**AR2**

Side-Scrolling Game Development with Unreal Engine 3

by

GURUVANICH, Aekasitt

MUALLEM, Melkar

WIBAWA, Genio Bernardo

Advisor:

ARYA, Sunil PhD.

Submitted in partial fulfillment

of the requirements for COMP 4982

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2013-2014

Date of submission: February 18, 2014

Table of Contents

[I. INTRODUCTION 5](#_Toc380443587)

[OVERVIEW 5](#_Toc380443588)

[OBJECTIVE 5](#_Toc380443589)

[ABOUT 6](#_Toc380443590)

[II. DESIGN 6](#_Toc380443591)

[GRAPHICS 6](#_Toc380443592)

[Maps 7](#_Toc380443593)

[Characters 7](#_Toc380443594)

[GAMEPLAY 8](#_Toc380443595)

[Storyline 8](#_Toc380443596)

[AI/Abilities 8](#_Toc380443597)

[AI/Mechanics 8](#_Toc380443598)

[AI/Enemy 9](#_Toc380443599)

[Music/Sound effects: 9](#_Toc380443600)

[HUD 9](#_Toc380443601)

[The Health Bar 10](#_Toc380443602)

[Crosshair 10](#_Toc380443603)

[Souluminescence Bar 10](#_Toc380443604)

[Mode Indicator 10](#_Toc380443605)

[Possession Countdown 10](#_Toc380443606)

[Enemy’s Alertness 10](#_Toc380443607)

[III. IMPLEMENTATION 11](#_Toc380443608)

[GRAPHICS 11](#_Toc380443609)

[GAMEPLAY 12](#_Toc380443610)

[AI/Enemy 12](#_Toc380443611)

[IV. CHALLENGES 14](#_Toc380443612)

[Graphics 14](#_Toc380443613)

[AI/GAMEPLAY 14](#_Toc380443614)

[HUD 14](#_Toc380443615)

[V. TESTING 15](#_Toc380443616)

[Remote Control Testing 15](#_Toc380443617)

[Alpha and Beta testing 16](#_Toc380443618)

[VI. WHAT WE WILL DO NEXT 16](#_Toc380443619)

# INTRODUCTION

## OVERVIEW

A side-scrolling game is a video game genre which involves guiding player-controlled avatar using side-view camera perspective. Typical side-scrolling games require the player to guide the avatar from the left side of the screen to the right side of the screen while having to jump over obstacles and hop to different platforms. The trend of side-scrolling games began back in the 1980s [1] when technology just reached sufficient maturity to support the animation of other graphics apart from the player’s graphics. However, as computing power increased exponentially in the last three decades, gaming community started to view side-scrolling game as ‘ancient’ gaming genre compared to other more sophisticated modern gaming genres such as First Person Shooter (FPS) or Role Playing Game (RPG).

Despite the decline in popularity, side-scrolling games were still able to penetrate the market [2]. This is mainly caused by their simple design and originality in addition to their portability and the ease to develop mobile side-scrolling gaming applications. On the other hand most First Person Shooter games are too complicated and too repetitive for gamers to enjoy which causes limited in game experience to the players. With this in mind, we are creating a simple yet arousing side-scrolling game with focus on creative game design (both graphics and gameplay) as well as original storyline.

## OBJECTIVE

Side-Scrolling games are unique due to their portability between different devices. They represent the most basic of digital games. For this reason though, side-scrolling games need to be superior in gameplay and story to capture the players.   
  
To achieve such feat, we broke down our development objectives of this game to the following; trying to assure an interesting gameplay as well as sophisticated, efficient design.

1. To create a capturinggame-world by creating a unique environment based on the horror driven storyline of our protagonist using gloomy graphics and sound effects.
2. To provide a dynamic and interesting gameplay to the users by designing challenging maps and puzzles in addition to a variety of different objectives to be accomplished in each map.
3. To develop a map with independent gameplay system which can be later extended and applied to other maps to assure the ability of updating and extending the game.

The player will undergo an adventure to solve the mind twisting plot of the game by finding the missing memory shards of our protagonist. In our fascinating game-world, the user will slowly pick up on the story behind the game events.

In order to guarantee the interest of players in our game, we will also incorporate different puzzles and challenges in the maps to assure diversity of game experiences. Maps will have an increasing difficulty to keep the player engaged and absorbed in the game. The game will require users to adopt different strategies to solve the various obstacles, which we believe is key element in gameplay.

To further improve our gameplay as well as game dynamics. We are developing map-independent game rules and gameplay options to help us sustain a continuous development process as well as an ease of exporting the game into different platforms.

