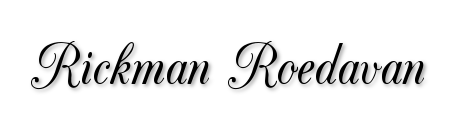


****







Buku "Zetcil 101: Fundamental Pengembangan Game Berbasis Unity" dirancang sebagai panduan pembelajaran pengembangan game menggunakan Unity Game Engine untuk pemula. Buku ini dapat disebarluaskan dan digunakan secara bebas untuk kepentingan edukasi. Penggunaan ulang konten pada media lain hanya diperbolehkan dengan izin dari penulis atau sesuai dengan lisensi yang berlaku.

Semua pihak diharapkan menjunjung tinggi hak cipta dan peraturan penggunaan, serta

menghindari pelanggaran yang dapat mengakibatkan tindakan hukum © 2023.

# Preface

Alhamdulillah, akhirnya buku "Zetcil 101: Fundamental Pengembangan Game Berbasis Unity" versi online dynamic bisa dirilis juga. Saya sebut buku ini sebagai versi online dynamic karena isi kontennya memang akan terus bertambah. Kalau lagi rajin ya saya tambahin, kalau lagi males ya saya cuekin. Suka-suka saya lah pokoknya, kan saya yang nulis.

Zetcil adalah kumpulan script Unity yang telah saya kembangkan sejak 2018 dan telah menjadi suatu master piece yang aji gile keren buset lah. Potongan script berbasis Unity yang sebagian besar saya tulis sendiri, dan sebagian besar lainnya dicolong dari internet ini (wkwkwk), telah mengantarkan banyak mahasiswa menuju kesuksesan. Dan yah, setelah mereka sukses dan ngerti sangat soal Unity, script Zetcil nya memang ga dipake lagi sih. Yang jelas, buku online dynamic ini adalah milestone bagi saya untuk mulai mendokumentasikan Zetcil ke dalam bentuk yang lebih terstuktur.

Saya juga ingin mengucapkan terima kasih kepada tim Multimedia Engineering Technology Research Labs (METALABS), Fakultas Ilmu Terapan, Telkom University, Aliansi Digigame Parade, Konsorsium ICE Institute, serta Program Microcredential Game Developer (PMGD) dari Program Merdeka Belajar Kampus Merdeka (MBKM) di bawah Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi Republik Indonesia yang sudah mendukung penulisan buku ini.

Biasanya kalau para penulis bikin prakata, ujung-ujungnya adalah kami sadari kalau buku ini masih jauh dari kata sempurna, tak ada gading yang tak retak, tak ada jomblo yang tak wibu, bla-bla-bla…, tapi khusus dalam prakata ini saya tidak akan menggunakan kalimat berikut. Alih-alih saya akan tulis: Kami sadari kalau buku ini masih jauh dari kata selesai, karena emang belum selesai. Anyway, semoga buku ini bisa bermanfaat untuk semuanya.

*Be a good pirates. Eh... be a good unity developer!*

**Penulis**

**Rickman Roedavan**

# Table of Contents

[**Preface 2**](#_9f0azsqr3z8e)

[**Table of Contents 3**](#_9h69kfin8tw2)

[**Chapter 1 Multimedia Application Overview 11**](#_cvpynzfss1h8)

[1. Introduction 13](#_wvnu2x3jjbwh)

[2. Navigation 14](#_9981306p3xqv)

[3. Exploration 14](#_b15vqiw52b4r)

[4. Mission 15](#_1xh578aei46v)

[5. Evaluation 16](#_t9edj86o429b)

[6. Flow Variation 18](#_q2pa3seil2jg)

[**Chapter 2 Programming Codification 19**](#_94l2ze21ljiq)

[1. Pola SPOK 20](#_t412dtul9mmt)

[2. Pola Dialog 21](#_iwatmomnq1l5)

[3. Pola Referensi 22](#_cobn8n1vw3ui)

[**Chapter 3 Unity Function Classification 24**](#_b0x1uqb3ueay)

[1. Run/Quit 25](#_ov84zx664pa9)

[1.1 Hello World 26](#_dwaregpe8y7y)

[1.2 Quit Application (via Key) 27](#_j9wrh7brhp49)

[1.3 Quit Application (via Listener) 27](#_2bi4j1tsrvl7)

[1.4 Quit Application (via Event) 28](#_oco0kfl09u3n)

[2. Show/Hide 28](#_2p4elohkaleo)

[3. Active/Deactive 29](#_r8vf0du7py58)

[3.1 Activated GameObject (via Key) 29](#_9u2w78wwou79)

[3.2 Activated GameObject (via Listener) 31](#_js1v4mbm4tof)

[3.3 Activated GameObject (via Event) 32](#_vjfc0x3gxpui)

[4. Enable/Disable 32](#_867ms37976io)

[4.1 Enabled GameObject (via Key) 33](#_5rtw6yntu07j)

[4.2 Enabled GameObject (via Listener) 34](#_oxgamprrcssk)

[4.3 Enabled GameObject (via Event) 35](#_3381emmr806k)

[5. Create/Delete 35](#_35wptytpa1po)

[6. Tag/Layer 37](#_lfrxovi03e75)

[6.1 Find Game Object With Tag 37](#_nyibopflfab8)

[6.2 Find Game Object Within Layer 38](#_vbqfko6ebpj3)

[7. Load/Remove 38](#_nc9jp6qexv90)

[7.1 Scene Loader (via Click) 39](#_sek7s7byi3zd)

[7.2 Scene Loader (via Delay) 39](#_4slig9zffheq)

[7.3 Scene Loader (with Loading Screen) 40](#_dmefkcx0ijhv)

[8. Play/Stop 41](#_oa8irrux0gtl)

[8.1 Audio Controller 41](#_ixgq9gf7acyb)

[8.2 Video Controller 42](#_z5umbluxmmd3)

[9. Save/Load 43](#_ii6u82cihra0)

[10. Export/Import 43](#_ti86g7da3z4m)

[**Chapter 4 Input Classification 44**](#_6smcx4ws5cq3)

[1. Mouse 45](#_kx5emz1dav5x)

[1.1 Mouse Click (Hardcode Version) 46](#_ewpwya5htuyc)

[1.2 Mouse Click (UnityEvent Version) 46](#_w0l0uqlwnh4u)

[1.3 Mouse Rotator 48](#_m12wxsxqy4ov)

[1.4 Mouse Look 50](#_ykwnjf5m2dks)

[1.5 Mouse Orbit 52](#_hqrk2ynkuivi)

[1.6 Mouse Isometric 54](#_mhrddbp0wq5i)

[1.7 Mouse Toggle 57](#_f5x0t35h6tw)

[1.8 Mouse Slider 59](#_egq9picutbii)

[1.9 Mouse DragDrop 61](#_5jtc6vib33vv)

[1.10 Mouse GrabRelease 63](#_9klpdxb6g2bq)

[2. Keyboard 66](#_323y03jn9pea)

[2.1 Keyboard Press 66](#_2up2yi4uja4u)

[2.2 Keyboard Switch 68](#_yflu5nx4xru9)

[2.3 Keyboard Tab 70](#_5sjyscwswuzr)

[3. Gamepad 71](#_oqtb9d8t5bhs)

[3.1 Gamepad Axis 71](#_kl3bittvbw7s)

[3.2 Gamepad Button 72](#_rz9h8ueu2aou)

[4. Touch 74](#_8nfo6t8rxnp3)

[4.1 Touch Tap 74](#_uci2nwkfv8e1)

[4.2 Touch Look 76](#_a2z8kj5rqlcw)

[4.3 Touch Orbit 77](#_qtoaplnu8wke)

[4.4 Touch Draw 80](#_g66jre3tkx51)

[4.5 Touch DragDrop 82](#_mkc4qjc8auf)

[4.6 Touch GrabRelease 83](#_1mrphsvx8fog)

[4.7 Touch Joystick 85](#_v8evzexc2908)

[5. Gesture 88](#_7e9emrwqf19x)

[5.1 Gesture Swipe 88](#_1w8y9nrtuaqj)

[5.2 Gesture Rotator 90](#_wpgfvkfrey7p)

[5.3 Gesture Pinch 91](#_myun06qy5smf)

[5.4 Gesture Observer 93](#_vqjsw38shzal)

[6. Voice 96](#_vtitydc5f2k2)

[6.1 Voice Recorder 96](#_xsgw68lyywdp)

[6.2 Voice Debugger 97](#_z8waht7i7p4e)

[6.3 Voice Recognition 98](#_10gcxzx7wf62)

[7. Eye 101](#_zg0y9xuvwc0m)

[7.1 Eye Raycast 102](#_2ekngiaaizmy)

[7.2 Eye Pattern 104](#_ugjrohlaety)

[7.3 Eye Tracking 105](#_cdjp1sxfrchg)

[8. Body 107](#_b9lsi4jh36dc)

[8.1 Body Detection 108](#_ymrzxrx4xplk)

[8.2 Body Skeleton 110](#_hy2eby7cyl14)

[9. Image 112](#_836l9lnoti41)

[9.1 Image Orientation 112](#_n7oxk2rditw5)

[9.2 Image Dialog 113](#_fj61h594g72z)

[9.3 Image Camera 115](#_fiqs5k6pmhg3)

[9.4 Image Capture 116](#_wxc0urn1wb8j)

[9.5 Image Carousel 118](#_ii6gfzekwp85)

[10. Sensor 119](#_6dxlrpv7m434)

[10.1 Arduino Reader 120](#_se8375vvp5jc)

[10.2 Arduino Writer 121](#_r0ra6th07k41)

[**Chapter 5 Event Classification 124**](#_sdcn0l7q7wuu)

[1. Calculation 125](#_4e19vd4iipom)

[1.1 Calculation Boolean 125](#_dzt2qe4h246u)

[1.2 Calculation Integer 128](#_z4g6chqqgmt4)

[1.3 Calculation Float 131](#_mj4qc956d6ky)

[1.4 Calculation Score 134](#_v7buxj45pwvm)

[1.5 Calculation Time 137](#_3l9dqvwlkadj)

[1.6 Calculation String 141](#_73caxvcovtgl)

[1.7 Calculation Transform 143](#_s2bzugdz9uwr)

[1.8 Calculation Cookies 145](#_8whv49ghnbf0)

[1.9 Calculation Request 147](#_fm9it8qc337h)

[1.10 Calculation Token 152](#_qkybebq8lkjo)

[2. Activation 155](#_7yklfs99xg8z)

[2.1 Activation Event 155](#_hu2evsg5u31a)

[2.2 Activation Delay 156](#_ltlm78bs4b81)

[2.3 Activation LoadScene 158](#_zd18wrrrsp47)

[2.4 Activation LoadSceneAsync 159](#_9gu3tkz6bzis)

[2.5 Activation LoadSceneManager 162](#_prjcczrt8mbk)

[2.6 Activation Language 163](#_4sor5pvqsbpg)

[2.7 Activation Pause 167](#_fb7953955bms)

[2.8 Activation Quit 168](#_tqrto4fwxv2f)

[2.9 Activation Resolution 169](#_wkumyy22w47r)

[2.10 Activation Timer 171](#_dm631i3cx1w4)

[3. Visualization 172](#_cijemomrv8hf)

[3.1 Visualization Fade 173](#_q6d7hklaep6v)

[3.2 Visualization Leaderboard 176](#_9outr51mgc75)

[3.3 Visualization Score 179](#_1dtq4tcx7rrr)

[3.4 Visualization Parallax 181](#_v11z00vbuhek)

[3.5 Visualization Text 184](#_pkvyykciexfm)

[3.6 Visualization InputField 188](#_fnoia8nfewt8)

[3.7 Visualization Toggle 189](#_1rty8s6dnxan)

[3.8 Visualization Camera Shake 190](#_bee2zdgcemn2)

[3.9 Visualization Headbob 191](#_o8fq2j1a158l)

[4. Interaction 191](#_n5cvw1qnqy1v)

[4.1 Interaction Mouse 191](#_cvxlkdee529)

[4.2 Interaction Keyboard 191](#_ealcdmpw16xj)

[4.3 Interaction Dialog 192](#_ectmcwnr23wh)

[4.4 Interaction Dialog Language 194](#_btjxcqu9f145)

[5. Transformation 197](#_wrzrk8n5jhbz)

[5.1 Transformation Vector 198](#_hamemlk0ddny)

[5.2 Transformation Translate 201](#_k53zfv814i9)

[5.3 Transformation PingPong 204](#_wi041sv21eh)

[5.4 Transformation Waypoint 206](#_jcqivevpmenh)

[5.5 Transformation Direction 208](#_khik84k9a579)

[6. Animation 211](#_m6e4k28d9t7)

[6.1 Animation String 212](#_gvv72mlk66tb)

[6.2 Animation Parameter 213](#_4s19sh7gdvv0)

[6.3 Animation Scheduler 215](#_bv8ejpcvdfjw)

[6.4 Animation Tween 216](#_ijebryz4f1ax)

[7. Instantiation 218](#_w0arvmlbeoqv)

[7.1 Instantiation Prefabs 218](#_4kfqxpknqxdy)

[7.2 Instantiation Resource 220](#_25xngrofv6cq)

[7.3 Instantiation Addressable 222](#_wjhm09oef0eo)

[8. Collision 224](#_y0g1rfshy0rp)

[8.1 Collision 2D 225](#_c9zlnrxq1v12)

[8.2 Collision 3D 229](#_n4avgfibf6r6)

[9. Iteration 233](#_rs056csjctts)

[9.1 Iteration GameObject Name 233](#_d0888iuwza7)

[9.2 Iteration GameObject Tag 235](#_f0xrxkdxwn5g)

[9.3 Iteration Mesh Renderer 236](#_c36lyj1ry3w)

[9.4 Iteration Box Collider 237](#_fuuf42j77a91)

[10. Condition 237](#_w3rmrdmof1ra)

[10.1 Condition Boolean 238](#_xo10njekfsqt)

[10.2 Condition Integer 241](#_gb2hldvynika)

[10.3 Condition Float 245](#_nyv613ugms1d)

[10.4 Condition Score 248](#_979r4z8lfukc)

[10.5 Condition String 250](#_t50zhbxk1nl)

[10.6 Condition Time 253](#_7m4ebnjpbdb5)

[**Chapter 6 Mechanic Classification 257**](#_wj35l9ys2eg0)

[1. Linear Progression 258](#_3mdauyonl53b)

[1.1 Splash Screen 259](#_e2qf8ocnpvzj)

[1.2 Main Menu 259](#_5l33ov2t5fxr)

[1.3 Cutscene Animation 259](#_kdg3udrewgy9)

[1.4 Folks Simulation 260](#_p1dfd8kr23ly)

[1.5 Environment Visualization 261](#_jevsemah2sy9)

[1.6 Cinematic Movie Mode 261](#_voyv4ydun847)

[2. Branching Progression 262](#_tqn6bx1xju3x)

[2.1 Word Map Navigation 262](#_o2gtbsvd0dss)

[2.2 Level Selection 262](#_4qxda01hil6c)

[3. Avatar Controlling 262](#_ogrokuvctbvo)

[3.1 Platformer Controller 264](#_xvc3lvvjeryf)

[a. Cole 264](#_ux7z71xzy3zv)

[b. Zagorskiy 266](#_6i6afheb37wr)

[3.2 Top Down Shooter Controller 267](#_u5cu13uyihug)

[a. Cole 268](#_dahsju8kreae)

[3.3 First Person Controller 270](#_bkv89myxmdmq)

[c. Bon3head 271](#_9uvibnydzxkk)

[d. ChatGPT 275](#_m55g9mf0v0dp)

[e. Cole 278](#_ydq5d5fdwsrc)

[f. Devy 279](#_r1ab2b6alntl)

[g. Vagsether 283](#_crmmwdcn68ec)

[3.4 Third Person Controller 285](#_90ftomvb85az)

[a. Cole 286](#_erqd9sxqj0pd)

[b. Debdas 288](#_dwdc7nxatjju)

[c. Zamojc 289](#_ibf3ctp981gn)

[3.5 Isometric Controller 290](#_y75o4w9vpdi)

[4. Entity Statistic 290](#_uyf79h6f2zgv)

[4.1 Console Debugger 291](#_c0y6a04l2imh)

[4.2 Web Request 291](#_wv4rcxxgbkoe)

[5. Tactical Maneuvering 291](#_ck8th4z7c2i2)

[5.1 Strategic Map Controller 291](#_pty1j1i99884)

[5.2 Gizmo Controller 291](#_tow3gojznckw)

[6. Physics Calculation 291](#_vx7dy7w8vd0h)

[6.1 Gravity System 292](#_2p39npfkgns0)

[6.2 Liquid Simulation 292](#_hovvl5yf87ij)

[7. Gameplay Balancing 292](#_pkx507hm76o3)

[7.1 Configuration System 292](#_3lh24pyi66j)

[7.2 AI Pathfinding 292](#_dxycp5y7kw26)

[7.3 Enemy Patroling 292](#_dramjf6estcc)

[8. Internal Economy 292](#_2onckzf6rnch)

[8.1 Inventory Controller 293](#_nsrpyn7xhlti)

[8.2 Item Mall System 293](#_6xfggy6jvwp6)

[9. Social Interaction 293](#_1gu6qeg0i5gl)

[9.1 Visual Novel Dialog 293](#_hqb040jzh4te)

[9.2 Multiplayer System 293](#_cpk4lvb7vyxt)

[10. User Generated Content 293](#_wrbqotcqpske)

[10.1 Avatar System Creation 294](#_mvrrxspvregz)

[10.2 Item Crafting System 294](#_t4eis212d85q)

[**Chapter 7 Game Goal Typology 295**](#_sgkz0gjtaqhl)

[1. Choose 296](#_p2kw5l9fh6cy)

[2. Configure 297](#_ahayoso30jx)

[3. Solve 297](#_j1an6b4wgwxf)

[4. Create 297](#_agpa7z3pkyqy)

[5. Find 298](#_oy8k38lephjg)

[6. Obtain 298](#_uyieo4fhjlz3)

[7. Optimize 298](#_ninykpg8t5d7)

[8. Reach 298](#_laogcbqxeat7)

[9. Synchronize 299](#_i2j390sxgt7n)

[10. Remove 299](#_wl0uuritrhvl)

[**Chapter 8 Augmented Reality 300**](#_ya27ft46ftqm)

[1. Native AR 301](#_6xh7ajnsg71q)

[2. Vuforia 301](#_5n7qkzdh9it6)

[2.1 Marker Based 302](#_i1g9npxq7azd)

[2.2. Object Based 302](#_mxuxgadg7cnv)

[2.3. Marker Less 303](#_aoqzk2fq5jgq)

[2.4. Gesture Interaction 303](#_ika91n8mtct7)

[2.5. Virtual Button 304](#_8u6uxqqoe3n5)

[2.6. Camera Focus 305](#_vks8xpk90v10)

[2.7. Multi Marker 305](#_xty5suiocet)

[**Chapter 9 Virtual Reality 307**](#_9suignyhsqy)

[1. Google VR 308](#_tnmwealvgr2x)

[2. Occulus VR 308](#_b3f2b7jyzt1k)

[**Chapter 10 Unity Third Party Plugins 309**](#_3pepr076dc1x)

[1. Dynamic Parkour System 310](#_ofeckm6lnhp4)

[2. Microsoft Playfab 310](#_1e67bvtpf9w7)

[2.1 Playfab Anonymous 311](#_6epea3f35kse)

[2.2 Playfab Register 312](#_m2yt9h5piook)

[2.3 Playfab Login 314](#_u3yd2qmaaq2y)

[2.4 Playfab Player 315](#_1iy0d5d4v8xd)

[2.5 Playfab Leaderboard 316](#_4l3cc8vhtc1k)

[2.6 Playfab Data Request 318](#_j3holhfuzi19)

[3. Tobii Eye Tracker 319](#_xynj96vn0wzb)

[4. Leap Motion 320](#_h4iobaxxaq1)

[**Chapter 11 Unity Troubleshooting 321**](#_3074pgsd2v4e)

[**Chapter 12 Special Game Courses 323**](#_pieefoqb6eq4)

[**Game Ideation Index 325**](#_j73rtl9e114w)

[Game Premis 325](#_ivpqfrg43ygp)

[Visual Novel & Cutscene Dialog 330](#_te5jrl4b5m0q)

[Avatar Controlling 330](#_cv0i8pi20b4u)

[Calculation & Internal Economy 331](#_ln4ho4cja7rr)

[Gameplay Balancing & Optimalization 332](#_tjeka0g4fa3b)

[**Game Generation Index 333**](#_303mnww05ke8)

[Text SFX Creator 333](#_q30vmg7jiudk)

[Character Creator 333](#_74ansiwy527c)

[Text Generator 333](#_m1nhebtzyomg)

[Text To Speech Generator 334](#_zf5dt1uz22ai)

[Image Generator 335](#_oj0toovbkmfz)

[Video Generator 335](#_4kebwgp5zdzs)

[**Daftar Pustaka 336**](#_7vi0ll1siels)

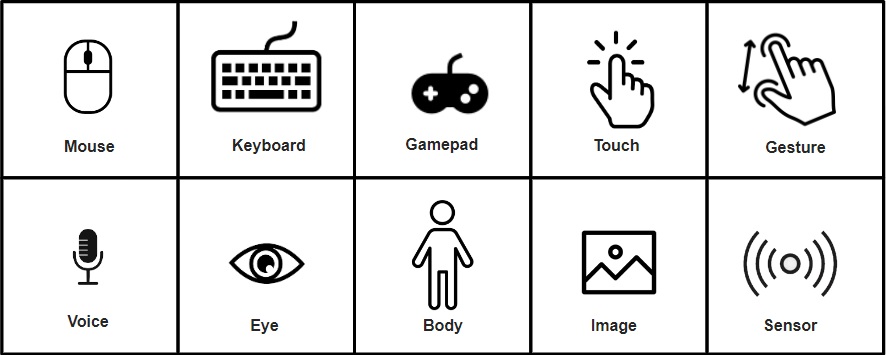
# Classification Model Overview

## Function Classification

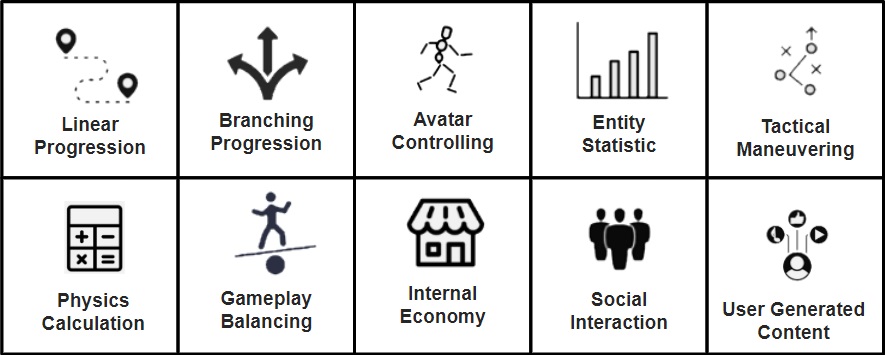
# 

# 

## Input Classification

****

## Event Classification

****

## Game Goal Classification

## 

# 

# 

# Chapter 1 Multimedia Application Overview

Aplikasi multimedia interaktif adalah program perangkat lunak atau platform yang menggabungkan berbagai elemen multimedia, seperti gambar, suara, video, dan teks, dengan fitur interaktif yang memungkinkan pengguna untuk berpartisipasi aktif dalam pengalaman tersebut. Salah satu contoh paling umum dari aplikasi multimedia interaktif adalah permainan video. Dalam permainan video, pemain dapat berinteraksi dengan lingkungan virtual, karakter, dan tantangan dalam permainan. Selain game, aplikasi multimedia interaktif juga memiliki berbagai bidang aplikasi, antara lain:

* **Hiburan**: Selain game, aplikasi multimedia interaktif digunakan dalam industri hiburan untuk membuat pengalaman yang lebih mendalam dan partisipatif. Contoh termasuk aplikasi streaming musik dan video yang memungkinkan pengguna untuk memilih lagu atau film yang mereka inginkan.
* **Pendidikan**: Aplikasi multimedia interaktif digunakan dalam pembelajaran online, modul interaktif, dan e-learning untuk membuat proses pembelajaran lebih menarik dan efektif. Contohnya adalah aplikasi pembelajaran bahasa yang memungkinkan pengguna untuk berlatih berbicara dan mendengar dengan berbagai konten multimedia.
* **Kesehatan**: Dalam bidang kesehatan, aplikasi multimedia interaktif digunakan untuk pelatihan medis, simulasi bedah, dan pemantauan kesehatan. Misalnya, simulasi bedah yang memungkinkan dokter untuk berlatih prosedur bedah sebelum melakukan operasi sebenarnya.
* **Pariwisata**: Aplikasi multimedia interaktif digunakan dalam industri pariwisata untuk memberikan pengalaman wisata yang lebih menarik dan informatif. Contohnya adalah aplikasi pemandu wisata virtual yang memberikan informasi tentang tempat-tempat wisata dengan dukungan gambar dan video.
* **Transportasi**: Dalam industri transportasi, aplikasi multimedia interaktif digunakan untuk perencanaan perjalanan, simulasi pengemudi, dan pengembangan kendaraan otonom. Contohnya adalah aplikasi navigasi yang memberikan petunjuk visual dan suara untuk membantu pengemudi mencapai tujuan mereka, atau simulator penerbangan yang digunakan dalam pelatihan pilot.
* **Arsitektur**: Dalam arsitektur, aplikasi multimedia interaktif digunakan untuk visualisasi desain bangunan, model 3D interaktif, dan simulasi lingkungan. Arsitek dapat menggunakan aplikasi ini untuk memberikan klien pandangan yang lebih baik tentang bagaimana proyek bangunan akan terlihat dan berfungsi. Selain itu, aplikasi ini juga dapat digunakan dalam perencanaan tata kota dan desain ruang publik.
* **Militer**: Dalam militer, aplikasi multimedia interaktif digunakan untuk pelatihan tentara, simulasi perang, dan pengembangan strategi. Contohnya adalah simulator tempur yang memungkinkan pilot untuk berlatih dalam kondisi yang mirip dengan situasi pertempuran sebenarnya, atau aplikasi pemodelan perangkat keras yang membantu dalam perencanaan operasi militer.

Aplikasi multimedia interaktif memiliki potensi besar dalam berbagai bidang untuk meningkatkan pengalaman pengguna dan memberikan manfaat yang lebih besar. Dengan teknologi yang terus berkembang, kita dapat mengharapkan lebih banyak inovasi dalam pengembangan aplikasi multimedia interaktif di masa depan.Dalam pengembangan aplikasi multimedia dan game, terdapat lima bagian utama yang membentuk struktur permainan yang koheren dan menarik bagi pengguna yaitu: Introduction, Navigation, Exploration, Mission dan Evaluation.

|  | * **Introduction (Pengenalan):** Bagian ini adalah awal dari permainan atau aplikasi, yang dirancang untuk memperkenalkan pemain atau pengguna pada dunia yang akan mereka eksplorasi. Biasanya, pengenalan mencakup narasi, karakter, tujuan, dan aturan dasar permainan. Ini adalah momen pertama yang akan menciptakan ketertarikan awal dan memberikan gambaran tentang apa yang akan dihadapi. * **Navigation (Navigasi):** Navigasi berfokus pada cara pemain atau pengguna berinteraksi dengan lingkungan dalam permainan atau aplikasi. Ini termasuk kontrol pergerakan, interaksi dengan objek, dan navigasi antar level atau bagian. Pengembang harus memastikan navigasi yang lancar dan intuitif untuk memungkinkan pemain atau pengguna menjelajahi dunia yang telah dibuat dengan mudah. * **Exploration (Eksplorasi):** Bagian eksplorasi memberikan pemain atau pengguna kesempatan untuk mengeksplorasi dunia yang telah dibangun lebih dalam. Mereka dapat menemukan rahasia, mengumpulkan barang, atau berinteraksi dengan karakter lain. Eksplorasi menciptakan rasa penasaran dan menjaga tingkat keterlibatan pengguna selama permainan. * **Mission (Misi):** Misi adalah bagian di mana pemain atau pengguna diberikan tugas atau tujuan yang harus dicapai. Ini bisa berupa pencapaian tertentu, pertarungan melawan musuh, atau penyelesaian teka-teki. Misi memberikan struktur pada permainan atau aplikasi, dan kesuksesan dalam menyelesaikannya dapat memberikan pemain atau pengguna rasa pencapaian dan motivasi. * **Evaluation (Evaluasi):** Bagian evaluasi muncul setelah pemain atau pengguna menyelesaikan misi atau mencapai akhir permainan. Ini memberikan umpan balik tentang kinerja mereka, termasuk poin, penghargaan, atau peringkat. Evaluasi membantu dalam memberikan pemahaman tentang sejauh mana pemain telah berhasil dan dapat memicu mereka untuk bermain lagi atau meningkatkan kinerja mereka. |
| --- | --- |

Dengan merancang aplikasi multimedia dan game dengan mempertimbangkan kelima bagian ini, pengembang dapat menciptakan pengalaman yang lebih menarik dan terstruktur bagi pengguna mereka. Setiap elemen berperan penting dalam menggabungkan hiburan, tantangan, dan eksplorasi dalam satu kesatuan yang memikat.

## 1. Introduction

Introduction (Pengenalan) adalah bagian awal dari permainan atau aplikasi multimedia yang bertujuan untuk memberikan pemain atau pengguna pemahaman dasar tentang dunia permainan, karakter, dan tujuan utama. Ini adalah momen pertama di mana pengembang memiliki kesempatan untuk menarik perhatian dan menciptakan ikatan emosional dengan pengguna. Ada beberapa elemen yang biasanya terkait dengan pengenalan:

* **Splash Screen** (Publisher << Telkom University/ Developer Screen << Biasanya logo tim anda): Untuk pembukaan aplikasi di awal.
* **Main Menu** (Menu Utama): Main menu adalah titik awal tempat pemain dapat memilih opsi seperti memulai permainan, mengakses pengaturan, atau melanjutkan permainan yang sudah ada. Pengembang sering memasukkan gambaran singkat tentang cerita atau karakter dalam latar belakang main menu, seperti yang terlihat dalam "The Last of Us," yang menampilkan gambar Joel dan Ellie dalam suasana pasca-apokaliptik sebagai latar belakang menu.
* **Opening Cutscene** (Adegan Awal): Banyak permainan dan aplikasi multimedia dimulai dengan adegan pembukaan yang menjelaskan latar belakang cerita atau konteks permainan. Contohnya, dalam permainan RPG seperti "The Elder Scrolls V: Skyrim," opening cutscene menjelaskan tentang dunia fantasi dan peran pemain sebagai pahlawan yang dipilih.
* **Narrative Introduction** (Pengenalan Naratif): Beberapa permainan menggunakan narasi atau dialog awal untuk menggambarkan dunia permainan atau karakter utama. Contohnya, dalam "Bioshock Infinite," narasi awal oleh Booker DeWitt memberikan konteks tentang dunia Columbia.
* **Cinematic Introductions** (Pengenalan Sinematik): Beberapa permainan, seperti "Metal Gear Solid" atau "Final Fantasy" series, sering memulai permainan dengan adegan sinematik dramatis yang memperkenalkan karakter utama, antagonis, atau elemen penting dalam cerita.
* **Tutorial** (Panduan): Bagian ini sering kali digunakan untuk mengenalkan pemain pada dasar-dasar permainan, seperti kontrol, mekanika dasar, atau tujuan utama. Misalnya, dalam "The Legend of Zelda: Breath of the Wild," pemain diberikan panduan awal tentang cara bergerak, menggunakan senjata, dan bertahan hidup di dunia yang luas.

Pengenalan yang baik adalah kunci untuk membuat pemain merasa terlibat dalam permainan atau aplikasi multimedia. Ini membantu menciptakan ikatan awal antara pengguna dan dunia yang telah dibangun, memotivasi mereka untuk menjelajahi lebih dalam, dan memberikan pemahaman dasar yang diperlukan untuk memainkan permainan atau menggunakan aplikasi dengan baik. Pengembangan struktur Introduction bisa menggunakan mekanik [**Linear Progression**](#_3mdauyonl53b).

## 2. Navigation

Navigation (Navigasi) adalah elemen penting dalam permainan atau aplikasi multimedia yang memungkinkan pemain atau pengguna berinteraksi dengan lingkungan, mengontrol karakter, atau memilih tingkat permainan. Ini mencakup berbagai aspek navigasi dalam konteks permainan:

* **Character Selection** (Pemilihan Karakter): Fitur ini memungkinkan pemain untuk memilih karakter yang akan mereka mainkan dalam permainan. Contoh yang baik adalah dalam permainan "Super Smash Bros," di mana pemain dapat memilih dari berbagai karakter ikonik dari berbagai franchise game.
* **Level Selection** (Pemilihan Tingkat): Ini memungkinkan pemain untuk memilih tingkat atau misi yang ingin mereka mainkan dalam permainan. Contoh dari ini dapat ditemukan dalam permainan seperti "Angry Birds," di mana pemain dapat memilih level yang berbeda dengan tantangan yang berbeda pula.
* **Map Exploration** (Eksplorasi Peta): Beberapa permainan memiliki peta yang memungkinkan pemain untuk menjelajahi dunia permainan dan menemukan lokasi baru. Sebagai contoh, dalam "The Legend of Zelda: Breath of the Wild," pemain dapat menggunakan peta untuk mengeksplorasi Kerajaan Hyrule yang luas.
* **Menu Navigation** (Navigasi Menu): Ini berkaitan dengan bagaimana pemain berinteraksi dengan menu dalam permainan untuk mengakses opsi seperti inventaris, pengaturan, atau pengisian ulang kesehatan. Sebagai contoh, dalam "The Witcher 3: Wild Hunt," pemain menggunakan menu untuk mengakses inventaris, mengubah senjata, atau memilih quest.
* **In-Game Navigation** (Navigasi dalam Permainan): Ini berkaitan dengan bagaimana karakter atau pemain bergerak dan berinteraksi dengan dunia dalam permainan. Contoh termasuk kontrol karakter di "Grand Theft Auto" atau "Assassin's Creed" untuk menjelajahi kota besar dengan bebas.
* **Dialog Choices** (Pilihan Dialog): Dalam permainan yang memasukkan unsur naratif, pemain sering dihadapkan pada pilihan dialog yang memengaruhi perkembangan cerita. Contoh ini dapat ditemukan dalam permainan seperti "Mass Effect," di mana pemain dapat memilih respons yang akan diucapkan oleh karakter utama.

Navigasi yang baik dalam permainan atau aplikasi multimedia sangat penting karena dapat mempengaruhi pengalaman pengguna. Kontrol yang mudah dipahami, antarmuka yang intuitif, dan navigasi yang lancar dapat membuat pemain merasa lebih terlibat dalam permainan dan meningkatkan tingkat kenyamanan mereka. Sebaliknya, navigasi yang rumit atau membingungkan dapat mengganggu pengalaman dan mengurangi daya tarik permainan. Pengembangan struktur Navigation bisa menggunakan mekanik [**Branching Progression**](#_tqn6bx1xju3x).

## 3. Exploration

Exploration (Eksplorasi) adalah elemen penting dalam permainan dan aplikasi multimedia yang memberikan pemain atau pengguna kesempatan untuk mengeksplorasi dunia yang telah dibangun. Eksplorasi menciptakan rasa penasaran, memungkinkan pemain untuk menemukan rahasia, mendapatkan pengalaman tambahan, dan merasakan kebebasan dalam interaksi dengan lingkungan. Berikut adalah penjelasan lebih detail tentang eksplorasi, beserta contohnya:

* **Point To Click (Klik dan Pilih):** Eksplorasi paling sederhana adalah dengan mengklik objek, suatu lokasi atau antarmuka di layar, dan mengklik objek-objek yang terdapat di dalam aplikasi. menggunakan alat input seperti mouse atau layar sentuh. Gaya eksplorasi ini memungkinkan pemain untuk mengikuti narasi dengan lebih santai dan intuitif.
* **Avatar Controlling** (Kontrol Avatar): Eksplorasi sering kali melibatkan pemain yang mengendalikan karakter atau avatar dalam dunia permainan. Pemain memiliki kendali atas gerakan, tindakan, dan interaksi karakter tersebut. Sebagai contoh, dalam permainan "The Legend of Zelda: Ocarina of Time," pemain mengendalikan Link untuk menjelajahi Hyrule, menyelesaikan teka-teki, dan menghadapi musuh.
* **Open World** (Dunia Terbuka): Konsep dunia terbuka memungkinkan pemain untuk menjelajahi lingkungan yang luas dan terbuka tanpa terlalu banyak batasan. Contoh populer dari permainan dunia terbuka adalah "Grand Theft Auto V," di mana pemain dapat menjelajahi kota Los Santos, melakukan misi, atau hanya berkeliaran bebas.
* **Side Quests** (Misi Sampingan): Selain misi utama, eksplorasi dapat mengungkapkan misi sampingan atau tantangan tambahan. Pemain dapat menemui karakter non-playable (NPC) yang meminta bantuan atau menawarkan hadiah. Misalnya, dalam "The Witcher 3: Wild Hunt," pemain dapat menemukan berbagai misi sampingan saat menjelajahi dunia permainan.
* **Collectibles** (Koleksi): Eksplorasi juga sering terkait dengan pencarian koleksi, seperti item tersembunyi, senjata legendaris, atau artefak berharga. Dalam permainan seperti "Assassin's Creed," pemain dapat mengumpulkan bendera, harta karun, atau artefak sejarah.
* **Hidden Areas** (Area Tersembunyi): Dunia permainan seringkali menyimpan rahasia dan area tersembunyi yang hanya dapat ditemukan melalui eksplorasi yang teliti. Contoh ini dapat ditemukan dalam "The Elder Scrolls V: Skyrim," di mana pemain dapat menemukan gua-gua tersembunyi, perkemahan, atau kuil yang berisi hadiah atau tantangan tambahan.
* **Environmental Storytelling** (Penceritaan Lingkungan): Eksplorasi dapat mengungkapkan cerita dan latar belakang dunia permainan melalui detail lingkungan, seperti tulisan di dinding, catatan, atau barang-barang terbengkalai. Misalnya, dalam "BioShock," pemain dapat memahami sejarah kota Rapture melalui pesan dan catatan yang tersebar di seluruh kota.

Eksplorasi adalah elemen yang penting dalam menciptakan kedalaman, kebebasan, dan daya tarik dalam permainan atau aplikasi multimedia. Ini memungkinkan pemain untuk merasakan dunia dalam permainan dan membuat pengalaman lebih mendalam dan memuaskan. Pengembangan struktur Exploration bisa menggunakan mekanik [**Avatar Controlling**](#_ogrokuvctbvo) dan [**Social Interaction**](#_1gu6qeg0i5gl).

## 4. Mission

Fase "Mission" dalam permainan adalah salah satu elemen penting dalam pengembangan permainan yang mengadopsi 10 tipe tujuan permainan yang beragam. Dalam fase ini, pemain akan dihadapkan pada tugas utama atau tujuan yang sesuai dengan salah satu dari 10 tipe tujuan tersebut: Choose, Configure, Solve, Create, Optimize, Obtain, Find, Synchronize, Reach, dan Remove.

* **Tipe "Choose"** pemain mungkin dihadapkan pada pilihan krusial yang akan memengaruhi alur cerita atau perkembangan karakter mereka. Mereka harus membuat keputusan yang cerdik untuk memenangkan misi.
* **Tipe "Configure"** dapat mengharuskan pemain untuk mengatur peralatan, sumber daya, atau karakter mereka dengan benar sebelum memulai misi, memastikan bahwa mereka siap untuk menghadapi tantangan yang akan datang.
* **Tipe "Solve"** dapat menghadirkan teka-teki atau tantangan intelektual yang perlu diselesaikan pemain untuk melanjutkan. Ini mungkin melibatkan pemecahan kode, teka-teki fisika, atau pemahaman terhadap alur cerita.
* **Tipe "Create"** mungkin meminta pemain untuk mendesain atau membangun sesuatu yang penting untuk progres dalam permainan, seperti membangun senjata atau mengembangkan rencana.
* **Tipe "Optimize"** pemain harus mengambil keputusan yang cerdas untuk memaksimalkan efisiensi atau efektivitas mereka dalam mencapai tujuan misi.
* **Tipe "Obtain"** mungkin mengharuskan pemain untuk mencari dan mengumpulkan barang-barang penting, seperti item kunci atau sumber daya yang langka.
* **Tipe "Find"** pemain perlu mencari lokasi atau objek yang mungkin tersembunyi atau sulit ditemukan dalam permainan.
* **Tipe "Synchronize"** bisa berarti pemain harus berkoordinasi dengan karakter lain atau elemen dalam permainan untuk menyelesaikan misi.
* **Tipe "Reach"** pemain harus mencapai tujuan tertentu, mungkin dengan menjelajahi wilayah yang luas atau mencapai lokasi yang terpencil.
* **Tipe "Remove"** pemain mungkin dihadapkan pada tugas menghilangkan ancaman atau menghancurkan musuh yang menghalangi kemajuan mereka dalam permainan.

Mission yang dibangun dengan beragam tipe tujuan permainan ini dapat menambahkan tingkat kompleksitas, tantangan, dan daya tarik dalam permainan, membuat pengalaman bermain lebih beragam dan memuaskan bagi pemain. Pengembangan struktur Mission bisa menggunakan mekanik [**Tactical Manuevering**](#_ck8th4z7c2i2), [**Physics Calculation**](#_vx7dy7w8vd0h) dan [**Gameplay Balancing**](#_pkx507hm76o3).

## 5. Evaluation

Evaluation (Evaluasi) adalah tahap penting dalam permainan dan aplikasi yang memungkinkan pemain atau pengguna untuk mengevaluasi kinerja mereka, melihat hasil, atau mendapatkan umpan balik terkait dengan pencapaian mereka. Evaluasi dapat memberikan penggunaan berbagai fitur, seperti umpan balik visual, dasbor, atau laporan yang memberikan wawasan tentang prestasi. Berikut adalah penjelasan lebih rinci tentang evaluasi beserta contohnya:

* **Feedback Ending** (Akhir Umpan Balik): Umpan balik adalah cara utama untuk memberi pemain atau pengguna informasi tentang kinerja mereka dalam permainan atau aplikasi. Ini dapat berupa pesan teks, suara, atau visual yang memberi tahu pemain tentang apa yang telah dicapai atau diabaikan. Contoh: Dalam permainan tembak-menembak seperti "Counter-Strike," pemain menerima umpan balik tentang berapa banyak musuh yang telah mereka eliminasi dalam sesi permainan atau seberapa baik akurasi tembakan mereka.
* **Leaderboards** (Papan Peringkat): Leaderboards adalah cara untuk membandingkan prestasi pemain dengan pemain lainnya dalam permainan atau aplikasi. Ini memberikan motivasi tambahan kepada pemain untuk bersaing dan meningkatkan kinerja mereka. Contoh: Dalam permainan mobile populer "Candy Crush Saga," pemain dapat melihat peringkat mereka dan membandingkannya dengan teman-teman mereka yang juga bermain.
* **Progress Reports** (Laporan Kemajuan): Laporan kemajuan adalah dokumen yang memberikan ringkasan tentang kemajuan pemain atau pengguna dalam mencapai tujuan tertentu. Ini dapat digunakan dalam pendidikan, pelatihan, atau permainan.Contoh: Dalam aplikasi belajar bahasa seperti Duolingo, pengguna dapat melihat laporan kemajuan mereka dalam memahami dan menguasai kosa kata dan tata bahasa.
* **User Dashboard** (Dasbor Pengguna): Dasbor pengguna adalah fitur yang memberikan gambaran lengkap tentang pencapaian, statistik, dan kemajuan pemain atau pengguna. Ini dapat mencakup informasi seperti skor, level yang sudah dicapai, prestasi yang diperoleh, atau statistik bermain.Contoh: Dalam permainan "Overwatch," pemain memiliki dasbor yang memberikan informasi tentang statistik bermain mereka, seperti jumlah kematian, jumlah "kill," dan tingkat keberhasilan karakter tertentu.
* **Educational Web Analytics** (Analisis Web Pendidikan): Dalam konteks pendidikan online atau aplikasi edukasi, analisis web adalah cara untuk melihat seberapa baik pengguna belajar dan mengevaluasi kemajuan mereka. Ini dapat mencakup laporan tentang seberapa sering materi dikunjungi, tingkat keberhasilan pada kuis, atau kemajuan dalam modul pembelajaran.Contoh: Sebuah platform e-learning dapat memberikan laporan kepada guru atau administrator yang mencakup seberapa sering siswa mengakses materi pembelajaran, berapa banyak tugas yang telah mereka selesaikan, atau seberapa baik mereka memahami konsep-konsep tertentu.

Evaluasi adalah alat penting untuk memberikan pemain atau pengguna umpan balik tentang kinerja mereka, membantu mereka melihat seberapa baik mereka telah mencapai tujuan, dan memberikan dorongan tambahan untuk terus berpartisipasi atau meningkatkan diri. Ini juga dapat memberikan informasi berharga kepada pengembang atau pendidik untuk memahami bagaimana pemain atau pengguna berinteraksi dengan permainan atau aplikasi mereka. Pengembangan struktur Evaluation bisa menggunakan mekanik [**Entity Statistic**](#_uyf79h6f2zgv)**,** [**Internal Economy**](#_2onckzf6rnch)**,** dan [**User Generated Content**](#_wrbqotcqpske)**.**

## 

## 6. Flow Variation

Berikut ini beberapa contoh dari variasi flow aplikasi multimedia interaktif yang merupkan kombinasi dari Introduction, Navigation, Exploration, Mission dan Evaluation.

| **Basic Flow** | **Deskripsi** |
| --- | --- |
|  | **Introduction**  Aplikasi multimedia interaktif biasanya dimulai dengan menampilkan splash screen dari publisher dan developer.  **Navigation**  Setelah itu, pemain akan dibawa ke layar yang terdiri dari main menu, yang memungkinkan akses ke setting dan informasi mengenai permainan (about).  **Exploration**  Selanjutnya pemain dapat memasuki fase, di mana bentuknya dapat bervariasi antara point-and-click, avatar controlling atau keduanya untuk menjelajahi dunia permainan.  **Mission**  Kemudian, permainan dapat mengarahkan pemain ke berbagai yang dapat berupa Choose, Reach, atau Find, yang memerlukan pemain untuk membuat keputusan, mencapai tujuan tertentu, atau mencari objek penting.  **Evaluation**  Akhirnya, pengalaman bermain ini ditutup dengan skor yang mencerminkan prestasi pemain sepanjang permainan. |

# 

# 

# 

# 

# Chapter 2 Programming Codification

Pemrograman komputer seringkali dapat dianggap sebagai bentuk bahasa tersendiri yang memiliki analogi dengan cara kita membentuk bahasa Indonesia, baik dalam bentuk struktur formula pembentuk kalimat (seperti SPOK), paragraf, maupun dialog. Seperti SPOK dalam bahasa Indonesia, dalam pemrograman kita memiliki "subjek" yang mewakili data atau variabel, "predikat" yang menggambarkan tindakan atau operasi yang dilakukan pada data tersebut, "objek" yang merujuk pada parameter atau input yang diperlukan untuk tindakan tersebut, dan "keterangan" yang memberikan konteks atau informasi tambahan.

Selain itu, dalam pemrograman kita juga dapat membuat "paragraf" yang terdiri dari serangkaian pernyataan atau instruksi yang membentuk alur logika dalam program. Ini mirip dengan cara kita menyusun beberapa kalimat menjadi paragraf untuk menjelaskan konsep atau ide yang lebih besar.Terakhir, dalam pemrograman, kita sering berinteraksi dengan "dialog" antara program dan pengguna.

Dalam analogi ini, pengguna bertindak sebagai pihak yang memberikan input atau pertanyaan (subjek), program adalah entitas yang menjalankan tindakan atau operasi (predikat) berdasarkan input pengguna, dan pesan keluaran atau respons program adalah "keterangan" yang diberikan kepada pengguna.Dengan memahami analogi ini, kita dapat menggambarkan pemrograman sebagai bentuk bahasa yang kompleks dan struktural, yang memungkinkan kita untuk berkomunikasi dengan komputer dan menciptakan perangkat lunak yang dapat menjalankan tugas-tugas yang beragam.

## 1. Pola SPOK

Pola kalimat dalam bahasa Indonesia sering kali mengikuti pola SPOK, yaitu Subjek-Predikat-Objek-Keterangan. Ini adalah struktur dasar dalam pembentukan kalimat dalam bahasa Indonesia. Berikut beberapa contoh pola kalimat SPOK:

* Saya (Subjek) makan (Predikat) nasi (Objek) di rumah (Keterangan).
* Anak-anak (Subjek) bermain (Predikat) bola (Objek) dengan gembira (Keterangan).
* Ibu (Subjek) memasak (Predikat) masakan lezat (Objek) untuk keluarga (Keterangan).
* Mereka (Subjek) membaca (Predikat) buku-buku (Objek) di perpustakaan (Keterangan).
* Dia (Subjek) menggambar (Predikat) lukisan indah (Objek) di taman (Keterangan).

Pola SPOK ini memberikan struktur yang jelas dalam kalimat Bahasa Indonesia, sehingga pesan atau informasi dapat disampaikan dengan lebih efektif. Subjek adalah pihak yang melakukan tindakan, predikat adalah tindakan itu sendiri, objek adalah pihak yang menerima tindakan, dan keterangan memberikan informasi tambahan tentang tindakan tersebut. Dengan memahami pola ini, penutur bahasa Indonesia dapat menyusun kalimat yang jelas dan komunikatif.

Dalam pemrograman C#, analogi pola kalimat SPOK dalam bahasa Indonesia dapat diterapkan dengan struktur dasar program sebagai berikut:

* Subjek (S) dalam bahasa C# adalah variabel yang melakukan tindakan.
* Predikat (P) dalam bahasa C# adalah metode atau fungsi yang menjalankan tindakan.
* Objek (O) dalam bahasa C# adalah argumen atau parameter yang digunakan oleh metode atau fungsi.
* Keterangan (K) dalam bahasa C# adalah berbagai jenis statement atau pernyataan tambahan yang menggambarkan tindakan yang sedang dilakukan.

Contoh dalam pemrograman C#:

**Menghitung Luas Segituga**

| double alas = 5.0; // Subjek  double tinggi = 7.0; // Subjek  double luas = HitungLuasSegitiga(alas, tinggi); // Predikat dengan Objek  Console.WriteLine("Luas segitiga dengan alas {0} dan tinggi {1} adalah {2}", alas, tinggi, luas); // Keterangan |
| --- |

Dalam contoh ini, alas dan tinggi adalah subjek yang digunakan dalam perhitungan luas segitiga. Predikatnya adalah metode HitungLuasSegitiga, yang menerima dua objek (alas dan tinggi) untuk menghitung luas. Pernyataan keterangan mencetak hasilnya.

**Manipulasi String**

| string kata = "Halo, "; // Subjek  string tambahan = "selamat datang!"; // Subjek  string kalimat = GabungkanString(kata, tambahan); // Predikat dengan Objek  Console.WriteLine(kalimat); // Keterangan |
| --- |

Dalam contoh ini, kata dan tambahan adalah subjek yang akan digabungkan. Predikatnya adalah metode GabungkanString, yang menerima dua objek (kata dan tambahan) untuk menghasilkan kalimat baru. Pernyataan keterangan mencetak hasil penggabungan string.

## 2. Pola Dialog

Pola pembuatan dialog dalam bahasa Indonesia memiliki aturan khusus yang berkaitan dengan penggunaan tanda baca. Dalam sebuah dialog, tanda kutip (" ") digunakan untuk mengapit ucapan karakter yang sedang berbicara. Tanda koma (,) dan titik (.) digunakan untuk memisahkan ucapan karakter dari tagar ucapan, atau sebagai penghubung antara kata-kata dalam dialog. Contoh-contoh di bawah ini menggambarkan penggunaan tanda baca dalam pembuatan dialog:

* **Dialog Dasar:**

John: "Halo, apa kabar?"

Dalam contoh ini, titik dua digunakan untuk memisahkan ucapan John dari tagar ucapan.

* **Dialog Sederhana:**

"Halo, apa kabar?" kata John.

Dalam contoh ini, tanda kutip digunakan untuk memisahkan ucapan John dari tagar ucapan. Tanda koma menghubungkan antara ucapan John dan tagar ucapan.

* **Dialog dengan Tagar Ucapan yang Panjang:**

"Saya sangat senang dapat bertemu dengan Anda hari ini," ujar Lisa, "saya telah menunggu momen ini sejak lama."

Di sini, tanda kutip menandai awal dan akhir ucapan Lisa, dengan tanda koma menghubungkan tagar ucapan yang panjang.

* **Dialog dengan Ucapan Ganda:**

"Kemarin," kata Anna, "saya pergi ke bioskop dan menonton film 'Interstellar'."

Dalam kasus ini, ada tanda kutip di dalam ucapan Anna yang mengapit judul film "Interstellar." Ini menunjukkan bahwa itu adalah kutipan dalam kutipan.

* **Dialog dengan Tanda Baca Lainnya:**

"Maaf," seru Tom, "saya akan segera tiba! Tunggu sebentar..."

Dalam contoh ini, tanda kutip mengapit ucapan Tom, dan tanda seru (!) dan elipsis (...) digunakan untuk mengekspresikan perasaan dan jeda dalam dialog.

Pola dan penggunaan tanda baca dalam pembuatan dialog dalam bahasa Indonesia penting untuk memahami siapa yang berbicara, apa yang mereka katakan, dan bagaimana mereka mengungkapkan perasaan atau maksud mereka. Ini membantu pembaca atau pendengar untuk lebih baik mengikuti alur cerita dan karakter dalam narasi. Dalam pemrograman C#, Anda dapat menggunakan pola dialog yang sama seperti bahasa Indonesia. Ini adalah contoh analogi pola dialog dalam bahasa Indonesia dalam pemrograman C#:

| string ucapanJohn = "Halo, kata John, apa kabar?"; |
| --- |

Dalam contoh di atas, karakter "" sebelum tanda kutip ganda ("") menghindari karakter tersebut dianggap sebagai penutup string dan sebaliknya, sehingga Anda dapat menyisipkan tanda kutip ganda di dalam string.

| int hasil = Penjumlahan(5, 3); // Menggunakan "()" untuk memanggil fungsi Penjumlahan |
| --- |

Dalam contoh ini, "()" digunakan untuk memanggil fungsi Penjumlahan dengan memberikan dua angka sebagai parameter. Fungsi ini akan menjalankan operasi penjumlahan dan mengembalikan hasilnya.

| if (nilai > 50)  {  Console.WriteLine("Nilai lebih besar dari 50.");  }  else  {  Console.WriteLine("Nilai kurang dari atau sama dengan 50.");  } |
| --- |

Dalam contoh ini, "{}" digunakan untuk menentukan blok kode yang akan dieksekusi tergantung pada hasil percabangan if-else. Jadi, "{}" digunakan untuk mengelompokkan serangkaian pernyataan yang akan dijalankan bersamaan dalam sebuah blok kode.

## 

## 3. Pola Referensi

Mengutip suatu referensi adalah praktik penting dalam penulisan akademik dan penelitian yang membantu penulis untuk memberikan kredit kepada sumber informasi yang digunakan dan memvalidasi klaim atau argumen yang mereka sampaikan. Teknik mengutip dapat berbeda-beda tergantung pada gaya penulisan yang digunakan, seperti APA, MLA, atau Chicago. Biasanya, dalam teks, penulis akan mencantumkan nama penulis atau penelitian yang dikutip, tahun publikasi, dan halaman atau paragraf spesifik yang relevan dari sumber tersebut. Misalnya, dalam format APA, sebuah kalimat yang mengutip sumber akan terlihat seperti ini:

"Menurut Smith (2020), perubahan iklim telah memiliki dampak signifikan pada ekosistem laut (hal. 45)."

**Daftar Pustaka:**

Smith, J. (2020). Climate Change and Marine Ecosystems. Academic Press.

Dalam daftar pustaka ini, kita mencantumkan penulis (Smith), judul artikel (Climate Change and Marine Ecosystems), dan penerbit (Academic Press) beserta tahun publikasi (2020). Ini memberikan pembaca informasi yang diperlukan untuk melacak sumber tersebut. Mengutip referensi dengan benar adalah bagian penting dari praktik penulisan yang akurat dan etis, serta membantu membangun kepercayaan pada karya penulis.

Analogi dalam konteks penggunaan function atau procedure dalam pemrograman adalah mirip dengan mengutip sumber dalam penulisan akademik. Ketika sebuah function atau procedure dari satu script dipanggil di script lain, penting untuk memberikan kredit kepada script atau modul asal yang menyediakan fungsi tersebut. Hal ini membantu dalam pemahaman dan pelacakan asal-usul kode tersebut.

Contoh dengan menggunakan pola referensi dalam pemrograman:

**Main Program**

| using System;  using **Calculator**;  class Program  {  static void Main()  {  double result = **Divide**(10.0, 2.0); // Memanggil function dari script A  Console.WriteLine("Hasil pembagian adalah: " + result);  }  } |
| --- |

**Referensi Class**

| using System;  public class **Calculator**  {  public static double **Divide**(double a, double b)  {  if (b == 0)  {  throw new DivideByZeroException("Denominator cannot be zero.");  }  return a / b;  }  } |
| --- |

Penting untuk diingat bahwa bahasa pemrograman yang efektif seringkali mengadopsi prinsip-prinsip bahasa dialog sehari-hari untuk membuat kode lebih mudah dipahami. Dengan mengikuti pola bahasa yang akrab, pengembang dapat mengekspresikan ide dan logika mereka dengan lebih jelas dan kohesif.

Kode yang terstruktur dengan baik, seperti sebuah dialog yang baik, memungkinkan kolaborasi yang lebih baik antara pengembang, mempercepat pemahaman dan pemeliharaan kode, dan memungkinkan transfer pengetahuan yang efektif di antara anggota tim pengembangan. Oleh karena itu, penggunaan prinsip-prinsip bahasa dialog sehari-hari dalam pemrograman adalah salah satu kunci untuk menciptakan perangkat lunak yang kuat dan dapat dipelihara.

# 

# 

# 

# 

# 

# Chapter 3 Unity Function Classification

Unity Integrated Development Environment (IDE) adalah platform utama tempat Anda akan mengembangkan dan membentuk permainan Anda. Ini adalah ruang di mana ide-ide Anda menjadi tindakan konkret, menggabungkan aspek kreatif dan teknis dari pembuatan game. Unity IDE menyediakan antarmuka yang terorganisir dengan baik, serta berbagai alat yang mendukung Anda dalam merancang dunia permainan, membuat skrip, mengatur sumber daya, dan mengujinya. Di dalam Unity IDE, Anda akan memiliki akses ke berbagai panel dan jendela yang membantu Anda mengelola setiap aspek proyek, dari tampilan visual hingga logika gameplay.

Dalam proses pembuatan permainan, Unity IDE memungkinkan Anda untuk merancang dunia virtual, mengatur perilaku objek, serta membuat interaksi antara elemen-elemen dalam permainan. Anda akan berinteraksi dengan panel Inspektor yang memungkinkan Anda untuk menyesuaikan properti objek dengan mudah, sambil melihat perubahan yang terjadi secara real-time. Selain itu, IDE ini juga menyediakan layanan untuk mengimpor, mengelola, dan mengoptimalkan aset-aset seperti model 3D, tekstur, suara, dan lainnya. Dengan editor visual yang kuat, Anda dapat membangun tampilan permainan Anda dengan drag-and-drop, dan mengaitkan perilaku dengan kode menggunakan bahasa pemrograman C#.

Unity IDE adalah tempat di mana kreativitas dan teknologi saling berpadu. Setiap proyek dimulai dari ide sederhana, dan Unity IDE memberikan Anda alat untuk mengembangkan ide tersebut menjadi sebuah permainan yang berjalan. Di sini, Anda akan belajar bagaimana mengatur, mengedit, dan mengelola elemen-elemen permainan Anda, serta mengembangkan keterampilan dalam pengkodean dan merancang pengalaman bermain yang menarik. Dalam perjalanan ini, Anda akan menemukan betapa kuatnya fungsi yang tersedia di Unity IDE dan bagaimana Anda dapat menggunakan alat ini untuk mewujudkan visi kreatif Anda dalam bentuk permainan yang unik.

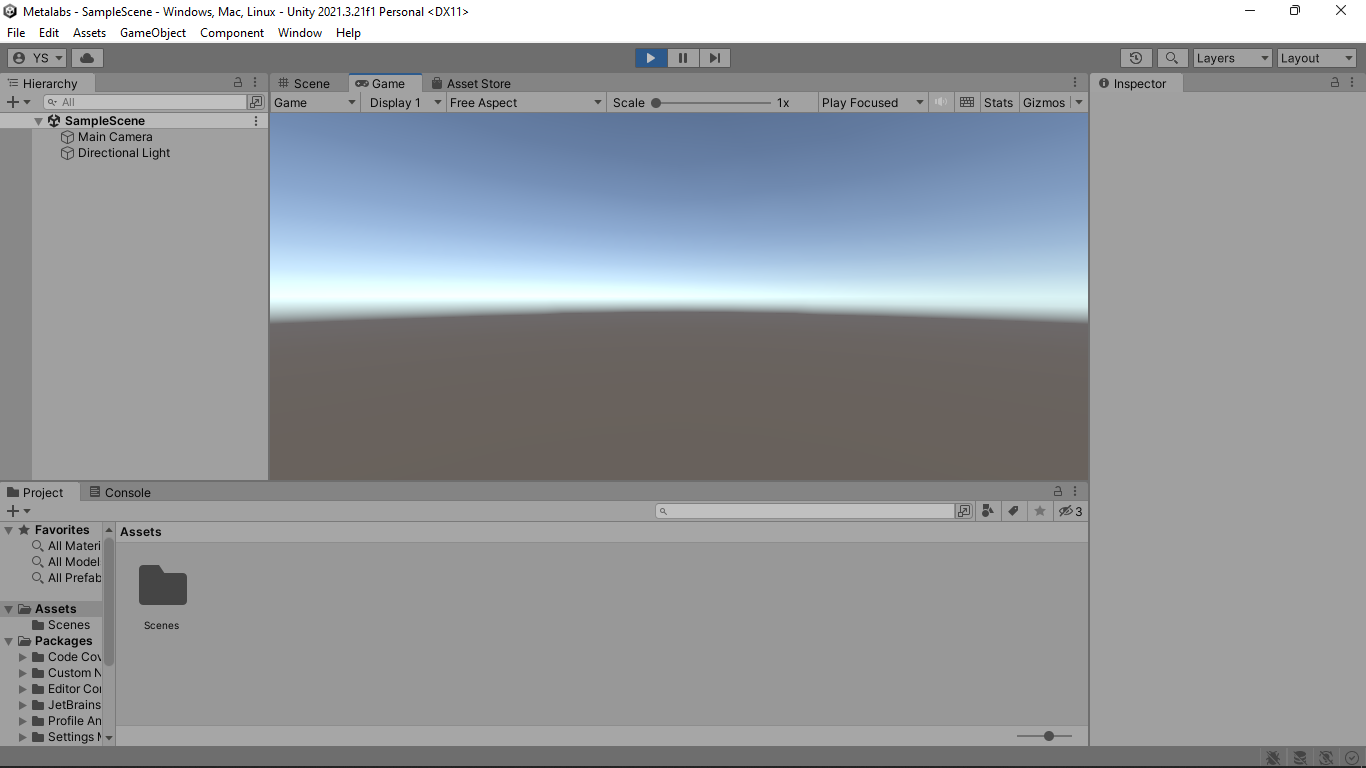
Fungsi-fungsi dasar dalam Unity IDE dapat dikelompokkan menjadi 10, yaitu: "Run/Quit" yang memungkinkan pengujian dan penghentian scene secara real-time; "Show/Hide" yang mengontrol visibilitas objek; "Active/Deactive" serta "Enable/Disable" yang mengelola perilaku komponen; "Create/Delete" untuk pembuatan dan penghapusan objek; "Tag/Layer" yang mengidentifikasi objek dengan detail; "Load/Remove" yang memanipulasi aliran permainan; "Play/Stop" untuk pengendalian elemen multimedia; dan akhirnya, "Save/Load" serta "Export/Import" yang mendukung manajemen dan kolaborasi proyek yang efisien. Kesepuluh fitur ini bersama-sama membentuk dasar esensial untuk mengelola berbagai aspek dalam pengembangan game melalui Unity.

****

**Gambar 1. Fungsi dasar Unity IDE**

## 1. Run/Quit

Tombol "Run" dalam lingkungan Unity, yang sering kali diidentifikasi dengan simbol segitiga berjalan (▶️), adalah perintah yang sangat penting dalam tahap pengembangan game. Fungsi tombol ini adalah untuk menjalankan scene yang sedang Anda kerjakan. Saat Anda mengklik tombol "Run", Unity akan memulai permainan dan memainkan scene yang sedang terbuka, memungkinkan Anda untuk menguji mekanika, visual, dan interaksi secara langsung. Tombol "Run" memberi Anda gambaran langsung tentang bagaimana permainan Anda akan berjalan ketika dimainkan oleh pemain, sehingga memfasilitasi iterasi cepat dan pengembangan yang lebih akurat.



**Gambar 2. Tombol Run/Quit Unity**

### 1.1 Hello World

| using UnityEngine;  public class HelloWorld : MonoBehaviour  {  void Start()  {  Debug.Log("Hello World!");  }  } |
| --- |

Langkah-langkah:

* Buka Unity dan buatlah proyek baru. Buat objek kosong (Empty GameObject) dalam Hierarki.
* Pilih objek kosong tersebut dan pada Inspector, tambahkan komponen "Script" dengan mengklik tombol "Add Component".
* Pilih "New Script" dari menu yang muncul dan beri nama "HelloWorld". Bisa juga dengan cara klik kanan pada “Assets” dan pilih menu “Create > C# Script”
* Double-click script "HelloWorld" untuk membukanya dalam editor kode. Ganti isi script dengan contoh di atas. Simpan script. Kembali ke Unity, tekan tombol "Play" untuk menjalankan permainan. Di console, Anda akan melihat pesan "Hello World!" dicetak.

| **Cara Membuat Hello World Di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/_QAVdl2Oa9k> |

### 1.2 Quit Application (via Key)

| using UnityEngine;  public class QuitApplicationKey : MonoBehaviour  {  void Update()  {  if (Input.GetKeyDown(KeyCode.Escape)) // Tekan tombol Escape untuk keluar  {  QuitGame();  }  }  public void QuitGame()  {  #if UNITY\_EDITOR  UnityEditor.EditorApplication.isPlaying = false;  #else  Application.Quit();  #endif  }  } |
| --- |

Dalam contoh ini, kita menggunakan Update() untuk memeriksa apakah tombol Escape ditekan. Jika ditekan, kita memanggil fungsi QuitGame() yang akan menghentikan permainan. Penting untuk dicatat bahwa di dalam Unity Editor, Application.Quit() mungkin tidak akan benar-benar keluar dari editor. Oleh karena itu, kita menggunakan UnityEditor.EditorApplication.isPlaying dalam kondisi praproses untuk menghentikan mode bermain di Unity Editor. Saat permainan di luar editor, fungsi Application.Quit() akan digunakan untuk keluar dari permainan.

### 1.3 Quit Application (via Listener)

| using UnityEngine;  using UnityEngine.UI;  public class QuitApplicationButton : MonoBehaviour  {  [SerializeField] private Button quitButton;  private void Start()  {  quitButton.onClick.AddListener(QuitGame);  }  private void QuitGame()  {  #if UNITY\_EDITOR  UnityEditor.EditorApplication.isPlaying = false;  #else  Application.Quit();  #endif  }  } |
| --- |

Langkah-langkah:

* Buatlah sebuah tombol di dalam Hierarki.
* Attach skrip QuitButtonController pada objek yang berisi tombol.

