A Project Report on

Video-Game Development in Unreal Engine

Submitted in partial fulfillment of the requirements for the award
of the degree of

Bachelor of Engineering

in

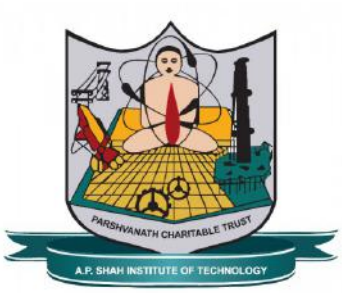
Computer Engineering

by

Falguni Mukesh Tailor(16102031)

Under the Guidance of

Prof. Sachin Malave



Department of Computer Engineering
A.P. Shah Institute of Technology
G.B.Road,Kasarvadavli, Thane(W), Mumbai-400615
UNIVERSITY OF MUMBAI

Academic Year 2019-2020

Approval Sheet

This Project Report entitled ***"Developing a videogame using unreal engine based on a four stages methodology"*** Submitted by ***Falguni Mukesh Tailor(16102031)***is approved for the partial fulfillment of the requirenment for the award of the degree of ***Bachelor of Engineering*** in ***Computer Engineering*** from ***University of Mumbai***.

Guide
Prof. Sachin Malave
Head Department of Computer Engineering

Place:A.P.Shah Institute of Technology, Thane
Date:

CERTIFICATE

This is to certify that the project entitled ***"Video-Game Development in Unreal Engine"*** submitted by ***"Falguni Mukesh Tailor" (16102031)*** for the partial fulfillment of the requirement for award of a degree ***Bachelor of Engineering*** in ***Computer Engineering***,to the University of Mumbai,is a bonafide work carried out during academic year 2019-2020.

Prof. Sachin Malve(Guide)
Head Department of Computer Engineering

Dr. Uttam D.Kolekar
Principal

External Examiner(s)

1.

2.

Place:A.P.Shah Institute of Technology, Thane
Date:

Declaration

I declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Falguni Mukesh Tailor 16102031)

Date:

Abstract

The goal of this project is to develop a simple game using Unreal Engine based on an agile methodology that is economic, sustainable and practical. This methodology comprises of four stages, viz. pre-production, production, testing and post-production. I achieve to prove the applicability of the four-stage methodology to make a first person game that includes switchable characters. The three major game development engines available to free-lance programmers are the Crytek engine, the Unity Engine and the Unreal engine, of which CrytekEngine is proprietary. The code generated in the backend of Unity Engine is in C, while that in Unreal Engine is Visual C++.

UE4 also makes usage of Blueprints and Environment Query System to program Artificial Intelligence and game mechanics. The Agile framework shall involve updating the game post release for newer versions, as well as removal of bugs in existing versions via beta testing

I achieve to prove the applicability of the four stages methodology since I made a high quality game in a short period of time, using limited resources. The breathtaking part about the game would be that, it shall be void of advertisements, post release DLC or microtransactions.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Video game development is the process of creating a video game. Game development is a software development process, as a video game is software with art, audio and gameplay. Planning is important for individual and group projects alike. One method employed for game development is agile development. It is based on iterative prototyping, a subset of software prototyping. This method is effective because most projects do not start with a clear requirement outline. A popular method of agile software development is Scrum

1.2 History

During the 1940s and 1950s, computers took up entire rooms and were so expensive that only universities and large companies could afford them. Games like tic-tac-toe were excellent ways to attract public interest and support. Computer programmers were able to learn from the creation of games as well because it allowed them to break away from the usual subroutines and challenge the computer's capabilities. It was this mindset that led a group of MIT students during the 1960s to create one of the first and most ground-breaking computer games.

1.3 Evolution

Later, sometime in the late 1970s, came the arcade machines (also called coin operated machines). The first popular "arcade games" included early amusement-park midway games such as shooting galleries, ball-toss games, etc. In 1966, Sega introduced an electro-mechanical game called Periscope – an early submarine simulator and light gun shooter which used lights and plastic waves to simulate sinking ships from a submarine.

Chapter 2

Literature Review

Video games have long been a part of entertainment well early since the 1970s. But they were very expensive, to say the least. Even the resources for developing such games were very limited, and mostly closed source. With the advent of recent computers (mostly due to their computing power, hardware, memory usage, resource allocation, and being open source), it is now possible to develop simple games using minimum expense and efforts..

Recent developments in computer graphics have concentrated on advancements in hardware and software equally. Greater computing power equals greater yield.One of the latest research areas in computer science and computer graphics is augmented reality and virtual reality. Virtual reality is an interactive computer-generated experience taking place within a simulated environment. It incorporates mainly auditory and visual feedback but may also allow other types of sensory feedback. On the other hand, augmented reality is a technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.

In the University of Massey, an Unreal Engine 4 simulator was made using Virtual Reality tools. The objective was to submerge the player into a virtual world where we could see a movie on real time. The immersion in a virtual world turned out to be a richer experience compared to a common movie, the interaction with the scene made a world of a difference. In the same University, the students designed and developed a war simulator on Unreal Engine 3. The purpose of this project was to reduce the cost on a real-life scenario, where the soldiers can attack the medics and the loss of one of these translates into a high cost due to the lack of specialist on this field. To solve the problem, the war simulator offers battlefield sceneries where the player can take the role of the medic and do the tasks given. Unreal Engine allowed the developers to create realistic environments and simulate weathering effects that made the interaction feel realistic. On the other hand, Christian Rubino, from the RMIT University made an investigation about real time optimization in games using Unreal Engine 2. The first and biggest differences between the cinema and videogames were the limitations with the hardware and frames per second (FPS). It is necessary to optimize the environments so the calculations made on real time can be greater than 30 FPS. To solve the problem Unreal Engine offers a complex architecture where is possible to reach high frames with a good aesthetic quality. Using Level of Detail (LOD) simplifies the 3D models depending on the viewing distance of the camera, reducing the number of polygons rendered on the screen. The collisions, which is a system that allows the objects collide

with others inside the level are pre-calculated to create a simplified version of the 3D model silhouette. Using Unreal Engine's tools made possible to increase the visual quality of the videogame's environments without having any performance issues.

Besides Unreal Engine, there are other videogame developer tools focused on other technologies like HTML5. For instance, Michael Weeks, from the Georgia's University developed a videogame using HTML5 and Javascript programming languages. The purpose of this project was to show that is possible to create a videogame only using the two technologies mentioned before. The videogame was an action platformer game using 2D concepts. It was in a 2D space where the Tiling textures were used to create the environment. A Tiling textures are used to duplicated them inside the level instead of create a much bigger texture that can take more memory space. Using this made possible to create a 2D type videogame. In addition, there are other technologies than HTML5. For instance, students of Fortaleza's University experimented on developing a videogame called Funcopter in the shortest time possible using the Unity 3D Game Engine. It was designed to run on mobile devices, so the simplicity of the game had an important role in the development and planning stages of the project. With that said, the team only focused in only two stages: Preproduction and Production. During the Preproduction stage, the documentation, storyboards and core concepts of the game were avoided to save production time. They also analyzed games with similar mechanics to take the best out of them and incorporated them into FunCopter. For this case, the simplicity on the development of the game took an important role in the scope of the game. Unity 3D turned out to be a good option to finish the game in 2 days due to it was a game made for mobile devices.

Students from Lisbon University, made a research of how lightning in games can give a richer user experience. During the research, the found techniques that allow the player to submerge into another universe if they are used in the right way. The lights must be placed in a smart way to call the attention of the player, due to technical limitations lights with much higher quality should only be used on places where the player will interact and lights with the lowest quality in shadows should be placed on inaccessible zones. The contrast of colors and illumination placed in important places turned out to be an important aspect to guide the player to the main objective of the game. These techniques resulted to be essential to create a high quality product and a better user experience.

Chapter 3

Problem Statement

A videogame that receives overwhelmingly positive reviews on steam is a one that has the perfect gameplay, a well-balanced story between main missions and side-quests, proper graphics optimization for all platforms including PC, XBOX ONE and PS4, a reasonable price and no needless post release DLC. This actually was the case when EA ruled the gaming market in the early 2000s, while as of today, it is rare for a production company to release a complete game that does not involve micro-transactions.

This has annoyed all gamers alike as they demand a standalone without pay-to-win DLC. This project intends to solve a major fraction of this gamer dilemma whilst deploying the finished product on the Epic Games Store.

