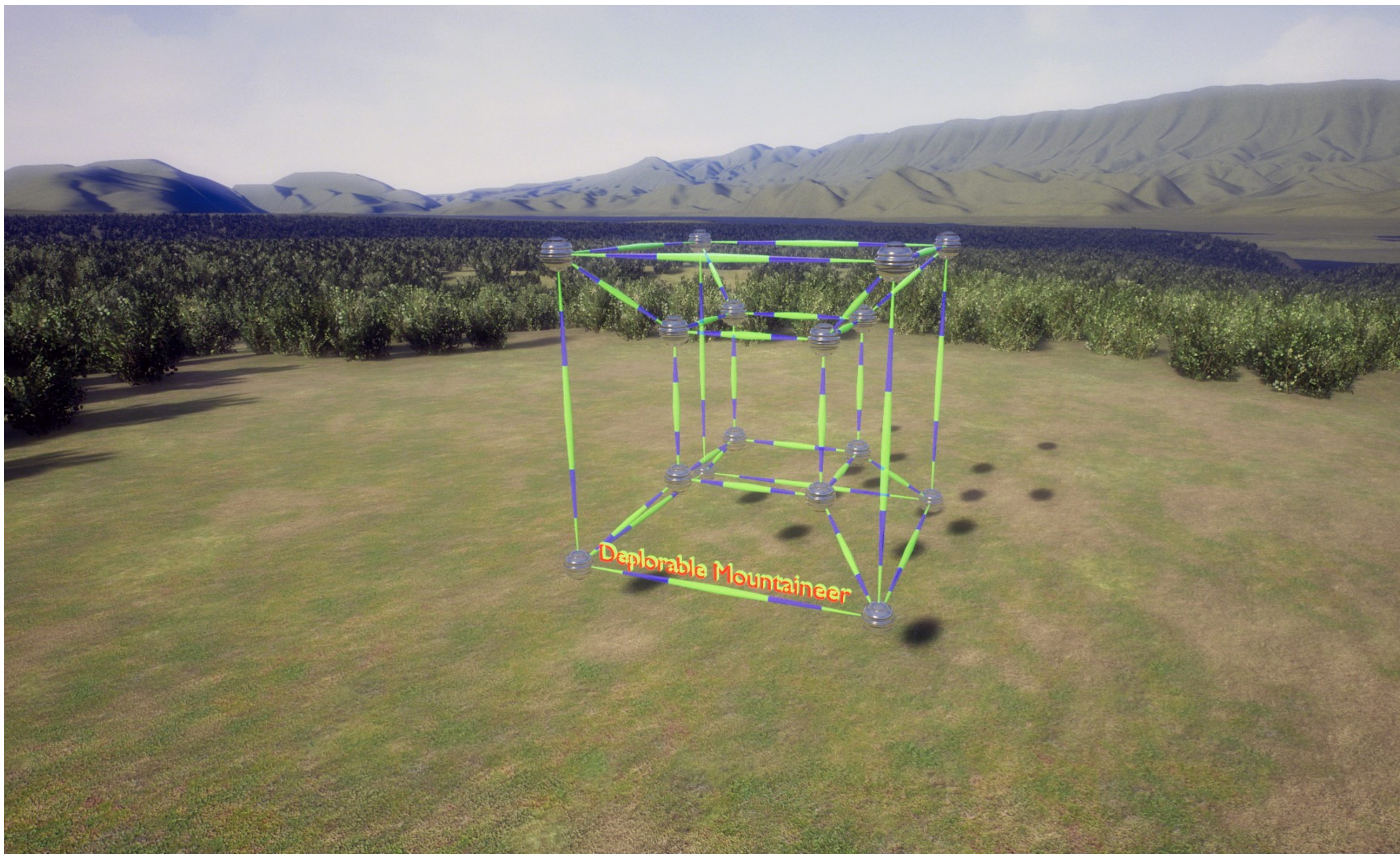
Deplorable Mountaineer Moorefield, WV

Classic FPS Game Project Design Document

By Todd D. Vance



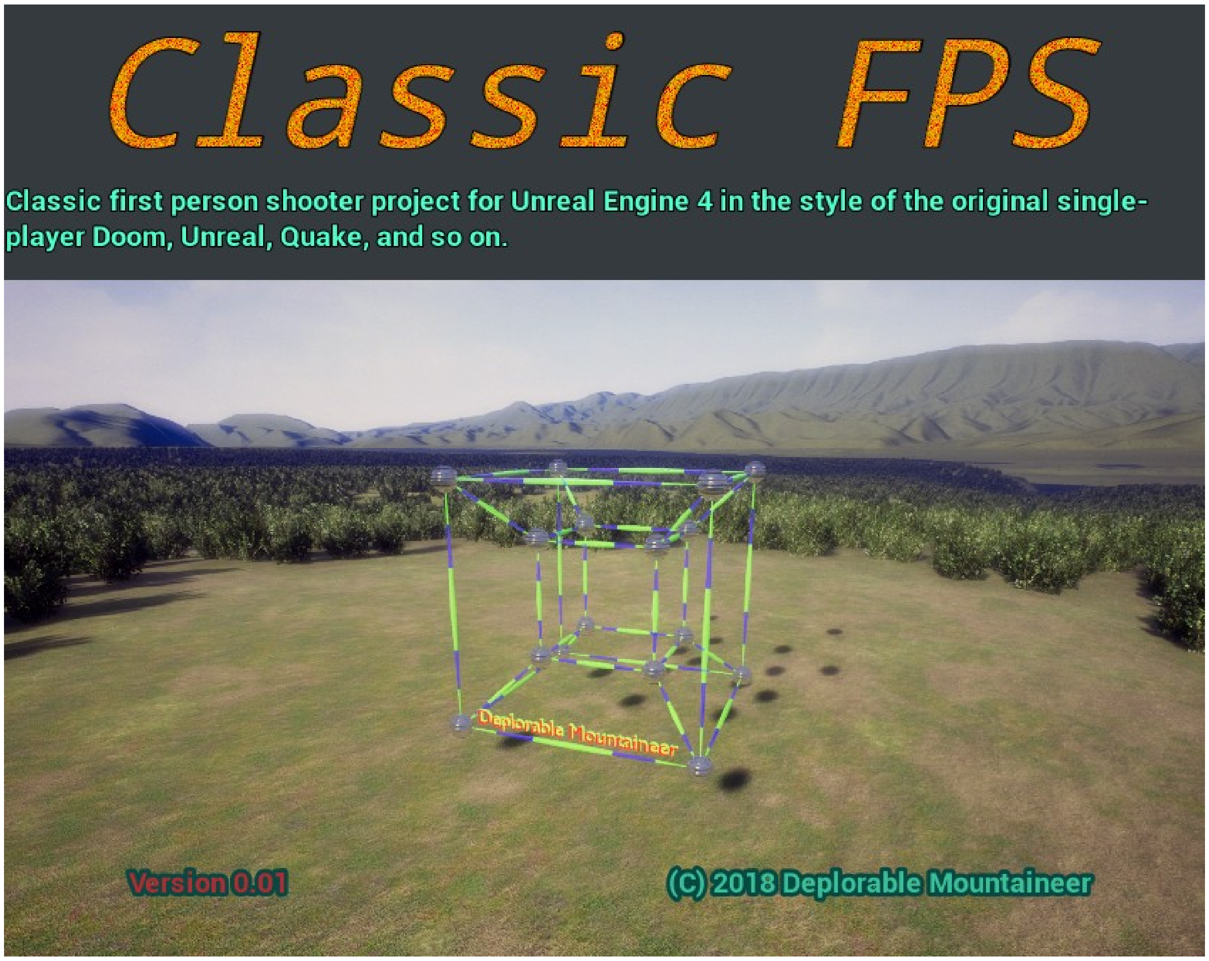
Foreword

Contents

Chapter 1: Game Information

1.1

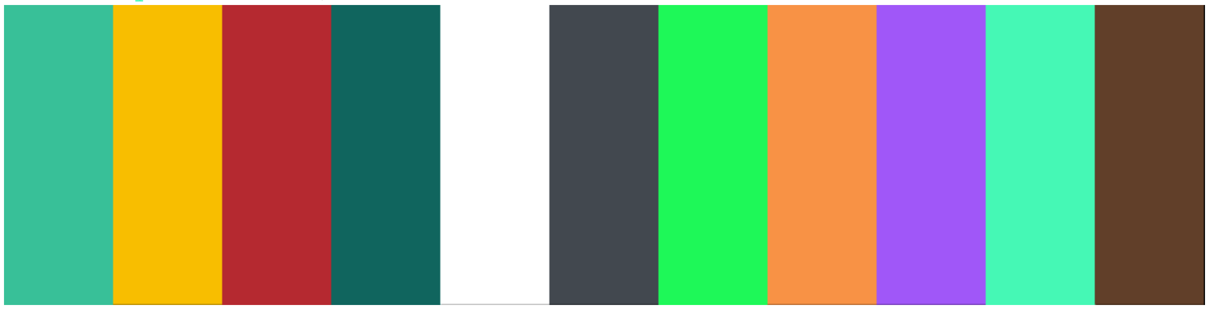
. **Game Information Body**


Classic FPS
Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Deplorable Mountaineer
Version 0.01
(C) 2018 Deplorable Mountaineer