### 1.4 Quit Application (via Event)

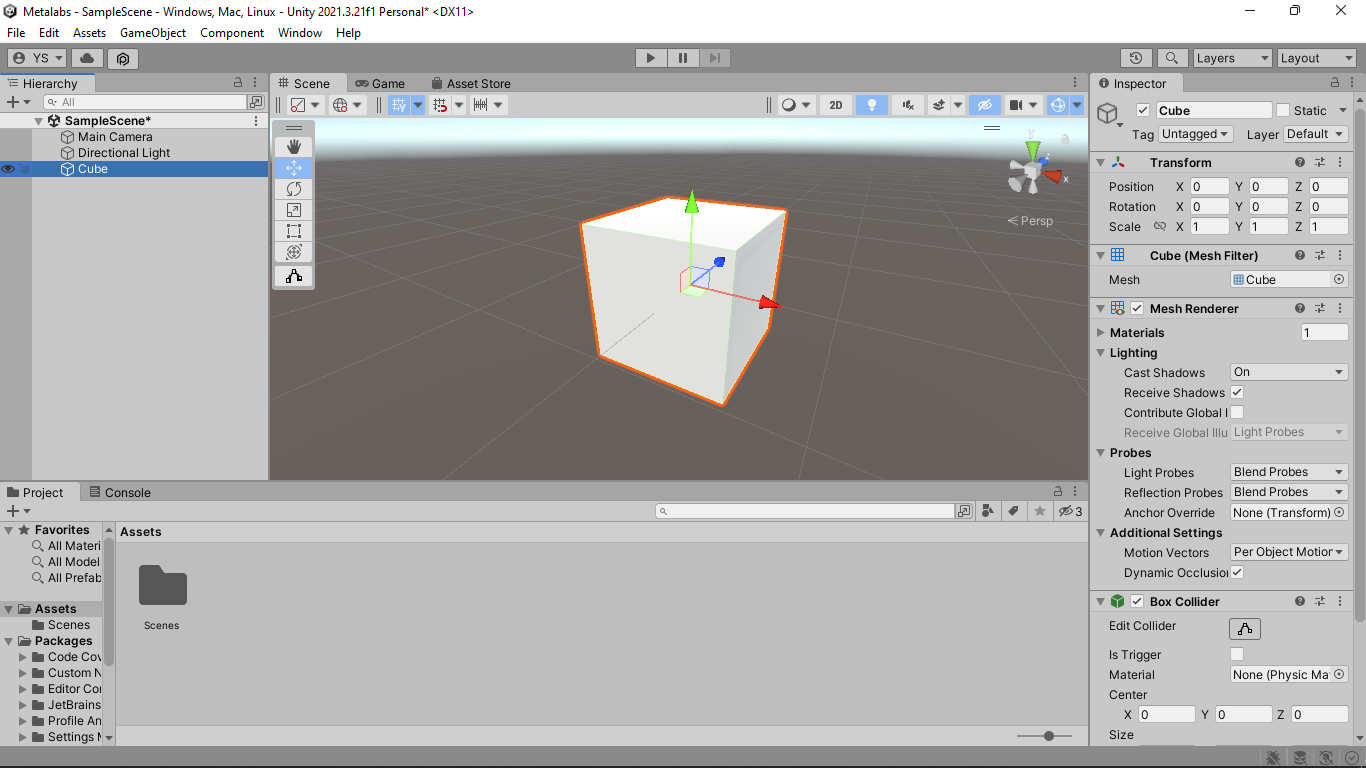
| using UnityEngine;  public class QuitApplicationEvent : MonoBehaviour  {  public void QuitGame()  {  #if UNITY\_EDITOR  UnityEditor.EditorApplication.isPlaying = false;  #else  Application.Quit();  #endif  }  } |
| --- |

Langkah-langkah:

* Buka tampilan Scene di Unity.
* Tambahkan UI Button ke tampilan dengan mengklik kanan di Hierarki, lalu pilih "UI" > "Button".
* Atur posisi dan teks tombol sesuai keinginan.
* Pilih tombol yang baru saja Anda buat di Hierarki.
* Di Inspector, temukan bagian "On Click ()" di komponen Button.
* Klik "+" untuk menambahkan elemen event.
* Seret objek yang memiliki skrip QuitApplication (atau buat objek baru) ke bagian "None (Object)". Dalam dropdown "No Function", pilih skrip QuitApplication dan metode QuitGame().

## 2. Show/Hide

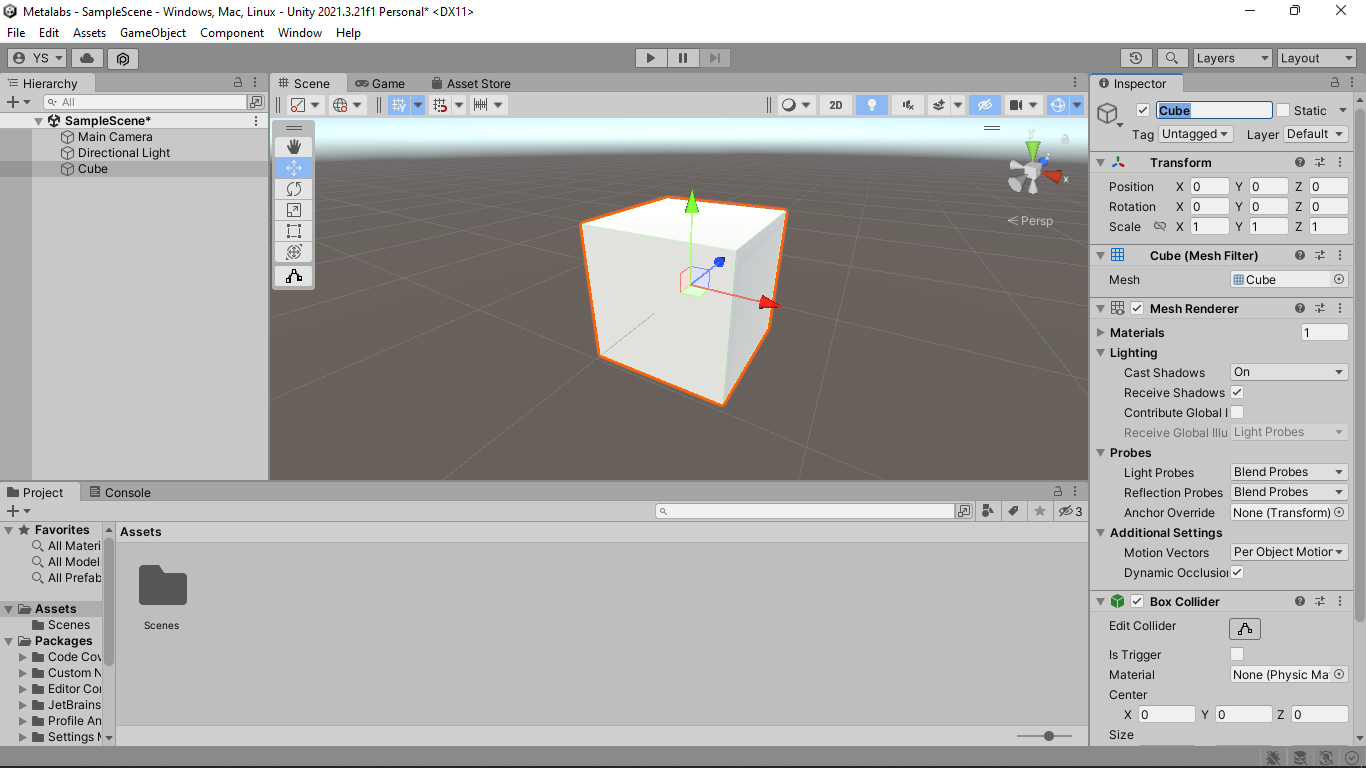
Fungsi "Show/Hide" dalam Unity mengacu pada kemampuan untuk mengontrol keterlihatan game object tertentu di dalam scene. Dengan mengklik Ikon Mata, Anda dapat mengaktifkan atau menonaktifkan visibilitas dari game object tertentu dalam scene. Ketika mata tersebut terlihat (aktif), game object akan terlihat dalam tampilan scene, dan ketika mata tersebut tertutup (nonaktif), game object akan disembunyikan dari tampilan scene. Fungsi ini sangat berguna untuk menyembunyikan atau menampilkan elemen tertentu ketika Anda ingin fokus pada bagian-bagian khusus dari proyek Anda tanpa harus menghapusnya dari scene secara permanen. Dengan demikian, Anda dapat dengan mudah memeriksa dan mengedit objek-objek yang ada dalam scene dengan lebih terperinci.



**Gambar 3. Tombol Show/Hide Unity**

## 3. Active/Deactive

Fungsi "Active/Deactive" pada GameObject dalam Unity mengacu pada kemampuan untuk mengubah Status Keterlihatan dan Keterlibatan sebuah objek dalam scene. Dengan mengaktifkan objek, objek tersebut akan terlihat dan berperilaku sesuai dengan peraturan yang telah ditentukan. Sebaliknya, dengan menonaktifkan objek, objek tersebut akan disembunyikan dan tidak akan berinteraksi dalam tampilan scene. Fungsi ini penting dalam mengontrol objek-objek dalam game, memungkinkan Anda untuk membuat perubahan visual dan perilaku secara dinamis, baik selama pengembangan maupun saat runtime.



**Gambar 4. Tombol Active/Deactive Unity**

### 3.1 Activated GameObject (via Key)

| using UnityEngine;  public class ActivatedGameObjectKey: MonoBehaviour  {  public GameObject objectToToggle;  private void Update()  {  // Tombol 1 untuk menghilangkan GameObject  if (Input.GetKeyDown(KeyCode.Alpha1))  {  if (objectToToggle != null)  {  objectToToggle.SetActive(false);  }  }  // Tombol 2 untuk mengaktifkan kembali GameObject  if (Input.GetKeyDown(KeyCode.Alpha2))  {  if (objectToToggle != null)  {  objectToToggle.SetActive(true);  }  }  }  } |
| --- |

Langkah-langkah:

* Buat sebuah GameObject di dalam Unity dan tambahkan komponen ToggleObject ke GameObject tersebut.
* Seret GameObject yang ingin Anda kendalikan ke dalam bidang "objectToToggle" di komponen ActivatedGameObject di Inspector.
* Ketika tombol 1 ditekan, GameObject yang ditentukan akan dinonaktifkan. Ketika tombol 2 ditekan, GameObject akan diaktifkan kembali.

### 3.2 Activated GameObject (via Listener)

| using UnityEngine;  using UnityEngine.UI;  public class ActivatedGameObjectListener: MonoBehaviour  {  public GameObject objectToToggle;  public Button hideButton;  public Button showButton;  private void Start()  {  hideButton.onClick.AddListener(HideObject);  showButton.onClick.AddListener(ShowObject);  }  private void HideObject()  {  if (objectToToggle != null)  {  objectToToggle.SetActive(false);  }  }  private void ShowObject()  {  if (objectToToggle != null)  {  objectToToggle.SetActive(true);  }  }  } |
| --- |

Langkah-langkah:

* Buat GameObject yang ingin Anda aktifkan dan nonaktifkan di dalam Hierarki Unity.
* Buat sebuah tombol di Hierarki.
* Buat skrip baru bernama ActivatedGameObject dan pasangkannya pada objek yang memiliki tombol.
* Di Inspector, berikan nilai pada variabel objectToToggle dengan GameObject yang ingin Anda kendalikan.
* Di dalam Inspector, seret tombol yang ingin Anda gunakan sebagai tombol pengaktifan/menonaktifkan ke bidang toggleButton dalam skrip.
* Pastikan tombol tersebut memiliki komponen Button.
* Dalam skrip ini, fungsi ToggleObject() akan dijalankan saat tombol ditekan.

### 3.3 Activated GameObject (via Event)

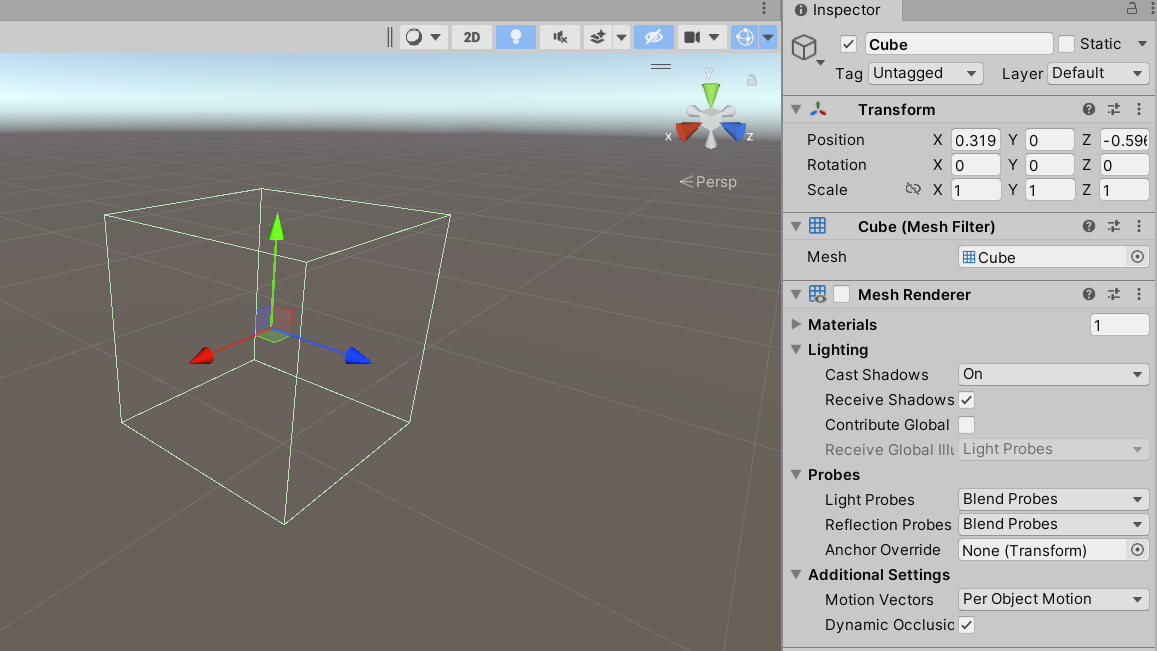
| using UnityEngine;  public class ActivatedGameObjectEvent: MonoBehaviour  {  public GameObject objectToToggle;  // Fungsi untuk menghilangkan GameObject  public void HideObject()  {  if (objectToToggle != null)  {  objectToToggle.SetActive(false);  }  }  // Fungsi untuk menampilkan kembali GameObject  public void ShowObject()  {  if (objectToToggle != null)  {  objectToToggle.SetActive(true);  }  }  } |
| --- |

Langkah-langkah:

* Dalam contoh ini, Anda tidak perlu menambahkan tombol atau listener. Cukup buat dua fungsi publik HideObject dan ShowObject yang mengubah status aktif/nonaktif GameObject yang ingin Anda kendalikan.
* Kemudian, dalam Inspector Unity, pengguna dapat menambahkan skrip ke GameObject yang ingin mereka kendalikan, dan mereka dapat memilih fungsi mana yang akan dijalankan saat suatu event (misalnya, saat tombol tertentu diklik atau suatu event lainnya) terjadi. Pengguna dapat melakukan ini melalui bagian "Events" pada komponen yang mereka tambahkan.

## 4. Enable/Disable

Fungsi "Enabled/Disabled" pada komponen GameObject dalam Unity merujuk pada kemampuan untuk mengaktifkan atau menonaktifkan suatu komponen tertentu pada objek tersebut. Dengan mengaktifkan komponen, Anda memberikan objek akses ke fungsionalitas yang terkait dengan komponen tersebut, sementara dengan menonaktifkannya, Anda menonaktifkan fungsionalitas tersebut. Ini memungkinkan Anda untuk mengontrol interaksi dan perilaku spesifik yang terkait dengan komponen tertentu, memberi Anda fleksibilitas dalam mengubah dinamika permainan dan efisiensi penggunaan sumber daya pada saat yang tepat.



**Gambar 4. Tombol Enabled/Disabled Unity**

### 4.1 Enabled GameObject (via Key)

| using UnityEngine;  public class EnabledGameObjectKey: MonoBehaviour  {  public GameObject objectWithRenderer;  private MeshRenderer meshRenderer;  private void Start()  {  meshRenderer = objectWithRenderer.GetComponent<MeshRenderer>();  if (meshRenderer != null)  {  // Pastikan Mesh Renderer awalnya aktif saat permainan dimulai  meshRenderer.enabled = true;  }  }  private void Update()  {  // Tombol 1 untuk menonaktifkan Mesh Renderer  if (Input.GetKeyDown(KeyCode.Alpha1))  {  if (meshRenderer != null)  {  meshRenderer.enabled = false;  }  }  // Tombol 2 untuk mengaktifkan kembali Mesh Renderer  if (Input.GetKeyDown(KeyCode.Alpha2))  {  if (meshRenderer != null)  {  meshRenderer.enabled = true;  }  }  }  } |
| --- |

Langkah-langkah:

* Seret GameObject yang memiliki komponen Mesh Renderer ke bidang "objectWithRenderer" di komponen ActivatedGameObject di Inspector.
* Di dalam skrip ini, saat tombol 1 ditekan, komponen Mesh Renderer pada GameObject akan dinonaktifkan. Saat tombol 2 ditekan, komponen Mesh Renderer akan diaktifkan kembali.

### 4.2 Enabled GameObject (via Listener)

| using UnityEngine;  using UnityEngine.UI;  public class EnabledGameObjectListener: MonoBehaviour  {  public GameObject objectWithRenderer;  public Button enableButton;  public Button disableButton;  private MeshRenderer meshRenderer;  private void Start()  {  meshRenderer = objectWithRenderer.GetComponent<MeshRenderer>();  enableButton.onClick.AddListener(EnableRenderer);  disableButton.onClick.AddListener(DisableRenderer);  if (meshRenderer != null)  {  // Pastikan Mesh Renderer awalnya aktif saat permainan dimulai  meshRenderer.enabled = true;  }  }  private void EnableRenderer()  {  if (meshRenderer != null)  {  meshRenderer.enabled = true;  }  }  private void DisableRenderer()  {  if (meshRenderer != null)  {  meshRenderer.enabled = false;  }  }  } |
| --- |

Langkah-langkah:

* Seret GameObject yang memiliki komponen Mesh Renderer ke bidang "objectWithRenderer" di komponen EnabledGameObject di Inspector.
* Seret tombol enableButton dan disableButton ke bidang masing-masing di komponen EnabledGameObject di Inspector.
* Tambahkan skrip ini sebagai komponen pada GameObject yang ingin Anda kendalikan.
* Di dalam Inspector, Anda akan melihat tombol-tombol enableButton dan disableButton. Seret GameObject ini ke bidang On Click untuk menghubungkannya ke fungsi EnableRenderer dan DisableRenderer.

### 4.3 Enabled GameObject (via Event)

| using UnityEngine;  public class EnabledGameObjectEvent: MonoBehaviour  {  public GameObject objectWithRenderer;  private MeshRenderer meshRenderer;  private void Start()  {  meshRenderer = objectWithRenderer.GetComponent<MeshRenderer>();  if (meshRenderer != null)  {  // Pastikan Mesh Renderer awalnya aktif saat permainan dimulai  meshRenderer.enabled = true;  }  }  public void EnableRenderer()  {  if (meshRenderer != null)  {  meshRenderer.enabled = true;  }  }  public void DisableRenderer()  {  if (meshRenderer != null)  {  meshRenderer.enabled = false;  }  }  } |
| --- |

Langkah-langkah:

* Seret GameObject yang memiliki komponen Mesh Renderer ke variabel "objectWithRenderer" di komponen EnabledGameObject di Inspector.
* Tambahkan skrip ini sebagai komponen pada GameObject yang ingin Anda kendalikan.
* Di dalam Inspector, Anda akan melihat fungsi EnableRenderer dan DisableRenderer. Seret tombol-tombol dari Hierachy ke fungsi-fungsi ini untuk menghubungkannya.

## 5. Create/Delete

Fungsi "Create/Delete" pada Unity memungkinkan pengguna untuk dengan mudah membuat atau menghapus GameObject dalam scene. Anda dapat membuat GameObject baru melalui menu "GameObject" di bagian atas layar, atau dengan mengklik kanan di dalam hierarki dan memilih opsi "Create". Ini memberi Anda kebebasan untuk dengan cepat menambahkan objek baru ke dalam scene atau menghapus objek yang tidak diperlukan, mempermudah proses pengembangan dan penyusunan elemen-elemen permainan.

**5.1 Create Folder**

Di Unity, pembuatan folder memainkan peran penting dalam pengaturan proyek game Anda. Fungsi ini membantu mengelompokkan asset, skrip, dan elemen-elemen lain menjadi kategori yang terorganisir, yang mempermudah manajemen proyek dan kerja kolaboratif. Untuk membuat folder di Unity, Anda bisa mengikuti langkah-langkah berikut:

* Hierarchy Panel: Panel Hierarchy adalah tempat untuk mengorganisir objek-objek dalam scene. Anda bisa membuat folder dengan mengklik kanan pada Hierarchy Panel, lalu pilih "Create Empty" atau "Create Empty Child". Ini akan membuat GameObject kosong yang dapat Anda gunakan sebagai folder.
* Project Panel: Panel Project adalah tempat untuk mengatur dan mengelola asset di proyek Anda. Untuk membuat folder di sini, klik kanan pada area kosong di Project Panel, lalu pilih "Create" dan pilih "Folder". Anda dapat memberi nama folder sesuai dengan kebutuhan Anda.
* Folder Dalam Folder: Anda juga dapat membuat folder dalam folder untuk menciptakan struktur hierarki yang lebih dalam. Ini berguna untuk mengatur asset secara lebih rinci dan terstruktur.

**5.2 Create GameObject**

Di Unity, pembuatan GameObject adalah tahap kunci dalam mengembangkan dunia visual dalam proyek game Anda. Misalnya, jika Anda ingin membuat objek 3D seperti kubus, langkah-langkah berikut akan memandu Anda melalui proses tersebut.

* Hierarchy Panel: Dalam Hierarchy Panel, klik kanan pada area kosong dan pilih "3D Object". Di sini Anda akan menemukan opsi seperti "Cube" untuk membuat kubus, serta objek-objek primitif lainnya seperti bola (sphere), silinder (cylinder), dan kapsul (capsule). Setelah Anda memilih "Cube", Unity akan membuat GameObject baru dengan komponen Mesh Filter, Mesh Renderer, dan Collider yang sesuai.
* Inspector: Setelah Anda membuat kubus atau objek primitif lainnya, Anda dapat menyesuaikannya melalui Inspector. Anda bisa mengubah skala, rotasi, posisi, dan juga menambahkan komponen lain seperti Material untuk memberikan penampilan visual yang diinginkan.
* Modifikasi: Jika Anda ingin membuat objek yang lebih kompleks atau disesuaikan, Anda dapat mengimpor model 3D yang sudah ada atau membuatnya sendiri menggunakan software 3D modeling seperti Blender atau Maya. Anda dapat menambahkan model tersebut ke dalam proyek Unity dengan menyeretnya ke dalam Hierarchy Panel atau Project Panel.
* Asset Store: Jika Anda tidak ingin membuat model 3D dari awal, Anda juga dapat mencari dan mendownload model 3D dari Unity Asset Store. Setelah diunduh, Anda dapat mengimpor model tersebut ke dalam proyek Anda dan menggunakannya sebagai GameObject.

**5.3 Create Prefabs**

Di Unity, Prefabs adalah elemen yang sangat penting untuk memudahkan pengembangan game yang efisien. Prefabs adalah template GameObject yang dapat digunakan berulang kali di berbagai bagian proyek Anda. Dengan Prefabs, Anda dapat membuat objek atau entitas yang memiliki komponen, skrip, dan konfigurasi khusus, kemudian menggunakannya berkali-kali tanpa harus membuat ulang dari awal setiap kali diperlukan.Proses pembuatan Prefabs adalah sebagai berikut:

* Create Prefabs: Untuk membuat Prefab, Anda dapat mengambil GameObject yang telah Anda buat di scene dan seretnya ke dalam Project Panel. Ini akan membuat salinan GameObject sebagai Prefab. Setiap perubahan yang Anda buat pada Prefab akan diterapkan pada semua instansi Prefab di dalam scene.
* Unpack Prefabs: Namun, kadang-kadang Anda mungkin ingin memutuskan koneksi antara Prefab dan instansi di scene. Untuk melakukannya, Anda dapat menggunakan opsi "Unpack Prefab" di menu konteks. Ini akan mengubah instansi Prefab menjadi GameObject biasa, yang memungkinkan Anda untuk memodifikasinya tanpa mempengaruhi Prefab lainnya.

Unpack Prefabs berguna jika Anda ingin melakukan perubahan khusus pada satu instansi Prefab tanpa mempengaruhi Prefab lainnya atau instansi lainnya. Ini memberi Anda fleksibilitas dalam mengubah hanya bagian-bagian tertentu dari game tanpa mempengaruhi keseluruhan Prefab atau instansi lainnya.

## 6. Tag/Layer

Tag dalam Unity adalah cara yang efektif untuk memberikan identifikasi atau label khusus pada GameObject. Dengan memberikan tag tertentu pada objek, Anda dapat dengan mudah mengenali dan merujuknya melalui skrip. Ini berguna dalam mengidentifikasi objek-objek tertentu seperti karakter, tujuan, atau item dalam permainan. Misalnya, Anda dapat menggunakan tag "Player" untuk mengidentifikasi karakter pemain dan dengan mudah mengelola interaksi atau logika yang berkaitan dengan karakter tersebut.

Layer dalam Unity adalah cara untuk mengelompokkan GameObject berdasarkan kategori tertentu. Setiap objek dapat ditempatkan dalam satu atau beberapa lapisan (layer), dan ini berguna untuk mengendalikan interaksi antara objek-objek tersebut. Misalnya, Anda dapat menempatkan objek-objek yang dapat berinteraksi dengan tanah pada layer "Ground", yang akan memungkinkan mereka untuk berinteraksi dengan objek-objek lain yang juga berada pada layer tersebut.

### 6.1 Find Game Object With Tag

| using UnityEngine;  public class FindGameObjectTag : MonoBehaviour  {  public string targetTag = "TargetTag";  private void Update()  {  if (Input.GetKeyDown(KeyCode.Space))  {  DeactivateObjectWithTag();  }  }  private void DeactivateObjectWithTag()  {  GameObject[] objectsWithTag = GameObject.FindGameObjectsWithTag(targetTag);    foreach (GameObject obj in objectsWithTag)  {  obj.SetActive(false);  }  }  } |
| --- |

Langkah-langkah:

* Tentukan tag target di Inspector, di bidang "targetTag". Misalnya, jika Anda ingin menonaktifkan objek dengan tag "TargetTag", maka setel nilai targetTag menjadi "TargetTag".
* Script ini akan memeriksa apakah tombol klik (dalam contoh ini menggunakan tombol mouse) ditekan.
* Jika tombol klik ditekan, script akan mencari semua GameObject dengan tag yang sesuai.
* Selanjutnya, untuk setiap GameObject dengan tag tersebut, script akan menonaktifkannya.

### 6.2 Find Game Object Within Layer

| using UnityEngine;  public class FindGameObjectLayer : MonoBehaviour  {  public LayerMask targetLayer;  private void Update()  {  if (Input.GetKeyDown(KeyCode.Space))  {  DeactivateObjectsOnLayer();  }  }  private void DeactivateObjectsOnLayer()  {  GameObject[] allObjects = GameObject.FindObjectsOfType<GameObject>();    foreach (GameObject obj in allObjects)  {  if ((targetLayer.value & (1 << obj.layer)) != 0)  {  obj.SetActive(false);  }  }  }  } |
| --- |

Langkah-langkah:

* Public field targetLayer yang merupakan LayerMask yang dapat Anda atur melalui Inspector. Anda dapat menentukan layer-layer tertentu yang ingin Anda proses.
* Script ini akan memeriksa apakah tombol spasi ditekan.
* Jika tombol spasi ditekan, script akan mencari semua GameObject yang ada dalam scene.
* Untuk setiap GameObject, script akan memeriksa apakah GameObject berada pada layer yang sesuai dengan targetLayer. Jika iya, GameObject akan dinonaktifkan.

## 7. Load/Remove

Fungsi "Load/Remove Scene" pada Unity memungkinkan pengembang untuk dinamis mengelola aliran permainan dengan cara memuat atau menghapus scene secara interaktif. Melalui fungsi "Load Scene", Anda dapat memuat scene baru ke dalam permainan, menggantikan scene yang sedang aktif, dan menciptakan perubahan suasana serta konten permainan. Di sisi lain, fungsi "Remove Scene" memungkinkan Anda menghapus scene yang tidak lagi diperlukan dari memori, memastikan efisiensi penggunaan sumber daya dan memungkinkan perpindahan yang mulus antara berbagai bagian permainan. Dengan penggunaan fungsi ini, pengembang dapat menciptakan pengalaman permainan yang terstruktur dan menyatu, sambil memberikan fleksibilitas dalam pengaturan aliran gameplay.

### 

### 7.1 Scene Loader (via Click)

| using UnityEngine;  using UnityEngine.SceneManagement;  public class SceneLoaderClick : MonoBehaviour  {  // Method ini akan dijalankan dengan parameter namaScene yang akan di-load  public void LoadScene(string sceneName)  {  SceneManager.LoadScene(sceneName);  }  } |
| --- |

Script di atas adalah sebuah skrip C# yang ditulis untuk Unity. Tujuannya adalah untuk memuat scene lain dalam permainan Unity. Berikut adalah penjelasan lebih rinci tentang script tersebut:

* using UnityEngine; dan using UnityEngine.SceneManagement;: Ini adalah pernyataan menggunakan namespace yang diperlukan untuk mengakses kelas-kelas Unity yang diperlukan dalam script. UnityEngine digunakan untuk akses umum ke Unity, sementara UnityEngine.SceneManagement digunakan untuk mengakses kelas yang berhubungan dengan manajemen scene.
* public class SceneLoader : MonoBehaviour: Ini adalah deklarasi kelas dengan nama "SceneLoader." Kelas ini adalah turunan dari kelas MonoBehaviour, yang merupakan dasar untuk semua skrip di Unity. Ini berarti kelas ini dapat digunakan sebagai komponen yang dapat ditambahkan ke GameObject di dalam scene Unity.
* public void LoadScene(string sceneName): Ini adalah sebuah metode (fungsi) yang dapat dipanggil dari luar kelas. Metode ini memiliki satu parameter yang disebut sceneName, yang akan digunakan untuk menentukan nama scene yang akan di-load.
* SceneManager.LoadScene(sceneName): Ini adalah pernyataan yang sebenarnya melakukan pekerjaan untuk memuat scene baru. Dalam hal ini, SceneManager.LoadScene digunakan dengan sceneName sebagai argumen. Ini akan memuat scene dengan nama yang diberikan sebagai parameter. Dengan ini, script ini memungkinkan pengguna untuk memuat scene lain dengan memanggil metode LoadScene dan memberikan nama scene yang ingin dimuat sebagai argumen.

### 7.2 Scene Loader (via Delay)

| using UnityEngine;  using UnityEngine.SceneManagement;  public class SceneLoaderDelay : MonoBehaviour  {  public string sceneNameToLoad; // Nama scene yang akan di-load  public float delayInSeconds = 0.0f; // Delay dalam detik sebelum memuat scene  private void Start()  {  // Memanggil fungsi LoadSceneWithDelay dengan nama scene dan delay yang sudah diatur  LoadSceneWithDelay(delayInSeconds);  }  // Method ini akan dijalankan dengan parameter namaScene yang akan di-load  private void LoadSceneWithDelay(float delay)  {  // Menggunakan fungsi Invoke untuk memuat scene dengan delay  Invoke("LoadSceneAfterDelay", delay);  }  private void LoadSceneAfterDelay()  {  SceneManager.LoadScene(sceneNameToLoad);  }  } |
| --- |

Dalam modifikasi ini, kita telah menambahkan dua variabel public, yaitu sceneNameToLoad untuk menyimpan nama scene yang akan di-load dan delayInSeconds untuk menentukan delay sebelum pembebanan scene. Pada method Start, kita memanggil LoadSceneWithDelay dengan menggunakan nilai-nilai ini.

Kemudian, method LoadSceneWithDelay dan LoadSceneAfterDelay tetap sama seperti sebelumnya untuk menangani pembebanan scene dengan delay.Dengan script ini, Anda dapat menambahkan komponen SceneLoader ke GameObject di scene Anda, lalu mengatur sceneNameToLoad dan delayInSeconds dalam Inspector sesuai kebutuhan Anda. Saat scene dimulai (Start dipanggil), scene yang ditentukan akan di-load dengan delay yang sesuai.

### 7.3 Scene Loader (with Loading Screen)

| using System.Collections;  using UnityEngine;  using UnityEngine.SceneManagement;  public class SceneLoaderScreen : MonoBehaviour  {  public string LoadingScene = "LoadingScene";  public float LoadingSceneDelay = 1;  private string targetSceneName; // Nama scene yang akan di-load  // Method ini akan dijalankan dengan parameter namaScene yang akan di-load  public void LoadScene(string sceneName)  {  targetSceneName = sceneName;  StartCoroutine(LoadSceneWithLoadingScreen());  }  IEnumerator LoadSceneWithLoadingScreen()  {  // Muat scene loading screen secara additif  AsyncOperation loadingScreenOperation = SceneManager.LoadSceneAsync(LoadingScene, LoadSceneMode.Additive);  // Tunggu hingga loading screen selesai di-load secara additif  while (!loadingScreenOperation.isDone)  {  yield return null;  }  yield return new WaitForSeconds(LoadingSceneDelay);  // Muat scene tujuan secara asinkron  AsyncOperation targetSceneOperation = SceneManager.LoadSceneAsync(targetSceneName, LoadSceneMode.Single);  // Tunggu hingga scene tujuan selesai di-load  while (!targetSceneOperation.isDone)  {  // Di sini, Anda dapat mengupdate elemen visual pada loading screen (misalnya: progress bar)  float loadingProgress = Mathf.Clamp01(targetSceneOperation.progress / 0.9f);  // Misalnya, Anda dapat mengirimkan nilai loadingProgress ke UI Loading Screen  // loadingProgressBar.value = loadingProgress;  yield return null;  }  // Hapus scene loading screen  SceneManager.UnloadSceneAsync("LoadingScene");  }  } |
| --- |

Dalam script ini, kita menggunakan dua coroutine yang berjalan secara bersamaan:

* Coroutine pertama (LoadSceneWithLoadingScreen) digunakan untuk memuat scene loading screen secara additif, menunggu hingga loading screen selesai di-load, kemudian memuat scene tujuan secara asinkron. Selama proses ini, Anda dapat mengupdate elemen visual pada loading screen untuk menampilkan progres loading.
* Coroutine kedua (LoadSceneWithLoadingScreen) digunakan untuk memuat scene tujuan secara asinkron. Kami menggunakan AsyncOperation untuk memantau progres pembebanan scene, yang dapat digunakan untuk memperbarui elemen visual pada loading screen.

## 8. Play/Stop

Fungsi "Play/Stop" pada elemen multimedia dalam Unity adalah cara untuk mengontrol pemutaran dan penghentian elemen multimedia seperti suara dan video dalam permainan. Dengan menggunakan fungsi "Play", Anda memulai pemutaran elemen multimedia yang terhubung ke objek tertentu, seperti suara latar atau animasi video. Sebaliknya, dengan fungsi "Stop", Anda menghentikan pemutaran elemen multimedia tersebut. Fungsi ini memberi pengembang kendali penuh atas bagaimana elemen multimedia berperilaku dalam permainan, menciptakan pengalaman audiovisual yang lebih dinamis dan mendukung suasana permainan secara keseluruhan.

### 8.1 Audio Controller

| using UnityEngine;  public class AudioController : MonoBehaviour  {  public AudioSource audioSource; // Audio source yang akan dikontrol  // Method untuk memainkan audio  public void Play()  {  if (audioSource != null)  {  audioSource.Play();  }  else  {  Debug.LogError("Audio source belum ditentukan!");  }  }  // Method untuk menghentikan audio  public void Stop()  {  if (audioSource != null && audioSource.isPlaying)  {  audioSource.Stop();  }  }  } |
| --- |

AudioSource adalah komponen AudioSource yang akan Anda kontrol. Anda perlu menambahkan AudioSource ini ke GameObject yang memiliki skrip ini atau menghubungkannya melalui Inspector.

* Play() adalah method untuk memainkan audio. Jika audioSource sudah ditentukan, itu akan memutar audio. Jika audioSource belum ditentukan, itu akan menampilkan pesan kesalahan di konsol.
* Stop() adalah method untuk menghentikan audio. Ini akan menghentikan audio jika audio sedang dimainkan dan audioSource sudah ditentukan.

Anda dapat menambahkan script ini ke GameObject apa pun yang memiliki AudioSource yang ingin Anda kontrol. Kemudian, Anda dapat memanggil PlayAudio() dan StopAudio() dari script lain atau melalui event tertentu dalam permainan Anda untuk mengendalikan pemutaran audio.

### 8.2 Video Controller

| using UnityEngine;  using UnityEngine.Video;  public class VideoController : MonoBehaviour  {  public VideoPlayer videoPlayer;  private void PlayVideo()  {  if (!videoPlayer.isPlaying)  {  // Memulai pemutaran video jika belum dimainkan  videoPlayer.Play();  }  }  private void StopVideo()  {  if (videoPlayer.isPlaying)  {  // Menghentikan pemutaran video jika sedang dimainkan  videoPlayer.Stop();  }  }  } |
| --- |

VideoPlayer adalah komponen Video yang akan Anda kontrol. Anda perlu menambahkan VideoPlayer ini ke GameObject yang memiliki skrip ini atau menghubungkannya melalui Inspector.

* Play() adalah method untuk memainkan audio. Jika audioSource sudah ditentukan, itu akan memutar audio. Jika audioSource belum ditentukan, itu akan menampilkan pesan kesalahan di konsol.
* Stop() adalah method untuk menghentikan audio. Ini akan menghentikan audio jika audio sedang dimainkan dan audioSource sudah ditentukan.

Anda dapat menambahkan script ini ke GameObject apa pun yang memiliki VideoPlayer yang ingin Anda kontrol. Kemudian, Anda dapat memanggil Play() dan Stop() dari script lain atau melalui event tertentu dalam permainan Anda untuk mengendalikan pemutaran video.

## 9. Save/Load

Proses "Save/Load" dalam proyek Unity merujuk pada kemampuan untuk menyimpan dan memuat proyek pengembangan permainan Anda. Melalui fungsi "Save", Anda menyimpan semua perubahan yang telah Anda buat dalam proyek Anda, termasuk skrip, asset, konfigurasi, dan lainnya. Ini memungkinkan Anda untuk melanjutkan pekerjaan dari titik terakhir saat Anda membuka proyek kembali. Sementara itu, dengan fungsi "Load", Anda dapat memuat kembali proyek yang telah disimpan sebelumnya, memungkinkan Anda untuk melanjutkan pengembangan dari titik tertentu dan melihat perubahan yang telah Anda buat. Fungsi ini sangat penting dalam memastikan kontinuitas pengembangan, memungkinkan Anda untuk eksplorasi kreatif dan pengujian tanpa khawatir kehilangan kemajuan yang telah Anda capai.

## 10. Export/Import

"Import/Export Package" dalam Unity adalah proses yang memungkinkan Anda untuk berbagi dan mengintegrasikan aset, skrip, atau konfigurasi dengan proyek lain atau dengan komunitas. Dengan fungsi "Export Package", Anda dapat memilih aset-aset tertentu dari proyek Anda untuk dikemas dalam paket yang dapat dibagikan. Di sisi lain, fungsi "Import Package" memungkinkan Anda untuk memasukkan paket-paket tersebut ke dalam proyek Anda, dengan opsi untuk memilih komponen-komponen yang ingin Anda gunakan. Ini mempermudah kolaborasi tim, berbagi aset dengan komunitas, dan menghindari duplikasi pekerjaan dengan memanfaatkan aset yang sudah ada.

# 

# 

# Chapter 4 Input Classification

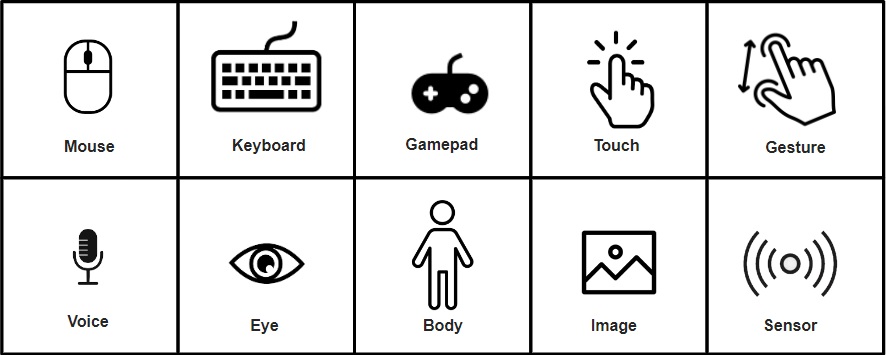
Interaksi antara pemain dan sistem permainan sangat penting untuk menciptakan pengalaman bermain yang memuaskan. Salah satu aspek penting dari interaksi ini adalah input, yaitu cara pemain memberikan perintah atau mengontrol permainan. Terdapat sepuluh jenis input yang berbeda, yaitu mouse, keyboard, gamepad, touch, gesture, voice, eye, body, image, dan sensor. Setiap jenis input ini memiliki karakteristik dan mekanik yang berbeda, yang akan mempengaruhi bagaimana pemain dapat berinteraksi dengan permainan.

Contoh pertama adalah input mouse, yang adalah salah satu jenis input yang paling umum digunakan dalam permainan komputer. Mouse look adalah mekanisme yang memungkinkan pemain untuk menggerakkan kamera atau pandangan karakter dalam game dengan menggerakkan mouse. Sedangkan mouse orbit adalah mekanisme yang memungkinkan kamera bergerak mengelilingi objek tertentu saat pemain menggerakkan mouse. Keyboard adalah jenis input lain yang umum digunakan. Biasanya, tombol-tombol tertentu pada keyboard diassign untuk melakukan tindakan tertentu dalam game, misalnya tombol 'W' untuk bergerak ke depan, 'A' untuk bergerak ke kiri, dan sebagainya.

Selanjutnya adalah gamepad, yang merupakan alat kontrol berbentuk genggam yang umum digunakan untuk bermain game konsol. Gamepad biasanya memiliki tombol-tombol dan stick analog yang dapat digerakkan untuk mengontrol karakter atau objek dalam game. Input touch adalah jenis input yang umum digunakan pada perangkat mobile, seperti smartphone dan tablet. Pemain dapat berinteraksi dengan game dengan menyentuh layar perangkat. Gesture adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan melakukan gerakan-gerakan tertentu, misalnya menggerakkan tangan ke kiri atau ke kanan.

Voice adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan menggunakan suara. Misalnya, pemain dapat memberikan perintah kepada karakter dalam game dengan mengucapkan kata-kata tertentu. Eye adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan gerakan mata. Misalnya, pemain dapat menggerakkan pandangan karakter dalam game dengan menggerakkan mata ke arah tertentu. Body adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan gerakan tubuh. Misalnya, pemain dapat menggerakkan karakter dalam game dengan berjalan atau berlari di tempat.

Terakhir, image adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan menggunakan gambar atau objek visual lainnya. Misalnya, pemain dapat menggerakkan karakter dalam game dengan menunjukkan gambar tertentu ke kamera. Sensor adalah jenis input yang memungkinkan pemain untuk mengontrol game dengan menggunakan sensor-sensor tertentu, seperti sensor gerak atau sensor kecepatan. Misalnya, pemain dapat menggerakkan karakter dalam game dengan menggerakkan smartphone ke arah tertentu. Setiap jenis input ini memiliki kelebihan dan kekurangannya masing-masing, dan pemilihan jenis input yang tepat akan sangat bergantung pada jenis game yang sedang dikembangkan.

****

**Gambar 5. Game Input Classification**

## 1. Mouse

Mouse adalah salah satu alat input utama dalam berbagai jenis permainan, terutama untuk permainan PC. Variasi input yang berkaitan dengan mouse sangat beragam, misalnya 'Mouse Look' yang memungkinkan pemain untuk menggerakkan kamera atau pandangan karakter dalam game dengan menggerakkan mouse. 'Mouse Orbit' adalah mekanisme lain yang memungkinkan kamera untuk bergerak mengelilingi objek tertentu saat pemain menggerakkan mouse. Selain itu, klik kiri dan klik kanan mouse sering digunakan untuk melakukan aksi seperti menembak atau memilih objek. Scroll wheel pada mouse juga dapat digunakan untuk berbagai fungsi, seperti zoom in/zoom out pandangan atau mengganti senjata. Dengan memanfaatkan berbagai fungsi dan mekanisme input mouse ini, pengembang game dapat menciptakan pengalaman bermain yang lebih responsif dan interaktif bagi pemain.

### 

### 1.1 Mouse Click (Hardcode Version)

| using UnityEngine;  public class MouseClick : MonoBehaviour  {  void Update()  {  if (Input.GetMouseButtonDown(0)) // 0 represents the left mouse button  {  Vector3 mousePosition = Input.mousePosition;  Debug.Log("Left mouse button clicked at (" + mousePosition.x + ", " + mousePosition.y + ")");  }    if (Input.GetMouseButtonDown(1)) // 1 represents the right mouse button  {  Vector3 mousePosition = Input.mousePosition;  Debug.Log("Right mouse button clicked at (" + mousePosition.x + ", " + mousePosition.y + ")");  }  }  } |
| --- |

Skrip "Mouse Click" adalah implementasi dalam Unity yang mendeteksi klik tombol mouse dan memicu aksi berdasarkan jenis tombol yang ditekan, seperti menampilkan pesan saat klik kiri dan membalik status saat klik kanan. Ini memungkinkan interaksi yang responsif dengan objek dalam permainan melalui input mouse.

Dalam contoh di atas, kita membuat sebuah Skrip dengan nama MouseClickDetection yang ditempatkan pada GameObject di Unity. Metode Update akan dipanggil setiap frame. Di dalamnya, kita menggunakan Input.GetMouseButtonDown untuk mendeteksi apakah tombol mouse telah ditekan. Angka 0 mewakili tombol kiri, dan angka 1 mewakili tombol kanan. Ketika klik terdeteksi, kita mendapatkan posisi mouse menggunakan Input.mousePosition dan kemudian mencetak pesan ke konsol Unity dengan posisi mouse yang sesuai. Pastikan Anda menghubungkan Skrip ini ke GameObject yang Anda inginkan di dalam Unity. Anda juga dapat menyesuaikan logika lebih lanjut atau berinteraksi dengan elemen-elemen permainan sesuai kebutuhan Anda.

### 1.2 Mouse Click (UnityEvent Version)

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class MouseClick : MonoBehaviour  {  [Header("Default Function")]  public UnityEvent StartEvent;  public UnityEvent UpdateEvent;  [Header("Click Event")]  public UnityEvent TrueClickEvent;  public UnityEvent FalseClickEvent;  bool ClickStatus;  // Start is called before the first frame update  void Start()  {  ClickStatus = false;  StartEvent?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvent?.Invoke();  }  void OnMouseDown()  {  ClickStatus = !ClickStatus;  if (ClickStatus)  {  TrueClickEvent?.Invoke();  }  else  {  FalseClickEvent?.Invoke();  }  }  void InvokeTrueEvent()  {  TrueClickEvent.Invoke();  }  void InvokeFalseEvent()  {  FalseClickEvent.Invoke();  }  } |
| --- |

Skrip "Mouse Click" versi ini mengubah eksekusi fungsi hasil deteksi input mouse menggunakan Unity Events. Teknik ini memungkinkan output dari Skrip dikonfigurasi melalui Inspector Unity.

UnityEvent Variabel dan Fungsi Callback:

Pada bagian atas Skrip, terdapat beberapa variabel tipe UnityEvent. UnityEvent adalah tipe khusus dalam Unity yang memungkinkan Anda mengaitkan berbagai fungsi (callback) dengan event. Dalam hal ini, ada empat jenis UnityEvent yang dideklarasikan:

* StartEvent: Akan dijalankan saat Start() dipanggil.
* UpdateEvent: Akan dijalankan pada setiap frame saat Update() dipanggil.
* TrueClickEvent: Akan dijalankan saat tombol mouse ditekan dan ClickStatus menjadi true.
* FalseClickEvent: Akan dijalankan saat tombol mouse ditekan dan ClickStatus menjadi false.
* ClickStatus:
* bool ClickStatus merupakan variabel yang digunakan untuk melacak apakah tombol mouse telah diklik atau tidak. Nilai awalnya diatur menjadi false.

Start():

Metode Start() dijalankan pada awal permainan atau saat GameObject pertama kali diaktifkan. Pada metode ini, ClickStatus diatur menjadi false, dan StartEvent dijalankan jika telah diisi. StartEvent?.Invoke(); digunakan untuk memanggil event jika tidak null.

Update():

Metode Update() dijalankan pada setiap frame permainan. Pada metode ini, UpdateEvent dijalankan jika telah diisi. UpdateEvent?.Invoke(); digunakan untuk memanggil event jika tidak null.

OnMouseDown():

Metode ini dijalankan saat GameObject menerima event klik dari tombol mouse. Di dalam metode ini, nilai ClickStatus dibalik (dari true menjadi false, atau sebaliknya). Selanjutnya, TrueClickEvent akan dijalankan jika ClickStatus menjadi true, dan FalseClickEvent akan dijalankan jika ClickStatus menjadi false.

InvokeTrueEvent() dan InvokeFalseEvent():

Dua metode ini adalah alternatif untuk secara manual memanggil event TrueClickEvent dan FalseClickEvent. Namun, dalam Skrip ini, kedua metode ini tidak digunakan.

Skrip ini dirancang untuk memberikan berbagai fungsi callback yang dapat dihubungkan dengan event-event yang terjadi saat permainan berjalan. Misalnya, Anda dapat menghubungkan TrueClickEvent untuk memulai animasi atau berbagai tindakan yang ingin diambil saat mouse diklik, dan FalseClickEvent untuk melakukan tindakan ketika mouse dilepas. Anda juga dapat menggunakan StartEvent dan UpdateEvent untuk menjalankan tindakan tertentu saat permainan dimulai atau pada setiap frame permainan.

### 1.3 Mouse Rotator

| using UnityEngine;  public class MouseRotator : MonoBehaviour  {  public enum RotationType  {  Horizontal,  Vertical,  Both  }  public RotationType rotationType = RotationType.Both;  public float rotationSpeed = 5.0f;  private Vector3 lastMousePosition;  void Update()  {  if (Input.GetMouseButtonDown(0))  {  lastMousePosition = Input.mousePosition;  }  else if (Input.GetMouseButton(0))  {  Vector3 deltaMousePosition = Input.mousePosition - lastMousePosition;  float rotationX = 0.0f;  float rotationY = 0.0f;  switch (rotationType)  {  case RotationType.Horizontal:  rotationY = deltaMousePosition.x \* rotationSpeed \* Time.deltaTime;  break;  case RotationType.Vertical:  rotationX = -deltaMousePosition.y \* rotationSpeed \* Time.deltaTime;  break;  case RotationType.Both:  rotationX = -deltaMousePosition.y \* rotationSpeed \* Time.deltaTime;  rotationY = deltaMousePosition.x \* rotationSpeed \* Time.deltaTime;  break;  }  transform.Rotate(rotationX, rotationY, 0.0f, Space.Self);  lastMousePosition = Input.mousePosition;  }  }  } |
| --- |

Skrip "Mouse Rotator" di Unity memungkinkan rotasi objek dengan mengikuti gerakan mouse. Dengan opsi untuk memilih rotasi horizontal, vertikal, atau keduanya, skrip ini memberi pengguna kendali intuitif terhadap perubahan orientasi objek dalam permainan 3D.

Enum RotationType:

Pada bagian awal skrip, Anda mendefinisikan enum bernama RotationType. Enum ini digunakan untuk memilih tipe rotasi yang akan diaktifkan.

* Horizontal: Objek akan dirotasi hanya pada sumbu Y.
* Vertical: Objek akan dirotasi hanya pada sumbu X.
* Both: Objek akan dirotasi pada kedua sumbu X dan Y.

Public Variables:

* rotationType: Variabel publik yang dapat diatur melalui Inspector di Unity. Ini menentukan tipe rotasi yang akan diaktifkan sesuai dengan nilai dari RotationType.
* rotationSpeed: Kecepatan rotasi objek ketika gerakan mouse terdeteksi.

Private Variables:

lastMousePosition: Digunakan untuk menyimpan posisi mouse pada frame sebelumnya.

Update() Method:

Metode ini dijalankan setiap frame dalam permainan.

* Pertama, kita memeriksa apakah tombol kiri mouse baru saja ditekan (Input.GetMouseButtonDown(0)). Jika iya, kita menyimpan posisi mouse saat ini sebagai lastMousePosition untuk perbandingan di frame berikutnya.
* Selanjutnya, jika tombol kiri mouse terus ditekan (Input.GetMouseButton(0)), kita menghitung perbedaan posisi mouse saat ini dengan lastMousePosition sebagai deltaMousePosition.
* Berdasarkan tipe rotasi yang dipilih dalam rotationType, kita menghitung rotasi pada sumbu X dan/atau sumbu Y sesuai dengan perubahan posisi mouse (deltaMousePosition). Nilai rotasi dihitung dengan mengalikan deltaMousePosition dengan rotationSpeed dan Time.deltaTime untuk menjaga konsistensi rotasi pada berbagai kecepatan frame.

Switch Case for Rotation:

Di dalam switch case, kita memeriksa nilai dari rotationType yang telah diatur:

* Jika rotationType adalah Horizontal, maka hanya rotasi sumbu Y yang dihitung (rotationY).
* Jika rotationType adalah Vertical, maka hanya rotasi sumbu X yang dihitung (rotationX).
* Jika rotationType adalah Both, maka kedua sumbu X dan Y dihitung (rotationX dan rotationY).

Applying Rotation:

Setelah menghitung nilai rotasi sesuai dengan tipe yang dipilih, kita menerapkan rotasi pada objek menggunakan transform.Rotate. Kami menggunakan Space.Self untuk memastikan rotasi relatif terhadap orientasi objek saat ini.

Updating lastMousePosition:

Pada akhir Update, kita memperbarui lastMousePosition dengan posisi mouse saat ini untuk digunakan dalam frame berikutnya. Dengan menggunakan skrip ini dan mengatur nilai rotationType melalui Inspector, Anda dapat mengendalikan cara objek di dalam permainan merespons gerakan mouse.

### 1.4 Mouse Look

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class MouseLook : MonoBehaviour  {  public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }  [Header ("Settings")]  public RotationAxes axes = RotationAxes.MouseXAndY;  public float sensitivityX = 15F;  public float sensitivityY = 15F;  public float minimumX = -360F;  public float maximumX = 360F;  public float minimumY = -60F;  public float maximumY = 60F;  float rotationX = 0F;  float rotationY = 0F;  Quaternion originalRotation;  void Start()  {  // Make the rigid body not change rotation  if (GetComponent<Rigidbody>())  GetComponent<Rigidbody>().freezeRotation = true;  originalRotation = transform.localRotation;  }  void Update()  {  if (axes == RotationAxes.MouseXAndY)  {  // Read the mouse input axis  rotationX += Input.GetAxis("Mouse X") \* sensitivityX;  rotationY += Input.GetAxis("Mouse Y") \* sensitivityY;  rotationX = ClampAngle(rotationX, minimumX, maximumX);  rotationY = ClampAngle(rotationY, minimumY, maximumY);  Quaternion xQuaternion = Quaternion.AngleAxis(rotationX, Vector3.up);  Quaternion yQuaternion = Quaternion.AngleAxis(rotationY, -Vector3.right);  transform.localRotation = originalRotation \* xQuaternion \* yQuaternion;  }  else if (axes == RotationAxes.MouseX)  {  rotationX += Input.GetAxis("Mouse X") \* sensitivityX;  rotationX = ClampAngle(rotationX, minimumX, maximumX);  Quaternion xQuaternion = Quaternion.AngleAxis(rotationX, Vector3.up);  transform.localRotation = originalRotation \* xQuaternion;  }  else  {  rotationY += Input.GetAxis("Mouse Y") \* sensitivityY;  rotationY = ClampAngle(rotationY, minimumY, maximumY);  Quaternion yQuaternion = Quaternion.AngleAxis(-rotationY, Vector3.right);  transform.localRotation = originalRotation \* yQuaternion;  }  }  public static float ClampAngle(float angle, float min, float max)  {  if (angle < -360F)  angle += 360F;  if (angle > 360F)  angle -= 360F;  return Mathf.Clamp(angle, min, max);  }  } |
| --- |

Skrip "Mouse Look" dalam Unity memampukan kontrol pandangan kamera dengan gerakan mouse. Melalui sensitivitas yang dapat disesuaikan, skrip ini memungkinkan pengguna menggeser pandangan horizontal atau vertikal, atau keduanya, untuk menciptakan pengalaman pandangan yang lebih interaktif dalam lingkungan permainan.

Enum RotationAxes:

Pada bagian atas skrip, terdapat enum bernama RotationAxes. Ini digunakan untuk memilih jenis rotasi yang akan diaktifkan.

MouseXAndY: Memungkinkan rotasi di sekitar sumbu X dan Y.

MouseX: Hanya memungkinkan rotasi di sekitar sumbu X.

MouseY: Hanya memungkinkan rotasi di sekitar sumbu Y.

Public Variables:

axes: Variabel publik yang dapat diatur melalui Inspector di Unity. Ini menentukan tipe rotasi yang akan diaktifkan sesuai dengan nilai dari RotationAxes.

sensitivityX: Kecepatan sensitivitas rotasi sumbu X.

sensitivityY: Kecepatan sensitivitas rotasi sumbu Y.

minimumX dan maximumX: Batasan minimum dan maksimum untuk rotasi sumbu X.

minimumY dan maximumY: Batasan minimum dan maksimum untuk rotasi sumbu Y.

Private Variables:

rotationX dan rotationY: Menyimpan nilai rotasi saat ini untuk sumbu X dan Y.

originalRotation: Menyimpan rotasi awal objek sebelum dimodifikasi.

Start() Method:

Pada Start(), skrip mengatur freezeRotation dari komponen Rigidbody jika ada, untuk mencegah perubahan rotasi oleh fisika.originalRotation diset dengan rotasi awal objek untuk penggunaan sebagai referensi.

Update() Method:

Metode Update() dijalankan setiap frame dalam permainan.Bergantung pada tipe rotasi yang diatur di axes, skrip akan mengambil input gerakan mouse dan menghitung rotasi yang sesuai untuk sumbu X dan/atau Y.Di bagian akhir, transform.localRotation diubah berdasarkan rotasi yang dihitung.

ClampAngle() Method:

Ini adalah metode statis yang mengambil tiga argumen: angle yang akan diklem, min (nilai minimum), dan max (nilai maksimum). Metode ini digunakan untuk memastikan bahwa nilai rotasi tetap berada dalam rentang yang diizinkan.

### 1.5 Mouse Orbit

| using UnityEngine;  using UnityEngine.Events;  using System.Collections;  public class MouseOrbit : MonoBehaviour  {  [Header("Target Object")]  public Transform target;    [Header("Mouse Settings")]  public float distance = 5.0f;  public float xSpeed = 120.0f;  public float ySpeed = 120.0f;  public float yMinLimit = -20f;  public float yMaxLimit = 80f;  public float distanceMin = .5f;  public float distanceMax = 15f;  [Header("Default Function")]  public UnityEvent StartEvent;  public UnityEvent UpdateEvent;  private Rigidbody rigidbody;  float x = 0.0f;  float y = 0.0f;  // Use this for initialization  void Start()  {  Vector3 angles = transform.eulerAngles;  x = angles.y;  y = angles.x;  rigidbody = GetComponent<Rigidbody>();  // Make the rigid body not change rotation  if (rigidbody != null)  {  rigidbody.freezeRotation = true;  }  StartEvent?.Invoke();  }  void LateUpdate()  {  if (target && Input.GetMouseButton(0))  {  x += Input.GetAxis("Mouse X") \* xSpeed \* distance \* 0.02f;  y -= Input.GetAxis("Mouse Y") \* ySpeed \* 0.02f;  y = ClampAngle(y, yMinLimit, yMaxLimit);  Quaternion rotation = Quaternion.Euler(y, x, 0);  distance = Mathf.Clamp(distance - Input.GetAxis("Mouse ScrollWheel") \* 5, distanceMin, distanceMax);  RaycastHit hit;  if (Physics.Linecast(target.position, transform.position, out hit))  {  distance -= hit.distance;  }  Vector3 negDistance = new Vector3(0.0f, 0.0f, -distance);  Vector3 position = rotation \* negDistance + target.position;  transform.rotation = rotation;  transform.position = position;  }  UpdateEvent?.Invoke();  }  public static float ClampAngle(float angle, float min, float max)  {  if (angle < -360F)  angle += 360F;  if (angle > 360F)  angle -= 360F;  return Mathf.Clamp(angle, min, max);  }  } |
| --- |

Skrip "Mouse Orbit" di Unity memfasilitasi perputaran kamera mengelilingi objek target berdasarkan gerakan mouse. Dengan fitur yang memungkinkan pengguna menggeser kamera secara horizontal atau vertikal, skrip ini menciptakan tampilan orbit interaktif untuk melihat objek dari berbagai sudut dalam permainan 3D.

Target Object:

Public Transform yang mengacu pada objek yang ingin diputari menggunakan kamera.

Mouse Settings:

Pengaturan sensitivitas dan batasan rotasi kamera.

* distance: Jarak antara kamera dan objek target.
* xSpeed dan ySpeed: Kecepatan rotasi horizontal dan vertikal saat berputar kamera.
* yMinLimit dan yMaxLimit: Batasan sudut vertikal kamera yang dapat diatur.
* distanceMin dan distanceMax: Batasan jarak minimum dan maksimum antara kamera dan objek target.

Default Function:

UnityEvent yang dapat diisi untuk aksi kustom saat skrip dimulai atau diperbarui.

* StartEvent: Diaktifkan saat skrip dimulai (Awake/Start).
* UpdateEvent: Diaktifkan setiap frame dalam Update.

Rigidbody and Initialization:

Pada Start(), rotasi awal objek dan rigidbody diinisialisasi. Jika ada komponen Rigidbody, rotasi fisika dibekukan.

LateUpdate() Method:

Metode LateUpdate() yang dijalankan setiap frame setelah Update. Mengatur perputaran kamera berdasarkan gerakan mouse jika tombol kiri mouse ditekan.

* x dan y diubah berdasarkan gerakan mouse.
* y diklamping ke rentang sudut vertikal yang diizinkan.
* Rotasi dihitung menggunakan Euler angles.
* Jarak kamera diatur berdasarkan gerakan mouse dan scroll wheel.
* Jika ada hambatan antara kamera dan target, jarak dikoreksi.
* Akhirnya, posisi dan rotasi kamera diatur sesuai perhitungan.

ClampAngle() Method:

Metode statis yang mengklamping sudut ke dalam rentang yang diizinkan.Jika sudut di luar rentang -360 hingga 360 derajat, sudut dikoreksi ke dalam rentang yang diizinkan.

### 1.6 Mouse Isometric

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.EventSystems;  public class MouseIsometric : MonoBehaviour  {  [Header("Main Settings")]  Camera TargetCamera;  public float MoveSpeed = 5;  public float LookSpeed = 500;  public float PanSpeed = 3;  public float ScrollSpeed = 50;  [Header("Mouse Settings")]  KeyCode MouseLookButton = KeyCode.Mouse1;  int MousePanButton = 2;  private float X;  private float Y;  Vector3 lastPosition;  RaycastHit RaycastResult;  [Header("Distance Setting")]  public float distance = 5f;  public float minDistance = 1f; //Min distance of the camera from the target  public float maxDistance = 10f;  public int yMinLimit = 10; //Lowest vertical angle in respect with the target.  public int yMaxLimit = 80;  public float mouseXSpeed = 1000.0f;  public float mouseYSpeed = 1000.0f;  private float x = 0.0f;  private float y = 0.0f;  [Header("Default Function")]  public UnityEvent StartEvent;  public UnityEvent UpdateEvent;  public static bool IsPointerOverUIElement()  {  return IsPointerOverUIElement(GetEventSystemRaycastResults());  }  ///Returns 'true' if we touched or hovering on Unity UI element.  public static bool IsPointerOverUIElement(List<RaycastResult> eventSystemRaysastResults)  {  for (int index = 0; index < eventSystemRaysastResults.Count; index++)  {  RaycastResult curRaysastResult = eventSystemRaysastResults[index];  if (curRaysastResult.gameObject.layer == LayerMask.NameToLayer("UI"))  return true;  }  return false;  }  ///Gets all event systen raycast results of current mouse or touch position.  static List<RaycastResult> GetEventSystemRaycastResults()  {  PointerEventData eventData = new PointerEventData(EventSystem.current);  eventData.position = Input.mousePosition;  List<RaycastResult> raysastResults = new List<RaycastResult>();  EventSystem.current.RaycastAll(eventData, raysastResults);  return raysastResults;  }  void StartPosition()  {  const float minX = 0.0f;  const float maxX = 360.0f;  const float minY = -90.0f;  const float maxY = 90.0f;  X += Input.GetAxis("Mouse X") \* (LookSpeed \* Time.deltaTime);  if (X < minX) X += maxX;  else if (X > maxX) X -= maxX;  Y -= Input.GetAxis("Mouse Y") \* (LookSpeed \* Time.deltaTime);  if (Y < minY) Y = minY;  else if (Y > maxY) Y = maxY;  Y = TargetCamera.transform.eulerAngles.x;  X = TargetCamera.transform.eulerAngles.y;  TargetCamera.transform.rotation = Quaternion.Euler(Y, X, 0.0f);  }  // Start is called before the first frame update  void Start()  {  TargetCamera = GetComponent<Camera>();  Vector3 euler = TargetCamera.transform.rotation.eulerAngles;  X = euler.x;  Y = euler.y;  StartPosition();  StartEvent?.Invoke();  }  // Update is called once per frame  void Update()  {  //-- MOVE BUTTON  if (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.LeftShift))  {  TargetCamera.transform.Translate(Vector3.forward \* Time.deltaTime \* MoveSpeed \* 0.1f);  }  else if (Input.GetKey(KeyCode.W))  {  TargetCamera.transform.Translate(Vector3.forward \* Time.deltaTime \* MoveSpeed);  }  if (Input.GetKey(KeyCode.A) && Input.GetKey(KeyCode.LeftShift))  {  TargetCamera.transform.Translate(Vector3.left \* Time.deltaTime \* MoveSpeed \* 0.1f);  }  else if (Input.GetKey(KeyCode.A))  {  TargetCamera.transform.Translate(Vector3.left \* Time.deltaTime \* MoveSpeed);  }  if (Input.GetKey(KeyCode.S) && Input.GetKey(KeyCode.LeftShift))  {  TargetCamera.transform.Translate(Vector3.back \* Time.deltaTime \* MoveSpeed \* 0.1f);  }  else if (Input.GetKey(KeyCode.S))  {  TargetCamera.transform.Translate(Vector3.back \* Time.deltaTime \* MoveSpeed);  }  if (Input.GetKey(KeyCode.D) && Input.GetKey(KeyCode.LeftShift))  {  TargetCamera.transform.Translate(Vector3.right \* Time.deltaTime \* MoveSpeed \* 0.1f);  }  else if (Input.GetKey(KeyCode.D))  {  TargetCamera.transform.Translate(Vector3.right \* Time.deltaTime \* MoveSpeed);  }  //-- LOOK BUTTON  if ((Input.GetKey(MouseLookButton)))  {  const float minX = 0.0f;  const float maxX = 360.0f;  const float minY = -90.0f;  const float maxY = 90.0f;  X += Input.GetAxis("Mouse X") \* (LookSpeed \* Time.deltaTime);  if (X < minX) X += maxX;  else if (X > maxX) X -= maxX;  Y -= Input.GetAxis("Mouse Y") \* (LookSpeed \* Time.deltaTime);  if (Y < minY) Y = minY;  else if (Y > maxY) Y = maxY;  TargetCamera.transform.rotation = Quaternion.Euler(Y, X, 0.0f);  }  //-- PAN BUTTON  if (Input.GetMouseButtonDown(MousePanButton))  {  lastPosition = Input.mousePosition;  }  if (Input.GetMouseButton(MousePanButton))  {  Vector3 delta = Input.mousePosition - lastPosition;  TargetCamera.transform.Translate(-delta.x \* PanSpeed \* Time.deltaTime, -delta.y \* PanSpeed \* Time.deltaTime, 0);  lastPosition = Input.mousePosition;  }  //-- ZOOM BUTTON  if (Input.GetAxis("Mouse ScrollWheel") > 0)  {  if (TargetCamera.orthographic)  {  if (TargetCamera.orthographicSize > 10)  {  TargetCamera.orthographicSize -= 1 \* ScrollSpeed;  }  }  else  {  TargetCamera.transform.Translate(Vector3.forward \* ScrollSpeed \* Time.deltaTime);  }  }  if (Input.GetAxis("Mouse ScrollWheel") < 0)  {  if (TargetCamera.orthographic)  {  TargetCamera.orthographicSize += 1 \* ScrollSpeed;  }  else  {  TargetCamera.transform.Translate(Vector3.back \* ScrollSpeed \* Time.deltaTime);  }  }    UpdateEvent?.Invoke();  }  static float ClampAngle(float angle, float min, float max)  {  if (angle < -360)  angle += 360;  if (angle > 360)  angle -= 360;  return Mathf.Clamp(angle, min, max);  }  } |
| --- |

Skrip "Mouse Isometric" di Unity mengimplementasikan kontrol kamera yang memungkinkan pengguna untuk menggerakkan kamera secara isometrik seperti di Unity Editor. Dengan gerakan mouse, skrip ini menghasilkan pergerakan kamera yang sejajar dengan bidang datar, menciptakan pandangan perspektif isometrik dalam permainan 3D.