Chapter 4

Scope and Objectives

The Unreal game engine makes use of C++ and Blueprints to efficiently render skeletal meshes, materials, assets and integrate them into a simulated environment. The infractions caused by human beings have been visually demonstrated by Artificial Intelligence (BOTS). The AI has been rationalized using behaviour trees, environmental query system to replicate the behaviour of humans and animals.The AI plays the role of the antagonists which deteriorate the environment around an industrial ecosystem. The player has a set of objectives to accomplish to restore balance in the habitat and perform some side quests parallel to go with the mission objective. The AI have a clearly defined range of interaction logically designed using the BTs. The interactive cases get execute one at a time. Certain nodes in the BTs have an embedded Environmental Query System which acts as a separate flow of execution. Once the ecosystem gets restored to its initial state, the flora and fauna return to initial status.The Unreal Engine version used for this project is 4.22; it's the first IDE that supports ray tracing and offers advanced AI scripts. The particle effects rendered in the engine emphasize the present and foreseeable effects on the environment. Updated lighting effects act as visual boost.

4.1 Objectives

Why : Purpose of this videogame
The prime aim of this project is to make its users aware of the current deterioration of the environment via air pollution, poaching and felling of trees, how they affect us and what steps we can and should take to solve these major issues in the form of a simulation.

Who: Market Conditions
The project also highlights the newly emerging videogame industry as a sub-set of the entertainment industry and focuses mostly towards youngsters.

What: The Point of creating the game
Until the early 10s creating a videogame was considered costly and outside the scope of a small production firm. With the introduction of video-game development engines by Epic Games and Unity, it has become possible for anyone to develop one and craft a source of income as videogame industry is hugely profit-based.

Chapter 5

Technology Stack

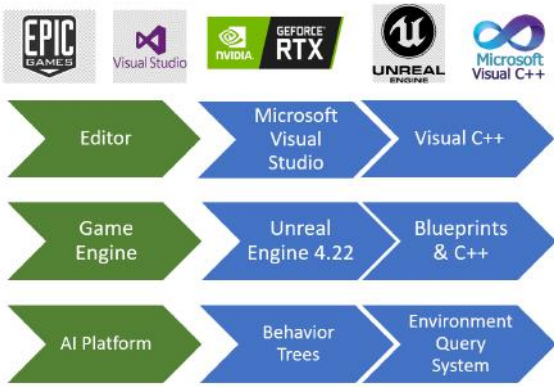


Figure 5.1: Technology Stack

Unreal Engine is a part of Epic games and every finished game is deployed on the Epic Games Store. The game uses assets that are created within the material editor or those available from the Epic Games Store. The logic for the game is designed using the Microsoft Visual Studio IDE in the Visual C++ programming language. The major difference between C++ and Visual C++ is that in the later, for an unreal project, the compiler ignores UFUNCTIONS() and UPROPERTIES() declared in the header files and makes use of engine functions in the editor. An important aspect of creating video games is rendering. Stronger the GPU, smoother and faster is the rendering. The GPU used in this project is the EVGA Nvidia GeForce RTX 2070.

Chapter 6

Benefits for the environment

This project aims to bring into notice the importance of environmental aspects in our everyday life and the deterioration caused by humanity to our surroundings including the atmosphere, flora and fauna, as well as aquatic bodies and make the users of this application aware of their role to the environment and the society.

The user can exhaust the in-game mechanics to alter the outcomes of each successive mission on the game environment in the form of particle effects rendered in game and the change in the behaviour of the AI. This acts as a direct simulation of how it works in the real world, the only difference being, out there, the changes are irreversible, but this simulation make an impact on users towards their responsibility, especially now that the Amazon rainforest has been adversely depleted, and species are on the verge of extinction.

Chapter 7

Benefits for the Society

Video games have long been a part of entertainment well early since the 1970s. But they were very expensive, to say the least. Even the resources for developing such games were very limited, and mostly closed source. With the advent of recent computers (mostly due to their computing power, hardware, memory usage, resource allocation, and being open source), it is now possible to develop simple games using minimum expense and efforts.

Chapter 8

Applications

One of the latest research areas in computer science and computer graphics is augmented reality and virtual reality. Virtual reality is an interactive computer-generated experience taking place within a simulated environment. It incorporates mainly auditory and visual feedback but may also allow other types of sensory feedback. On the other hand, augmented reality is a technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view. Recent developments in computer graphics have concentrated on advancements in hardware and software equally. The latest advancements in videogame industry have been made in the healthcare sector.

The in-focus videogame highlights creation and usage of artificial intelligence using behaviour trees, based on Environmental Query System that runs in a separate blueprint in the form of a stack. The Behaviour Trees follow a flow of logic that executes sequentially in pre-order traversal, such that the first node has the highest priority and last one has the lowest, showing us how the AI thinks. AI can also be hardcoded to perform a specific task.

A heavily rendered environment is aesthetically pleasing to all the viewers and attracts a whole generation quite easily. Sending a message to a wider audience through the means of a game is relatively easier as its objectives subconsciously tend to affect the mindset of a youth better than the currently available means

Chapter 9

Storyline

9.1 The Green Kingdom

It's another fine day around the countryside. Ash is about to take a early morning stroll before he comes home back for breakfast. Serena's still asleep. He feels he's happily married. He is out on his front porch, fills up his bird feed and is on his way. He's been doing this for quite a while now. Out of the usual, he takes a shortcut for the lake, through the Viridian forest. Something doesn't feel right... He tries to investigate. A quarter mile through the forest, he's to far in to go back empty-handed now. There's mild smog around. Must be that it's the winter season; but why in the Viridian...a place where one can always see the clear blue skies. He keeps moving on.

It's 7:30 a.m. Serena's awake now. Ash is nowhere to be seen. There's no note on the fridge either. She thinks to herself, "Maybe Ash might have rushed in a hurry or he didn't wanna wake me up. I'll make some tea, the aroma might certainly swing him around." Ash is midway through the forest. He sees a wide area of flat lands and a pile of wooden logs freshly cut. It rained overnight, but there's no moss on them. Some water has seeped into the axe marks. The sun is out and bright. Right in the center there's an industry under construction. It already has a signboard, "Vogler Enterprises". "What's going on here ? When was this authorized ?" Ash is overwhelmed. He notices a dead deer, a few on the verge of dying.

There is thick dark smoke everywhere in the vicinity."this is going to be a long day." He sees a old guard on duty; there is no one else. Looks like they have gone on a vacation, or maybe the construction was halted. Wither way this place looks deserted. "Excuse me, I don't mean to be rude, but could you fill me in, what in God's name is going on here ?" "Sir, the owner suffered a major loss recently has sent all employees on leave. That's all I know." "While st leaving this monstrosity behind ?" Ash furiously enquires. "It's hard to breathe here. Can I take a look around ?" "Sir, You need to leave" the guard is losing his cool now. "No way, I am going in. Either you step aside, or I'll move you myself." The argument is heated now. There is exchange of blows. Eventually Ash prevails. The guard has taken the high road. It will be a while before he notifies the authorities. Ash moves inside the building. Meanwhile, the tea has gone cold for a while now. Ash hasn't returned. Serena's worried. Why wouldn't she be ? Ash has never been gone out for so long. She steps outside the house looking for him.

Serena has retraced the path Ash takes everyday to the lake. The smog can seen to the other side. "Maybe Ash got curious ?" Visibility is low. "ASH !!! ASH !!!" She yells... "Over here !" It's Ash's voice. "Where are you ? I can't see." Ash comes out of the building and comes towards Serena. "Oh..there you are...You had me bothered. You could've at least woken me up..." "It's alright, dear. I'm here" Ash interrupts her. He explains the situation to Serena. "We have to do something. We grew up here and I cannot let our neighbourhood die out on us. You stay right here and give me a call if you see anyone. I'm going to find the cause of the smoke and see if I can fix it. If not, we'll call the authorities." "Okay" says Serena, "But we really shouldn't be here. We are breaking into private property. I'll help you out, so we can get it over soon."

They get to the boiler room. It's misbehaving. Sparks can be seen flying out of it. "Oh my God, what do we do now ? It can explode any second !" "Take it easy Serena, I am turning it off. There, now to put out the fire. There must be an extinguisher nearby." Serena finds one in the adjoining room and hands it over to Ash. He sprays it all over the room. The fire is out and the smoke has stopped as well. "Let's get out of here now, before anyone sees us." They start running towards home. "Ash wait ! these deer ! We can't just leave them here; we have to save them." "Right...do one thing press hard on the wounds, while I gather some goldenrod leaves. I'll be back in a flash." Ash arrives with the leaves and crafts a healing recipe. He gives it to Serena to heal the deer. "We also have to replenish them. Let's quickly go home and do the rest." Ash and Serena take the two deer with them through the route Ash had discovered in the morning.

A day has passed. The deer have started to walk again, a few more days and they can

be released into the wild again. Ash and Serena, both had an exceptional experience, but they are happy they have saved lives. Everything is back to normal, what if Ash had never taken that unusual turn ?

Chapter 10

Proposed System Architecture/Working

During the development of the C++ shooter game, stages meeting the standards of the IGDA (International Game Developer Association) were followed in accordance to the Unreal Academy were used. Which are: Critical Game Studies, Games and Society, Game Design, Game Programming, Visual Design, Audio Design, Interactive Storytelling, Game Production and Business of Gaming. Since this was an academic project, business and marketing of the same was not looked at.