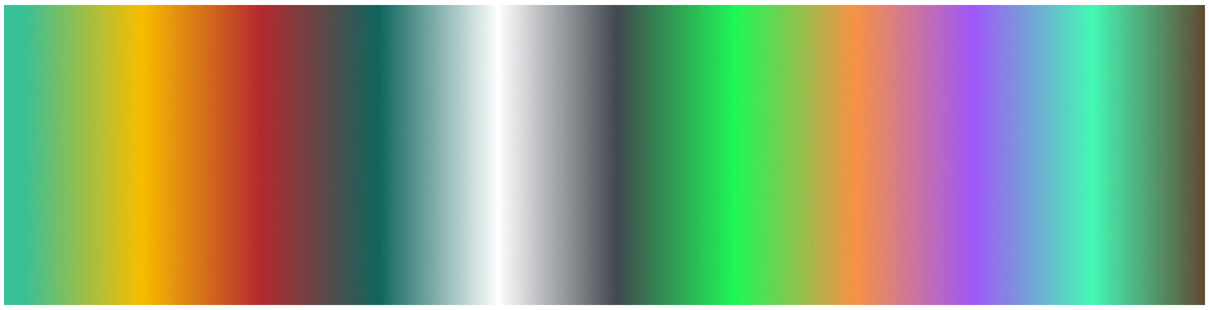
Unique Project ID: {AED64A86-47D8-D36A-F0A9-2093602951A6}
Name: Classic FPS Version 0.01
Description: Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Company: Deplorable Mountaineer
Website: https://deplorablemountain.wixsite.com/games
Copyright: © 2018 Deplorable Mountaineer

Contact: Deplorable.Mountaineer@gmail.com

Palette Colors: #38c098 (Shamrock), #F8BE00 (Amber), #B52930 (Tall Poppy), #10655E (Eden), #ffffff (White), #42484f (Mako), #1EF858 (Malachite), #F89245 (Tan Hide), #A057F8 (Heliotrope), #45F8B5 (Aquamarine), #613F29 (Irish Coffee)



Palette Gradient:



Chapter 2: Folder Structure

2.1

. Folder Structure Body

Under Content:

file://D:\\Documents\\Repos\\Deplorable_Mountaineer\\UE4\\Classic FPS\\Content

Chapter 3: Assets

3.1. Material Bases

Materials

3.1.1. PBR Settings

For game use only--I made up some of the values.

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/Materials.ods

3.1.2. Material Masters

Base

Most materials use this shader; has parameters for almost everything

Translucent_Expensive

A shader for translucent materials using the most expensive operations; use sparingly

Mask

A shader for RGB material masks

3.1.3. Parameter Names

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/MaterialParameters.ods

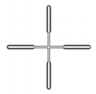
3.1.4. Material Functions and Layers

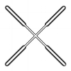
MF_MaskLerpRGB

A function used for making complex RGB masked materials with smooth transitions

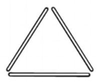
3.1.5. Textures

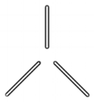
Crosshairs

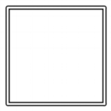

Cross

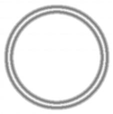

X

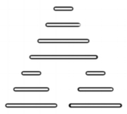
Dot

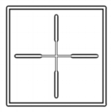

Triangle

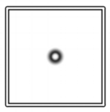

Trefoil

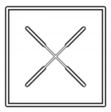

Box

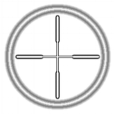

Circle

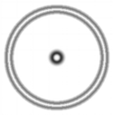

Rangefinder


Box Cross


Box Dot


Box X


Circle Cross


Circle Dot

Circle X

Metal

dimpled, rough and scratchy, Rusty, Holey, verdigris, Brushed, galvanized, Corrugated, Grate, Fence, Louvred Vent, Duct

Rock

smooth, conglomerate, Sandy, Gravelly, Pitted, jagged, marble, slate, limestone, Granite, Sandstone

Icons

Ammo

Up Arrow

Carryable

Controllable

Controller

Door

Health

Key

Light

Sky

Pawn

Ruler

Shield

Stairs

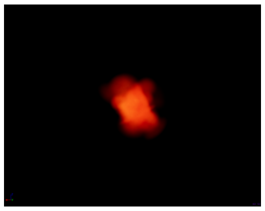
Strength

Switch

Weapon

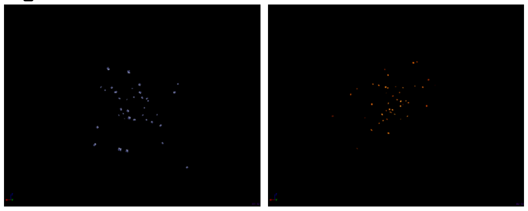
Cloth
Fine Weave, Coarse Weave, Carpet
Decals
bullet holes, footprints, Stain, signs, puddles, Dirt, Scratches,**Wall**
Plaster, wood, wood with grain, particleboard, Wood with knots, brick, concrete, concrete block, Stone
Floor
tiles of various types and shapes, cobblestone, Concrete, Concrete grime, Concrete Panels, Concrete Tiles, Concrete grooved, Concrete Pitted, Concrete Plank-like **Ground**
Dirt, Mud, Gravel, Grass, Snow, Patchy Grass, Ice, Dirty Ice, Rough Ice, Dirty Snow, Patchy Snow, Exposed Rock, muddy rocks, Moss, Dirt and Pebbles, Ground with Foliage
Utility
noise, clouds, gradients
Other
Painted metal, colored glass, patterned cloth, etcLeather, Rubber, Plastic, Ceramic, etc. to match PhysMats
Hazard patterns
Grids and tile patterns, hex and square, etc. Holey patterns, Vent louvre patterns, etc.
Diamond Plate, Duct, Planks, Parquet

3.1.6. Particles

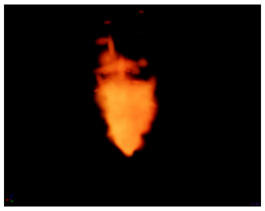


Hit effect

Sparks



Fires



Beams

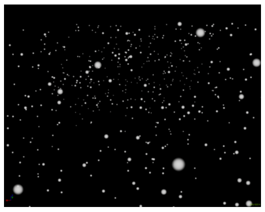
Projectile particles

Splash from water

Slime

Lava

Snow



Steam

Smoke

Lightning

Shrapnel

Gibs
Dust

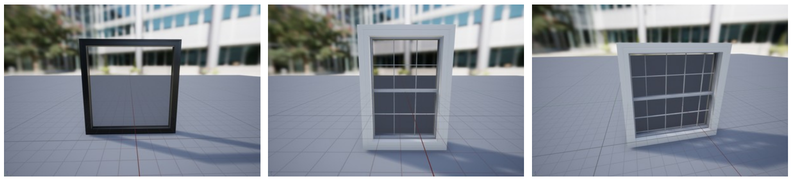
Contrail
Light halo

3.1.7. Architectural Parameters

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/Architectural.ods

3.1.8. Static Meshes

Window



Shapes

Cube, sphere, slope, Square pyramid, Tetrahedron, Octahedron, Icosahedron, Torus, Plane, Cube Chamfer, Cube Inset, Cone, Platform, Circular platform, Platforms of various polygon shapes, Annular platform, Arrows

Arch

Walls, Walls with window/door openings
Pipes and joints, fence parts,. Stair parts, hose parts

Props

Barrels, Crates, including exploding barrel, Boxes, bags
Bricks, Cement blocks, Rocks, Boulders
Cones, Saw horses, Signs, Manhole covers
Trash cans, chairs, desks, tables, Benches, Couches, Barstools, Grates, vents,