### 1.7 Mouse Toggle

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class MouseToggle : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent OnEvent;  public UnityEvent OffEvent;  private Toggle CurrentToggle;  int ToInt(bool aValue)  {  if (aValue) return 1;  else return 0;  }  bool ToBool(int aValue)  {  if (aValue == 1) return true; else return false;  }  // Start is called before the first frame update  void Start()  {  CurrentToggle = GetComponent<Toggle>();  if (PlayerPrefs.HasKey(transform.name))  {  int temp = PlayerPrefs.GetInt(transform.name);  CurrentToggle.isOn = ToBool(temp);  InvokeToggle();  }  CurrentToggle.onValueChanged.AddListener((value) =>  {  PlayerPrefs.SetInt(transform.name, ToInt(value));  if (value)  {  OnEvent.Invoke();  }  else  {  OffEvent.Invoke();  }  });  StartEvents?.Invoke();  }  public void InvokeToggle()  {  int temp = PlayerPrefs.GetInt(transform.name);  PlayerPrefs.SetInt(transform.name, temp);  if (temp == 1)  {  OnEvent.Invoke();  }  else  {  OffEvent.Invoke();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini mengendalikan perilaku suatu objek dalam permainan berdasarkan input dari pemain menggunakan tombol toggle (Toggle UI dari Unity). Penjelasan script:

* Header("Event Settings"): Ini adalah atribut yang digunakan untuk memberikan judul header dalam editor Unity. Di bawah header ini, terdapat beberapa properti publik untuk mengatur peristiwa yang akan terjadi.
* public UnityEvent StartEvents: Ini adalah UnityEvent yang dapat digunakan untuk menambahkan tindakan yang akan dijalankan saat objek ini mulai aktif.
* public UnityEvent UpdateEvents: Ini adalah UnityEvent yang dapat digunakan untuk menambahkan tindakan yang akan dijalankan dalam setiap frame pembaruan.
* public UnityEvent OnEvent: Ini adalah UnityEvent yang akan diaktifkan saat tombol toggle diaktifkan (ON).
* public UnityEvent OffEvent: Ini adalah UnityEvent yang akan diaktifkan saat tombol toggle dimatikan (OFF).
* private Toggle CurrentToggle: Ini adalah referensi ke komponen Toggle yang terpasang pada objek yang sama dengan script ini. Ini digunakan untuk mengendalikan status aktif/nonaktif tombol toggle.
* int ToInt(bool aValue): Ini adalah metode yang mengubah nilai boolean menjadi nilai integer. Jika aValue adalah true, metode ini mengembalikan 1, dan jika false, mengembalikan 0.
* bool ToBool(int aValue): Ini adalah metode yang mengubah nilai integer menjadi nilai boolean. Jika aValue adalah 1, metode ini mengembalikan true, dan jika tidak, mengembalikan false.
* void Start(): Ini adalah metode yang dipanggil saat objek ini pertama kali diaktifkan. Di dalamnya:
* CurrentToggle diinisialisasi dengan komponen Toggle yang terpasang pada objek ini.
* Nilai dari Toggle ini diperiksa dalam PlayerPrefs berdasarkan nama objek ini. PlayerPrefs adalah cara untuk menyimpan data yang dapat diakses kembali di sesi permainan selanjutnya. Jika data tersedia (dalam hal ini, nilai toggle yang disimpan dalam bentuk integer), maka nilai Toggle diatur sesuai dengan data tersebut dan InvokeToggle() dipanggil.
* Sebuah listener ditambahkan ke Toggle untuk memantau perubahan nilai Toggle. Ketika nilai Toggle berubah, PlayerPrefs diperbarui dengan nilai yang sesuai dan UnityEvent OnEvent atau OffEvent dipanggil tergantung pada nilai Toggle yang baru.
* Terakhir, StartEvents, jika ada, diaktifkan.
* public void InvokeToggle(): Ini adalah metode yang digunakan untuk memicu ulang Toggle berdasarkan nilai yang disimpan dalam PlayerPrefs. Ini memungkinkan pemain untuk mengatur ulang Toggle ke status sebelumnya.
* void Update(): Ini adalah metode yang dipanggil setiap frame dalam permainan. Di dalamnya, UpdateEvents (jika ada) diaktifkan, yang memungkinkan Anda untuk menambahkan tindakan yang akan dijalankan dalam setiap frame. Namun, dalam kode yang diberikan, UpdateEvents tidak digunakan, dan metode ini hanya digunakan untuk memicu UnityEvent tersebut.

### 1.8 Mouse Slider

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class MouseSlider : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent ValueEvent;  private Slider CurrentSlider;  // Start is called before the first frame update  void Start()  {  CurrentSlider = GetComponent<Slider>();  if (PlayerPrefs.HasKey(transform.name))  {  float temp = PlayerPrefs.GetFloat(transform.name);  CurrentSlider.value = temp;  InvokeSlider();  }  CurrentSlider.onValueChanged.AddListener((value) =>  {  PlayerPrefs.SetFloat(transform.name, CurrentSlider.value);  ValueEvent.Invoke();  });  StartEvents?.Invoke();  }  public void InvokeSlider()  {  if (PlayerPrefs.HasKey(transform.name))  {  float temp = PlayerPrefs.GetFloat(transform.name);  CurrentSlider.value = temp;  }  ValueEvent.Invoke();  }  public void InvokeAudio(AudioSource audio)  {  audio.volume = CurrentSlider.value;  ValueEvent.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini mengendalikan perilaku suatu objek Slider dalam permainan, yang memungkinkan pemain untuk mengatur suatu nilai dengan menggeser slider. Penjelasan Script:

* Header("Event Settings"): Ini adalah atribut yang digunakan untuk memberikan judul header dalam editor Unity. Di bawah header ini, terdapat beberapa properti publik untuk mengatur peristiwa yang akan terjadi.
* public UnityEvent StartEvents: Ini adalah UnityEvent yang dapat digunakan untuk menambahkan tindakan yang akan dijalankan saat objek ini mulai aktif.
* public UnityEvent UpdateEvents: Ini adalah UnityEvent yang dapat digunakan untuk menambahkan tindakan yang akan dijalankan dalam setiap frame pembaruan.
* public UnityEvent ValueEvent: Ini adalah UnityEvent yang akan diaktifkan saat nilai Slider berubah. Ini memungkinkan Anda untuk menambahkan tindakan yang akan dijalankan saat nilai Slider berubah.
* private Slider CurrentSlider: Ini adalah referensi ke komponen Slider yang terpasang pada objek yang sama dengan script ini. Ini digunakan untuk mengendalikan nilai dari Slider.
* void Start(): Ini adalah metode yang dipanggil saat objek ini pertama kali diaktifkan. Di dalamnya:
* CurrentSlider diinisialisasi dengan komponen Slider yang terpasang pada objek ini.
* Nilai dari Slider ini diperiksa dalam PlayerPrefs berdasarkan nama objek ini. PlayerPrefs adalah cara untuk menyimpan data yang dapat diakses kembali di sesi permainan selanjutnya. Jika data tersedia (dalam hal ini, nilai Slider yang disimpan dalam bentuk float), maka nilai Slider diatur sesuai dengan data tersebut dan InvokeSlider() dipanggil.
* Sebuah listener ditambahkan ke Slider untuk memantau perubahan nilai Slider. Ketika nilai Slider berubah, PlayerPrefs diperbarui dengan nilai yang sesuai dan UnityEvent ValueEvent dipanggil.
* public void InvokeSlider(): Ini adalah metode yang digunakan untuk memicu ulang nilai Slider berdasarkan nilai yang disimpan dalam PlayerPrefs. Ini memungkinkan pemain untuk mengatur ulang nilai Slider ke status sebelumnya.
* public void InvokeAudio(AudioSource audio): Ini adalah metode yang digunakan untuk mengatur volume suara (AudioSource) berdasarkan nilai Slider saat ini. Metode ini mengambil komponen AudioSource sebagai argumen dan mengatur volume sesuai dengan nilai Slider saat ini. Ini memungkinkan pengendalian suara dengan menggunakan Slider.
* void Update(): Ini adalah metode yang dipanggil setiap frame dalam permainan. Di dalamnya, UpdateEvents (jika ada) diaktifkan, yang memungkinkan Anda untuk menambahkan tindakan yang akan dijalankan dalam setiap frame. Namun, dalam kode yang diberikan, UpdateEvents tidak digunakan, dan metode ini hanya digunakan untuk memicu UnityEvent tersebut.

### 1.9 Mouse DragDrop

| using UnityEngine;  using UnityEngine.UI;  using UnityEngine.EventSystems;  using UnityEngine.Events;  public class MouseDragDrop : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler  {  [Header("Drag Drop Speed")]  public float speed = 1f;  public string TargetTag;  bool canDrag = true;  [Header("Drag Drop Setting")]  public bool autoSnap;  public UnityEvent DragEvent;  public UnityEvent DropEvent;  [Header("Trigger Setting")]  public UnityEvent TriggerEnterEvent;  public UnityEvent TriggerStayEvent;  public UnityEvent TriggerExitEvent;  private Image image;  private RectTransform rectTransform;  private Canvas canvas;  private CanvasGroup canvasGroup;  private void Awake()  {  image = GetComponent<Image>();  rectTransform = GetComponent<RectTransform>();  canvas = GetComponentInParent<Canvas>();  canvasGroup = GetComponent<CanvasGroup>();  if (image == null)  {  image = gameObject.AddComponent<Image>();  }  if (rectTransform == null)  {  rectTransform = gameObject.GetComponent<RectTransform>();  if (rectTransform == null)  {  rectTransform = gameObject.AddComponent<RectTransform>();  }  }  if (canvasGroup == null)  {  canvasGroup = gameObject.AddComponent<CanvasGroup>();  }  }  void Start()  {  Rigidbody2D rb2D = gameObject.AddComponent<Rigidbody2D>();  rb2D.constraints = RigidbodyConstraints2D.FreezePositionX | RigidbodyConstraints2D.FreezePositionY;  // Menambahkan komponen BoxCollider2D  BoxCollider2D boxCollider2D = gameObject.AddComponent<BoxCollider2D>();  // Mengatur ukuran BoxCollider2D sesuai dengan ukuran UIImage  boxCollider2D.size = new Vector2(rectTransform.rect.width, rectTransform.rect.height);  boxCollider2D.isTrigger = true;  }  void Update()  {  if (Input.GetKeyUp(KeyCode.Mouse0))  {  canDrag = true;  }  }  public void OnBeginDrag(PointerEventData eventData)  {  canvasGroup.alpha = 0.6f;  canvasGroup.blocksRaycasts = false;  canDrag = true;  }  public void OnDrag(PointerEventData eventData)  {  if (canDrag)  {  rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor \* speed;  DragEvent?.Invoke();  }  }  public void OnEndDrag(PointerEventData eventData)  {  canvasGroup.alpha = 1f;  canvasGroup.blocksRaycasts = true;  DropEvent?.Invoke();  }  void OnTriggerEnter2D(Collider2D other)  {  if (autoSnap)  {  canDrag = false;  transform.position = other.transform.position;  }  TriggerEnterEvent?.Invoke();  }  void OnTriggerStay2D(Collider2D other)  {  TriggerStayEvent?.Invoke();  }  void OnTriggerExit2D(Collider2D other)  {  TriggerExitEvent?.Invoke();  }  } |
| --- |

Script ini adalah komponen untuk mengatur drag-and-drop objek menggunakan Unity dengan UI (User Interface) dan berinteraksi dengan berbagai event yang terjadi selama proses drag-and-drop, serta berinteraksi dengan objek lain yang memenuhi kondisi tertentu.

* Variabel Public:

speed: Mengontrol kecepatan perpindahan objek saat di-drag.

autoSnap: Mengaktifkan atau menonaktifkan fungsi auto snap objek ke target saat bersentuhan.

DragEvent: UnityEvent yang dapat diaktifkan saat objek sedang di-drag.

DropEvent: UnityEvent yang dapat diaktifkan saat objek di-drop ke area tertentu.

TargetTag: Tag yang akan digunakan untuk mencari objek target saat objek ini bersentuhan dengan objek lain.

TriggerEnterEvent: UnityEvent yang dapat diaktifkan saat objek ini memasuki area objek lain dengan tag yang sesuai.

TriggerStayEvent: UnityEvent yang dapat diaktifkan saat objek ini berada dalam area objek lain dengan tag yang sesuai.

TriggerExitEvent: UnityEvent yang dapat diaktifkan saat objek ini keluar dari area objek lain dengan tag yang sesuai.

* Variabel Private:

image: Komponen Image yang digunakan untuk objek ini.

rectTransform: Komponen RectTransform yang digunakan untuk mengubah posisi objek.

canvas: Canvas yang merupakan parent dari objek ini.

canvasGroup: Komponen CanvasGroup yang digunakan untuk mengendalikan alpha dan raycast-blocking selama drag-and-drop.

* Awake() Method:

Pada awal permainan, script ini mencari atau menambahkan komponen Image, RectTransform, dan CanvasGroup pada objek ini jika belum ada.

* Start() Method:

Membuat dan mengkonfigurasi Rigidbody2D untuk mengontrol pemrosesan fisika (untuk mencegah objek ini bergerak selama drag-and-drop).

Menambahkan dan mengatur BoxCollider2D untuk deteksi sentuhan dengan objek lain dan menyesuaikan ukurannya dengan ukuran objek gambar.

* Update() Method:

Memeriksa apakah tombol kiri mouse (Mouse0) telah dilepas sehingga objek dapat di-drag lagi.

* OnBeginDrag() Method:

Dipanggil saat drag dimulai.

Mengurangi alpha pada CanvasGroup sehingga objek menjadi lebih transparan dan tidak menghalangi raycast.Memungkinkan objek untuk di-drag.

* OnDrag() Method:

Dipanggil selama drag berlangsung.

Mengubah posisi objek berdasarkan perubahan posisi pointer mouse.

Mengaktifkan event DragEvent jika objek sedang di-drag.

* OnEndDrag() Method:

Dipanggil saat drag selesai.

Mengembalikan alpha pada CanvasGroup ke nilai penuh dan mengaktifkan kembali raycast.

Mengaktifkan event DropEvent jika objek telah di-drop.

* TriggerEnter2D(), TriggerStay2D(), TriggerExit2D() Methods:

Dipanggil saat objek ini bersentuhan dengan objek lain yang memiliki collider yang sesuai dengan kondisi tertentu (dalam hal ini, objek dengan tag yang sesuai).

Jika autoSnap aktif, objek ini akan berhenti dapat di-drag dan akan menempel ke objek lain.

Mengaktifkan event TriggerEnterEvent saat objek memasuki area objek lain dengan tag yang sesuai. Mengaktifkan event TriggerStayEvent saat objek berada dalam area objek lain dengan tag yang sesuai. Mengaktifkan event TriggerExitEvent saat objek keluar dari area objek lain dengan tag yang sesuai. Script ini memungkinkan objek untuk di-drag, di-drop, dan berinteraksi dengan objek lain berdasarkan kondisi tertentu saat bersentuhan.

### 1.10 Mouse GrabRelease

| using UnityEngine;  using UnityEngine.Events;  public class MouseGrabRelease : MonoBehaviour  {  [Header("Grab Release Speed")]  public float speed = 1f;  public string TargetTag;  bool canDrag = false;  [Header("Grab Release Setting")]  public bool autoSnap;  public UnityEvent GrabEvent;  public UnityEvent ReleaseEvent;  [Header("Trigger Setting")]  public UnityEvent TriggerEnterEvent;  public UnityEvent TriggerStayEvent;  public UnityEvent TriggerExitEvent;  private Vector3 initialPosition; // Menyimpan posisi awal objek  private Vector3 screenPoint;  private Vector3 offset;  private void Start()  {  Rigidbody rb = gameObject.AddComponent<Rigidbody>();  rb.useGravity = false;  }  void Update()  {  if (Input.GetMouseButtonUp(0))  {  canDrag = false;  }  }  private void OnMouseDown()  {  canDrag = true;  initialPosition = transform.position; // Menyimpan posisi awal objek saat mouse di-klik  // Menghitung offset antara klik mouse dan posisi objek  screenPoint = Camera.main.WorldToScreenPoint(gameObject.transform.position);  offset = gameObject.transform.position - Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenPoint.z));  }  private void OnMouseDrag()  {  if (canDrag)  {  // Menghitung posisi baru berdasarkan perubahan posisi mouse  Vector3 curScreenPoint = new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenPoint.z);  Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenPoint) + offset;  transform.position = curPosition;  GrabEvent?.Invoke();  }  }  private void OnMouseUp()  {  canDrag = false;  ReleaseEvent?.Invoke();  }  private void OnTriggerEnter(Collider other)  {  if (autoSnap)  {  canDrag = false;  transform.position = other.transform.position;  }  TriggerEnterEvent?.Invoke();  }  private void OnTriggerStay(Collider other)  {  TriggerStayEvent?.Invoke();  }  private void OnTriggerExit(Collider other)  {  TriggerExitEvent?.Invoke();  }  } |
| --- |

Script ini adalah komponen untuk mengatur fungsi grab dan release (menggenggam dan melepaskan) objek dalam Unity. Dengan skrip ini, Anda dapat menggerakkan objek saat menekan mouse (grab) dan mengembalikannya ke posisi awal saat melepaskan mouse (release). Berikut adalah penjelasan komponen utama dalam script ini:

* Variabel Public:

speed: Menentukan kecepatan perpindahan objek saat di-grab.

TargetTag: Tag yang akan digunakan untuk berinteraksi dengan objek lain yang memiliki tag yang sesuai.

autoSnap: Mengaktifkan atau menonaktifkan fungsi auto snap objek ke target saat bersentuhan.

GrabEvent: UnityEvent yang dapat diaktifkan saat objek di-grab.

ReleaseEvent: UnityEvent yang dapat diaktifkan saat objek di-release.

* Variabel Private:

initialPosition: Menyimpan posisi awal objek saat mouse pertama kali ditekan.

screenPoint: Menyimpan posisi dalam koordinat layar saat objek di-grab.

offset: Menyimpan offset antara posisi klik mouse dan posisi objek saat di-grab.

canDrag: Menentukan apakah objek sedang di-grab atau tidak.

* Start() Method:

Pada awal permainan, script ini menambahkan komponen Rigidbody ke objek. Rigidbody digunakan untuk mengontrol pergerakan objek selama grab dan release. useGravity diatur ke false untuk menghindari pengaruh gravitasi pada objek.

* Update() Method:

Memeriksa apakah tombol kiri mouse (Mouse0) telah dilepas sehingga objek dapat di-release.

* OnMouseDown() Method:

Dipanggil saat mouse pertama kali ditekan pada objek.

Menyimpan posisi awal objek saat mouse di-klik.

Menghitung offset antara posisi klik mouse dan posisi objek untuk memastikan objek tetap di bawah cursor mouse.

* OnMouseDrag() Method:

Dipanggil saat mouse ditarik (drag) pada objek.

Menghitung posisi baru berdasarkan perubahan posisi mouse.

Mengaktifkan event GrabEvent jika objek sedang di-grab.

* OnMouseUp() Method:

Dipanggil saat tombol mouse dilepas setelah objek di-grab.

Mengaktifkan event ReleaseEvent dan menghentikan objek dari pergerakan jika ada.

* OnTriggerEnter(), OnTriggerStay(), OnTriggerExit() Methods:

Dipanggil saat objek ini bersentuhan dengan objek lain yang memiliki collider yang sesuai dengan kondisi tertentu (dalam hal ini, objek dengan tag yang sesuai).

Jika autoSnap aktif, objek ini akan berhenti dapat di-grab dan akan menempel ke objek lain.

Mengaktifkan event TriggerEnterEvent saat objek memasuki area objek lain dengan tag yang sesuai. Mengaktifkan event TriggerStayEvent saat objek berada dalam area objek lain dengan tag yang sesuai. Mengaktifkan event TriggerExitEvent saat objek keluar dari area objek lain dengan tag yang sesuai. Dengan script ini, Anda dapat mengimplementasikan fungsi grab dan release objek yang dapat digunakan dalam permainan Anda. Pastikan objek memiliki komponen Collider yang sesuai agar trigger berfungsi dengan baik.

## 2. Keyboard

Keyboard merupakan perangkat input yang sangat esensial, terutama untuk permainan komputer. Bentuk-bentuk variasi inputnya meliputi penggunaan tombol-tombol khusus untuk berbagai aksi dalam game, misalnya tombol 'W', 'A', 'S', 'D' untuk menggerakkan karakter ke depan, ke kiri, ke belakang, dan ke kanan, atau tombol 'space' untuk melompat. Selain itu, tombol-tombol fungsi (F1, F2, dll.) dan tombol angka bisa digunakan untuk mengakses menu cepat, mengganti senjata, atau menggunakan item. Beberapa game juga memungkinkan pemain untuk melakukan kombinasi tombol tertentu untuk melakukan aksi khusus, misalnya melakukan combo serangan dalam game pertarungan. Keyboard juga sering digunakan untuk mengakses menu permainan, mengatur opsi, dan melakukan pencarian dalam game. Dengan mengoptimalkan penggunaan keyboard, pengembang game dapat menyediakan kontrol yang lebih fleksibel dan responsif bagi pemain.

### 2.1 Keyboard Press

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class KeyboardPress : MonoBehaviour  {  [Header("Key Settings")]  public KeyCode TargetKey;  public float Cooldown;  bool usingCooldown = false;  bool isCooldown = false;  [Header("Event Settings")]  public UnityEvent OnKeyDown;  public UnityEvent OnKeyPress;  public UnityEvent OnKeyUp;  // Start is called before the first frame update  void Start()  {  if (Cooldown > 0)  {  usingCooldown = true;  }  }  // Update is called once per frame  void Update()  {  InvokeKey();  }  void StartCooldown()  {  isCooldown = false;  }  void InvokeKey()  {  if (Input.GetKeyDown(TargetKey))  {  if (usingCooldown)  {  if (!isCooldown)  {  OnKeyDown.Invoke();  isCooldown = true;  Invoke("StartCooldown", Cooldown);  }  }  else  {  OnKeyDown.Invoke();  }  }  if (Input.GetKey(TargetKey))  {  if (usingCooldown)  {  if (!isCooldown)  {  OnKeyPress.Invoke();  isCooldown = true;  Invoke("StartCooldown", Cooldown);  }  }  else  {  OnKeyPress.Invoke();  }  }  if (Input.GetKeyUp(TargetKey))  {  if (usingCooldown)  {  if (!isCooldown)  {  OnKeyUp.Invoke();  isCooldown = true;  Invoke("StartCooldown", Cooldown);  }  }  else  {  OnKeyUp.Invoke();  }  }  }  } |
| --- |

Script ini adalah sebuah komponen dalam game yang digunakan untuk mendeteksi input dari keyboard dan memicu event-event tertentu berdasarkan input tersebut. Mari kita bahas lebih detail:

* Key Settings:

TargetKey: Ini adalah tombol keyboard yang akan dipantau. Anda dapat mengaturnya melalui Inspector di Unity Editor.

Cooldown: Ini adalah waktu cooldown (dalam detik) yang mengatur berapa sering event-event dapat dipicu. Jika nilainya lebih dari 0, maka akan ada cooldown setelah event pertama dipicu.

* Event Settings:

OnKeyDown: Ini adalah event Unity yang akan dipicu ketika tombol yang ditentukan (TargetKey) ditekan.

OnKeyPress: Ini adalah event Unity yang akan dipicu ketika tombol yang ditentukan (TargetKey) ditekan dan tetap ditekan (tahan).

OnKeyUp: Ini adalah event Unity yang akan dipicu ketika tombol yang ditentukan (TargetKey) dilepas setelah ditekan sebelumnya.

* Start():

Di dalam method Start(), script ini memeriksa apakah Cooldown lebih besar dari 0. Jika ya, maka usingCooldown diatur menjadi true, yang berarti cooldown akan digunakan.

* Update():

Di dalam method Update(), InvokeKey() dipanggil setiap frame. Ini berarti bahwa setiap frame, script akan memeriksa status tombol keyboard yang ditentukan dan memicu event yang sesuai.

* StartCooldown():

Ini adalah method yang digunakan untuk mengatur ulang status cooldown (isCooldown) menjadi false. Ini akan dipanggil setelah cooldown selesai.

* InvokeKey():

Method ini digunakan untuk memeriksa status tombol keyboard yang ditentukan dan memicu event yang sesuai. Jika tombol TargetKey ditekan pertama kali (Input.GetKeyDown(TargetKey)), maka OnKeyDown.Invoke() akan dipanggil.

Jika usingCooldown aktif, maka cooldown akan dimulai dengan mengatur isCooldown menjadi true dan memanggil StartCooldown() untuk mengatur ulang cooldown setelah Cooldown detik. Jika tombol TargetKey tetap ditekan (Input.GetKey(TargetKey)), maka OnKeyPress.Invoke() akan dipanggil dengan logika yang mirip dengan di atas.

Jika tombol TargetKey dilepas setelah ditekan sebelumnya (Input.GetKeyUp(TargetKey)), maka OnKeyUp.Invoke() akan dipanggil dengan logika yang mirip dengan di atas. Dengan script ini, Anda dapat dengan mudah menambahkan respons berdasarkan input dari keyboard untuk keperluan tertentu dalam game Unity Anda.

### 2.2 Keyboard Switch

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class KeyboardSwitch : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Switch Settings")]  public KeyCode InputKey;  public UnityEvent OnEvent;  public UnityEvent OffEvent;  bool isOn = false;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  if (!isOn)  {  if (Input.GetKeyDown(InputKey))  {  isOn = true;  OnEvent?.Invoke();  }  }  else  {  if (Input.GetKeyDown(InputKey))  {  isOn = false;  OffEvent?.Invoke();  }  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini adalah komponen dalam game Unity yang digunakan untuk mengontrol dua set event berdasarkan input dari keyboard, yang bisa digunakan untuk mengaktifkan atau menonaktifkan suatu tindakan atau efek dalam permainan. Mari kita bahas script ini lebih detail:

* Event Settings:

StartEvents: Ini adalah event Unity yang akan dipicu pada saat komponen ini dimulai (pada metode Start()). Event ini bisa digunakan untuk menginisialisasi sesuatu saat komponen dimulai.

UpdateEvents: Ini adalah event Unity yang akan dipicu setiap frame dalam metode Update(). Anda bisa menggunakan event ini untuk melakukan tindakan yang harus diperbarui setiap frame.

* Switch Settings:

InputKey: Ini adalah tombol keyboard yang akan digunakan untuk mengaktifkan dan menonaktifkan event-event. Anda mengaturnya melalui Inspector di Unity Editor.

OnEvent: Ini adalah event Unity yang akan dipicu saat tombol InputKey ditekan untuk mengaktifkan sesuatu.

OffEvent: Ini adalah event Unity yang akan dipicu saat tombol InputKey ditekan lagi untuk menonaktifkan sesuatu.

isOn: Ini adalah variabel boolean yang menunjukkan apakah "switch" (atau "toggle") sedang dalam posisi "On" atau "Off." Awalnya diatur sebagai false, yang berarti "Off."

* Start():

Di dalam method Start(), event StartEvents dipicu. Ini memungkinkan Anda untuk menjalankan tindakan awal ketika komponen ini dimulai.

* Update():

Di dalam method Update(), script ini memeriksa status isOn untuk menentukan apakah "switch" dalam posisi "On" atau "Off."

Jika isOn adalah false (artinya dalam posisi "Off") dan tombol InputKey ditekan (Input.GetKeyDown(InputKey)), maka isOn diubah menjadi true (posisi "On") dan event OnEvent dipicu. Ini mengaktifkan tindakan yang sesuai.

Jika isOn adalah true (artinya dalam posisi "On") dan tombol InputKey ditekan lagi, maka isOn diubah menjadi false (posisi "Off") dan event OffEvent dipicu. Ini menonaktifkan tindakan yang sesuai. Terlepas dari apakah "switch" dalam posisi "On" atau "Off," event UpdateEvents selalu dipicu setiap frame. Anda dapat menggunakannya untuk melakukan pembaruan yang diperlukan setiap frame. Dengan script ini, Anda dapat mengontrol berbagai tindakan dalam permainan berdasarkan input dari keyboard, yang bisa diaktifkan dan dinonaktifkan dengan tombol yang ditentukan.

### 2.3 Keyboard Tab

| using UnityEngine;  using UnityEngine.UI;  using UnityEngine.EventSystems;  public class KeyboardTab : MonoBehaviour  {  public InputField[] inputFields;  private void Update()  {  if (Input.GetKeyDown(KeyCode.Tab))  {  Selectable currentSelectable = EventSystem.current.currentSelectedGameObject.GetComponent<Selectable>();  if (currentSelectable != null)  {  int currentIndex = System.Array.IndexOf(inputFields, currentSelectable);  if (currentIndex >= 0)  {  int nextIndex = (currentIndex + 1) % inputFields.Length;  inputFields[nextIndex].Select();  }  }  }  }  } |
| --- |

Script ini adalah komponen dalam game Unity yang digunakan untuk mengatur navigasi antara InputField menggunakan tombol "Tab" pada keyboard. Ini memungkinkan pemain untuk dengan cepat beralih antara input field dengan tombol "Tab" seperti yang biasa terjadi dalam formulir atau antarmuka pengguna. Mari kita bahas script ini lebih detail:

* public InputField[] inputFields: Ini adalah array InputField yang harus diisi dengan semua InputField yang ingin Anda navigasikan menggunakan tombol "Tab". Anda mengaturnya melalui Inspector di Unity Editor.
* Update(): Metode Update() dijalankan setiap frame dan digunakan untuk memeriksa apakah tombol "Tab" pada keyboard ditekan.
* Pemeriksaan if (Input.GetKeyDown(KeyCode.Tab)):

Ini memeriksa apakah tombol "Tab" pada keyboard ditekan.

* Selectable currentSelectable = EventSystem.current.currentSelectedGameObject. GetComponent<Selectable>();:

Ini mendapatkan Selectable (komponen yang dapat dipilih) yang saat ini terpilih dalam EventSystem. EventSystem adalah sistem yang digunakan untuk mengelola interaksi permainan, termasuk pemilihan objek saat ini.

* Pemeriksaan if (currentSelectable != null):

Ini memeriksa apakah ada Selectable yang terpilih. Jika tidak ada, tidak ada navigasi yang diperlukan.

* int currentIndex = System.Array.IndexOf(inputFields, currentSelectable):

Ini mencari indeks dari currentSelectable dalam array inputFields. Ini membantu untuk mengetahui InputField mana yang saat ini terpilih.

* Pemeriksaan if (currentIndex >= 0):
* Ini memeriksa apakah currentSelectable ditemukan dalam array inputFields. Jika ditemukan, maka navigasi "Tab" akan dilakukan. int nextIndex = (currentIndex + 1) %
* inputFields.Length;

Ini menghitung indeks berikutnya dalam array inputFields untuk navigasi "Tab." Jika kita sudah di akhir array, ini akan membungkus kembali ke indeks awal.

* inputFields[nextIndex].Select();:

Ini mengatur InputField berikutnya sebagai yang terpilih dalam EventSystem. Dengan cara ini, pemain dapat beralih ke InputField berikutnya dengan tombol "Tab" setelah tombol "Tab" ditekan.Dengan script ini, Anda dapat dengan mudah mengatur navigasi antara InputField menggunakan tombol "Tab" dalam game Unity Anda, yang dapat meningkatkan pengalaman pengguna dan kemudahan pengisian formulir atau interaksi dalam permainan.

## 3. Gamepad

Gamepad adalah perangkat input yang sering digunakan untuk konsol permainan, namun juga kompatibel dengan banyak permainan PC. Gamepad biasanya memiliki tombol-tombol digital, stick analog, dan tombol bahu (shoulder buttons). Tombol-tombol digital digunakan untuk berbagai aksi dasar seperti melompat, menyerang, atau berinteraksi dengan objek. Stick analog digunakan untuk menggerakkan karakter atau kamera dalam game dengan presisi yang lebih tinggi dibandingkan tombol digital. Tombol bahu digunakan untuk berbagai fungsi, misalnya mengganti senjata, mengaktifkan perisai, atau mengakses menu cepat. Beberapa gamepad modern juga memiliki fitur vibrasi yang memberikan umpan balik haptik kepada pemain berdasarkan kejadian dalam game, misalnya saat karakter terkena serangan. Dengan mengoptimalkan penggunaan gamepad, pengembang dapat menciptakan pengalaman bermain yang lebih nyaman dan imersif, terutama untuk permainan yang memerlukan kontrol yang lebih kompleks dan presisi.

### 3.1 Gamepad Axis

| using UnityEngine;  using UnityEngine.Events;  public class GamepadAxis : MonoBehaviour  {  [Header("Axis Settings")]  public string TargetAxis;  public float AxisThreshold = 0.5f;  [Header("Event Settings")]  public UnityEvent OnAxisPositive;  public UnityEvent OnAxisNegative;  public UnityEvent OnAxisNeutral;  // Update is called once per frame  void Update()  {  float axisValue = Input.GetAxis(TargetAxis);  if (axisValue > AxisThreshold)  {  OnAxisPositive.Invoke();  }  else if (axisValue < -AxisThreshold)  {  OnAxisNegative.Invoke();  }  else  {  OnAxisNeutral.Invoke();  }  }  } |
| --- |

Berikut adalah perubahan yang telah dilakukan pada script:

* TargetAxis: Ini adalah string yang mendefinisikan axis pada gamepad yang akan dipantau. Anda dapat mengaturnya melalui Inspector di Unity Editor. Contohnya, jika Anda ingin memantau Axis horizontal (misalnya, left stick horizontal) pada gamepad, Anda akan mengatur TargetAxis menjadi "Horizontal."
* AxisThreshold: Ini adalah nilai ambang batas yang digunakan untuk mengidentifikasi apakah axis berada dalam posisi positif, negatif, atau netral. Nilai defaultnya adalah 0.5, tetapi Anda dapat mengubahnya sesuai dengan preferensi Anda.
* OnAxisPositive: Ini adalah event Unity yang akan dipicu ketika nilai dari axis (TargetAxis) lebih besar dari AxisThreshold, menunjukkan bahwa axis berada dalam posisi positif.
* OnAxisNegative: Ini adalah event Unity yang akan dipicu ketika nilai dari axis (TargetAxis) lebih kecil dari -AxisThreshold, menunjukkan bahwa axis berada dalam posisi negatif.
* OnAxisNeutral: Ini adalah event Unity yang akan dipicu ketika nilai dari axis (TargetAxis) berada di antara -AxisThreshold dan AxisThreshold, menunjukkan bahwa axis berada dalam posisi netral.
* Di dalam method Update(), script ini mengambil nilai dari axis gamepad yang ditentukan (TargetAxis) menggunakan Input.GetAxis(TargetAxis). Kemudian, script memeriksa nilai tersebut dengan threshold yang telah ditentukan (AxisThreshold). Jika nilai axis lebih besar dari AxisThreshold, maka event OnAxisPositive akan dipicu, menunjukkan posisi positif. Jika nilai axis lebih kecil dari -AxisThreshold, maka event OnAxisNegative akan dipicu, menunjukkan posisi negatif. Jika nilai axis berada di antara -AxisThreshold dan AxisThreshold, maka event OnAxisNeutral akan dipicu, menunjukkan posisi netral.

### 3.2 Gamepad Button

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class GamepadKey : MonoBehaviour  {  [Header("Key Settings")]  public string TargetButton;  [Header("Event Settings")]  public UnityEvent OnButtonDown;  public UnityEvent OnButtonPress;  public UnityEvent OnButtonUp;  // Update is called once per frame  void Update()  {  if (Input.GetButtonDown(TargetButton))  {  Debug.Log("Key Down!");  OnButtonDown.Invoke();  }  if (Input.GetButton(TargetButton))  {  Debug.Log("Key Pressed!");  OnButtonPress.Invoke();  }  if (Input.GetButtonUp(TargetButton))  {  Debug.Log("Key Up!");  OnButtonUp.Invoke();  }  }  } |
| --- |

Script ini adalah komponen dalam game Unity yang digunakan untuk mendeteksi input dari gamepad (atau controller) dan memicu event-event tertentu berdasarkan input tersebut. Mari kita bahas script ini lebih detail:

* Key Settings:

TargetButton: Ini adalah string yang mendefinisikan tombol pada gamepad yang akan dipantau. Anda dapat mengaturnya melalui Inspector di Unity Editor. Contohnya, jika Anda ingin memantau tombol "A" pada gamepad, Anda akan mengatur TargetButton menjadi "A."

* Event Settings:

OnButtonDown: Ini adalah event Unity yang akan dipicu ketika tombol pada gamepad yang ditentukan (TargetButton) ditekan (ke bawah).

OnButtonPress: Ini adalah event Unity yang akan dipicu ketika tombol pada gamepad yang ditentukan (TargetButton) tetap ditekan (tahan).

OnButtonUp: Ini adalah event Unity yang akan dipicu ketika tombol pada gamepad yang ditentukan (TargetButton) dilepas setelah ditekan sebelumnya.

* Start():

Di dalam method Start(), tidak ada tindakan yang diambil. Anda bisa menggunakan method ini untuk inisialisasi awal jika diperlukan.

* Update():

Di dalam method Update(), script ini memeriksa status tombol gamepad yang ditentukan (TargetButton) setiap frame dan memicu event yang sesuai.

* Pemeriksaan Tombol Down (Input.GetButtonDown(TargetButton)):

Jika tombol gamepad TargetButton ditekan pertama kali, maka akan dicetak pesan "Key Down!" ke konsol dan event OnButtonDown akan dipicu.

* Pemeriksaan Tombol Pressed (Input.GetButton(TargetButton)):

Jika tombol gamepad TargetButton tetap ditekan (tahan), maka akan dicetak pesan "Key Pressed!" ke konsol dan event OnButtonPress akan dipicu.

* Pemeriksaan Tombol Up (Input.GetButtonUp(TargetButton)):

Jika tombol gamepad TargetButton dilepas setelah ditekan sebelumnya, maka akan dicetak pesan "Key Up!" ke konsol dan event OnButtonUp akan dipicu.

* Debug Log:

Setiap kali salah satu dari tiga kondisi di atas terpenuhi, pesan teks akan dicetak ke konsol untuk tujuan debugging. Anda dapat menghapus atau menonaktifkan pesan debugging ini setelah menguji script sesuai kebutuhan.

Dengan script ini, Anda dapat dengan mudah menambahkan respons berdasarkan input dari gamepad dalam game Unity Anda, seperti mengontrol karakter atau objek dalam permainan menggunakan tombol pada gamepad.

## 4. Touch

Input touch adalah jenis input yang umum digunakan pada perangkat mobile, seperti smartphone dan tablet. Variasi input touch meliputi tap (mengetuk), swipe (menggeser), pinch (mencubit), dan drag (menarik). Tap digunakan untuk berbagai aksi dasar seperti memilih objek, menyerang, atau berinteraksi dengan elemen dalam game. Swipe digunakan untuk menggerakkan karakter atau kamera, menggulir menu, atau melakukan serangan khusus. Pinch digunakan untuk zoom in atau zoom out pandangan kamera. Drag digunakan untuk menggerakkan objek, mengatur arah serangan, atau menggulir peta. Beberapa game juga mendukung multi-touch, yang memungkinkan pemain untuk melakukan lebih dari satu aksi sekaligus dengan menggunakan lebih dari satu jari. Misalnya, pemain dapat menggerakkan karakter dengan satu jari sambil menyerang musuh dengan jari lainnya. Dengan memanfaatkan berbagai macam input touch ini, pengembang dapat menciptakan pengalaman bermain yang lebih intuitif dan responsif pada perangkat mobile.

### 4.1 Touch Tap

| using UnityEngine;  using UnityEngine.Events;  public class TouchTap : MonoBehaviour  {  [Header("Tap Event")]  public UnityEvent TrueTapEvent;  public UnityEvent FalseTapEvent;  private bool isTouching; // status touch saat ini  bool TapStatus;  void Update()  {  if (Input.touchCount > 0) // jika terdapat input touch  {  Touch touch = Input.GetTouch(0); // ambil input touch pertama  if (touch.phase == TouchPhase.Began) // jika touch dimulai  {  Ray ray = Camera.main.ScreenPointToRay(touch.position); // buat ray dari posisi touch  RaycastHit hit;  if (Physics.Raycast(ray, out hit)) // lakukan raycast  {  if (hit.collider == GetComponent<Collider>()) // jika objek yang terkena raycast memiliki Collider yang sama dengan script ini  {  isTouching = true; // ubah status touch menjadi aktif  }  }  }  else if (touch.phase == TouchPhase.Ended) // jika touch berakhir  {  TapStatus = !TapStatus;  if (isTouching) // jika touch sebelumnya aktif  {  if (TapStatus)  {  TrueTapEvent.Invoke(); // jalankan event onTouch  } else  {  FalseTapEvent.Invoke(); // jalankan event onTouch  }  }  isTouching = false; // ubah status touch menjadi tidak aktif  }  }  }  } |
| --- |

Berikut penjelasan komponen utama dari script ini:

* public UnityEvent TrueTapEvent; dan public UnityEvent FalseTapEvent;: Ini adalah dua peristiwa (events) Unity yang dapat dihubungkan dengan skrip ini melalui antarmuka pengguna Unity. TrueTapEvent akan diaktifkan ketika tap terdeteksi, sementara FalseTapEvent akan diaktifkan ketika tap terdeteksi setelah tap sebelumnya. Anda dapat menghubungkan fungsi-fungsi tertentu ke peristiwa-peristiwa ini dalam editor Unity.
* private bool isTouching;: Variabel ini digunakan untuk melacak apakah layar sedang disentuh saat ini atau tidak. Nilainya akan diubah menjadi true ketika sentuhan pertama kali terdeteksi pada objek dengan collider yang sesuai.
* bool TapStatus;: Ini adalah variabel boolean yang digunakan untuk melacak status tap sebelumnya. Ini akan mengganti nilai antara true dan false setiap kali sentuhan berakhir. Hal ini digunakan untuk membedakan antara dua jenis tap yang berbeda.
* void Update(): Ini adalah metode yang dijalankan setiap frame dalam permainan. Di dalamnya, kita akan memeriksa input sentuhan dan menangani logika ketukan.
* if (Input.touchCount > 0): Ini adalah kondisi untuk memeriksa apakah ada setidaknya satu input sentuhan yang terdeteksi.
* Touch touch = Input.GetTouch(0);: Ini digunakan untuk mendapatkan informasi tentang input sentuhan pertama (indeks 0) yang terdeteksi.
* if (touch.phase == TouchPhase.Began): Ini adalah kondisi yang memeriksa apakah sentuhan baru dimulai (touch phase adalah "Began"). Jika ya, maka kita akan melanjutkan dengan melakukan raycast untuk menentukan apakah objek yang dikenai oleh sentuhan adalah objek yang memiliki collider yang sama dengan objek yang memuat skrip ini.
* Ray ray = Camera.main.ScreenPointToRay(touch.position);: Ini membuat sinar (ray) dari posisi sentuhan pada layar ke dalam dunia game. Ray ini akan digunakan untuk melakukan raycast.
* if (Physics.Raycast(ray, out hit)): Ini adalah operasi raycast yang dilakukan untuk mengecek apakah ray tersebut bertemu dengan objek dalam permainan.
* if (hit.collider == GetComponent<Collider>()): Ini memeriksa apakah objek yang terkena raycast memiliki collider yang sama dengan objek yang memuat skrip ini. Jika iya, maka isTouching akan diubah menjadi true, menunjukkan bahwa objek sedang disentuh.
* else if (touch.phase == TouchPhase.Ended): Ini adalah kondisi yang memeriksa apakah sentuhan telah berakhir (touch phase adalah "Ended"). Jika ya, maka kita akan melanjutkan dengan mengubah TapStatus dan memeriksa apakah sentuhan sebelumnya aktif atau tidak.
* if (isTouching): Jika sentuhan sebelumnya adalah aktif (berarti objek ini sedang disentuh), maka script akan memeriksa TapStatus dan akan memicu peristiwa yang sesuai (TrueTapEvent jika TapStatus adalah true, dan FalseTapEvent jika TapStatus adalah false). Setelah itu, isTouching akan diubah menjadi false untuk menunjukkan bahwa sentuhan sudah berakhir.

Dengan demikian, script ini memungkinkan Anda untuk mengenali ketukan pada objek dengan collider tertentu dan menjalankan peristiwa berdasarkan jenis ketukan yang dikenali.

### 4.2 Touch Look

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class TouchLook : MonoBehaviour  {  [Header("Touch Settings")]  public bool TouchLookEnabled;  [Header("Movement Settings")]  [SerializeField] private float sensitivity = 0.1f; // sensitivitas gerakan kamera  [SerializeField] private float smoothness = 0.5f; // kehalusan gerakan kamera  private Vector2 lastTouchPosition; // posisi touch terakhir  private Vector2 currentTouchPosition; // posisi touch saat ini  private Vector2 smoothDeltaPosition; // perpindahan touch yang telah disampirkan  void Update()  {  if (TouchLookEnabled && Input.touchCount > 0) // jika terdapat input touch  {  Touch touch = Input.GetTouch(0); // ambil input touch pertama  if (touch.phase == TouchPhase.Moved) // jika touch bergerak  {  currentTouchPosition = touch.position; // simpan posisi touch saat ini  Vector2 deltaPosition = currentTouchPosition - lastTouchPosition; // hitung perpindahan touch  smoothDeltaPosition = Vector2.Lerp(smoothDeltaPosition, deltaPosition, smoothness); // sampaikan perpindahan touch yang telah disampirkan  transform.Rotate(-smoothDeltaPosition.y \* sensitivity, 0, 0);  transform.Rotate(0, smoothDeltaPosition.x \* sensitivity, 0, Space.World); // putar kamera sesuai perpindahan touch  lastTouchPosition = currentTouchPosition; // simpan posisi touch terakhir  }  else if (touch.phase == TouchPhase.Began) // jika touch dimulai  {  lastTouchPosition = touch.position; // simpan posisi touch terakhir  smoothDeltaPosition = Vector2.zero; // reset perpindahan touch yang telah disampirkan  }  else if (touch.phase == TouchPhase.Ended) // jika touch berakhir  {  smoothDeltaPosition = Vector2.zero; // reset perpindahan touch yang telah disampirkan  }  }  }  } |
| --- |

Berikut penjelasan komponen utama dari script ini:

* public bool TouchLookEnabled;: Ini adalah boolean yang dapat diatur dari editor Unity. Jika diaktifkan (true), maka kamera akan merespons input sentuhan; jika dinonaktifkan (false), maka input sentuhan tidak akan memengaruhi pergerakan kamera.
* float sensitivity = 0.1f;: Ini adalah sensitivitas pergerakan kamera. Nilainya mengatur seberapa cepat kamera akan bergerak berdasarkan input sentuhan. Anda dapat mengatur nilai ini untuk mengendalikan kecepatan pergerakan kamera.
* float smoothness = 0.5f;: Ini adalah faktor kehalusan pergerakan kamera. Nilainya mengatur seberapa halus pergerakan kamera akan terasa. Semakin tinggi nilai ini, semakin halus pergerakan kamera.
* private Vector2 lastTouchPosition; dan private Vector2 currentTouchPosition;: Variabel-variabel ini digunakan untuk menyimpan posisi sentuhan terakhir dan posisi sentuhan saat ini.
* private Vector2 smoothDeltaPosition;: Ini adalah perpindahan sentuhan yang telah disampirkan. Perpindahan ini akan digunakan untuk menggerakkan kamera.
* void Update(): Ini adalah metode yang dijalankan setiap frame dalam permainan. Di dalamnya, kita akan memeriksa input sentuhan dan menggerakkan kamera berdasarkan input tersebut.
* if (TouchLookEnabled && Input.touchCount > 0): Ini adalah kondisi untuk memeriksa apakah pengguna mengaktifkan mode "Touch Look" dan ada setidaknya satu input sentuhan yang terdeteksi.
* Touch touch = Input.GetTouch(0);: Ini digunakan untuk mendapatkan informasi tentang input sentuhan pertama (indeks 0) yang terdeteksi.
* if (touch.phase == TouchPhase.Moved): Ini adalah kondisi yang memeriksa apakah sentuhan bergerak (touch phase adalah "Moved"). Jika ya, maka kita akan menghitung perpindahan sentuhan saat ini dari sentuhan sebelumnya dan kemudian menerapkan perpindahan ini pada kamera dengan menggunakan transform.Rotate. Ini memungkinkan kamera untuk berputar sesuai dengan arah pergerakan sentuhan.
* else if (touch.phase == TouchPhase.Began): Ini adalah kondisi yang memeriksa apakah sentuhan baru dimulai (touch phase adalah "Began"). Jika ya, maka kita akan menyimpan posisi sentuhan ini sebagai posisi terakhir dan mengatur smoothDeltaPosition ke nol. Ini memungkinkan untuk menghindari pergerakan tiba-tiba saat sentuhan dimulai.
* else if (touch.phase == TouchPhase.Ended): Ini adalah kondisi yang memeriksa apakah sentuhan telah berakhir (touch phase adalah "Ended"). Jika ya, maka smoothDeltaPosition akan diatur ke nol untuk menghentikan pergerakan kamera setelah sentuhan selesai.

Dengan script ini, Anda dapat mengaktifkan mode "Touch Look" yang memungkinkan pengguna menggerakkan kamera dengan menggesek layar sentuh. Sensitivitas dan kehalusan pergerakan dapat disesuaikan dengan preferensi pengguna.

### 4.3 Touch Orbit

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class TouchOrbit : MonoBehaviour  {  [Header("Touch Settings")]  public bool TouchZoomEnabled;  public Transform target;  [Header("Movement Settings")]  public Vector3 targetOffset;  public float distance = 5.0f;  public float maxDistance = 20;  public float minDistance = .6f;  public float xSpeed = 5.0f;  public float ySpeed = 5.0f;  public int yMinLimit = -80;  public int yMaxLimit = 80;  public float zoomRate = 10.0f;  public float panSpeed = 0.3f;  public float zoomDampening = 5.0f;  private float xDeg = 0.0f;  private float yDeg = 0.0f;  private float currentDistance;  private float desiredDistance;  private Quaternion currentRotation;  private Quaternion desiredRotation;  private Quaternion rotation;  private Vector3 position;  private Vector3 FirstPosition;  private Vector3 SecondPosition;  private Vector3 delta;  private Vector3 lastOffset;  private Vector3 lastOffsettemp;  void Start() { Init(); }  void OnEnable() { Init(); }  public void Init()  {  //If there is no target, create a temporary target at 'distance' from the cameras current viewpoint  if (!target)  {  GameObject go = new GameObject("Cam Target");  go.transform.position = transform.position + (transform.forward \* distance);  target = go.transform;  }  distance = Vector3.Distance(transform.position, target.position);  currentDistance = distance;  desiredDistance = distance;  //be sure to grab the current rotations as starting points.  position = transform.position;  rotation = transform.rotation;  currentRotation = transform.rotation;  desiredRotation = transform.rotation;  xDeg = Vector3.Angle(Vector3.right, transform.right);  yDeg = Vector3.Angle(Vector3.up, transform.up);  }  /\*  \* Camera logic on LateUpdate to only update after all character movement logic has been handled.  \*/  void LateUpdate()  {  // If Control and Alt and Middle button? ZOOM!  if (Input.touchCount == 2)  {  Touch touchZero = Input.GetTouch(0);  Touch touchOne = Input.GetTouch(1);  Vector2 touchZeroPreviousPosition = touchZero.position - touchZero.deltaPosition;  Vector2 touchOnePreviousPosition = touchOne.position - touchOne.deltaPosition;  float prevTouchDeltaMag = (touchZeroPreviousPosition - touchOnePreviousPosition).magnitude;  float TouchDeltaMag = (touchZero.position - touchOne.position).magnitude;  float deltaMagDiff = prevTouchDeltaMag - TouchDeltaMag;  desiredDistance += deltaMagDiff \* Time.deltaTime \* zoomRate \* 0.0025f \* Mathf.Abs(desiredDistance);  }  // If middle mouse and left alt are selected? ORBIT  if (Input.touchCount == 1 && Input.GetTouch(0).phase == TouchPhase.Moved)  {  Vector2 touchposition = Input.GetTouch(0).deltaPosition;  xDeg += touchposition.x \* xSpeed \* 0.002f;  yDeg -= touchposition.y \* ySpeed \* 0.002f;  yDeg = ClampAngle(yDeg, yMinLimit, yMaxLimit);  }  desiredRotation = Quaternion.Euler(yDeg, xDeg, 0);  currentRotation = transform.rotation;  rotation = Quaternion.Lerp(currentRotation, desiredRotation, Time.deltaTime \* zoomDampening);  transform.rotation = rotation;  if (Input.GetMouseButtonDown(1))  {  FirstPosition = Input.mousePosition;  lastOffset = targetOffset;  }  if (Input.GetMouseButton(1))  {  SecondPosition = Input.mousePosition;  delta = SecondPosition - FirstPosition;  targetOffset = lastOffset + transform.right \* delta.x \* 0.003f + transform.up \* delta.y \* 0.003f;  }  // affect the desired Zoom distance if we roll the scrollwheel  desiredDistance = Mathf.Clamp(desiredDistance, minDistance, maxDistance);  currentDistance = Mathf.Lerp(currentDistance, desiredDistance, Time.deltaTime \* zoomDampening);  position = target.position - (rotation \* Vector3.forward \* currentDistance);  position = position - targetOffset;  transform.position = position;  }  private static float ClampAngle(float angle, float min, float max)  {  if (angle < -360)  angle += 360;  if (angle > 360)  angle -= 360;  return Mathf.Clamp(angle, min, max);  }  } |
| --- |

Di bawah ini adalah penjelasan komponen utama dari script ini:

* public bool TouchZoomEnabled;: Ini adalah boolean yang dapat diatur dari editor Unity. Jika diaktifkan (true), maka zoom menggunakan sentuhan akan diaktifkan; jika dinonaktifkan (false), maka zoom tidak akan berfungsi.
* public Transform target;: Transform ini adalah target yang akan diorbitkan oleh kamera. Jika tidak ada target yang ditentukan dalam editor Unity, maka akan dibuat target sementara yang ditempatkan pada jarak tertentu dari posisi awal kamera.
* public Vector3 targetOffset;: Offset (perpindahan) dari target. Ini memungkinkan Anda untuk menggeser posisi target yang diorbitkan oleh kamera.
* public float distance = 5.0f;: Jarak awal kamera dari target.
* public float maxDistance = 20;: Jarak maksimum yang dapat di-zoom oleh kamera.
* public float minDistance = .6f;: Jarak minimum yang dapat di-zoom oleh kamera.
* public float xSpeed = 5.0f; dan public float ySpeed = 5.0f;: Kecepatan putaran (orbit) horizontal dan vertikal kamera.
* public int yMinLimit = -80; dan public int yMaxLimit = 80;: Batasan sudut elevasi (y-rotation) kamera. Ini memastikan kamera tidak dapat diorbitkan ke atas atau ke bawah melewati sudut yang ditentukan.
* public float zoomRate = 10.0f;: Kecepatan zoom kamera saat melakukan pinch-to-zoom (sentuhan dua jari).
* public float panSpeed = 0.3f;: Kecepatan geser (pan) kamera saat menggesek dengan satu jari.
* public float zoomDampening = 5.0f;: Dampening (perlambatan) zoom untuk memberikan efek zoom yang lebih halus.

Berbagai variabel float seperti xDeg, yDeg, currentDistance, desiredDistance, xDeg, yDeg, currentRotation, desiredRotation, rotation, dan position digunakan untuk melacak dan mengelola rotasi dan posisi kamera.

void LateUpdate(): Ini adalah metode yang dijalankan setelah semua logika pergerakan karakter selesai dijalankan. Pada bagian pertama LateUpdate(), script memeriksa input sentuh untuk zoom. Jika dua sentuhan terdeteksi, maka script akan mengukur perubahan jarak antara kedua sentuhan dan menggunakan nilai ini untuk mengatur desiredDistance (jarak zoom yang diinginkan).

Pada bagian kedua LateUpdate(), script memeriksa input sentuh untuk orbit kamera. Jika satu sentuhan terdeteksi dengan fase "Moved", maka script akan menghitung perubahan posisi sentuhan dan mengubah nilai xDeg dan yDeg untuk mengontrol rotasi kamera. Batasan sudut elevasi (y-rotation) juga diperhatikan.

Pada bagian ketiga LateUpdate(), script memeriksa input sentuh untuk menggeser (pan) kamera. Ini dilakukan dengan mengukur perubahan posisi sentuhan dan mengubah targetOffset untuk menggeser kamera.

Terakhir, script mengatur desiredDistance agar tetap berada dalam batas yang diizinkan (minDistance dan maxDistance), lalu menggunakan interpolasi linier untuk menghaluskan perubahan jarak kamera. Kemudian, posisi kamera dihitung berdasarkan rotasi dan jarak yang diinginkan, termasuk offset target, dan transformasi kamera diubah sesuai dengan posisi yang dihitung.

Dengan script ini, Anda dapat mengendalikan rotasi dan zoom kamera dengan sentuhan di perangkat seluler Anda, serta menggeser kamera dengan menggesek satu jari.

### 

### 4.4 Touch Draw

| using UnityEngine;  public class TouchDraw : MonoBehaviour  {  public float lineWidth = 0.1f;  public Color lineColor = Color.black;  private Camera \_camera;  private LineRenderer \_lineRenderer;  private RaycastHit \_hit;  private bool \_isDrawing = false;  private GameObject \_currentDrawing;  private void Start()  {  // mendapatkan referensi kamera utama  \_camera = Camera.main;  }  private void Update()  {  // membuat ray dari posisi mouse  Ray ray = \_camera.ScreenPointToRay(Input.mousePosition);  // melakukan raycasting untuk mencari posisi di mana mouse berada di dunia game  if (Physics.Raycast(ray, out \_hit))  {  // mulai menggambar ketika tombol mouse kiri ditekan  if (Input.GetMouseButtonDown(0))  {  // membuat objek baru untuk menggambar  \_currentDrawing = new GameObject("Drawing");  \_lineRenderer = \_currentDrawing.AddComponent<LineRenderer>();  // inisialisasi Line Renderer  \_lineRenderer.startWidth = lineWidth;  \_lineRenderer.endWidth = lineWidth;  \_lineRenderer.material.color = lineColor;  // menambahkan titik awal ke Line Renderer  \_lineRenderer.positionCount = 1;  \_lineRenderer.SetPosition(0, \_hit.point);  \_isDrawing = true;  }  // menambah titik baru ke Line Renderer setiap kali mouse digerakkan  if (Input.GetMouseButton(0) && \_isDrawing)  {  int positionCount = \_lineRenderer.positionCount;  \_lineRenderer.positionCount = positionCount + 1;  \_lineRenderer.SetPosition(positionCount, \_hit.point);  }  }  // berhenti menggambar ketika tombol mouse kiri dilepas  if (Input.GetMouseButtonUp(0) && \_isDrawing)  {  \_isDrawing = false;  }  }  } |
| --- |

**B**erikut penjelasan komponen utama dari script ini:

* public float lineWidth = 0.1f; dan public Color lineColor = Color.black;: Variabel ini memungkinkan pengguna untuk mengatur ketebalan garis yang akan digambar (lineWidth) dan warna garis (lineColor) melalui editor Unity.
* private Camera \_camera;: Ini adalah referensi ke kamera utama dalam permainan.
* private LineRenderer \_lineRenderer;: Ini adalah komponen LineRenderer yang akan digunakan untuk menggambar garis.
* private RaycastHit \_hit;: Ini adalah variabel yang akan digunakan untuk menyimpan informasi tentang objek yang dikenai oleh raycast (sinar) dari mouse.
* private bool \_isDrawing = false;: Ini adalah status yang menunjukkan apakah pengguna sedang dalam proses menggambar atau tidak.
* private GameObject \_currentDrawing;: Ini adalah objek sementara yang digunakan untuk menggambar garis.
* private void Start(): Ini adalah metode yang dijalankan pada awal permainan. Di dalamnya, script mendapatkan referensi ke kamera utama dengan menggunakan Camera.main.

Dengan script ini, pemain dapat menggambar garis pada dunia game dengan menekan dan menahan tombol kiri mouse, serta berhenti menggambar ketika tombol mouse kiri dilepas. Garis-garis ini akan terbuat dari objek LineRenderer dan dapat dikustomisasi dalam hal ketebalan dan warna sesuai dengan preferensi pengguna.

### 4.5 Touch DragDrop

| using UnityEngine;  using UnityEngine.UI;  using UnityEngine.EventSystems;  using UnityEngine.Events;  public class TouchDragDrop : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler  {  [Header("Drag Drop Speed")]  public float speed = 1f;  public string TargetTag;  bool canDrag = true;  [Header("Drag Drop Setting")]  public bool autoSnap;  public UnityEvent DragEvent;  public UnityEvent DropEvent;  [Header("Trigger Setting")]  public UnityEvent TriggerEnterEvent;  public UnityEvent TriggerStayEvent;  public UnityEvent TriggerExitEvent;  private Image image;  private RectTransform rectTransform;  private Canvas canvas;  private CanvasGroup canvasGroup;  private Vector2 touchOffset;  private void Awake()  {  image = GetComponent<Image>();  rectTransform = GetComponent<RectTransform>();  canvas = GetComponentInParent<Canvas>();  canvasGroup = GetComponent<CanvasGroup>();  if (image == null)  {  image = gameObject.AddComponent<Image>();  }  if (rectTransform == null)  {  rectTransform = gameObject.GetComponent<RectTransform>();  if (rectTransform == null)  {  rectTransform = gameObject.AddComponent<RectTransform>();  }  }  if (canvasGroup == null)  {  canvasGroup = gameObject.AddComponent<CanvasGroup>();  }  }  public void OnBeginDrag(PointerEventData eventData)  {  canvasGroup.alpha = 0.6f;  canvasGroup.blocksRaycasts = false;  canDrag = true;  // Calculate touch offset from the center of the object  touchOffset = eventData.position - (Vector2)rectTransform.position;  }  public void OnDrag(PointerEventData eventData)  {  if (canDrag)  {  Vector2 touchPosition = eventData.position;  Vector2 newPosition = touchPosition - touchOffset;  // Ensure the object stays within the canvas boundaries  newPosition.x = Mathf.Clamp(newPosition.x, rectTransform.rect.width / 2, canvas.pixelRect.width - rectTransform.rect.width / 2);  newPosition.y = Mathf.Clamp(newPosition.y, rectTransform.rect.height / 2, canvas.pixelRect.height - rectTransform.rect.height / 2);  rectTransform.position = newPosition;  DragEvent?.Invoke();  }  }  public void OnEndDrag(PointerEventData eventData)  {  canvasGroup.alpha = 1f;  canvasGroup.blocksRaycasts = true;  DropEvent?.Invoke();  }  void OnTriggerEnter2D(Collider2D other)  {  if (autoSnap)  {  canDrag = false;  transform.position = other.transform.position;  }  TriggerEnterEvent?.Invoke();  }  void OnTriggerStay2D(Collider2D other)  {  TriggerStayEvent?.Invoke();  }  void OnTriggerExit2D(Collider2D other)  {  TriggerExitEvent?.Invoke();  }  } |
| --- |

### 4.6 Touch GrabRelease

| using UnityEngine;  using UnityEngine.Events;  public class TouchGrabRelease : MonoBehaviour  {  [Header("Grab Release Speed")]  public float speed = 1f;  public string TargetTag;  bool canDrag = false;  [Header("Grab Release Setting")]  public bool autoSnap;  public UnityEvent GrabEvent;  public UnityEvent ReleaseEvent;  [Header("Trigger Setting")]  public UnityEvent TriggerEnterEvent;  public UnityEvent TriggerStayEvent;  public UnityEvent TriggerExitEvent;  private Vector3 initialPosition; // Menyimpan posisi awal objek  private Vector3 touchOffset;  private void Start()  {  Rigidbody rb = gameObject.AddComponent<Rigidbody>();  rb.useGravity = false;  rb.isKinematic = true; // Membuat objek menjadi kinematik untuk menghindari interaksi fisika saat dipegang  }  void Update()  {  if (Input.touchCount == 0)  {  canDrag = false;  }  }  private void OnMouseDown()  {  if (Input.touchCount > 0)  {  canDrag = true;  initialPosition = transform.position; // Menyimpan posisi awal objek saat sentuhan dimulai  Touch touch = Input.GetTouch(0);  touchOffset = transform.position - Camera.main.ScreenToWorldPoint(touch.position);  }  }  private void OnMouseDrag()  {  if (canDrag && Input.touchCount > 0)  {  Touch touch = Input.GetTouch(0);  Vector3 touchPosition = Camera.main.ScreenToWorldPoint(touch.position);  Vector3 newPosition = touchPosition + touchOffset;  transform.position = newPosition;  GrabEvent?.Invoke();  }  }  private void OnMouseUp()  {  canDrag = false;  ReleaseEvent?.Invoke();  }  private void OnTriggerEnter(Collider other)  {  if (autoSnap)  {  canDrag = false;  transform.position = other.transform.position;  }  TriggerEnterEvent?.Invoke();  }  private void OnTriggerStay(Collider other)  {  TriggerStayEvent?.Invoke();  }  private void OnTriggerExit(Collider other)  {  TriggerExitEvent?.Invoke();  }  } |
| --- |

### 4.7 Touch Joystick

Joystick mobile adalah elemen kontrol dalam permainan mobile yang memungkinkan pemain untuk menggerakkan karakter atau objek dalam game dengan sentuhan layar. Dalam konteks pengembangan game menggunakan Unity, joystick atau D-pad sering diimplementasikan sebagai elemen antarmuka pengguna yang dapat diatur untuk menggerakkan karakter dalam berbagai arah atau mengontrol pergerakan objek dalam game. Unity menyediakan komponen UI dan skrip yang memudahkan pengembang untuk mengimplementasikan kontrol joystick atau D-pad dalam permainan mobile, sehingga memungkinkan pemain untuk berinteraksi dengan game secara intuitif melalui layar sentuh perangkat seluler.

| using UnityEngine ;  using UnityEngine.Events ;  using UnityEngine.EventSystems ;  public class TouchJoystick: MonoBehaviour,IPointerDownHandler,IDragHandler,IPointerUpHandler {  private RectTransform container ;  private RectTransform handle ;  [Header("Input Settings")]  public bool ArrowKeysSimulationEnabled = false ;  private Vector2 point ;  private Vector2 normalizedPoint ;  private float maxLength ;  private bool \_isTouching = false ;  public bool IsTouching { get { return \_isTouching ; } }  public UnityAction OnJoystickDownAction ;  public UnityAction OnJoystickUpAction ;  private PointerEventData pointerEventData ;  private Camera cam ;  private void OnEnable () {  OnPointerUp (null) ;  }  private void Awake () {  container = transform.GetComponent <RectTransform> () ;  handle = container.GetChild (0).GetComponent <RectTransform> () ;  maxLength = (container.sizeDelta.x / 2f) - (handle.sizeDelta.x / 2f) - 5f ;  }  public void OnPointerDown (PointerEventData e) {  if (OnJoystickDownAction != null)  OnJoystickDownAction.Invoke () ;  \_isTouching = true ;  cam = e.pressEventCamera ;  OnDrag (e) ;  }  public void OnDrag (PointerEventData e) {  pointerEventData = e ;  }  void Update () {  if (\_isTouching && RectTransformUtility.ScreenPointToLocalPointInRectangle (container, pointerEventData.position, cam, out point)) {  point = Vector2.ClampMagnitude (point, maxLength) ;  handle.anchoredPosition = point ;  float length = Mathf.InverseLerp (0f, maxLength, point.magnitude) ;  normalizedPoint = Vector2.ClampMagnitude (point, length) ;  }  }  public void OnPointerUp (PointerEventData e) {  if (OnJoystickUpAction != null)  OnJoystickUpAction.Invoke () ;  \_isTouching = false ;  normalizedPoint = Vector3.zero ;  handle.anchoredPosition = Vector3.zero ;  }  public float Horizontal () {  if (ArrowKeysSimulationEnabled)  return (normalizedPoint.x != 0) ? normalizedPoint.x : Input.GetAxis ("Horizontal") ;  return normalizedPoint.x ;  }  public float Vertical () {  if (ArrowKeysSimulationEnabled)  return (normalizedPoint.y != 0) ? normalizedPoint.y : Input.GetAxis ("Vertical") ;  return normalizedPoint.y ;  }  } |
| --- |

Script ini adalah skrip yang mengimplementasikan joystick virtual di Unity. Joystick virtual ini dapat digunakan untuk mengontrol pergerakan atau tindakan dalam permainan, seperti pergerakan karakter atau objek. Skrip aslinya dikembangkan oleh Hamza Herbou dan telah dimodifikasi sesuai dengan kebutuhan. Mari kita jelaskan bagian-bagian utama dari skrip ini:

* private RectTransform container; dan private RectTransform handle;: Ini adalah referensi ke dua objek RectTransform dalam hierarki GameObject saat ini. container adalah area latar belakang joystick, dan handle adalah pegangan yang dapat digerakkan oleh pemain.
* [Header("Input Settings")] public bool ArrowKeysSimulationEnabled = false;: Ini adalah pengaturan yang memungkinkan simulasi penggunaan tombol panah (Arrow Keys) untuk mengontrol joystick jika diaktifkan.
* private Vector2 point; dan private Vector2 normalizedPoint;: Ini adalah vektor yang digunakan untuk menyimpan posisi pointer di dalam joystick (dalam koordinat lokal) dan posisi pointer yang sudah dinormalisasi (dalam rentang -1 hingga 1).
* private float maxLength;: Ini adalah panjang maksimum yang pegangan joystick dapat digerakkan dari tengah joystick.
* private bool \_isTouching = false;: Ini adalah status yang menunjukkan apakah pengguna sedang menyentuh joystick (bergerak di atasnya) atau tidak.
* public UnityAction OnJoystickDownAction; dan public UnityAction OnJoystickUpAction;: Ini adalah event yang dapat Anda gunakan untuk menetapkan tindakan yang akan dijalankan ketika joystick mulai disentuh (down) atau ketika joystick dilepaskan (up).
* private PointerEventData pointerEventData; dan private Camera cam;: Ini adalah variabel untuk menyimpan data acara pointer saat joystick disentuh, serta kamera yang digunakan.
* private void OnEnable(): Metode ini dipanggil saat GameObject ini diaktifkan. Pada bagian ini, metode OnPointerUp(null) dipanggil untuk memastikan joystick berada dalam kondisi awal.
* private void Awake(): Metode ini dipanggil saat GameObject pertama kali dibuat. Pada bagian ini, referensi ke objek RectTransform container dan handle diambil, dan panjang maksimum joystick dihitung.
* public void OnPointerDown(PointerEventData e): Metode ini dipanggil ketika pengguna menyentuh joystick. Ini akan mengaktifkan joystick dan memindahkan pegangan ke posisi sentuhan pengguna.
* public void OnDrag(PointerEventData e): Metode ini dipanggil ketika pengguna menggerakkan joystick. Ini mengupdate pointerEventData dengan data pergerakan pengguna.
* void Update(): Metode ini dipanggil setiap frame. Jika joystick sedang disentuh (\_isTouching adalah true) dan posisi pointer ada dalam batas container, maka posisi pegangan joystick akan diperbarui sesuai dengan pergerakan pengguna. Selanjutnya, normalizedPoint akan dihitung sebagai vektor yang dinormalisasi berdasarkan posisi pegangan.
* public void OnPointerUp(PointerEventData e): Metode ini dipanggil ketika pengguna melepaskan joystick. Ini akan menonaktifkan joystick, mengatur normalizedPoint kembali ke nol, dan mengembalikan pegangan ke posisi tengah.
* public float Horizontal(): Metode ini mengembalikan nilai komponen X dari normalizedPoint, yang mengindikasikan pergerakan horizontal joystick. Jika ArrowKeysSimulationEnabled diaktifkan, metode akan mengembalikan nilai dari input sumbu horizontal (tombol panah atau input horizontal yang lain).
* public float Vertical(): Metode ini mengembalikan nilai komponen Y dari normalizedPoint, yang mengindikasikan pergerakan vertikal joystick. Jika ArrowKeysSimulationEnabled diaktifkan, metode akan mengembalikan nilai dari input sumbu vertikal (tombol panah atau input vertikal yang lain).

| **Zetcil Framework** |
| --- |
| **Joystick Controller PAD:**  <https://drive.google.com/drive/u/2/folders/18MVqZkfTwz8_fM-Fy0AHgBVEB8uHFf9y> |

| **Cara Membuat Touch Joystick di Unity (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/4Y2ry0PkFCI> |

| **Cara Membuat Touch Joystick di Unity (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/odyMlECAIXk> |

| **Update Script: Touch Joystick x TransformDirectionXYZ** |
| --- |
| **Tutorial:** <https://youtu.be/1VhWZlVFul0> |

## 5. Gesture

Input gesture adalah jenis input yang mengidentifikasi gerakan fisik pemain sebagai perintah dalam game. Variasi input gesture meliputi gerakan tangan, gerakan kepala, dan ekspresi wajah. Misalnya, pemain dapat menggerakkan tangan ke kiri atau ke kanan untuk menggerakkan karakter atau objek dalam game, menganggukkan kepala untuk mengkonfirmasi pilihan, atau menggerakkan alis untuk mengaktifkan kekuatan khusus. Beberapa game juga mendukung gesture kompleks yang melibatkan kombinasi gerakan dan ekspresi, misalnya menggerakkan tangan ke atas sambil tersenyum untuk melompat dan menyerang secara bersamaan. Teknologi sensor gerak dan kamera 3D diperlukan untuk mendeteksi dan menginterpretasikan input gesture pemain. Dengan memanfaatkan input gesture, pengembang game dapat menciptakan pengalaman bermain yang lebih imersif dan interaktif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih alami dan intuitif.