10.1 The Four Stage Methodology

Before starting out, it was necessary to plot the development cycle in terms of stages. For this purpose, we used a four-stage methodology that consisted of four different stages:

- Pre-production
- Production
- Testing
- Post-production

Each stage had been undergone through with caution and proper experimentation.

Chapter 11

Pre-production

In the initial stage, the requirements and information about the project were collected and met.

Hardware Requirements According to wiki.unrealengine.com,

- Quad-core Intel or AMD processor, 2.5GHz or Faster.
- Nvidia GeForce 470 GTX or AMD Radeon G870 HD Series card or higher.
- 8 GB of RAM(Recommended).

After verifying the critical requirements, three platforms of succeeding generations of Intel Processors and Graphics Cards were chosen to implement the project.

Software used:

- Unreal Engine 4
- Visual Studio 2017 Community Edition
- Autodesk Maya 2018

Hardware used:

Platform 1:

- Intel Core i5 8600K (8th Gen)
- EVGA Nvidia GeForce RTX 2070 (8 GB DDR6)
- 16 GB DDR4 3000MHz RAM
- 22” FHD Screen

Platform 2:

- Intel Core i7 6700HQ (6th Gen)
- Nvidia GeForce GTX 960M
- 16 GB DDR3L 1666MHz RAM
- 15.6” 4K Screen (HDR)

After gathering feedback about the type of game to develop, it was decided to go with a simpler approach and then build on it further.

Chapter 12

Production

During the production stage, all the necessary asset files were created within Unreal Engine itself, while external assets, such as audio files, were imported.Unreal Engine contains 2 different types of 3D models: a static mesh and a skeletal mesh. Static meshes are not used for movement, while skeletal meshes use vertices to connect their joints, hence can be used for movement.Point lights are used for luminosity/radiance/particle effects.

The Character Map

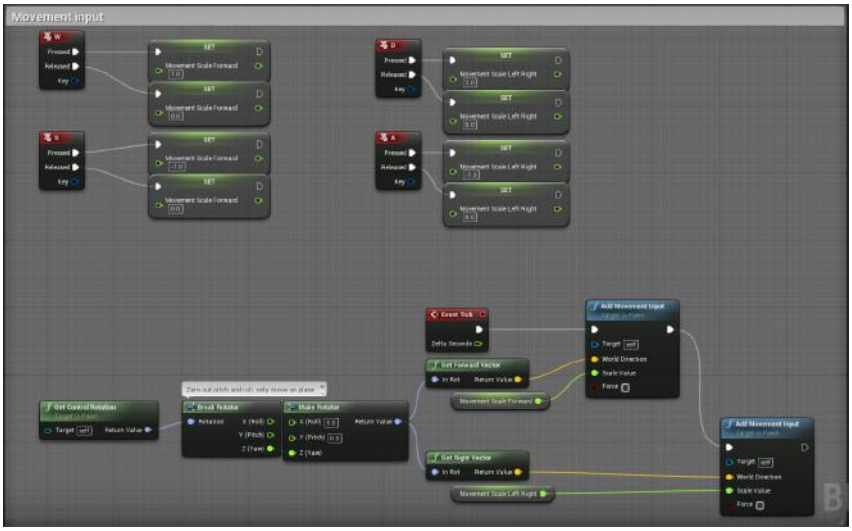


Figure 12.1: Third Person Character Movement Blueprint

12.1 Characters

The sample characters used for this project are Paragon props that were released by Epic Games for with general licence for all Unreal developers.

Figure 12.2: Male Character

Figure 12.3: Female Character

Chapter 13

The C++ Classes

13.1 The Unreal Build Tool

UE4 is consists of multiple modules. Every one of them has their own .build.cs file that controls the build of that module, including options for defining module dependencies, additional libraries, include paths, etc. These modules are in DLLs by default and are altogether loaded using the ProjectName.build.cs executable file.

```
using UnrealBuildTool;

public class UnrealAcademyProject : ModuleRules
{
   public UnrealAcademyProject(ReadOnlyTargetRules Target) : base(Target)
   {
      PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

      PublicDependencyModuleNames.AddRange(new string[] { "Core", "
   CoreUObject", "Engine", "InputCore" });

      PrivateDependencyModuleNames.AddRange(new string[] {  });
   }
}
```

13.2 The Character Class

The Third Person Character Class is the general class used to assume a player pawn in Unreal Engine. Inside the C++ editor, the pawn can be assigned movement and interactions, while the blueprint editor also allows to provide additional skins and particle effects and animations to the pawn.

```
#HEADER
#pragma once
```

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "SCharacter.generated.h"

UCLASS()
class UNREALACADEMYPROJECT_API ASCharacter : public ACharacter
{
   GENERATED_BODY()

public:
   // Sets default values for this character's properties
   ASCharacter();

protected:
   // Called when the game starts or when spawned
   virtual void BeginPlay() override;

public:
   // Called every frame
   virtual void Tick(float DeltaTime) override;

   // Called to bind functionality to input
   virtual void SetupPlayerInputComponent(class UInputComponent*
   PlayerInputComponent) override;

};

#CPP FILE

#include "SCharacter.h"

// Sets default values
ASCharacter::ASCharacter()
{
   // Set this character to call Tick() every frame.  You can turn this off
   to improve performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;

}
// Called when the game starts or when spawned
void ASCharacter::BeginPlay()
{
   Super::BeginPlay();

}

// Called every frame
```

```
void ASCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

}
// Called to bind functionality to input
void ASCharacter::SetupPlayerInputComponent(UInputComponent*
    PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

}
```

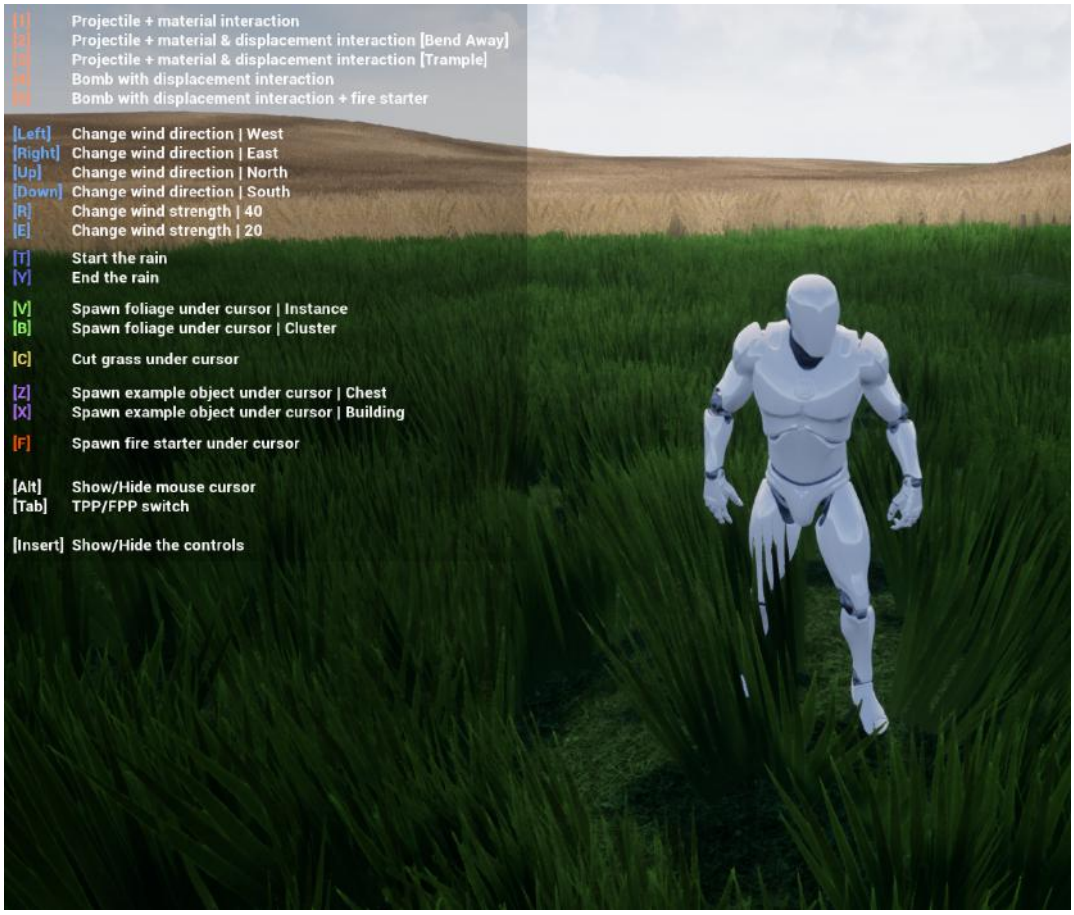


Figure 13.1: Third Person Character Mesh and UI

13.3 Health Component