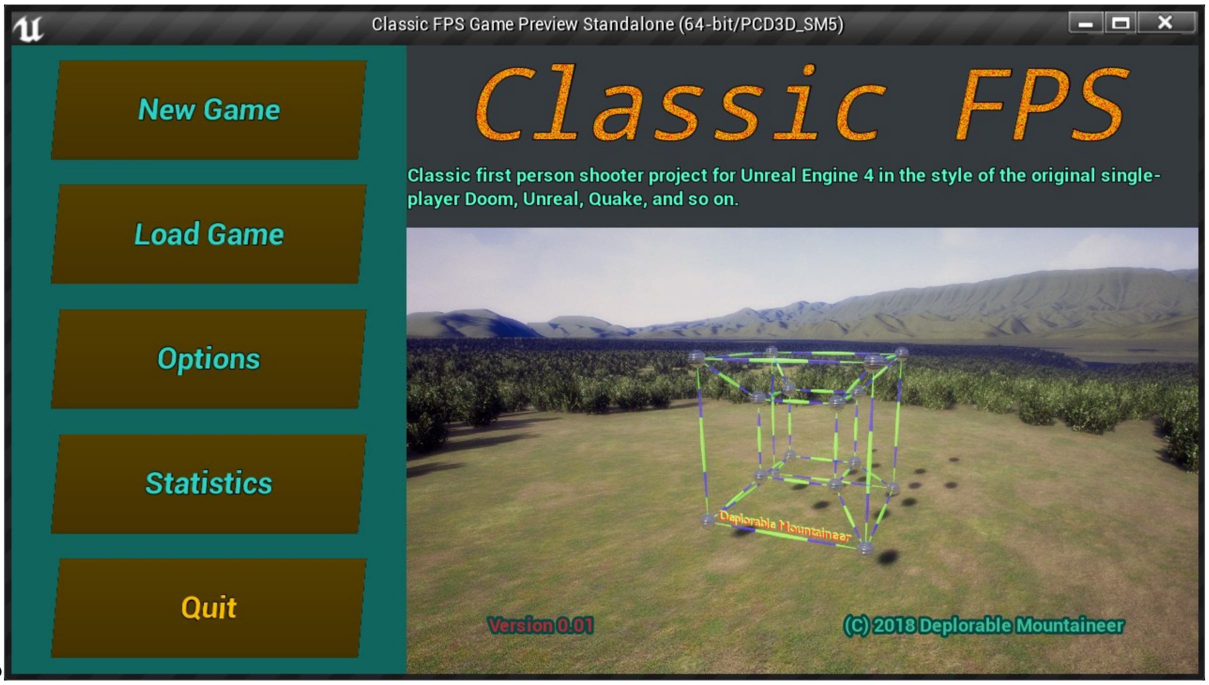
Foliage

3.2. Gameplay Base

Gameplay Base

3.2.1. Gameplay States

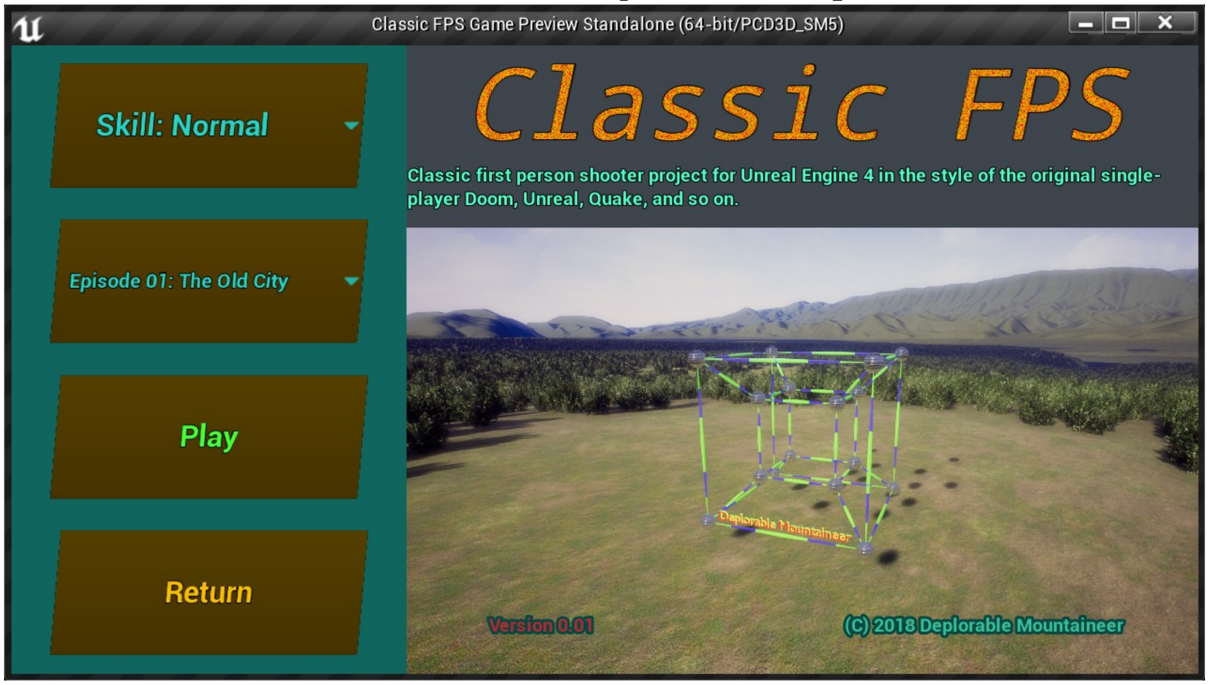
3.2.1.1. Main Menu

- 
Classic FPS Game Preview Standalone (64-bit/PCD3D_SM5)
New Game
Load Game
Options
Statistics
Quit
Classic FPS
Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Deplorable Mountaineer
Version 0.01
(C) 2018 Deplorable Mountaineer

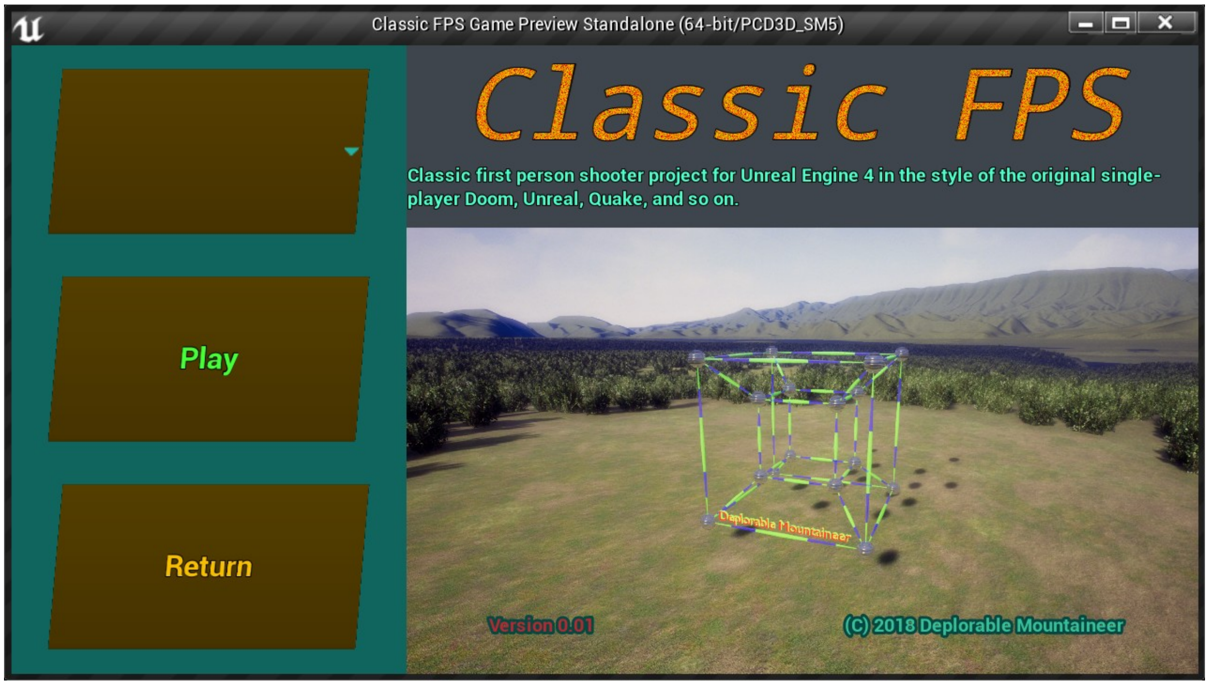
Available choices: New Game, Load Game, Options, Statistics, Quit

The main menu is brought up by the BeginPlay event of the Level Blueprint of the Entry map. This same level blueprint has properties UI Colors and UI Text for changing various menu properties for all menus.

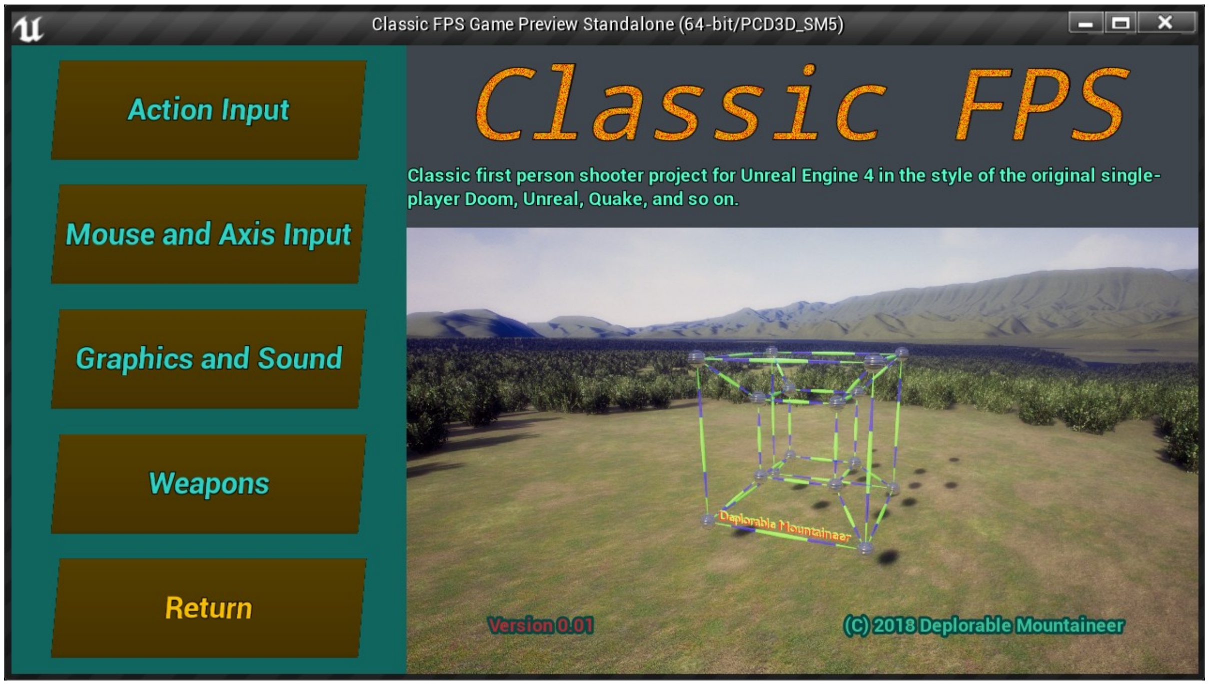
New Game: Select Skill, Select Episode, Play, Return