The biggest difficulty we faced was adapting and learning the new tools and programs used in game development. Nevertheless, we know our main objective is developing a capturing and stimulating game to all players, and thus we pour our most intensive focus on developing distinctive gameplay and levels.

## ABOUT

To accomplish the objectives listed above, we have decided to use a cross-platform graphics engine called **Unreal Engine 3** using the development environment called **Unreal Development Kit (UDK)**. There are three main reasons why we decided to work on Unreal Engine 3. Firstly, the use of Unreal Engine 3 for noncommercial purposes is completely free of charge, as compared to other engines which require commercial licensing. Secondly, Unreal Engine comes with a set of built-in tools which have specific and important functions in developing games. For example, the **Unreal Kismet** tools which is usable for in-game event programming is available built-in Unreal Development Kit. Thirdly, the community of Unreal Engine programmers offers online support for new Unreal Engine language (which is known as **UnrealScript**) programmers [3].

Our project is broken down into three main overlapping parts which require us to work together for the end product. First of the three is creating the environment and physical objects in the maps and levels. The tools which will be used are Unreal Editor, to visually draw the 3-Dimensional environments and place them in the level. Second, we develop our own game logic and game mechanics using UnrealScript. Some of the important gameplay factors that will be built are camera properties, player controller, player ability, pawn interactions, enemy A.I, and game physics. Lastly, we need to present the game to the player with our User Interface (UI) system. The most important part of the User Interface system is the Heads-Up Display (HUD) which shows game state and the character’s information as well as showing the cursor to the player.

The storyline of this game is also one of the main goals of our project. To support the idea that the gaming industry is a part of the art industry; as it makes interactive media that can be both captioning to the audiences and still holds true to the story telling.

# DESIGN

## GRAPHICS

### Maps

In our game, 4 maps have been designed for different levels of gameplay to allow the player to experiment different situations using the mechanics and rules we have invented for them. These four maps are the following:

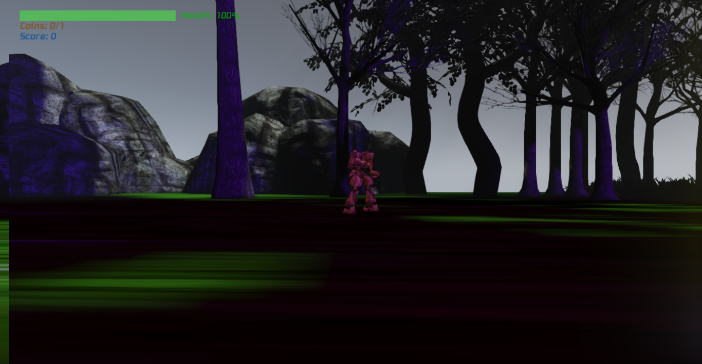
* 1. Pipework: This is a map that most resembles a maze where the player gets to explore the world and avoid detection by deciding the best possible path to take.
  2. Museum: Designed with a clear goal and is therefore more linear as how to approach it. We control the difficulty of this map by adding more Bots in this level with more alarm points.
  3. Grasslands: Open map with a forest backdrop added on with lightning and rain effects for the horror experience.
  4. Traps: This is a final map in the game which we would add a countdown timer in order for the player to escape before the time limit. It is filled with teleportation points, Bots and traps.

Figure 1: Grasslands Map when a lightning hits.

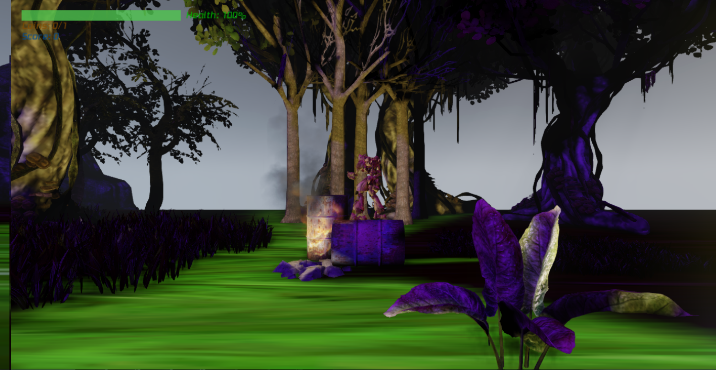


Figure 2: Lighting made from a Light Source placed next to a flame.



Figure 3: Map design we call a Museum.