```
Health Component Header
#include "SHealthComponent.h"

// Sets default values for this component's properties
USHealthComponent::USHealthComponent()
{


   DefaultHealth = 100;
}


// Called when the game starts
void USHealthComponent::BeginPlay()
{
   Super::BeginPlay();

   AActor *MyOwner = GetOwner();
   if (MyOwner)
      MyOwner->OnTakeAnyDamage.AddDynamic(this, &USHealthComponent::
   HandleTakeAnyDamage);
   Health = DefaultHealth;
}

void USHealthComponent::HandleTakeAnyDamage(AActor* DamagedActor, float
    Damage, const class UDamageType* DamageType, class AController*
    InstigatedBy, AActor* DamageCauser);
{
   if (Damage <= 0.0f)
      return;
   Health = FMath::Clamp(Health - Damage, 0.0f, DefaultHealth);
   UE_LOG(LogTemp, Log, ("Health Reduced: %s"), *FString::SanitizeFloat(
   Health));
}

Health Component Cpp File

#pragma once

#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "SHealthComponent.generated.h"
```

```
UCLASS( ClassGroup=(COOP), meta=(BlueprintSpawnableComponent) )
class UNREALACADEMYPROJECT_API USHealthComponent : public UActorComponent
{
   GENERATED_BODY()

public:
   // Sets default values for this component's properties
   USHealthComponent();

protected:
   // Called when the game starts
   virtual void BeginPlay() override;

   UPROPERTY(BlueprintReadOnly, Category = "HealthComponent")
      float Health;


   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HealthComponent")
      float DefaultHealth;

   void USHealthComponent::HandleTakeAnyDamage(AActor* DamagedActor, float
   Damage, const class UDamageType* DamageType, class AController*
   InstigatedBy, AActor* DamageCauser);

};
```

Figure 13.2: Health Component

13.4 Adding a environmental aspect to the set-A Black-hole

```
#HEADER
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "BlackHole.generated.h"

class USphereComponent;
class UStaticMeshComponent;

UCLASS()
class FPSGAME_API ABlackHole : public AActor
{
   GENERATED_BODY()

public:
   // Sets default values for this actor's properties
   ABlackHole();

protected:

   UPROPERTY(VisibleAnywhere, Category = "Components")
      UStaticMeshComponent *MeshComp;

   UPROPERTY(VisibleAnywhere, Category = "Components")
      USphereComponent *SuckInside;
```

```
    UPROPERTY(VisibleAnywhere, Category = "Components")
        USphereComponent *Attract;

    UFUNCTION()
    void OverlapInnerSphere(UPrimitiveComponent *OverlappedComponent, AActor
    *OtherActor, UPrimitiveComponent *OtherComp, int32 OtherBodyIndex, bool
    bFromSweep, const FHitResult &SweepResult);

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};


#SOURCE CPP FILE

#include "BlackHole.h"
#include "Components/SphereComponent.h"
#include "Components/StaticMeshComponent.h"
#include <Components/PrimitiveComponent.h>
//#include <Engine/EngineTypes.h>                                    //Defines
     SweepResult
//#include <Steamworks/Steamv139/sdk/public/steam/steamtypes.h>   //Defines
    bFromSweep
//#include <GameFramework/Actor.h>

// Sets default values
ABlackHole::ABlackHole()
{
    // Set this actor to call Tick() every frame.  You can turn this off to
    improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    MeshComp = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshComp"))
    ;
    MeshComp->SetCollisionEnabled(ECollisionEnabled::NoCollision);
    RootComponent = MeshComp;

    SuckInside = CreateDefaultSubobject<USphereComponent>(TEXT("SuckInside"))
    ;
    SuckInside->SetSphereRadius(100);
    SuckInside->SetupAttachment(MeshComp);

    //Bind to event
    SuckInside->OnComponentBeginOverlap.AddDynamic(this, &ABlackHole::
```

```
    OverlapInnerSphere);

    Attract = CreateDefaultSubobject<USphereComponent>(TEXT("Attract"));
    Attract->SetSphereRadius(3000);
    Attract->SetupAttachment(MeshComp);



}

void ABlackHole::OverlapInnerSphere(UPrimitiveComponent *OverlappedComponent
    , AActor *OtherActor, UPrimitiveComponent *OtherComp, int32
    OtherBodyIndex, bool bFromSweep, const FHitResult &SweepResult)
{
    if (OtherActor)
    {
        OtherActor->Destroy();
    }
}


// Called when the game starts or when spawned


// Called every frame
void ABlackHole::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    //Find all overlapping components that can collide and may be physically
    simulating.
    TArray<UPrimitiveComponent*> OverlappingComponents;
    Attract->GetOverlappingComponents(OverlappingComponents);

    for (int32 i = 0; i < OverlappingComponents.Num(); i++)
    {
        UPrimitiveComponent *PrimeComponent = OverlappingComponents[i];
        if (PrimeComponent && PrimeComponent->IsSimulatingPhysics())
        {
            //Suck inside all the prime components if they exist and obey
    physics.
            const float SphereRadius = Attract->GetScaledSphereRadius();
            const float ForceStrength = -2500; // +ve to push everything away,
    -ve to suck inside.

            PrimeComponent->AddRadialForce(GetActorLocation(), SphereRadius,
    ForceStrength, ERadialImpulseFalloff::RIF_Constant, true);
```

```
        }
    }


}
```

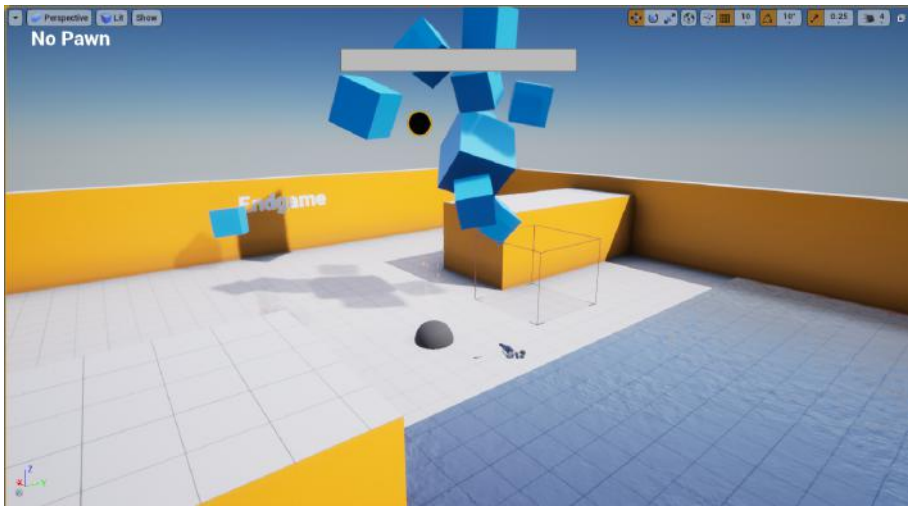


Figure 13.3: Blackhole Simulation

13.5 Setting up a crosshair

```
#HEADER
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/HUD.h"
#include "FPSHUD.generated.h"

class UTexture2D;

UCLASS()
class AFPSHUD : public AHUD
{
    GENERATED_BODY()

protected:

    /** Crosshair asset pointer */
```

```
    UTexture2D* CrosshairTex;

public:

    AFPSHUD();

    /** Primary draw call for the HUD */
    virtual void DrawHUD() override;

};

#SOURCE CPP FILE
```

13.6 Setting projectile physics for bullets

```
#HEADER
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "FPSProjectile.generated.h"


class UProjectileMovementComponent;
class USphereComponent;


UCLASS()
class AFPSProjectile : public AActor
{
    GENERATED_BODY()

protected:

    /** Sphere collision component */
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category= "Projectile")
    USphereComponent* CollisionComp;

    /** Projectile movement component */
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Movement")
    UProjectileMovementComponent* ProjectileMovement;
```

```
public:

    AFPSProjectile();

    /** called when projectile hits something */
    UFUNCTION()
    void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, FVector NormalImpulse, const FHitResult&
     Hit);



    /** Returns CollisionComp subobject **/
    USphereComponent* GetCollisionComp() const { return CollisionComp; }

    /** Returns ProjectileMovement subobject **/
    UProjectileMovementComponent* GetProjectileMovement() const { return
    ProjectileMovement; }
};

#SOURCE CPP FILE
#include "FPSProjectile.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Components/SphereComponent.h"



AFPSProjectile::AFPSProjectile()
{
    // Use a sphere as a simple collision representation
    CollisionComp = CreateDefaultSubobject<USphereComponent>(TEXT("SphereComp
    "));
    CollisionComp->InitSphereRadius(5.0f);
    CollisionComp->SetCollisionProfileName("Projectile");
    CollisionComp->OnComponentHit.AddDynamic(this, &AFPSProjectile::OnHit);
    // set up a notification for when this component hits something blocking