Classic FPS Game Preview Standalone (64-bit/PCD3D_SM5)
Skill: Normal
Episode 01: The Old City
Play
Return
Classic FPS
Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Deplorable Mountaineer
Version 0.01
(C) 2018 Deplorable Mountaineer

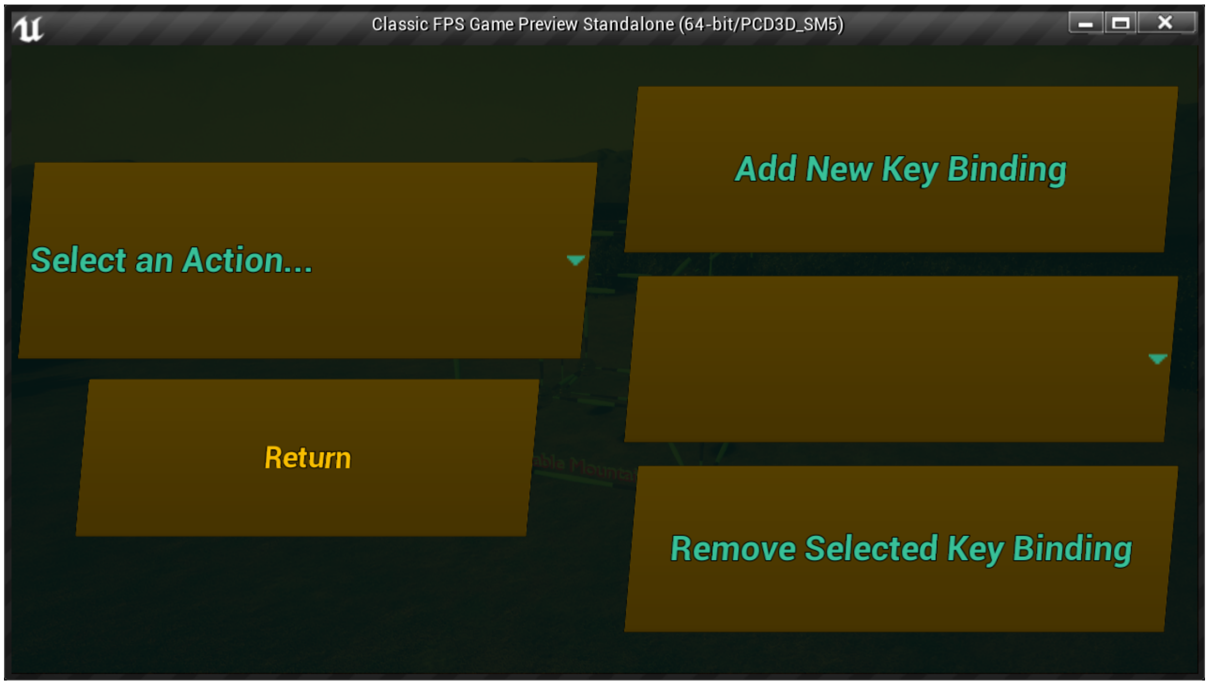
Load Game: Select Game, Play, Return


Classic FPS Game Preview Standalone (64-bit/PCD3D_SM5)
Play
Return
Classic FPS
Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Deplorable Mountaineer
Version 0.01
(C) 2018 Deplorable Mountaineer

Options: Action Input, Mouse and Axis Input, Graphics and Sound, Weapons, Return


Classic FPS Game Preview Standalone (64-bit/PCD3D_SM5)
Action Input
Mouse and Axis Input
Graphics and Sound
Weapons
Return
Classic FPS
Classic first person shooter project for Unreal Engine 4 in the style of the original single-player Doom, Unreal, Quake, and so on.
Deplorable Mountaineer
Version 0.01
(C) 2018 Deplorable Mountaineer

Action Input: Add New Key Binding, Remove Selected Key Binding


Classic FPS Game Preview Standalone (64-bit/PCD3D_SM5)
Select an Action...
Add New Key Binding
Return
Remove Selected Key Binding

Mouse and Axis Input: invert mouse, mouse sensitivity
Statistics: top scores, achievements
Graphics and Sound: Resolution, Effects Volume, Music Volume
Weapons: Priority, Auto Switch on Pickup (always, never, if better) Switch to Best Weapon when Out of Ammo

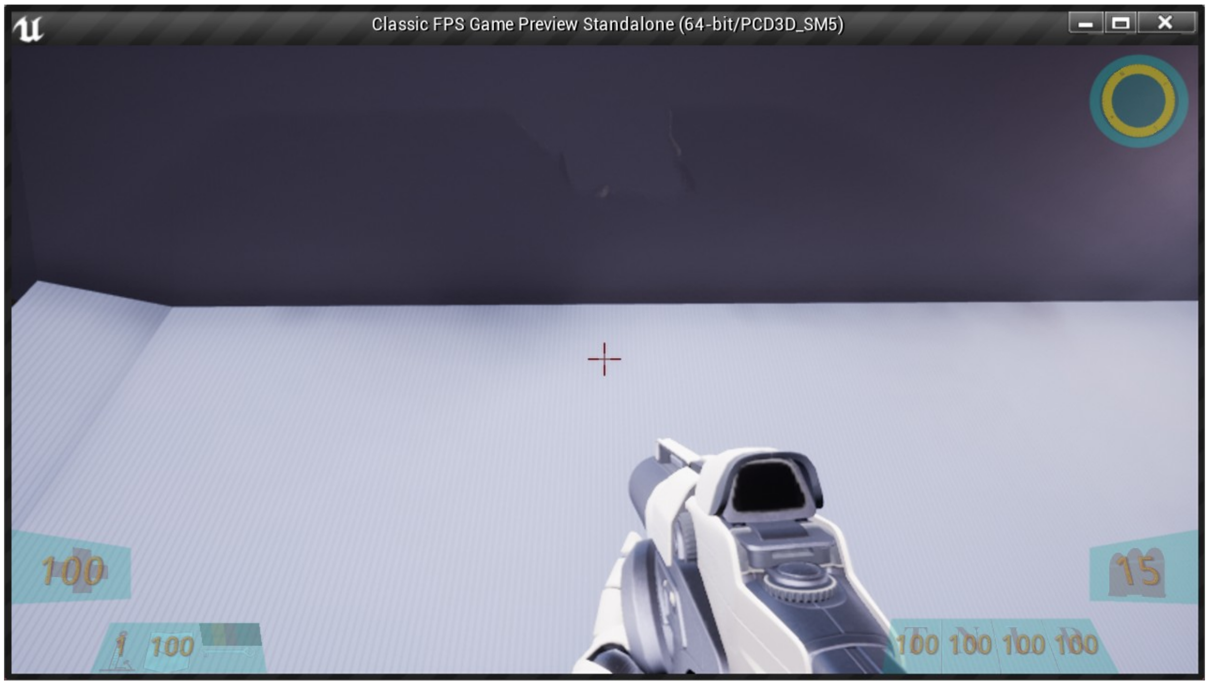
3.2.1.1.1. Main Menu Classes

BP_Pawn_Menu:
MainMenu_GameMode:
BP_MainMenu (HUD class):

3.2.1.2. Pause Menu

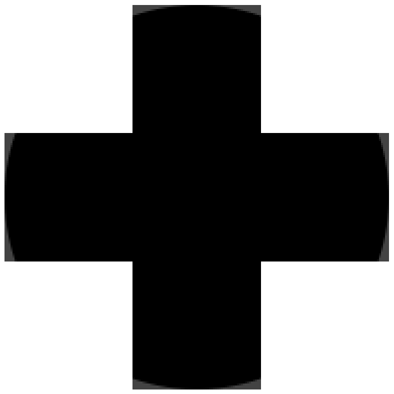
3.2.1.3. HUD

Ammo on bottom right, change color when low, with flash when it turns low and audible noise;

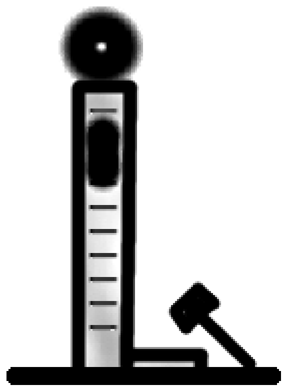


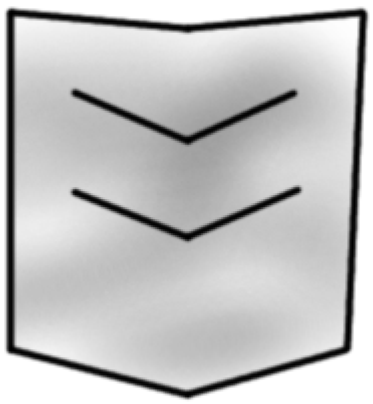
Radar Top right (as a powerup?), different color for enemies that are close; compass style

Health on bottom left; flash, audio warning when turns low, change color when low.