### Characters

The storyline for this game dictates the design for characters in this game. We have 2 work-in-progress designs for the characters for the basic beta testing, which are our main character controlled by the player, and basic enemy characters controlled by AI. Our main character is designed as a ghostly character with passive abilities to fly and glow which are portrayed out in the graphics. Its active ability of possession will also be portrayed using UDK’s particle system as well. Different Pawn subclasses will be created for each type of enemy. We find this much easier to manage than having one standard Pawn with different graphics. The basic enemy characters are designed to look like museum guards.

## GAMEPLAY

### Storyline

The story begins with the protagonist waking up in a dark and gloomy forest with no recollection of how he wound up in the forest, or who he was before. As he scoured the dark forest for clues, he slowly realizes that he is no longer in his corporeal form and inhabiting ghostly form.

### AI/Abilities

The protagonist comes with a set of abilities which the player can use depending on the approach that the player chooses. Players who choose to complete the game using stealth approach would tend to use the ability set to find the best path with less enemy presence, to minimalize combats. On the other hand, players who choose to be confrontational will use the ability for acquiring extra firepower to excel in combats. In general, we are implementing three basic abilities for the player to use accordingly. The list is as follows:

1. **Possession**. Using his ethereal form, the player looks inside the enemy’s soul while bypassing his central nerve system with sheer terror which allows the player to take control of the enemy unit for a short amount of time until the target totally loses consciousness.
2. **Ghostly glide**. The player’s ethereal form renders him almost massless and can neglect the force of gravity, at the cost of his spiritual energy.
3. **Souluminescence.** The player is able to actively convert his spiritual energy into light energy and radiates the environment around him. The amount of energy drained is proportional to the intensity of light emitted.

In addition to those abilities, there are a list of other abilities that are still under consideration. The initial reaction that we get from subsequent alpha/beta tests will determine if additional abilities are necessary to increase the variability of the game.

### AI/Mechanics

Basic side-scrolling game mechanics apply to this game. The main character and its interactive environment are plotted in a single ‘line’ or axis and the player’s movement is limited to that particular axis only. Instead of allowing only one line of action, we are implementing different heights that the player can choose to occupy like different levels in a building. This design will allow players to freely choose which height (if available) to move on, which can be used to evade (or confront) enemies.

### AI/Enemy

As mentioned in the proposal, we have set specific goals and standards for the enemy’s behavior. Enemy pawn has to be good, entertaining, beatable, realistic, and possess decent amount of challenges for the player. In order to reach our design goals, we have built our enemy pawn with following states/behavior:

1. **Patrolling**. Enemy pawn normally patrols between two nodes in the map indefinitely
2. **Alertness**. The enemy will be alerted of the presence of the player if the player is inside the enemy’s field of vision for an extended amount of time.
3. **Hostile – Attack**. Enemy will start shooting at player after being alerted.
4. **Hostile – Raising the alarm**. Enemy will run away from the player and try to raise nearby alarm after being alerted.
5. **Scared**. Enemy is terrified by the player’s presence, if the player comes too close to the enemy without the enemy realizing it.
6. **ChasePlayer**. Enemy started chasing the player if the player is beyond the enemy’s firing range.
7. **HearSound**. When a player makes noises (by moving or bumping into objects) the enemy will become suspicious and try to trace the source of the noise.

### Music/Sound effects:

To enhance the game experience and induce a feeling of horror while playing, we will be integrating ambient music and sound effects. We are aiming to have several theme songs playing around the maps to give appropriate feel of the atmosphere of each map. We will also use a combination of music and sound effects to put the user in specific moods. For example, the music will tend to be more upbeat when the character is detected by guards, compared to a more mellow but melancholic music when moving stealthy around the map.

## HUD

During gameplay, there are few important statuses a user need to be constantly aware of; such as the character’s health value, power meter indicator, enemy alertness and more. The most convenient of displaying such information is through the use of Heads-Up Display (HUD). A HUD is simply defined as a “visual overlay used to keep track of useful statistics.”[1] A HUD has two components, the first is the visual representation displayed as an overlay on the map, and the second is the functionality attached to the HUD; that is the statistics it represents, such as a bar representing the value of the character health.

Since the game genre falls under horror and darkness, the game graphics, including HUDs and menus, should properly represent that. Our Head-up Displays and menus will reflect the gloomy theme through the use of dark colors ominous visuals. The functionality of the HUDs will now be further explained base. We have 6 HUD elements in total, and one menu. The elements are as follows:

1. Health bar (user’s current health)
2. Crosshair (indicates where the use is aiming)
3. Souluminescence Bar (indicates how much luminescence energy is available)
4. Mode indicator (whether the user is in ghost or in possession mode)
5. Possession countdown (time left for possession; only active when in possession mode)
6. Enemy Alertness (how alert are the enemy pawns)

### The Health Bar

The Health Bar is an important indication that displays a user’s health; once health is over the user usually loses the game or has to repeat, as out of health means unable to play. With that said, a health bar needs two show two important numbers, the maximum health, and the current health. In modern games health bars are displayed in the elegant way of a visual bar without the use of numbers, as seen in the figure below. Our health bar will be matching to our dark theme, and will work like a normal health bar; that is, decreasing when damage is taken.