    // Players can't walk on it
    CollisionComp->SetWalkableSlopeOverride(FWalkableSlopeOverride(
    WalkableSlope_Unwalkable, 0.f));
    CollisionComp->CanCharacterStepUpOn = ECB_No;

    // Set as root component
    RootComponent = CollisionComp;

    // Use a ProjectileMovementComponent to govern this projectile's movement
```

```
    ProjectileMovement = CreateDefaultSubobject<UProjectileMovementComponent
    >(TEXT("ProjectileComp"));
    ProjectileMovement->UpdatedComponent = CollisionComp;
    ProjectileMovement->InitialSpeed = 3000.f;
    ProjectileMovement->MaxSpeed = 3000.f;
    ProjectileMovement->bRotationFollowsVelocity = true;
    ProjectileMovement->bShouldBounce = true;

    // Die after 3 seconds by default
    InitialLifeSpan = 3.0f;
}


void AFPSProjectile::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor,
     UPrimitiveComponent* OtherComp, FVector NormalImpulse, const FHitResult
    & Hit)
{
    // Only add impulse and destroy projectile if we hit a physics
    if ((OtherActor != NULL) && (OtherActor != this) && (OtherComp != NULL)
    && OtherComp->IsSimulatingPhysics())
    {
        OtherComp->AddImpulseAtLocation(GetVelocity() * 100.0f,
    GetActorLocation());
        FVector Scale = OtherComp->GetComponentScale();
        Scale *= 0.8f;
        if (Scale.GetMin() < 0.5f)
            OtherActor->Destroy();
        else
            OtherComp->SetWorldScale3D(Scale);
        UMaterialInstanceDynamic *MatInst = OtherComp->
    CreateAndSetMaterialInstanceDynamic(0);
        if (MatInst)
        {
            MatInst->SetVectorParameterValue("Color", FLinearColor::
    MakeRandomColor());
        }



        Destroy();
    }
}
```

13.7 Adding a simple objective

```
#HEADER
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "FPSObjectiveActor.generated.h"

class USphereComponent;

UCLASS()
class FPSGAME_API AFPSObjectiveActor : public AActor
{
   GENERATED_BODY()

public:
   // Sets default values for this actor's properties
   AFPSObjectiveActor();

protected:

   UPROPERTY(VisibleAnywhere, Category = "Component")
   UStaticMeshComponent *MeshComp;

   UPROPERTY(VisibleAnywhere, Category = "Component")
   USphereComponent *SphereComp;

   UPROPERTY(EditDefaultsOnly, Category= "Effects")
   UParticleSystem *PickupFX;

   // Called when the game starts or when spawned
   virtual void BeginPlay() override;

   void PlayEffects();


public:


   virtual void NotifyActorBeginOverlap(AActor *OtherActor) override;

};



#SOURCE CPP FILE
```

```cpp
#include "FPSObjectiveActor.h"
#include "Components/SphereComponent.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"
#include "FPSCharacter.h"

// Sets default values
AFPSObjectiveActor::AFPSObjectiveActor()
{

   MeshComp = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshComp"))
   ;
   MeshComp->SetCollisionEnabled(ECollisionEnabled::NoCollision);
   RootComponent = MeshComp;

   SphereComp = CreateDefaultSubobject<USphereComponent>(TEXT("SphereComp"))
   ;
   SphereComp->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
   SphereComp->SetCollisionResponseToAllChannels(ECR_Ignore);
   SphereComp->SetCollisionResponseToChannel(ECC_Pawn, ECR_Overlap);
   SphereComp->SetupAttachment(MeshComp);
}


// Called when the game starts or when spawned
void AFPSObjectiveActor::BeginPlay()
{
   Super::BeginPlay();

   PlayEffects();
}


void AFPSObjectiveActor::PlayEffects()
{
   UGameplayStatics::SpawnEmitterAtLocation(this, PickupFX, GetActorLocation
   ());
}




void AFPSObjectiveActor::NotifyActorBeginOverlap(AActor *OtherActor)
{
   Super::NotifyActorBeginOverlap(OtherActor);
   PlayEffects();
```

```
    AFPSCharacter *MyCharacter = Cast<AFPSCharacter>(OtherActor);
    if (MyCharacter)
    {
        MyCharacter->bIsCarryingObjective = true;
        Destroy();
    }
}
```

Figure 13.4: Objective Actor

13.8 Adding an extraction zone for the objective

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "ExtractionZone.generated.h"


class UBoxComponent;
class UDecalComponent;
```

```
UCLASS()
class FPSGAME_API AExtractionZone : public AActor
{
   GENERATED_BODY()

public:
   // Sets default values for this actor's properties
   AExtractionZone();

protected:


   UPROPERTY(VisibleAnywhere, Category = "Components")
   UBoxComponent *OverlapComponent;

   UPROPERTY(VisibleAnywhere, Category = "Components")
   UDecalComponent *DecalComp;

   UFUNCTION()
   void HandleOverlap(UPrimitiveComponent *OverlappedComponent, AActor *
   OtherActor, UPrimitiveComponent *OtherComp, int32 OtherBodyIndex, bool
   bFromSweep, const FHitResult &SweepResult);

   UPROPERTY(EditDefaultsOnly, Category = "Sounds")
   USoundBase *ObjectiveMissingSound;
};

#SOURCE CPP FILE

#include "ExtractionZone.h"
#include "Components/BoxComponent.h"
#include "Components/DecalComponent.h"
#include "FPSCharacter.h"
#include "FPSGameMode.h"
#include "Kismet/GameplayStatics.h"



// Sets default values
AExtractionZone::AExtractionZone()
{

   OverlapComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("
   OverlapComponent"));
   OverlapComponent->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
   OverlapComponent->SetCollisionResponseToAllChannels(ECR_Ignore);
   OverlapComponent->SetCollisionResponseToChannel(ECC_Pawn,ECR_Overlap);
```

```
    OverlapComponent->SetBoxExtent(FVector(200.0f));
    OverlapComponent->SetHiddenInGame(false);
    RootComponent = OverlapComponent;

    OverlapComponent->OnComponentBeginOverlap.AddDynamic(this, &
    AExtractionZone::HandleOverlap);

    DecalComp = CreateDefaultSubobject<UDecalComponent>(TEXT("DecalComp"));
    DecalComp->DecalSize = FVector(200.0f, 200.0f, 200.0f);

    DecalComp->SetupAttachment(RootComponent);

}



void AExtractionZone::HandleOverlap(UPrimitiveComponent *OverlappedComponent
    , AActor *OtherActor, UPrimitiveComponent *OtherComp, int32
    OtherBodyIndex, bool bFromSweep, const FHitResult &SweepResult)
{

    AFPSCharacter *MyPawn = Cast<AFPSCharacter>(OtherActor);
    if (MyPawn == NULL) //&&
        return;
    if((MyPawn->bIsCarryingObjective))
    {
        UE_LOG(LogTemp, Log, TEXT("Overlapped␣with␣extraction␣zone␣!"));
        AFPSGameMode *GM = Cast<AFPSGameMode>(GetWorld()->GetAuthGameMode());
        if (GM)
            GM->MissionComplete(MyPawn);
    }
    else
    {
        UGameplayStatics::PlaySound2D(this, ObjectiveMissingSound);
    }
}
```

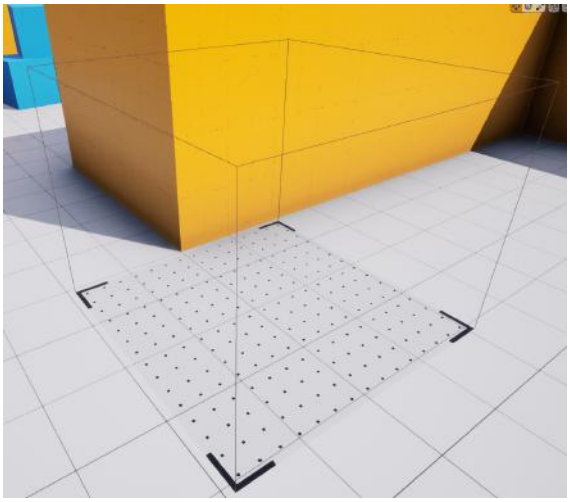


Figure 13.5: Extraction Zone

Figure 13.6: Interaction with the Zone

Chapter 14

Environment Query System

14.1 Artificial Intelligence

The Environment Query System or simply EQS is a feature that represents the working of Artificial Intelligence inside Unreal Engine 4. An EQS query follows the path of a Behavior Tree which allows it to reach suitable outcomes based on the decisions the BT takes dynamically to simulate the behavior of a sentient organism.

Figure 14.1: Player Pawn

14.1.1 The Behavior Tree

A Behavior Tree follows pre-order traversal for every decision. Each node itself is a macro. If it satisfies a certain condition, then it is executed, else the next node is traversed for the same.