### 5.1 Gesture Swipe

| using UnityEngine;  using UnityEngine.Events;  public class GestureSwipe : MonoBehaviour  {  [Header("Main Settings")]  [SerializeField] private float swipeThreshold = 50f;  [SerializeField] private UnityEvent onSwipeUp;  [SerializeField] private UnityEvent onSwipeDown;  [SerializeField] private UnityEvent onSwipeLeft;  [SerializeField] private UnityEvent onSwipeRight;  private Vector2 fingerDownPosition;  private Vector2 fingerUpPosition;  private void Update()  {  if (Input.touchCount == 1)  {  Touch touch = Input.GetTouch(0);  if (touch.phase == TouchPhase.Began)  {  fingerDownPosition = touch.position;  fingerUpPosition = touch.position;  }  if (touch.phase == TouchPhase.Moved)  {  fingerUpPosition = touch.position;  }  if (touch.phase == TouchPhase.Ended)  {  float deltaX = fingerUpPosition.x - fingerDownPosition.x;  float deltaY = fingerUpPosition.y - fingerDownPosition.y;  if (Mathf.Abs(deltaX) > swipeThreshold || Mathf.Abs(deltaY) > swipeThreshold)  {  if (Mathf.Abs(deltaX) > Mathf.Abs(deltaY))  {  if (deltaX > 0)  {  onSwipeRight.Invoke();  }  else  {  onSwipeLeft.Invoke();  }  }  else  {  if (deltaY > 0)  {  onSwipeUp.Invoke();  }  else  {  onSwipeDown.Invoke();  }  }  }  }  }  }  } |
| --- |

Script ini adalah sebuah komponen dalam Unity yang digunakan untuk mendeteksi gestur swipe (gesek) pada layar perangkat berbasis sentuhan, seperti ponsel atau tablet. Script ini memungkinkan Anda untuk merespons swipe atas, bawah, kiri, dan kanan, serta menjalankan peristiwa yang sesuai untuk setiap swipe. Berikut penjelasan komponen utama dari script ini:

* [SerializeField] private float swipeThreshold = 50f;: Ini adalah batasan (threshold) jarak minimum yang harus ditempuh oleh swipe agar dianggap sebagai swipe yang valid. Nilai ini dapat diatur melalui editor Unity dan menentukan seberapa jauh pengguna harus melakukan swipe agar dianggap sebagai gerakan swipe.
* [SerializeField] private UnityEvent onSwipeUp;, [SerializeField] private UnityEvent onSwipeDown;, [SerializeField] private UnityEvent onSwipeLeft;, dan [SerializeField] private UnityEvent onSwipeRight;: Ini adalah peristiwa-peristiwa Unity yang dapat diaktifkan ketika swipe ke atas, bawah, kiri, atau kanan terdeteksi. Anda dapat menghubungkan fungsi-fungsi tertentu ke peristiwa-peristiwa ini melalui editor Unity.
* private Vector2 fingerDownPosition; dan private Vector2 fingerUpPosition;: Ini adalah posisi awal (down position) dan posisi akhir (up position) sentuhan pengguna. Kedua posisi ini digunakan untuk menghitung jarak dan arah swipe.
* void Update(): Ini adalah metode yang dijalankan setiap frame dalam permainan. Di dalamnya, script melakukan hal-hal berikut: Mengecek apakah ada satu sentuhan aktif (Input.touchCount == 1), artinya hanya satu sentuhan yang sedang terjadi. Memeriksa fase sentuhan saat ini (touch.phase) untuk mengidentifikasi awal, pergerakan, dan berakhirnya sentuhan. Ketika sentuhan dimulai (TouchPhase.Began), script menyimpan posisi awal sentuhan (fingerDownPosition) dan posisi akhir sentuhan (fingerUpPosition) ke posisi yang sama. Selama sentuhan digerakkan (TouchPhase.Moved), script terus memperbarui posisi akhir sentuhan (fingerUpPosition) dengan posisi saat ini.

Dengan script ini, Anda dapat mengenali gestur swipe pengguna pada perangkat berbasis sentuhan dan menghubungkannya dengan berbagai peristiwa dalam permainan Unity, seperti menggerakkan karakter atau objek berdasarkan arah swipe yang terdeteksi.

### 5.2 Gesture Rotator

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GestureRotator : MonoBehaviour  {  [Header("Main Settings")]  public bool RotatorEnabled;  [Header("Movement Settings")]  public float rotationSpeed = 1.0f; // Kecepatan rotasi objek  private bool isRotating = false; // Apakah objek sedang dalam proses rotasi  private Vector2 lastTouchPosition; // Posisi sentuhan sebelumnya  void Update()  {  // Mendeteksi input sentuhan  if (RotatorEnabled && Input.touchCount == 1)  {  Touch touch = Input.GetTouch(0);  switch (touch.phase)  {  case TouchPhase.Began:  // Menyimpan posisi sentuhan saat ini sebagai posisi sebelumnya  lastTouchPosition = touch.position;  break;  case TouchPhase.Moved:  // Menghitung perubahan posisi sentuhan  float deltaPositionX = touch.position.x - lastTouchPosition.x;  // Menghitung rotasi berdasarkan perubahan posisi  float rotation = deltaPositionX \* -rotationSpeed;  // Memutar objek horizontal  transform.Rotate(Vector3.up, rotation);  // Memperbarui posisi sentuhan terakhir  lastTouchPosition = touch.position;  // Objek sedang dalam proses rotasi  isRotating = true;  break;  case TouchPhase.Ended:  // Menandakan bahwa objek selesai dalam proses rotasi  isRotating = false;  break;  }  }  else  {  // Menghentikan rotasi saat tidak ada sentuhan  isRotating = false;  }  }  void LateUpdate()  {  }  } |
| --- |

Script ini digunakan untuk mengizinkan pengguna untuk memutar objek dalam permainan menggunakan input sentuhan horizontal pada layar perangkat seluler atau perangkat lain yang mendukung input sentuhan. Berikut adalah penjelasan komponen utama dari script ini:

* private float rotationSpeed = 2.0f;: Variabel ini menentukan kecepatan rotasi objek. Nilai ini dapat diubah di dalam editor Unity untuk mengontrol seberapa cepat objek akan berputar.
* private bool isRotating = false;: Ini adalah status yang menunjukkan apakah objek sedang dalam proses rotasi atau tidak.
* private Vector2 lastTouchPosition;: Ini adalah variabel yang digunakan untuk menyimpan posisi sentuhan terakhir.
* void Update(): Metode ini dijalankan setiap frame dalam permainan. Di dalamnya, script melakukan hal-hal berikut:
* Mengecek apakah ada satu sentuhan aktif (Input.touchCount == 1), artinya hanya satu sentuhan yang sedang terjadi.
* Mengambil informasi tentang sentuhan tersebut dengan Touch touch = Input.GetTouch(0);.
* Kemudian, script menggunakan switch statement untuk memeriksa fase sentuhan (touch.phase):
* TouchPhase.Began: Pada fase ini, posisi sentuhan saat ini disimpan sebagai lastTouchPosition. Ini digunakan sebagai referensi untuk mengukur perubahan posisi saat sentuhan digerakkan.
* TouchPhase.Moved: Pada fase ini, script menghitung perubahan posisi sentuhan dalam arah horizontal (deltaPositionX) dan kemudian menghitung rotasi yang harus diterapkan pada objek berdasarkan perubahan posisi tersebut. Objek akan diputar sekitar sumbu Y (Vector3.up) sesuai dengan perubahan posisi.
* TouchPhase.Ended: Pada fase ini, sentuhan telah berakhir, sehingga status isRotating diatur menjadi false untuk menandakan bahwa objek selesai dalam proses rotasi.
* void LateUpdate(): Metode ini juga dijalankan setiap frame dalam permainan, tetapi setelah metode Update() selesai dijalankan. Di dalamnya, script menangani rotasi berkelanjutan setelah sentuhan berakhir:

Jika isRotating adalah false, artinya objek tidak sedang dalam proses rotasi (sentuhan telah berakhir), maka objek akan terus berputar secara horizontal dengan kecepatan yang telah ditentukan (rotationSpeed) menggunakan transform.Rotate(Vector3.up, Time.deltaTime \* rotationSpeed). Dengan script ini, objek dalam permainan akan berputar ketika pengguna menggesekkan jari mereka di layar perangkat seluler secara horizontal. Rotasi akan berhenti saat sentuhan berakhir, dan objek akan terus berputar secara perlahan jika tidak ada sentuhan yang sedang terjadi.

### 5.3 Gesture Pinch

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GesturePinch : MonoBehaviour  {  [Header("Main Settings")]  public bool PinchEnabled;  [Header("Movement Settings")]  public float scaleFactor = 1;  private float initialDistance;  private Vector3 initialScale;  // Start is called before the first frame update  void Start()  {  }  public void SetTouchLookEnabled(bool aValue)  {  PinchEnabled = aValue;  }  // Update is called once per frame  void Update()  {  if (PinchEnabled && Input.touchCount == 2)  {  var touchZero = Input.GetTouch(0);  var touchOne = Input.GetTouch(1);  // if any one of touchzero or touchOne is cancelled or maybe ended then do nothing  if (touchZero.phase == TouchPhase.Ended || touchZero.phase == TouchPhase.Canceled ||  touchOne.phase == TouchPhase.Ended || touchOne.phase == TouchPhase.Canceled)  {  return; // basically do nothing  }  if (touchZero.phase == TouchPhase.Began || touchOne.phase == TouchPhase.Began)  {  initialDistance = Vector2.Distance(touchZero.position, touchOne.position);  initialScale = transform.localScale;  }  else // if touch is moved  {  var currentDistance = Vector2.Distance(touchZero.position, touchOne.position);  //if accidentally touched or pinch movement is very very small  if (Mathf.Approximately(initialDistance, 0))  {  return; // do nothing if it can be ignored where inital distance is very close to zero  }  var factor = currentDistance / initialDistance \* scaleFactor;  transform.localScale = initialScale \* factor; // scale multiplied by the factor we calculated  }  }  }  } |
| --- |

Script ini adalah komponen dalam Unity yang memungkinkan pengguna untuk melakukan skalasi (perubahan ukuran) objek menggunakan gestur pinch-to-zoom, yaitu dengan menggesekkan dua jari ke dalam atau ke luar di layar perangkat sentuh. Berikut adalah penjelasan komponen utama dari script ini:

* [Header("Touch Settings")] public bool TouchScalerEnabled;: Ini adalah boolean yang dapat diatur dari editor Unity. Jika diaktifkan (true), maka skalasi menggunakan gestur pinch-to-zoom akan diaktifkan; jika dinonaktifkan (false), maka skalasi tidak akan berfungsi.
* [Header("Movement Settings")] public float scaleFactor = 1;: Ini adalah faktor skalasi yang menentukan seberapa cepat objek akan membesar atau mengecil saat dilakukan pinch. Nilai ini dapat diubah di editor Unity untuk mengontrol seberapa besar atau kecil efek skalasi.
* private float initialDistance; dan private Vector3 initialScale;: Variabel ini digunakan untuk menyimpan jarak awal antara dua titik sentuhan (jari) pada layar (initialDistance) dan skala awal objek (initialScale) sebelum dilakukan skalasi.
* void Start(): Ini adalah metode yang dijalankan pada awal permainan. Di dalamnya, tidak ada kode yang dijalankan, sehingga metode ini kosong.
* public void SetTouchLookEnabled(bool aValue): Ini adalah metode yang digunakan untuk mengatur nilai TouchScalerEnabled dari luar script. Meskipun ada dalam script, namun tampaknya tidak digunakan dalam script ini dan mungkin digunakan di luar script untuk mengaktifkan atau menonaktifkan skalasi.
* void Update(): Ini adalah metode yang dijalankan setiap frame dalam permainan. Di dalamnya, script melakukan hal-hal berikut:
* Mengecek apakah TouchScalerEnabled diaktifkan dan apakah ada dua sentuhan aktif (Input.touchCount == 2).

Jika salah satu dari dua sentuhan (touchZero atau touchOne) dibatalkan atau berakhir (TouchPhase.Ended atau TouchPhase.Canceled), maka script tidak akan melakukan apa-apa dan akan langsung keluar dari metode Update().

Ketika salah satu sentuhan dimulai (TouchPhase.Began), script menyimpan jarak awal antara kedua sentuhan (initialDistance) dan skala awal objek (initialScale). Jika salah satu sentuhan digerakkan (TouchPhase.Moved), script menghitung jarak saat ini antara kedua sentuhan (currentDistance) dan kemudian menghitung faktor skalasi berdasarkan perbandingan jarak saat ini dengan jarak awal. Skala objek diubah dengan mengalikan skala awal (initialScale) dengan faktor skalasi yang dihitung.

Jika jarak awal (initialDistance) sangat mendekati nol (menggunakan Mathf.Approximately), maka script tidak akan melakukan apa-apa, mengabaikan perubahan jarak yang sangat kecil yang dapat diabaikan. Dengan script ini, pengguna dapat melakukan skalasi objek dengan menggunakan gestur pinch-to-zoom pada layar perangkat sentuh. Objek akan membesar atau mengecil seiring dengan gerakan jari pengguna.

### 5.4 Gesture Observer

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GestureObserver : MonoBehaviour  {  [Header("Main Settings")]  public float rotationSpeed = 1.0f; // Kecepatan rotasi objek  public float zoomFactor = 1f;  public bool slowRotate = false;  private bool isRotating = false; // Apakah objek sedang dalam proses rotasi  private Vector2 lastTouchPosition; // Posisi sentuhan sebelumnya  private float initialDistance; // Jarak awal antara dua sentuhan  void Update()  {  // Mendeteksi input sentuhan  if (Input.touchCount == 1)  {  Touch touch = Input.GetTouch(0);  switch (touch.phase)  {  case TouchPhase.Began:  // Menyimpan posisi sentuhan saat ini sebagai posisi sebelumnya  lastTouchPosition = touch.position;  break;  case TouchPhase.Moved:  // Menghitung perubahan posisi sentuhan  float deltaPositionX = touch.position.x - lastTouchPosition.x;  // Menghitung rotasi berdasarkan perubahan posisi  float rotation = deltaPositionX \* -rotationSpeed;  // Memutar objek horizontal  transform.Rotate(Vector3.up, rotation);  // Memperbarui posisi sentuhan terakhir  lastTouchPosition = touch.position;  // Objek sedang dalam proses rotasi  isRotating = true;  break;  case TouchPhase.Ended:  // Menandakan bahwa objek selesai dalam proses rotasi  isRotating = false;  break;  }  }  else if (Input.touchCount == 2)  {  // Mendeteksi input pinch (dua sentuhan)  Touch touch1 = Input.GetTouch(0);  Touch touch2 = Input.GetTouch(1);  switch (touch1.phase)  {  case TouchPhase.Began:  // Menyimpan jarak awal antara dua sentuhan  initialDistance = Vector2.Distance(touch1.position, touch2.position);  break;  case TouchPhase.Moved:  // Menghitung jarak saat ini antara dua sentuhan  float currentDistance = Vector2.Distance(touch1.position, touch2.position);  // Menghitung perubahan jarak  float pinchDelta = currentDistance - initialDistance;  // Menghitung faktor zoom  float localzoomFactor = pinchDelta \* 0.01f \* zoomFactor; // Nilai ini dapat disesuaikan  // Zoom in dan zoom out  transform.localScale += new Vector3(localzoomFactor, localzoomFactor, localzoomFactor);  // Memperbarui jarak awal  initialDistance = currentDistance;  break;  }  }  else  {  // Menghentikan rotasi saat tidak ada sentuhan  isRotating = false;  }  }  void LateUpdate()  {  // Menangani rotasi berkelanjutan setelah sentuhan berakhir  if (!isRotating)  {  if (slowRotate)  {  // Putar objek horizontal secara perlahan  transform.Rotate(Vector3.up, Time.deltaTime \* rotationSpeed);  }  }  }  } |
| --- |

Script ini adalah sebuah komponen dalam Unity yang memungkinkan pengguna untuk memanipulasi objek dalam permainan menggunakan input sentuhan. Script ini mendukung dua jenis interaksi utama: rotasi objek melalui sentuhan tunggal dan zoom in/zoom out objek melalui gestur pinch-to-zoom menggunakan dua sentuhan. Berikut adalah penjelasan komponen utama dari script ini:

* private float rotationSpeed = 2.0f;: Ini adalah variabel yang menentukan kecepatan rotasi objek ketika menggunakan sentuhan tunggal. Anda dapat mengubah nilai ini untuk mengontrol seberapa cepat objek akan berputar.
* private bool isRotating = false;: Ini adalah status yang menunjukkan apakah objek sedang dalam proses rotasi atau tidak.
* private Vector2 lastTouchPosition;: Ini adalah variabel yang digunakan untuk menyimpan posisi sentuhan terakhir saat menggunakan sentuhan tunggal.
* private float initialDistance;: Ini adalah variabel yang digunakan untuk menyimpan jarak awal antara dua sentuhan saat menggunakan gestur pinch-to-zoom.
* void Update(): Ini adalah metode yang dijalankan setiap frame dalam permainan. Di dalamnya, script melakukan hal-hal berikut:
* Mengecek apakah ada satu sentuhan aktif (Input.touchCount == 1).

Jika ada satu sentuhan aktif, script memeriksa fase sentuhan (touch.phase) dan melakukan rotasi objek berdasarkan perubahan posisi sentuhan horizontal. Posisi sentuhan saat ini disimpan sebagai lastTouchPosition, dan rotasi dihitung berdasarkan perubahan posisi sentuhan horizontal.

Jika ada dua sentuhan aktif (Input.touchCount == 2), script akan mendeteksi gestur pinch-to-zoom. Pada sentuhan pertama (touch1), saat fase TouchPhase.Began, script menyimpan jarak awal antara dua sentuhan menggunakan initialDistance. Selama sentuhan pertama (touch1) digerakkan (TouchPhase.Moved), script menghitung jarak saat ini antara dua sentuhan (currentDistance) dan perubahan jarak (pinchDelta) yang terjadi sejak sentuhan pertama dimulai. Selanjutnya, script menghitung faktor zoom (zoomFactor) berdasarkan perubahan jarak dan mengubah skala objek untuk melakukan zoom in atau zoom out.

Jika isRotating adalah false, artinya objek tidak sedang dalam proses rotasi (sentuhan telah berakhir), maka objek akan terus berputar secara horizontal dengan kecepatan yang telah ditentukan (rotationSpeed) menggunakan transform.Rotate(Vector3.up, Time.deltaTime \* rotationSpeed).

Dengan script ini, pengguna dapat memutar objek dengan menggesekkan jari mereka secara horizontal dan melakukan zoom in/zoom out objek dengan melakukan gestur pinch-to-zoom pada layar perangkat sentuh. Objek akan merespons sentuhan pengguna sesuai dengan interaksi yang dijelaskan di atas.

## 6. Voice

Input voice adalah jenis input yang menggunakan suara pemain sebagai perintah dalam game. Variasi input voice meliputi perintah verbal, nada suara, dan intensitas suara. Misalnya, pemain dapat berbicara "serang" untuk membuat karakter menyerang musuh, "berhenti" untuk menghentikan gerakan, atau "bantuan" untuk meminta bantuan. Nada suara juga dapat digunakan untuk mengendalikan karakter atau objek dalam game, misalnya menaikkan nada suara untuk membuat karakter melompat atau menurunkan nada suara untuk membuat karakter berjongkok. Intensitas suara, seperti berbisik atau berteriak, juga bisa digunakan untuk melakukan aksi tertentu, seperti menyelinap atau mengaktifkan kekuatan khusus. Pengenalan suara dan teknologi pengolahan suara diperlukan untuk mendeteksi dan menginterpretasikan input voice pemain. Dengan memanfaatkan input voice, pengembang dapat menciptakan pengalaman bermain yang lebih dinamis dan inovatif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih intuitif dan alami.

### 6.1 Voice Recorder

| using UnityEngine;  using UnityEngine.Audio;  [RequireComponent(typeof(AudioSource))]  public class VoiceRecorder : MonoBehaviour  {  [Header("Voice Recorder Setting")]  public float volume = 1.0f;  public float pitch = 1.0f;  public float maxTime = 5.0f; // Durasi rekaman dalam detik  public float recordingTime = 5.0f; // Durasi rekaman dalam detik  bool isRecording = false;  private AudioSource audioSource;  private AudioClip recordedClip;  private void Start()  {  audioSource = GetComponent<AudioSource>();  audioSource.volume = volume;  audioSource.pitch = pitch;  }  private void Update()  {  if (isRecording)  {  recordingTime -= Time.deltaTime;  if (recordingTime <= 0)  {  StopRecording();  }  }  }  public void StartRecording()  {  if (!Microphone.IsRecording(null))  {  recordingTime = maxTime;  recordedClip = Microphone.Start(null, false, (int)recordingTime, 44100);  isRecording = true;  }  }  public void StopRecording()  {  if (Microphone.IsRecording(null))  {  Microphone.End(null);  isRecording = false;  PlayRecordedAudio();  }  }  private void PlayRecordedAudio()  {  if (recordedClip != null)  {  audioSource.clip = recordedClip;  audioSource.Play();  }  }  } |
| --- |

Script ini adalah komponen dalam Unity yang digunakan untuk merekam suara menggunakan mikrofon perangkat dan memainkan kembali rekaman suara. Berikut adalah penjelasan komponen utama dari script ini:

* [Header("Voice Recorder Setting")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan terkait perekaman suara.
* public float volume = 1.0f;: Variabel ini menentukan volume dari rekaman suara yang akan diputar kembali. Nilai defaultnya adalah 1.0, yang merupakan volume penuh.
* public float pitch = 1.0f;: Variabel ini menentukan pitch dari rekaman suara yang akan diputar kembali. Nilai defaultnya adalah 1.0, yang tidak mengubah pitch asli.
* public float maxTime = 5.0f;: Ini adalah durasi maksimal rekaman suara dalam detik. Nilai defaultnya adalah 5 detik.
* public float recordingTime = 5.0f;: Ini adalah variabel yang digunakan untuk menghitung sisa waktu rekaman saat sedang dalam proses perekaman.
* bool isRecording = false;: Ini adalah status yang menunjukkan apakah perekaman suara sedang berlangsung atau tidak.
* private AudioSource audioSource;: Ini adalah komponen AudioSource yang akan digunakan untuk memainkan kembali rekaman suara.
* private AudioClip recordedClip;: Ini adalah variabel yang akan digunakan untuk menyimpan rekaman suara yang direkam.
* void Start(): Ini adalah metode yang dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan.

Dengan script ini, Anda dapat merekam suara dari mikrofon perangkat, mengatur parameter suara seperti volume dan pitch, dan memainkan rekaman suara kembali pada objek dengan komponen VoiceRecorder.

### 6.2 Voice Debugger

| using System;  using System.Text;  using UnityEngine;  using UnityEngine.Windows.Speech;  public class VoiceDebugger : MonoBehaviour  {  [Header("Keywords Setting")]  public string[] m\_Keywords;  private KeywordRecognizer m\_Recognizer;  void Start()  {  m\_Recognizer = new KeywordRecognizer(m\_Keywords);  m\_Recognizer.OnPhraseRecognized += OnPhraseRecognized;  m\_Recognizer.Start();  }  private void OnPhraseRecognized(PhraseRecognizedEventArgs args)  {  StringBuilder builder = new StringBuilder();  builder.AppendFormat("{0} ({1}){2}", args.text, args.confidence, Environment.NewLine);  builder.AppendFormat("\tTimestamp: {0}{1}", args.phraseStartTime, Environment.NewLine);  builder.AppendFormat("\tDuration: {0} seconds{1}", args.phraseDuration.TotalSeconds, Environment.NewLine);  Debug.Log(builder.ToString());  }  } |
| --- |

Script ini adalah komponen dalam Unity yang digunakan untuk mendeteksi kata kunci (keywords) yang diucapkan oleh pengguna melalui pengenalan suara dan menampilkan informasi terkait kata kunci yang diucapkan ke dalam log Unity. Berikut adalah penjelasan komponen utama dari script ini:

* [Header("Keywords Setting")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan terkait kata kunci yang akan dideteksi.
* public string[] m\_Keywords;: Ini adalah array dari kata kunci yang akan dideteksi oleh sistem pengenalan suara. Anda dapat menambahkan kata kunci yang ingin Anda deteksi ke dalam array ini.
* private KeywordRecognizer m\_Recognizer;: Ini adalah objek KeywordRecognizer yang digunakan untuk mengenali kata kunci yang diucapkan oleh pengguna.
* void Start(): Ini adalah metode yang dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya:
* Membuat objek KeywordRecognizer menggunakan array m\_Keywords yang telah ditentukan.
* Menambahkan event handler OnPhraseRecognized ke KeywordRecognizer. Event ini akan dipanggil ketika salah satu kata kunci yang ditentukan diucapkan oleh pengguna.
* Memulai pengenalan kata kunci dengan memanggil m\_Recognizer.Start().
* private void OnPhraseRecognized(PhraseRecognizedEventArgs args): Ini adalah metode yang akan dipanggil ketika salah satu kata kunci yang ditentukan diucapkan oleh pengguna. Di dalamnya:

Dengan script ini, Anda dapat mendeteksi kata kunci yang diucapkan oleh pengguna melalui mikrofon perangkat dan menampilkan informasi terkait pengenalan kata kunci ke dalam log Unity. Informasi tersebut mencakup kata kunci yang diucapkan, tingkat kepercayaan, waktu ketika kata kunci diucapkan, dan durasi pengenalan kata kunci. Ini dapat digunakan untuk membuat interaksi suara dalam permainan Anda atau untuk tujuan debugging.

### 6.3 Voice Recognition

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.Windows.Speech;  public class VoiceRecognition : MonoBehaviour  {  public enum CRecognitionInterval { None, Once, Repeat }  public enum CRecognitionBehavior { None, ClearAfterRecognized }  [Header("Main Settings")]  public CRecognitionInterval RecognitionInterval = CRecognitionInterval.Repeat;  public CRecognitionBehavior RecognitionBehavior = CRecognitionBehavior.ClearAfterRecognized;  public ConfidenceLevel Confidence = ConfidenceLevel.Low;  [System.Serializable]  public class CEventSpeech  {  [Header("Speech List")]  public List<string> SpeechRecognized;  [Header("Speech Event")]  public UnityEvent SpeechEvent;  }  [Header("Speech Recognition Settings")]  public List<CEventSpeech> Keywords;  protected PhraseRecognizer recognizer;  [HideInInspector]  public string[] dictionary;  [Header("Variable Settings")]  public string TargetString;  // Start is called before the first frame update  void Start()  {  int dictionary\_length = 0;  for (int i=0; i< Keywords.Count; i++)  {  dictionary\_length += Keywords[i].SpeechRecognized.Count;  }  dictionary = new string[dictionary\_length];  int k = 0;  for (int i = 0; i < Keywords.Count; i++)  {  for (int j = 0; j < Keywords[i].SpeechRecognized.Count; j++)  {  dictionary[k] = Keywords[i].SpeechRecognized[j];  k++;  }  }  /\*dictionary = new string[Keywords.Length];  for (int i = 0; i < Keywords.Length; i++)  {  dictionary[i] = Keywords[i];  }  \*/  recognizer = new KeywordRecognizer(dictionary, Confidence);  recognizer.OnPhraseRecognized += Recognizer\_OnPhraseRecognized;  recognizer.Start();  }  void FixedUpdate()  {  if (this.gameObject.activeSelf)  {  foreach (CEventSpeech temp in Keywords)  {  foreach (string speechcompare in temp.SpeechRecognized)  {  if (speechcompare == TargetString)  {  if (RecognitionInterval == CRecognitionInterval.Once)  {  RecognitionInterval = CRecognitionInterval.None;  temp.SpeechEvent.Invoke();  }  else if (RecognitionInterval == CRecognitionInterval.Repeat)  {  temp.SpeechEvent.Invoke();  }  if (RecognitionBehavior == CRecognitionBehavior.ClearAfterRecognized)  {  TargetString = "";  }  }  }  }  }  }  private void Recognizer\_OnPhraseRecognized(PhraseRecognizedEventArgs args)  {  if (this.gameObject.activeSelf)  {  TargetString = args.text;  foreach (CEventSpeech temp in Keywords)  {  foreach (string speechcompare in temp.SpeechRecognized)  {  if (speechcompare == TargetString)  {  if (RecognitionInterval == CRecognitionInterval.Once)  {  RecognitionInterval = CRecognitionInterval.None;  temp.SpeechEvent.Invoke();  }  else if (RecognitionInterval == CRecognitionInterval.Repeat)  {  temp.SpeechEvent.Invoke();  }  if (RecognitionBehavior == CRecognitionBehavior.ClearAfterRecognized)  {  TargetString = "";  }  }  }  }  }  }  private void OnApplicationQuit()  {  if (recognizer != null && recognizer.IsRunning)  {  recognizer.OnPhraseRecognized -= Recognizer\_OnPhraseRecognized;  recognizer.Stop();  }  }  } |
| --- |

Script ini adalah komponen dalam Unity yang digunakan untuk mengenali kata kunci (keywords) yang diucapkan oleh pengguna melalui pengenalan suara. Ini juga memungkinkan Anda untuk menghubungkan peristiwa (events) dengan kata kunci yang diucapkan, sehingga Anda dapat mengeksekusi kode tertentu ketika kata kunci tertentu diucapkan. Berikut adalah penjelasan komponen utama dari script ini:

* public enum CRecognitionInterval { None, Once, Repeat } dan public enum CRecognitionBehavior { None, ClearAfterRecognized }: Ini adalah enumerasi yang mendefinisikan opsi untuk interval pengenalan kata kunci dan perilaku setelah kata kunci diakui.
* CRecognitionInterval: Enum ini digunakan untuk mengatur seberapa sering kata kunci diakui. Pilihan meliputi:

None: Tidak ada pengenalan kata kunci yang akan terjadi.

Once: Kata kunci akan diakui hanya sekali.

Repeat: Kata kunci akan diakui secara berulang.

* CRecognitionBehavior: Enum ini digunakan untuk mengatur perilaku setelah kata kunci diakui. Pilihan meliputi:

None: Tidak ada tindakan tambahan yang diambil.

ClearAfterRecognized: Menghapus kata kunci yang diakui setelah pengenalan.

* [Header("Main Settings")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan utama.
* public CRecognitionInterval RecognitionInterval = CRecognitionInterval.Repeat;: Ini adalah variabel yang menentukan interval pengenalan kata kunci. Nilai defaultnya adalah Repeat.
* public CRecognitionBehavior RecognitionBehavior = CRecognitionBehavior.ClearAfterRecognized;: Ini adalah variabel yang menentukan perilaku setelah kata kunci diakui. Nilai defaultnya adalah ClearAfterRecognized.
* public ConfidenceLevel Confidence = ConfidenceLevel.Low;: Ini adalah tingkat kepercayaan yang digunakan untuk pengenalan kata kunci. Tingkat kepercayaan mengindikasikan seberapa yakin pengenalan harus dilakukan.
* [System.Serializable] public class CEventSpeech { ... }: Ini adalah kelas serializable yang digunakan untuk mengelompokkan kata kunci yang diakui dan peristiwa yang terkait dengan kata kunci tersebut.
* SpeechRecognized: List kata kunci yang akan diakui.
* SpeechEvent: UnityEvent yang akan dipicu ketika salah satu kata kunci diucapkan.
* [Header("Speech Recognition Settings")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan pengenalan kata kunci.
* public List<CEventSpeech> Keywords;: List dari objek CEventSpeech yang berisi kata kunci yang akan diakui dan peristiwa yang akan dipicu ketika kata kunci tersebut diucapkan.
* protected PhraseRecognizer recognizer;: Ini adalah objek PhraseRecognizer yang digunakan untuk mengenali kata kunci yang diucapkan oleh pengguna.
* [HideInInspector] public string[] dictionary;: Ini adalah array yang berisi daftar kata kunci yang akan digunakan oleh PhraseRecognizer.
* [Header("Variable Settings")] public string TargetString;: Ini adalah variabel string yang digunakan untuk menyimpan kata kunci yang baru-baru ini diakui oleh pengenal suara.

## 7. Eye

Input eye adalah jenis input yang menggunakan gerakan mata dan fokus pandangan pemain sebagai perintah dalam game. Variasi input eye meliputi tracking mata, gaze control, dan kedipan mata. Tracking mata memungkinkan game untuk mengikuti gerakan mata pemain dan menggerakkan kamera atau fokus pandangan karakter sesuai dengan arah pandang pemain. Gaze control memungkinkan pemain untuk memilih objek, menargetkan musuh, atau berinteraksi dengan elemen dalam game hanya dengan memandangnya. Kedipan mata dapat digunakan untuk melakukan aksi seperti menembak, mengaktifkan perisai, atau mengkonfirmasi pilihan. Teknologi tracking mata diperlukan untuk mendeteksi dan menginterpretasikan input eye pemain. Dengan memanfaatkan input eye, pengembang dapat menciptakan pengalaman bermain yang lebih responsif dan imersif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih intuitif dan alami.

### 7.1 Eye Raycast

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class EyeRaycast : MonoBehaviour  {  public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }  [Header("Settings")]  public RotationAxes axes = RotationAxes.MouseXAndY;  public float sensitivityX = 15F;  public float sensitivityY = 15F;  public float minimumX = -360F;  public float maximumX = 360F;  public float minimumY = -60F;  public float maximumY = 60F;  float rotationX = 0F;  float rotationY = 0F;  Quaternion originalRotation;  [Header("Target Object")]  public string TargetName;  public UnityEvent IdleEvent;  public UnityEvent HitEvent;  void Start()  {  // Make the rigid body not change rotation  if (GetComponent<Rigidbody>())  GetComponent<Rigidbody>().freezeRotation = true;  originalRotation = transform.localRotation;  }  void Update()  {  if (axes == RotationAxes.MouseXAndY)  {  // Read the mouse input axis  rotationX += Input.GetAxis("Mouse X") \* sensitivityX;  rotationY += Input.GetAxis("Mouse Y") \* sensitivityY;  rotationX = ClampAngle(rotationX, minimumX, maximumX);  rotationY = ClampAngle(rotationY, minimumY, maximumY);  Quaternion xQuaternion = Quaternion.AngleAxis(rotationX, Vector3.up);  Quaternion yQuaternion = Quaternion.AngleAxis(rotationY, -Vector3.right);  transform.localRotation = originalRotation \* xQuaternion \* yQuaternion;  // Raycast untuk mendeteksi objek yang terlihat oleh kamera  Ray ray = Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height / 2, 0));  RaycastHit hit;  if (Physics.Raycast(ray, out hit))  {  if (hit.collider.gameObject.name == TargetName)  {  HitEvent.Invoke();  }  }  else  {  IdleEvent?.Invoke();  }  }  else if (axes == RotationAxes.MouseX)  {  rotationX += Input.GetAxis("Mouse X") \* sensitivityX;  rotationX = ClampAngle(rotationX, minimumX, maximumX);  Quaternion xQuaternion = Quaternion.AngleAxis(rotationX, Vector3.up);  transform.localRotation = originalRotation \* xQuaternion;  }  else  {  rotationY += Input.GetAxis("Mouse Y") \* sensitivityY;  rotationY = ClampAngle(rotationY, minimumY, maximumY);  Quaternion yQuaternion = Quaternion.AngleAxis(-rotationY, Vector3.right);  transform.localRotation = originalRotation \* yQuaternion;  }  }  public static float ClampAngle(float angle, float min, float max)  {  if (angle < -360F)  angle += 360F;  if (angle > 360F)  angle -= 360F;  return Mathf.Clamp(angle, min, max);  }  } |
| --- |

Script ini digunakan untuk mengatur rotasi kamera berdasarkan input mouse (atau input perangkat lainnya) dan melakukan raycast dari kamera ke titik tengah layar untuk mendeteksi objek yang dilihat oleh kamera. Berikut adalah penjelasan komponen utama dari script ini:

* public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }: Ini adalah enumerasi yang digunakan untuk memilih jenis rotasi yang ingin diaktifkan.
* MouseXAndY: Memungkinkan rotasi pada sumbu X dan Y.
* MouseX: Hanya memungkinkan rotasi pada sumbu X.
* MouseY: Hanya memungkinkan rotasi pada sumbu Y.
* [Header("Settings")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan.
* public RotationAxes axes = RotationAxes.MouseXAndY;: Ini adalah variabel yang menentukan jenis rotasi yang akan digunakan. Nilai defaultnya adalah MouseXAndY.
* public float sensitivityX = 15F; dan public float sensitivityY = 15F;: Ini adalah sensitivitas rotasi pada sumbu X dan Y. Semakin tinggi nilainya, semakin cepat kamera akan berputar berdasarkan input mouse.
* public float minimumX = -360F; dan public float maximumX = 360F;: Ini adalah pembatas minimum dan maksimum untuk rotasi pada sumbu X. Mereka mengatur batas putaran horizontal kamera.
* public float minimumY = -60F; dan public float maximumY = 60F;: Ini adalah pembatas minimum dan maksimum untuk rotasi pada sumbu Y. Mereka mengatur batas putaran vertikal kamera.
* float rotationX = 0F; dan float rotationY = 0F;: Ini adalah variabel-variabel yang menyimpan rotasi kamera pada sumbu X dan Y.
* Quaternion originalRotation;: Ini adalah variabel yang menyimpan rotasi awal (default) dari objek yang terpasang komponen ini.
* [Header("Target Object")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan terkait objek target yang akan dideteksi.
* public string TargetName;: Ini adalah nama objek yang akan dideteksi oleh raycast.
* public UnityEvent IdleEvent; dan public UnityEvent HitEvent;: Ini adalah event Unity yang akan dipicu saat kamera tidak mengenali objek target (IdleEvent) dan saat kamera mengenali objek target (HitEvent)
* public static float ClampAngle(float angle, float min, float max): Ini adalah metode statis yang digunakan untuk membatasi nilai sudut (angle) dalam rentang tertentu antara min dan max. Metode ini memastikan bahwa sudut tetap dalam rentang yang diinginkan tanpa melewati batas minimum dan maksimum yang ditentukan.

### 7.2 Eye Pattern

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using System.IO;  public class EyePattern : MonoBehaviour  {  [Header("Settings")]  public string fileName = "eyeTrackingData.txt"; // Nama file yang dapat diatur melalui Inspector  public float saveInterval = 1f; // Interval waktu penyimpanan (1 detik)  public bool showDebug;  private string filePath;  private float saveTimer = 0f;  void Start()  {  // Membuat folder StreamingAssets jika belum ada  string streamingAssetsPath = Application.streamingAssetsPath;  filePath = Path.Combine(streamingAssetsPath, fileName);  Directory.CreateDirectory(streamingAssetsPath);  // Menghapus file jika sudah ada  if (File.Exists(filePath))  {  File.Delete(filePath);  }  }  void Update()  {  saveTimer += Time.deltaTime;  if (saveTimer >= saveInterval)  {  // Mendapatkan posisi mouse dalam pixel layar  Vector3 mousePosition = Input.mousePosition;  // Menyimpan posisi X dan Y ke dalam file di folder StreamingAssets  string dataToSave = "X: " + mousePosition.x + ", Y: " + mousePosition.y + "\n";  File.AppendAllText(filePath, dataToSave);  if (showDebug)  {  Debug.Log(dataToSave);  }  // Reset timer  saveTimer = 0f;  }  }  } |
| --- |

Script ini digunakan untuk melacak posisi mouse pada layar dan menyimpan data posisi tersebut ke dalam sebuah file teks. Berikut adalah penjelasan komponen utama dari script ini:

* [Header("Settings")]: Ini adalah judul yang digunakan dalam Inspector Unity untuk mengelompokkan pengaturan.
* public string fileName = "eyeTrackingData.txt";: Ini adalah nama file teks yang akan digunakan untuk menyimpan data posisi mouse. Nama file ini dapat diatur melalui Inspector.
* public float saveInterval = 1f;: Ini adalah interval waktu (dalam detik) di antara setiap penyimpanan posisi mouse. Nilai defaultnya adalah 1 detik.
* public bool showDebug;: Ini adalah opsi untuk menampilkan data posisi mouse di konsol debug Unity saat data disimpan.
* private string filePath;: Ini adalah variabel yang menyimpan jalur lengkap ke file yang akan digunakan untuk menyimpan data. Jalur ini akan mencakup folder "StreamingAssets" di direktori proyek Unity.
* private float saveTimer = 0f;: Ini adalah timer yang digunakan untuk melacak waktu sejak penyimpanan posisi mouse terakhir kali dilakukan.
* void Start(): Ini adalah metode yang dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya:
* Membuat folder "StreamingAssets" jika belum ada. Folder ini biasanya digunakan untuk menyimpan berkas-berkas yang akan diakses dalam runtime.
* Menyusun jalur lengkap ke file yang akan digunakan dengan menggabungkan jalur folder "StreamingAssets" dan nama file yang ditentukan.
* Jika file dengan nama yang sama sudah ada, file tersebut akan dihapus untuk memastikan bahwa kita akan membuat file yang baru.

Script ini berguna untuk melacak pergerakan mouse dan menyimpan data posisi tersebut ke dalam file teks dengan interval waktu tertentu. Data tersebut dapat digunakan untuk analisis atau tujuan lain dalam pengembangan permainan.

### 7.3 Eye Tracking

| using UnityEngine;  using Tobii.EyeTracking;  using System.IO;  public class EyeTrackingDemo : MonoBehaviour  {  private StreamWriter dataStreamWriter;  private float interval = 1.0f; // Interval waktu untuk menyimpan data (detik)  private float timer = 0f;  void Start()  {  // Pastikan Tobii Eye Tracker telah diinisialisasi  if (EyeTrackingHost.GetInstance().IsConnected)  {  // Aktifkan tracking mata  EyeTrackingHost.GetInstance().EyeTrackingDevice.StartTracking();  }  else  {  Debug.LogError("Tobii Eye Tracker tidak terhubung atau tidak ditemukan.");  }  // Inisialisasi StreamWriter untuk menyimpan data  string filePath = Path.Combine(Application.streamingAssetsPath, "EyeTrackingData.txt");  dataStreamWriter = new StreamWriter(filePath, false);  dataStreamWriter.WriteLine("Timestamp,PosX,PosY");  }  void Update()  {  // Periksa jika Tobii Eye Tracker terhubung  if (EyeTrackingHost.GetInstance().IsConnected)  {  // Baca data mata  EyeTrackingData eyeTrackingData = EyeTrackingHost.GetInstance().EyeTrackingDevice.GetEyeTrackingData();  // Periksa jika mata terdeteksi  if (eyeTrackingData.IsLeftEyeValid && eyeTrackingData.IsRightEyeValid)  {  // Ambil posisi titik pandang mata kanan dan kiri  Vector2 leftEyePosition = eyeTrackingData.LeftEye.GazePointOnDisplayNormalized;  Vector2 rightEyePosition = eyeTrackingData.RightEye.GazePointOnDisplayNormalized;  // Tampilkan hasilnya di Debug.Log  Debug.Log("Posisi Mata Kiri: " + leftEyePosition);  Debug.Log("Posisi Mata Kanan: " + rightEyePosition);  // Tambahkan data ke file setiap interval  timer += Time.deltaTime;  if (timer >= interval)  {  timer = 0f;  SaveEyeTrackingData(Time.time, leftEyePosition, rightEyePosition);  }  }  }  }  void SaveEyeTrackingData(float timestamp, Vector2 leftEyePosition, Vector2 rightEyePosition)  {  // Format data dan tulis ke file  string dataLine = string.Format("{0},{1},{2}", timestamp, leftEyePosition.x, leftEyePosition.y);  dataStreamWriter.WriteLine(dataLine);  dataLine = string.Format("{0},{1},{2}", timestamp, rightEyePosition.x, rightEyePosition.y);  dataStreamWriter.WriteLine(dataLine);  dataStreamWriter.Flush(); // Pastikan data tersimpan dalam file  }  void OnDestroy()  {  // Hentikan tracking mata saat aplikasi berakhir  EyeTrackingHost.GetInstance().EyeTrackingDevice.StopTracking();  // Tutup StreamWriter saat aplikasi berakhir  dataStreamWriter.Close();  }  } |
| --- |

Script ini digunakan untuk mendemonstrasikan penggunaan Tobii Eye Tracker dalam mengambil data posisi pandangan mata pengguna dan menyimpannya ke dalam file teks. Berikut adalah penjelasan komponen utama dari script ini:

* private StreamWriter dataStreamWriter;: Ini adalah objek StreamWriter yang digunakan untuk menulis data ke dalam file teks.
* private float interval = 1.0f;: Ini adalah interval waktu (dalam detik) di antara setiap penyimpanan data posisi pandangan mata.
* private float timer = 0f;: Ini adalah timer yang digunakan untuk melacak waktu sejak data posisi pandangan mata terakhir kali disimpan.
* void Start(): Ini adalah metode yang dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya:
* Dilakukan pemeriksaan apakah Tobii Eye Tracker telah terhubung. Jika tidak terhubung, akan muncul pesan kesalahan di konsol debug.
* Jika Tobii Eye Tracker terhubung, maka tracking mata akan diaktifkan dengan memanggil EyeTrackingHost.GetInstance().EyeTrackingDevice.StartTracking();.
* Inisialisasi objek StreamWriter untuk menyimpan data. File tersebut akan diberi nama "EyeTrackingData.txt" dan akan disimpan di folder "StreamingAssets" dalam proyek Unity.Tulis header kolom ("Timestamp,PosX,PosY") ke dalam file.

Periksa apakah Tobii Eye Tracker terhubung.

* Baca data mata menggunakan EyeTrackingHost.GetInstance().EyeTrackingDevice.GetEyeTrackingData();.
* Periksa apakah mata kiri dan mata kanan terdeteksi (IsLeftEyeValid dan IsRightEyeValid).
* Ambil posisi titik pandang mata kiri dan mata kanan menggunakan LeftEye.GazePointOnDisplayNormalized dan RightEye.GazePointOnDisplayNormalized.
* Tampilkan hasil posisi mata kiri dan mata kanan di konsol debug Unity.
* Tambahkan data ke dalam file setiap interval waktu yang telah ditentukan. Interval ini diatur oleh interval dan dihitung menggunakan timer. Ketika timer mencapai atau melebihi interval, data akan disimpan menggunakan SaveEyeTrackingData().
* void SaveEyeTrackingData(float timestamp, Vector2 leftEyePosition, Vector2 rightEyePosition): Ini adalah metode yang digunakan untuk menyimpan data posisi pandangan mata ke dalam file. Metode ini menerima tiga parameter:
* timestamp: Waktu ketika data diterima.
* leftEyePosition: Posisi titik pandang mata kiri.
* rightEyePosition: Posisi titik pandang mata kanan.
* Metode ini akan memformat data dan menulisnya ke dalam file menggunakan objek StreamWriter.
* Metode Flush() digunakan untuk memastikan data tersimpan dalam file setelah penulisan.
* void OnDestroy(): Ini adalah metode yang dijalankan saat aplikasi atau objek ini dihancurkan. Di dalamnya:
* Tracking mata akan dihentikan dengan memanggil EyeTrackingHost.GetInstance().EyeTrackingDevice.StopTracking(); untuk memastikan bahwa perangkat Tobii Eye Tracker tidak terus mengambil data setelah aplikasi berakhir.
* Objek StreamWriter akan ditutup dengan memanggil Close() untuk memastikan bahwa data telah tersimpan dengan baik sebelum aplikasi berakhir.

Script ini memungkinkan Anda untuk mengambil data posisi pandangan mata pengguna yang ditangkap oleh Tobii Eye Tracker dan menyimpannya dalam file teks untuk keperluan analisis atau tugas lainnya dalam pengembangan permainan.

## 8. Body

Input body adalah jenis input yang menggunakan gerakan tubuh pemain sebagai perintah dalam game. Variasi input body meliputi gerakan tangan, kaki, dan posisi tubuh. Misalnya, pemain dapat mengayunkan tangan untuk menyerang, melangkah ke depan atau ke belakang untuk menggerakkan karakter, atau membungkukkan tubuh untuk berjongkok. Beberapa game juga mendukung gerakan tubuh kompleks seperti melompat, berputar, atau berguling. Teknologi sensor gerak dan kamera 3D diperlukan untuk mendeteksi dan menginterpretasikan gerakan tubuh pemain. Dengan memanfaatkan input body, pengembang dapat menciptakan pengalaman bermain yang lebih fisik dan interaktif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih alami dan intuitif.

### 8.1 Body Detection

| using UnityEngine;  using Windows.Kinect;  public class BodyDetection : MonoBehaviour  {  private KinectSensor kinectSensor;  private BodyFrameReader bodyFrameReader;  private Body[] bodies;  void Start()  {  // Inisialisasi Kinect Sensor  kinectSensor = KinectSensor.GetDefault();  if (kinectSensor != null)  {  bodyFrameReader = kinectSensor.BodyFrameSource.OpenReader();    if (!kinectSensor.IsOpen)  {  kinectSensor.Open();  }  }  }  void Update()  {  if (bodyFrameReader != null)  {  // Baca frame tubuh  BodyFrame bodyFrame = bodyFrameReader.AcquireLatestFrame();    if (bodyFrame != null)  {  if (bodies == null)  {  bodies = new Body[kinectSensor.BodyFrameSource.BodyCount];  }    bodyFrame.GetAndRefreshBodyData(bodies);    // Loop melalui semua tubuh yang terdeteksi  foreach (Body body in bodies)  {  if (body.IsTracked)  {  // Tubuh terdeteksi, lakukan sesuatu di sini  // Anda dapat mengakses posisi sendi-sendi tubuh, seperti body.Joints[JointType.Head]  }  }    bodyFrame.Dispose();  bodyFrame = null;  }  }  }  void OnDestroy()  {  if (bodyFrameReader != null)  {  bodyFrameReader.Dispose();  bodyFrameReader = null;  }    if (kinectSensor != null)  {  if (kinectSensor.IsOpen)  {  kinectSensor.Close();  }    kinectSensor = null;  }  }  } |
| --- |

Script ini digunakan untuk mendeteksi tubuh manusia menggunakan perangkat Kinect dan mengakses data tentang posisi sendi-sendi tubuh yang terdeteksi. Berikut adalah penjelasan komponen utama dari script ini:

* private KinectSensor kinectSensor;: Ini adalah objek yang digunakan untuk mengakses Kinect Sensor. Kinect Sensor adalah perangkat keras yang digunakan untuk mendeteksi tubuh manusia.
* private BodyFrameReader bodyFrameReader;: Objek ini digunakan untuk membaca frame tubuh manusia dari Kinect Sensor.
* private Body[] bodies;: Ini adalah array yang akan digunakan untuk menyimpan data tentang tubuh manusia yang terdeteksi.
* void Start(): Ini adalah metode yang dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya Dilakukan inisialisasi Kinect Sensor dengan memanggil KinectSensor.GetDefault(); untuk mendapatkan perangkat Kinect yang tersedia.

Jika perangkat Kinect ditemukan (kinectSensor != null), maka dibuka dengan memanggil kinectSensor.Open();.

Jika berhasil dibuka, maka bodyFrameReader akan diinisialisasi untuk membaca frame tubuh.

* void Update(): Ini adalah metode yang dijalankan setiap frame dalam game. Di dalamnya:

Pemeriksaan apakah bodyFrameReader telah diinisialisasi. Jika ya, maka frame tubuh manusia akan dibaca dengan memanggil bodyFrameReader.AcquireLatestFrame();.

Jika frame tubuh berhasil dibaca (bodyFrame != null), maka array bodies akan dibuat jika belum ada.

* Data tubuh manusia akan diperoleh dan diperbarui dengan memanggil bodyFrame.GetAndRefreshBodyData(bodies);.

Selanjutnya, terjadi loop melalui semua tubuh yang terdeteksi (foreach (Body body in bodies)).

Jika suatu tubuh sedang terdeteksi (body.IsTracked), Anda dapat melakukan berbagai aksi atau mengakses informasi tentang posisi sendi-sendi tubuh, seperti kepala (body.Joints[JointType.Head]).

Setelah semua data diakses, frame tubuh akan dibebaskan dengan memanggil bodyFrame.Dispose(); dan mengosongkan referensinya.

* void OnDestroy(): Ini adalah metode yang dijalankan saat aplikasi atau objek ini dihancurkan. Di dalamnya:

Script ini memungkinkan Anda untuk menggunakan perangkat Kinect untuk mendeteksi tubuh manusia dan mengakses data tentang posisi sendi-sendi tubuh tersebut, yang dapat digunakan dalam berbagai jenis aplikasi interaktif dan permainan.

### 8.2 Body Skeleton

| using UnityEngine;  using Windows.Kinect;  public class BodySkeleton : MonoBehaviour  {  private KinectSensor kinectSensor;  private BodyFrameReader bodyFrameReader;  private Body[] bodies;  public GameObject jointPrefab; // Prefab untuk menampilkan setiap sendi  private GameObject[] jointObjects;  private const int JointCount = 25; // Jumlah sendi tubuh pada Kinect  void Start()  {  // Inisialisasi Kinect Sensor  kinectSensor = KinectSensor.GetDefault();  if (kinectSensor != null)  {  bodyFrameReader = kinectSensor.BodyFrameSource.OpenReader();    if (!kinectSensor.IsOpen)  {  kinectSensor.Open();  }  }  // Inisialisasi array untuk menampilkan setiap sendi  jointObjects = new GameObject[JointCount];  for (int i = 0; i < JointCount; i++)  {  jointObjects[i] = Instantiate(jointPrefab);  jointObjects[i].SetActive(false);  }  }  void Update()  {  if (bodyFrameReader != null)  {  // Baca frame tubuh  BodyFrame bodyFrame = bodyFrameReader.AcquireLatestFrame();    if (bodyFrame != null)  {  if (bodies == null)  {  bodies = new Body[kinectSensor.BodyFrameSource.BodyCount];  }    bodyFrame.GetAndRefreshBodyData(bodies);    // Loop melalui semua tubuh yang terdeteksi  foreach (Body body in bodies)  {  if (body.IsTracked)  {  // Tubuh terdeteksi, tampilkan tulangnya  for (JointType jt = JointType.SpineBase; jt <= JointType.ThumbRight; jt++)  {  // Ambil posisi sendi dalam ruang 3D  CameraSpacePoint position = body.Joints[jt].Position;    // Ubah posisi sendi menjadi posisi dalam ruang Unity  Vector3 jointPosition = new Vector3(position.X, position.Y, -position.Z);    // Tampilkan setiap sendi sebagai objek  jointObjects[(int)jt].SetActive(true);  jointObjects[(int)jt].transform.position = jointPosition;  }  }  else  {  // Tubuh tidak terdeteksi, sembunyikan semua setiap sendi  for (int i = 0; i < JointCount; i++)  {  jointObjects[i].SetActive(false);  }  }  }    bodyFrame.Dispose();  bodyFrame = null;  }  }  }  void OnDestroy()  {  if (bodyFrameReader != null)  {  bodyFrameReader.Dispose();  bodyFrameReader = null;  }    if (kinectSensor != null)  {  if (kinectSensor.IsOpen)  {  kinectSensor.Close();  }    kinectSensor = null;  }  }  } |
| --- |

Script ini digunakan untuk mendeteksi tubuh manusia dengan menggunakan perangkat Kinect dan menampilkan tulang-tulang tubuh manusia dalam ruang 3D di Unity. Ini adalah bagian penting dalam pembuatan aplikasi atau permainan yang melibatkan interaksi dengan gerakan tubuh manusia. Berikut penjelasan komponen utama dari script ini:

* private KinectSensor kinectSensor;: Objek ini digunakan untuk mengakses Kinect Sensor, yang merupakan perangkat keras untuk mendeteksi gerakan tubuh manusia.
* private BodyFrameReader bodyFrameReader;: Objek ini digunakan untuk membaca frame tubuh manusia dari perangkat Kinect.
* private Body[] bodies;: Ini adalah array yang akan digunakan untuk menyimpan data tentang tubuh manusia yang terdeteksi.
* public GameObject jointPrefab;: Prefab yang akan digunakan untuk menampilkan setiap sendi tubuh manusia. Prefab ini harus diatur melalui Unity Inspector.
* private GameObject[] jointObjects;: Ini adalah array objek GameObject yang akan digunakan untuk menampilkan setiap sendi tubuh manusia. Jumlahnya sesuai dengan jumlah sendi tubuh manusia yang terdeteksi.
* private const int JointCount = 25;: Konstanta yang menentukan jumlah sendi tubuh manusia yang akan ditampilkan. Pada perangkat Kinect, terdapat 25 sendi yang dapat dideteksi.

Script ini memungkinkan Anda untuk menggunakan perangkat Kinect untuk mendeteksi dan menampilkan tubuh manusia dalam Unity, yang dapat digunakan dalam berbagai jenis aplikasi interaktif dan permainan.

## 9. Image

Input image adalah jenis input yang menggunakan gambar atau objek fisik sebagai perintah dalam game. Variasi input image meliputi pengenalan bentuk, warna, dan pola. Misalnya, pemain dapat menggambar simbol dengan tangan untuk mengaktifkan kekuatan khusus, menunjukkan objek berwarna tertentu untuk memilih senjata, atau mengatur pola objek untuk menyelesaikan teka-teki. Beberapa game juga mendukung pengenalan objek 3D, yang memungkinkan pemain untuk berinteraksi dengan game dengan menggunakan objek nyata, seperti boneka, kartu, atau mainan. Teknologi pengenalan gambar dan kamera 3D diperlukan untuk mendeteksi dan menginterpretasikan input image pemain. Dengan memanfaatkan input image, pengembang dapat menciptakan pengalaman bermain yang lebih kreatif dan inovatif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih intuitif dan alami. Selain itu, input image juga dapat digunakan untuk meningkatkan interaktivitas dan imersi dalam game augmented reality (AR).

### 9.1 Image Orientation

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class ImageOrientation : MonoBehaviour  {  public enum CCameraOrientation { Portrait, Landscape }  [Header("Camera Orientation Settings")]  public CCameraOrientation CameraSetting;  public void InvokeSetCameraPortrait()  {  Screen.orientation = ScreenOrientation.Portrait;  }  public void InvokeSetCameraLandscape()  {  Screen.orientation = ScreenOrientation.Landscape;  }  // Start is called before the first frame update  void Start()  {  if (CameraSetting == CCameraOrientation.Landscape)  {  InvokeSetCameraLandscape();  }  else  {  InvokeSetCameraPortrait();  }  }  } |
| --- |

Script ini digunakan untuk mengatur orientasi layar kamera dalam permainan Unity, baik menjadi potret (portrait) atau lanskap (landscape), berdasarkan pengaturan yang telah ditentukan. Ini berguna ketika Anda ingin mengendalikan bagaimana permainan Anda menampilkan tampilan potret atau lanskap secara dinamis. Berikut penjelasan komponen utama dari script ini:

* public enum CCameraOrientation { Portrait, Landscape }: Ini adalah enum (enumeration) yang digunakan untuk menentukan pengaturan orientasi kamera. Enum ini memiliki dua opsi: "Portrait" (potret) dan "Landscape" (lanskap).
* public CCameraOrientation CameraSetting;: Variabel publik yang memungkinkan Anda untuk mengatur orientasi kamera melalui Unity Inspector. Anda dapat memilih antara "Potret" atau "Lanskap" sesuai dengan kebutuhan.
* public void InvokeSetCameraPortrait(): Metode ini digunakan untuk mengatur orientasi layar kamera menjadi potret. Ketika metode ini dipanggil, orientasi layar diubah menjadi ScreenOrientation.Portrait.
* public void InvokeSetCameraLandscape(): Metode ini digunakan untuk mengatur orientasi layar kamera menjadi lanskap. Ketika metode ini dipanggil, orientasi layar diubah menjadi ScreenOrientation.Landscape.
* void Start(): Metode ini dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya:
* Dilakukan pengecekan CameraSetting untuk menentukan apakah orientasi kamera harus menjadi potret atau lanskap.
* Jika CameraSetting adalah CCameraOrientation.Landscape, maka metode InvokeSetCameraLandscape() akan dipanggil, sehingga orientasi layar diatur menjadi lanskap.
* Jika CameraSetting adalah CCameraOrientation.Portrait, maka metode InvokeSetCameraPortrait() akan dipanggil, sehingga orientasi layar diatur menjadi potret.

Ini adalah script yang sederhana namun berguna untuk mengatur orientasi kamera dalam permainan Unity berdasarkan preferensi pengembang atau kebutuhan permainan tertentu. Dengan script ini, Anda dapat mengendalikan tampilan potret atau lanskap permainan Anda sesuai keinginan.

### 9.2 Image Dialog

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class ImageDialog : MonoBehaviour  {  private RawImage rawImage;  // Start is called before the first frame update  void Start()  {  rawImage = GetComponent<RawImage>();  }  public void InvokeImageOpen()  {  // Menggunakan File Explorer untuk memilih file gambar  string imagePath = UnityEditor.EditorUtility.OpenFilePanel("Select Image", "", "png,jpg,jpeg");  // Memuat gambar dari file yang telah dipilih  if (imagePath.Length != 0)  {  Texture2D texture = new Texture2D(2, 2);  byte[] imageData = System.IO.File.ReadAllBytes(imagePath);  texture.LoadImage(imageData);  // Menampilkan gambar pada sebuah komponen "Raw Image" pada game object  rawImage.texture = texture;  }  }  } |
| --- |

Script ini digunakan untuk membuka dan menampilkan gambar dari file yang dipilih oleh pengguna melalui File Explorer. Script ini biasanya digunakan dalam permainan atau aplikasi Unity yang memungkinkan pengguna untuk memilih gambar dan menampilkannya dalam sebuah komponen RawImage. Berikut penjelasan komponen utama dari script ini:

* private RawImage rawImage;: Ini adalah variabel pribadi yang akan menyimpan referensi ke komponen RawImage pada game object yang sama dengan script ini. Komponen RawImage digunakan untuk menampilkan gambar dalam permainan Unity.
* void Start(): Metode ini dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya:
* Script mencari dan menyimpan referensi ke komponen RawImage pada game object yang sama. Ini dilakukan dengan menggunakan GetComponent<RawImage>().
* public void InvokeImageOpen(): Metode ini digunakan untuk membuka jendela File Explorer sehingga pengguna dapat memilih file gambar. Ini adalah metode yang biasanya dipanggil melalui sebuah tombol atau aksi dalam permainan ketika pengguna ingin memilih gambar.
* string imagePath = UnityEditor.EditorUtility.OpenFilePanel("Select Image", "", "png,jpg,jpeg");: Ini adalah perintah yang membuka File Explorer untuk memilih gambar. Pengguna akan melihat jendela dialog yang memungkinkan mereka memilih file gambar dengan ekstensi .png, .jpg, atau .jpeg. imagePath akan berisi path file gambar yang dipilih.
* Setelah pengguna memilih file gambar, gambar tersebut akan dimuat dan ditampilkan dalam permainan.
* Texture2D texture = new Texture2D(2, 2);: Ini adalah pembuatan objek Texture2D yang akan digunakan untuk memuat gambar.
* byte[] imageData = System.IO.File.ReadAllBytes(imagePath);: Ini adalah membaca seluruh data gambar dari file yang telah dipilih.
* texture.LoadImage(imageData);: Ini adalah memuat gambar ke dalam objek Texture2D dari data gambar yang telah dibaca.
* rawImage.texture = texture;: Ini adalah mengatur texture pada komponen RawImage dengan Texture2D yang telah dimuat. Sehingga gambar akan ditampilkan dalam komponen RawImage.

Script ini memungkinkan pengguna untuk memilih gambar dari sistem file mereka dan menampilkannya dalam permainan Unity. Biasanya, script ini digunakan dalam konteks aplikasi atau permainan yang memerlukan pengguna untuk mengunggah atau memilih gambar sebagai bagian dari fungsionalitas permainan atau aplikasi tersebut.

### 9.3 Image Camera

| using UnityEngine;  using UnityEngine.UI;  using System.Collections.Generic;  using UnityEngine.Events;  public class ImageCamera : MonoBehaviour  {  [Header("WebCamera List")]  public int WebCamIndex = 0;  public List<string> availableDevices = new List<string>();  public WebCamDevice[] devices;  private WebCamTexture webCameraTexture;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  private RawImage rawImage;  void Start()  {  StartEvents?.Invoke();  }  public void Stop()  {  webCameraTexture.Stop();  }  public void Play()  {  devices = WebCamTexture.devices;  foreach (var device in devices)  {  availableDevices.Add(device.name);  }  rawImage = GetComponent<RawImage>();  webCameraTexture = new WebCamTexture(devices[WebCamIndex].name);  webCameraTexture.requestedWidth = 640; // Atur sesuai kebutuhan  webCameraTexture.requestedHeight = 480; // Atur sesuai kebutuhan  webCameraTexture.Play();  }  public void InvokeImageCamera()  {  devices = WebCamTexture.devices;  foreach (var device in devices)  {  availableDevices.Add(device.name);  }  rawImage = GetComponent<RawImage>();  webCameraTexture = new WebCamTexture(devices[WebCamIndex].name);  webCameraTexture.requestedWidth = 640; // Atur sesuai kebutuhan  webCameraTexture.requestedHeight = 480; // Atur sesuai kebutuhan  webCameraTexture.Play();  }  void Update()  {  if (webCameraTexture.isPlaying)  {  rawImage.texture = webCameraTexture;  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini digunakan untuk menampilkan feed kamera webcam atau perangkat kamera lainnya di dalam sebuah komponen RawImage dalam permainan Unity. Ini memungkinkan Anda untuk mengintegrasikan tampilan kamera langsung ke dalam permainan Anda. Berikut penjelasan tentang komponen utama dari script ini:

* private WebCamTexture webCameraTexture;: Ini adalah variabel pribadi yang akan menyimpan referensi ke objek WebCamTexture. WebCamTexture adalah kelas dalam Unity yang digunakan untuk menangkap video dari kamera perangkat.
* private RawImage rawImage;: Ini adalah variabel pribadi yang akan menyimpan referensi ke komponen RawImage pada game object yang sama dengan script ini. Komponen RawImage digunakan untuk menampilkan tampilan dari kamera.
* void Start(): Metode ini akan dijalankan saat objek dengan komponen ini pertama kali aktif dalam permainan. Di dalamnya, InvokeImageCamera() akan dipanggil untuk memulai feed kamera saat permainan dimulai.
* public void Stop(): Metode ini digunakan untuk menghentikan feed kamera. Ini akan memanggil webCameraTexture.Stop() untuk menghentikan perangkat kamera yang sedang berjalan.
* public void Play(): Metode ini digunakan untuk memulai feed kamera. Di dalamnya:
* rawImage = GetComponent<RawImage>();: Ini adalah mendapatkan komponen RawImage dari game object yang sama dengan script ini.
* webCameraTexture = new WebCamTexture();: Ini adalah membuat objek WebCamTexture yang akan digunakan untuk menangkap feed kamera.
* webCameraTexture.requestedWidth dan webCameraTexture.requestedHeight digunakan untuk mengatur resolusi feed kamera yang diinginkan. Anda dapat menyesuaikannya sesuai dengan kebutuhan.
* webCameraTexture.Play();: Ini adalah memulai feed kamera sehingga kamera mulai menangkap video.
* public void InvokeImageCamera(): Metode ini mirip dengan Play(), digunakan untuk memulai feed kamera. Ini juga digunakan pada Start() untuk memulai feed kamera saat permainan dimulai.