### Crosshair

The crosshair is a small HUD element the represents the target of the users aim. A crosshair in games represent the aiming scope used in weapons. This element will follow the eyesight of the character, determined by the rotation and angle of its head.

### Souluminescence Bar

This bar indicates the amount of energy the user has which can be used to activate the Souluminescence ability. The bar will decrease as energy is consumed, and will be refilled via collectables found around the map.

### Mode Indicator

The protagonist has one power, possession, a HUD needs to indicate when the user is entering, using, or leaving the possession mode. This can be done through the use of words or other sorts of visualization. The indicator changes to the appropriate display according to the mode the user is in.

### Possession Countdown

A user needs to know the time left in possession mode, a countdown indicator is necessary to relay to the user the amount of time left. Visually, this can also be displayed in a bar or similar representation, actual number countdown is not as appealing. The countdown indicator only appears when possession mode is activated, and disappears after the protagonist reverts back to his original form.

### Enemy’s Alertness

The enemy pawn is a non-playable character (NPC) which we are putting in the world throughout the game. These pawns will have different functions and uses throughout the game. This HUD needs to indicate whether the pawns in the game are generally aware of the actions by the user, and how alert they are to indicate how stealthy the player is.

# IMPLEMENTATION

## GRAPHICS

Graphics in our game including maps, static meshes, texture for 3-dimensional models and sound effects can be edited inside the Unreal Engine Platform. Below is a figure showing the workspace in Unreal Development Kit. Map creation method using the 3-dimensional brushes and other contents provided by the Content Browser.

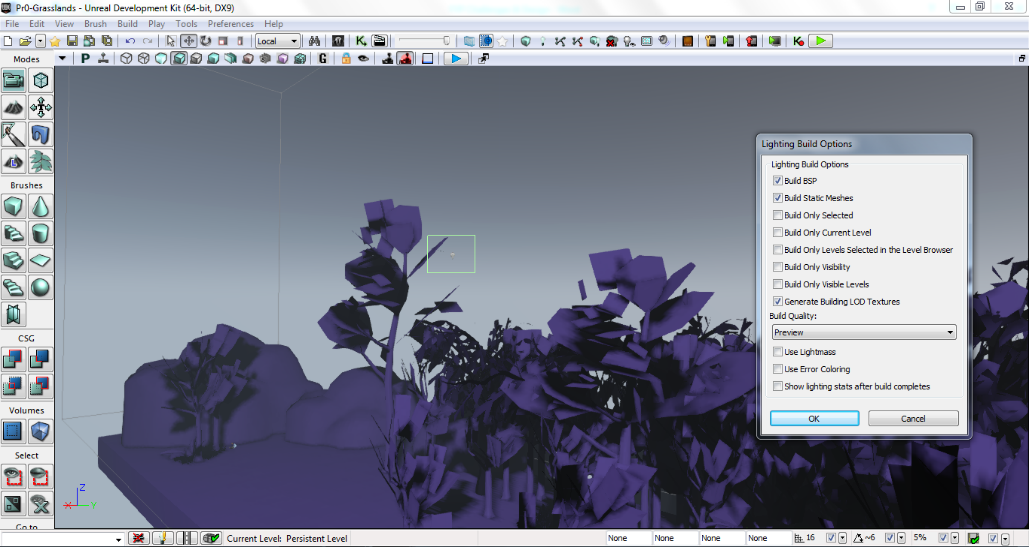


Figure 4: shows the tools and brushes on the left-hand side of the workspace and a window for Lighting Build Options which make the Map look real.