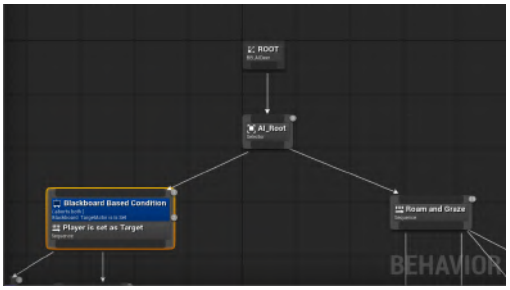


Figure 14.2: AI Deer Behavior Tree

AI's interaction with the player

When the ThirdPersonCharacter or the Deer comes within a setup range of the AI, the are scripted to follow certain rules to interact with the player. The two instances of AI Deer here are the ones that start following the player as long as the player is in visible range of the AI. To stop them from following the player, the player can either hide behind a rock to break the line of sight or outrun the AI to go outside their range of interaction.

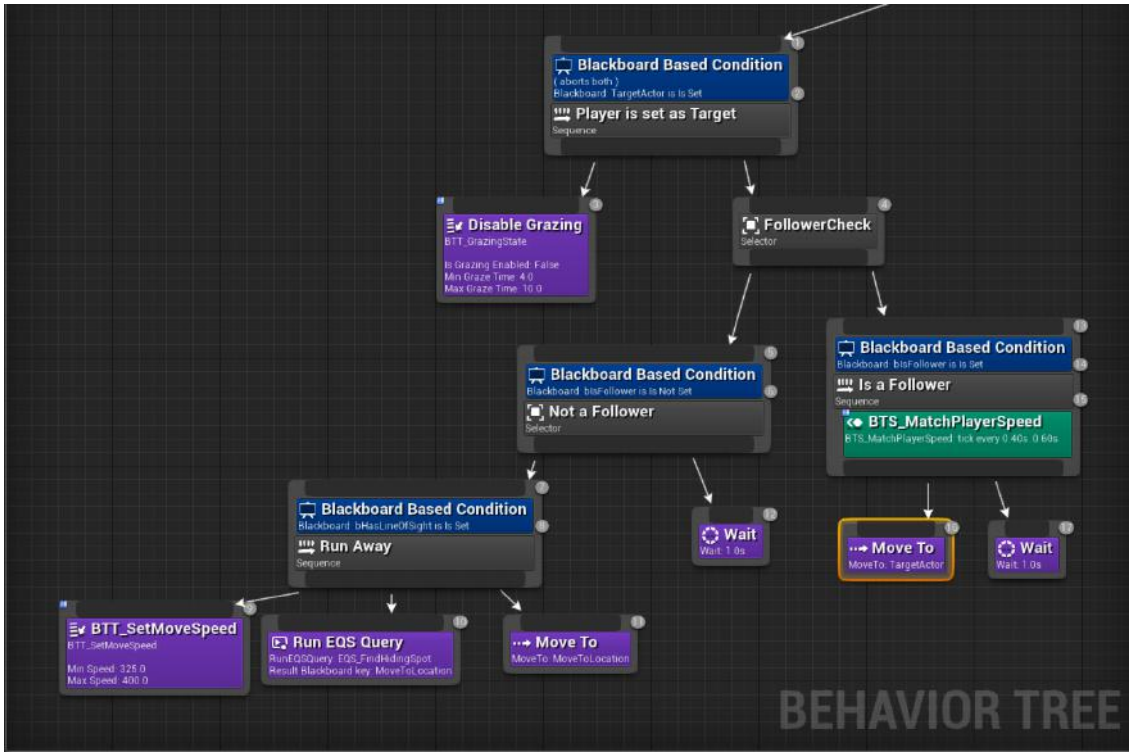


Figure 14.3: AI LeftSubTree

The other instance being a timid set of deers; once the player comes near them, they start running away from the player-pawn until they are at a certain distance from it.

Randomized AI Behavior

When the AI pawns are outside the range of the player pawn, they are scripted to perform random acts such as grazing(with animations), running in an arbitrary direction and so on. These functions are scripted in separate macros at the leaf nodes of the Behavior Tree.

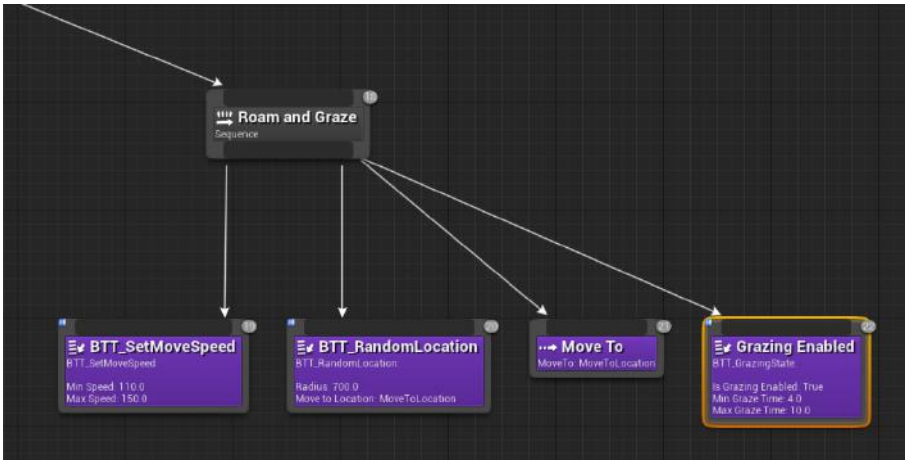


Figure 14.4: AI RightSubTree

Chapter 15

Description of Use Case Diagram

The programmer launches Microsoft Visual Studio IDE with the current Visual C++ project that links to the Unreal Engine. Upon successful compilation, it shall launch the UE4 editor with the game ready to be played in the editor only. It can also be opened directly from the Epic Games launcher, for in case the C++ build fails, it shall run the last successful build.

On hitting the play button, it opens the in-game viewport for the programmer; when the game is launched, it creates an exe file to be played oustide the editor, and packaging the game allows everyone to play it.

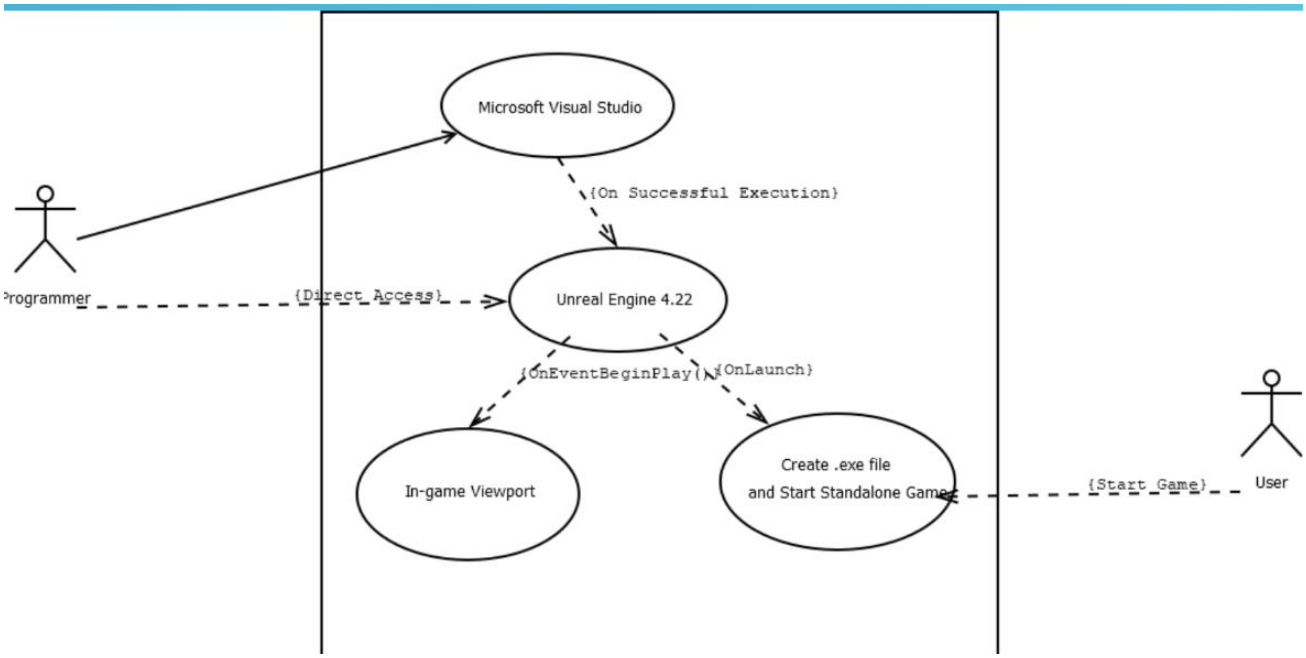


Figure 15.1: Use Case Diagram

Chapter 16

Blueprints and Visual C++

The **Blueprints Visual Scripting** system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. As with many common scripting languages, it is used to define object-oriented (OO) classes or objects in the engine. As you use UE4, you'll often find that objects defined using Blueprint are colloquially referred to as just "Blueprints."