Next to health are Strength, Shield, and Keys.





Next to ammo are the powerups: Turbo, Night vision, Lift belt, and Radar.

T | N | L | R

Crosshair: fine lines, duplex (outer parts of lines highlighted); also dot and range-finding for long-range weapons.

3.2.2. ClassicFPS Game Mode

3.2.3. ClassicFPS Game State

3.2.4. ClassicFPS Player State

3.2.5. Gameplay Effects

Gameplay Effects

Hurt screen

Red tint, red dirt mask, extreme vignetting, extreme chromatic aberration; effect stronger as health decreases

Dead screen
Night Vision screen
Turbo screen
Low health screen?
Health increase screen
Shield used screen
Effects when enemy hit or killed
Pickup Effects
Low Ammo/Health/Inventory effects
In Battle Effects
Boredom Effects
Goal Achieved Effects

3.3. Character

Character

3.3.1. Default Character Properties

- **file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/DefaultPlayerCharacter.ods**

3.3.2. Character Capabilities

Assuming for falling damage, falling on default surface, health at 100 and no shielding. Computed values are rounded. Max survivable fall height actually varies according to physmat of surface hit. For jump height, remember to add max step height to determine max height of platform that can be jumped onto. Also all values appear to be approximate: in-game values differ slightly from computed values for reasons the author doesn't know.

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/CharacterCapabilitiesComputed.ods

Formulas used (assumes no air resistance, assumes character movement is physically accurate):

Jump Height: jump velocity * time + 0.5*-980*G*time*time
Velocity curve when jumping: Jump velocity – 980*G*time
Time for (single) jump: time = jump velocity / 980 / G
Double jumps double the height and time, but not velocity.

3.3.3. Action Bindings

- **file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/DefaultActionBindings.ods**

3.3.4. Axis Bindings

3.3.5. Character Properties

- **file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/DefaultCharacterProperties.ods**

3.3.6. Character Property Value Modification

3.3.6.1. Health

On picking up a health item, its value is added to the current health, capped at 100 (no cap if it is a super health item). If there is no change in health from the item, the item will not be picked up.

On taking damage, the health is diminished by the modified amount of damage, where the amount of damage is modified both by strength and amount of shielding possessed. Sometimes damage is multiplied if it hits certain bones instead of others (such as a head shot).

3.3.6.2. Strength

The strength of most NPCs is fixed according to the character the NPC represents. The strength of the player can increase with "experience." The strength can temporarily increase upon consuming a turbo power boost ("Nitro") power up, but reverts when the power up runs out.

3.3.6.3. Current Weapon Slot

This is the slot number of the weapon currently being carried. Weapons are numbered from 1 through unlimited (no hard limit to number of weapons, but only 1 through 9 can be selected by a keypress, others must be scrolled to with the scroll wheel, for example). It is changed when a weapon is dropped or selected. If dropped or tossed or stashed, the slot changes to 0 (no weapon). If selected (manually, or if auto-select is triggered in certain cases), it changes to whatever weapon slot is selected.

3.3.6.4. Third Person Mode

If true, the character is in third-person mode (relevant to player only).

3.3.6.5. Enemies Killed

A list of unique names of enemies killed, for statistics purposes. Added to whenever this character kills someone, and reset on starting or restarting a level.

3.3.6.6. Secrets Found

A list of names of secrets found, for statistics purposes. Secrets given the same name will be considered the same secret and not be counted more than once. Added to whenever this character overlaps a "Secret Area" volume not yet found, and reset on starting or restarting a level.

3.3.6.7. Powerups Found

A list of unique names of powerups found, for statistics purposes. Added to whenever a character walks over or attempts to pick up a powerup, whether or not successful, and reset on starting or restarting a level.

3.3.6.8. Weapons Found

A list of unique names of weapons found, for statistics purposes. Added to whenever a character walks over or attempts to pick up a weapon, whether or not successful, and reset on starting or restarting a level.

3.3.6.9. Ammo Found

A list of unique names of ammo pickups found, for statistics purposes. Added to whenever a character walks over or attempts to pick up ammo, whether or not successful, and reset on starting or restarting a level.

3.3.6.10. Health Found

A list of unique names of health pickups found, for statistics purposes. Added to whenever a character walks over or attempts to pick up health, whether or not successful, and reset on starting or restarting a level.

3.3.6.11. Keys Found

A list of names of keys found, for statistics purposes, and to determine if character can pass through certain areas or not. Keys with the same name are considered equal. Added to whenever a character picks up a key not yet found, and reset on starting or restarting a level.

3.3.6.12. Falling Counter

How long (in seconds) a character has been falling at more than a specified minimum velocity, used to compute damage from falling. It is incremented while the character is falling at greater than the minimum velocity, and reset when the player lands (whence damage is taken if appropriate).

3.3.6.13. FOV

Field of view in degrees, relevant only to the player. Adjusted when the player changes zoom level.

3.3.6.14. Carried Item

A struct holding data for the item currently being carried (so it can be reproduced on loading a saved game). It is modified when a player picks up or drops a physics item.

3.3.6.15. Inventory

An array of structs, each of which holds data for an inventory item being carried (so it can be reproduced on loading a saved game). It is modified when a player picks up or drops an inventory item of the type that stays with the character (such as a weapon).

3.3.6.16. Third Person Camera Arm Length

Follow distance of the third-person camera. Relevant only for the player, and then only in third-person mode. Can be adjusted by the player.

3.3.7. Character Structure

Note that there is currently a bug (as of Version 4.18.3) in UE4 that causes the third-person character mesh to sometimes disappear in a level when the game isn't being played. The first-person arms mesh is still visible, though. It can be fixed by recompiling the appropriate character blueprint. It appears to be harmless: the mesh reappears when you play the game.

A character has a BP_Character_Base as its parent, which in turn has the built-in Character class as its parent. Thus it already comes with a Capsule component as its root, as well as an arrow component (for convenience), and a skeletal mesh component. It also has a character movement component.

The assets in this project are designed for a character 192cm high and 84cm wide, so the capsule half-height is 96, and the capsule radius is 42. One can use a different-sized character, but things like doorway sizes might have to be adjusted.

The camera boom is attached to the capsule component, and the camera is attached to the camera boom. By shortening the camera boom to 0 length, we achieve first-person mode; otherwise, there is an adjustable-length boom third person mode.

The camera has a first-person arms mesh attached to it, visible only to the owning player and in first person mode (in which the other mesh, the third-person mesh, is made invisible to the owning player, but not to others).

The camera also has a box volume attached to it, which in turn has postprocessing components attached for various in-game effects.

3.3.8. Character Operating Items

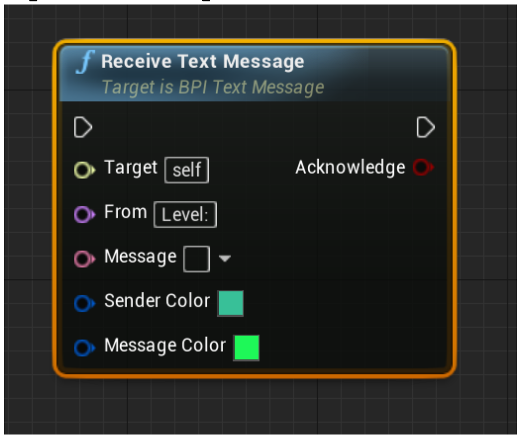
When a character "operates" an item, it first checks if the item is an actor implementing the BPI_Operatable interface. If so, it calls the actor's Operate function.

If not, it checks if it is a pickup. If so, it tries to pick up the item. If not or if picking up fails, it checks if it simulates physics. If so, and if it is not too heavy, the character carries the item.

When the Operate action occurs and the character is already carrying something, it drops the item instead of performing the usual Operate procedure.

3.3.9. Text Messages

Characters can receive text messages. In a blueprint, on a Character actor, just call "Receive Text Message" with the appropriate inputs.



Mostly, the messages have no effect. But with a player character, the message is displayed on the Player's HUD.