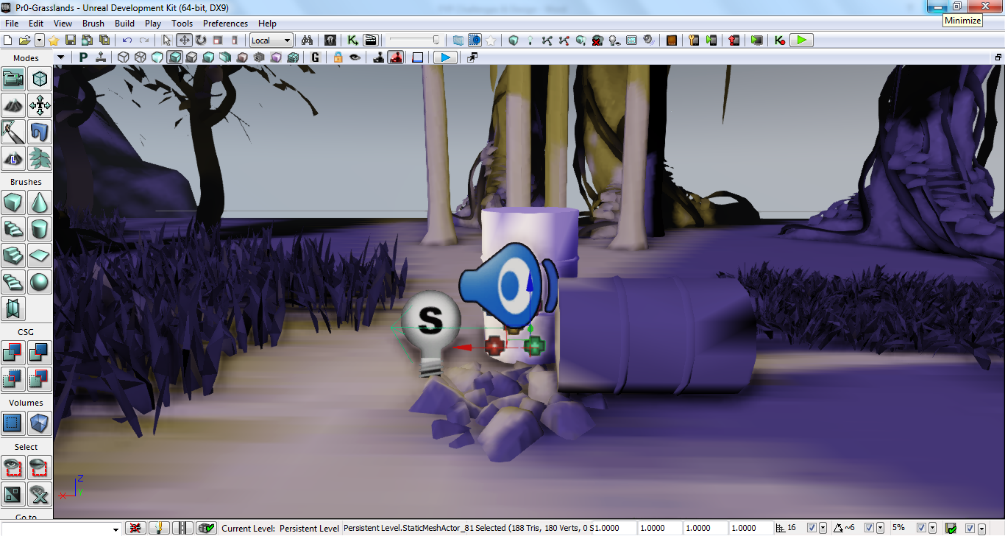
Aside from 3-dimensional models, other objects like Particle System, in this case a flame, Ambient Sound (crackling sounds) and a Light Source is added to the map.

Figure 5: A light, ambient sound, and particle system placed in the map.

The finished product of adding these elements can be seen when the player starts the game.

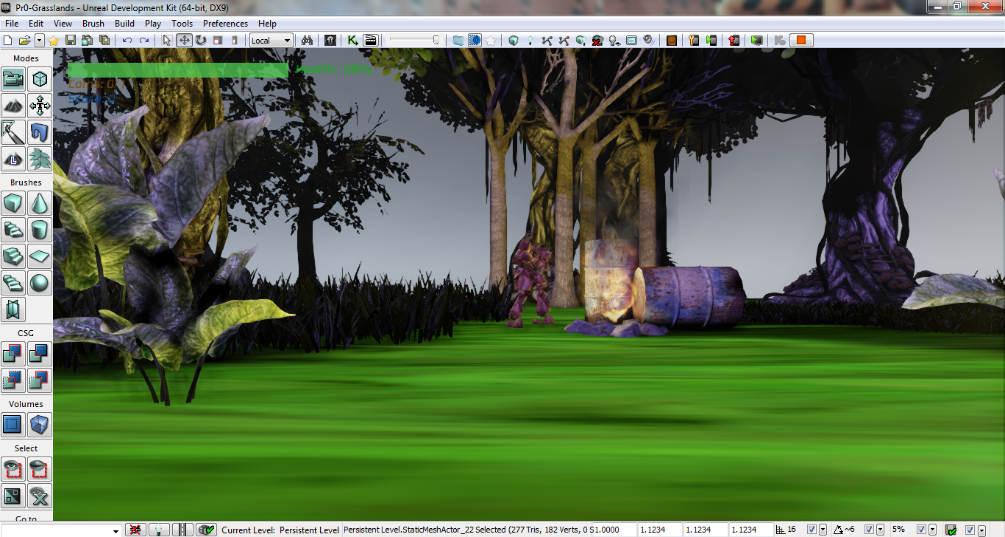


Figure 6: The particle effect and lighting effect can be observed when player is near.

## GAMEPLAY

### AI/Enemy

Most of the enemy functionality will be done using UnrealScript, a scripting language specifically built to write gameplay events. We also coupled the usage of UnrealScript with Unreal Kismet, a tool to link multiple gameplay events together sequentially. Going into detail, a breakdown of the UnrealScript – Kismet interaction that we have in our game is as follows:

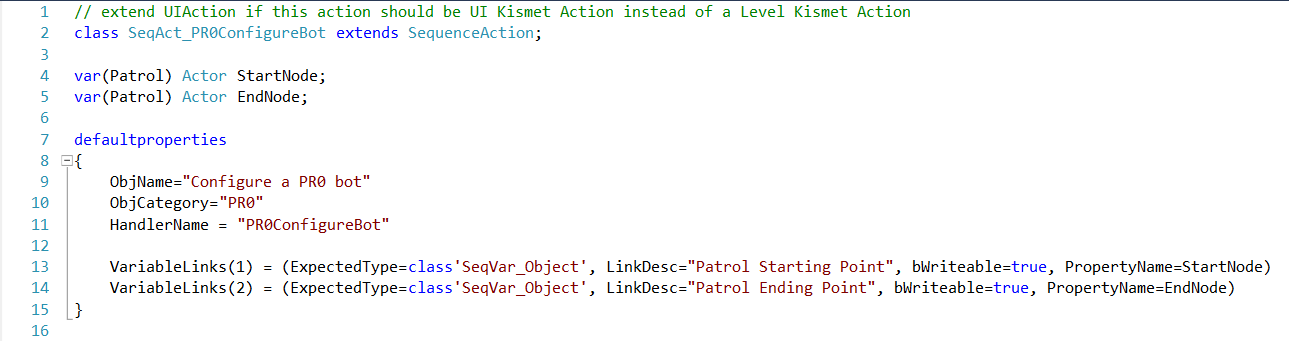
1. Create custom Unreal Kismet Sequence Action using UnrealScript.

Figure 7: Creating a custom Kismet Sequence Action in UnrealScript

1. Using Unreal Kismet editor, instantiate the newly created Sequence Action.

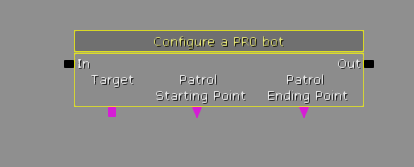


Figure 8: Place the Kismet Sequence Action in UDK Editor.

1. Logically connect the variables, inputs, and outputs of the Sequence Action with other predefined Actions and objects (such as objects placed in the world using the editor).
2. The example in figure below shows how each enemy unit (Bot) is instantiated with a patrol starting point and a patrol ending point (which are placed somewhere in the world and can be referenced by Kismet), just as we have specified in the UnrealScript file.

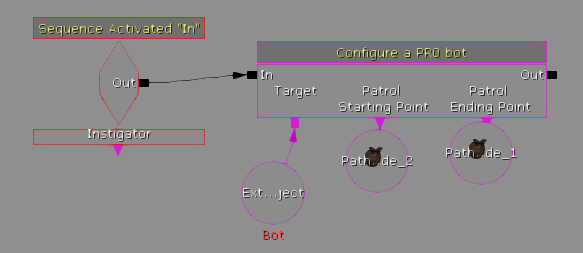


Figure 9: Logically connect multiple Kismet objects together. This figure is showing the 'Patrolling' game event.

UnrealScript and Kismet integration will be used to program enemy AI states and behaviors as described in the design section of this report paper. However, enemy AI custom animation will be done using another Unreal Engine built-ins called Unreal Matinee, which we will cover extensively in coming weeks.

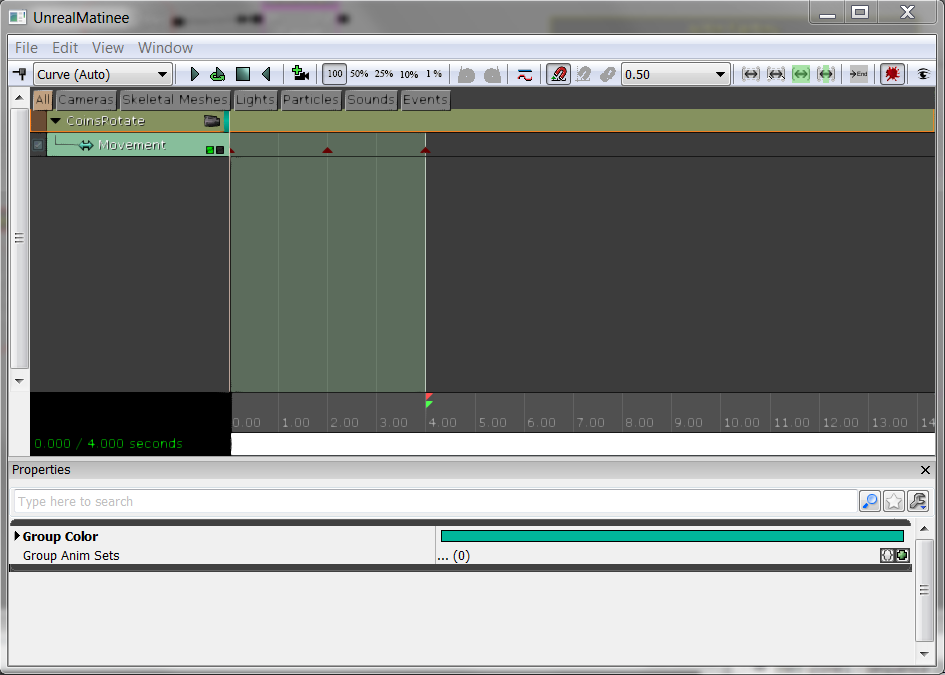


Figure 10 Using UnrealMatinee to animate a coin in the world.