16.1 Class Diagram

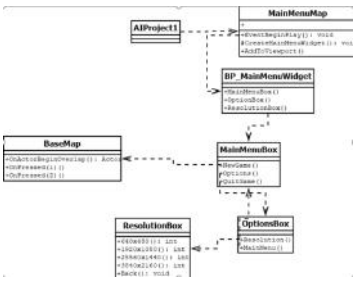


Figure 16.1: Class Diagram

16.2 Functions

Function Calls are actions that can be formed within Blueprints that correspond to functions belonging to a targeted Actor or Object. In the case of Level Blueprints, the associated Actor in many cases is the Level Blueprint itself. Function Calls are displayed as boxes with titles that show the name of the function. Different types of function calls have different colour titles.

There are basically two possible function types used in blueprints, i.e. the getter function and the setter functions. Getter functions are used to get parameters or inputs from the LOC, or the user and the setter functions set certain values to parameters.

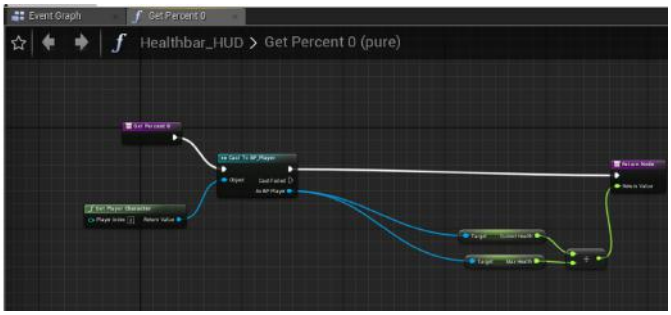


Figure 16.2: Blueprint Function

16.3 Key-bindings

Key-Bindings are an integral part of developing a game as they reflect on how a player interacts with the UI in the game. For every bound key pressed, a function in the blueprint or C++ is triggered to perform the required action.

Common commands that are used universally in every game such as 'Jump', 'Crouch',

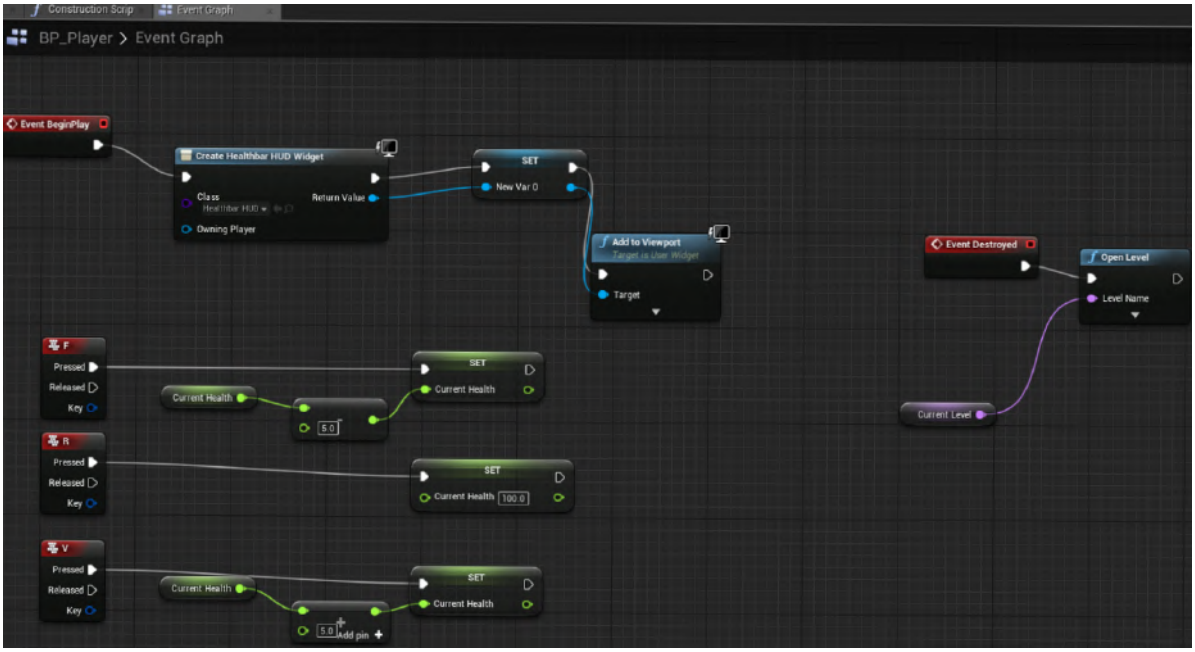


Figure 16.3: Key Bindings

'Sprint' etc. have pre-defined macros assigned to special keys on the keyboard. One can configure keyboard shortcuts in the Unreal Editor to adjust to one's workflow or preferences by using key-binds to perform standard actions.

The custom keybinds can be set in the Editor Preferences tab of the editor.

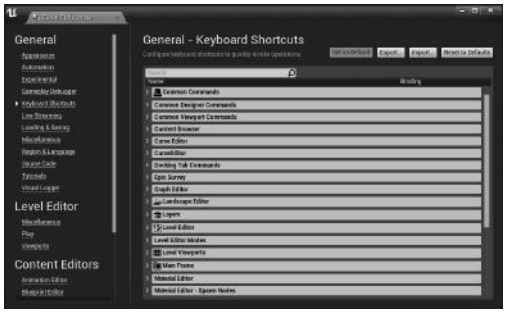


Figure 16.4: Key Bindings

16.4 Materials

A Material is an asset that can be applied to a mesh to control the visual look of the scene. At a high level, it is probably easiest to think of a Material as the "paint" that is applied to an object. But even that can be a little misleading, since a Material literally defines the type of surface from which your object appears to be made. You can define its color, how shiny it is, whether you can see through the object, and much more.

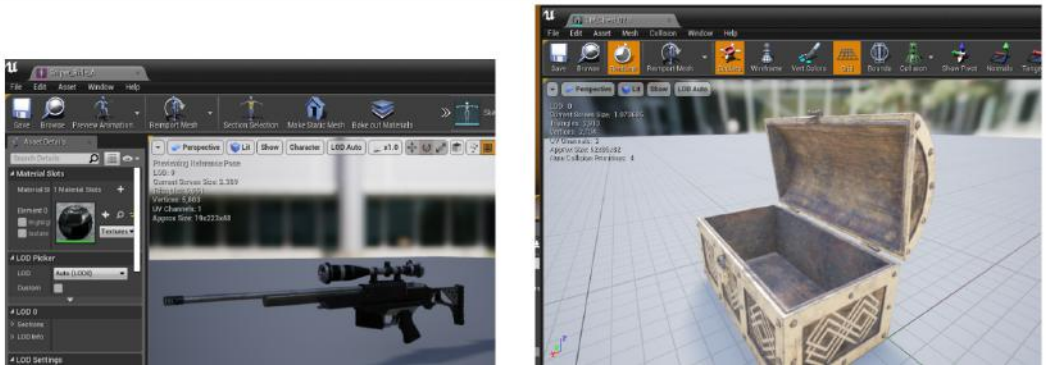


Figure 16.5: Blueprint Materials

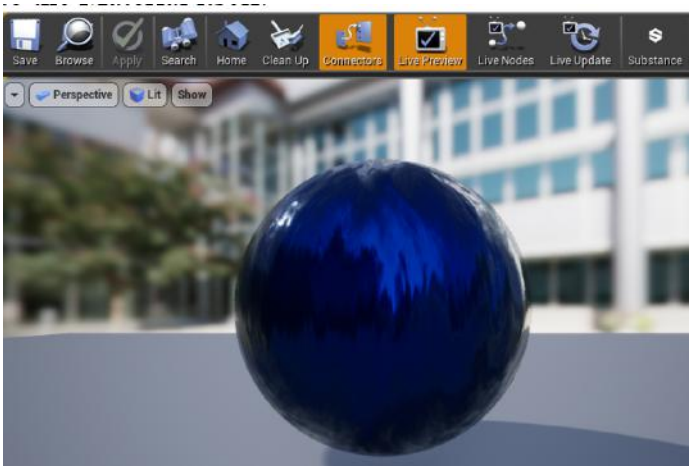
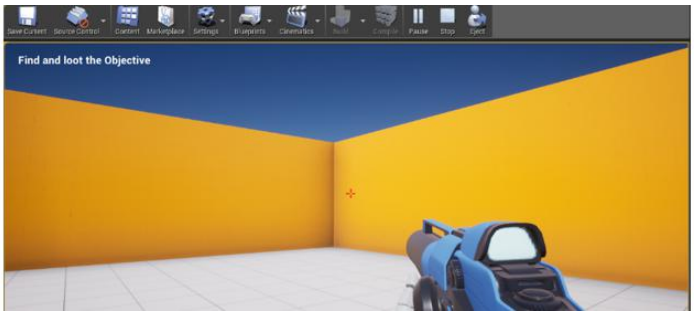


Figure 16.6: Blueprint Water material

16.5 UI and Notifications

Notifications about events can be generated in console log or directly in-game by creating an HUD



16.6 C++ Header and Source Files

Each class and its constituent function are declared in a header(.h) file and defined in the .cpp file of the same name. If declared in some other .cpp file, the header file is included in the beginning. There is no main function for the flow of execution of the code, but rather the "UnrealBuildTool", which uses C for building the project using a string of Game Modules.