Dengan menggunakan script ini, Anda dapat dengan mudah menambahkan tampilan dari kamera ke dalam permainan Anda, yang dapat berguna untuk berbagai aplikasi seperti permainan augmented reality, pengenalan objek, atau tampilan langsung dari kamera dalam permainan.

### 9.4 Image Capture

| using System;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ImageCapture : MonoBehaviour  {  [Header("Capture Setting")]  public string Path;  public int Resolution;  string filename;  [Header("UI Objects Setting")]  public List<GameObject> UIObjects;  [Header("After Capture Setting")]  public UnityEvent AfterCaptureEvent;  void WaitCapture()  {  filename = Path + "/" + DateTime.Now.ToString("MM\_dd\_yyyy\_h\_mm\_ss") + ".jpg";  ScreenCapture.CaptureScreenshot(filename, Resolution);  Debug.Log(filename);  }  void ShowObjects()  {  SetStatusObjects(true);  }  void HideObjects()  {  SetStatusObjects(false);  }  void SetStatusObjects(bool aStatus)  {  for (int i = 0; i < UIObjects.Count; i++)  {  UIObjects[i].SetActive(aStatus);  }  }  public void InvokeCameraCapture()  {  HideObjects();  WaitCapture();  Invoke("ShowObjects", 1);  AfterCaptureEvent.Invoke();  }  } |
| --- |

Script ini digunakan untuk mengambil tangkapan layar (screenshot) dari permainan Unity dan menyimpannya sebagai file gambar dengan format JPG. Script ini juga memberikan kontrol untuk menampilkan dan menyembunyikan objek UI tertentu sebelum dan setelah tangkapan layar diambil. Berikut adalah penjelasan komponen utama dari script ini:

* Path: String yang menentukan lokasi penyimpanan file tangkapan layar. Ini harus merupakan direktori relatif dalam proyek Unity. Misalnya, "Assets/Captures".
* Resolution: Resolusi tangkapan layar yang diinginkan. Ini adalah lebar dalam piksel, dan tinggi akan disesuaikan sesuai dengan rasio aspek permainan. Misalnya, 1920 untuk resolusi lebar 1920 piksel.
* UIObjects: Daftar objek GameObject yang akan disetel untuk ditampilkan atau disembunyikan sebelum dan setelah tangkapan layar. Objek-objek ini adalah elemen UI yang mungkin ingin Anda sembunyikan saat mengambil tangkapan layar.
* AfterCaptureEvent: UnityEvent yang dapat Anda konfigurasi untuk menjalankan tindakan tertentu setelah tangkapan layar diambil. Ini memungkinkan Anda menghubungkan berbagai fungsi atau peristiwa yang harus terjadi setelah tangkapan layar selesai.
* WaitCapture(): Metode ini digunakan untuk menunggu sebentar sebelum mengambil tangkapan layar. Ini mengonstruksi nama file tangkapan layar berdasarkan format tanggal dan waktu, dan kemudian menggunakan ScreenCapture.CaptureScreenshot untuk mengambil tangkapan layar.
* ShowObjects(): Metode ini digunakan untuk menampilkan objek-objek UI yang mungkin telah disembunyikan sebelum mengambil tangkapan layar. Ini memanggil SetStatusObjects(true).
* HideObjects(): Metode ini digunakan untuk menyembunyikan objek-objek UI yang mungkin ingin disembunyikan sebelum mengambil tangkapan layar. Ini memanggil SetStatusObjects(false).
* SetStatusObjects(bool aStatus): Metode ini digunakan untuk mengatur status tampilan (aktif/non-aktif) objek-objek UI. Ini digunakan untuk menyembunyikan atau menampilkan objek-objek UI sebelum dan setelah pengambilan tangkapan layar.

### 9.5 Image Carousel

| using UnityEngine;  using UnityEngine.UI;  public class ImageCarousel : MonoBehaviour  {  [Header("Main Settings")]  public Image[] images;  public Button back;  public Button next;  [Header("Index Settings")]  public int currentIndex = 0;  private void Start()  {  // Pastikan hanya Image dengan index 0 yang aktif  for (int i = 0; i < images.Length; i++)  {  images[i].gameObject.SetActive(i == 0);  }  // Tambahkan listener ke tombol  back.onClick.AddListener(ShowPreviousImage);  next.onClick.AddListener(ShowNextImage);  }  public void ShowNextImage()  {  if (images == null || images.Length == 0) return;  // Nonaktifkan Image saat ini  images[currentIndex].gameObject.SetActive(false);  // Update index  currentIndex = (currentIndex + 1) % images.Length;  // Aktifkan Image berikutnya  images[currentIndex].gameObject.SetActive(true);  UpdateBackButtonState();  UpdateNextButtonState();  }  public void ShowPreviousImage()  {  if (images == null || images.Length == 0) return;  // Nonaktifkan Image saat ini  images[currentIndex].gameObject.SetActive(false);  // Update index  currentIndex--;  if (currentIndex < 0)  {  currentIndex = images.Length - 1;  }  // Aktifkan Image sebelumnya  images[currentIndex].gameObject.SetActive(true);  UpdateBackButtonState();  UpdateNextButtonState();  }  public void ShowImageByIndex(int index)  {  if (images == null || images.Length == 0) return;  if (index < 0 || index >= images.Length)  {  Debug.LogError("Index di luar rentang!");  return;  }  // Nonaktifkan semua Image  foreach (var img in images)  {  img.gameObject.SetActive(false);  }  // Set index ke index yang diberikan dan aktifkan Image tersebut  currentIndex = index;  images[currentIndex].gameObject.SetActive(true);  }  private void UpdateNextButtonState()  {  if (currentIndex == images.Length - 1)  {  next.gameObject.SetActive(false); // Nonaktifkan tombol next  }  else  {  next.gameObject.SetActive(true); // Nonaktifkan tombol next  }  }  private void UpdateBackButtonState()  {  if (currentIndex == 0)  {  back.gameObject.SetActive(false); // Nonaktifkan tombol next  }  else  {  back.gameObject.SetActive(true); // Nonaktifkan tombol next  }  }  } |
| --- |

## 10. Sensor

Input sensor adalah jenis input yang menggunakan sensor fisik untuk mendeteksi perubahan lingkungan atau gerakan pemain sebagai perintah dalam game. Variasi input sensor meliputi sensor gerakan, sensor kecepatan, sensor tekanan, dan sensor suhu. Misalnya, sensor gerakan dapat digunakan untuk mengendalikan karakter atau objek dalam game dengan menggerakkan perangkat, seperti ponsel atau controller. Sensor kecepatan dapat digunakan untuk mengendalikan kecepatan karakter atau kendaraan. Sensor tekanan dapat digunakan untuk mengendalikan kekuatan serangan atau lompatan. Sensor suhu dapat digunakan untuk mendeteksi suhu lingkungan dan mengadaptasi gameplay sesuai. Perangkat seperti Arduino dapat digunakan untuk membuat controller khusus dengan sensor yang terintegrasi, dan kemudian diprogram untuk berinteraksi dengan game. Dengan memanfaatkan input sensor, pengembang dapat menciptakan pengalaman bermain yang lebih dinamis dan responsif, serta memungkinkan pemain untuk berinteraksi dengan game dengan cara yang lebih fisik dan alami. Selain itu, input sensor juga dapat digunakan untuk meningkatkan realisme dan imersi dalam game simulasi atau game realitas virtual (VR).

### 10.1 Arduino Reader

| using UnityEngine;  using System.IO.Ports;  public class ArduinoReader : MonoBehaviour  {  public string portName = "COM3"; // Ganti dengan nama port Arduino yang sesuai  public int baudRate = 9600; // Sesuaikan baud rate dengan Arduino  private SerialPort serialPort;  private void Start()  {  // Inisialisasi koneksi serial  serialPort = new SerialPort(portName, baudRate);  try  {  // Buka koneksi serial  serialPort.Open();  }  catch (System.Exception ex)  {  Debug.LogError("Error opening serial port: " + ex.Message);  }  }  private void Update()  {  if (serialPort != null && serialPort.IsOpen)  {  try  {  // Baca data dari Arduino  string data = serialPort.ReadLine();  Debug.Log("Data from Arduino: " + data);  // Di sini Anda dapat melakukan sesuatu dengan data yang telah Anda baca  }  catch (System.Exception ex)  {  Debug.LogError("Error reading from serial port: " + ex.Message);  }  }  }  private void OnDestroy()  {  // Tutup koneksi serial saat objek dihancurkan  if (serialPort != null && serialPort.IsOpen)  {  serialPort.Close();  }  }  } |
| --- |

Script ini digunakan untuk membaca data yang dikirimkan oleh Arduino melalui koneksi serial (UART) dan menampilkan data tersebut di Unity. Ini memungkinkan komunikasi dua arah antara Unity dan Arduino melalui port serial. Berikut penjelasan lebih detail mengenai script ini:

* portName: Nama port serial yang digunakan untuk menghubungkan ke Arduino. Misalnya, jika Arduino terhubung ke port "COM3" pada Windows atau "/dev/ttyUSB0" pada Linux, maka Anda perlu mengatur nilai ini sesuai dengan port yang digunakan.
* baudRate: Baud rate (kecepatan transmisi) yang digunakan dalam komunikasi serial antara Unity dan Arduino. Baud rate ini harus sesuai dengan pengaturan yang digunakan pada program Arduino. Misalnya, jika Anda mengatur Arduino dengan baud rate 9600, Anda juga harus mengatur nilai ini menjadi 9600.
* Objek SerialPort: serialPort: Objek yang digunakan untuk mengelola komunikasi serial. Ini digunakan untuk membuka, membaca, dan menutup koneksi serial.
* Metode Start(): Metode ini dipanggil saat objek Unity pertama kali diaktifkan. Pada saat ini, objek SerialPort akan diinisialisasi dan koneksi serial akan dibuka. Jika terjadi kesalahan saat membuka koneksi, pesan kesalahan akan ditampilkan di Debug Log.
* Metode Update(): Metode ini dipanggil setiap frame. Pada setiap pemanggilan, script akan mencoba untuk membaca data yang dikirimkan oleh Arduino melalui koneksi serial.Data dari Arduino dibaca menggunakan serialPort.ReadLine(), yang akan mengambil satu baris data dari buffer masukan serial. Data ini kemudian ditampilkan dalam Debug Log Unity. Anda dapat menambahkan lebih banyak logika di dalam metode ini untuk menangani data yang dibaca dari Arduino sesuai dengan kebutuhan Anda.

### 10.2 Arduino Writer

| using UnityEngine;  using System.IO.Ports;  public class ArduinoCommunication : MonoBehaviour  {  public string portName = "COM3"; // Sesuaikan dengan port USB Arduino Anda  public int baudRate = 9600; // Sesuaikan dengan baud rate Arduino Anda  private SerialPort serialPort;  void Start()  {  // Inisialisasi koneksi serial  serialPort = new SerialPort(portName, baudRate);  serialPort.Open();  // Pastikan port serial terbuka  if (serialPort.IsOpen)  {  Debug.Log("Koneksi serial terbuka pada " + portName + " dengan baud rate " + baudRate);  }  else  {  Debug.LogError("Gagal membuka koneksi serial pada " + portName);  }  }  void Update()  {  // Contoh pengiriman data ke Arduino  if (Input.GetKeyDown(KeyCode.Space))  {  SendDataToArduino("Hello, Arduino!");  }  }  void SendDataToArduino(string data)  {  if (serialPort != null && serialPort.IsOpen)  {  serialPort.Write(data);  Debug.Log("Data terkirim ke Arduino: " + data);  }  else  {  Debug.LogError("Koneksi serial ke Arduino belum terbuka atau terputus.");  }  }  void OnDestroy()  {  // Tutup port serial saat aplikasi berakhir  if (serialPort != null && serialPort.IsOpen)  {  serialPort.Close();  }  }  } |
| --- |

Script ini digunakan untuk mengatur komunikasi antara Unity dan Arduino melalui koneksi serial (UART). Dengan script ini, Anda dapat mengirim data dari Unity ke Arduino dan sebaliknya. Berikut adalah penjelasan lebih lanjut tentang script ini:

* Variabel Port Serial:portName: Nama port serial yang digunakan untuk menghubungkan ke Arduino. Anda perlu mengatur nilai ini sesuai dengan port yang digunakan oleh Arduino Anda, misalnya, "COM3" di Windows atau "/dev/ttyUSB0" di Linux.
* baudRate: Baud rate (kecepatan transmisi) yang digunakan dalam komunikasi serial antara Unity dan Arduino. Nilai ini harus sesuai dengan pengaturan baud rate yang Anda tentukan dalam kode Arduino Anda. Biasanya, nilai ini diatur ke 9600, tetapi Anda harus memastikan bahwa ini cocok dengan pengaturan Arduino Anda.
* Objek SerialPort: serialPort: Objek ini digunakan untuk mengelola koneksi serial, membuka, menutup, dan mengirim data melalui port serial.
* Metode Start(): Metode ini dipanggil saat objek Unity pertama kali diaktifkan. Pada saat ini, objek SerialPort akan diinisialisasi dan koneksi serial akan dibuka. Setelah koneksi terbuka, script akan mencoba untuk menampilkan pesan di Debug Log yang memberi tahu Anda bahwa koneksi serial telah berhasil dibuka atau jika ada masalah saat membukanya.
* Metode Update(): Metode ini dipanggil setiap frame. Pada contoh ini, jika tombol spasi (KeyCode.Space) ditekan, maka metode SendDataToArduino() akan dipanggil dengan string "Hello, Arduino!" sebagai data yang akan dikirim ke Arduino.

# 

# 

# Chapter 5 Event Classification

Salah satu aspek terpenting dalam pengembangan game adalah event game yang merupakan aturan dan prosedur yang mengatur bagaimana game dimainkan. Terdapat sepuluh jenis event game yang berbeda: Calculation, Activation, Visualization, Interaction, Transformation, Animation, Instantiation, Collision, Iteration, dan Condition. Setiap event ini memainkan peran penting dalam pengembangan game dan membantu menciptakan pengalaman yang menarik dan menantang bagi pemain.

Event Calculation mencakup semua perhitungan matematis yang terjadi di balik layar saat bermain game. Ini bisa melibatkan perhitungan damage yang diterima karakter dari serangan musuh, penghitungan kecepatan karakter, atau bahkan perhitungan probabilitas untuk acara acak dalam game. Mengerti bagaimana menghitung dan mengoptimalkan perhitungan ini sangat penting untuk menciptakan gameplay yang adil dan menantang.

Event Activation terkait dengan bagaimana objek atau karakter dalam game diaktifkan atau dideaktifkan. Ini bisa melibatkan mengaktifkan perangkap saat pemain mendekat, atau mengaktifkan cutscene saat pemain mencapai titik tertentu dalam game. Pengertian mendalam tentang event Activation ini adalah kunci untuk menciptakan dunia game yang interaktif dan dinamis.Visualization adalah tentang bagaimana informasi disajikan kepada pemain melalui antarmuka pengguna atau UI. Ini bisa mencakup segala hal dari tampilan HUD (Head-Up Display), menu, peta, dan sebagainya. UI yang baik sangat penting untuk pengalaman bermain game yang baik, karena ini adalah cara utama pemain menerima informasi tentang apa yang terjadi dalam game.

Transformation terkait dengan perubahan status objek atau karakter dalam game. Ini bisa melibatkan transformasi fisik seperti ukuran atau bentuk objek, atau transformasi status seperti karakter yang naik level atau mendapatkan kekuatan baru. Memahami dan memanfaatkan Transformation ini adalah kunci untuk menciptakan gameplay yang menarik dan menantang.Event Instantiation, Collision, Iteration, dan Condition terkait dengan penciptaan objek, deteksi tabrakan antar objek, pengulangan tindakan, dan kondisi untuk pemicu tertentu dalam game. Instantiation melibatkan penciptaan objek baru dalam game, seperti musuh atau item. Collision adalah tentang bagaimana game mendeteksi dan menangani tabrakan antara objek. Iteration melibatkan pengulangan tindakan atau peristiwa dalam game, seperti gelombang musuh yang berulang. Dan Condition adalah tentang bagaimana game menanggapi keadaan tertentu, seperti apa yang terjadi saat pemain kehabisan kesehatan.

****

**Gambar 6. Game Event Classification**

## 1. Calculation

Event Calculation dalam game sering kali terkait dengan logika pemrograman yang diperlukan untuk melakukan perhitungan matematis. Misalnya, saat menciptakan game RPG, pemrogram mungkin perlu menghitung damage yang diberikan oleh karakter pemain kepada musuh berdasarkan statistik karakter seperti kekuatan, tingkat, dan jenis senjata yang digunakan. Pemrograman untuk event Calculation ini akan melibatkan penggunaan variabel, operator matematika, dan fungsi khusus untuk menghitung nilai yang diperlukan. Sebagai contoh, sebuah fungsi dalam bahasa pemrograman bisa menerima statistik karakter dan senjata yang digunakan sebagai input, lalu menghitung dan mengembalikan jumlah damage yang harus diberikan kepada musuh. Memahami bagaimana menulis dan mengoptimalkan kode untuk melakukan perhitungan ini adalah kunci untuk menciptakan gameplay yang adil dan menantang.

### 1.1 Calculation Boolean

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationBoolean : MonoBehaviour  {  [Header("Current Value")]  public bool CurrentValue;  public bool ForceUpdate;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent TrueEvents;  public UnityEvent FalseEvents;  bool lastStatus;  bool somethingChange;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  lastStatus = CurrentValue;  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (lastStatus != CurrentValue)  {  somethingChange = true;  lastStatus = CurrentValue;  }  if (somethingChange)  {  if (CurrentValue) TrueEvents?.Invoke();  else FalseEvents?.Invoke();  somethingChange = false;  }  if (ForceUpdate)  {  if (CurrentValue) TrueEvents?.Invoke();  else FalseEvents?.Invoke();  }  }  public void SaveData()  {  PlayerPrefs.SetInt(transform.gameObject.name, CurrentValue ? 1 : 0 );  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetInt(transform.name) == 1 ? true : false;  Debug.Log(CurrentValue);  }  }  public void SetCurrentValue(bool aValue)  {  CurrentValue = aValue;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(Toggle aValue)  {  aValue.isOn = CurrentValue;  }  public void ReadFrom(Toggle aValue)  {  CurrentValue = aValue.isOn;  }  } |
| --- |

Script ini digunakan untuk mengelola perhitungan dan manipulasi data boolean dalam Unity, serta menyediakan integrasi dengan UI. Berikut adalah penjelasan tentang bagian-bagian utama dari script ini:

* Variabel CurrentValue:

Variabel ini digunakan untuk menyimpan nilai boolean saat ini yang akan dihitung dan dimanipulasi oleh script ini.

* Variabel ForceUpdate:

Variabel ini adalah boolean yang, jika diaktifkan (diatur menjadi true), akan memaksa pembaruan ke event yang sesuai saat ada perubahan nilai CurrentValue.

* Event Events:

Script ini memiliki beberapa event yang dapat dipicu pada berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

TrueEvents: Event yang dipicu jika CurrentValue saat ini adalah true.

FalseEvents: Event yang dipicu jika CurrentValue saat ini adalah false.

* Variabel Tambahan:

lastStatus: Variabel ini digunakan untuk melacak nilai CurrentValue pada frame sebelumnya. Ini digunakan untuk mendeteksi perubahan nilai.

* somethingChange: Boolean yang menandakan apakah terjadi perubahan pada CurrentValue sejak frame sebelumnya.
* Metode Start():

Metode ini dipanggil saat objek pertama kali aktif. Pada saat ini, event StartEvents dipicu dan nilai lastStatus diatur ke nilai awal CurrentValue.

* Metode Update():

Metode ini dipanggil setiap frame saat objek ini aktif. Pada setiap pemanggilan, event UpdateEvents dipicu.

Selanjutnya, metode ini memeriksa apakah nilai CurrentValue telah berubah sejak frame sebelumnya dengan membandingkannya dengan lastStatus. Jika ada perubahan, somethingChange diatur sebagai true.

* Jika somethingChange adalah true, maka event yang sesuai (TrueEvents jika CurrentValue adalah true, atau FalseEvents jika CurrentValue adalah false) akan dipicu, dan somethingChange diatur kembali menjadi false.
* Jika ForceUpdate diaktifkan, maka event yang sesuai akan selalu dipicu pada setiap pembaruan frame tanpa memeriksa perubahan CurrentValue.
* Metode SaveData():

Metode ini digunakan untuk menyimpan nilai CurrentValue ke PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode LoadData():

Metode ini digunakan untuk memuat nilai CurrentValue dari PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode SetCurrentValue(bool aValue):

Metode ini memungkinkan pengaturan manual nilai CurrentValue ke nilai yang diberikan (aValue).

* Metode WriteTo():

Tiga metode WriteTo berbeda yang memungkinkan Anda menampilkan nilai CurrentValue di UI:

WriteTo(Text aValue): Mengubah teks dari komponen UI Text dengan nilai CurrentValue.

WriteTo(TextMesh aValue): Mengubah teks dari komponen UI TextMesh dengan nilai CurrentValue.

WriteTo(Toggle aValue): Mengatur status Toggle UI sesuai dengan nilai CurrentValue.

### 1.2 Calculation Integer

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationInteger : MonoBehaviour  {  [Header("Current Value")]  public int CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  int TargetInteger;  bool StartTransfer = false;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (StartTransfer)  {  CurrentValue += 1;  if (CurrentValue > TargetInteger)  {  CurrentValue = TargetInteger;  StartTransfer = false;  }  }  }  public void SaveData()  {  PlayerPrefs.SetInt(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetInt(transform.name);  }  }  public void SetCurrentValue(int aValue)  {  CurrentValue = aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void AddToCurrentValue(int aValue)  {  CurrentValue += aValue;  }  public void SubtractFromCurrentValue(int aValue)  {  CurrentValue -= aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void WriteTo(Slider aValue)  {  aValue.value = CurrentValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void TransferToCurrentValue(int aValue)  {  TargetInteger = CurrentValue + aValue;  StartTransfer = true;  Invoke("ThreeSecond", 3);  }  public void ReadFrom(Slider aValue)  {  CurrentValue = Mathf.RoundToInt(aValue.value);  }  public void ReadFrom(InputField aValue)  {  CurrentValue = int.Parse(aValue.text);  }  public void ReadFrom(Text aValue)  {  CurrentValue = int.Parse(aValue.text);  }  public void ReadFrom(TextMesh aValue)  {  CurrentValue = int.Parse(aValue.text);  }  void ThreeSecond()  {  CurrentValue = TargetInteger;  }  } |
| --- |

### 

Script ini digunakan untuk mengelola perhitungan dan manipulasi data integer dalam Unity, serta menyediakan integrasi dengan UI. Berikut adalah penjelasan tentang bagian-bagian utama dari script ini:

* Variabel CurrentValue:

Variabel ini digunakan untuk menyimpan nilai integer saat ini yang akan dihitung dan dimanipulasi oleh script ini.

* Event Events:

Script ini memiliki dua event yang dapat dipicu pada berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

* Variabel Tambahan:

TargetInteger: Variabel yang digunakan untuk menyimpan nilai target yang akan dicapai oleh CurrentValue.

StartTransfer: Boolean yang menandakan apakah proses transfer CurrentValue sedang berlangsung.

* Metode Start():

Metode ini dipanggil saat objek pertama kali aktif. Pada saat ini, event StartEvents dipicu.

* Metode Update():

Metode ini dipanggil setiap frame saat objek ini aktif. Pada setiap pemanggilan, event UpdateEvents dipicu.

Jika StartTransfer aktif, CurrentValue akan bertambah satu setiap frame sampai mencapai TargetInteger. Begitu mencapai target, StartTransfer akan diubah menjadi false.

* Metode SaveData():

Metode ini digunakan untuk menyimpan nilai CurrentValue ke PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode LoadData():

Metode ini digunakan untuk memuat nilai CurrentValue dari PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode SetCurrentValue(int aValue):

Metode ini memungkinkan pengaturan manual nilai CurrentValue ke nilai yang diberikan (aValue).

Jika nilai aValue kurang dari atau sama dengan 0, CurrentValue diatur ke 0.

* Metode AddToCurrentValue(int aValue):

Metode ini digunakan untuk menambahkan nilai aValue ke CurrentValue.

* Metode SubtractFromCurrentValue(int aValue):

Metode ini digunakan untuk mengurangkan nilai aValue dari CurrentValue.

Jika nilai CurrentValue menjadi kurang dari atau sama dengan 0 setelah pengurangan, maka CurrentValue diatur ke 0.

* Metode WriteTo():

Empat metode WriteTo berbeda yang memungkinkan Anda menampilkan nilai CurrentValue di UI:

WriteTo(Slider aValue): Mengatur nilai Slider UI sesuai dengan nilai CurrentValue.

WriteTo(InputField aValue): Mengatur teks dari komponen UI InputField dengan nilai CurrentValue sebagai teks.

WriteTo(Text aValue): Mengatur teks dari komponen UI Text dengan nilai CurrentValue sebagai teks.

WriteTo(TextMesh aValue): Mengatur teks dari komponen UI TextMesh dengan nilai CurrentValue sebagai teks.

* Metode TransferToCurrentValue(int aValue):

Metode ini digunakan untuk memulai proses transfer nilai CurrentValue ke TargetInteger dengan menambahkannya sebesar aValue.

StartTransfer diatur ke true dan akan mencapai target dalam tiga detik, dan kemudian menghentikan proses transfer.

### 1.3 Calculation Float

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationFloat : MonoBehaviour  {  [Header("Current Value")]  public float CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  float TargetFloat;  bool StartTransfer = false;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (StartTransfer)  {  CurrentValue += 1f;  if (CurrentValue > TargetFloat)  {  CurrentValue = TargetFloat;  StartTransfer = false;  }  }  }  public void SaveData()  {  PlayerPrefs.SetFloat(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetFloat(transform.name);  }  }  public void LoadData(string aName)  {  if (PlayerPrefs.HasKey(aName))  {  CurrentValue = PlayerPrefs.GetFloat(aName);  }  }  public void SetCurrentValue(float aValue)  {  CurrentValue = aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void AddToCurrentValue(float aValue)  {  CurrentValue += aValue;  }  public void SubtractFromCurrentValue(float aValue)  {  CurrentValue -= aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void WriteTo(Slider aValue)  {  aValue.value = CurrentValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(AudioSource aValue)  {  aValue.volume = CurrentValue;  }  public void TransferToCurrentValue(float aValue)  {  TargetFloat = CurrentValue + aValue;  StartTransfer = true;  Invoke("ThreeSecond", 3);  }  public void ReadFrom(Slider aValue)  {  CurrentValue = aValue.value;  }  public void ReadFrom(InputField aValue)  {  CurrentValue = float.Parse(aValue.text);  }  public void ReadFrom(Text aValue)  {  CurrentValue = float.Parse(aValue.text);  }  public void ReadFrom(TextMesh aValue)  {  CurrentValue = float.Parse(aValue.text);  }  void ThreeSecond()  {  CurrentValue = TargetFloat;  }  } |
| --- |

Script ini mirip dengan skrip CalculationInteger, tetapi berfokus pada manipulasi dan perhitungan nilai float. Ini memungkinkan Anda untuk mengontrol dan menampilkan nilai float dalam Unity serta mengintegrasikannya dengan antarmuka pengguna. Berikut adalah penjelasan tentang bagian utama dari skrip ini:

* Variabel CurrentValue:

Variabel ini digunakan untuk menyimpan nilai float saat ini yang akan dihitung dan dimanipulasi oleh skrip ini.

* Event Events:

Skrip ini memiliki dua event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

* Variabel Tambahan:

TargetFloat: Variabel yang digunakan untuk menyimpan nilai target yang akan dicapai oleh CurrentValue.

StartTransfer: Boolean yang menandakan apakah proses transfer CurrentValue sedang berlangsung.

* Metode Start():

Metode ini dipanggil saat objek pertama kali aktif. Pada saat ini, event StartEvents dipicu.

* Metode Update():

Metode ini dipanggil setiap frame saat objek ini aktif. Pada setiap pemanggilan, event UpdateEvents dipicu.

* Jika StartTransfer aktif, CurrentValue akan bertambah 1.0 setiap frame sampai mencapai TargetFloat. Begitu mencapai target, StartTransfer akan diubah menjadi false.
* Metode SaveData():

Metode ini digunakan untuk menyimpan nilai CurrentValue ke PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode LoadData():

Metode ini digunakan untuk memuat nilai CurrentValue dari PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode LoadData(string aName):

Metode ini mirip dengan LoadData(), tetapi Anda dapat memberikan nama yang akan digunakan untuk mencari kunci PlayerPrefs.

* Metode SetCurrentValue(float aValue):

Metode ini memungkinkan pengaturan manual nilai CurrentValue ke nilai yang diberikan (aValue).

Jika nilai aValue kurang dari atau sama dengan 0, CurrentValue diatur ke 0.

* Metode AddToCurrentValue(float aValue):

Metode ini digunakan untuk menambahkan nilai aValue ke CurrentValue.

Metode SubtractFromCurrentValue(float aValue):

Metode ini digunakan untuk mengurangkan nilai aValue dari CurrentValue.

Jika nilai CurrentValue menjadi kurang dari atau sama dengan 0 setelah pengurangan, maka CurrentValue diatur ke 0.

* Metode WriteTo():

Empat metode WriteTo berbeda yang memungkinkan Anda menampilkan nilai CurrentValue di UI atau mengatur properti dalam komponen lain:

WriteTo(Slider aValue): Mengatur nilai Slider UI sesuai dengan nilai CurrentValue.

WriteTo(InputField aValue): Mengatur teks dari komponen UI InputField dengan nilai CurrentValue sebagai teks.

WriteTo(Text aValue): Mengatur teks dari komponen UI Text dengan nilai CurrentValue sebagai teks.

WriteTo(TextMesh aValue): Mengatur teks dari komponen UI TextMesh dengan nilai CurrentValue sebagai teks.

WriteTo(AudioSource aValue): Mengatur volume AudioSource dengan nilai CurrentValue.

### 1.4 Calculation Score

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationScore : MonoBehaviour  {  [Header("Current Value")]  public float CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent AfterTransferEvents;  float TotalScore;  bool StartTransfer = false;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (StartTransfer)  {  CurrentValue += 1f;  if (CurrentValue > TotalScore)  {  CurrentValue = TotalScore;  StartTransfer = false;  AfterTransferEvents?.Invoke();  }  }  }  public void SaveData()  {  PlayerPrefs.SetFloat(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetFloat(transform.name);  }  }  public void SetCurrentValue(float aValue)  {  CurrentValue = aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void AddToCurrentValue(float aValue)  {  CurrentValue += aValue;  }  public void SubtractFromCurrentValue(float aValue)  {  CurrentValue -= aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void WriteTo(Slider aValue)  {  aValue.value = CurrentValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void TransferToCurrentValue(float aValue)  {  TotalScore = CurrentValue + aValue;  StartTransfer = true;  Invoke("ThreeSecond", 3);  }  public void ReadFrom(Slider aValue)  {  CurrentValue = aValue.value;  }  public void ReadFrom(InputField aValue)  {  CurrentValue = float.Parse(aValue.text);  }  public void ReadFrom(Text aValue)  {  CurrentValue = float.Parse(aValue.text);  }  public void ReadFrom(TextMesh aValue)  {  CurrentValue = float.Parse(aValue.text);  }  void ThreeSecond()  {  CurrentValue = TotalScore;  }  } |
| --- |

Script ini adalah varian dari skrip perhitungan yang telah dijelaskan sebelumnya, tetapi kali ini dirancang untuk mengelola nilai skor atau poin dalam permainan. Berikut adalah penjelasan tentang bagian utama dari skrip ini:

* Variabel CurrentValue:

Variabel ini digunakan untuk menyimpan nilai skor atau poin saat ini yang akan dihitung dan dimanipulasi oleh skrip ini.

* Event Events:

Skrip ini memiliki tiga event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

AfterTransferEvents: Event yang dipicu setelah proses transfer skor selesai.

* Variabel Tambahan:

TotalScore: Variabel yang digunakan untuk menyimpan nilai total yang akan dicapai oleh CurrentValue.

StartTransfer: Boolean yang menandakan apakah proses transfer CurrentValue sedang berlangsung.

* Metode Start():

Metode ini dipanggil saat objek pertama kali aktif. Pada saat ini, event StartEvents dipicu.

* Metode Update():

Metode ini dipanggil setiap frame saat objek ini aktif. Pada setiap pemanggilan, event UpdateEvents dipicu.

Jika StartTransfer aktif, CurrentValue akan bertambah 1.0 setiap frame sampai mencapai TotalScore. Begitu mencapai target, StartTransfer akan diubah menjadi false, dan event AfterTransferEvents akan dipicu.

* Metode SaveData():

Metode ini digunakan untuk menyimpan nilai CurrentValue ke PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode LoadData():

Metode ini digunakan untuk memuat nilai CurrentValue dari PlayerPrefs dengan nama yang sesuai (nama objek GameObject ini).

* Metode SetCurrentValue(float aValue):

Metode ini memungkinkan pengaturan manual nilai CurrentValue ke nilai yang diberikan (aValue).

Jika nilai aValue kurang dari atau sama dengan 0, CurrentValue diatur ke 0.

* Metode AddToCurrentValue(float aValue):

Metode ini digunakan untuk menambahkan nilai aValue ke CurrentValue.

* Metode SubtractFromCurrentValue(float aValue):

Metode ini digunakan untuk mengurangkan nilai aValue dari CurrentValue.

Jika nilai CurrentValue menjadi kurang dari atau sama dengan 0 setelah pengurangan, maka CurrentValue diatur ke 0.

* Metode WriteTo():

Empat metode WriteTo berbeda yang memungkinkan Anda menampilkan nilai CurrentValue di UI atau mengatur properti dalam komponen lain:

WriteTo(Slider aValue): Mengatur nilai Slider UI sesuai dengan nilai CurrentValue.

WriteTo(InputField aValue): Mengatur teks dari komponen UI InputField dengan nilai CurrentValue sebagai teks.

WriteTo(Text aValue): Mengatur teks dari komponen UI Text dengan nilai CurrentValue sebagai teks.

WriteTo(TextMesh aValue): Mengatur teks dari komponen UI TextMesh dengan nilai CurrentValue sebagai teks.

Metode TransferToCurrentValue(float aValue):

Metode ini digunakan untuk memulai proses transfer nilai CurrentValue ke TotalScore dengan menambahkannya sebesar aValue.

* StartTransfer diatur ke true dan akan mencapai target dalam tiga detik, dan kemudian menghentikan proses transfer.

### 1.5 Calculation Time

| using System.Collections;  using System.Collections.Generic;  using System;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationTime : MonoBehaviour  {  public enum CTimeCalculation { None, Increment, Decrement }  public enum CTimeFormat { Normal, SS, MMSS, HHMMSS }  [Header("Current Value")]  public int CurrentValue;  public bool Activated;  [Header("Time Settings")]  public CTimeCalculation TimeCalculation;  public CTimeFormat TimeFormat;  public int MinValue = 0;  public int MaxValue = 60;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent EndTimeEvents;  bool isActive;  int TotalScore;  bool StartTransfer = false;  void ExecuteIncrement()  {  if (isActive)  {  CurrentValue = CurrentValue + 1;  if (CurrentValue >= MaxValue)  {  isActive = false;  EndTimeEvents.Invoke();  CancelInvoke();  }  }  }  void ExecuteDecrement()  {  if (isActive)  {  CurrentValue = CurrentValue - 1;  if (CurrentValue <= MinValue)  {  isActive = false;  EndTimeEvents.Invoke();  CancelInvoke();  }  }  }  public void StartTimer()  {  isActive = true;  if (MaxValue == 0)  {  MaxValue = CurrentValue;  }  if (TimeCalculation == CTimeCalculation.Increment)  {  InvokeRepeating("ExecuteIncrement", 1, 1);  }  if (TimeCalculation == CTimeCalculation.Decrement)  {  InvokeRepeating("ExecuteDecrement", 1, 1);  }  }  public void StopTimer()  {  isActive = false;  CancelInvoke();  }  // Start is called before the first frame update  void Start()  {  if (MaxValue == 0)  {  MaxValue = CurrentValue;  }  if (Activated)  {  StartTimer();  }  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (StartTransfer)  {  CurrentValue += 1;  if (CurrentValue > TotalScore)  {  CurrentValue = TotalScore;  StartTransfer = false;  }  }  }  string SetFormatTime()  {  TimeSpan time = TimeSpan.FromSeconds(CurrentValue);  string formattedTime = CurrentValue.ToString();  if (TimeFormat == CTimeFormat.SS)  {  // Format waktu menjadi jam:menit:detik  formattedTime = string.Format("{0:D2}", time.Seconds);  }  else if (TimeFormat == CTimeFormat.MMSS)  {  // Format waktu menjadi jam:menit:detik  formattedTime = string.Format("{0:D2}:{1:D2}",  time.Minutes,  time.Seconds);  }  else if (TimeFormat == CTimeFormat.HHMMSS)  {  // Format waktu menjadi jam:menit:detik  formattedTime = string.Format("{0:D2}:{1:D2}:{2:D2}",  time.Hours,  time.Minutes,  time.Seconds);  }  return formattedTime;  }  public void SaveData()  {  PlayerPrefs.SetInt(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetInt(transform.name);  }  }  public void SetCurrentValue(int aValue)  {  CurrentValue = aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void AddToCurrentValue(int aValue)  {  CurrentValue += aValue;  }  public void SubtractFromCurrentValue(int aValue)  {  CurrentValue -= aValue;  if (CurrentValue <= 0) CurrentValue = 0;  }  public void WriteTo(Slider aValue)  {  aValue.value = CurrentValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.ToString();  aValue.text = SetFormatTime();  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  aValue.text = SetFormatTime();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  aValue.text = SetFormatTime();  }  public void TransferToCurrentValue(int aValue)  {  TotalScore = CurrentValue + aValue;  StartTransfer = true;  Invoke("ThreeSecond", 3);  }  public void ReadFrom(Slider aValue)  {  CurrentValue = Mathf.RoundToInt(aValue.value);  }  public void ReadFrom(InputField aValue)  {  CurrentValue = int.Parse(aValue.text);  }  void ThreeSecond()  {  CurrentValue = TotalScore;  }  } |
| --- |

Script ini adalah komponen Unity yang mengelola perhitungan waktu. Skrip ini memiliki banyak fitur, dan berikut adalah penjelasan bagian utamanya:

* Enum CTimeCalculation:

Ini adalah enumerasi yang berisi dua opsi: "Increment" dan "Decrement". Opsi ini digunakan untuk mengatur apakah perhitungan waktu akan bertambah atau berkurang.

* Enum CTimeFormat:

Ini adalah enumerasi yang berisi beberapa format waktu yang berbeda, termasuk "Normal", "SS" (detik), "MMSS" (menit:detik), dan "HHMMSS" (jam:menit:detik).

* Variabel CurrentValue:

Variabel ini digunakan untuk menyimpan nilai waktu saat ini yang akan dihitung dan dimanipulasi oleh skrip ini.

* Variabel Activated:

Ini adalah boolean yang menandakan apakah perhitungan waktu harus dimulai segera setelah objek aktif.

* Variabel TimeCalculation:

Variabel ini memungkinkan Anda memilih apakah waktu akan dihitung dengan cara "Increment" atau "Decrement" berdasarkan enumerasi CTimeCalculation.

* Variabel TimeFormat:

Variabel ini memungkinkan Anda memilih format tampilan waktu, seperti "SS", "MM:SS", atau "HH:MM:SS", berdasarkan enumerasi CTimeFormat.

* Variabel MinValue dan MaxValue:

Variabel ini digunakan untuk menentukan nilai waktu minimum dan maksimum yang dapat dicapai oleh CurrentValue.

* Event Settings:

Skrip ini memiliki tiga event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

EndTimeEvents: Event yang dipicu saat CurrentValue mencapai MinValue atau MaxValue, tergantung pada arah perhitungan waktu.

* Variabel Tambahan:

isActive: Boolean yang menunjukkan apakah perhitungan waktu aktif saat ini.

TotalScore: Variabel yang digunakan untuk menyimpan nilai total yang akan dicapai oleh CurrentValue.

StartTransfer: Boolean yang menandakan apakah proses transfer CurrentValue sedang berlangsung.

Metode ExecuteIncrement() dan ExecuteDecrement():

Metode-metode ini memungkinkan manipulasi nilai CurrentValue secara manual atau dengan menambah atau mengurangkan nilai.

Metode WriteTo():

Empat metode WriteTo berbeda yang memungkinkan Anda menampilkan nilai CurrentValue di UI atau mengatur properti dalam komponen lain:

* WriteTo(Slider aValue): Mengatur nilai Slider UI sesuai dengan nilai CurrentValue.
* WriteTo(InputField aValue): Mengatur teks dari komponen UI InputField dengan nilai CurrentValue sebagai teks, dengan format waktu sesuai.
* WriteTo(Text aValue): Mengatur teks dari komponen UI Text dengan nilai CurrentValue sebagai teks, dengan format waktu sesuai.
* WriteTo(TextMesh aValue): Mengatur teks dari komponen UI TextMesh dengan nilai CurrentValue sebagai teks, dengan format waktu sesuai.

### 1.6 Calculation String

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class CalculationString : MonoBehaviour  {  [Header("Current Value")]  public string CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetString(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetString(transform.name);  }  }  public void SetCurrentValue(string aValue)  {  CurrentValue = aValue;  }  public void SetCurrentValue(Text aValue)  {  CurrentValue = aValue.text;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void ReadFrom(InputField aValue)  {  CurrentValue = aValue.text;  }  public void ReadFrom(Text aValue)  {  CurrentValue = aValue.text;  }  public void ReadFrom(TextMesh aValue)  {  CurrentValue = aValue.text;  }  } |
| --- |

Script ini adalah komponen Unity yang mengelola nilai string. Ini memungkinkan manipulasi, penyimpanan, dan pemulihan string yang disebut CurrentValue. Berikut adalah penjelasan komponen utama dari skrip ini:

* Variabel CurrentValue:

Ini adalah string yang digunakan untuk menyimpan dan mengelola nilai string saat ini yang dapat diakses dan dimanipulasi melalui skrip ini.

* Event Settings:

Skrip ini memiliki dua event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

* Metode SaveData() dan LoadData():

Metode ini digunakan untuk menyimpan dan memuat nilai string dari PlayerPrefs berdasarkan nama GameObject ini. Ini memungkinkan Anda untuk menyimpan dan memulihkan nilai CurrentValue saat permainan dimulai atau berakhir.

* Metode SetCurrentValue(string aValue):

Metode ini memungkinkan Anda mengatur nilai CurrentValue dengan mengirimkan string sebagai parameter.

Metode SetCurrentValue(Text aValue):

Metode ini memungkinkan Anda mengatur nilai CurrentValue dengan mengambil teks dari komponen UI Text dan menggunakannya sebagai nilai CurrentValue.

Metode WriteTo():

Tiga metode WriteTo yang berbeda memungkinkan Anda menampilkan nilai CurrentValue di UI atau mengatur properti dalam komponen lain:

* WriteTo(InputField aValue): Mengatur teks dari komponen UI InputField dengan nilai CurrentValue.
* WriteTo(Text aValue): Mengatur teks dari komponen UI Text dengan nilai CurrentValue.
* WriteTo(TextMesh aValue): Mengatur teks dari komponen UI TextMesh dengan nilai CurrentValue.

### 1.7 Calculation Transform

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class CalculationTransform : MonoBehaviour  {  [Header("Current Value")]  public Vector3 Position;  public Vector3 Rotation;  public Vector3 Scale;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  Position = transform.position;  Rotation = transform.eulerAngles;  Scale = transform.localScale;  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetFloat(transform.gameObject.name + "px", transform.position.x);  PlayerPrefs.SetFloat(transform.gameObject.name + "py", transform.position.y);  PlayerPrefs.SetFloat(transform.gameObject.name + "pz", transform.position.z);  PlayerPrefs.SetFloat(transform.gameObject.name + "rx", transform.eulerAngles.x);  PlayerPrefs.SetFloat(transform.gameObject.name + "ry", transform.eulerAngles.y);  PlayerPrefs.SetFloat(transform.gameObject.name + "rz", transform.eulerAngles.z);  PlayerPrefs.SetFloat(transform.gameObject.name + "sx", transform.localScale.x);  PlayerPrefs.SetFloat(transform.gameObject.name + "sy", transform.localScale.y);  PlayerPrefs.SetFloat(transform.gameObject.name + "sz", transform.localScale.z);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name + "px"))  {  Vector3 px = new Vector3(  PlayerPrefs.GetFloat(transform.gameObject.name + "px"),  PlayerPrefs.GetFloat(transform.gameObject.name + "py"),  PlayerPrefs.GetFloat(transform.gameObject.name + "pz"));  Vector3 rx = new Vector3(  PlayerPrefs.GetFloat(transform.gameObject.name + "rx"),  PlayerPrefs.GetFloat(transform.gameObject.name + "ry"),  PlayerPrefs.GetFloat(transform.gameObject.name + "rz"));  Vector3 sx = new Vector3(  PlayerPrefs.GetFloat(transform.gameObject.name + "sx"),  PlayerPrefs.GetFloat(transform.gameObject.name + "sy"),  PlayerPrefs.GetFloat(transform.gameObject.name + "sz"));  transform.position = px;  transform.rotation = Quaternion.Euler(rx);  transform.localScale = sx;  }  }  } |
| --- |

Script ini adalah komponen Unity yang digunakan untuk mengelola transformasi (posisi, rotasi, dan skala) suatu GameObject. Ini memungkinkan Anda untuk menyimpan dan memuat transformasi GameObject ini menggunakan PlayerPrefs dan mengatur serta mengambil transformasi dari objek tersebut. Berikut adalah penjelasan komponen utama dari skrip ini:

* Variabel Transformasi:

Skrip ini memiliki tiga set variabel yang mewakili transformasi GameObject:

Position: Vector3 yang menyimpan posisi objek.

Rotation: Vector3 yang menyimpan rotasi objek.

Scale: Vector3 yang menyimpan skala objek.

* Event Settings:

Skrip ini memiliki dua event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

* Metode SaveData() dan LoadData():

Metode SaveData() digunakan untuk menyimpan transformasi objek saat ini ke PlayerPrefs. Ini menyimpan posisi, rotasi, dan skala objek.

Metode LoadData() digunakan untuk memuat transformasi objek dari PlayerPrefs dan mengatur transformasi objek ini berdasarkan data yang tersimpan.

* Update Method:

Dalam metode Update(), nilai Position, Rotation, dan Scale akan diperbarui dengan nilai aktual transformasi objek pada setiap frame. Ini memastikan bahwa nilai transformasi selalu mencerminkan keadaan aktual objek.

* Penyimpanan dan Pemuatan Transformasi:

Transformasi objek (posisi, rotasi, dan skala) disimpan dalam PlayerPrefs dengan menggunakan kunci unik berdasarkan nama GameObject. Ketika objek diubah, transformasi akan disimpan dalam PlayerPrefs, dan saat objek diinisialisasi, transformasi akan dimuat kembali dari PlayerPrefs dan diterapkan pada objek.

Dengan komponen ini, Anda dapat dengan mudah mengelola dan menyimpan transformasi objek untuk kemudian dimuat kembali saat permainan dimulai kembali atau diinisialisasi kembali.

### 

### 1.8 Calculation Cookies

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using UnityEngine.Networking;  public class CalculationCookies : MonoBehaviour  {  [Header("Target Cookie")]  public string Cookie;  [Header("Current Value")]  public string CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public void StartRequest()  {  StartCoroutine(GetDataFromCookie());  }  IEnumerator GetDataFromCookie()  {  // Panggil fungsi JavaScript untuk membaca nilai cookie  string javaScriptCode = "document.cookie.match(/"+Cookie+"=([^;]+)/)[1];";  UnityWebRequest www = UnityWebRequest.Get("javascript:" + javaScriptCode);  yield return www.SendWebRequest();  if (www.result == UnityWebRequest.Result.ConnectionError)  {  Debug.LogError("Error: " + www.error);  }  else  {  CurrentValue = www.downloadHandler.text;  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetString(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetString(transform.name);  }  }  public void SetCurrentValue(string aValue)  {  CurrentValue = aValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void ReadFrom(InputField aValue)  {  CurrentValue = aValue.text;  }  } |
| --- |

Script ini digunakan untuk mengambil nilai dari cookie pada halaman web menggunakan Unity. Script ini memungkinkan Anda untuk melakukan permintaan HTTP menggunakan UnityWebRequest untuk mengambil data cookie tertentu dari halaman web. Berikut adalah penjelasan komponen utama dari skrip ini:

* Variabel Cookie:

Cookie: Nama cookie yang akan diambil nilainya dari halaman web.

* Variabel CurrentValue:

CurrentValue: String yang menyimpan nilai dari cookie yang diambil dari halaman web.

* Event Settings:

Skrip ini memiliki dua event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

* Metode StartRequest():

Metode ini digunakan untuk memulai permintaan HTTP untuk mengambil nilai dari cookie yang ditentukan dari halaman web.

* Metode GetDataFromCookie():

Metode IEnumerator yang digunakan untuk melakukan permintaan HTTP menggunakan UnityWebRequest. Ini akan memanggil sebuah fungsi JavaScript pada halaman web yang akan membaca dan mengembalikan nilai dari cookie yang ditentukan.

UnityWebRequest akan mengirim permintaan ke fungsi JavaScript dengan menggunakan "javascript:" + javaScriptCode.Hasil dari permintaan akan disimpan dalam variabel CurrentValue.

* Update Method:

Dalam metode Update(), event UpdateEvents akan dipicu pada setiap frame, memberikan fleksibilitas dalam menjalankan fungsi tambahan jika diperlukan.

* Penyimpanan dan Pemuatan Data:

Transformasi objek (dalam hal ini, nilai dari cookie) dapat disimpan dalam PlayerPrefs menggunakan metode SaveData().

Data yang disimpan dapat dimuat kembali menggunakan metode LoadData().

* Metode SetCurrentValue():

Metode ini memungkinkan Anda untuk mengatur nilai CurrentValue secara langsung.

* Metode WriteTo():

Metode ini digunakan untuk menampilkan nilai CurrentValue pada komponen UI seperti InputField, Text, atau TextMesh.

* Metode ReadFrom():

Metode ini digunakan untuk membaca nilai dari komponen UI seperti InputField dan memasukkannya ke dalam CurrentValue.

Dengan script ini, Anda dapat mengambil dan memanfaatkan nilai cookie dari halaman web dalam lingkungan Unity. Dengan cara ini, Anda dapat menghubungkan permainan Unity Anda dengan informasi yang disimpan dalam cookie pada situs web eksternal.

### 1.9 Calculation Request

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using UnityEngine.Networking;  public class CalculationRequestGet : MonoBehaviour  {  [Header("Target API")]  public string API;  [Header("Current Value")]  public string CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent RequestEvents;  public void StartRequest()  {  StartCoroutine(GetRequest(API));  }  IEnumerator GetRequest(string uri)  {  using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))  {  yield return webRequest.SendWebRequest();  if (webRequest.result == UnityWebRequest.Result.ConnectionError)  {  Debug.LogError("Error: " + webRequest.error);  }  else  {  CurrentValue = webRequest.downloadHandler.text;  Debug.Log("Result: " + CurrentValue);  RequestEvents?.Invoke();  SaveData();  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetString(transform.gameObject.name, CurrentValue);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue = PlayerPrefs.GetString(transform.name);  }  }  public void SetCurrentValue(string aValue)  {  CurrentValue = aValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void ReadFrom(InputField aValue)  {  CurrentValue = aValue.text;  }  } |
| --- |

Script ini digunakan untuk melakukan permintaan HTTP GET ke suatu API dan mengambil data yang diberikan oleh API. Berikut adalah penjelasan komponen utama dari skrip ini:

* Variabel API:

API: URL atau alamat API yang akan diakses untuk mengambil data.

* Variabel CurrentValue:

CurrentValue: String yang menyimpan data yang diambil dari API.

* Event Settings:

Skrip ini memiliki tiga event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

RequestEvents: Event yang dipicu setelah permintaan ke API selesai dan data berhasil diambil.

* Metode StartRequest():

Metode ini digunakan untuk memulai permintaan HTTP GET ke API yang ditentukan.

Metode GetRequest(string uri):

Metode IEnumerator yang melakukan permintaan HTTP GET menggunakan UnityWebRequest ke URI yang ditentukan.

Hasil dari permintaan akan disimpan dalam variabel CurrentValue.

Jika permintaan berhasil, event RequestEvents dipicu dan data disimpan dengan memanggil metode SaveData().

* Update Method:

Dalam metode Update(), event UpdateEvents akan dipicu pada setiap frame, memberikan fleksibilitas dalam menjalankan fungsi tambahan jika diperlukan.

* Penyimpanan dan Pemuatan Data:

Data yang diambil dari API dapat disimpan dalam PlayerPrefs menggunakan metode SaveData().

Data yang disimpan dapat dimuat kembali menggunakan metode LoadData().

* Metode SetCurrentValue():

Metode ini memungkinkan Anda untuk mengatur nilai CurrentValue secara langsung.

* Metode WriteTo():

Metode ini digunakan untuk menampilkan nilai CurrentValue pada komponen UI seperti InputField, Text, atau TextMesh.

* Metode ReadFrom():

Metode ini digunakan untuk membaca nilai dari komponen UI seperti InputField dan memasukkannya ke dalam CurrentValue.

Dengan script ini, Anda dapat melakukan permintaan HTTP GET ke API dan mengambil data yang diberikan oleh API tersebut. Data ini kemudian dapat digunakan dalam permainan Unity Anda atau ditampilkan dalam antarmuka pengguna.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using UnityEngine.Networking;  public class CalculationRequestPost : MonoBehaviour  {  [Header("Target API")]  public string API;  public Text Data;  [Header("Current Value")]  public Text CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent RequestEvents;  public void StartRequest()  {  StartCoroutine(PostRequest());  }  IEnumerator PostRequest()  {  WWWForm FormRequest = new WWWForm();  FormRequest.AddField("data", Data.text);  using (UnityWebRequest webRequest = UnityWebRequest.Post(API, FormRequest))  {  yield return webRequest.SendWebRequest();  if (webRequest.result == UnityWebRequest.Result.ConnectionError)  {  Debug.LogError("Error: " + webRequest.error);  }  else  {  CurrentValue.text = webRequest.downloadHandler.text;  Debug.Log("Result: " + CurrentValue.text);  RequestEvents?.Invoke();  SaveData();  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetString(transform.gameObject.name, CurrentValue.text);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue.text = PlayerPrefs.GetString(transform.name);  }  }  public void SetCurrentValue(string aValue)  {  CurrentValue.text = aValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.text;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void ReadFrom(InputField aValue)  {  CurrentValue.text = aValue.text;  }  } |
| --- |

Script ini digunakan untuk melakukan permintaan HTTP POST ke suatu API dengan mengirimkan data ke server. Berikut adalah penjelasan komponen utama dari skrip ini:

* Variabel Target API:

API: URL atau alamat API yang akan digunakan untuk melakukan permintaan POST.

Data: Komponen teks (Text) yang berisi data yang akan dikirimkan ke API.

* Variabel CurrentValue:

CurrentValue: Komponen teks (Text) yang akan menampilkan data yang diterima dari API setelah permintaan POST berhasil.

* Event Settings:

Skrip ini memiliki tiga event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

RequestEvents: Event yang dipicu setelah permintaan ke API selesai dan data berhasil diterima dan ditampilkan.

* Metode StartRequest():

Metode ini digunakan untuk memulai permintaan HTTP POST ke API yang ditentukan.

Metode PostRequest():

Metode IEnumerator yang melakukan permintaan HTTP POST menggunakan UnityWebRequest ke URI/API yang ditentukan.

Data yang akan dikirimkan ke server diambil dari komponen teks Data.

Hasil dari permintaan akan ditampilkan di komponen teks CurrentValue.

Jika permintaan berhasil, event RequestEvents dipicu dan data ditampilkan serta disimpan dengan memanggil metode SaveData().

* Update Method:

Dalam metode Update(), event UpdateEvents akan dipicu pada setiap frame, memberikan fleksibilitas dalam menjalankan fungsi tambahan jika diperlukan.

* Penyimpanan dan Pemuatan Data:

Data yang diterima dari API dapat disimpan dalam PlayerPrefs menggunakan metode SaveData().

Data yang disimpan dapat dimuat kembali menggunakan metode LoadData().

* Metode SetCurrentValue():

Metode ini memungkinkan Anda untuk mengatur nilai CurrentValue secara langsung.

* Metode WriteTo():

Metode ini digunakan untuk menampilkan nilai CurrentValue pada komponen UI seperti InputField, Text, atau TextMesh.

* Metode ReadFrom():

Metode ini digunakan untuk membaca nilai dari komponen UI seperti InputField dan memasukkannya ke dalam CurrentValue.

Dengan skrip ini, Anda dapat mengirim data ke server melalui permintaan HTTP POST, menerima data yang diberikan oleh server, dan menampilkannya dalam permainan Unity Anda atau dalam antarmuka pengguna.

### 1.10 Calculation Token

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using UnityEngine.Networking;  public class CalculationToken : MonoBehaviour  {  [Header("Target API")]  public string API;  public Text Token;  [Header("Current Value")]  public Text CurrentValue;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent RequestEvents;  public void StartRequest()  {  StartCoroutine(PostRequest());  }  IEnumerator PostRequest()  {  WWWForm FormRequest = new WWWForm();  FormRequest.AddField("token", Token.text);  using (UnityWebRequest webRequest = UnityWebRequest.Post(API, FormRequest))  {  yield return webRequest.SendWebRequest();  if (webRequest.result == UnityWebRequest.Result.ConnectionError)  {  Debug.LogError("Error: " + webRequest.error);  }  else  {  CurrentValue.text = webRequest.downloadHandler.text;  Debug.Log("Result: " + CurrentValue.text);  RequestEvents?.Invoke();  SaveData();  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SaveData()  {  PlayerPrefs.SetString(transform.gameObject.name, CurrentValue.text);  }  public void LoadData()  {  if (PlayerPrefs.HasKey(transform.name))  {  CurrentValue.text = PlayerPrefs.GetString(transform.name);  }  }  public void SetCurrentValue(string aValue)  {  CurrentValue.text = aValue;  }  public void WriteTo(InputField aValue)  {  aValue.text = CurrentValue.text;  }  public void WriteTo(Text aValue)  {  aValue.text = CurrentValue.ToString();  }  public void WriteTo(TextMesh aValue)  {  aValue.text = CurrentValue.ToString();  }  public void ReadFrom(InputField aValue)  {  CurrentValue.text = aValue.text;  }  } |
| --- |

Script ini digunakan untuk melakukan permintaan HTTP POST ke suatu API dengan mengirimkan token ke server. Di bawah ini adalah penjelasan komponen utama dari skrip ini:

* Variabel Target API:

API: URL atau alamat API yang akan digunakan untuk melakukan permintaan POST.

Token: Komponen teks (Text) yang berisi token yang akan dikirimkan ke API.

* Variabel CurrentValue:

CurrentValue: Komponen teks (Text) yang akan menampilkan data yang diterima dari API setelah permintaan POST berhasil.

* Event Settings:

Skrip ini memiliki tiga event yang dapat dipicu dalam berbagai kondisi:

StartEvents: Event yang dipicu pada awal frame pertama saat objek ini aktif.

UpdateEvents: Event yang dipicu setiap frame saat objek ini aktif.

RequestEvents: Event yang dipicu setelah permintaan ke API selesai dan data berhasil diterima dan ditampilkan.

* Metode StartRequest():

Metode ini digunakan untuk memulai permintaan HTTP POST ke API yang ditentukan.

* Metode PostRequest():

Metode IEnumerator yang melakukan permintaan HTTP POST menggunakan UnityWebRequest ke URI/API yang ditentukan.

Token yang akan dikirimkan ke server diambil dari komponen teks Token.

Hasil dari permintaan akan ditampilkan di komponen teks CurrentValue.

Jika permintaan berhasil, event RequestEvents dipicu dan data ditampilkan serta disimpan dengan memanggil metode SaveData().

* Update Method:

Dalam metode Update(), event UpdateEvents akan dipicu pada setiap frame, memberikan fleksibilitas dalam menjalankan fungsi tambahan jika diperlukan.

* Penyimpanan dan Pemuatan Data:

Data yang diterima dari API dapat disimpan dalam PlayerPrefs menggunakan metode SaveData().

Data yang disimpan dapat dimuat kembali menggunakan metode LoadData().

* Metode SetCurrentValue():

Metode ini memungkinkan Anda untuk mengatur nilai CurrentValue secara langsung.

* Metode WriteTo():

Metode ini digunakan untuk menampilkan nilai CurrentValue pada komponen UI seperti InputField, Text, atau TextMesh.

* Metode ReadFrom():

Metode ini digunakan untuk membaca nilai dari komponen UI seperti InputField dan memasukkannya ke dalam CurrentValue.

## 2. Activation

Event Activation dalam game terkait dengan pengaktifan atau penonaktifan objek, karakter, atau mekanisme tertentu dalam game berdasarkan kondisi atau peristiwa tertentu. Dalam pemrograman game, ini biasanya melibatkan penggunaan kondisional (if-else statements) dan variabel boolean. Misalnya, jika pemain mendekati perangkap tertentu dalam game, perangkap tersebut harus diaktifkan. Pemrograman untuk event Activation ini mungkin melibatkan penggunaan sensor atau trigger yang mendeteksi kedekatan pemain, dan kemudian mengubah variabel boolean yang terkait dengan perangkap dari 'false' menjadi 'true' untuk mengaktifkannya. Selanjutnya, kode yang terkait dengan mekanisme perangkap akan dijalankan, seperti menimbulkan damage kepada pemain atau mengunci pintu tertentu. Pengertian mendalam tentang logika pemrograman dan penggunaan kondisional dan variabel adalah kunci untuk mengimplementasikan event Activation dengan efektif dalam game.

### 2.1 Activation Event

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationEvent : MonoBehaviour  {  [Header("Main Settings")]  public UnityEvent MainEvent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  public void InvokeMainEvents()  {  MainEvent?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini adalah komponen yang memungkinkan Anda untuk mengatur peristiwa (events) yang akan terpicu saat berbagai kondisi tertentu terjadi dalam permainan Unity. Berikut adalah penjelasan komponen utama dari skrip ini:

* MainEvent: Ini adalah peristiwa utama yang akan terpicu saat metode InvokeMainEvents() dipanggil atau saat kondisi tertentu dipenuhi.
* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif di dalam permainan. Biasanya terpicu di awal permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan. Ini memungkinkan Anda untuk menjalankan logika permainan yang perlu diperbarui setiap frame.
* Metode Start(): Metode ini adalah bagian dari siklus hidup MonoBehaviour yang diwarisi oleh objek dalam permainan Unity.
* Pada saat objek diaktifkan dalam permainan, metode StartEvents akan dipicu jika telah diatur. Ini biasanya digunakan untuk menginisialisasi objek pada awal permainan.
* Metode InvokeMainEvents(): Metode ini digunakan untuk memicu peristiwa utama (MainEvent). Anda dapat memanggil metode ini dari objek lain atau dari komponen lain dalam permainan untuk memulai eksekusi logika yang terkait dengan MainEvent.
* Metode Update(): Metode ini juga adalah bagian dari siklus hidup MonoBehaviour yang dipanggil pada setiap frame permainan. Pada setiap frame, metode UpdateEvents akan dipicu jika telah diatur. Ini berguna untuk menjalankan logika yang perlu diperbarui setiap frame, seperti pergerakan objek atau interaksi pengguna yang berjalan dalam permainan.

### 2.2 Activation Delay

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationDelay : MonoBehaviour  {  [Header("Main Settings")]  public UnityEvent MainEvent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Delay Settings")]  public bool usingDelay;  public float Delay;  public UnityEvent DelayEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (usingDelay)  {  Invoke("InvokeDelay", Delay);  }  }  public void InvokeMainEvents()  {  MainEvent?.Invoke();  }  public void InvokeMainEventDelay()  {  Invoke("InvokeMainEvents", Delay);  }  void InvokeDelay()  {  DelayEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini adalah komponen yang memungkinkan Anda mengatur peristiwa (events) yang akan terpicu dengan atau tanpa penundaan (delay) saat berbagai kondisi tertentu terjadi dalam permainan Unity. Di sini, ada beberapa komponen dan metode yang memungkinkan Anda mengatur peristiwa yang berbeda:

* MainEvent: Ini adalah peristiwa utama yang akan terpicu saat metode InvokeMainEvents() dipanggil atau saat kondisi tertentu terpenuhi.
* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan. Biasanya terpicu di awal permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan. Ini memungkinkan Anda menjalankan logika permainan yang perlu diperbarui setiap frame.
* usingDelay: Jika diaktifkan, ini memungkinkan Anda mengatur penundaan sebelum peristiwa tertentu dipicu.
* Delay: Menentukan durasi penundaan dalam detik sebelum peristiwa dipicu.
* DelayEvents: Ini adalah peristiwa yang akan terpicu setelah penundaan (Delay) selesai. Biasanya digunakan untuk mengeksekusi tindakan tertentu setelah penundaan tertentu.
* Metode Start(): Metode ini adalah bagian dari siklus hidup MonoBehaviour yang diwarisi oleh objek dalam permainan Unity.
* Pada saat objek diaktifkan dalam permainan, metode StartEvents akan dipicu jika telah diatur. Ini biasanya digunakan untuk menginisialisasi objek pada awal permainan.
* Jika usingDelay diaktifkan, metode InvokeDelay() akan dipanggil dengan penundaan yang telah ditentukan menggunakan Invoke().
* Metode InvokeMainEvents(): Metode ini digunakan untuk memicu peristiwa utama (MainEvent). Anda dapat memanggil metode ini dari objek lain atau dari komponen lain dalam permainan untuk memulai eksekusi logika yang terkait dengan MainEvent.
* Metode InvokeMainEventDelay(): Metode ini memicu peristiwa utama (MainEvent) dengan penundaan. Ini berguna jika Anda ingin memicu MainEvent setelah sejumlah waktu tertentu.
* Metode InvokeDelay(): Metode ini dipanggil saat penundaan selesai (jika usingDelay diaktifkan) dan akan memicu peristiwa DelayEvents.
* Metode Update(): Metode ini adalah bagian dari siklus hidup MonoBehaviour yang dipanggil pada setiap frame permainan.
* Pada setiap frame, metode UpdateEvents akan dipicu jika telah diatur. Ini berguna untuk menjalankan logika yang perlu diperbarui setiap frame, seperti pergerakan objek atau interaksi pengguna yang berjalan dalam permainan.