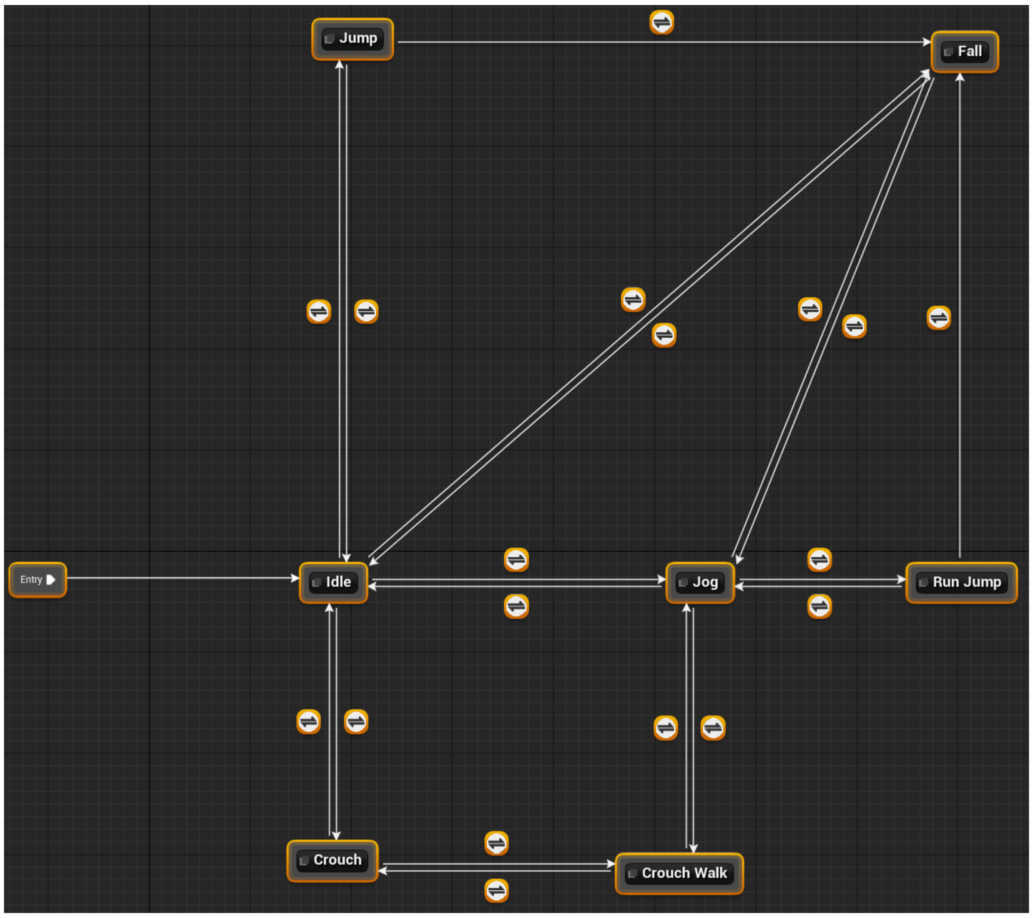
3.3.10. Character Animation

Character Animation

Animation for the default third-person character mesh is controlled by the ABP_Character blueprint.

On each animation update, the BP_Character actor is queried for rotation, velocity, and whether jump button and crouch button are down. The Direction, Speed, Horizontal Speed, Falling, Enable Jump, and Crouching variables are set as a result. These variables then control the animation through the state machine ("Locomotion"). After a brief delay, Enable Jump is reset.

3.3.10.1. State Machine



3.3.10.1.1. Idle

The entry state. Idle animation.

If Jump is enabled and Horizontal Speed is less than 10, transition to the Jump state.

If Jump is not enabled and Falling is true, transition to the Fall state.

If Horizontal Speed is greater than or equal to 10, transition to the Jog state.

If Crouching is true, transition to the Crouch state.

3.3.10.1.2. Jog

Blend Space BS_Jog animation, parametrized by Direction and Horizontal Speed.

If Horizontal Speed is less than 10, transition to the Idle state.

If Horizontal Speed is greater than or equal to 10 and Jump is enabled, transition to the Run Jump state.

If Falling is true and Jump is not enabled, transition to the Fall state.

If Crouching is true, transition to the Crouch Walk state.

3.3.10.1.3. Run Jump

Jump from Jog animation.

If animation done and falling, transition to Fall state.
If animation done and not falling, transition to Jog state.

3.3.10.1.4. Fall
Idle animation.
If not Falling and Horizontal Speed is greater than or equal to 10, transition to Jog state.
If not Falling and Horizontal Speed is less than 10, transition to Idle state.

3.3.10.1.5. Jump
Jump from Stand animation.
If animation finished and falling, transition to Fall state.
If animation finished and not falling, transition to Idle state.

3.3.10.1.6. Crouch
Crouch Idle animation.
If not Crouching, transition to Idle state.
If Horizontal Speed is greater than or equal to 10, transition to Crouch Walk state.

3.3.10.1.7. Crouch Walk
Blend Space BS_CrouchWalk parametrized by Direction and Horizontal Speed.
If not Crouching, transition to Jog state.
If Horizontal Speed is less than 10, transition to Crouch state.

3.4. AI
AI

3.4.1. Bot_Best
Following are definitions of Bot "substates". Numbers mentioned are usually configurable. Blue

3.4.1.1. Wants Health
True if health is less than 75%

3.4.1.2. Needs Health
True if health is less than 50%

3.4.1.3. Wants Ammo
True if less than 100 hit points worth of ammo left

3.4.1.4. Focused Ammo is Full
True if the Focused Ammo is set and the bot has that weapon and it can take no more ammo

3.4.1.5. Focused Powerup is Full
True if the Focused Powerup is set and the bot has that powerup

and it can take no more charge

3.4.1.6. Needs Ammo
True if less than 50 hit points worth of ammo left

3.4.1.7. In Mortal Danger
True if health is less than 25%

3.4.1.8. Has Ammo
True if has any ammo in any possessed weapon

3.4.1.9. Knows Enemy Location
True if enemy (i.e. the player) detected by sight, hearing, or being shot at

3.4.1.10. Suspects Enemy Location
True if enemy detected less than 5 seconds ago

3.4.1.11. In Attack Range of Enemy
True if enemy within range of its current weapon

3.4.1.12. Focused Bot
Teammate bot that is referred to in the next three boolean variables.

3.4.1.13. Focused Bot Forward
A point 400 Unreal Units forward from the Focused Bot

3.4.1.14. Focused Bot Backward
A point 400 Unreal Units backward from the Focused Bot

3.4.1.15. Near Another Bot Neither Attacking or Fleeing
If neither Attacking nor Fleeing, nor Needs Ammo nor Needs Health, and suspects location of other bot (on same team), within 4000 Unreal Units of the bot, and the bot is neither attacking or fleeing

3.4.1.16. Near Another Bot that is Attacking
If neither Attacking nor Fleeing, nor Needs Ammo nor Needs Health, and suspects location of other bot (on same team), within a range of 4000 Unreal Units of the bot, and the bot is Attacking

3.4.1.17. Near Another Bot that is Fleeing
If neither Attacking nor Fleeing, nor Needs Ammo nor Needs Health, and suspects location of other bot (on same team), within 4000 Unreal Units of the bot, and the bot is Fleeing

3.4.1.18. Focused Health
Health that is referred to in the next two boolean variables

3.4.1.19. Near Health
If Knows Location of Health and health is within 4000 Unreal Units of self

3.4.1.20. Knows Location of Health
If knows location of health (bot never forgets if has seen the health, or if it is granted sufficient "knowledge of map" even if it hasn't seen the health; but will forget once the health is taken up unless it is a respawnable health)

3.4.1.21. Bored
In the same state for more than 60 seconds (with some randomness)

3.4.1.22. Focused Waypoint
Waypoint that is referred to in the next boolean variable

3.4.1.23. Near Waypoint
If a waypoint is within 4000 Unreal Units of self

3.4.1.24. Far From Home
If no waypoint is within 10000 Unreal Units of self and there are waypoints for this bot

3.4.1.25. Focused Ammo or Weapon
Ammo or weapon that is referred to in the next two boolean variables

3.4.1.26. Near Ammo or Weapon
If Knows Location of Ammo or Weapon and ammo or weapon is within 4000 Unreal Units of self

3.4.1.27. Knows Location of Ammo or Weapon
If knows location of ammo or weapon (bot never forgets if has seen the health, or if it is granted sufficient "knowledge of map" even if it hasn't seen the ammo or weapon; but will forget once the ammo or weapon is taken up unless it is a respawnable ammo or weapon)

3.4.1.28. Focused Powerup
Powerup that is referred to in the next two boolean variables

3.4.1.29. Near Powerup
If Knows Location of Powerup and powerup is within 4000 Unreal Units of self

3.4.1.30. Knows Location of Powerup
If knows location of powerup (bot never forgets if has seen the powerup, or if it is granted sufficient "knowledge of map" even if it hasn't seen the powerup; but will forget once the powerup is taken up

unless it is a respawnable powerup)

3.4.1.31. States

Here are the states the Bot can be in. The conditionals for transitioning to other states are listed in priority order, that is the first is checked, then if no transition occurs, the second is checked, and so on.

3.4.1.31.1. Attacking

If Suspects Enemy Location AND (In Mortal Danger OR NOT Has Ammo), transition to Fleeing

If Needs Health and Near Health, transition to Collecting Health

If Needs Ammo and Near Ammo or Weapon and can use that ammo or weapon, transition to Collecting Ammo or Weapon

If NOT Suspects Enemy Location, transition to Wandering

(Action) Move toward enemy unless closer than 400 Unreal units, back up instead; Also move left or right randomly

3.4.1.31.2. Fleeing

(Action) Move away from enemy

If Has Ammo AND In Attack Range of Enemy AND Knows Enemy Location AND NOT In Mortal Danger, transition to Attacking

If Needs Health and Near Health, transition to Collecting Health

If Needs Ammo and Near Ammo or Weapon and can use that ammo or weapon, transition to Collecting Ammo or Weapon

After 10 seconds, transition to patrolling

If NOT Suspects Enemy Location, transition to Wandering

3.4.1.31.3. Patrolling

If there are no waypoints, transition to Wandering

(Action) If there is a Focused Waypoint, make that the next waypoint then clear the Focused Waypoint

(Action) go to next waypoint, then increment waypoint

If Needs Health, transition to Finding Health

If Needs Ammo, transition to Finding Ammo or Weapon

If suspects enemy location, transition to Attacking.