Figure 16.7: C++ Classes in the UE Editor

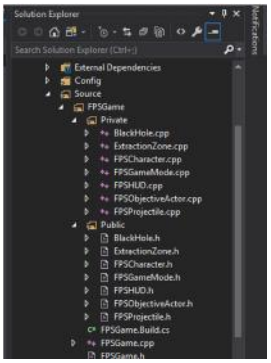


Figure 16.8: C++ Header and Source files

Chapter 17

Testing

In this stage, all content used in the previous stage was used for testing purposes. Once the game is packaged and deployed on a dedicated server, it is Beta tested on several platforms in multi-player mode. Here the various bugs and issues such as Spawn-glitches, recoil-control auras, projectile range and accuracy and other pre-rendered components are fixed. The point lights and assets created in the production phase were put to proper use and tweaked if necessary.

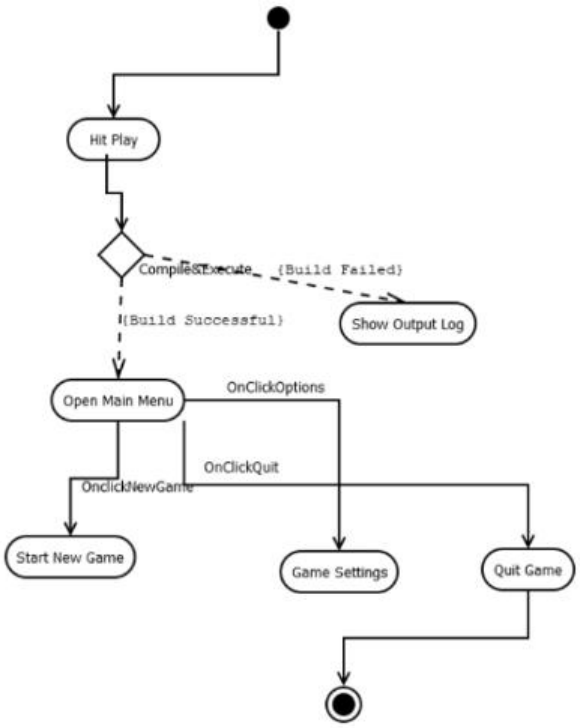


Figure 17.1: Activity Diagram and working

Chapter 18

Post Production

In this final stage, after gathering enough feedback, the packaged game is compressed in size by removing unnecessary assets, packages and un-used code blocks. It is first checked on all platforms that include Nvidia GeForce RTX or GeForce GTX builds as well AMD Vega and Radeon builds to check for performance. The game is then re-rendered to be aesthetically pleasing for all users. Finally, the game is deployed on Epic Games Store as a product..

Figure 18.1: Rendered effects on two separate platforms (Battlefield V)

Chapter 19

Gantt Chart

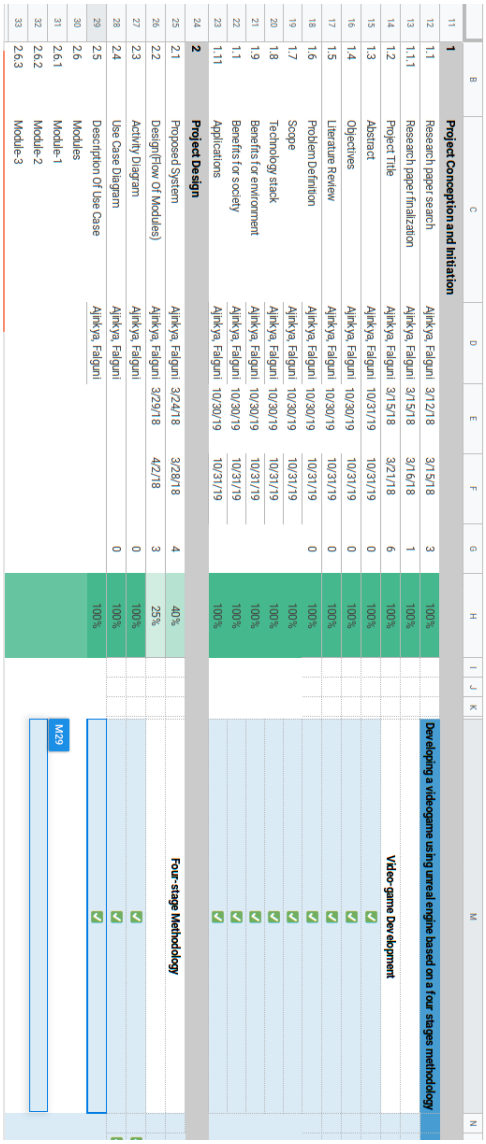


Figure 19.1: Gantt Chart

Phase-2

32	**3**	**Project Implementation**				
33	3.1	Characters (Male or Female)	Ajinkya, Falguni	2	100%	
34	3.2	C++ class	Ajinkya, Falguni	1	100%	**Week 2** ✅
35	3.3	The Character Class	Ajinkya, Falguni	1	100%	**Week 1** ✅
36	3.4	The Behavior Tree	Ajinkya, Falguni	1	100%	**Week 9** ✅
37	3.5	Functions	Ajinkya, Falguni	1	100%	**Week 12** ✅
38	3.6	Key-bindings	Ajinkya, Falguni	1	100%	**Week 5** ✅
39	3.7	Weather Effects	Ajinkya, Falguni	1	100%	**Week 8** ✅
40	3.8	Cinematics	Ajinkya, Falguni	1	50%	**Week 13**
41	3.9	Animations	Ajinkya, Falguni	1	50%	**Week 13**
42	3.10	Healthbar	Ajinkya, Falguni	1	75%	**Week 14** ✅
43	3.11	Composite project	Ajinkya, Falguni	1	100%	**Week14** ✅
44	**4**	**Testing**				
45	4.1	Design of Test Cases	Ajinkya, Falguni	1	75%	
46	4.2	Testing	Ajinkya, Falguni	1	75%	
47	**5**	**Post Production**				
48	5.1	Ray Tracing	Ajinkya, Falguni	1	50%	
49	5.2	Deployment	Ajinkya, Falguni	0	0%	
50	5.3	Report Preparation		1	100%	**Week14** ✅
51						

Gantt Chart

Figure 19.2: Gantt Chart

Chapter 20

Conclusions and Future Scope

The methodologies/phases of game design differ significantly depending on the skill and number of people working on it. These phases differ in some industries in some context, mainly due to the availability of skilled programmers, designers, artists, sound engineers, etc.

The availability of easy-to-access resources such as Unreal Engine and Visual Studio prove that, given the right time and right people, and using the right methodologies for the project, one could work wonders given the right time.

Hence, we can say that it is possible to create simple games without too much expense and resources using the right tools (in our case, Unreal Engine).

Bibliography

[1] Carlos Mauriccio Torres-Ferreyros ; Matthew Alexander Festini-Wendorff ; Pedro Nelson Shiguihara-Juárez ,"Developing a videogame using unreal engine based on a four stages methodology",Publisher: IEEE. https://ieeexplore.ieee.org/document/7836249

[2] Unreal Engine Documentation https://www.unrealengine.com/en-US/blog/a-new-look-for-the-unreal-engine-documentation

[3] Unreal Engine Marketplace https://www.unrealengine.com/marketplace/en-US/store

Acknowledgement

I have great pleasure in presenting the report on **Video-Game Development in Unreal Engine.** I take this opportunity to express my sincere thanks towards my guide **Prof. Sachin Malave** Department of Computer Engineering, APSIT thane for providing the technical guidelines and suggestions regarding line of work. I would like to express my gratitude towards his constant encouragement, support and guidance through the development of project.

I thank **Prof. Sachin Malave** Head of Department,Computer Engineering, APSIT for his encouragement during progress meeting and providing guidelines to write this report.

I thank **Prof. Amol Kalugade** BE project co-ordinator, Department of Computer Engineering, APSIT for being encouraging throughout the course and for guidance.

I also thank the entire staff of APSIT for their invaluable help rendered during the course of this work. I wish to express my deep gratitude towards all my colleagues of APSIT for their encouragement.

Student Name1: Falguni Mukesh Tailor
Student ID1: 16102031