### 2.3 Activation LoadScene

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.SceneManagement;  public class ActivationLoadScene : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Delay Settings")]  public bool usingDelay;  public float Delay;  public UnityEvent DelayEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (usingDelay)  {  Invoke("LoadScene", Delay);  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  void LoadScene()  {  DelayEvents?.Invoke();  }  public void LoadScene(string aValue)  {  //Melakukan perpindahan antar scene. Catatan: Scene yang dipanggil sudah didaftarkan di Build Setting  SceneManager.LoadScene(aValue);  }  public void RestartScene()  {  Scene activeScene = SceneManager.GetActiveScene();  SceneManager.LoadScene(activeScene.buildIndex);  }  } |
| --- |

Script ini adalah komponen yang memungkinkan Anda untuk memuat atau memulai ulang scene dalam permainan Unity. Berikut adalah penjelasan komponen ini:

* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan. Biasanya terpicu di awal permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan. Ini memungkinkan Anda menjalankan logika permainan yang perlu diperbarui setiap frame.
* Delay Settings: usingDelay: Jika diaktifkan, ini memungkinkan Anda mengatur penundaan sebelum peristiwa tertentu dipicu.
* Delay: Menentukan durasi penundaan dalam detik sebelum peristiwa dipicu.
* DelayEvents: Ini adalah peristiwa yang akan terpicu setelah penundaan (Delay) selesai. Biasanya digunakan untuk mengeksekusi tindakan tertentu setelah penundaan tertentu.
* Metode Start(): Metode ini adalah bagian dari siklus hidup MonoBehaviour yang diwarisi oleh objek dalam permainan Unity.
* Pada saat objek diaktifkan dalam permainan, metode StartEvents akan dipicu jika telah diatur. Ini biasanya digunakan untuk menginisialisasi objek pada awal permainan.
* Jika usingDelay diaktifkan, metode LoadScene() akan dipanggil dengan penundaan yang telah ditentukan menggunakan Invoke().
* Metode Update(): Metode ini adalah bagian dari siklus hidup MonoBehaviour yang dipanggil pada setiap frame permainan.
* Pada setiap frame, metode UpdateEvents akan dipicu jika telah diatur. Ini berguna untuk menjalankan logika yang perlu diperbarui setiap frame, seperti pergerakan objek atau interaksi pengguna yang berjalan dalam permainan.
* Metode LoadScene(string aValue): Metode ini digunakan untuk memuat scene baru berdasarkan nama scene yang diberikan sebagai argumen (aValue).
* Ini adalah cara yang umum digunakan untuk memindahkan pemain antar scene dalam permainan. Scene yang dituju harus telah didaftarkan di Build Settings dalam proyek Unity.
* Metode RestartScene(): Metode ini memungkinkan Anda untuk memulai ulang scene saat ini yang sedang aktif. Ini dilakukan dengan mendapatkan indeks build scene aktif menggunakan SceneManager.GetActiveScene().buildIndex dan kemudian memuat scene dengan indeks yang sama.

### 2.4 Activation LoadSceneAsync

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.SceneManagement;  public class ActivationLoadSceneAsync : MonoBehaviour  {  [Header("Loading Screen")]  public LoadSceneMode LoadingType;  public string LoadingScene;  string TargetScene;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Delay Settings")]  public bool usingDelay;  public float Delay;  public UnityEvent DelayEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (usingDelay)  {  Invoke("LoadScene", Delay);  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  void LoadScene()  {  DelayEvents?.Invoke();  }  public void LoadScene(string aValue)  {  //Melakukan perpindahan antar scene. Catatan: Scene yang dipanggil sudah didaftarkan di Build Setting  TargetScene = aValue;  StartCoroutine(LoadAsyncScene());  }  public void LoadScene(CalculationString aValue)  {  //Melakukan perpindahan antar scene. Catatan: Scene yang dipanggil sudah didaftarkan di Build Setting  TargetScene = aValue.CurrentValue;  StartCoroutine(LoadAsyncScene());  }  IEnumerator LoadAsyncScene()  {  SceneManager.LoadSceneAsync(LoadingScene, LoadSceneMode.Additive);  yield return new WaitForSeconds(1);  AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(TargetScene, LoadingType);  while (!asyncLoad.isDone)  {  PlayerPrefs.SetFloat("Loading", asyncLoad.progress);  yield return null;  }  SceneManager.UnloadSceneAsync(LoadingScene);  }  public void RestartScene()  {  StartCoroutine(RestartAsyncScene());  }  IEnumerator RestartAsyncScene()  {  SceneManager.LoadSceneAsync(LoadingScene, LoadSceneMode.Additive);  yield return new WaitForSeconds(1);  Scene activeScene = SceneManager.GetActiveScene();  TargetScene = activeScene.name;  AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(TargetScene, LoadingType);  while (!asyncLoad.isDone)  {  PlayerPrefs.SetFloat("Loading", asyncLoad.progress);  yield return null;  }  SceneManager.UnloadSceneAsync(LoadingScene);  }  } |
| --- |

Script ini adalah komponen yang memungkinkan Anda untuk memuat atau memulai ulang scene secara asinkron dalam permainan Unity. Ini digunakan untuk mengimplementasikan layar loading dan memuat scene secara paralel. Berikut adalah penjelasan komponen ini:

* Loading Screen: LoadingType: Menentukan cara scene baru akan dimuat. Mode ini dapat berupa "Single" atau "Additive". "Single" menggantikan scene yang aktif saat ini dengan scene yang baru dimuat, sedangkan "Additive" memuat scene baru tanpa menggantikan scene yang ada.
* LoadingScene: Nama scene yang akan digunakan sebagai layar loading. Scene ini akan dimuat terlebih dahulu sebelum scene target dimuat.
* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* usingDelay: Jika diaktifkan, ini memungkinkan Anda mengatur penundaan sebelum peristiwa tertentu dipicu.
* Delay: Menentukan durasi penundaan dalam detik sebelum peristiwa dipicu.
* DelayEvents: Ini adalah peristiwa yang akan terpicu setelah penundaan (Delay) selesai.
* Metode LoadScene(string aValue): Metode ini digunakan untuk memuat scene baru berdasarkan nama scene yang diberikan sebagai argumen (aValue).
* Ini adalah cara yang umum digunakan untuk memindahkan pemain antar scene dalam permainan. Scene yang dituju harus telah didaftarkan di Build Settings dalam proyek Unity.
* Sebelum memuat scene target, metode ini terlebih dahulu memuat scene "LoadingScene" yang digunakan sebagai layar loading.
* Metode LoadScene(CalculationString aValue): Metode ini memungkinkan Anda memuat scene baru berdasarkan nilai yang ada dalam komponen CalculationString yang diberikan sebagai argumen (aValue).
* Ini berguna jika Anda ingin memuat scene berdasarkan nilai string yang berasal dari komponen lain.
* Metode LoadAsyncScene(): Metode ini digunakan untuk memuat scene secara asinkron. Ini memulai dengan memuat scene "LoadingScene" yang digunakan sebagai layar loading.
* Kemudian, ini memulai operasi asinkron untuk memuat scene target dengan mode yang telah ditentukan.
* Selama proses memuat, perintah PlayerPrefs.SetFloat("Loading", asyncLoad.progress); digunakan untuk menyimpan kemajuan memuat sebagai float dalam PlayerPrefs. Ini dapat digunakan untuk menghubungkan kemajuan loading dengan elemen GUI lainnya.
* Metode RestartScene(): Metode ini digunakan untuk memulai ulang scene yang saat ini aktif.
* Ini melakukan operasi yang serupa dengan LoadScene(string aValue), tetapi scene yang akan dimuat adalah scene aktif saat ini.
* Metode RestartAsyncScene(): Metode ini memungkinkan Anda memulai ulang scene secara asinkron. Ini juga memulai dengan memuat scene "LoadingScene" yang digunakan sebagai layar loading. Kemudian, ini memulai operasi asinkron untuk memuat scene target, yang dalam hal ini adalah scene aktif saat ini. Selama proses memuat, perintah PlayerPrefs.SetFloat("Loading", asyncLoad.progress); digunakan untuk menyimpan kemajuan memuat sebagai float dalam PlayerPrefs.

### 2.5 Activation LoadSceneManager

| using System;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.SceneManagement;  using UnityEngine.AddressableAssets;  using UnityEngine.ResourceManagement.AsyncOperations;  using UnityEngine.ResourceManagement.ResourceProviders;  public class ActivationLoadSceneManager : MonoBehaviour  {  [Header("Loading Screen")]  public LoadSceneMode LoadingType;  public AssetReference LoadingScene;  [Header("Main Settings")]  public AssetReference TargetScene;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Delay Settings")]  public bool usingDelay;  public float Delay;  public UnityEvent DelayEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (usingDelay)  {  Invoke("LoadScene", Delay);  }  }  public void LoadScene()  {  //Melakukan perpindahan antar scene. Catatan: Scene yang dipanggil sudah didaftarkan di Build Setting  StartCoroutine(LoadAsyncScene());  }  IEnumerator LoadAsyncScene()  {  if (LoadingScene != null)  {  Addressables.LoadSceneAsync(LoadingScene, LoadSceneMode.Additive);  yield return new WaitForSeconds(1);  }  Addressables.LoadSceneAsync(TargetScene, UnityEngine.SceneManagement.LoadSceneMode.Single);  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini adalah komponen yang memungkinkan Anda untuk memuat atau memulai ulang scene menggunakan Addressable Assets System dalam permainan Unity. Ini digunakan untuk mengimplementasikan layar loading dan memuat scene secara asinkron. Berikut adalah penjelasan komponen ini:

* Loading Screen: LoadingType: Menentukan cara scene baru akan dimuat. Mode ini dapat berupa "Single" atau "Additive". "Single" menggantikan scene yang aktif saat ini dengan scene yang baru dimuat, sedangkan "Additive" memuat scene baru tanpa menggantikan scene yang ada.
* LoadingScene: AssetReference ke scene yang digunakan sebagai layar loading. Scene ini akan dimuat terlebih dahulu sebelum scene target dimuat. Ini adalah optional dan dapat diisi null jika tidak ada scene loading yang digunakan.
* TargetScene: AssetReference ke scene target yang akan dimuat. Scene ini adalah scene yang akan dimuat setelah scene loading (jika ada).
* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* usingDelay: Jika diaktifkan, ini memungkinkan Anda mengatur penundaan sebelum peristiwa tertentu dipicu.
* Delay: Menentukan durasi penundaan dalam detik sebelum peristiwa dipicu.
* DelayEvents: Ini adalah peristiwa yang akan terpicu setelah penundaan (Delay) selesai.
* Metode LoadScene(): Metode ini digunakan untuk memuat scene baru. Ini adalah cara yang umum digunakan untuk memindahkan pemain antar scene dalam permainan. Scene yang dituju harus telah didaftarkan di Build Settings dalam proyek Unity. Metode ini memulai operasi asinkron untuk memuat scene target dengan mode yang telah ditentukan. Jika LoadingScene tidak null, metode ini juga memuat scene loading terlebih dahulu (secara Additive) sebelum memuat scene target.
* Metode LoadAsyncScene(): Metode ini digunakan untuk memuat scene secara asinkron menggunakan Addressable Assets System.
* Jika LoadingScene tidak null, ini memuat scene loading terlebih dahulu (secara Additive) menggunakan Addressable Assets System. Setelah itu, scene target dimuat secara asinkron menggunakan Addressable Assets System.
* Update(): Ini adalah fungsi pembaruan yang dipanggil setiap frame. Di sini, peristiwa UpdateEvents dipicu.

### 2.6 Activation Language

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationLanguage : MonoBehaviour  {  [System.Serializable]  public enum CLanguageType  {  Indonesian, English, Korean, Chinese, Arabian, Japanese  }  [Header("Main Settings")]  public CLanguageType LanguageType;  public bool SaveOnAwake;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public UnityEvent IndonesianEvents;  public UnityEvent EnglishEvents;  public UnityEvent ChineseEvents;  public UnityEvent KoreanEvents;  public UnityEvent ArabianEvents;  public UnityEvent JapaneseEvents;  public void LoadLanguage()  {  if (PlayerPrefs.GetString("LANGUAGE") == "INDONESIAN")  {  LanguageType = CLanguageType.Indonesian;  IndonesianEvents?.Invoke();  }  if (PlayerPrefs.GetString("LANGUAGE") == "ENGLISH")  {  LanguageType = CLanguageType.English;  EnglishEvents?.Invoke();  }  if (PlayerPrefs.GetString("LANGUAGE") == "CHINESE")  {  LanguageType = CLanguageType.Chinese;  ChineseEvents?.Invoke();  }  if (PlayerPrefs.GetString("LANGUAGE") == "KOREAN")  {  LanguageType = CLanguageType.Korean;  KoreanEvents?.Invoke();  }  if (PlayerPrefs.GetString("LANGUAGE") == "ARABIAN")  {  LanguageType = CLanguageType.Arabian;  ArabianEvents?.Invoke();  }  if (PlayerPrefs.GetString("LANGUAGE") == "JAPANESE")  {  LanguageType = CLanguageType.Japanese;  JapaneseEvents?.Invoke();  }  Debug.Log(PlayerPrefs.GetString("LANGUAGE"));  }  public void SaveLanguage()  {  if (LanguageType == CLanguageType.Indonesian)  {  PlayerPrefs.SetString("LANGUAGE", "INDONESIAN");  }  if (LanguageType == CLanguageType.English)  {  PlayerPrefs.SetString("LANGUAGE", "ENGLISH");  }  if (LanguageType == CLanguageType.Chinese)  {  PlayerPrefs.SetString("LANGUAGE", "CHINESE");  }  if (LanguageType == CLanguageType.Korean)  {  PlayerPrefs.SetString("LANGUAGE", "KOREAN");  }  if (LanguageType == CLanguageType.Arabian)  {  PlayerPrefs.SetString("LANGUAGE", "ARABIAN");  }  if (LanguageType == CLanguageType.Japanese)  {  PlayerPrefs.SetString("LANGUAGE", "JAPANESE");  }  }  public void SelectIndonesian()  {  LanguageType = CLanguageType.Indonesian;  }  public void SelectEnglish()  {  LanguageType = CLanguageType.English;  }  public void SelectChinese()  {  LanguageType = CLanguageType.Chinese;  }  public void SelectKorean()  {  LanguageType = CLanguageType.Korean;  }  public void SelectArabian()  {  LanguageType = CLanguageType.Arabian;  }  public void SelectJapanese()  {  LanguageType = CLanguageType.Japanese;  }  public void SaveIndonesian()  {  LanguageType = CLanguageType.Indonesian;  SaveLanguage();  }  public void SaveEnglish()  {  LanguageType = CLanguageType.English;  SaveLanguage();  }  public void SaveChinese()  {  LanguageType = CLanguageType.Chinese;  SaveLanguage();  }  public void SaveKorean()  {  LanguageType = CLanguageType.Korean;  SaveLanguage();  }  public void SaveArabian()  {  LanguageType = CLanguageType.Arabian;  SaveLanguage();  }  public void SaveJapanese()  {  LanguageType = CLanguageType.Japanese;  SaveLanguage();  }  // Start is called before the first frame update  void Start()  {  if (SaveOnAwake)  {  SaveLanguage();  }  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini digunakan untuk mengelola pengaturan bahasa dalam permainan Unity. Ini memungkinkan pemain untuk memilih bahasa yang ingin digunakan dalam permainan dan menyimpan pengaturan bahasa tersebut. Berikut adalah penjelasan komponen ini:

* CLanguageType Enum: Ini adalah enumerasi yang digunakan untuk mendefinisikan berbagai jenis bahasa yang dapat dipilih oleh pemain. Enum ini mencakup bahasa-bahasa yang mungkin digunakan dalam permainan, seperti Bahasa Indonesia, Bahasa Inggris, Korea, Tionghoa, Arab, dan Jepang.
* LanguageType: Menunjukkan bahasa yang saat ini digunakan dalam permainan. Nilainya dapat diatur dari enum CLanguageType sesuai dengan bahasa yang ingin digunakan.
* SaveOnAwake: Jika diaktifkan, bahasa yang saat ini dipilih akan disimpan dalam PlayerPrefs saat permainan dimulai.
* StartEvents: Ini adalah peristiwa yang terpicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: Ini adalah peristiwa yang terpicu pada setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan. Terdapat peristiwa khusus untuk setiap jenis bahasa yang dapat dipilih (misalnya, IndonesianEvents, EnglishEvents, dll.), yang akan dipicu saat bahasa dipilih atau diubah.
* LoadLanguage(): Metode ini digunakan untuk memuat pengaturan bahasa yang telah disimpan dari PlayerPrefs.Ini memeriksa nilai PlayerPrefs dengan kunci "LANGUAGE" dan mengatur LanguageType sesuai dengan bahasa yang telah disimpan. Selanjutnya, itu memicu peristiwa khusus bahasa yang sesuai.
* SaveLanguage(): Metode ini digunakan untuk menyimpan pengaturan bahasa saat ini dalam PlayerPrefs. Ini memeriksa LanguageType saat ini dan menyimpannya dalam PlayerPrefs dengan kunci "LANGUAGE" sesuai dengan bahasa yang dipilih.
* Metode SelectXLanguage(): Ada metode khusus untuk setiap jenis bahasa yang dapat dipilih (misalnya, SelectIndonesian(), SelectEnglish(), dll.). Metode-metode ini digunakan untuk mengatur LanguageType sesuai dengan bahasa yang dipilih oleh pemain tanpa menyimpan pengaturan bahasa saat itu.
* Metode SaveXLanguage(): Ada metode khusus untuk setiap jenis bahasa yang dapat dipilih (misalnya, SaveIndonesian(), SaveEnglish(), dll.). Metode-metode ini digunakan untuk mengatur LanguageType sesuai dengan bahasa yang dipilih oleh pemain dan kemudian menyimpan pengaturan bahasa dalam PlayerPrefs.

### 2.7 Activation Pause

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationPause : MonoBehaviour  {  [Header("Active Status")]  public bool ApplicationActive = true;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Pause Settings")]  public UnityEvent PauseEvents;  public UnityEvent ResumeEvents;  bool CanTrigger;  public void Pause()  {  CanTrigger = true;  ApplicationActive = false;  Time.timeScale = 0f;  PauseEvents?.Invoke();  }  public void Play()  {  CanTrigger = true;  ApplicationActive = true;  Time.timeScale = 1f;  ResumeEvents?.Invoke();  }  // Start is called before the first frame update  void Start()  {  CanTrigger = false;  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  if (!ApplicationActive && CanTrigger)  {  Pause();  CanTrigger = false;  } else if (ApplicationActive && CanTrigger)  {  Play();  CanTrigger = false;  }  }  } |
| --- |

Script ini digunakan untuk mengontrol status jeda (pause) dan melanjutkan permainan dalam Unity. Dengan menggunakan script ini, Anda dapat menjeda dan melanjutkan permainan saat bermain. Berikut adalah penjelasan komponen-komponen dalam script ini:

* ApplicationActive: Boolean yang menunjukkan apakah aplikasi atau permainan saat ini dalam mode aktif (tidak dijeda) atau tidak.
* StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* PauseEvents: UnityEvent yang dipicu saat permainan dijeda.
* ResumeEvents: UnityEvent yang dipicu saat permainan dilanjutkan setelah dijeda.
* CanTrigger: Boolean yang digunakan untuk menghindari pemanggilan berulang dari metode Pause() dan Play(). Ini memastikan bahwa peristiwa hanya dipicu satu kali saat status berubah.
* Metode ini digunakan untuk menjeda permainan. CanTrigger diatur ke true untuk mengizinkan pemanggilan metode Play() saat permainan dilanjutkan.
* ApplicationActive diatur ke false untuk menunjukkan bahwa permainan saat ini dalam mode jeda.
* Time.timeScale diatur ke 0f untuk menghentikan waktu permainan sehingga permainan menjadi diam.
* Play(): Metode ini digunakan untuk melanjutkan permainan setelah dijeda. CanTrigger diatur ke true untuk mengizinkan pemanggilan metode Pause() saat permainan dijeda.
* ApplicationActive diatur ke true untuk menunjukkan bahwa permainan saat ini dalam mode aktif.
* Time.timeScale diatur ke 1f untuk mengembalikan waktu permainan ke kecepatan normal.
* Start(): Pada awalnya, CanTrigger diatur ke false. Ini memicu peristiwa StartEvents saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* Update(): Metode Update() ini dipanggil setiap frame. UpdateEvents dipicu setiap frame.Jika ApplicationActive adalah false (permainan dijeda) dan CanTrigger adalah true, maka metode Pause() dipanggil untuk menjeda permainan dan CanTrigger diatur kembali ke false untuk menghindari pemanggilan berulang. Jika ApplicationActive adalah true (permainan sedang berjalan) dan CanTrigger adalah true, maka metode Play() dipanggil untuk melanjutkan permainan dan CanTrigger diatur kembali ke false untuk menghindari pemanggilan berulang.

### 2.8 Activation Quit

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationQuit : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void QuitApplication()  {  //mematikan game saat dalam editor  #if UNITY\_EDITOR  UnityEditor.EditorApplication.isPlaying = false;  #endif  //mematikan game setelah di build  Application.Quit();  }  } |
| --- |

Script ini digunakan untuk mengimplementasikan fitur keluar (quit) dari aplikasi atau permainan Unity. Dengan menggunakan script ini, Anda dapat menghentikan permainan saat bermain atau berhenti dari mode pengeditan Unity Editor. Berikut adalah penjelasan komponen-komponen dalam script ini:

* StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* QuitApplication(): Metode ini dipanggil saat Anda ingin keluar dari aplikasi atau permainan.
* Pertama, kode ini memeriksa apakah permainan sedang berjalan di Unity Editor atau dalam mode build. Ini dilakukan dengan menggunakan preprocessor directives #if UNITY\_EDITOR.
* Jika permainan berjalan di Unity Editor (UNITY\_EDITOR aktif), maka permainan akan dihentikan dengan mengatur UnityEditor.EditorApplication.isPlaying ke false. Ini akan menghentikan permainan dalam mode pengeditan.
* Jika permainan berjalan dalam mode build (di luar Unity Editor), maka permainan akan dihentikan dengan memanggil Application.Quit(). Ini akan menghentikan permainan saat berada dalam mode build.
* Start(): Ini adalah metode yang dipanggil pada awal permainan atau saat objek yang menggunakan skrip ini pertama kali aktif. Peristiwa StartEvents dipicu saat metode ini dipanggil.
* Update(): Ini adalah metode yang dipanggil setiap frame permainan. Peristiwa UpdateEvents dipicu setiap frame saat metode ini dipanggil.

### 2.9 Activation Resolution

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationResolution : MonoBehaviour  {  [Header("Main Settings")]  public bool FullScreenMode;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  int Resolution = 1;    // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void SetFullScreenMode(bool aValue)  {  FullScreenMode = aValue;  }  public void SetResolution()  {  switch (Resolution)  {  case 1: SetResolution1366x768();  break;  case 2:  SetResolution1280x720();  break;  case 3:  SetResolution1920x1080();  break;  case 4:  SetResolution2560x1440();  break;  case 5:  SetResolution3840x2160();  break;  }  }  public void SetResolution1366x768()  {  Screen.SetResolution(1366, 768, FullScreenMode);  Resolution = 1;  }  public void SetResolution1280x720()  {  Screen.SetResolution(1280, 720, FullScreenMode);  Resolution = 2;  }  public void SetResolution1920x1080()  {  Screen.SetResolution(1920, 1080, FullScreenMode);  Resolution = 3;  }  public void SetResolution2560x1440()  {  Screen.SetResolution(2560, 1440, FullScreenMode);  Resolution = 4;  }  public void SetResolution3840x2160()  {  Screen.SetResolution(3840, 2160, FullScreenMode);  Resolution = 5;  }  } |
| --- |

Script ini digunakan untuk mengatur resolusi layar dan mode layar penuh (fullscreen) dalam permainan Unity. Script ini memberikan beberapa opsi resolusi yang dapat dipilih, dan Anda juga dapat mengubah mode layar penuh. Berikut adalah penjelasan komponen-komponen dalam script ini:

* FullScreenMode: Boolean yang menentukan apakah permainan akan berjalan dalam mode layar penuh atau tidak.
* StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* Resolution: Variabel integer yang digunakan untuk melacak resolusi yang dipilih saat ini. Nilai ini digunakan dalam metode SetResolution() untuk menentukan resolusi yang akan diatur.
* SetFullScreenMode(bool aValue): Metode ini digunakan untuk mengatur mode layar penuh. Ketika dipanggil dengan aValue yang merupakan nilai boolean, mode layar penuh akan diatur sesuai dengan nilai tersebut.
* SetResolution(): Metode ini digunakan untuk mengatur resolusi layar berdasarkan nilai yang disimpan dalam variabel Resolution. Ini adalah metode utama yang akan memanggil metode resolusi khusus sesuai dengan nilai Resolution saat ini.
* SetResolution1366x768() hingga SetResolution3840x2160(): Metode-metode ini digunakan untuk mengatur resolusi layar ke resolusi tertentu yang telah ditentukan. Masing-masing metode mengatur resolusi layar dengan menggunakan Screen.SetResolution() dengan resolusi yang sesuai.
* Start(): Ini adalah metode yang dipanggil pada awal permainan atau saat objek yang menggunakan skrip ini pertama kali aktif. Peristiwa StartEvents dipicu saat metode ini dipanggil.
* Update(): Ini adalah metode yang dipanggil setiap frame permainan. Peristiwa UpdateEvents dipicu setiap frame saat metode ini dipanggil.

### 2.10 Activation Timer

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ActivationTimer : MonoBehaviour  {  [System.Serializable]  public class TimedEvent  {  public float time;  public UnityEvent unityEvent;  }  [Header("Main Settings")]  public UnityEvent MainEvent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Timer Events")]  public List<TimedEvent> timerEvents = new List<TimedEvent>();  private float startTime;  // Start is called before the first frame update  void Start()  {  startTime = Time.time;  StartEvents?.Invoke();  }  public void InvokeMainEvents()  {  MainEvent?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  float currentTime = Time.time - startTime;  for (int i = timerEvents.Count - 1; i >= 0; i--)  {  if (currentTime >= timerEvents[i].time)  {  timerEvents[i].unityEvent?.Invoke();  timerEvents.RemoveAt(i); // Remove the event after invoking to prevent it from being called again  }  }  }  } |
| --- |

## 3. Visualization

Event Visualization dalam game berhubungan dengan cara elemen-elemen dalam game, seperti karakter, objek, atau latar belakang, dirender atau ditampilkan. Ini termasuk efek visual seperti cahaya, bayangan, dan partikel. Dalam pemrograman, ini biasanya melibatkan penggunaan library atau framework khusus yang dibuat untuk memudahkan pengelolaan grafis, seperti Unity atau Unreal Engine. Misalnya, jika sebuah karakter menggunakan sihir dalam game, programmer perlu menulis kode untuk menampilkan efek visual yang terkait dengan sihir tersebut, seperti kilauan cahaya atau partikel yang terbang. Pemrograman untuk event Visualization ini melibatkan penggunaan berbagai fungsi dan metode yang disediakan oleh library atau framework yang digunakan, serta pemahaman yang baik tentang konsep grafis komputer seperti buffer, tekstur, dan shader. Menulis kode yang efisien dan efektif untuk mengelola visual dalam game adalah kunci untuk menciptakan pengalaman yang menarik dan mendalam bagi pemain.

### 

### 3.1 Visualization Fade

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class VisualizationFadeIn : MonoBehaviour  {  [Header("Fade In")]  public Color SplashColor;  public float StartDelay;  public float AlphaSpeed = 1;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [HideInInspector] public float \_alpha = 1;  [HideInInspector] public Texture2D \_texture;  [HideInInspector] public bool \_done;  [HideInInspector] public float \_time;  [HideInInspector] public bool \_isfading = false;  // Start is called before the first frame update  void Start()  {  if (StartDelay > 0)  {  \_alpha = 1;  Invoke("ExecFadeIn", StartDelay);  }  else  {  \_isfading = true;  }  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  void ExecFadeIn()  {  \_isfading = true;  }  void OnGUI()  {  if (\_isfading)  {  if (\_done) return;  if (\_texture == null) \_texture = new Texture2D(1, 1);  \_texture.SetPixel(0, 0, new Color(SplashColor.r, SplashColor.g, SplashColor.b, \_alpha));  \_texture.Apply();  \_time += Time.deltaTime \* AlphaSpeed;  \_alpha -= \_time \* Time.deltaTime;  GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), \_texture);  if (\_alpha <= 0) \_done = true;  }  }  } |
| --- |

Script ini digunakan untuk membuat efek fade-in (perlahan muncul) dengan latar belakang berwarna. Efek ini dapat digunakan sebagai efek awal permainan atau transisi antara layar permainan. Berikut adalah penjelasan komponen-komponen dalam script ini:

* SplashColor: Warna latar belakang yang akan ditampilkan selama efek fade-in.
* StartDelay: Jeda sebelum efek fade-in dimulai. Jika lebih dari 0, maka efek akan dimulai setelah jeda ini berakhir. AlphaSpeed: Kecepatan perubahan alpha (transparansi) selama efek fade-in.
* StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* Variabel \_alpha: Menyimpan nilai alpha (transparansi) saat ini dari efek fade-in.
* Variabel \_texture: Sebuah objek Texture2D yang digunakan untuk menggambar latar belakang berwarna selama efek fade-in.
* Variabel \_done: Boolean yang menandakan apakah efek fade-in sudah selesai atau belum.
* Variabel \_time: Menyimpan waktu sejak efek fade-in dimulai.
* Variabel \_isfading: Boolean yang menandakan apakah efek fade-in sedang berlangsung atau tidak.
* Start(): Metode ini dipanggil pada awal permainan atau saat objek yang menggunakan skrip ini pertama kali aktif. Jika StartDelay lebih dari 0, maka efek fade-in akan dijalankan setelah jeda waktu sesuai dengan nilai StartDelay. Jika tidak, efek fade-in akan dimulai segera.
* Peristiwa StartEvents dipicu saat metode ini dipanggil.
* Update(): Metode ini dipanggil setiap frame permainan. Peristiwa UpdateEvents dipicu setiap frame saat metode ini dipanggil.
* ExecFadeIn(): Metode ini digunakan untuk mulai menjalankan efek fade-in dengan mengatur \_isfading menjadi true.
* OnGUI(): Metode ini digunakan untuk menggambar efek fade-in di layar. Jika \_isfading adalah true, maka efek fade-in akan digambar. Selama efek fade-in berlangsung, latar belakang berwarna dengan alpha yang berkurang akan digambar pada seluruh layar. Perubahan alpha ini diatur berdasarkan waktu (\_time) dan kecepatan (AlphaSpeed). Ketika alpha mencapai 0 (sepenuhnya transparan), \_done akan diatur ke true, menandakan bahwa efek fade-in telah selesai.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class VisualizationFadeOut : MonoBehaviour  {  [Header("Fade Out")]  public Color SplashColor;  public float StartDelay;  public float AlphaSpeed = 1;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [HideInInspector] public float \_alpha = 1;  [HideInInspector] public Texture2D \_texture;  [HideInInspector] public bool \_done;  [HideInInspector] public float \_time;  [HideInInspector] public bool \_isfading = false;  // Start is called before the first frame update  void Start()  {  if (StartDelay > 0)  {  \_alpha = 0;  Invoke("ExecFadeOut", StartDelay);  }  else  {  \_alpha = 0;  \_isfading = true;  }  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  void ExecFadeOut()  {  \_isfading = true;  }  void OnGUI()  {  if (\_isfading)  {  if (\_texture == null) \_texture = new Texture2D(1, 1);  \_texture.SetPixel(0, 0, new Color(SplashColor.r, SplashColor.g, SplashColor.b, \_alpha));  \_texture.Apply();  if (!\_done)  {  \_time += Time.deltaTime \* AlphaSpeed;  \_alpha += \_time \* Time.deltaTime;  }  GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), \_texture);  if (\_alpha > 1)  {  \_done = true;  Destroy(GameObject.Find("Canvas"));  Destroy(this.gameObject);  }  }  }  } |
| --- |

Script ini hampir sama dengan script "VisualizationFadeIn", tetapi digunakan untuk membuat efek fade-out (perlahan memudar) dengan latar belakang berwarna. Ini adalah efek visual yang biasanya digunakan untuk mengakhiri permainan atau transisi antara layar permainan. Berikut adalah penjelasan komponen-komponen dalam script ini:

* SplashColor: Warna latar belakang yang akan ditampilkan selama efek fade-out.
* StartDelay: Jeda sebelum efek fade-out dimulai. Jika lebih dari 0, maka efek akan dimulai setelah jeda ini berakhir.
* AlphaSpeed: Kecepatan perubahan alpha (transparansi) selama efek fade-out.
* StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* Variabel \_alpha: Menyimpan nilai alpha (transparansi) saat ini dari efek fade-out.
* Variabel \_texture: Sebuah objek Texture2D yang digunakan untuk menggambar latar belakang berwarna selama efek fade-out.
* Variabel \_done: Boolean yang menandakan apakah efek fade-out sudah selesai atau belum.
* Variabel \_time: Menyimpan waktu sejak efek fade-out dimulai.
* Variabel \_isfading: Boolean yang menandakan apakah efek fade-out sedang berlangsung atau tidak.
* Start(): Metode ini dipanggil pada awal permainan atau saat objek yang menggunakan skrip ini pertama kali aktif. Jika StartDelay lebih dari 0, maka efek fade-out akan dijalankan setelah jeda waktu sesuai dengan nilai StartDelay. Jika tidak, efek fade-out akan dimulai segera.
* Peristiwa StartEvents dipicu saat metode ini dipanggil.
* Update(): Metode ini dipanggil setiap frame permainan. Peristiwa UpdateEvents dipicu setiap frame saat metode ini dipanggil.
* ExecFadeOut(): Metode ini digunakan untuk mulai menjalankan efek fade-out dengan mengatur \_isfading menjadi true.
* OnGUI(): Metode ini digunakan untuk menggambar efek fade-out di layar. Jika \_isfading adalah true, maka efek fade-out akan digambar. Selama efek fade-out berlangsung, latar belakang berwarna dengan alpha yang berubah akan digambar pada seluruh layar. Perubahan alpha ini diatur berdasarkan waktu (\_time) dan kecepatan (AlphaSpeed). Ketika alpha mencapai lebih dari 1 (sepenuhnya tidak transparan), \_done akan diatur ke true. Pada saat ini, objek "Canvas" yang memiliki efek ini juga akan dihancurkan bersama dengan objek ini sendiri.

### 3.2 Visualization Leaderboard

| using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class VisualizationLeaderboard : MonoBehaviour  {  [System.Serializable]  public class ScoreData  {  public string Name;  public float Score;  }  [System.Serializable]  public class TextElements  {  public Text NameText;  public Text ScoreText;  }  [Header("Leaderboard Settings")]  public ScoreData[] Scores = new ScoreData[10];  public TextElements[] Texts = new TextElements[10];  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  private const string PrefKey = "Leaderboard";  void Start()  {  StartEvents?.Invoke();  }  public void AddNewName(string name)  {  Scores[9].Name = name;  }  public void AddNewScore(float value)  {  Scores[9].Score = value;  }  public void SortLeaderboard()  {  SortScores();  }  public void AddNew(string name, int score)  {  Scores[9].Name = name;  Scores[9].Score = score;  SortScores();  SaveScores();  LoadScoresUI();  }  void SortScores()  {  System.Array.Sort(Scores, (x, y) => y.Score.CompareTo(x.Score));  }  public void SaveScores()  {  for (int i = 0; i < 10; i++)  {  PlayerPrefs.SetString(PrefKey + i.ToString() + "Name", Scores[i].Name);  PlayerPrefs.SetFloat(PrefKey + i.ToString() + "Score", Scores[i].Score);  }  }  public void LoadScores()  {  for (int i = 0; i < 10; i++)  {  Scores[i].Name = PlayerPrefs.GetString(PrefKey + i.ToString() + "Name", "Player");  Scores[i].Score = PlayerPrefs.GetFloat(PrefKey + i.ToString() + "Score", 100);  }  }  public void ShowScoreData()  {  Debug.Log(PlayerPrefs.GetString(PrefKey));  }  public void LoadScoresUI()  {  LoadScores();  for (int i = 0; i < 10; i++)  {  if (Scores[i] != null)  {  Texts[i].NameText.text = Scores[i].Name;  Texts[i].ScoreText.text = Scores[i].Score.ToString();  }  }  }  private void Update()  {  UpdateEvents?.Invoke();  }  public void InitLeaderboard()  {  LoadScores();  SaveScores();  LoadScoresUI();  }  } |
| --- |

Script ini digunakan untuk mengatur leaderboard dalam permainan dengan menggunakan Unity. Leaderboard adalah daftar pemain teratas yang mencakup nama pemain dan skor mereka. Berikut adalah penjelasan komponen-komponen dalam script ini:

* ScoreData Class: Ini adalah class yang digunakan untuk menyimpan data pemain, termasuk nama dan skor. Setiap elemen dalam array Scores akan berisi instance dari class ini.
* TextElements Class: Ini adalah class yang digunakan untuk menyimpan elemen-elemen teks yang akan digunakan untuk menampilkan leaderboard. Setiap elemen dalam array Texts akan berisi instance dari class ini.
* Leaderboard Settings: Scores: Array dari ScoreData yang akan digunakan untuk menyimpan data pemain dalam leaderboard. Array ini memiliki panjang 10, yang berarti leaderboard dapat menyimpan hingga 10 pemain teratas.
* Texts: Array dari TextElements yang akan digunakan untuk menampilkan data pemain pada UI leaderboard.
* Event Settings: StartEvents: UnityEvent yang dipicu saat objek yang menggunakan skrip ini pertama kali aktif dalam permainan.
* UpdateEvents: UnityEvent yang dipicu setiap frame saat objek yang menggunakan skrip ini aktif dalam permainan.
* PrefKey: Konstanta string yang digunakan sebagai kunci untuk menyimpan data leaderboard di PlayerPrefs.
* Start(): Metode ini dipanggil pada awal permainan atau saat objek yang menggunakan skrip ini pertama kali aktif. Peristiwa StartEvents dipicu saat metode ini dipanggil.
* AddNewName(string name): Metode untuk menambahkan nama pemain baru ke dalam data leaderboard. Nama pemain ini akan ditempatkan pada posisi terakhir dalam leaderboard.
* AddNewScore(float value): Metode untuk menambahkan skor pemain baru ke dalam data leaderboard. Skor pemain ini akan ditempatkan pada posisi terakhir dalam leaderboard.
* SortLeaderboard(): Metode untuk mengurutkan leaderboard berdasarkan skor tertinggi. Metode ini memanggil SortScores() untuk mengurutkan array Scores.
* AddNew(string name, int score): Metode untuk menambahkan data pemain baru ke dalam leaderboard, termasuk nama dan skor. Setelah data ditambahkan, leaderboard akan diurutkan.
* SortScores(): Metode untuk mengurutkan data leaderboard berdasarkan skor tertinggi dengan bantuan fungsi System.Array.Sort().
* SaveScores(): Metode untuk menyimpan data leaderboard ke PlayerPrefs. Setiap pemain dan skornya disimpan dengan menggunakan kunci yang menggabungkan indeks pemain dalam array.
* LoadScores(): Metode untuk memuat data leaderboard dari PlayerPrefs dan mengisinya kembali ke array Scores.
* ShowScoreData(): Metode yang digunakan untuk menampilkan data leaderboard dalam bentuk string di konsol untuk debugging.
* LoadScoresUI(): Metode untuk menampilkan data leaderboard pada UI. Ini memuat data dari array Scores ke elemen-elemen teks yang ada di dalam array Texts.
* Update(): Metode ini dipanggil setiap frame permainan. Peristiwa UpdateEvents dipicu setiap frame saat metode ini dipanggil.

### 3.3 Visualization Score

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class VisualizationScore : MonoBehaviour  {  public string dataKey;  public float maxScore;  public float factorScore;  public Text scoreText;  public Toggle toggle1;  public Toggle toggle2;  public Toggle toggle3;  private bool isAnimating;  private float animationTime = 3f; // durasi animasi dalam detik  float playerScore;  float uiscore;  void Start()  {  playerScore = PlayerPrefs.GetFloat(dataKey, 0f) \* factorScore;  maxScore = maxScore \* factorScore;  isAnimating = true;  }  void Update()  {  if (isAnimating)  {  float newScore = Mathf.Lerp(0f, playerScore, Time.time / animationTime);  uiscore = Mathf.Round(newScore);  scoreText.text = "SCORE: " + uiscore.ToString();  if (uiscore >= playerScore)  {  isAnimating = false;  uiscore = playerScore;  scoreText.text = "SCORE: " + uiscore.ToString();  Debug.Log(uiscore / maxScore);  // Aktifkan toggle jika total score mencapai persentase tertentu  if ((uiscore / maxScore) >= 0.3f)  {  toggle1.isOn = true;  }  if ((uiscore / maxScore) >= 0.5f)  {  toggle2.isOn = true;  }  if ((uiscore / maxScore) >= 0.8f)  {  toggle3.isOn = true;  }  }  }  }  } |
| --- |

Script ini digunakan untuk mengatur tampilan skor dalam permainan dan melakukan animasi peningkatan skor. Berikut adalah penjelasan komponen-komponen dalam script ini:

* dataKey: Variabel string yang digunakan sebagai kunci untuk mengakses data skor pemain dari PlayerPrefs. PlayerPrefs digunakan untuk menyimpan data permainan di Unity.
* maxScore: Variabel float yang menentukan skor maksimum yang dapat dicapai oleh pemain dalam permainan.
* factorScore: Faktor yang digunakan untuk mengalikan skor pemain dari nilai yang disimpan di PlayerPrefs. Ini memungkinkan untuk mengubah nilai skor yang disimpan di PlayerPrefs menjadi nilai yang lebih relevan dalam permainan.
* scoreText: Referensi ke komponen teks di UI yang akan digunakan untuk menampilkan skor pemain.
* toggle1, toggle2, toggle3: Referensi ke komponen Toggle di UI. Ini adalah toggle yang akan diaktifkan berdasarkan persentase skor pemain terhadap skor maksimum.
* isAnimating: Variabel boolean yang menunjukkan apakah animasi peningkatan skor sedang berlangsung.
* animationTime: Variabel float yang menentukan durasi animasi peningkatan skor dalam detik.
* playerScore: Variabel float yang menyimpan skor pemain setelah diambil dari PlayerPrefs dan dikalikan dengan factorScore.
* uiscore: Variabel float yang menyimpan skor yang akan ditampilkan pada UI. Ini akan secara bertahap ditingkatkan selama animasi.
* Start(): Pada awalnya, skor pemain diambil dari PlayerPrefs dengan menggunakan dataKey dan dikalikan dengan factorScore.
* Nilai maxScore juga dikalikan dengan factorScore untuk memastikan bahwa kedua nilai sesuai dalam skala yang sama.
* Variabel isAnimating diatur sebagai true, yang menandakan bahwa animasi peningkatan skor akan dimulai.
* Update(): Dalam setiap frame, script ini akan memeriksa apakah animasi peningkatan skor masih berlangsung (ditentukan oleh isAnimating). Jika animasi sedang berlangsung, skor yang ditampilkan pada UI (scoreText.text) akan diupdate dengan menggunakan interpolasi linear (Lerp) antara 0 dan skor pemain. Ini memberikan efek animasi peningkatan skor. Skor ini dibulatkan ke bilangan bulat terdekat (Mathf.Round(newScore)) sebelum ditampilkan pada UI. Jika nilai uiscore mencapai skor pemain (playerScore), animasi dihentikan (isAnimating diatur sebagai false), dan skor terakhir ditampilkan. Selanjutnya, script ini memeriksa persentase skor pemain terhadap skor maksimum (uiscore / maxScore) dan mengaktifkan toggle tertentu (toggle1, toggle2, atau toggle3) berdasarkan nilai persentase tertentu. Debug.Log() digunakan untuk menampilkan persentase skor pemain terhadap skor maksimum di konsol.

### 3.4 Visualization Parallax

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class VisualizationParallax : MonoBehaviour  {  public enum Action  {  ZoomIn,  ZoomOut,  MoveLeft,  MoveRight,  MoveUp,  MoveDown  }  [Header("Parallax Setting")]  public bool Active;  public float translateFactor = 0.01f;  [Header("Image Setting")]  public Image background;  public Image foreground;  public Image textground;  [Header("Speed Setting")]  public Action actionBackground = Action.ZoomIn;  public float speedBackground = 1.0f;  public Action actionForeground = Action.ZoomOut;  public float speedForeground = 2.0f;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Time Settings")]  public float AfterDelay;  public UnityEvent AfterDelaySettings;  float timeElapsed;  private float currentScale1 = 1.0f;  private float currentScale2 = 1.0f;  private void Start()  {  background.gameObject.SetActive(Active);  foreground.gameObject.SetActive(Active);  textground.gameObject.SetActive(Active);  StartEvents?.Invoke();  }  public void SetActive(bool Value)  {  Active = Value;  }  public void Show()  {  background.gameObject.SetActive(true);  foreground.gameObject.SetActive(true);  textground.gameObject.SetActive(true);  SetActive(true);  }  public void Hide()  {  background.gameObject.SetActive(false);  foreground.gameObject.SetActive(false);  textground.gameObject.SetActive(false);  SetActive(false);  }  private void Update()  {  if (Active)  {  if (timeElapsed >= AfterDelay)  {  AfterDelaySettings.Invoke();  }  timeElapsed += Time.deltaTime;  if (actionBackground == Action.ZoomIn)  {  currentScale1 += Time.deltaTime \* speedBackground \* translateFactor;  }  else if (actionBackground == Action.ZoomOut)  {  currentScale1 -= Time.deltaTime \* speedBackground \* translateFactor;  }  else if (actionBackground == Action.MoveLeft)  {  background.transform.position += new Vector3(-Time.deltaTime \* speedBackground \* (1/translateFactor), 0.0f, 0.0f);  }  else if (actionBackground == Action.MoveRight)  {  background.transform.position += new Vector3(Time.deltaTime \* speedBackground \* (1 / translateFactor), 0.0f, 0.0f);  }  else if (actionBackground == Action.MoveUp)  {  background.transform.position += new Vector3(0.0f, Time.deltaTime \* speedBackground \* (1 / translateFactor), 0.0f);  }  else if (actionBackground == Action.MoveDown)  {  background.transform.position += new Vector3(0.0f, -Time.deltaTime \* speedBackground \* (1 / translateFactor), 0.0f);  }  if (actionForeground == Action.ZoomIn)  {  currentScale2 += Time.deltaTime \* speedForeground \* translateFactor;  }  else if (actionForeground == Action.ZoomOut)  {  currentScale2 -= Time.deltaTime \* speedForeground \* translateFactor;  }  else if (actionForeground == Action.MoveLeft)  {  foreground.transform.position += new Vector3(-Time.deltaTime \* speedForeground, 0.0f, 0.0f);  }  else if (actionForeground == Action.MoveRight)  {  foreground.transform.position += new Vector3(Time.deltaTime \* speedForeground, 0.0f, 0.0f);  }  else if (actionForeground == Action.MoveUp)  {  foreground.transform.position += new Vector3(0.0f, Time.deltaTime \* speedForeground, 0.0f);  }  else if (actionForeground == Action.MoveDown)  {  foreground.transform.position += new Vector3(0.0f, -Time.deltaTime \* speedForeground, 0.0f);  }  background.transform.localScale = new Vector3(currentScale1, currentScale1, currentScale1);  foreground.transform.localScale = new Vector3(currentScale2, currentScale2, currentScale2);  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini digunakan untuk mengimplementasikan efek paralaks pada elemen-elemen gambar (Image) dalam permainan. Efek paralaks ini memberikan ilusi gerakan atau perubahan ukuran pada elemen-elemen gambar untuk menciptakan efek visual yang menarik. Berikut adalah penjelasan komponen-komponen dalam script ini:

* Active: Variabel boolean yang menentukan apakah efek paralaks aktif atau tidak. Ketika tidak aktif, elemen-elemen gambar (background, foreground, textground) akan disembunyikan.
* translateFactor: Faktor yang digunakan untuk mengatur sejauh mana pergerakan atau perubahan ukuran elemen gambar.
* background, foreground, textground: Referensi ke komponen Image yang akan diberi efek paralaks.
* actionBackground, actionForeground: Jenis aksi yang akan dilakukan pada elemen-elemen gambar. Ini bisa berupa zoom in, zoom out, atau pergerakan horizontal dan vertikal.
* speedBackground, speedForeground: Kecepatan perubahan yang akan diterapkan pada elemen-elemen gambar.
* StartEvents, UpdateEvents: UnityEvent yang digunakan untuk memicu peristiwa pada awal dan selama permainan.
* AfterDelay: Waktu tunda sebelum memicu peristiwa yang ditentukan dalam AfterDelaySettings.
* currentScale1, currentScale2: Skala saat ini untuk elemen-elemen gambar. Digunakan untuk menganimasikan efek zoom in atau zoom out.
* timeElapsed: Waktu yang telah berlalu sejak permainan dimulai.
* Start(): Pada awalnya, elemen-elemen gambar diaktifkan atau dinonaktifkan sesuai dengan nilai Active. StartEvents diaktifkan.
* SetActive(bool Value): Method ini digunakan untuk mengatur apakah efek paralaks aktif atau tidak. Ketika Value diset sebagai true, elemen-elemen gambar diaktifkan, dan ketika false, elemen-elemen gambar dinonaktifkan.
* Show() dan Hide(): Method ini digunakan untuk menampilkan atau menyembunyikan elemen-elemen gambar secara langsung.
* Update(): Jika efek paralaks aktif, maka dalam setiap frame, script ini akan memeriksa apakah waktu yang telah berlalu (timeElapsed) sudah mencapai AfterDelay. Jika sudah mencapai, AfterDelaySettings akan dipicu.
* Nilai timeElapsed akan terus bertambah seiring berjalannya waktu.
* Terdapat perulangan yang mengubah skala atau posisi elemen-elemen gambar sesuai dengan jenis aksi yang diatur (actionBackground dan actionForeground). Aksi ini dapat berupa zoom in, zoom out, atau pergerakan horizontal dan vertikal.
* Skala elemen gambar diupdate dengan menggunakan nilai currentScale1 dan currentScale2.
* Efek perubahan skala dan pergerakan ini menciptakan efek paralaks yang terlihat oleh pemain dalam permainan.
* UpdateEvents diaktifkan dalam setiap frame.

### 3.5 Visualization Text

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class VisualizationText : MonoBehaviour  {  [Header("Language Settings")]  [TextArea(5, 5)]  public string IndonesianText;  [TextArea(5, 5)]  public string EnglishText;  [TextArea(5, 5)]  public string KoreanText;  [TextArea(5, 5)]  public string ChineseText;  [TextArea(5, 5)]  public string ArabianText;  [TextArea(5, 5)]  public string JapaneseText;  private Text CurrentText;  string CurrentLanguage;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  CurrentText = GetComponent<Text>();  InvokeRepeating("UpdateText", 0, 1f);  StartEvents?.Invoke();  }  // Update is called once per frame  void UpdateText()  {  CurrentLanguage = PlayerPrefs.GetString("LANGUAGE");  if (PlayerPrefs.GetString("LANGUAGE") == "INDONESIAN")  {  CurrentText.text = IndonesianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ENGLISH")  {  CurrentText.text = EnglishText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "CHINESE")  {  CurrentText.text = ChineseText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "KOREAN")  {  CurrentText.text = KoreanText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ARABIAN")  {  CurrentText.text = ArabianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "JAPANESE")  {  CurrentText.text = JapaneseText;  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini digunakan untuk mengatur teks pada komponen UI Text berdasarkan bahasa yang dipilih dalam permainan. Script ini memungkinkan pemain untuk memilih bahasa, dan teks akan diperbarui sesuai dengan bahasa yang dipilih. Berikut adalah penjelasan komponen-komponen dalam script ini:

* IndonesianText, EnglishText, KoreanText, ChineseText, ArabianText, JapaneseText: Teks dalam bahasa-bahasa yang berbeda. Setiap variabel ini berisi teks yang akan ditampilkan dalam bahasa yang sesuai.
* CurrentText: Variabel untuk menyimpan komponen UI Text yang terdapat pada objek ini. Komponen ini akan digunakan untuk mengubah teks yang ditampilkan.
* CurrentLanguage: Variabel untuk menyimpan bahasa yang sedang dipilih oleh pemain. Nilai ini akan diambil dari PlayerPrefs.
* StartEvents, UpdateEvents: UnityEvent yang digunakan untuk memicu peristiwa pada awal dan selama permainan.
* Start(): Pada awal permainan, komponen UI Text yang ada pada objek ini diambil dan disimpan dalam variabel CurrentText.
* Method UpdateText() akan dijalankan dengan interval waktu tertentu menggunakan InvokeRepeating() untuk memperbarui teks yang ditampilkan.
* StartEvents diaktifkan.
* UpdateText(): Method ini digunakan untuk memperbarui teks yang ditampilkan sesuai dengan bahasa yang dipilih oleh pemain.
* Bahasa yang dipilih diambil dari PlayerPrefs dengan menggunakan PlayerPrefs.GetString("LANGUAGE").
* Berdasarkan bahasa yang dipilih, teks yang sesuai (IndonesianText, EnglishText, dll.) akan ditampilkan pada komponen UI Text yang ada pada objek ini (CurrentText).

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class VisualizationTextMesh : MonoBehaviour  {  [Header("Language Settings")]  [TextArea(5, 5)]  public string IndonesianText;  [TextArea(5, 5)]  public string EnglishText;  [TextArea(5, 5)]  public string KoreanText;  [TextArea(5, 5)]  public string ChineseText;  [TextArea(5, 5)]  public string ArabianText;  [TextArea(5, 5)]  public string JapaneseText;  private TextMesh CurrentText;  string CurrentLanguage;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  CurrentText = GetComponent<TextMesh>();  InvokeRepeating("UpdateText", 1, 1);  StartEvents?.Invoke();  }  // Update is called once per frame  void UpdateText()  {  CurrentLanguage = PlayerPrefs.GetString("LANGUAGE");  if (PlayerPrefs.GetString("LANGUAGE") == "INDONESIAN")  {  CurrentText.text = IndonesianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ENGLISH")  {  CurrentText.text = EnglishText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "CHINESE")  {  CurrentText.text = ChineseText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "KOREAN")  {  CurrentText.text = KoreanText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ARABIAN")  {  CurrentText.text = ArabianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "JAPANESE")  {  CurrentText.text = JapaneseText;  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini serupa dengan skrip sebelumnya, tetapi digunakan untuk mengatur teks pada komponen TextMesh berdasarkan bahasa yang dipilih dalam permainan. Seperti sebelumnya, script ini memungkinkan pemain untuk memilih bahasa, dan teks akan diperbarui sesuai dengan bahasa yang dipilih. Berikut adalah penjelasan komponen-komponen dalam script ini:

* IndonesianText, EnglishText, KoreanText, ChineseText, ArabianText, JapaneseText: Teks dalam bahasa-bahasa yang berbeda. Setiap variabel ini berisi teks yang akan ditampilkan dalam bahasa yang sesuai.
* CurrentText: Variabel untuk menyimpan komponen TextMesh yang terdapat pada objek ini. Komponen ini akan digunakan untuk mengubah teks yang ditampilkan.
* CurrentLanguage: Variabel untuk menyimpan bahasa yang sedang dipilih oleh pemain. Nilai ini akan diambil dari PlayerPrefs.
* StartEvents, UpdateEvents: UnityEvent yang digunakan untuk memicu peristiwa pada awal dan selama permainan.
* Start(): Pada awal permainan, komponen TextMesh yang ada pada objek ini diambil dan disimpan dalam variabel CurrentText.
* Method UpdateText() akan dijalankan dengan interval waktu tertentu menggunakan InvokeRepeating() untuk memperbarui teks yang ditampilkan.
* StartEvents diaktifkan.
* UpdateText(): Method ini digunakan untuk memperbarui teks yang ditampilkan sesuai dengan bahasa yang dipilih oleh pemain.
* Bahasa yang dipilih diambil dari PlayerPrefs dengan menggunakan PlayerPrefs.GetString("LANGUAGE").
* Berdasarkan bahasa yang dipilih, teks yang sesuai (IndonesianText, EnglishText, dll.) akan ditampilkan pada komponen TextMesh yang ada pada objek ini (CurrentText).

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using TMPro;  using UnityEngine.Events;  public class VisualizationTextMeshPro : MonoBehaviour  {  [Header("Language Settings")]  [TextArea(5, 5)]  public string IndonesianText;  [TextArea(5, 5)]  public string EnglishText;  [TextArea(5, 5)]  public string KoreanText;  [TextArea(5, 5)]  public string ChineseText;  [TextArea(5, 5)]  public string ArabianText;  [TextArea(5, 5)]  public string JapaneseText;  private TMP\_Text CurrentText;  string CurrentLanguage;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  // Start is called before the first frame update  void Start()  {  CurrentText = GetComponent<TMP\_Text>();  InvokeRepeating("UpdateText", 1, 1);  StartEvents?.Invoke();  }  // Update is called once per frame  void UpdateText()  {  CurrentLanguage = PlayerPrefs.GetString("LANGUAGE");  if (PlayerPrefs.GetString("LANGUAGE") == "INDONESIAN")  {  CurrentText.text = IndonesianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ENGLISH")  {  CurrentText.text = EnglishText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "CHINESE")  {  CurrentText.text = ChineseText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "KOREAN")  {  CurrentText.text = KoreanText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "ARABIAN")  {  CurrentText.text = ArabianText;  }  if (PlayerPrefs.GetString("LANGUAGE") == "JAPANESE")  {  CurrentText.text = JapaneseText;  }  UpdateEvents?.Invoke();  }  } |
| --- |

Script ini sangat mirip dengan skrip sebelumnya, tetapi digunakan untuk mengatur teks pada komponen TextMeshPro (TMP\_Text) berdasarkan bahasa yang dipilih dalam permainan. Seperti sebelumnya, script ini memungkinkan pemain untuk memilih bahasa, dan teks akan diperbarui sesuai dengan bahasa yang dipilih. Berikut adalah penjelasan komponen-komponen dalam script ini:

* IndonesianText, EnglishText, KoreanText, ChineseText, ArabianText, JapaneseText: Teks dalam bahasa-bahasa yang berbeda. Setiap variabel ini berisi teks yang akan ditampilkan dalam bahasa yang sesuai.
* CurrentText: Variabel untuk menyimpan komponen TextMeshPro (TMP\_Text) yang terdapat pada objek ini. Komponen ini akan digunakan untuk mengubah teks yang ditampilkan.
* CurrentLanguage: Variabel untuk menyimpan bahasa yang sedang dipilih oleh pemain. Nilai ini akan diambil dari PlayerPrefs.
* StartEvents, UpdateEvents: UnityEvent yang digunakan untuk memicu peristiwa pada awal dan selama permainan.
* Pada awal permainan, komponen TextMeshPro (TMP\_Text) yang ada pada objek ini diambil dan disimpan dalam variabel CurrentText.
* Method UpdateText() akan dijalankan dengan interval waktu tertentu menggunakan InvokeRepeating() untuk memperbarui teks yang ditampilkan.
* UpdateText(): Method ini digunakan untuk memperbarui teks yang ditampilkan sesuai dengan bahasa yang dipilih oleh pemain.
* Bahasa yang dipilih diambil dari PlayerPrefs dengan menggunakan PlayerPrefs.GetString("LANGUAGE"). Berdasarkan bahasa yang dipilih, teks yang sesuai (IndonesianText, EnglishText, dll.) akan ditampilkan pada komponen TextMeshPro (TMP\_Text) yang ada pada objek ini (CurrentText).

### 3.6 Visualization InputField

|  |
| --- |

|  |
| --- |

|  |
| --- |

### 3.7 Visualization Toggle

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class VisualizationToggle : MonoBehaviour  {  Toggle CurrentToggle;  AudioSource CurrentAudio;  // Start is called before the first frame update  void Start()  {  CurrentToggle = GetComponent<Toggle>();  CurrentAudio = GetComponent<AudioSource>();  }  // Update is called once per frame  void Update()  {    }  public void ActiveOnDelay(float aTime)  {  Invoke("InvokeActive", aTime);  }  void InvokeActive()  {  CurrentToggle.isOn = true;  CurrentAudio.Play();  }  public void ActiveOffDelay(float aTime)  {  Invoke("InvokeNonActive", aTime);  }  void InvokeNonActive()  {  CurrentToggle.isOn = false;  }  } |
| --- |

Script ini digunakan untuk mengendalikan toggle (kotak centang) dalam permainan, khususnya untuk mengaktifkannya atau menonaktifkannya dengan penundaan waktu tertentu. Selain itu, skrip ini juga memainkan suara ketika toggle diaktifkan. Berikut adalah penjelasan komponen-komponen dalam script ini:

* CurrentToggle: Variabel yang digunakan untuk menyimpan komponen Toggle yang terdapat pada objek ini. Komponen Toggle adalah elemen UI yang dapat diaktifkan dan dinonaktifkan oleh pemain.
* CurrentAudio: Variabel yang digunakan untuk menyimpan komponen AudioSource yang terdapat pada objek ini. AudioSource digunakan untuk memainkan suara saat toggle diaktifkan.
* Pada awal permainan, komponen Toggle dan AudioSource yang ada pada objek ini diambil dan disimpan dalam variabel CurrentToggle dan CurrentAudio.
* ActiveOnDelay(float aTime): Method ini digunakan untuk mengaktifkan toggle setelah penundaan waktu tertentu.
* Parameter aTime adalah waktu penundaan dalam detik. Method InvokeActive() akan dijalankan setelah penundaan waktu yang ditentukan, dan pada saat itu, CurrentToggle akan diaktifkan dan CurrentAudio akan memainkan suara.
* InvokeActive(): Method ini digunakan untuk mengaktifkan CurrentToggle dan memainkan suara dengan menggunakan CurrentAudio.
* ActiveOffDelay(float aTime): Method ini digunakan untuk menonaktifkan toggle setelah penundaan waktu tertentu, tanpa memainkan suara.
* Parameter aTime adalah waktu penundaan dalam detik. Method InvokeNonActive() akan dijalankan setelah penundaan waktu yang ditentukan, dan pada saat itu, CurrentToggle akan dinonaktifkan.

### 3.8 Visualization Camera Shake

| using UnityEngine;  using System.Collections;  public class VisualizationShake : MonoBehaviour  {  public float shakeAmount = 5.0f; // Besarnya getaran dalam derajat  public float shakeDuration = 1.0f; // Durasi getaran  private Quaternion originalRotation;  void Start()  {  originalRotation = transform.localRotation;  }  public void ShakeCamera()  {  StartCoroutine(Shake());  }  IEnumerator Shake()  {  float elapsed = 0.0f;  while (elapsed < shakeDuration)  {  float x = Random.Range(-1f, 1f) \* shakeAmount;  float y = Random.Range(-1f, 1f) \* shakeAmount;  float z = Random.Range(-1f, 1f) \* shakeAmount;  transform.localRotation = Quaternion.Euler(x, y, z);  elapsed += Time.deltaTime;  yield return null;  }  transform.localRotation = originalRotation;  }  } |
| --- |

### 3.9 Visualization Headbob

| using UnityEngine;  public class VisualizationBob : MonoBehaviour  {  public float bobbingSpeed = 0.18f;  public float rotationAmount = 5.0f; // Besarnya rotasi  private float timer = 0.0f;  void Update()  {  float waveslice = Mathf.Sin(timer);  timer += bobbingSpeed;  if (timer > Mathf.PI \* 2)  {  timer -= (Mathf.PI \* 2);  }  // Rotasi headbob  float rotationChange = waveslice \* rotationAmount;  transform.localRotation = Quaternion.Euler(rotationChange, 0, 0);  }  } |
| --- |

## 

## 4. Interaction

Event Interaction dalam game berkaitan dengan bagaimana objek atau karakter dalam game berinteraksi satu sama lain atau dengan pemain. Ini bisa mencakup interaksi seperti bertarung, berbicara, atau menggunakan objek. Dalam pemrograman game, ini biasanya melibatkan penggunaan fungsi yang dipanggil saat terjadi interaksi, serta penggunaan variabel untuk menyimpan informasi tentang status objek atau karakter yang terlibat. Misalnya, jika pemain berbicara dengan karakter non-pemain (NPC) dalam game, pemrograman untuk event Interaction ini mungkin melibatkan pemanggilan fungsi yang menampilkan teks dialog dari NPC tersebut, serta pengubahan variabel yang terkait dengan quest atau tugas yang diberikan oleh NPC tersebut. Pemahaman yang baik tentang logika pemrograman dan penggunaan fungsi dan variabel adalah kunci untuk mengimplementasikan event Interaction dengan efektif dalam game. Dengan demikian, pemain dapat merasakan pengalaman yang lebih mendalam dan terlibat secara emosional dalam permainan.

### 4.1 Interaction Mouse

|  |
| --- |

### 4.2 Interaction Keyboard

|  |
| --- |

### 4.3 Interaction Dialog

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class InteractionDialog : MonoBehaviour  {  public enum Character { Player, NPC };  [System.Serializable]  public class DialogData  {  public Character character;  public string name;  [TextArea(5, 6)]  public string dialog;  }  [Header("Player Name")]  public string PlayerName;  [Header("Active Dialog")]  [TextArea(5, 3)]  public string activeDialog;  public bool typewriting;  public float delay;  [Header("Visual Setting")]  public GameObject ParentObject;  public Image dialogPanel;  public Image playerImage;  public Image npcImage;  public Text nameText;  public Text dialogText;  [Header("Dialog Setting")]  public bool AutoStartDialog;  public bool AutoFinishDialog;  public List<DialogData> playerDialogs = new List<DialogData>();  [Header("Event Dialog Setting")]  public UnityEvent StartDialogEvent;  public UnityEvent FinishDialogEvent;  string currentText = "";  private int currentDialogIndex = 0;  private Coroutine DialogCoroutine;  public void StartDialog()  {  currentDialogIndex = 0;  ActivateDialog(playerDialogs[currentDialogIndex]);  StartDialogEvent?.Invoke();  }  void Start()  {  if (AutoStartDialog) StartDialog();  }  void Update()  {  if (Input.GetKeyDown(KeyCode.Mouse0) || Input.GetKeyDown(KeyCode.Return))  {  NextDialog();  }  }  void NextDialog()  {  if (typewriting && DialogCoroutine!= null)  {  StopCoroutine(DialogCoroutine);  }  currentDialogIndex++;  currentText = "";  dialogText.text = "";  if (currentDialogIndex < playerDialogs.Count)  {  ActivateDialog(playerDialogs[currentDialogIndex]);  }  else  {  // Dialogs end  // Perform any necessary actions or close the dialog window  if (AutoFinishDialog)  {  SetChildStatus(ParentObject, false);  }  FinishDialogEvent?.Invoke();  }  }  void ActivateDialog(DialogData dialogData)  {  string player = PlayerName;  SetChildStatus(ParentObject, true);  if (dialogData.character == Character.Player)  {  playerImage.gameObject.SetActive(true);  npcImage.gameObject.SetActive(false);  }  else if (dialogData.character == Character.NPC)  {  playerImage.gameObject.SetActive(false);  npcImage.gameObject.SetActive(true);  }  nameText.text = dialogData.name;  if (dialogData.name == "<name>")  {  nameText.text = player;  }  //-- transfer dialog  activeDialog = dialogData.dialog;  string editedString = activeDialog;  if (activeDialog.Contains("<name>"))  {  editedString = EditString(activeDialog, "<name>", player);  }  if (typewriting)  {  DialogCoroutine = StartCoroutine(TypeText(editedString));  }  else  {  dialogText.text = editedString;  }  }  IEnumerator TypeText(string fullString)  {  for (int i = 0; i < fullString.Length; i++)  {  currentText += fullString[i];  dialogText.text = currentText;  yield return new WaitForSeconds(delay);  }  }  string EditString(string originalString, string targetWord, string replacementWord)  {  string[] words = originalString.Split(' ');  for (int i = 0; i < words.Length; i++)  {  if (words[i] == targetWord)  {  words[i] = replacementWord;  break;  }  }  return string.Join(" ", words);  }  public void SetChildStatus(GameObject parentObject, bool aValue)  {  // Mendapatkan semua komponen Transform dari anak-anak (children) objek  Transform[] childTransforms = parentObject.GetComponentsInChildren<Transform>(true);  // Melakukan iterasi untuk menonaktifkan semua objek anak  foreach (Transform childTransform in childTransforms)  {  // Pastikan objek tersebut bukan parentObject itu sendiri  if (childTransform.gameObject != parentObject)  {  childTransform.gameObject.SetActive(aValue);  }  }  }  } |
| --- |

### 4.4 Interaction Dialog Language

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  public class InteractionDialogLanguage : MonoBehaviour  {  public enum Character { Player, NPC };  [System.Serializable]  public class DialogData  {  public Character character;  public string name;  [TextArea(5, 5)]  public string IndonesianText;  [TextArea(5, 5)]  public string EnglishText;  [TextArea(5, 5)]  public string KoreanText;  [TextArea(5, 5)]  public string ChineseText;  [TextArea(5, 5)]  public string ArabianText;  [TextArea(5, 5)]  public string JapaneseText;  }  [Header("Player Name")]  public string PlayerName;  [Header("Active Dialog")]  [TextArea(5, 3)]  public string activeDialog;  public bool typewriting;  public float delay;  [Header("Visual Setting")]  public GameObject ParentObject;  public Image dialogPanel;  public Image playerImage;  public Image npcImage;  public Text nameText;  public Text dialogText;  [Header("Dialog Setting")]  public bool AutoStartDialog;  public bool AutoFinishDialog;  public List<DialogData> playerDialogs = new List<DialogData>();  [Header("Event Dialog Setting")]  public UnityEvent StartDialogEvent;  public UnityEvent FinishDialogEvent;  string currentText = "";  private int currentDialogIndex = 0;  private Coroutine DialogCoroutine;  public void StartDialog()  {  currentDialogIndex = 0;  ActivateDialog(playerDialogs[currentDialogIndex]);  StartDialogEvent?.Invoke();  }  void Start()  {  if (AutoStartDialog) StartDialog();  }  void Update()  {  if (Input.GetKeyDown(KeyCode.Mouse0) || Input.GetKeyDown(KeyCode.Return))  {  NextDialog();  }  }  void NextDialog()  {  if (typewriting && DialogCoroutine != null)  {  StopCoroutine(DialogCoroutine);  }  currentDialogIndex++;  currentText = "";  dialogText.text = "";  if (currentDialogIndex < playerDialogs.Count)  {  ActivateDialog(playerDialogs[currentDialogIndex]);  }  else  {  // Dialogs end  // Perform any necessary actions or close the dialog window  if (AutoFinishDialog)  {  SetChildStatus(ParentObject, false);  }  FinishDialogEvent?.Invoke();  }  }  void ActivateDialog(DialogData dialogData)  {  string player = PlayerName;  SetChildStatus(ParentObject, true);  if (dialogData.character == Character.Player)  {  playerImage.gameObject.SetActive(true);  npcImage.gameObject.SetActive(false);  }  else if (dialogData.character == Character.NPC)  {  playerImage.gameObject.SetActive(false);  npcImage.gameObject.SetActive(true);  }  nameText.text = dialogData.name;  if (dialogData.name == "<name>")  {  nameText.text = player;  }  //-- transfer dialog  if (PlayerPrefs.GetString("LANGUAGE") == "INDONESIAN")  {  activeDialog = dialogData.IndonesianText;  }  else if (PlayerPrefs.GetString("LANGUAGE") == "ENGLISH")  {  activeDialog = dialogData.EnglishText;  }  else if (PlayerPrefs.GetString("LANGUAGE") == "CHINESE")  {  activeDialog = dialogData.ChineseText;  }  else if (PlayerPrefs.GetString("LANGUAGE") == "KOREAN")  {  activeDialog = dialogData.KoreanText;  }  else if (PlayerPrefs.GetString("LANGUAGE") == "ARABIAN")  {  activeDialog = dialogData.ArabianText;  }  else if (PlayerPrefs.GetString("LANGUAGE") == "JAPANESE")  {  activeDialog = dialogData.JapaneseText;  } else  {  activeDialog = dialogData.IndonesianText;  }  string editedString = activeDialog;  if (activeDialog.Contains("<name>"))  {  editedString = EditString(activeDialog, "<name>", player);  }  if (typewriting)  {  DialogCoroutine = StartCoroutine(TypeText(editedString));  }  else  {  dialogText.text = editedString;  }  }  IEnumerator TypeText(string fullString)  {  for (int i = 0; i < fullString.Length; i++)  {  currentText += fullString[i];  dialogText.text = currentText;  yield return new WaitForSeconds(delay);  }  }  string EditString(string originalString, string targetWord, string replacementWord)  {  string[] words = originalString.Split(' ');  for (int i = 0; i < words.Length; i++)  {  if (words[i] == targetWord)  {  words[i] = replacementWord;  break;  }  }  return string.Join(" ", words);  }  public void SetChildStatus(GameObject parentObject, bool aValue)  {  // Mendapatkan semua komponen Transform dari anak-anak (children) objek  Transform[] childTransforms = parentObject.GetComponentsInChildren<Transform>(true);  // Melakukan iterasi untuk menonaktifkan semua objek anak  foreach (Transform childTransform in childTransforms)  {  // Pastikan objek tersebut bukan parentObject itu sendiri  if (childTransform.gameObject != parentObject)  {  childTransform.gameObject.SetActive(aValue);  }  }  }  } |
| --- |

## 5. Transformation

Event Transformation dalam game terkait dengan perubahan posisi, orientasi, atau ukuran dari objek atau karakter dalam game. Ini bisa mencakup transformasi seperti translasi (perpindahan), rotasi (putaran), dan skala (perubahan ukuran). Dalam pemrograman game, ini biasanya melibatkan penggunaan matriks transformasi dan operasi vektor. Misalnya, jika pemain menggerakkan karakter ke kiri, pemrograman untuk event Transformation ini mungkin melibatkan pengubahan koordinat posisi karakter dalam matriks transformasi. Framework atau engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola transformasi objek. Pemrograman untuk event Transformation melibatkan pemahaman yang baik tentang matematika dan geometri, serta kemampuan untuk bekerja dengan matriks dan vektor. Dengan mengelola transformasi dengan efektif, programmer dapat menciptakan gameplay yang dinamis dan responsif.