If Near Another Bot that is Attacking, transition to Attacking With Teammate

If Near Another Bot that is Fleeing, transition to Defending Teammate

If Wants Health and Near Health, transition to Collecting Health

If Wants Ammo and Near Ammo or Weapon and (Weapon is not Full OR doesn't have weapon), transition to Collecting Ammo or Weapon

If Near Powerup and NOT Focused Powerup is Full, transition to Collecting Powerup

If Bored, transition to Wandering

3.4.1.31.4. Wandering

(Action) Move randomly, avoiding obstacles

If Needs Health, transition to Finding Health
If Needs Ammo, transition to Finding Ammo or Weapon
If suspects enemy location, transition to Attacking.
If Near Another Bot Neither Attacking nor Fleeing, transition to Patrolling
If Near Another Bot that is Attacking, transition to Attacking With Teammate
If Near Another Bot that is Fleeing, transition to Defending Teammate
If Wants Health and Near Health, transition to Collecting Health
If Wants Ammo and Near Ammo or Weapon and can use that ammo or weapon, transition to Collecting Ammo or Weapon
If Near Powerup and can use that powerup, transition to Collecting Powerup
If Far From Home, transition to Patrolling

3.4.1.31.5. Finding Health
If Knows Location of Health, transition to Collecting Health
Otherwise, transition to Wander

3.4.1.31.6. Finding Ammo or Weapon
If Knows Location of Ammo or Weapon, transition to Collecting Ammo or Weapon
Otherwise transition to Wander

3.4.1.31.7. Collecting Health
(Action) Move toward Focused Health
On collection of Health, or failure, transition to Wander

3.4.1.31.8. Collecting Ammo or Weapon
(Action) Move toward Focused Ammo or Weapon
On collection of Ammo or Weapon, or failure, transition to Wander

3.4.1.31.9. Collecting Powerup
(Action) Move toward Focused Powerup
On collection of Powerup, or failure, transition to Wander

3.4.1.31.10. Defending Teammate
If Suspects Enemy Location AND (In Mortal Danger OR NOT Has Ammo), transition to Fleeing
(Action) Move toward Focused Bot Backward
If Knows Enemy Location, transition to Attacking
If NOT Near Another Bot that is Fleeing, transition to Wandering

3.4.1.31.11. Attacking With Teammate
If Suspects Enemy Location AND (In Mortal Danger OR NOT Has Ammo), transition to Fleeing
(Action) Move toward Focused Bot Forward
If Knows Enemy Location, transition to Attacking

If NOT Near Another Bot that is Attacking, transition to Wandering

3.5. Inventory

Inventory

The pickup procedure is roughly as follows: if a pickup is overlapped, or if the player "operates" the pickup, its unique name (or just the regular name in the case of keys) is added to the list of pickups found. Then it is determined whether or not the item can in fact be picked up.

If the player does not already have the item, it is picked up (depending on the pickup, it might be destroyed and its data added to the player rather than the object literally being attached to the player). If the player already has the item but it is of a type that can be depleted (such as ammo or a weapon), the item the player has is replenished with the pickup's amount (if it is positive) and the pickup is destroyed. Otherwise, the pickup is ignored (though if it is a physics body, it may be nudged by the player).

3.5.1. Key Base

3.5.2. Health Base

Two properties: Amount (of health to be added when picked up, float), and Super (if true, ignores the maximum of 100 health points, Boolean).

3.5.3. Powerup Base

3.5.4. Ammo Base

3.5.5. Weapon Base

- **file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/Weapon.ods**

Also current ammo, last-fire-time

Weapon types include:

Hitscan (beam, fast projectile, or invisible), Projectile (particle or mesh), Bouncing Projectile (grenade), Controllable Projectile, Homing Projectile, Sticky Projectile (Mine), Wide-angle hit (flame thrower).

Effects on target include damage to health and physics. Can add other effects, like stealing powerups.

3.5.6. Comparison of Weapons

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/Weapon_Comparison.ods

3.6. Physics Mesh Base

Physics Mesh Base

3.7. Other Blueprint Objects

Other Blueprint Objects

Characters, Weapons, ammo, pickups, projectiles, Pickup Spawner

3.7.1. Stair builder

Enter the number of steps. Default step has a rise and run both of 15. If using a different step mesh, it must have a socket named "Next" for the next step.

To ensure bots can walk up and down the stairs, click the RecastNavMesh actor and change Cell Height to 22.5.

3.7.2. Pipe builder

3.7.3. Fence builder

3.7.4. Hose Builder

3.7.5. Railing builder

3.7.6. Jump pad

Use any actor for jump target (suggested: Target Point actor). Set this in the Target property of the jump pad. If jump pad fails, it could be that there is no solution in the velocity range given, so try changing the min speed and max speed. Use the curved beam to help with positioning (it can be left on or turned off in game--it only gives the approximate path, since it is a spline and not a parabola). The landing area should be at least as large as the jump pad's trigger volume (default radius of 80 so landing area needs diameter of 160 or more) because of inaccuracies due to approaching the pad from different directions.

3.7.7. Placeable shooters

3.7.8. Day and night skies

3.7.9. Lighted signs

3.7.10. Monitors

3.7.11. security cameras

3.7.12. Teleporter

3.7.13. Spawner

3.7.14. Fan

3.8. Sounds

Sounds

Guns firing, reloading
Footsteps, thud from fall
Object dropped
Projectile impact
Door open close lock
computery
fans
Elevator
Pickup, injury, death sounds
Powerup use sounds
Menu UI sounds
Fire, Explosion, Steam hiss
Rocket travel
Grenade clink
Running water
Wind
Outdoor ambience
City Ambience
Starter music
Water drip

3.9. Actor Base

Actor Base

3.9.1. Properties

Name
Unique Name (just use name-in-world)
Icon
Colors
Additional Tags (automatically populates list of actor tags)

3.9.2. Controllable Objects

Controllable objects have actions (of type Name) that can be called by a switch (a switch is really a state machine). Doors and Lights are examples, but so is a switch: so one switch can affect the state of another, allowing three-way switches, combo lock switches, etc.

A controllable object inherits from the BPI_Controllable interface, which provides the method "Do Action", which takes an Action:Name input to select the action, as well as input parameters (which are used or ignored depending on the action) Float Input:Float, Vector Input:Vector, Rotator Input:Rotator, Color Input:Linear Color, Transform Input:Transform, and Name Input:Name. It returns Success:Bool, True if successful, and False if not (for example, parameter out of range or action of given name does not exist).

One way to implement this method is to switch on the Action name, each branch calling a function named the same as the action and taking exactly the arguments they need. Another way is to inherit from BP_Controllable, and set the Allowed Actions array to be the list of action names, and override Do Action By Number to switch on the integer index of the action and perform the actions using the variables Float Input, etc., for inputs.

3.9.2.1. Doors

3.9.2.1.1. Door Actions

Open
Close
Open Reverse
Close Reverse
Lock
Unlock
Operate

- if open, close
- if open reverse, close reverse
- if closed and unlocked, open

●If operated from X direction

- if open, close
- if open reverse, close reverse
- if closed and unlocked and reversible, open reverse
- if closed and unlocked and not reversible, open (note sliding doors typically not reversible)

3.9.2.1.2. Door Properties

Auto Close Delay: if 0, never auto close, otherwise after delay of that many seconds
Open Speed
Close Speed
Auto Open: if character is in trigger volume, opens automatically
Auto Open Reverse: if character is in reverse trigger volume, opens automatically in reverse
Operatable: "operate" opens the door; otherwise need to find a switch somewhere.
reversible: if false, open reverse fails.
Locked: if true, open and open reverse fail; if physics mode and closed, door becomes immovable
open state, closed state, open reversed state

3.9.2.2. Lights

3.9.2.2.1. Light Actions

Actions typically follow a curve (ease-in ease-out) though some lights might have other behavior (E.G. fluorescent light flicker).

On: turn on to previous dimmer amount

Off: turn off regardless of dimmer amount

Toggle: (usual action for Operate)

Set Dimmer Amount: value from 0 to 1, min brightness to max brightness

Set Light Color

Set Light Direction

3.9.2.2.2. Light Properties

Max Brightness

Min Brightness (for dimmer, nonzero for technical reasons, but can be so near 0 it is essentially off)

Direction (rotator): direction a light faces, if applicable

Color

Dimmer Amount (from 0 to 1)

3.9.2.3. Switches

3.9.2.3.1. Switch Actions

Toggle: Change to next state (states are ordered in an array by integer; next state is the next-indexed state)

Set State: set to one of the states available to the switch.

