
    rigidbody = GetComponent<Rigidbody>();
    collider = GetComponent<CapsuleCollider>();
    missileEffect = GetComponent<AAMissileEffects>();
}

0 references
private void Start()
{
    // Sets it so that missile cannot collide with the thing that launched it.
    if (ownShip != null)
    {
        foreach (Collider col in ownShip.GetComponentsInChildren<Collider>())
            Physics.IgnoreCollision(collider, col);
    }

    // Find attach point if necessary.
    if (attachPoint == null)
    {
        Transform[] potentialAttach = GetComponentsInChildren<Transform>();
        foreach (Transform xform in potentialAttach)
            if (xform.name == "Attach")
                attachPoint = xform;

        if (attachPoint == null)
            Debug.LogError("No attach point found for missile " + transform.name + ". Using missile center instead.");
```


Fig.(c): Ace Arcade Missiles

5.14 Reference:

1. **tactical shooter AI**

   https://assetstore.unity.com/packages/tools/ai/tactical-shooter-ai-47248

2. **In vector**

   https://assetstore.unity.com/packages/templates/systems/third-person-controller-shooter-template-84583

3. **TerrainComposer2**

   https://assetstore.unity.com/packages/tools/terrain/terraincomposer-2-65563

4. **Physics Tank unity**

   https://assetstore.unity.com/packages/templates/systems/physics-tank-maker-50485

5. **Packt Publishing**
6. www.packtpub.com.
7. www.almo3allem.com
8. www.reskinning-app.com
9. www.quora.com
10. www.youtube.com
11. . www.mohammadaltahan.tk
12. .www.gamasutra.com
13. www.gamedev.net
14. sloperama.com/advice.
15. www.cs.tufts.eduf
16. www.scribd.com
17. code.tutsplus.com

Demo

https://youtu.be/M3dqBT15GZA?si=SXC2t1m4ZUcS7Ydq

Contact me :bandarr.ameen@gmail.com

https://www.facebook.com/bandarr.ameen