### 5.1 Transformation Vector

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class TransformationPosition : MonoBehaviour  {  [Header("Main Settings")]  public Vector3 ParameterVector;  public bool UpdatePosition;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public void InvokeTransformPosition()  {  transform.position = ParameterVector;  }  public void InvokeAddTransformPosition()  {  transform.position += ParameterVector;  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  if (UpdatePosition)  {  InvokeAddTransformPosition();  }  UpdateEvents?.Invoke();  }  } |
| --- |

Skrip TransformationPosition ini digunakan untuk mengatur transformasi posisi dari objek yang memiliki komponen Transform di Unity. Mari kita jelaskan bagian-bagian utama dari skrip ini:

* ParameterVector: Ini adalah vektor tiga dimensi (Vector3) yang akan digunakan untuk mengatur atau menambahkan posisi objek.
* UpdatePosition: Ini adalah boolean yang menentukan apakah posisi objek akan diperbarui setiap frame. Jika diaktifkan (true), maka posisi objek akan diperbarui setiap frame sesuai dengan vektor ParameterVector.
* StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif atau muncul di dalam permainan (saat metode Start dipanggil).
* UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).
* Metode InvokeTransformPosition(): Metode ini mengatur posisi objek yang terkait dengan nilai dari ParameterVector. Ini akan mengatur posisi objek secara langsung ke nilai yang diberikan dalam ParameterVector.
* Metode InvokeAddTransformPosition(): Metode ini menambahkan posisi objek yang terkait dengan nilai dari ParameterVector. Ini akan menambahkan posisi objek saat ini dengan nilai dari ParameterVector.
* Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.
* Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Jika UpdatePosition diaktifkan (true), maka metode InvokeAddTransformPosition() akan dipanggil setiap frame untuk memperbarui posisi objek. Selain itu, UpdateEvents diaktifkan jika ada.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class TransformationRotation : MonoBehaviour  {  [Header("Main Settings")]  public Vector3 ParameterVector;  public bool UpdateRotation;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public void InvokeTransformPosition()  {  transform.eulerAngles = ParameterVector;  }  public void InvokeAddTransformRotation()  {  transform.Rotate(ParameterVector);  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  if (UpdateRotation)  {  InvokeAddTransformRotation();  }  UpdateEvents?.Invoke();  }  } |
| --- |

Skrip TransformationRotation ini digunakan untuk mengatur rotasi dari objek yang memiliki komponen Transform di Unity. Mari kita jelaskan bagian-bagian utama dari skrip ini:

* ParameterVector: Ini adalah vektor tiga dimensi (Vector3) yang akan digunakan untuk mengatur atau menambahkan rotasi objek.
* UpdateRotation: Ini adalah boolean yang menentukan apakah rotasi objek akan diperbarui setiap frame. Jika diaktifkan (true), maka rotasi objek akan diperbarui setiap frame sesuai dengan vektor ParameterVector.
* StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif atau muncul di dalam permainan (saat metode Start dipanggil).
* UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).
* Metode InvokeTransformPosition(): Metode ini mengatur rotasi objek yang terkait dengan nilai dari ParameterVector. Ini akan mengatur rotasi objek secara langsung ke nilai yang diberikan dalam ParameterVector.
* Metode InvokeAddTransformRotation(): Metode ini menambahkan rotasi objek yang terkait dengan nilai dari ParameterVector. Ini akan menambahkan rotasi objek saat ini dengan nilai dari ParameterVector.
* Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.
* Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Jika UpdateRotation diaktifkan (true), maka metode InvokeAddTransformRotation() akan dipanggil setiap frame untuk memperbarui rotasi objek. Selain itu, UpdateEvents diaktifkan jika ada.

Dengan skrip ini, Anda dapat mengendalikan perubahan rotasi objek dengan mudah menggunakan vektor ParameterVector. Juga, Anda dapat menggunakan event StartEvents dan UpdateEvents untuk menambahkan logika tambahan yang berkaitan dengan perubahan rotasi ini atau berbagai perubahan lain yang Anda perlukan dalam permainan Anda.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class TransformationScale : MonoBehaviour  {  [Header("Main Settings")]  public Vector3 ParameterVector;  public bool UpdateScale;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  public void InvokeTransformScale()  {  transform.localScale = ParameterVector;  }  public void InvokeAddTransformScale()  {  transform.localScale += ParameterVector;  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  if (UpdateScale)  {  InvokeAddTransformScale();  }  UpdateEvents?.Invoke();  }  } |
| --- |

Skrip TransformationScale digunakan untuk mengatur skala (scale) dari objek yang memiliki komponen Transform di Unity. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

* ParameterVector: Ini adalah vektor tiga dimensi (Vector3) yang digunakan untuk mengatur atau menambahkan skala objek.
* UpdateScale: Ini adalah boolean yang menentukan apakah skala objek akan diperbarui setiap frame. Jika diaktifkan (true), maka skala objek akan diperbarui setiap frame sesuai dengan vektor ParameterVector.
* StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif atau muncul di dalam permainan (saat metode Start dipanggil).
* UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).
* Metode InvokeTransformScale(): Metode ini mengatur skala objek yang terkait dengan nilai dari ParameterVector. Ini akan mengatur skala objek secara langsung ke nilai yang diberikan dalam ParameterVector.
* Metode InvokeAddTransformScale(): Metode ini menambahkan skala objek yang terkait dengan nilai dari ParameterVector. Ini akan menambahkan skala objek saat ini dengan nilai dari ParameterVector.
* Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.
* Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Jika UpdateScale diaktifkan (true), maka metode InvokeAddTransformScale() akan dipanggil setiap frame untuk memperbarui skala objek. Selain itu, UpdateEvents diaktifkan jika ada.

Dengan skrip ini, Anda dapat mengendalikan perubahan skala objek dengan mudah menggunakan vektor ParameterVector. Anda juga dapat menggunakan event StartEvents dan UpdateEvents untuk menambahkan logika tambahan yang berkaitan dengan perubahan skala ini atau berbagai perubahan lain yang Anda perlukan dalam permainan Anda.

### 

### 5.2 Transformation Translate

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class TransformationTranslate : MonoBehaviour  {  public enum Direction  {  Up,  Down,  Left,  Right,  Forward,  Back  }  [Header("Main Settings")]  public Direction moveDirection; // Enum untuk memilih arah pergerakan  public float moveSpeed = 1.0f; // Kecepatan pergerakan  public bool updatePosition;  [Header("Event Settings")]  public UnityEvent startEvents;  public UnityEvent updateEvents;  // Start is called before the first frame update  void Start()  {  startEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  if (updatePosition)  {  MoveInDirection();  }  updateEvents?.Invoke();  }  private void MoveInDirection()  {  Vector3 moveVector = Vector3.zero;  switch (moveDirection)  {  case Direction.Up:  moveVector = Vector3.up;  break;  case Direction.Down:  moveVector = Vector3.down;  break;  case Direction.Left:  moveVector = Vector3.left;  break;  case Direction.Right:  moveVector = Vector3.right;  break;  case Direction.Forward:  moveVector = Vector3.forward;  break;  case Direction.Back:  moveVector = Vector3.back;  break;  }  // Menggunakan Translate untuk memindahkan objek sesuai dengan arah dan kecepatan yang dipilih  transform.Translate(moveVector \* moveSpeed \* Time.deltaTime);  }  } |
| --- |

Skrip TransformationTranslate digunakan untuk mengendalikan pergerakan objek dalam berbagai arah (atas, bawah, kiri, kanan, maju, mundur) menggunakan komponen Transform di Unity. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

* moveDirection: Ini adalah variabel enum Direction yang digunakan untuk memilih arah pergerakan objek. Anda dapat memilih dari enam arah yang tersedia: atas (Up), bawah (Down), kiri (Left), kanan (Right), maju (Forward), dan mundur (Back).
* moveSpeed: Ini adalah kecepatan pergerakan objek. Anda dapat mengatur kecepatan seberapa cepat objek bergerak dalam arah yang dipilih.
* updatePosition: Ini adalah boolean yang menentukan apakah pergerakan objek akan diperbarui setiap frame. Jika diaktifkan (true), maka objek akan terus bergerak sesuai dengan arah dan kecepatan yang ditentukan.
* startEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif atau muncul di dalam permainan (saat metode Start dipanggil).
* updateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).
* Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, startEvents diaktifkan jika ada.
* Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Jika updatePosition diaktifkan (true), maka metode MoveInDirection() akan dipanggil setiap frame untuk memperbarui pergerakan objek. Selain itu, updateEvents diaktifkan jika ada.
* Metode MoveInDirection(): Metode ini menghitung vektor pergerakan (moveVector) berdasarkan arah yang telah dipilih (moveDirection) menggunakan struktur kontrol switch-case. Setelah vektor pergerakan dihitung, metode ini menggunakan metode Translate pada komponen Transform untuk memindahkan objek sesuai dengan arah dan kecepatan yang telah ditentukan. Ini memastikan objek bergerak sesuai dengan arah yang dipilih.

Dengan skrip ini, Anda dapat mengendalikan pergerakan objek dalam berbagai arah dengan mudah. Anda dapat mengatur arah, kecepatan, dan aktivasi pergerakan sesuai kebutuhan dalam permainan Anda.

### 

### 5.3 Transformation PingPong

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class TransformationPingPong : MonoBehaviour  {  public enum Axis  {  AxisX,  AxisY,  AxisZ  }  [Header("Main Settings")]  public Axis moveAxis; // Enum untuk memilih sumbu pergerakan  public float moveSpeed = 1.0f; // Kecepatan pergerakan  public bool pingPong = true; // Mode pingpong  public float pingPongSpeed = 1.0f; // Kecepatan pingpong  public bool updatePosition;  [Header("Event Settings")]  public UnityEvent startEvents;  public UnityEvent updateEvents;  private bool isMovingForward = true;  private float startPosition;  // Start is called before the first frame update  void Start()  {  startEvents?.Invoke();  startPosition = GetAxisValue();  }  // Update is called once per frame  void Update()  {  if (updatePosition)  {  MoveInDirection();  }  updateEvents?.Invoke();  }  private void MoveInDirection()  {  float axisValue = GetAxisValue();  if (pingPong)  {  if (isMovingForward)  {  axisValue += pingPongSpeed \* Time.deltaTime;  if (axisValue >= startPosition + 1.0f)  {  isMovingForward = false;  }  }  else  {  axisValue -= pingPongSpeed \* Time.deltaTime;  if (axisValue <= startPosition - 1.0f)  {  isMovingForward = true;  }  }  }  else  {  axisValue += moveSpeed \* Time.deltaTime;  }  SetAxisValue(axisValue);  }  private float GetAxisValue()  {  Vector3 position = transform.position;  switch (moveAxis)  {  case Axis.AxisX:  return position.x;  case Axis.AxisY:  return position.y;  case Axis.AxisZ:  return position.z;  default:  return 0.0f;  }  }  private void SetAxisValue(float value)  {  Vector3 position = transform.position;  switch (moveAxis)  {  case Axis.AxisX:  position.x = value;  break;  case Axis.AxisY:  position.y = value;  break;  case Axis.AxisZ:  position.z = value;  break;  }  transform.position = position;  }  } |
| --- |

Skrip TransformationPingPong digunakan untuk mengendalikan pergerakan objek di sepanjang sumbu tertentu (X, Y, atau Z) dalam permainan Unity. Berikut penjelasan komponen-komponen utama dari skrip ini:

* moveAxis: Ini adalah variabel enum Axis yang digunakan untuk memilih sumbu pergerakan objek. Anda dapat memilih dari tiga sumbu yang tersedia: X, Y, atau Z.
* moveSpeed: Ini adalah kecepatan pergerakan objek. Anda dapat mengatur seberapa cepat objek bergerak di sepanjang sumbu yang dipilih.
* pingPong: Ini adalah boolean yang menentukan apakah pergerakan objek akan berada dalam mode "pingpong" atau tidak. Jika diaktifkan (true), objek akan bergerak bolak-balik di antara dua titik sepanjang sumbu yang dipilih.
* pingPongSpeed: Ini adalah kecepatan pergerakan dalam mode "pingpong". Kecepatan ini hanya berlaku jika mode "pingpong" diaktifkan.
* updatePosition: Ini adalah boolean yang menentukan apakah pergerakan objek akan diperbarui setiap frame. Jika diaktifkan (true), maka objek akan terus bergerak sesuai dengan konfigurasi yang ditentukan.
* startEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif atau muncul dalam permainan (saat metode Start dipanggil).
* updateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).
* isMovingForward: Ini adalah variabel boolean yang menunjukkan apakah objek sedang bergerak ke depan atau ke belakang dalam mode "pingpong".
* startPosition: Ini adalah posisi awal objek pada sumbu yang dipilih. Posisi ini digunakan untuk mengontrol pergerakan dalam mode "pingpong".
* Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, startEvents diaktifkan jika ada, dan posisi awal objek pada sumbu yang dipilih (startPosition) dihitung.
* Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Jika updatePosition diaktifkan (true), maka metode MoveInDirection() akan dipanggil setiap frame untuk memperbarui pergerakan objek. Selain itu, updateEvents diaktifkan jika ada.
* Metode MoveInDirection(): Metode ini menghitung pergerakan objek sesuai dengan konfigurasi yang telah ditentukan. Jika mode "pingpong" diaktifkan, objek akan bergerak bolak-balik antara dua titik sepanjang sumbu yang dipilih. Jika tidak, objek akan bergerak ke depan. Metode ini memastikan bahwa objek bergerak sesuai dengan kecepatan dan mode yang dipilih.
* Metode GetAxisValue(): Metode ini mengembalikan nilai posisi objek pada sumbu yang dipilih (X, Y, atau Z) berdasarkan konfigurasi moveAxis.
* Metode SetAxisValue(): Metode ini mengatur nilai posisi objek pada sumbu yang dipilih (X, Y, atau Z) berdasarkan konfigurasi moveAxis.

Dengan skrip ini, Anda dapat mengendalikan pergerakan objek di sepanjang sumbu tertentu, termasuk pergerakan bolak-balik dalam mode "pingpong" jika diperlukan. Anda dapat mengatur sumbu, kecepatan, mode, dan aktivasi pergerakan sesuai dengan kebutuhan dalam permainan Anda.

### 5.4 Transformation Waypoint

| using UnityEngine;  public class TransformationWaypoint : MonoBehaviour  {  [Header("Main Setting")]  public float speed = 5.0f; // Vehicle speed  public float rotationSpeed = 45.0f; // Vehicle rotation speed  public float distanceLimit = 0.1f;  public Transform[] waypoints; // Array of waypoints to follow  [Header("Current Value")]  public int currentWaypointIndex = 0; // Current waypoint index  public float currentDistance = 0;  private void Start()  {  // After initialization, start moving the vehicle towards the first waypoint  if (waypoints.Length > 0)  {  MoveToWaypoint(currentWaypointIndex);  }  }  private void Update()  {  // Check if the vehicle has reached the current waypoint  MoveToWaypoint(currentWaypointIndex);  currentDistance = Vector3.Distance(transform.position, waypoints[currentWaypointIndex].position);  if (currentDistance < distanceLimit)  {  // If so, move to the next waypoint  currentWaypointIndex++;  if (currentWaypointIndex >= waypoints.Length)  {  // If reached the end of waypoints, return to the first waypoint  currentWaypointIndex = 0;  }  // Move to the next waypoint  MoveToWaypoint(currentWaypointIndex);  }  }  private void MoveToWaypoint(int index)  {  // Rotate the vehicle towards the waypoint  Vector3 direction = waypoints[index].position - transform.position;  Quaternion targetRotation = Quaternion.LookRotation(direction);  transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation, rotationSpeed \* Time.deltaTime);  // Calculate the movement direction based on the vehicle's forward direction  Vector3 moveDirection = transform.forward;  // Move the vehicle towards the waypoint  transform.position += moveDirection \* speed \* Time.deltaTime;  }  } |
| --- |

Skrip TransformationWaypoint digunakan untuk membuat simulasi trafik dan mengendalikan pergerakan objek di sepanjang sumbu (X, Y, Z) dalam permainan Unity. Berikut penjelasan komponen-komponen utama dari skrip ini:

* speed: Variabel float yang menentukan kecepatan kendaraan dalam pergerakan menuju waypoint.
* rotationSpeed: Variabel float yang menentukan kecepatan rotasi kendaraan saat bergerak menuju waypoint.
* waypoints: Array dari objek Transform yang akan digunakan sebagai waypoint yang akan diikuti oleh kendaraan.
* currentWaypointIndex: Variabel integer yang digunakan untuk melacak indeks waypoint saat ini yang harus diikuti oleh kendaraan.
* Pada saat inisialisasi objek, metode ini memeriksa apakah terdapat waypoint yang ada dalam array waypoints. Jika ada, metode MoveToWaypoint dipanggil untuk mulai pergerakan kendaraan menuju waypoint pertama.
* Metode Update(): Metode ini dijalankan setiap frame. Pertama, metode ini memanggil MoveToWaypoint dengan parameter currentWaypointIndex. Ini memastikan bahwa kendaraan selalu bergerak menuju waypoint saat ini. Kemudian, metode ini memeriksa apakah kendaraan telah mencapai waypoint saat ini. Ini dilakukan dengan memeriksa jarak antara posisi kendaraan dan posisi waypoint saat ini. Jika jaraknya sangat kecil (kurang dari 0.1 unit), kendaraan dianggap telah mencapai waypoint tersebut. Jika kendaraan telah mencapai waypoint saat ini, maka akan beralih ke waypoint berikutnya dengan menambahkan 1 ke currentWaypointIndex. Jika sudah mencapai akhir dari array waypoints, maka akan kembali ke waypoint pertama. Setelah menentukan waypoint yang baru, kendaraan akan diarahkan ke waypoint tersebut dan pergerakan menuju waypoint tersebut akan dilakukan dengan memanggil kembali MoveToWaypoint.
* Metode MoveToWaypoint(int index): Metode ini menggerakkan kendaraan menuju waypoint dengan indeks yang diberikan. Pertama, metode ini menghitung vektor arah (direction) antara posisi kendaraan dan posisi waypoint yang dituju. Kemudian, kendaraan akan dirotasi ke arah waypoint dengan menggunakan Quaternion.LookRotation. Setelah itu, pergerakan kendaraan dihitung berdasarkan arah kendaraan ke depan (transform.forward) dan kecepatan (speed). Ini memastikan kendaraan bergerak menuju waypoint. Akhirnya, posisi kendaraan diperbarui sesuai dengan pergerakan yang dihitung menggunakan transform.position.

### 

### 5.5 Transformation Direction

| using UnityEngine;  using UnityEngine.UI;  public class TransformationDirection : MonoBehaviour  {  [Header("Movement Settings")]  public float kecepatanGerak = 5.0f; // Kecepatan gerakan GameObject  public bool TouchJoystickInput;  float inputHorizontal;  float inputVertical;  void Update()  {  if (!TouchJoystickInput)  {  inputHorizontal = Input.GetAxis("Horizontal");  inputVertical = Input.GetAxis("Vertical");  Vector3 gerakan = new Vector3(inputHorizontal, 0.0f, inputVertical) \* kecepatanGerak \* Time.deltaTime;  // Menggerakkan GameObject berdasarkan input  transform.Translate(gerakan);  } else  {  Vector3 gerakan = new Vector3(inputHorizontal, 0.0f, inputVertical) \* kecepatanGerak \* Time.deltaTime;  // Menggerakkan GameObject berdasarkan input  transform.Translate(gerakan);  }  }  public void SetInputX(Text value)  {  inputHorizontal = float.Parse(value.text);  }  public void SetInputY(Text value)  {  inputVertical = float.Parse(value.text);  }  } |
| --- |

| using UnityEngine;  using UnityEngine.UI;  public class TransformationDirectionXYZ : MonoBehaviour  {  [Header("Movement Settings")]  public float kecepatanGerak = 5.0f; // Kecepatan gerakan GameObject  public bool TouchJoystickInput;  float moveX = 0;  float moveY = 0;  float moveZ = 0;  void Update()  {  if (!TouchJoystickInput)  {  Vector3 gerakan = new Vector3(moveX, moveY, moveZ) \* kecepatanGerak \* Time.deltaTime;  // Menggerakkan GameObject berdasarkan input  transform.Translate(gerakan);  }  else  {  Vector3 gerakan = new Vector3(moveX, moveY, moveZ) \* kecepatanGerak \* Time.deltaTime;  // Menggerakkan GameObject berdasarkan input  transform.Translate(gerakan);  }  }  public void SetInputX(Text value)  {  moveX = Input.GetAxis("Horizontal");  moveX = float.Parse(value.text);  }  public void SetInputY(Text value)  {  moveY = Input.GetAxis("Vertical");  moveY = float.Parse(value.text);  }  public void SetInputZ(Text value)  {  moveZ = Input.GetAxis("Vertical");  moveZ = float.Parse(value.text);  }  } |
| --- |

Script di atas adalah sebuah komponen dalam Unity yang mengatur transformasi (pergerakan) suatu GameObject. Dalam script ini, ada beberapa variabel yang dapat disesuaikan, seperti kecepatan gerakan dan opsi untuk menggunakan input dari joystick sentuh. Pada metode Update(), jika opsi "TouchJoystickInput" tidak diaktifkan, maka script akan membaca input horizontal dan vertical dari keyboard atau kontroler, menghitung vektor gerakan, dan menggerakkan GameObject sesuai dengan input tersebut. Jika "TouchJoystickInput" diaktifkan, maka input horizontal dan vertical akan diambil dari fungsi SetInputX() dan SetInputY() yang dapat dipanggil dari elemen UI, seperti teks. Dengan ini, script ini memungkinkan pergerakan GameObject berdasarkan input yang dapat disesuaikan baik dari keyboard atau kontroler, maupun melalui antarmuka pengguna.

| using UnityEngine;  using UnityEngine.UI;  public class TransformationDirectRotation : MonoBehaviour  {  [Header("Main Settings")]  public float moveSpeed = 5f;  public float rotationSpeed = 100f;  public bool TouchJoystickInput;  float moveX = 0;  float moveZ = 0;  private Rigidbody rb;  void Start()  {  rb = GetComponent<Rigidbody>();  }  void Update()  {  if (!TouchJoystickInput)  {  // Mendapatkan input horizontal dan vertical dari pemain  moveX = Input.GetAxis("Horizontal");  moveZ = Input.GetAxis("Vertical");  }  // Membuat vektor pergerakan berdasarkan input pemain  Vector3 movement = new Vector3(moveX, 0f, moveZ);  // Menggerakkan karakter berdasarkan vektor pergerakan  rb.velocity = movement.normalized \* moveSpeed;  // Rotasi karakter sesuai arah pergerakan  if (movement != Vector3.zero)  {  Quaternion toRotation = Quaternion.LookRotation(movement, Vector3.up);  transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed \* Time.deltaTime);  }  }  public void SetInputX(Text value)  {  moveX = float.Parse(value.text);  }  public void SetInputY(Text value)  {  moveZ = float.Parse(value.text);  }  } |
| --- |

| using UnityEngine;  using UnityEngine.UI;  [RequireComponent(typeof(Rigidbody))]  public class TransformationDirectRotationXYZ : MonoBehaviour  {  [Header("Main Settings")]  public float moveSpeed = 5f;  public float rotationSpeed = 100f;  public bool TouchJoystickInput;  float moveX = 0;  float moveY = 0;  float moveZ = 0;  private Rigidbody rb;  void Start()  {  rb = GetComponent<Rigidbody>();  }  void Update()  {  if (!TouchJoystickInput)  {  // Mendapatkan input horizontal dan vertical dari pemain  }  // Membuat vektor pergerakan berdasarkan input pemain  Vector3 movement = new Vector3(moveX, moveY, moveZ);  // Menggerakkan karakter berdasarkan vektor pergerakan  rb.velocity = movement.normalized \* moveSpeed;  // Rotasi karakter sesuai arah pergerakan  if (movement != Vector3.zero)  {  Quaternion toRotation = Quaternion.LookRotation(movement, Vector3.up);  transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed \* Time.deltaTime);  }  }  public void SetInputX(Text value)  {  moveX = Input.GetAxis("Horizontal");  moveX = float.Parse(value.text);  }  public void SetInputY(Text value)  {  moveY = Input.GetAxis("Vertical");  moveY = float.Parse(value.text);  }  public void SetInputZ(Text value)  {  moveZ = Input.GetAxis("Vertical");  moveZ = float.Parse(value.text);  }  } |
| --- |

Script di atas adalah komponen dalam Unity yang mengatur transformasi dan rotasi suatu GameObject. Terdapat beberapa pengaturan utama, termasuk kecepatan pergerakan (moveSpeed), kecepatan rotasi (rotationSpeed), dan opsi untuk menggunakan input dari joystick sentuh. Pada saat Update(), jika opsi "TouchJoystickInput" tidak diaktifkan, script akan membaca input horizontal dan vertical dari keyboard atau kontroler. Selanjutnya, script menciptakan sebuah vektor pergerakan berdasarkan input pemain dan menggerakkan karakter dengan kecepatan yang diinginkan.

Selain itu, karakter juga akan dirotasi sesuai dengan arah pergerakan menggunakan metode Quaternion.LookRotation(). Jika karakter sedang bergerak (movement tidak nol), maka script akan menghitung rotasi yang diinginkan (toRotation) berdasarkan vektor pergerakan, lalu secara perlahan akan mengubah rotasi karakter menuju toRotation menggunakan Quaternion.RotateTowards(). Ini membuat karakter menghadap arah pergerakan, memberikan pengalaman yang lebih realistis. Terakhir, terdapat fungsi SetInputX() dan SetInputY() yang memungkinkan input horizontal dan vertical untuk diatur melalui elemen UI, seperti teks. Dengan ini, script ini memungkinkan pergerakan dan rotasi karakter berdasarkan input pemain dan dapat diintegrasikan dengan antarmuka pengguna.

## 6. Animation

Event Animation dalam game berkaitan dengan gerakan dan perubahan visual karakter atau objek dalam game. Hal ini meliputi animasi seperti berjalan, berlari, melompat, atau menyerang. Dalam pemrograman game, ini biasanya melibatkan penggunaan sprite, frame, dan tweens. Misalnya, saat pemain menggerakkan karakter untuk berlari, pemrograman untuk event Animation ini melibatkan pemanggilan serangkaian frame atau sprite yang menggambarkan gerakan berlari karakter. Pemrograman untuk event Animation juga melibatkan penggunaan timer atau penghitung waktu untuk mengendalikan kecepatan dan durasi animasi. Engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola animasi. Pemahaman yang baik tentang konsep animasi dan penggunaan timer adalah kunci untuk mengimplementasikan event Animation dengan efektif dalam game. Dengan mengelola animasi dengan efektif, programmer dapat menciptakan pengalaman yang lebih realistis dan menarik bagi pemain.

### 6.1 Animation String

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class AnimationString : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  Animator animator; // Komponen Animator  // Start is called before the first frame update  void Start()  {  animator = GetComponent<Animator>();  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Method untuk memainkan Animator Controller berdasarkan string  public void Play(string animationName)  {  if (animator != null && !string.IsNullOrEmpty(animationName))  {  animator.Play(animationName);  }  }  } |
| --- |

Script ini memungkinkan pengembang untuk menentukan event-event yang akan dipicu pada saat objek ini diaktifkan dan selama setiap frame. Selain itu, itu juga menyediakan metode Play yang dapat digunakan untuk memainkan animasi berdasarkan nama animasi yang diberikan.

* public class AnimationString : MonoBehaviour: Ini adalah definisi kelas yang disebut AnimationString. Kelas ini merupakan turunan dari MonoBehaviour, yang berarti itu adalah komponen yang dapat digunakan dalam game Unity.
* public UnityEvent StartEvents; dan public UnityEvent UpdateEvents;: Ini adalah dua variabel yang bertipe UnityEvent. Mereka digunakan untuk menyimpan daftar event yang akan dipicu pada saat metode Start dan Update dijalankan. UnityEvent adalah tipe data yang digunakan untuk membuat event yang dapat dihubungkan dan diatur dari editor Unity.
* Animator animator;: Ini adalah variabel yang digunakan untuk menyimpan referensi ke komponen Animator yang terpasang pada objek ini.
* void Start(): Ini adalah metode yang dipanggil pada saat objek pertama kali aktif dalam permainan. Di dalam metode ini, pertama-tama kita menginisialisasi variabel animator dengan komponen Animator yang terpasang pada objek ini menggunakan GetComponent<Animator>(). Selanjutnya, kita memicu event StartEvents jika event tersebut telah ditetapkan.
* void Update(): Ini adalah metode yang dipanggil sekali setiap frame selama objek ini aktif dalam permainan. Di dalam metode ini, kita memicu event UpdateEvents jika event tersebut telah ditetapkan.
* public void Play(string animationName): Ini adalah metode publik yang dapat dipanggil dari luar kelas ini untuk memainkan animasi pada komponen Animator. Metode ini menerima satu argumen, yaitu animationName, yang merupakan nama animasi yang ingin diputar. Metode ini memeriksa apakah komponen Animator telah diinisialisasi dan apakah animationName tidak kosong atau null. Jika keduanya memenuhi syarat, maka metode Play dari komponen Animator akan dipanggil untuk memulai animasi dengan nama yang sesuai.

### 6.2 Animation Parameter

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class AnimationParameter : MonoBehaviour  {  [Header("Parameter Settings")]  public string ParameterName;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  Animator animator; // Komponen Animator  // Start is called before the first frame update  void Start()  {  animator = GetComponent<Animator>();  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Method untuk memainkan Animator Controller berdasarkan string  public void PlayBoolean(bool aValue)  {  if (animator != null && !string.IsNullOrEmpty(ParameterName))  {  animator.SetBool(ParameterName, aValue);  }  }  public void PlayInteger(int aValue)  {  if (animator != null && !string.IsNullOrEmpty(ParameterName))  {  animator.SetInteger(ParameterName, aValue);  }  }  public void PlayFloat(float aValue)  {  if (animator != null && !string.IsNullOrEmpty(ParameterName))  {  animator.SetFloat(ParameterName, aValue);  }  }  public void PlayTrigger()  {  if (animator != null && !string.IsNullOrEmpty(ParameterName))  {  animator.SetTrigger(ParameterName);  }  }  } |
| --- |

Script ini memungkinkan pengembang untuk mengendalikan parameter-parameter pada komponen Animator, seperti boolean, integer, dan float, serta mengaktifkan trigger berdasarkan nama parameter yang ditentukan dari luar kelas ini.

* public class AnimationParameter : MonoBehaviour: Ini adalah definisi kelas yang disebut AnimationParameter. Kelas ini merupakan turunan dari MonoBehaviour, yang berarti itu adalah komponen yang dapat digunakan dalam game Unity.
* public string ParameterName;: Ini adalah variabel publik yang digunakan untuk menyimpan nama parameter yang ingin diubah pada komponen Animator.
* public UnityEvent StartEvents; dan public UnityEvent UpdateEvents;: Ini adalah dua variabel yang bertipe UnityEvent. Mereka digunakan untuk menyimpan daftar event yang akan dipicu pada saat metode Start dan Update dijalankan. UnityEvent adalah tipe data yang digunakan untuk membuat event yang dapat dihubungkan dan diatur dari editor Unity.
* Animator animator;: Ini adalah variabel yang digunakan untuk menyimpan referensi ke komponen Animator yang terpasang pada objek ini.
* void Start(): Ini adalah metode yang dipanggil pada saat objek pertama kali aktif dalam permainan. Di dalam metode ini, pertama-tama kita menginisialisasi variabel animator dengan komponen Animator yang terpasang pada objek ini menggunakan GetComponent<Animator>(). Selanjutnya, kita memicu event StartEvents jika event tersebut telah ditetapkan.
* void Update(): Ini adalah metode yang dipanggil sekali setiap frame selama objek ini aktif dalam permainan. Di dalam metode ini, kita memicu event UpdateEvents jika event tersebut telah ditetapkan.
* public void PlayBoolean(bool aValue): Ini adalah metode publik yang dapat dipanggil dari luar kelas ini untuk mengubah parameter boolean pada komponen Animator. Metode ini menerima satu argumen, yaitu aValue, yang merupakan nilai boolean yang akan diberikan ke parameter Animator. Metode ini memeriksa apakah komponen Animator telah diinisialisasi dan apakah ParameterName tidak kosong atau null. Jika keduanya memenuhi syarat, maka metode SetBool dari komponen Animator akan digunakan untuk mengubah nilai parameter boolean yang sesuai.
* public void PlayInteger(int aValue): Ini adalah metode publik yang dapat dipanggil dari luar kelas ini untuk mengubah parameter integer pada komponen Animator. Metode ini menerima satu argumen, yaitu aValue, yang merupakan nilai integer yang akan diberikan ke parameter Animator. Metode ini memeriksa apakah komponen Animator telah diinisialisasi dan apakah ParameterName tidak kosong atau null. Jika keduanya memenuhi syarat, maka metode SetInteger dari komponen Animator akan digunakan untuk mengubah nilai parameter integer yang sesuai.
* public void PlayFloat(float aValue): Ini adalah metode publik yang dapat dipanggil dari luar kelas ini untuk mengubah parameter float pada komponen Animator. Metode ini menerima satu argumen, yaitu aValue, yang merupakan nilai float yang akan diberikan ke parameter Animator. Metode ini memeriksa apakah komponen Animator telah diinisialisasi dan apakah ParameterName tidak kosong atau null. Jika keduanya memenuhi syarat, maka metode SetFloat dari komponen Animator akan digunakan untuk mengubah nilai parameter float yang sesuai.
* public void PlayTrigger(): Ini adalah metode publik yang dapat dipanggil dari luar kelas ini untuk mengaktifkan trigger pada komponen Animator. Metode ini memeriksa apakah komponen Animator telah diinisialisasi dan apakah ParameterName tidak kosong atau null. Jika keduanya memenuhi syarat, maka metode SetTrigger dari komponen Animator akan digunakan untuk mengaktifkan trigger yang sesuai.

### 6.3 Animation Scheduler

| using UnityEngine;  using UnityEngine.Events;  public class AnimationScheduler : MonoBehaviour  {  [System.Serializable]  public enum CAnimationType  {  ByAnimationString, ByParameterBoolean, ByParameterTrigger  }  [System.Serializable]  public struct ScheduledAnimation  {  public float animationSecond; // Detik ketika animasi dijalankan  public string animationName; // Nama animasi yang ingin dijalankan  public bool isPlayed;  public UnityEvent animationEvents;  }  [Header("Animation Settings")]  public CAnimationType AnimationType;  [Header("Scheduled Settings")]  public float currentSecond;  public ScheduledAnimation[] scheduledAnimations; // Daftar animasi yang dijadwalkan  Animator targetAnimator; // Animator dari karakter yang ingin Anda animasikan  private float elapsedTime = 0f; // Waktu yang telah berlalu  private float lastSecond = -1; // Detik terakhir yang telah diproses  private float lastIndex = -1; // Detik terakhir yang telah diproses  private string lastAnimationName = ""; // Nama animasi terakhir yang dijalankan  void Start()  {  targetAnimator = GetComponent<Animator>();  }  private void Update()  {  elapsedTime += Time.deltaTime;  currentSecond = elapsedTime;  // Jika detik saat ini berbeda dengan detik terakhir  if (currentSecond != lastSecond)  {  for (int i = 0; i < scheduledAnimations.Length; i++)  {  if (scheduledAnimations[i].animationSecond <= currentSecond && scheduledAnimations[i].animationName != lastAnimationName && !scheduledAnimations[i].isPlayed)  {  if (AnimationType == CAnimationType.ByAnimationString)  {  scheduledAnimations[i].isPlayed = true;  targetAnimator.Play(scheduledAnimations[i].animationName);  lastAnimationName = scheduledAnimations[i].animationName;  scheduledAnimations[i].animationEvents?.Invoke();  }  if (AnimationType == CAnimationType. ByParameterTrigger)  {  scheduledAnimations[i].isPlayed = true;  targetAnimator.SetTrigger(scheduledAnimations[i].animationName);  lastAnimationName = scheduledAnimations[i].animationName;  scheduledAnimations[i].animationEvents?.Invoke();  }  if (AnimationType == CAnimationType.ByParameterBoolean)  {  scheduledAnimations[i].isPlayed = true;  targetAnimator.SetBool(scheduledAnimations[i].animationName, true);  lastAnimationName = scheduledAnimations[i].animationName;  scheduledAnimations[i].animationEvents?.Invoke();  }  }  }  lastSecond = currentSecond;  }  }  public void PlayAnimation(string player)  {  targetAnimator.Play(player);  }  } |
| --- |

### 6.4 Animation Tween

| using UnityEngine;  using UnityEngine.UI;  public class AnimationTween : MonoBehaviour  {  public enum Direction  {  Left,  Right,  Up,  Down  }  [Header("Main Settings")]  public Image imageToMove;  public float moveSpeed = 10f; // Kecepatan per detik  public float moveDistance = 100f; // Jarak pergerakan  private bool isMoving = false;  private Vector3 startPosition;  private Vector3 targetPosition;  private float timer = 0f;  private void Start()  {  startPosition = imageToMove.transform.position;  targetPosition = startPosition;  }  private void Update()  {  if (isMoving)  {  // Menghitung perpindahan posisi  float step = moveSpeed \* Time.deltaTime \* 100;  imageToMove.transform.position = Vector3.MoveTowards(imageToMove.transform.position, targetPosition, step);  // Memeriksa jika target posisi tercapai  if (Vector3.Distance(imageToMove.transform.position, targetPosition) < 0.01f)  {  isMoving = false;  }  }  }  public void Move(Direction direction)  {  if (!isMoving)  {  // Menentukan arah pergerakan berdasarkan enum  float distance = (direction == Direction.Left) ? -moveDistance : moveDistance;  targetPosition = startPosition + new Vector3(distance, 0f, 0f);  // Memulai pergerakan  isMoving = true;  timer = 0f;  }  }  public void MoveLeft()  {  if (!isMoving)  {  // Menentukan arah pergerakan berdasarkan enum  float distance = -moveDistance;  targetPosition = startPosition + new Vector3(distance, 0f, 0f);  // Memulai pergerakan  isMoving = true;  timer = 0f;  }  }  public void MoveRight()  {  if (!isMoving)  {  // Menentukan arah pergerakan berdasarkan enum  float distance = moveDistance;  targetPosition = startPosition + new Vector3(distance, 0f, 0f);  // Memulai pergerakan  isMoving = true;  timer = 0f;  }  }  public void MoveUp()  {  if (!isMoving)  {  // Menentukan arah pergerakan berdasarkan enum  float distance = moveDistance;  targetPosition = startPosition + new Vector3(0f, distance, 0f);  // Memulai pergerakan  isMoving = true;  timer = 0f;  }  }  public void MoveDown()  {  if (!isMoving)  {  // Menentukan arah pergerakan berdasarkan enum  float distance = -moveDistance;  targetPosition = startPosition + new Vector3(0f, distance, 0f);  // Memulai pergerakan  isMoving = true;  timer = 0f;  }  }  } |
| --- |

## 7. Instantiation

Event Instantiation dalam game berkaitan dengan pembuatan objek atau karakter baru dalam game saat berlangsung. Hal ini bisa mencakup menciptakan musuh baru, item, atau peluru saat pemain bergerak melalui level. Dalam pemrograman game, ini biasanya melibatkan penggunaan fungsi yang memanggil atau 'menginstansiasi' objek baru dari kelas atau prefab yang telah ditentukan. Misalnya, saat pemain menembakkan senjata, pemrograman untuk event Instantiation ini mungkin melibatkan pemanggilan fungsi yang menciptakan peluru baru dari prefab peluru yang telah ditentukan. Engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola instansiasi objek. Pemahaman yang baik tentang konsep pemrograman objek dan penggunaan kelas dan prefab adalah kunci untuk mengimplementasikan event Instantiation dengan efektif dalam game. Dengan mengelola instansiasi objek dengan efektif, programmer dapat menciptakan gameplay yang dinamis dan menarik bagi pemain.

### 7.1 Instantiation Prefabs

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class InstantiationPrefabs : MonoBehaviour  {  [Header("Prefabs Settings")]  public GameObject TargetPrefabs;  public bool SetAsParent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  GameObject instantiatedObject;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void InvokeInstantiation()  {  instantiatedObject = Instantiate(TargetPrefabs, transform.position, transform.rotation, transform.parent);  if (SetAsParent)  {  instantiatedObject.transform.parent = transform;  instantiatedObject.transform.localScale = new Vector3(transform.localScale.x / instantiatedObject.transform.localScale.x,  transform.localScale.y / instantiatedObject.transform.localScale.y,  transform.localScale.z / instantiatedObject.transform.localScale.z);  }  }  public void InvokeInstantiationDelay()  {  instantiatedObject = Instantiate(TargetPrefabs, transform.position, transform.rotation, transform.parent);  if (SetAsParent)  {  instantiatedObject.transform.parent = transform;  instantiatedObject.transform.localScale = new Vector3(transform.localScale.x / instantiatedObject.transform.localScale.x,  transform.localScale.y / instantiatedObject.transform.localScale.y,  transform.localScale.z / instantiatedObject.transform.localScale.z);  }  Destroy(instantiatedObject, 3f);  }  public void InvokeInstantiationDelay(float aValue)  {  Invoke("InvokeInstantiation", aValue);  }  public void InvokeInstantiationDelayAndDestroy(float aValue)  {  Invoke("InvokeInstantiationDelay", aValue);  }  public void DestroyInstantiationDelay(float aValue)  {  Destroy(instantiatedObject, aValue);  }  } |
| --- |

Skrip InstantiationPrefabs digunakan untuk meng-instantiate objek (prefab) dalam permainan Unity. Skrip ini memberikan beberapa opsi saat meng-instantiate objek, seperti menentukan apakah objek yang di-instantiate akan dijadikan sebagai parent dari objek yang memanggil skrip ini, serta opsi untuk menghancurkan objek yang di-instantiate setelah jangka waktu tertentu. Berikut penjelasan komponen-komponen utama dari skrip ini:

Prefabs Settings:

TargetPrefabs: Ini adalah objek (prefab) yang akan di-instantiate. Anda harus menetapkan objek prefab ini melalui Inspector Unity.

SetAsParent: Jika diaktifkan (checklist tercentang), objek yang di-instantiate akan dijadikan sebagai parent dari objek yang memanggil skrip ini. Ini akan membuat objek yang di-instantiate mengikuti perubahan posisi, rotasi, dan skala objek yang memanggil skrip ini.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Variabel instantiatedObject: Ini adalah referensi ke objek yang telah di-instantiate. Variabel ini digunakan untuk mengakses objek yang telah di-instantiate jika Anda ingin mengubah atau menghapusnya nantinya.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode InvokeInstantiation(): Metode ini digunakan untuk meng-instantiate objek yang telah ditetapkan sebagai TargetPrefabs. Objek tersebut akan di-instantiate pada posisi, rotasi, dan parent yang sesuai dengan objek yang memanggil skrip ini. Jika SetAsParent diaktifkan, objek yang di-instantiate akan dijadikan sebagai parent dan ukuran skala objek tersebut akan disesuaikan dengan objek yang memanggil skrip ini.

Metode InvokeInstantiationDelay(): Metode ini mirip dengan InvokeInstantiation(), namun juga menghancurkan objek yang di-instantiate setelah 3 detik.

Metode InvokeInstantiationDelay(float aValue): Metode ini memungkinkan Anda untuk menunda pemanggilan InvokeInstantiation() selama jumlah waktu tertentu yang ditentukan oleh aValue.

Metode InvokeInstantiationDelayAndDestroy(float aValue): Metode ini memungkinkan Anda untuk menunda pemanggilan InvokeInstantiationDelay() selama jumlah waktu tertentu yang ditentukan oleh aValue.

Metode DestroyInstantiationDelay(float aValue): Metode ini digunakan untuk menghancurkan objek yang di-instantiate setelah jumlah waktu tertentu yang ditentukan oleh aValue.

Skrip ini berguna untuk menghasilkan efek instansiasi objek dalam permainan, seperti ledakan, efek khusus, atau untuk menghasilkan objek-objek dinamis lainnya yang diperlukan dalam permainan Anda. Dengan berbagai opsi yang disediakan, Anda dapat mengatur efek instansiasi sesuai dengan kebutuhan permainan Anda.

### 7.2 Instantiation Resource

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class InstantiationFromResources : MonoBehaviour  {  [Header("Resource Settings")]  public string ResourcePath; // Nama file di folder Resources  [Header("Prefabs Settings")]  public bool SetAsParent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  GameObject instantiatedObject;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void LoadAndInstantiate()  {  // Menggunakan metode Resources.Load untuk mengambil objek dari folder Resources  GameObject loadedPrefab = Resources.Load<GameObject>(ResourcePath);  if (loadedPrefab != null)  {  instantiatedObject = Instantiate(loadedPrefab, transform.position, transform.rotation, transform.parent);  if (SetAsParent)  {  instantiatedObject.transform.parent = transform;  instantiatedObject.transform.localScale = new Vector3(  transform.localScale.x / instantiatedObject.transform.localScale.x,  transform.localScale.y / instantiatedObject.transform.localScale.y,  transform.localScale.z / instantiatedObject.transform.localScale.z);  }  }  else  {  Debug.LogError("Resource not found at path: " + ResourcePath);  }  }  // Metode tambahan untuk menghancurkan objek yang diinstansiasi  public void DestroyInstantiation()  {  if (instantiatedObject != null)  {  Destroy(instantiatedObject);  }  }  } |
| --- |

Skrip InstantiationFromResources digunakan untuk meng-instantiate objek (prefab) dari folder "Resources" dalam proyek Unity. Skrip ini memungkinkan Anda untuk mengambil objek dari folder Resources berdasarkan nama file yang ditentukan dan meng-instantiate objek tersebut dalam permainan. Berikut penjelasan komponen-komponen utama dari skrip ini:

Resource Settings:

ResourcePath: Ini adalah nama file dari objek yang akan diambil dari folder "Resources". Anda perlu menentukan nama file ini melalui Inspector Unity.

Prefabs Settings:

SetAsParent: Jika diaktifkan (checklist tercentang), objek yang di-instantiate akan dijadikan sebagai parent dari objek yang memanggil skrip ini. Ini akan membuat objek yang di-instantiate mengikuti perubahan posisi, rotasi, dan skala objek yang memanggil skrip ini.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Variabel instantiatedObject: Ini adalah referensi ke objek yang telah di-instantiate. Variabel ini digunakan untuk mengakses objek yang telah di-instantiate jika Anda ingin mengubah atau menghapusnya nantinya.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode LoadAndInstantiate(): Metode ini digunakan untuk mengambil objek dari folder "Resources" berdasarkan nama file yang ditentukan dalam ResourcePath. Objek tersebut kemudian di-instantiate pada posisi, rotasi, dan parent yang sesuai dengan objek yang memanggil skrip ini. Jika SetAsParent diaktifkan, objek yang di-instantiate akan dijadikan sebagai parent dan ukuran skala objek tersebut akan disesuaikan dengan objek yang memanggil skrip ini.

Metode DestroyInstantiation(): Metode ini digunakan untuk menghancurkan objek yang telah di-instantiate. Jika objek telah di-instantiate sebelumnya dan masih ada dalam memori, maka metode ini akan menghancurkannya.

Skrip ini berguna saat Anda ingin mengambil objek dari folder "Resources" secara dinamis dalam permainan, yang berguna untuk menghasilkan berbagai efek atau objek sesuai dengan kebutuhan permainan Anda. Dengan menggunakan skrip ini, Anda dapat memanfaatkan objek-objek yang telah disiapkan sebelumnya dalam folder "Resources" dengan mudah.

### 7.3 Instantiation Addressable

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using UnityEngine.AddressableAssets;  using UnityEngine.ResourceManagement.AsyncOperations;  public class InstantiationFromAddressables : MonoBehaviour  {  [Header("Asset Address Settings")]  public string AssetAddress; // Alamat dari objek Addressable  [Header("Prefabs Settings")]  public bool SetAsParent;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  GameObject instantiatedObject;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void LoadAndInstantiate()  {  // Menggunakan Addressables untuk memuat objek Addressable  Addressables.LoadAssetAsync<GameObject>(AssetAddress).Completed += OnLoadCompleted;  }  // Callback yang dipanggil ketika objek Addressable selesai dimuat  private void OnLoadCompleted(AsyncOperationHandle<GameObject> obj)  {  if (obj.Status == AsyncOperationStatus.Succeeded)  {  instantiatedObject = Instantiate(obj.Result, transform.position, transform.rotation, transform.parent);  if (SetAsParent)  {  instantiatedObject.transform.parent = transform;  instantiatedObject.transform.localScale = new Vector3(  transform.localScale.x / instantiatedObject.transform.localScale.x,  transform.localScale.y / instantiatedObject.transform.localScale.y,  transform.localScale.z / instantiatedObject.transform.localScale.z);  }  }  else  {  Debug.LogError("Failed to load Addressable Asset with address: " + AssetAddress);  }  }  // Metode tambahan untuk menghancurkan objek yang diinstansiasi  public void DestroyInstantiation()  {  if (instantiatedObject != null)  {  Destroy(instantiatedObject);  }  }  } |
| --- |

Skrip InstantiationFromAddressables digunakan untuk meng-instantiate objek (prefab) dari Addressable Assets System di Unity. Addressable Assets System memungkinkan Anda mengatur, memuat, dan mengakses sumber daya secara dinamis dalam permainan. Skrip ini memungkinkan Anda untuk mengambil objek yang telah ditentukan dalam Addressable Assets System berdasarkan alamatnya dan meng-instantiate objek tersebut dalam permainan. Berikut penjelasan komponen-komponen utama dari skrip ini:

Asset Address Settings:

AssetAddress: Ini adalah alamat atau label dari objek yang ingin Anda muat dari Addressable Assets System. Anda perlu menentukan alamat ini melalui Inspector Unity.

Prefabs Settings:

SetAsParent: Jika diaktifkan (checklist tercentang), objek yang di-instantiate akan dijadikan sebagai parent dari objek yang memanggil skrip ini. Ini akan membuat objek yang di-instantiate mengikuti perubahan posisi, rotasi, dan skala objek yang memanggil skrip ini.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Variabel instantiatedObject: Ini adalah referensi ke objek yang telah di-instantiate. Variabel ini digunakan untuk mengakses objek yang telah di-instantiate jika Anda ingin mengubah atau menghapusnya nantinya.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode LoadAndInstantiate(): Metode ini digunakan untuk memuat objek dari Addressable Assets System berdasarkan alamat yang ditentukan dalam AssetAddress. Metode ini menggunakan Addressables.LoadAssetAsync<GameObject>(AssetAddress) untuk memuat objek secara asinkron. Ketika pembebanan selesai, callback OnLoadCompleted akan dipanggil.

Callback OnLoadCompleted(): Metode ini dipanggil ketika pembebanan objek Addressable selesai. Jika pembebanan berhasil (status Succeeded), objek yang dimuat di-instantiate pada posisi, rotasi, dan parent yang sesuai dengan objek yang memanggil skrip ini. Jika SetAsParent diaktifkan, objek yang di-instantiate akan dijadikan sebagai parent dan ukuran skala objek tersebut akan disesuaikan dengan objek yang memanggil skrip ini.

Metode DestroyInstantiation(): Metode ini digunakan untuk menghancurkan objek yang telah di-instantiate. Jika objek telah di-instantiate sebelumnya dan masih ada dalam memori, maka metode ini akan menghancurkannya.

Skrip ini sangat berguna saat Anda ingin mengambil objek dari Addressable Assets System, yang memungkinkan Anda untuk mengatur sumber daya secara terpisah dari proyek utama Anda dan memuatnya hanya saat dibutuhkan dalam permainan. Dengan menggunakan skrip ini, Anda dapat mengambil objek-objek tersebut secara dinamis selama permainan berlangsung, menghemat memori dan waktu pembebanan.

## 8. Collision

Event Collision dalam game berkaitan dengan interaksi antara objek atau karakter saat mereka bersentuhan atau tumpul satu sama lain. Ini bisa mencakup tabrakan antara pemain dan musuh, pemain dan tembok, atau peluru dan musuh. Dalam pemrograman game, ini biasanya melibatkan penggunaan fungsi dan sensor khusus yang mendeteksi ketika dua objek saling bersinggungan atau tumpul. Misalnya, saat peluru yang ditembakkan oleh pemain mengenai musuh, pemrograman untuk event Collision ini mungkin melibatkan pemanggilan fungsi yang mengurangi kesehatan musuh atau menghancurkannya. Engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola tabrakan antar objek. Pemahaman yang baik tentang fisika dan geometri, serta kemampuan untuk bekerja dengan sensor dan fungsi khusus adalah kunci untuk mengimplementasikan event Collision dengan efektif dalam game. Dengan mengelola tabrakan dengan efektif, programmer dapat menciptakan gameplay yang lebih dinamis dan interaktif bagi pemain.

### 8.1 Collision 2D

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class Collision2DNormal : MonoBehaviour  {  [Header("Tag Settings")]  public string Tag;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Collision Settings")]  public UnityEvent OnCollisionEnterEvent;  public UnityEvent OnCollisionStayEvent;  public UnityEvent OnCollisionExitEvent;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void Destroy()  {  Destroy(gameObject);  }  void OnCollisionEnter2D(Collision2D collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionEnterEvent.Invoke();  }  }  void OnCollisionStay2D(Collision2D collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionStayEvent.Invoke();  }  }  void OnCollisionExit2D(Collision2D collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionExitEvent.Invoke();  }  }  } |
| --- |

Skrip Collision2DNormal digunakan untuk mengelola deteksi dan respons terhadap tumpukan (collision) 2D antara objek yang menggunakan skrip ini dan objek lain dalam permainan Unity. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

Tag: Ini adalah tag yang digunakan sebagai kriteria untuk memicu event. Skrip ini akan menanggapi tumpukan (collision) hanya jika objek yang bersentuhan memiliki tag yang sesuai dengan nilai Tag yang Anda tentukan melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

OnCollisionEnterEvent: Ini adalah event yang dipicu saat objek ini pertama kali bersentuhan dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnCollisionStayEvent: Ini adalah event yang terus dipicu selama objek ini bersentuhan dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnCollisionExitEvent: Ini adalah event yang dipicu saat objek ini berhenti bersentuhan dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode Destroy(): Ini adalah metode sederhana yang digunakan untuk menghancurkan objek yang menggunakan skrip ini. Metode ini dapat dipanggil dari luar skrip (misalnya, saat kondisi tertentu terpenuhi).

Metode OnCollisionEnter2D(Collision2D collision): Metode ini dipanggil saat objek ini pertama kali bersentuhan dengan objek lain yang memiliki tumpukan (collision) 2D. Metode ini akan memeriksa tag objek lain yang bersentuhan dan jika sesuai dengan nilai Tag, maka OnCollisionEnterEvent akan dipanggil.

Metode OnCollisionStay2D(Collision2D collision): Metode ini dipanggil setiap frame selama objek ini bersentuhan dengan objek lain yang memiliki tumpukan (collision) 2D. Metode ini juga akan memeriksa tag objek lain dan memicu OnCollisionStayEvent jika tag sesuai.

Metode OnCollisionExit2D(Collision2D collision): Metode ini dipanggil saat objek ini berhenti bersentuhan dengan objek lain yang memiliki tumpukan (collision) 2D. Seperti sebelumnya, metode ini memeriksa tag objek lain dan memicu OnCollisionExitEvent jika tag sesuai.

Skrip ini berguna untuk mengatur respons yang spesifik terhadap tumpukan objek dalam permainan Anda berdasarkan tag yang sesuai. Misalnya, Anda dapat menggunakan ini untuk mendeteksi ketika pemain bersentuhan dengan koin dalam permainan, yang memungkinkan Anda untuk memicu peristiwa seperti penambahan poin atau menghilangkan koin dari dunia permainan.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class Collision2DTrigger : MonoBehaviour  {  [Header("Tag Settings")]  public string Tag;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Collision Settings")]  public UnityEvent OnTriggerEnterEvent;  public UnityEvent OnTriggerStayEvent;  public UnityEvent OnTriggerExitEvent;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void Destroy()  {  Destroy(gameObject);  }  void OnTriggerEnter2D(Collider2D collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerEnterEvent.Invoke();  }  }  void OnTriggerStay2D(Collider2D collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerStayEvent.Invoke();  }  }  void OnTriggerExit2D(Collider2D collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerExitEvent.Invoke();  }  }  } |
| --- |

Skrip Collision2DTrigger digunakan untuk mengelola deteksi dan respons saat objek yang menggunakan skrip ini berada dalam bentuk trigger terhadap objek lain dalam permainan Unity. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

Tag: Ini adalah tag yang digunakan sebagai kriteria untuk memicu event. Skrip ini akan menanggapi objek lain yang bersentuhan dengan objek ini dalam bentuk trigger hanya jika objek lain memiliki tag yang sesuai dengan nilai Tag yang Anda tentukan melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

OnTriggerEnterEvent: Ini adalah event yang dipicu saat objek ini pertama kali masuk ke dalam bentuk trigger dari objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnTriggerStayEvent: Ini adalah event yang terus dipicu selama objek ini berada dalam bentuk trigger dari objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnTriggerExitEvent: Ini adalah event yang dipicu saat objek ini keluar dari bentuk trigger objek lain yang memiliki tag yang sesuai dengan nilai Tag.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode Destroy(): Ini adalah metode sederhana yang digunakan untuk menghancurkan objek yang menggunakan skrip ini. Metode ini dapat dipanggil dari luar skrip (misalnya, saat kondisi tertentu terpenuhi).

Metode OnTriggerEnter2D(Collider2D collider): Metode ini dipanggil saat objek ini pertama kali masuk ke dalam bentuk trigger dari objek lain yang memiliki tumpukan (collision) 2D. Metode ini akan memeriksa tag objek lain yang bersentuhan dan jika sesuai dengan nilai Tag, maka OnTriggerEnterEvent akan dipanggil.

Metode OnTriggerStay2D(Collider2D collider): Metode ini dipanggil setiap frame selama objek ini berada dalam bentuk trigger dari objek lain yang memiliki tumpukan (collision) 2D. Metode ini juga akan memeriksa tag objek lain dan memicu OnTriggerStayEvent jika tag sesuai.

Metode OnTriggerExit2D(Collider2D collider): Metode ini dipanggil saat objek ini keluar dari bentuk trigger objek lain yang memiliki tumpukan (collision) 2D. Seperti sebelumnya, metode ini memeriksa tag objek lain dan memicu OnTriggerExitEvent jika tag sesuai.

Skrip ini berguna untuk mengatur respons yang spesifik terhadap tumpukan objek dalam bentuk trigger berdasarkan tag yang sesuai. Ini dapat digunakan, misalnya, untuk mendeteksi ketika pemain memasuki daerah tertentu dalam permainan atau berinteraksi dengan objek tertentu dalam permainan.

### 8.2 Collision 3D

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class Collision3DNormal : MonoBehaviour  {  [Header("Tag Settings")]  public string Tag;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Collision Settings")]  public UnityEvent OnCollisionEnterEvent;  public UnityEvent OnCollisionStayEvent;  public UnityEvent OnCollisionExitEvent;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void Destroy()  {  Destroy(gameObject);  }  void OnCollisionEnter(Collision collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionEnterEvent.Invoke();  }  }  void OnCollisionStay(Collision collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionStayEvent.Invoke();  }  }  void OnCollisionExit(Collision collision)  {  if (collision.transform.CompareTag(Tag))  {  OnCollisionExitEvent.Invoke();  }  }  } |
| --- |

Skrip Collision3DNormal digunakan untuk mengelola deteksi dan respons dalam bentuk tumpukan (collision) 3D antara objek yang menggunakan skrip ini dan objek lain dalam permainan Unity. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

Tag: Ini adalah tag yang digunakan sebagai kriteria untuk memicu event. Skrip ini akan menanggapi objek lain yang bersentuhan dalam bentuk tumpukan 3D dengan objek ini hanya jika objek lain memiliki tag yang sesuai dengan nilai Tag yang Anda tentukan melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

OnCollisionEnterEvent: Ini adalah event yang dipicu saat objek ini pertama kali bersentuhan dalam bentuk tumpukan 3D dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnCollisionStayEvent: Ini adalah event yang terus dipicu selama objek ini bersentuhan dalam bentuk tumpukan 3D dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

OnCollisionExitEvent: Ini adalah event yang dipicu saat objek ini keluar dari bentuk tumpukan 3D dengan objek lain yang memiliki tag yang sesuai dengan nilai Tag.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode Destroy(): Ini adalah metode sederhana yang digunakan untuk menghancurkan objek yang menggunakan skrip ini. Metode ini dapat dipanggil dari luar skrip (misalnya, saat kondisi tertentu terpenuhi).

Metode OnCollisionEnter(Collision collision): Metode ini dipanggil saat objek ini pertama kali bersentuhan dalam bentuk tumpukan 3D dengan objek lain. Metode ini akan memeriksa tag objek lain yang bersentuhan dan jika sesuai dengan nilai Tag, maka OnCollisionEnterEvent akan dipanggil.

Metode OnCollisionStay(Collision collision): Metode ini dipanggil setiap frame selama objek ini bersentuhan dalam bentuk tumpukan 3D dengan objek lain. Metode ini juga akan memeriksa tag objek lain dan memicu OnCollisionStayEvent jika tag sesuai.

Metode OnCollisionExit(Collision collision): Metode ini dipanggil saat objek ini keluar dari bentuk tumpukan 3D dengan objek lain. Seperti sebelumnya, metode ini memeriksa tag objek lain dan memicu OnCollisionExitEvent jika tag sesuai.

Skrip ini berguna untuk mengatur respons yang spesifik terhadap tumpukan objek dalam bentuk collision 3D berdasarkan tag yang sesuai. Ini dapat digunakan, misalnya, untuk mendeteksi ketika pemain berinteraksi dengan objek tertentu dalam permainan atau saat objek tertentu bersentuhan dengan objek lain yang memiliki tag yang sesuai.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class Collision3DTrigger : MonoBehaviour  {  [Header("Tag Settings")]  public string Tag;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Collision Settings")]  public UnityEvent OnTriggerEnterEvent;  public UnityEvent OnTriggerStayEvent;  public UnityEvent OnTriggerExitEvent;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void Destroy()  {  Destroy(gameObject);  }  void OnTriggerEnter(Collider collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerEnterEvent.Invoke();  }  }  void OnTriggerStay(Collider collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerStayEvent.Invoke();  }  }  void OnTriggerExit(Collider collider)  {  if (collider.transform.CompareTag(Tag))  {  OnTriggerExitEvent.Invoke();  }  }  } |
| --- |

Skrip Collision3DTrigger digunakan untuk mengelola deteksi dan respons dalam bentuk "trigger" 3D antara objek yang menggunakan skrip ini dan objek lain dalam permainan Unity. Trigger dalam Unity adalah area yang digunakan untuk mendeteksi ketika objek memasuki atau keluar dari area tersebut, tetapi tidak mengakibatkan pergeseran fisik atau tumpukan seperti dalam collision 3D.

Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

Tag: Ini adalah tag yang digunakan sebagai kriteria untuk memicu event. Skrip ini akan menanggapi objek lain yang masuk atau keluar dari trigger ini hanya jika objek lain memiliki tag yang sesuai dengan nilai Tag yang Anda tentukan melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

OnTriggerEnterEvent: Ini adalah event yang dipicu saat objek lain memasuki trigger ini.

OnTriggerStayEvent: Ini adalah event yang dipicu selama objek lain berada dalam trigger ini.

OnTriggerExitEvent: Ini adalah event yang dipicu saat objek lain keluar dari trigger ini.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode Destroy(): Ini adalah metode sederhana yang digunakan untuk menghancurkan objek yang menggunakan skrip ini. Metode ini dapat dipanggil dari luar skrip (misalnya, saat kondisi tertentu terpenuhi).

Metode OnTriggerEnter(Collider collider): Metode ini dipanggil saat objek lain memasuki trigger ini. Metode ini akan memeriksa tag objek lain yang memasuki trigger dan jika sesuai dengan nilai Tag, maka OnTriggerEnterEvent akan dipanggil.

Metode OnTriggerStay(Collider collider): Metode ini dipanggil selama objek lain berada dalam trigger ini. Seperti sebelumnya, metode ini memeriksa tag objek lain yang berada dalam trigger dan memicu OnTriggerStayEvent jika tag sesuai.

Metode OnTriggerExit(Collider collider): Metode ini dipanggil saat objek lain keluar dari trigger ini. Metode ini juga akan memeriksa tag objek lain yang keluar dari trigger dan memicu OnTriggerExitEvent jika tag sesuai.

Skrip ini berguna untuk mengatur respons yang spesifik terhadap trigger 3D berdasarkan tag yang sesuai. Anda dapat menggunakannya, misalnya, untuk mengimplementasikan berbagai jenis logika berdasarkan interaksi objek dengan area trigger dalam permainan Anda.

## 9. Iteration

Event Iteration dalam game berkaitan dengan perulangan aksi atau proses selama periode waktu tertentu atau sampai kondisi tertentu terpenuhi. Ini bisa mencakup pengulangan gerakan musuh, pembaruan posisi pemain, atau pemutakhiran status game. Dalam pemrograman game, ini biasanya melibatkan penggunaan struktur kontrol perulangan seperti for, while, atau do-while loops. Misalnya, untuk menggerakkan musuh maju dan mundur di sepanjang jalan tertentu, pemrograman untuk event Iteration ini mungkin melibatkan penggunaan loop yang mengulangi gerakan maju dan mundur musuh sampai musuh itu dihancurkan atau level selesai. Engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola perulangan aksi dan proses. Pemahaman yang baik tentang struktur kontrol perulangan dan penggunaan kondisi adalah kunci untuk mengimplementasikan event Iteration dengan efektif dalam game. Dengan mengelola perulangan dengan efektif, programmer dapat menciptakan gameplay yang lebih dinamis dan menantang bagi pemain.