3.9.2.3.2. Switch Properties

State: usually on or off, though could be more; represented by a value of type "Name"

On Enter State: mapping from states to actions on a controlled object (door, light, etc.); action taken when state is entered.

On Exit State: mapping from states to actions on a controlled object (door, light, etc.); action taken when state is exited.

Chapter 4: Physics

4.1. Collision

Note, most values are the UE4 Defaults.

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/Collision.ods

4.2. Physical Surfaces

These values are for game purposes--in some cases I just made them up.

file:///D:/Documents/Repos/Deplorable_Mountaineer/Spreadsheets/PhysicalSurfaces.ods

Chapter 5: Utility

Chapter 6: Tutorials

6.1. Make A New Touchplate Switch

A touchplate is a type of switch that is activated by a character standing on it. This tutorial can be used as a guide for making other kinds of switches as well. Switches already in the project include the BP_Switch_Toggle (a standard light switch the character "operates") and BP_Switch_Button (a big button to push the character "operates"). These can be studied for further information.

1. Open Blender and delete the default Cube Mesh
2. Shift-A to add mesh, select Mesh→Cylinder.
3. In the Add Cylinder properties panel (by default, lower left side), ensure the following properties:
    1. Vertices: 32
    2. Radius: 1
    3. Depth: .1
    4. Cap Fill Type: Ngon
    5. Generate UVs: Yes
    6. Align to View: No
    7. Location: x=0.000, y=0.000, z=0.000
    8. Rotation: x=0°, y=0°, z=0°
9. Ctrl-S to save. Choose an appropriate directory and filename (such as "Touchplate.blend"), and save it.
10. Make sure the touchplate mesh is selected (right-click to select), and nothing else.
11. In the top menubar, select File→Export→FBX (.fbx)
12. In the Export FBX properties panel (by default, lower left side), ensure the following properties under the given tabs:
    13. Main Tab:
        1. Version: FBX 7.4 binary
        2. Scale: 1.00
        3. Apply Scale: All Local
        4. Forward: -Z Forward
        5. Up: Y Up
        6. Selected options: all of them (Empty, Camera, Lamp, Armature, Mesh, Other)

7. !EXPERIMENTAL! Apply Transform: No
8. Custom Properties: No
9. Path Mode: Auto
10. Batch Mode: Off

11. Geometry Tab
12. Apply Modifiers: Yes
13. Use Modifiers Render Setting: Yes
14. Smoothing: Face
15. Loose Edges: No
16. Tangent Space: No

17. Armature Tab
18. Only Deform Bones: No
19. Add Leaf Bones: No
20. Primary Bone Axis: Y Axis
21. Secondary Bone Axis: X Axis
22. Armature Node Type: Null

23. Animation Tab
24. Baked Animation: Yes
25. Key All Bones: Yes
26. NLA Strips: Yes
27. D: All Actions: Yes
28. Force Start/End Keying: Yes
29. Sampling Rate: 1.00
30. Simplify: 1.00

31. Navigate to your Unreal project, then Content directory, then Tutorials directory, then Touchplate directory.
32. Set the name of the file to save to be SM_Touchplate.fbx
33. Click the Export FBX button.
34. Go to UE4 Editor.
35. You should see a popup; select "Import" (if not, click the Import button in the content browser, navigate to where you saved the FBX file, and import it).
36. Now you see an FBX Import Options panel. Ensure the following properties:

37. Skeletal Mesh: No
38. Auto Generate Collision: Yes
39. Leave advanced properties at their defaults
40. Transform: leave default all zero except scale is 1.0
41. Miscellaneous: Convert Scene yes, Force Front X Axis No, Convert Scene Unit No, advanced properties, leave defaults
42. LOD Settings: Auto Compute LOD Distance Yes
43. Materials: Search Location: Local, Import Materials: no, Import Textures: no, leave advanced options as defaults

44. Click Import All.
45. Your touchplate mesh is now in the Content Browser under Content/Tutorials/Touchplate.
46. Optional: give the touchplate a material of some kind (such aSs MI_Rubber_Blue2).
47. Close the static mesh editor window.

48. **Create a Touchplate** Blueprint.

49. In the Content Browser in the UE4 Editor, navigate to the Content/Tutorials/Touchplate folder, the same folder where the SM_Touchplate mesh asset is.

50. Right click, then select Blueprint Class. A popup appears titled "Pick Parent Class"

51. If necessary, click the arrow next to "All Classes" to show all classes. In the search bar, begin typing BP_Switch_Base. You do not need to type it all; the BP_Switch_Base class will appear in the hierarchy below the search box.

52. Select BP_Switch_Base (because of a bug, you might have to select it twice to get it highlighted). Then click the green Select button at the bottom of the popup.

53. A new blueprint with the name "NewBlueprint" will appear in the content browser. While the name is still highlighted (if it is not, clicking the asset and hitting F2 will let you rename it), change its name to BP_Touchplate.

54. Double click the new BP_Touchplate asset to open the blueprint editor for it.

55. Make sure the Viewport tab is active (you will see a white sphere in the viewport).

56. To the left (by default), in the Components panel, click the green Add Component button. A dropdown menu will appear

57. Start typing "Static Mesh", and select the Static Mesh option (not either of the two instanced static mesh options, just plain "Static Mesh"). Accept the entry. Hit ENTER to keep the name "StaticMesh" without renaming it.

58. With the StaticMesh component still highlighted, find the Static Mesh section of the Details panel (by default, on the right).

59. Click the box with the down arrow on it and start typing "SM_Touchplate", and when it comes up, select SM_Touchplate for the static mesh. The newly-created Touchplate mesh will appear in the viewport.

60. Find the Collision section in the Details tab. Ensure Generate Overlap Events is checked, and that Collision Presets is set to Overlap All Dynami9c

61. Open the Event Graph tab. If the tab is not there, you can open it by double-clicking the Event Graph item under Graphs in the My Blueprint panel (by default, to the left below the Components panel; if the My Blueprint panel is not there, it can be opened from the menu bar, Window menu). There is a bug in some versions of UE4 so if clicking the EventGraph item doesn't work, right-click and select "Open in New Tab".

62. Highlight the Static Mesh component in the Components tab.

63. To the right (by default), in the Details tab, scroll down, and find a green button with a plus sign in it labeled On Component Hit. Click it. An On Component Begin Overlap (StaticMesh) node appears in the Event Graph blueprint.

64. On the On Component Begin Overlap event node, click and

drag rightward from the right-pointing white-outlined arrow to place a new node. When you let go of the mouse button, an Executable Actions popup will appear.

65. Type "Toggle Action" in the search bar, and when it appears, click on Toggle Action. A new node should appear called Toggle Action, connected to the On Component Begin Overlap node by a white line. (This represents an execution path from the event node to the toggle action node, so that whenever the touchplate is hit, the toggle action event will be thrown. In the Switch Base, this toggle action just toggles the switch between on and off).

66. Compile the blueprint (Click the compile button in the toolbar), then save the blueprint (click the Save button in the toolbar). Close the blueprint.

67. **Place the Touchplate in the Level and Wire it up to something.**

68. Create a new level: select File→New Level, and click Default. Use Ctrl-S to save it, and give it the name Touchplate_Test, and save it in the Content/Tutorials/Touchplate directory.

69. Ensure the Content Browser is in Content/Tutorials/Touchplate where you can see the BP_Touchplate and the SM_Touchplate assets.

70. Drag the BP_Touchplate asset into the scene in the viewport, and set it on the default platform somewhere. Currently the touchplate does not do anything because it is not connected to anything controllable.

71. Navigate to Content/ClassicFPS/Environment/Props/Lights in the Content Browser, and find BP_Lamp_Ceiling. Drag this asset into the scene and move it somewhere reasonable (lift it up above the platform by selecting the viewport, clicking W to show the Move widget, and clicking and dragging the blue arrow on the Move widget) to make it visible.

72. Click the Touchplate actor in the scene to select it.

73. In the details panel, in the State section, are a lot of inputs. We shall only use a few of them. Click the white Plus sign to the right of Controlled Objects to add a new array element. Element 0, set to the value None, appears

74. Click the box containing None and the down-pointing triangle to see a list of assets in the current scene. Find and select the BP_Lamp_Ceiling asset. Thus, this touchplate will control the lamp we just placed in the level.

75. Click the right-pointing right triangle next to On Enter State Actions. Currently the action for state 0 is None. Change that to "Off", so when the touchplate is in state 0, the light will be turned off.

76. Click the white Plus sign to the right of On Enter State Actions to add another element to the array. A new element 1 will appear, with value None.

77. Change the None to "On". Thus, when the touchplate enters

state 1, the light will be turned on.

78. Note the touchplate's current state is 0. Thus, it starts "Off". Let us make sure the light is off so the states are sync'd when the level is played. Click the light asset in the scene. In the controllable section, click the right-pointing white arrow next to Light. Change the Light is On checkbox to "Unchecked". In the scene, the light will turn off. Now the light and the touchplate are sync'd, both in the "Off" state.

79. Test by clicking the Play button in the toobar. When you walk on the touchplate, the light should turn on. When you walk away and walk back onto the touchplate, the light should turn off.

80. Type CTRL-S to save the level.

81.

6.2. New Weapon Creation

The following tasks need to be performed to create a new weapon.

Endnotes