# CHALLENGES

## Graphics

1. As we go along with this project according to original plan of making a blend between 3-dimensional world using 2-dimensional characters, we find very few documentations of such implementations. We found a reference to such method using Unreal Development Kit by creating a new subclass of *Pawn* class, a built-in class in Unreal used to control SkeletalMeshes and connect to either Player’s input or AI, called *SpritePawn* which carries a set of 2 dimensional image files, (in this case .PNG files)
2. Unreal Development Kit is known to be very versatile for it does not offer a lot of Editor tools built in to its own platform but lets you connect with other Editor programs e.g. Audacity for sound-editing, Adobe Photoshop Creative Suite to design the texture pasted on objects in the 3-dimensional platform, Maya to craft models used as Static Meshes, Skeletal Meshes and etc. This creates separate learning curves in using both the Unreal Editor and other object editors used in the game.
3. Keeping to the theme of horror, trials and errors are required to make the characters and the maps appeal to the audience.

## AI/GAMEPLAY

In terms of gameplay development, most of the work are done in UnrealScript. Creating player ability, creating custom enemy movement, and tweaking camera settings and effects are all done in UnrealScript. Based on that fact, the challenges that we faced when developing gameplay mainly revolves around UnrealScript familiarity and maximizing the UnrealScript potential. The first challenge that we faced is to understand UnrealScript’s API in a level sufficient for good program designs. However, UnrealScript seems to be lacking API documentation online. We managed to relieve this problem in a brute-force manner by going through the inheritance hierarchy every time we could not find a reference to a particular API. Secondly, we are challenged to make enemy Artificial Intelligence (AI) to possess decent challenge to players. Nonetheless, given the simple nature of side-scrolling games, it is challenging for us to come up with a simple, yet hard enough to be entertaining to players.

## HUD

HUD development relies mostly on developing .SWF (Flash movie) files, which the UDK Engine renders into the game as a heads-up component. Integrating Flash movies into UDK is automated by the engine and did not pose any challenge. However, to give HUDS their proper functionality, the unreal scripting language needs to directly control the action script (Flash’s programming language) to apply changes to the HUD dynamically. This is not very straightforward, and required long hours of work. On the other hand, Flash development was a new technology which had to be learned. This was the most difficult part of developing the HUD elements as it also meant having to learn other tools such as Adobe Photoshop.

# TESTING

The nature of the project being divided into three main interrelated parts required us to do multiple kinds of testing to ensure that each part works individually and collectively with other parts. There are two main testing methods that we employ to achieve this feat.

## Remote Control Testing

RemoteControl testing is a testing method native to Unreal Engine 3 and Unreal Development Kit (UDK). This can be achieved by adding –remotecontrol into the command line when running UDK. RemoteControl allows developers to change any in-game variables on the run, which saves some amount of time. Without using RemoteControl, developers would have to exit UDK, manually change the hardcoded values in our UnrealScript files before compiling it in order to make the change. RemoteControl is useful when we are trying to find a suitable value of a variable with no other way to do it except trial and error. For example, finding out the Rotational value of the any pawn and making it facing sideways would be easier when using RemoteControl instead of changing hardcoded value in the script, since it requires multiple tries to find the exact rotation.

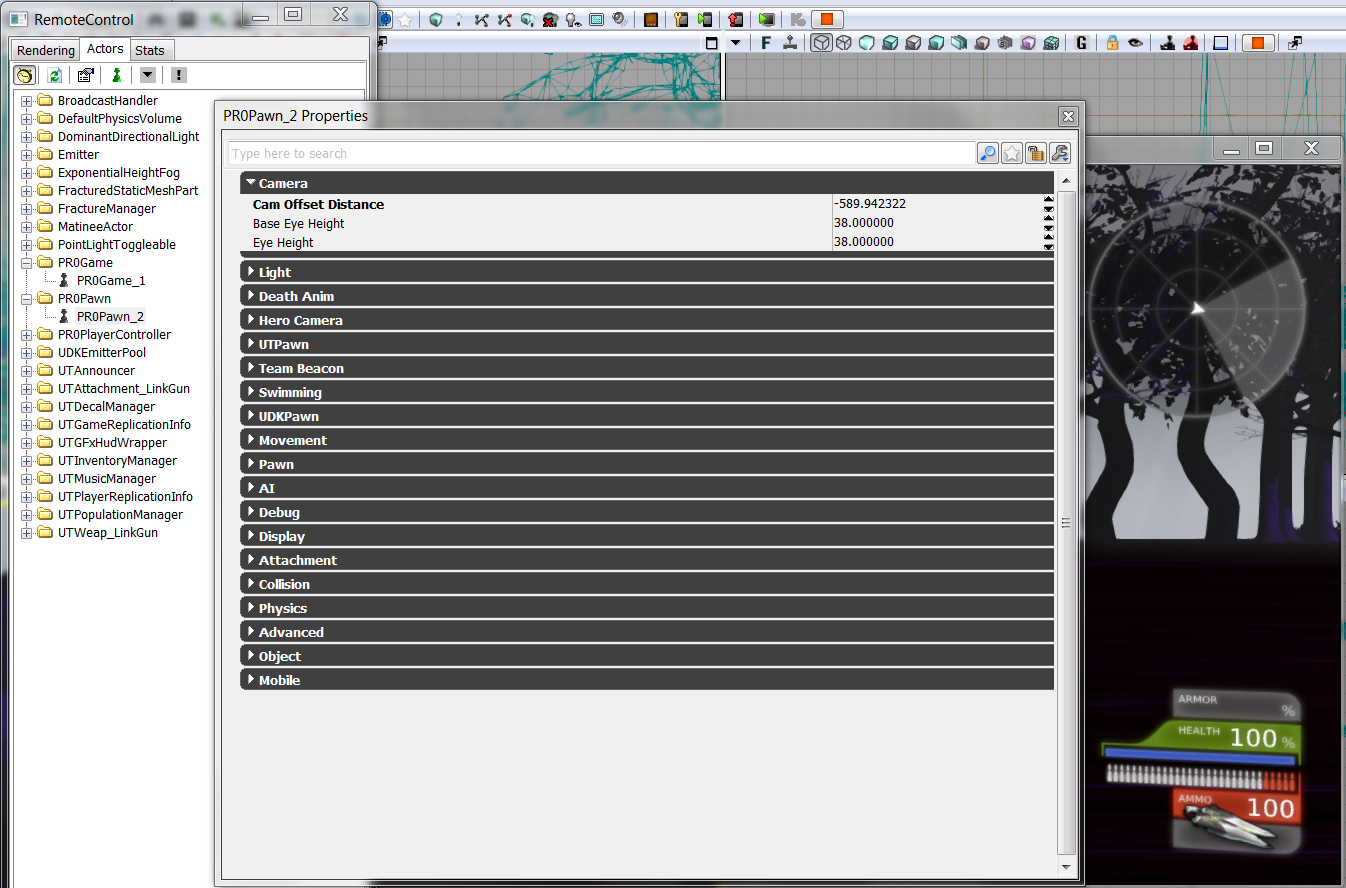


Figure 11: Trying to change the camera distance of PR0Pawn on the run using RemoteControl.

In addition, RemoceControl is also useful to track the values of otherwise invisible internal variables, such as the amount of frames (images produced by the engine) per second and the light intensity on the map in the top left hand corner of the player’s screen.

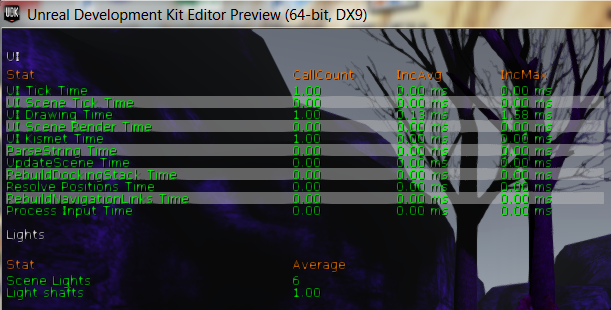


Figure 12: RemoteControl to track invisible variables and information.

## Alpha and Beta testing

We are planning to have both Alpha and Beta tests in short future. Both tests have different intended outcome and audiences. On one hand, Alpha test will be closed to a few testers only and it is intended to be an acceptance testing to test the balance of the game. E.g. when the game is judged to be too easy or too complicated by testers, we would conduct some changes to balance the game before beta testing. On the other hand, Beta test will be open and available to public, mainly UST public. Beta testing will try to uncover glitches that the development team overlooked during the development process.

# WHAT WE WILL DO NEXT

[1]<http://www.icheg.org/icheg-game-history/timeline/>

[2]<http://www.gameinformer.com/b/features/archive/2012/12/27/2462440.aspx>

[3]<http://forums.epicgames.com/forum.php>