### 9.1 Iteration GameObject Name

| using System.Collections;  using System.Collections.Generic;  using UnityEditor;  using UnityEngine;  using UnityEngine.Events;  public class IterationGameObjectName : MonoBehaviour  {  [Header("Tag Settings")]  public string TargetName;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Find Object")]  public GameObject TargetObject;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void FindObject()  {  TargetObject = GameObject.Find(TargetName);  }  public void HideObject()  {  TargetObject = GameObject.Find(TargetName);  TargetObject.SetActive(false);  }  public void DestroyObject()  {  TargetObject = GameObject.Find(TargetName);  Destroy(TargetObject);  }  } |
| --- |

Skrip IterationGameObjectName adalah skrip yang digunakan untuk mengelola operasi terhadap objek dalam permainan Unity berdasarkan nama objeknya. Skrip ini memungkinkan Anda untuk mencari, menyembunyikan, atau menghancurkan objek berdasarkan nama objek yang sesuai.

Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

TargetName: Ini adalah nama objek yang akan dioperasikan. Anda harus mengatur nama ini melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Find Object:

TargetObject: Ini adalah objek yang akan menjadi target dari operasi yang akan dijalankan. Anda dapat menggunakan metode FindObject() untuk mencari objek dengan nama yang sesuai dan menetapkan objek tersebut ke variabel TargetObject.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode FindObject(): Metode ini digunakan untuk mencari objek dalam permainan berdasarkan nama yang sesuai dengan nilai TargetName. Hasil pencarian kemudian disimpan dalam variabel TargetObject.

Metode HideObject(): Metode ini mencari objek dengan nama yang sesuai dan kemudian menonaktifkan (menyembunyikan) objek tersebut dengan mengatur properti SetActive(false) pada TargetObject.

Metode DestroyObject(): Metode ini mencari objek dengan nama yang sesuai dan kemudian menghancurkan (menghapus) objek tersebut dengan menggunakan metode Destroy() pada TargetObject.

Skrip ini berguna ketika Anda ingin mengakses, menyembunyikan, atau menghapus objek dalam permainan berdasarkan nama objeknya. Ini bisa digunakan dalam berbagai situasi, seperti mengaktifkan atau menonaktifkan objek tertentu, atau menghapus objek saat diperlukan dalam permainan Anda.

### 9.2 Iteration GameObject Tag

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class IterationGameObjectTag : MonoBehaviour  {  [Header("Tag Settings")]  public string TargetTag;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Find Objects")]  public GameObject[] ArrayObjects;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  public void FindAllObjects()  {  ArrayObjects = GameObject.FindGameObjectsWithTag(TargetTag);  Debug.Log("Total GameObject Found: " + ArrayObjects.Length);  }  public void HideAllObjects()  {  ArrayObjects = GameObject.FindGameObjectsWithTag(TargetTag);  foreach (GameObject obj in ArrayObjects)  {  obj.SetActive(false);  }  }  public void DestroyAllObjects()  {  ArrayObjects = GameObject.FindGameObjectsWithTag(TargetTag);  foreach (GameObject obj in ArrayObjects)  {  Destroy(obj);  }  }  } |
| --- |

Skrip IterationGameObjectTag adalah skrip yang digunakan untuk mengelola operasi terhadap objek-objek dalam permainan Unity berdasarkan tag objek mereka. Skrip ini memungkinkan Anda untuk mencari, menyembunyikan, atau menghancurkan semua objek yang memiliki tag tertentu.

Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Tag Settings:

TargetTag: Ini adalah tag objek yang akan dioperasikan. Anda harus mengatur tag ini melalui Inspector Unity.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Find Objects:

ArrayObjects: Ini adalah array objek yang akan menjadi target dari operasi yang akan dijalankan. Anda dapat menggunakan metode FindAllObjects() untuk mencari semua objek yang memiliki tag yang sesuai dan menetapkan array objek tersebut ke variabel ArrayObjects.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada.

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. UpdateEvents diaktifkan jika ada.

Metode FindAllObjects(): Metode ini digunakan untuk mencari semua objek dalam permainan yang memiliki tag yang sesuai dengan nilai TargetTag. Hasil pencarian kemudian disimpan dalam array ArrayObjects. Jumlah objek yang ditemukan juga ditampilkan dalam konsol dengan menggunakan Debug.Log().

Metode HideAllObjects(): Metode ini mencari semua objek dalam permainan yang memiliki tag yang sesuai dan kemudian menonaktifkan (menyembunyikan) semua objek tersebut dengan mengatur properti SetActive(false) pada masing-masing objek dalam ArrayObjects.

Metode DestroyAllObjects(): Metode ini mencari semua objek dalam permainan yang memiliki tag yang sesuai dan kemudian menghancurkan (menghapus) semua objek tersebut dengan menggunakan metode Destroy() pada masing-masing objek dalam ArrayObjects.

Skrip ini berguna ketika Anda ingin mengakses, menyembunyikan, atau menghapus sejumlah objek dalam permainan berdasarkan tag objek mereka. Ini bisa digunakan untuk mengatur dan mengelola sekelompok objek yang memiliki tag yang sama dalam permainan Anda.

### 9.3 Iteration Mesh Renderer

| using UnityEngine;  public class IterationMeshRenderer : MonoBehaviour  {  public bool MeshRenderer = true;  private MeshRenderer[] childMeshRenderers;  private void Awake()  {  // Dapatkan semua MeshRenderer dari child objects  childMeshRenderers = GetComponentsInChildren<MeshRenderer>();  }  private void Update()  {  ToggleRenderers();  }  private void ToggleRenderers()  {  foreach (MeshRenderer renderer in childMeshRenderers)  {  renderer.enabled = MeshRenderer;  }  }  } |
| --- |

### 9.4 Iteration Box Collider

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class IterationBoxCollider : MonoBehaviour  {  public bool ObjectCollider = true;  private BoxCollider[] childBoxColliders;  private void Awake()  {  // Dapatkan semua BoxCollider dari child objects  childBoxColliders = GetComponentsInChildren<BoxCollider>();  }  private void Update()  {  ToggleColliders();  }  private void ToggleColliders()  {  foreach (BoxCollider collider in childBoxColliders)  {  collider.enabled = ObjectCollider;  }  }  } |
| --- |

## 10. Condition

Event Condition dalam game berkaitan dengan pengujian kondisi tertentu untuk menentukan aksi atau hasil yang harus diambil. Ini bisa mencakup pengecekan apakah pemain memiliki cukup poin kesehatan untuk bertahan hidup, apakah musuh sudah dihancurkan, atau apakah semua level sudah selesai. Dalam pemrograman game, ini biasanya melibatkan penggunaan struktur kontrol kondisional seperti if-else atau switch-case. Misalnya, untuk menentukan apakah pemain menang atau kalah, pemrograman untuk event Condition ini mungkin melibatkan pengecekan kondisi kesehatan pemain atau jumlah musuh yang tersisa. Engine game seperti Unity atau Unreal Engine menyediakan alat dan fungsi khusus untuk mengelola kondisi dan pengambilan keputusan. Pemahaman yang baik tentang struktur kontrol kondisional dan penggunaan variabel dan logika adalah kunci untuk mengimplementasikan event Condition dengan efektif dalam game. Dengan mengelola kondisi dengan efektif, programmer dapat menciptakan gameplay yang lebih interaktif dan menantang bagi pemain.

### 10.1 Condition Boolean

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionBoolean : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationBoolean CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public UnityEvent TrueEvents;  public UnityEvent FalseEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (CurrentVariable.CurrentValue == true)  {  TrueEvents?.Invoke();  }  if (CurrentVariable.CurrentValue == false)  {  FalseEvents?.Invoke();  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionBoolean adalah skrip yang digunakan untuk mengatur alur permainan berdasarkan kondisi boolean tertentu. Dengan skrip ini, Anda dapat memeriksa nilai boolean dari suatu variabel atau objek dan menjalankan event tertentu tergantung pada apakah nilai tersebut adalah true atau false.

Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Variable Settings:

CurrentVariable: Ini adalah variabel atau objek yang akan digunakan sebagai sumber nilai boolean untuk kondisi yang akan diperiksa. Anda harus menghubungkan objek atau variabel ini melalui Inspector Unity.

Activated: Ini adalah boolean yang mengontrol apakah pemeriksaan kondisi akan dijalankan atau tidak.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Condition Settings:

TrueEvents: Ini adalah event yang akan dipanggil jika kondisi yang diperiksa (CurrentVariable.CurrentValue) adalah true.

FalseEvents: Ini adalah event yang akan dipanggil jika kondisi yang diperiksa (CurrentVariable.CurrentValue) adalah false.

Metode StartChecking(): Metode ini digunakan untuk mengaktifkan pemeriksaan kondisi dengan mengatur nilai Activated menjadi true.

Metode StopChecking(): Metode ini digunakan untuk menonaktifkan pemeriksaan kondisi dengan mengatur nilai Activated menjadi false.

Metode ConditionChecking(): Metode ini adalah inti dari skrip ini. Ini memeriksa nilai Activated dan, jika diaktifkan, memeriksa nilai boolean dari CurrentVariable.CurrentValue. Jika nilai tersebut adalah true, maka TrueEvents diaktifkan; jika nilai tersebut adalah false, maka FalseEvents diaktifkan.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada. Jika Activated telah diatur sebagai true, maka pemeriksaan kondisi dimulai dengan memanggil StartChecking().

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Di dalamnya, UpdateEvents diaktifkan jika ada, dan pemeriksaan kondisi dipanggil dengan memanggil ConditionChecking().

Dengan skrip ini, Anda dapat mengatur peristiwa-peristiwa dalam permainan Anda berdasarkan nilai boolean dari suatu variabel atau objek, dan Anda dapat mengendalikan apakah pemeriksaan kondisi ini aktif atau tidak dengan mengubah nilai Activated.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionBooleanPref : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public UnityEvent TrueEvents;  public UnityEvent FalseEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (PlayerPrefs.GetInt(CurrentVariable, 0) == 1)  {  TrueEvents?.Invoke();  }  if (PlayerPrefs.GetInt(CurrentVariable, 0) == 0)  {  FalseEvents?.Invoke();  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionBooleanPref adalah versi lain dari skrip ConditionBoolean yang Anda jelaskan sebelumnya. Skrip ini digunakan untuk mengatur alur permainan berdasarkan kondisi boolean, tetapi dalam hal ini, kondisi yang diperiksa didasarkan pada nilai yang disimpan dalam PlayerPrefs.

### 10.2 Condition Integer

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionInteger : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationInteger CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public int LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public int EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public int BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (CurrentVariable.CurrentValue < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (CurrentVariable.CurrentValue == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (CurrentVariable.CurrentValue > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionInteger digunakan untuk mengatur alur permainan berdasarkan kondisi yang melibatkan nilai integer. Skrip ini memungkinkan Anda memeriksa apakah nilai dari suatu variabel (dalam hal ini, CurrentVariable) memenuhi beberapa kondisi tertentu, seperti kurang dari, sama dengan, atau lebih besar dari suatu nilai tertentu. Berikut adalah penjelasan komponen-komponen utama dari skrip ini:

Variable Settings:

CurrentVariable: Ini adalah objek dari tipe CalculationInteger yang memiliki properti CurrentValue yang akan digunakan sebagai sumber nilai integer untuk kondisi yang akan diperiksa.

Activated: Ini adalah boolean yang mengontrol apakah pemeriksaan kondisi akan dijalankan atau tidak.

Event Settings:

StartEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil saat objek ini pertama kali aktif dalam permainan (saat metode Start dipanggil).

UpdateEvents: Ini adalah event yang dapat Anda tambahkan dalam Inspector Unity dan akan dipanggil setiap frame saat objek ini aktif (saat metode Update dipanggil).

Condition Settings:

ifLessThan: Ini adalah boolean yang mengaktifkan pemeriksaan apakah nilai CurrentVariable.CurrentValue kurang dari LessThanCondition.

LessThanCondition: Ini adalah nilai integer yang digunakan sebagai batasan dalam kondisi "kurang dari".

LessThanEvents: Ini adalah event yang akan dipanggil jika kondisi "kurang dari" terpenuhi.

ifEqual: Ini adalah boolean yang mengaktifkan pemeriksaan apakah nilai CurrentVariable.CurrentValue sama dengan EqualCondition.

EqualCondition: Ini adalah nilai integer yang digunakan sebagai batasan dalam kondisi "sama dengan".

EqualEvents: Ini adalah event yang akan dipanggil jika kondisi "sama dengan" terpenuhi.

ifBiggerThan: Ini adalah boolean yang mengaktifkan pemeriksaan apakah nilai CurrentVariable.CurrentValue lebih besar dari BiggerThanCondition.

BiggerThanCondition: Ini adalah nilai integer yang digunakan sebagai batasan dalam kondisi "lebih besar dari".

BiggerThanEvents: Ini adalah event yang akan dipanggil jika kondisi "lebih besar dari" terpenuhi.

Metode StartChecking(): Metode ini digunakan untuk mengaktifkan pemeriksaan kondisi dengan mengatur nilai Activated menjadi true.

Metode StopChecking(): Metode ini digunakan untuk menonaktifkan pemeriksaan kondisi dengan mengatur nilai Activated menjadi false.

Metode ConditionChecking(): Metode ini adalah inti dari skrip ini. Ini memeriksa nilai Activated dan, jika diaktifkan, memeriksa berbagai kondisi yang Anda tentukan. Jika kondisi-kondisi tersebut terpenuhi, maka event yang sesuai akan diaktifkan.

Metode Start(): Metode ini dipanggil saat objek ini pertama kali aktif dalam permainan. Di dalamnya, StartEvents diaktifkan jika ada, dan jika Activated telah diatur sebagai true, maka pemeriksaan kondisi dimulai dengan memanggil StartChecking().

Metode Update(): Metode ini dipanggil setiap frame saat objek ini aktif dalam permainan. Di dalamnya, UpdateEvents diaktifkan jika ada, dan pemeriksaan kondisi dipanggil dengan memanggil ConditionChecking().

Dengan skrip ini, Anda dapat mengatur alur permainan berdasarkan nilai integer yang Anda tentukan dan mengendalikan apakah pemeriksaan kondisi ini aktif atau tidak dengan mengubah nilai Activated.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionIntegerPref : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public int LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public int EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public int BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (PlayerPrefs.GetInt(CurrentVariable, 0) < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (PlayerPrefs.GetInt(CurrentVariable, 0) == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (PlayerPrefs.GetInt(CurrentVariable, 0) > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionIntegerPref hampir identik dengan skrip ConditionInteger yang telah dijelaskan sebelumnya, tetapi dengan perbedaan utama dalam sumber nilai integer yang digunakan untuk kondisi. Dalam kasus ConditionIntegerPref, sumber nilai integer diambil dari penyimpanan PlayerPrefs di Unity, bukan dari variabel CalculationInteger.

### 10.3 Condition Float

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionFloat : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationFloat CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public int LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public int EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public int BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (CurrentVariable.CurrentValue < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (CurrentVariable.CurrentValue == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (CurrentVariable.CurrentValue > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionFloat sangat mirip dengan skrip ConditionInteger yang telah dijelaskan sebelumnya, dengan perbedaan utama dalam jenis variabel yang digunakan. Dalam skrip ini, Anda menggunakan variabel tipe CalculationFloat dan melakukan pemeriksaan kondisi terhadap nilai-nilai float.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionFloatPref : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public float LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public float EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public float BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionFloatPref adalah skrip yang mirip dengan ConditionFloat, tetapi dalam kasus ini, nilai-nilai float yang digunakan dalam pemeriksaan kondisi diambil dari penyimpanan lokal dengan menggunakan PlayerPrefs.

### 10.4 Condition Score

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionScore : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationScore CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public float LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public float EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public float BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (CurrentVariable.CurrentValue < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (CurrentVariable.CurrentValue == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (CurrentVariable.CurrentValue > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionScore adalah skrip yang mirip dengan ConditionFloat, tetapi dalam kasus ini, nilai-nilai yang digunakan dalam pemeriksaan kondisi diambil dari suatu objek CalculationScore.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionScorePref : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public float LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public float EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public float BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionScorePref adalah skrip yang digunakan untuk memeriksa nilai float yang disimpan dalam PlayerPrefs. Ini hampir mirip dengan skrip ConditionFloatPref yang telah Anda sebutkan sebelumnya, tetapi kali ini Anda memeriksa nilai float yang disimpan dalam PlayerPrefs daripada integer. Skrip ini memungkinkan Anda untuk mengatur kondisi berdasarkan apakah nilai float dalam PlayerPrefs dengan kunci yang diberikan (CurrentVariable) kurang dari, sama dengan, atau lebih besar dari nilai yang Anda tentukan dalam LessThanCondition, EqualCondition, atau BiggerThanCondition.

### 10.5 Condition String

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionString : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationString CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifSame;  public string CompareText;  public UnityEvent TrueEvents;  public UnityEvent FalseEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifSame)  {  if (CurrentVariable.CurrentValue == CompareText)  {  TrueEvents?.Invoke();  }  if (CurrentVariable.CurrentValue != CompareText)  {  FalseEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionString adalah skrip yang digunakan untuk memeriksa nilai string yang disimpan dalam objek CalculationString. Skrip ini memungkinkan Anda untuk mengatur kondisi berdasarkan apakah string dalam CurrentVariable.CurrentValue sama dengan string yang Anda tentukan dalam CompareText.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionString : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifSame;  public string CompareText;  public UnityEvent TrueEvents;  public UnityEvent FalseEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifSame)  {  if (PlayerPrefs.GetString(CurrentVariable, "") == CompareText)  {  TrueEvents?.Invoke();  }  if (PlayerPrefs.GetString(CurrentVariable, "") != CompareText)  {  FalseEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionString adalah skrip yang digunakan untuk memeriksa nilai string yang disimpan dalam PlayerPrefs. Skrip ini memungkinkan Anda untuk mengatur kondisi berdasarkan apakah string dalam PlayerPrefs dengan kunci yang diberikan (CurrentVariable) sama dengan string yang Anda tentukan dalam CompareText.

### 10.6 Condition Time

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionTime : MonoBehaviour  {  [Header("Variable Settings")]  public CalculationTime CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public int LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public int EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public int BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (CurrentVariable.CurrentValue < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (CurrentVariable.CurrentValue == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (CurrentVariable.CurrentValue > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionTime adalah skrip yang digunakan untuk memeriksa nilai waktu (dalam bentuk CalculationTime) dan menerapkan kondisi berdasarkan waktu yang berlalu. Skrip ini memungkinkan Anda untuk mengatur kondisi berdasarkan apakah waktu yang berlalu kurang dari, sama dengan, atau lebih besar dari nilai yang Anda tentukan dalam LessThanCondition, EqualCondition, atau BiggerThanCondition.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  public class ConditionTimePref : MonoBehaviour  {  [Header("Variable Settings")]  public string CurrentVariable;  public bool Activated;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Condition Settings")]  public bool ifLessThan;  public float LessThanCondition;  public UnityEvent LessThanEvents;  public bool ifEqual;  public float EqualCondition;  public UnityEvent EqualEvents;  public bool ifBiggerThan;  public float BiggerThanCondition;  public UnityEvent BiggerThanEvents;  public void StartChecking()  {  Activated = true;  }  public void StopChecking()  {  Activated = false;  }  void ConditionChecking()  {  if (Activated)  {  if (ifLessThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) < LessThanCondition)  {  LessThanEvents?.Invoke();  }  }  if (ifEqual)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) == EqualCondition)  {  EqualEvents?.Invoke();  }  }  if (ifBiggerThan)  {  if (PlayerPrefs.GetFloat(CurrentVariable, 0) > BiggerThanCondition)  {  BiggerThanEvents?.Invoke();  }  }  }  }  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  if (Activated)  {  StartChecking();  }  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  ConditionChecking();  }  } |
| --- |

Skrip ConditionTimePref adalah skrip yang serupa dengan skrip ConditionTime, tetapi berfokus pada penggunaan PlayerPrefs untuk menyimpan dan memeriksa nilai waktu. Ini memungkinkan Anda untuk memeriksa kondisi waktu yang disimpan dalam PlayerPrefs dan menerapkan logika yang sama dengan ConditionTime.

# 

# 

# 

# Chapter 6 Mechanic Classification

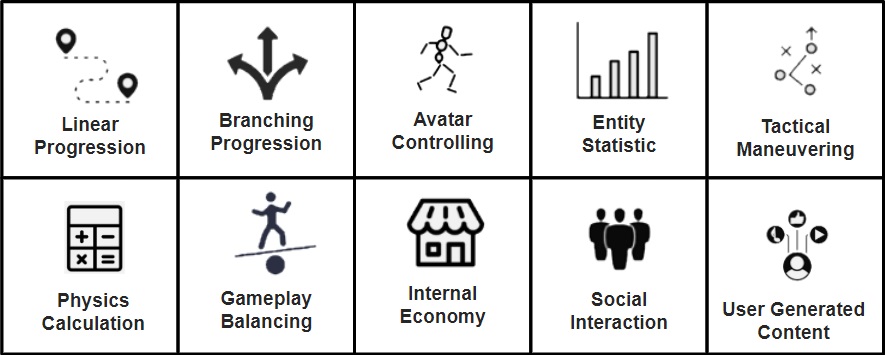
Game Mechanic Classification tidak hanya menjelaskan teori dari masing-masing mekanik game tetapi juga memberikan contoh praktis dari setiap mekanik. Contohnya, untuk Linear Progression, bisa dijelaskan bagaimana dalam game klasik seperti 'Super Mario', pemain harus menyelesaikan setiap level dalam urutan tertentu untuk maju ke akhir game. Sedangkan untuk Branching Progression, game seperti 'Mass Effect' bisa menjadi contoh dimana keputusan yang diambil pemain mempengaruhi jalannya cerita, dan hasil dari game tersebut. Untuk Avatar Controlling, game 'The Legend of Zelda' adalah contoh klasik, di mana pemain mengendalikan avatar Link melalui berbagai petualangan.

Mekanik game lain yang dibahas adalah Entity Statistic dan Tactical Maneuvering. Dalam Entity Statistic, game 'World of Warcraft' dapat dijadikan contoh, di mana setiap karakter memiliki statistik seperti kekuatan, kecerdasan, dan kesehatan yang mempengaruhi kemampuan mereka dalam pertempuran. Sedangkan untuk Tactical Maneuvering, 'StarCraft' adalah contoh yang bagus, di mana pemain harus merencanakan dan menjalankan strategi mereka dengan cermat untuk mengalahkan musuh. Physics Calculation adalah mekanik lain yang penting, terutama untuk game yang ingin menciptakan pengalaman yang realistis. Sebagai contoh, dalam game 'Angry Birds', hukum fisika seperti gravitasi dan momentum diterapkan untuk menentukan bagaimana burung akan terbang melalui udara dan bagaimana mereka akan berinteraksi dengan struktur dan musuh.

Selanjutnya, Gameplay Balancing dan Internal Economy adalah dua aspek penting lainnya dari pengembangan game. Gameplay Balancing memastikan bahwa semua aspek game, dari tingkat kesulitan musuh hingga kekuatan item, seimbang sehingga game tersebut menantang namun adil. Sebagai contoh, dalam game 'League of Legends', tim pengembang secara teratur mengupdate game untuk memastikan bahwa tidak ada karakter yang terlalu kuat atau terlalu lemah. Sementara itu, Internal Economy berhubungan dengan pengelolaan sumber daya dalam game. Sebagai contoh, dalam game 'SimCity', pemain harus mengelola sumber daya seperti uang, pekerja, dan material untuk membangun dan mengelola kota mereka.

Dua mekanik terakhir yang dibahas dalam adalah Social Interaction dan User Generated Content. Social Interaction adalah mekanik yang memungkinkan pemain untuk berinteraksi satu sama lain dalam game. Contohnya, dalam game 'World of Warcraft', pemain dapat berbicara, berdagang, dan bekerja sama untuk menyelesaikan tantangan. Sementara itu, User Generated Content adalah mekanik yang memungkinkan pemain untuk membuat dan berbagi konten mereka sendiri. Sebagai contoh, dalam game 'Minecraft', pemain dapat membuat struktur, item, dan dunia mereka sendiri dan berbagi dengan pemain lain. Kedua mekanik ini sangat penting dalam era digital saat ini, di mana bermain game online dengan orang lain dan berbagi konten adalah bagian penting dari pengalaman bermain game.

Game Mechanic Classification memberikan panduan mendalam tentang sepuluh mekanik game yang penting, dengan contoh dari game populer dan petunjuk tentang bagaimana mengimplementasikannya dalam pengembangan game. Ini adalah sumber daya yang sangat berharga bagi para pengembang game, baik yang baru memulai atau yang lebih berpengalaman, yang ingin memahami lebih lanjut tentang bagaimana mekanik game berfungsi dan bagaimana menggunakannya untuk menciptakan pengalaman bermain game yang menarik dan menantang.

****

**Gambar 7. Game Mechanic Classification**

## 1. Linear Progression

Mechanic game 'Linear Progression' berarti bahwa pemain harus menyelesaikan tugas atau misi dalam urutan tertentu untuk maju dalam game. Dari segi pemrograman, ini bisa diwujudkan dengan membuat serangkaian 'states' atau 'levels' yang harus diselesaikan oleh pemain sebelum mereka bisa pindah ke yang berikutnya. Sebagai contoh, dalam game platformer, pemain mungkin harus menyelesaikan level 1 sebelum mereka dapat mengakses level 2. Pemrograman ini melibatkan pengecekan apakah semua syarat untuk menyelesaikan suatu level atau tugas telah terpenuhi, seperti mengalahkan semua musuh atau mengumpulkan semua item, sebelum memungkinkan pemain untuk pindah ke level atau tugas berikutnya. Ini dapat dicapai dengan menggunakan variabel yang menyimpan informasi tentang prestasi pemain, dan dengan memprogram 'gates' atau 'locks' dalam game yang hanya akan terbuka setelah pemain memenuhi syarat tertentu. Package lengkap dapat di download pada link: [1. Linear Progression - Google Drive](https://drive.google.com/drive/u/0/folders/1d2wgdkORSJfNym33sqqosiajZ1T__tD2)

### 1.1 Splash Screen

Splash screen adalah gambar atau grafis yang muncul ketika pemain pertama kali menjalankan sebuah game sebelum masuk ke tampilan utama permainan. Dalam konteks pengembangan game menggunakan Unity, splash screen sering digunakan untuk menampilkan logo pengembang atau penerbit game, serta memberikan kesan pertama kepada pemain. Unity menyediakan alat untuk dengan mudah membuat dan mengatur splash screen, termasuk kemampuan untuk menampilkan gambar atau animasi selama beberapa detik sebelum permainan dimulai, yang dapat membantu dalam membangun identitas merek dan memberikan pengalaman awal yang menarik kepada pemain. Beberapa contoh video Splash Screen bisa anda temukan di sini: [Video Splash Screen](https://drive.google.com/drive/u/2/folders/1-AIgOmLafX8s5FKs_ZyBEu6vetsXc3iH)

| **Cara Membuat Splash Screen Di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/U0tz21MM2zs> |

### 1.2 Main Menu

Main menu adalah tampilan awal yang dilihat oleh pemain ketika mereka memulai sebuah game, di mana pemain dapat memilih untuk memulai permainan, mengatur pengaturan, atau menjelajahi opsi lain yang tersedia dalam game tersebut. Dalam konteks pengembangan game menggunakan Unity, main menu adalah komponen penting yang dapat dibuat dengan menggunakan elemen-elemen antarmuka pengguna seperti tombol, gambar, dan teks. Unity menyediakan alat yang memudahkan pengembang untuk merancang dan mengatur main menu, serta menghubungkannya dengan berbagai fitur dalam game, seperti pilihan level, pengaturan, atau pemilihan karakter, sehingga pemain dapat dengan mudah mengakses berbagai aspek permainan.

| **Cara Membuat Game Menu Di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/cmds2WUtnV8> |

| **Cara Membuat Pengaturan Bahasa (Versi Zetcil)** |
| --- |
| **Tutorial:** <https://youtu.be/oOb0E_NSsLg> |

### 1.3 Cutscene Animation

Cutscene animation dalam game development adalah adegan animasi yang digunakan untuk menggerakkan cerita, mengungkap karakter, atau memberikan informasi kepada pemain. Mereka dapat berupa pre-rendered video dengan kualitas grafis tinggi atau real-time yang terintegrasi dengan permainan, dengan pengembangan yang melibatkan skrip, animasi karakter, voice acting, dan elemen audiovisual untuk meningkatkan pengalaman pemain sesuai dengan genre permainan yang dituju.

| **Cara Record Animasi di Unity (Part 1) x Cube** |
| --- |
| **Tutorial:** <https://youtu.be/K701CJ06I2k> |

| **Cara Record Animasi di Unity (Part 2) x Pro Builder** |
| --- |
| **Tutorial:** <https://youtu.be/SWFsdaqrAV8> |

| **Cara Record Animasi di Unity (Part 3) x Asset Store** |
| --- |
| **Tutorial:** <https://youtu.be/vT9kUg4gNSg> |

### 1.4 Folks Simulation

Folks simulation adalah sebuah fitur dalam pengembangan game yang bertujuan untuk memberikan kesan kota atau dunia dalam game yang lebih hidup dan autentik. Fitur ini memungkinkan para pengembang game untuk mengisi dunia virtual dengan karakter, aktivitas, dan interaksi yang menciptakan nuansa kehidupan yang lebih nyata. Dengan folks simulation, pengalaman bermain game menjadi lebih mendalam karena pemain dapat merasakan berbagai aktivitas sehari-hari seperti pekerjaan, perjalanan, dan interaksi sosial dalam dunia virtual, sehingga meningkatkan daya tarik dan kedalaman dari pengalaman bermain game tersebut.

| **Cara Membuat Folks Simulation di Unity (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/eB0gn-yu9gY> |

| **Cara Membuat Folks Simulation di Unity (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/17H8Q_36ffI> |

| **Cara Membuat Folks Simulation di Unity (Part 3)** |
| --- |
| **Tutorial:** <https://youtu.be/ifcLGrPASYM> |

| **Cara Membuat Folks Simulation di Unity (Part 4)** |
| --- |
| **Tutorial:** <https://youtu.be/cssDD45YWYU> |

### 1.5 Environment Visualization

Environment Visualization dalam Unity adalah upaya untuk menciptakan lingkungan game yang realistis dan menarik melalui penggunaan terrain untuk menciptakan topografi yang kompleks, efek angin untuk memberikan pergerakan alamiah pada objek, pengaturan permukaan air untuk air yang realistis, dan efek api yang dramatis. Semua ini dapat diintegrasikan dengan pencahayaan, material, dan tekstur yang tepat untuk meningkatkan kualitas visual lingkungan dalam game dan memberikan pengalaman bermain yang lebih menarik bagi pemain. Berikut ini adalah link package pendukung pembuatan environment pada Unity yang dikumpulkan dari berbagai sumber: [Environment Package](https://drive.google.com/drive/u/2/folders/1szRjtie5VE2KuNw1uGorP3wjvIrjqCh_)

| **Cara Membuat Terrain di Unity (Part 1) : Terrain, Pohon, Rumput, Angin** |
| --- |
| **Tutorial:** <https://youtu.be/DhqBX3SHXbE> |

| **Cara Membuat Terrain di Unity (Part 2) : Air, Hujan** |
| --- |
| **Tutorial:** <https://youtu.be/9yTnhFRTRaY> |

| **Cara Membuat Terrain di Unity (Part 3) : Api** |
| --- |
| **Tutorial:** <https://youtu.be/J0d0Qhb_-M0> |

### 1.6 Cinematic Movie Mode

Cinematic Movie Mode dalam game adalah fitur yang sering ditemui dalam permainan video modern yang memungkinkan pemain untuk merasakan pengalaman visual yang lebih dramatis dan sinematik. Mode ini mengubah tampilan permainan dengan menghadirkan sudut kamera yang lebih artistik, pergerakan kamera yang lebih dinamis, dan penggunaan efek-efek visual yang mengesankan. Dengan Cinematic Movie Mode, pemain dapat merasakan adegan-adegan penting dalam permainan dengan kualitas produksi yang tinggi, menghasilkan momen-momen epik dan berkesan yang sering kali membuat permainan menjadi lebih mendalam dan imersif.

| **Cara Membuat Cinematic Movie (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/gI6fsOG3Vfc> |

| **Cara Membuat Cinematic Movie (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/lMllq2EDmkM> |

## 2. Branching Progression

Mechanic game 'Branching Progression' mengacu pada struktur progresi game di mana pemain dapat membuat pilihan yang mempengaruhi hasil atau jalannya cerita. Dalam hal pemrograman, ini bisa menjadi cukup kompleks karena memerlukan penciptaan berbagai cabang atau jalur yang berbeda yang dapat diambil oleh pemain, dan logika yang sesuai untuk mengelola keputusan pemain dan konsekuensinya. Ini mungkin melibatkan penciptaan 'pohon keputusan' atau struktur data serupa untuk mengelola semua kemungkinan jalur dan hasil, dan logika pemrograman untuk mengubah keadaan game berdasarkan keputusan yang dibuat oleh pemain. Sebagai contoh, jika dalam sebuah game petualangan pemain harus memilih untuk membantu atau menyerang karakter non-pemain (NPC), pemrograman harus menangani semua hasil yang mungkin dari keputusan tersebut, seperti mendapatkan bantuan dari NPC di kemudian hari, atau harus menghadapi mereka sebagai musuh. Package lengkap dapat di download pada link: [2. Branching Progression - Google Drive](https://drive.google.com/drive/u/0/folders/17n_MEVqnbD6CVc_8XbEImVE3n9Utk3HY)

### 2.1 Word Map Navigation

|  |
| --- |

### 2.2 Level Selection

|  |
| --- |

### 2.3 Visual Novel Dialog

Pembuatan visual novel dialog di Unity melibatkan penggunaan berbagai komponen dan script untuk menyusun narasi interaktif dalam permainan. Pertama, Anda perlu membuat karakter dan latar belakang dengan bantuan grafis 2D atau 3D, yang akan digunakan dalam cerita visual novel Anda. Selanjutnya, Anda dapat menggunakan sistem UI Unity untuk membuat jendela dialog dan tombol interaktif yang memungkinkan pemain untuk memilih dialog atau pilihan cerita. Scripting dalam Unity digunakan untuk mengatur alur cerita, perubahan karakter, dan animasi yang akan diperlihatkan saat dialog berlangsung. Selain itu, Anda juga bisa menggabungkan suara dan musik untuk meningkatkan pengalaman pemain. Dengan menggabungkan elemen-elemen ini, Anda dapat menciptakan visual novel yang menarik dan mengundang pemain untuk terlibat dalam cerita dengan berbagai pilihan dan hasil yang berbeda.

| **Cara Membuat Visual Novel Dialog (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/7Fdyh1913wU> |

| **Cara Membuat Visual Novel Dialog (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/XL5Ei9Rb9CQ> |

| **Cara Membuat Visual Novel Dialog (Part 3)** |
| --- |
| **Tutorial:** <https://youtu.be/2zA-N7B3x4Y> |

## 3. Avatar Controlling

Avatar adalah karakter digital (umumnya humanoid) yang dapat dimanipulasi oleh pemain untuk mewakili diri mereka dalam dunia virtual permainan. Avatar ini dapat disesuaikan dengan penampilan dan fungsinya dalam permainan, memungkinkan pemain untuk berinteraksi dengan lingkungan permainan dan pemain lainnya sesuai dengan aturan yang berlaku, menjadikannya elemen penting dalam pengalaman bermain game. Unity mendukung beberapa jenis format Avatar yaitu:

1. **PNG** atau singkatan dari Portable Network Graphics dalam konteks sprite game adalah format file gambar yang sering digunakan untuk menyimpan grafis sprite atau gambar karakter, objek, atau elemen lain dalam permainan. PNG memiliki keunggulan dalam mendukung latar belakang transparan, yang memungkinkan sprite untuk tampak rapi dan terlihat baik saat ditempatkan di atas berbagai latar belakang permainan tanpa adanya kotak atau latar belakang yang terlihat. Ini membuat PNG menjadi pilihan populer untuk menyimpan dan menggunakan sprite dalam pengembangan game. PNG juga memiliki kompresi tanpa kehilangan, yang mempertahankan kualitas gambar tanpa mengorbankan detail visual.
2. **FBX** atau singkatan dari "Filmbox," yang merupakan format file yang digunakan untuk pertukaran data 3D antara berbagai perangkat lunak dan aplikasi dalam industri pembuatan film, permainan, desain, dan animasi. Format file FBX mendukung berbagai jenis data 3D, termasuk geometri, tekstur, animasi, dan informasi lainnya, yang memungkinkan para profesional untuk mengimpor dan mengekspor proyek 3D mereka antara berbagai platform dan perangkat lunak dengan lebih mudah. Contoh file FBX: [Avatar - Google Drive](https://drive.google.com/drive/u/0/folders/17fxJOfOnHLHk_QoL33JWdxxbZawqIiKZ)

| **Cara Membuat Karakter FBX Menggunakan Blender** |
| --- |
| **Tutorial (Harusnya Gitu):** <https://youtu.be/LuYEBXxuSFU> |

| **Cara Membuat Animasi Karakter FBX di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/SnyAtzgzCsc> |

1. **VRM** adalah singkatan dari "Virtual Reality Modeling Language" atau "Virtual Reality Markup Language." Ini adalah bahasa markah (markup language) yang digunakan untuk membuat konten dan lingkungan virtual dalam dunia VR (Virtual Reality). VRM memungkinkan pembuat konten untuk mendefinisikan objek-objek 3D, interaksi, tekstur, dan elemen-elemen lainnya dalam lingkungan virtual. Ini memainkan peran penting dalam pengembangan aplikasi VR, simulasi, dan permainan yang memungkinkan pengguna untuk berinteraksi dan menjelajahi dunia virtual secara realistis. VRM biasanya digunakan bersama dengan platform pengembangan VR seperti Unity atau Unreal Engine untuk menciptakan pengalaman VR yang imersif. Contoh file VRM: [Avatar - Google Drive](https://drive.google.com/drive/u/0/folders/17fxJOfOnHLHk_QoL33JWdxxbZawqIiKZ) untuk menjalankan file VRM di Unity membutuhkan plugin: [VRM - Google Drive](https://drive.google.com/drive/u/0/folders/1qXNAuzB0589p5h4VH1RHo6tk2865LiH8)

| **Cara Membuat Karakter VRM Menggunakan VRoid Studio** |
| --- |
| **Tutorial:** <https://youtu.be/Bu41FcF27pA> |

| **Cara Membuat Animasi Karakter VRM di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/wbpM8AdWA4U> |

1. **GLB** adalah format file yang digunakan dalam grafika komputer dan pengembangan 3D. Ini adalah singkatan dari "**Binary GLTF**" atau "Binary Graphics Language Transmission Format." Format GLB menggabungkan file GLTF (Graphics Language Transmission Format) dan semua sumber daya terkait seperti tekstur, model 3D, dan animasi ke dalam satu file biner tunggal. Format ini sering digunakan dalam pengembangan permainan, simulasi, dan aplikasi VR/AR karena memungkinkan penyimpanan dan pertukaran data 3D yang efisien dan mudah diintegrasikan ke dalam berbagai platform dan lingkungan pengembangan. GLB biasanya digunakan dalam kombinasi dengan platform pengembangan seperti Unity atau Three.js. Contoh file GLB: [Avatar - Google Drive](https://drive.google.com/drive/u/0/folders/17fxJOfOnHLHk_QoL33JWdxxbZawqIiKZ) untuk menjalankan file GLB/GLTF di Unity membutuhkan plugin: [GLTF - Google Drive](https://drive.google.com/drive/u/0/folders/1BCRRUi37pj6xXcnNVTRTCwOnEK4UkUlW)

Mechanic game 'Avatar Controlling' mengacu pada kemampuan pemain untuk mengendalikan karakter atau avatar dalam game. Pemrograman untuk kontrol avatar melibatkan beberapa aspek berbeda, termasuk input pemain, fisika game, dan animasi karakter. Input pemain, seperti pergerakan mouse, ketukan keyboard, atau gerakan gamepad, harus diinterpretasikan dan diterjemahkan menjadi gerakan atau tindakan avatar dalam game. Ini melibatkan pemrograman untuk mendeteksi input, mengubahnya menjadi perintah yang sesuai, dan kemudian memperbarui keadaan avatar dalam game. Fisika game juga harus dipertimbangkan, seperti gravitasi, gesekan, dan deteksi tabrakan, untuk memastikan bahwa gerakan avatar terlihat realistis dan berinteraksi dengan dunia game dengan cara yang benar. Akhirnya, animasi karakter harus disinkronkan dengan gerakan dan tindakan avatar, yang melibatkan pemrograman untuk memilih dan memainkan animasi yang sesuai berdasarkan keadaan dan tindakan avatar saat ini. Package lengkap dapat didownload pada link berikut: [3. Avatar Controlling - Google Drive](https://drive.google.com/drive/u/0/folders/18MVqZkfTwz8_fM-Fy0AHgBVEB8uHFf9y)

Khusus untuk avatar bertipe humanoid terdapat beberapa web services yang memiliki prosedur khusus untuk dapat menggunakan karakter mereka. Salah satunya adalah **ReadyPlayerMe**. Untuk menggunakan karakter Ready Player Me terdapat beberapa langkah yang harus anda lakukan yaitu:

1. Install GIT: <https://git-scm.com/>
2. Add GIT via Unity Package Manager:

<https://github.com/readyplayerme/rpm-unity-sdk-core.git#v1.1.0>

| **Cara Membuat Karakter GLB Menggunakan Ready Player Me** |
| --- |
| **Tutorial:** <https://youtu.be/S-v4z8q63b8> |

| **Cara Membuat Animasi Karakter GLB di Unity** |
| --- |
| **Tutorial (Versi Gagal):** <https://youtu.be/nVD17eZqQmA> |

### 

### 3.1 Platformer Controller

Platformer adalah genre permainan video di mana pemain mengendalikan karakter yang berusaha untuk melompat dan bergerak melewati serangkaian platform, rintangan, dan tantangan dalam lingkungan permainan. Dalam pengembangan game menggunakan Unity, platformer sering melibatkan penggunaan fisika yang realistis untuk mengatur perilaku karakter dan platform, serta animasi karakter yang halus untuk menghidupkan gerakan melompat dan berlari. Unity menyediakan alat-alat yang kuat untuk membuat lingkungan platformer yang interaktif dan menarik, serta memungkinkan pengembang untuk dengan mudah mengatur tingkat permainan dan mekanik permainan lainnya.

| **Unity Technology** |
| --- |
| **Platformer :** <https://assetstore.unity.com/packages/templates/tutorials/2d-game-kit-107098> |

| **Cara Membuat Pergerakan Platformer (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/ETAY1TLdCpc> |

| **Cara Membuat Pergerakan Platformer (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/edsVEHXT4XU> |

#### Cole

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /\*  \* Created by: Joshua Cole  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody2D))]  [RequireComponent(typeof(CapsuleCollider2D))]  public class PlatformerCole : MonoBehaviour  {  // Move player in 2D space  [Header("Target Camera")]  public Camera mainCamera;  [Header("Movement Settings")]  public float maxSpeed = 3.4f;  public float jumpHeight = 6.5f;  public float gravityScale = 1.5f;  bool facingRight = true;  float moveDirection = 0;  bool isGrounded = false;  Vector3 cameraPos;  Rigidbody2D r2d;  CapsuleCollider2D mainCollider;  Transform t;  // Use this for initialization  void Start()  {  t = transform;  r2d = GetComponent<Rigidbody2D>();  mainCollider = GetComponent<CapsuleCollider2D>();  r2d.freezeRotation = true;  r2d.collisionDetectionMode = CollisionDetectionMode2D.Continuous;  r2d.gravityScale = gravityScale;  facingRight = t.localScale.x > 0;  if (mainCamera)  {  cameraPos = mainCamera.transform.position;  }  }  // Update is called once per frame  void Update()  {  // Movement controls  if ((isGrounded || Mathf.Abs(r2d.velocity.x) > 0.01f))  {  moveDirection = Input.GetAxis("Horizontal");  }  else  {  if (isGrounded || r2d.velocity.magnitude < 0.01f)  {  moveDirection = 0;  }  }  // Change facing direction  if (moveDirection != 0)  {  if (moveDirection > 0 && !facingRight)  {  facingRight = true;  t.localScale = new Vector3(Mathf.Abs(t.localScale.x), t.localScale.y, transform.localScale.z);  }  if (moveDirection < 0 && facingRight)  {  facingRight = false;  t.localScale = new Vector3(-Mathf.Abs(t.localScale.x), t.localScale.y, t.localScale.z);  }  }  // Jumping  if (Input.GetKeyDown(KeyCode.Space) && isGrounded)  {  r2d.velocity = new Vector2(r2d.velocity.x, jumpHeight);  }  // Camera follow  if (mainCamera)  {  mainCamera.transform.position = new Vector3(t.position.x, cameraPos.y, cameraPos.z);  }  }  void FixedUpdate()  {  Bounds colliderBounds = mainCollider.bounds;  float colliderRadius = mainCollider.size.x \* 0.4f \* Mathf.Abs(transform.localScale.x);  Vector3 groundCheckPos = colliderBounds.min + new Vector3(colliderBounds.size.x \* 0.5f, colliderRadius \* 0.9f, 0);  // Check if player is grounded  Collider2D[] colliders = Physics2D.OverlapCircleAll(groundCheckPos, colliderRadius);  //Check if any of the overlapping colliders are not player collider, if so, set isGrounded to true  isGrounded = false;  if (colliders.Length > 0)  {  for (int i = 0; i < colliders.Length; i++)  {  if (colliders[i] != mainCollider)  {  isGrounded = true;  break;  }  }  }  // Apply movement velocity  r2d.velocity = new Vector2((moveDirection) \* maxSpeed, r2d.velocity.y);  // Simple debug  Debug.DrawLine(groundCheckPos, groundCheckPos - new Vector3(0, colliderRadius, 0), isGrounded ? Color.green : Color.red);  Debug.DrawLine(groundCheckPos, groundCheckPos - new Vector3(colliderRadius, 0, 0), isGrounded ? Color.green : Color.red);  }  } |
| --- |

#### Zagorskiy

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /\*  \* Created by: Artyom Zagorskiy  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody2D))]  [RequireComponent(typeof(CapsuleCollider2D))]  public class PlatformerZagorskiy : MonoBehaviour  {  [Header("Main Settings Settings")]  public Animator animator;  Rigidbody2D rigidbody;  [Header("Movement Settings")]  public string animatorState = "animatorState";  public float movingSpeed;  public float jumpForce;  private float moveInput;  private bool facingRight = false;  [Header("Ground Settings")]  public Transform groundCheck;  public float groundOverlap = 5;  public bool isGrounded;  void Start()  {  rigidbody = GetComponent<Rigidbody2D>();  }  private void FixedUpdate()  {  CheckGround();  }  void Update()  {  if (Input.GetButton("Horizontal"))  {  moveInput = Input.GetAxis("Horizontal");  Vector3 direction = transform.right \* moveInput;  transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, movingSpeed \* Time.deltaTime);  animator.SetInteger(animatorState, 1); // Turn on run animation  }  else  {  if (isGrounded) animator.SetInteger(animatorState, 0); // Turn on idle animation  }  if (Input.GetKeyDown(KeyCode.Space) && isGrounded)  {  GetComponent<Rigidbody2D>().AddForce(transform.up \* jumpForce, ForceMode2D.Impulse);  }  if (!isGrounded) animator.SetInteger(animatorState, 2); // Turn on jump animation  if (facingRight == false && moveInput > 0)  {  Flip();  }  else if (facingRight == true && moveInput < 0)  {  Flip();  }  }  private void Flip()  {  facingRight = !facingRight;  Vector3 Scaler = transform.localScale;  Scaler.x \*= -1;  transform.localScale = Scaler;  }  private void CheckGround()  {  Collider2D[] colliders = Physics2D.OverlapCircleAll(groundCheck.transform.position, groundOverlap);  isGrounded = colliders.Length > 1;  }  } |
| --- |

### 3.2 Top Down Shooter Controller

Top-down shooter adalah genre permainan video di mana pemain mengendalikan karakter dari sudut pandang atas, yang berarti pemain melihat karakter dan lingkungan dari atas ke bawah. Dalam konteks pengembangan game menggunakan Unity, top-down shooter biasanya melibatkan penggunaan kamera yang diposisikan di atas permainan dan karakter yang bisa diarahkan ke segala arah. Pemain akan menghadapi berbagai musuh dan mengumpulkan senjata atau item yang berbeda dalam permainan ini. Unity menyediakan alat-alat yang memungkinkan pengembang untuk mengatur mekanik permainan, seperti peluru, gerakan karakter, dan interaksi dengan lingkungan, sehingga menciptakan pengalaman top-down shooter yang seru.

#### Cole

| using UnityEngine;  /\*  \* Created by: Joshua Cole  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody))]  [RequireComponent(typeof(CapsuleCollider))]  public class TopDownCole : MonoBehaviour  {  //Player Camera variables  public enum CameraDirection { x, z }  public CameraDirection cameraDirection = CameraDirection.x;  public float cameraHeight = 20f;  public float cameraDistance = 7f;  public Camera playerCamera;  public GameObject targetIndicatorPrefab;  //Player Controller variables  public float speed = 5.0f;  public float gravity = 14.0f;  public float maxVelocityChange = 10.0f;  public bool canJump = true;  public float jumpHeight = 2.0f;  //Private variables  bool grounded = false;  Rigidbody r;  GameObject targetObject;  //Mouse cursor Camera offset effect  Vector2 playerPosOnScreen;  Vector2 cursorPosition;  Vector2 offsetVector;  //Plane that represents imaginary floor that will be used to calculate Aim target position  Plane surfacePlane = new Plane();  void Awake()  {  r = GetComponent<Rigidbody>();  r.freezeRotation = true;  r.useGravity = false;  //Instantiate aim target prefab  if (targetIndicatorPrefab)  {  targetObject = Instantiate(targetIndicatorPrefab, Vector3.zero, Quaternion.identity) as GameObject;  }  //Hide the cursor  Cursor.visible = false;  }  void FixedUpdate()  {  //Setup camera offset  Vector3 cameraOffset = Vector3.zero;  if (cameraDirection == CameraDirection.x)  {  cameraOffset = new Vector3(cameraDistance, cameraHeight, 0);  }  else if (cameraDirection == CameraDirection.z)  {  cameraOffset = new Vector3(0, cameraHeight, cameraDistance);  }  if (grounded)  {  Vector3 targetVelocity = Vector3.zero;  // Calculate how fast we should be moving  if (cameraDirection == CameraDirection.x)  {  targetVelocity = new Vector3(Input.GetAxis("Vertical") \* (cameraDistance >= 0 ? -1 : 1), 0, Input.GetAxis("Horizontal") \* (cameraDistance >= 0 ? 1 : -1));  }  else if (cameraDirection == CameraDirection.z)  {  targetVelocity = new Vector3(Input.GetAxis("Horizontal") \* (cameraDistance >= 0 ? -1 : 1), 0, Input.GetAxis("Vertical") \* (cameraDistance >= 0 ? -1 : 1));  }  targetVelocity \*= speed;  // Apply a force that attempts to reach our target velocity  Vector3 velocity = r.velocity;  Vector3 velocityChange = (targetVelocity - velocity);  velocityChange.x = Mathf.Clamp(velocityChange.x, -maxVelocityChange, maxVelocityChange);  velocityChange.z = Mathf.Clamp(velocityChange.z, -maxVelocityChange, maxVelocityChange);  velocityChange.y = 0;  r.AddForce(velocityChange, ForceMode.VelocityChange);  // Jump  if (canJump && Input.GetButton("Jump"))  {  r.velocity = new Vector3(velocity.x, CalculateJumpVerticalSpeed(), velocity.z);  }  }  // We apply gravity manually for more tuning control  r.AddForce(new Vector3(0, -gravity \* r.mass, 0));  grounded = false;  //Mouse cursor offset effect  playerPosOnScreen = playerCamera.WorldToViewportPoint(transform.position);  cursorPosition = playerCamera.ScreenToViewportPoint(Input.mousePosition);  offsetVector = cursorPosition - playerPosOnScreen;  //Camera follow  playerCamera.transform.position = Vector3.Lerp(playerCamera.transform.position, transform.position + cameraOffset, Time.deltaTime \* 7.4f);  playerCamera.transform.LookAt(transform.position + new Vector3(-offsetVector.y \* 2, 0, offsetVector.x \* 2));  //Aim target position and rotation  targetObject.transform.position = GetAimTargetPos();  targetObject.transform.LookAt(new Vector3(transform.position.x, targetObject.transform.position.y, transform.position.z));  //Player rotation  transform.LookAt(new Vector3(targetObject.transform.position.x, transform.position.y, targetObject.transform.position.z));  }  Vector3 GetAimTargetPos()  {  //Update surface plane  surfacePlane.SetNormalAndPosition(Vector3.up, transform.position);  //Create a ray from the Mouse click position  Ray ray = playerCamera.ScreenPointToRay(Input.mousePosition);  //Initialise the enter variable  float enter = 0.0f;  if (surfacePlane.Raycast(ray, out enter))  {  //Get the point that is clicked  Vector3 hitPoint = ray.GetPoint(enter);  //Move your cube GameObject to the point where you clicked  return hitPoint;  }  //No raycast hit, hide the aim target by moving it far away  return new Vector3(-5000, -5000, -5000);  }  void OnCollisionStay()  {  grounded = true;  }  float CalculateJumpVerticalSpeed()  {  // From the jump height and gravity we deduce the upwards speed  // for the character to reach at the apex.  return Mathf.Sqrt(2 \* jumpHeight \* gravity);  }  } |
| --- |

### 3.3 First Person Controller

First Person Shooter (FPS) atau First Person Perspective (FPP) adalah jenis permainan video yang memungkinkan pemain mengalami dunia permainan melalui sudut pandang karakter utama, yang berarti pemain melihat dunia permainan seperti melalui mata karakter tersebut. Dalam FPS, pemain sering menggunakan senjata untuk berinteraksi dengan lingkungan dan musuh, menjadikannya genre yang fokus pada pertempuran dan strategi. Keunggulan dari sudut pandang orang pertama adalah memberikan imersi yang tinggi dan pengalaman yang mendalam dalam permainan, membuat pemain merasa seolah-olah mereka benar-benar berada di dalam dunia permainan.

| **Unity Technology** |
| --- |
| **First Person Shooter Controller (FPS):**  <https://assetstore.unity.com/packages/essentials/starter-assets-first-person-character-controller-urp-196525> |

| **Zetcil Framework** |
| --- |
| **Avatar Controlling:**  <https://drive.google.com/drive/u/2/folders/18MVqZkfTwz8_fM-Fy0AHgBVEB8uHFf9y> |

| **Cara Membuat Pergerakan First Person** |
| --- |
| **Tutorial:** <https://youtu.be/sLjpuQnaiKY> |

#### Bon3head

| using UnityEngine;  /\*  \* Created by: Liam B0N3head  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody))]  public class FirstPersonBonhead : MonoBehaviour  {  [Header("Camera Settings")]  public GameObject playerCamera;  //----------------------------------------------------  [Space]  [Header("Mouse Settings")]  public bool lockCursor = true;  public Vector2 clampInDegrees = new Vector2(360f, 180f);  public Vector2 sensitivity = new Vector2(2f, 2f);  [Header("Movement Settings")]  public Vector2 smoothing = new Vector2(1.5f, 1.5f);  public float walkMoveSpeed = 7.5f;  public float sprintMoveSpeed = 11f;  public float jumpMoveSpeed = 6f;  public float crouchMoveSpeed = 4f;  //----------------------------------------------------  [Header("Crouch Settings")]  public float crouchDownSpeed = 0.2f;  public float crouchHeight = 0.68f; //change for how large you want when crouching  public float standingHeight = 1f;  public bool smoothCrouch = true;  public bool jumpCrouching = true;  //----------------------------------------------------  [Header("Jump Settings")]  public float jumpForce = 10f;  public float jumpAccel = 10f;  public float jumpTime = 0.4f;  public float coyoteTime = 0.2f;  public float jumpBuffer = 0.1f;  public float jumpCooldown = 0.6f;  public float extraGravity = 0.1f;  public string groundTag = "Ground";  //----------------------------------------------------  [Space]  [Header("Keyboard Settings")]  public KeyCode jump = KeyCode.Space;  public KeyCode sprint = KeyCode.LeftShift;  public KeyCode crouch = KeyCode.Z;  public KeyCode lockToggle = KeyCode.Q;  //----------------------------------------------------  [Space]  [Header("Debug Info")]  public bool areWeGrounded = true;  public bool areWeCrouching = false;  public float currentSpeed;  //----------------------------------------------------  // Reference vars (These are the vars used in calculations, they don't need to be set by the user)  private Rigidbody rb;  Vector3 input = new Vector3();  Vector2 \_mouseAbsolute, \_smoothMouse, targetDirection, targetCharacterDirection;  private float coyoteTimeCounter, jumpBufferCounter, startJumpTime, endJumpTime;  private bool wantingToJump = false, wantingToCrouch = false, wantingToSprint = false, jumpCooldownOver = true;  void Awake()  {  // Just set rb to the rigidbody of the gameobject containing this script  rb = gameObject.GetComponent<Rigidbody>();  // Set the currentSpeed to walking as no keys should be pressed yet  currentSpeed = walkMoveSpeed;  // Set target direction to the camera's initial orientation.  targetDirection = transform.localRotation.eulerAngles;  // Set target direction for the character body to its inital state.  targetCharacterDirection = transform.localRotation.eulerAngles;  }  private void Update()  {  // Update the camera pos  cameraUpdate();  // Move all input to Update(), then use given input on FixedUpdate()  // WSAD movement  input = new Vector3(Input.GetAxisRaw("Horizontal"), 0, Input.GetAxisRaw("Vertical"));  // Jump key  wantingToJump = Input.GetKey(jump);  // Crouch key  wantingToCrouch = Input.GetKey(crouch);  // Sprint key  wantingToSprint = Input.GetKey(sprint);  // Mouse lock toggle (KeyDown only fires once)  if (Input.GetKeyDown(lockToggle))  lockCursor = !lockCursor;  }  public void cameraUpdate()  {  // Allow the script to clamp based on a desired target value.  var targetOrientation = Quaternion.Euler(targetDirection);  var targetCharacterOrientation = Quaternion.Euler(targetCharacterDirection);  // Get raw mouse input for a cleaner reading on more sensitive mice.  var mouseDelta = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));  // Scale input against the sensitivity setting and multiply that against the smoothing value.  mouseDelta = Vector2.Scale(mouseDelta, new Vector2(sensitivity.x \* smoothing.x, sensitivity.y \* smoothing.y));  // Interpolate mouse movement over time to apply smoothing delta.  \_smoothMouse.x = Mathf.Lerp(\_smoothMouse.x, mouseDelta.x, 1f / smoothing.x);  \_smoothMouse.y = Mathf.Lerp(\_smoothMouse.y, mouseDelta.y, 1f / smoothing.y);  // Find the absolute mouse movement value from point zero.  \_mouseAbsolute += \_smoothMouse;  // Clamp and apply the local x value first, so as not to be affected by world transforms.  if (clampInDegrees.x < 360)  \_mouseAbsolute.x = Mathf.Clamp(\_mouseAbsolute.x, -clampInDegrees.x \* 0.5f, clampInDegrees.x \* 0.5f);  // Then clamp and apply the global y value.  if (clampInDegrees.y < 360)  \_mouseAbsolute.y = Mathf.Clamp(\_mouseAbsolute.y, -clampInDegrees.y \* 0.5f, clampInDegrees.y \* 0.5f);  playerCamera.transform.localRotation = Quaternion.AngleAxis(-\_mouseAbsolute.y, targetOrientation \* Vector3.right) \* targetOrientation;  var yRotation = Quaternion.AngleAxis(\_mouseAbsolute.x, Vector3.up);  transform.localRotation = yRotation \* targetCharacterOrientation;  }  void FixedUpdate()  {  // Lock cursor handling  if (lockCursor)  Cursor.lockState = CursorLockMode.Locked;  else  Cursor.lockState = CursorLockMode.None;  // Double check if we are on the ground or not (Changes current speed if true)  // --- QUICK EXPLINATION ---  // transform.position.y - transform.localScale.y + 0.1f  // This puts the start of the ray 0.1f above the bottom of the player  // We then shoot a ray 0.15f down, this exists the player with 0.5f to hit objects  // Removing this +- of 0.1f and having it shoot directly under the player can skip the ground as sometimes the capsules bottom clips through the ground  if (Physics.Raycast(new Vector3(transform.position.x, transform.position.y - transform.localScale.y + 0.1f, transform.position.z), Vector3.down, 0.15f))  handleHitGround();  // Sprinting  if (wantingToSprint && areWeGrounded && !areWeCrouching)  currentSpeed = sprintMoveSpeed;  else if (!areWeCrouching && areWeGrounded)  currentSpeed = walkMoveSpeed;  // Crouching  // Can be simplified to Crouch((wantingToCrouch && jumpCrouching)); though the bellow is more readable  if (wantingToCrouch && jumpCrouching)  Crouch(true);  else  Crouch(false);  // Coyote timer (When the player leaves the ground, start counting down from the set value coyoteTime)  // This allows players to jump late. After they have left  if (areWeGrounded)  coyoteTimeCounter = coyoteTime;  else  coyoteTimeCounter -= Time.deltaTime;  // Jump buffer timer (When the player leaves the ground, start counting down from the set value jumpBuffer)  // This will "buffer" the input and allow for early space presses to be valid and no longer ignored  if (wantingToJump)  jumpBufferCounter = jumpBuffer;  else  jumpBufferCounter -= Time.deltaTime;  // If the coyote timer has not run out and our jump buffer has not run out and we our cool down (canJump) is now over  if (coyoteTimeCounter > 0f && jumpBufferCounter > 0f && jumpCooldownOver)  {  rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);  rb.AddForce(transform.up \* jumpForce, ForceMode.Impulse);  jumpCooldownOver = false;  areWeGrounded = false;  jumpBufferCounter = 0f;  currentSpeed = jumpMoveSpeed;  endJumpTime = Time.time + jumpTime;  // Wait jumpCooldown (1f = 1 second) then run the jumpCoolDownCountdown() void  Invoke(nameof(jumpCoolDownCountdown), jumpCooldown);  }  else if (wantingToJump && !areWeGrounded && endJumpTime > Time.time)  {  // Hold down space for a further jump (until the timer runs out)  rb.AddForce(Vector3.up \* jumpAccel, ForceMode.Acceleration);  }  // WSAD movement  input = input.normalized;  Vector3 forwardVel = transform.forward \* currentSpeed \* input.z;  Vector3 horizontalVel = transform.right \* currentSpeed \* input.x;  rb.velocity = horizontalVel + forwardVel + new Vector3(0, rb.velocity.y, 0);  //Extra gravity for more nicer jumping  rb.AddForce(new Vector3(0, -extraGravity, 0), ForceMode.Impulse);  }  private void jumpCoolDownCountdown()  {  jumpCooldownOver = true;  }  // Crouch handling  private void Crouch(bool crouch)  {  areWeCrouching = crouch;  if (crouch)  {  // If the player is crouching  currentSpeed = crouchMoveSpeed;  if (smoothCrouch)  {  transform.localScale = new Vector3(transform.localScale.x, Mathf.Lerp(transform.localScale.y, crouchHeight, crouchDownSpeed), transform.localScale.z);  transform.position = Vector3.Lerp(transform.position, new Vector3(transform.position.x, transform.position.y - crouchHeight, transform.position.z), crouchDownSpeed);  }  else if (transform.localScale != new Vector3(transform.localScale.x, crouchHeight, transform.localScale.z))  {  transform.localScale = new Vector3(transform.localScale.x, crouchHeight, transform.localScale.z);  transform.position = new Vector3(transform.position.x, transform.position.y - crouchHeight / 2, transform.position.z);  }  }  else  {  // If the player is standing  if (smoothCrouch)  {  transform.localScale = new Vector3(transform.localScale.x, Mathf.Lerp(transform.localScale.y, standingHeight, crouchDownSpeed), transform.localScale.z);  transform.position = Vector3.Lerp(transform.position, new Vector3(transform.position.x, transform.position.y - standingHeight / 2, transform.position.z), crouchDownSpeed);  }  else if (transform.localScale != new Vector3(transform.localScale.x, standingHeight, transform.localScale.z))  {  transform.localScale = new Vector3(transform.localScale.x, standingHeight, transform.localScale.z);  transform.position = new Vector3(transform.position.x, transform.position.y + standingHeight / 2, transform.position.z);  }  }  }  // Ground check  //\*\*\*\*\*\* make sure whatever you want to be the ground in your game matches the tag set in the script  private void OnCollisionEnter(Collision other)  {  if (other.gameObject.tag == groundTag)  handleHitGround();  }  // This is separated in its own void as this code needs to be run on two separate occasions, saves copy pasting code  // Just double checking if we are crouching and setting the speed accordingly  public void handleHitGround()  {  if (areWeCrouching)  currentSpeed = crouchMoveSpeed;  else  currentSpeed = walkMoveSpeed;  areWeGrounded = true;  }  // Dw about understanding this, it's just the code for setting up the player character  public void setupCharacter()  {  gameObject.tag = "Player";  if (!gameObject.GetComponent<Rigidbody>())  {  Rigidbody rb = gameObject.AddComponent(typeof(Rigidbody)) as Rigidbody;  rb.mass = 10;  }  else Debug.Log("Rigidbody already exists");  if (!gameObject.transform.Find("Camera"))  {  Vector3 old = transform.position;  gameObject.transform.position = new Vector3(0, -0.8f, 0);  GameObject go = new GameObject("Camera");  go.AddComponent<Camera>();  go.AddComponent<AudioListener>();  go.transform.rotation = new Quaternion(0, 0, 0, 0);  go.transform.localScale = new Vector3(1, 1, 1);  go.transform.parent = transform;  gameObject.transform.position = old;  Debug.Log("Camera created");  }  else Debug.Log("Camera already exists");  }  } |
| --- |

#### 

#### ChatGPT

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.EventSystems;  /\*  \* Created by: ChatGPT  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(CharacterController))]  public class FirstPersonChatGPT : MonoBehaviour  {  public enum InputType  {  Mouse,  Touch,  Both  }  [Header("Camera Setting")]  public Camera playerCamera;  public InputType inputType = InputType.Mouse;  private int activeTouchID = -1; //  [Header("Movement Setting")]  public float walkingSpeed = 7.5f;  public float runningSpeed = 11.5f;  public float jumpSpeed = 8.0f;  public float gravity = 20.0f;  public float lookSpeed = 2.0f;  public float lookXLimit = 45.0f;  [Header("Cursor Setting")]  public bool CursorVisible;  public bool CursorLocked;  bool isTouching = false;  CharacterController characterController;  Vector3 moveDirection = Vector3.zero;  float rotationX = 0;  [HideInInspector]  public bool canMove = true;  void Start()  {  characterController = GetComponent<CharacterController>();  // Lock cursor  if (CursorLocked)  {  Cursor.lockState = CursorLockMode.Locked;  } else  {  Cursor.lockState = CursorLockMode.None;  }  Cursor.visible = CursorVisible;  }  void Update()  {  // We are grounded, so recalculate move direction based on axes  Vector3 forward = transform.TransformDirection(Vector3.forward);  Vector3 right = transform.TransformDirection(Vector3.right);  // Press Left Shift to run  bool isRunning = Input.GetKey(KeyCode.LeftShift);  float curSpeedX = canMove ? (isRunning ? runningSpeed : walkingSpeed) \* Input.GetAxis("Vertical") : 0;  float curSpeedY = canMove ? (isRunning ? runningSpeed : walkingSpeed) \* Input.GetAxis("Horizontal") : 0;  float movementDirectionY = moveDirection.y;  moveDirection = (forward \* curSpeedX) + (right \* curSpeedY);  if (Input.GetKeyDown(KeyCode.Space) && canMove && characterController.isGrounded)  {  moveDirection.y = jumpSpeed;  }  else  {  moveDirection.y = movementDirectionY;  }  // Apply gravity. Gravity is multiplied by deltaTime twice (once here, and once below  // when the moveDirection is multiplied by deltaTime). This is because gravity should be applied  // as an acceleration (ms^-2)  if (!characterController.isGrounded)  {  moveDirection.y -= gravity \* Time.deltaTime;  }  // Move the controller  characterController.Move(moveDirection \* Time.deltaTime);  // Input dari mouse  if (canMove && (inputType == InputType.Mouse || inputType == InputType.Both))  {  rotationX += -Input.GetAxis("Mouse Y") \* lookSpeed;  rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);  playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);  transform.rotation \*= Quaternion.Euler(0, Input.GetAxis("Mouse X") \* lookSpeed, 0);  }  if (inputType == InputType.Touch || inputType == InputType.Both)  {  // Cek apakah ada setidaknya satu sentuhan  if (Input.touchCount > 0)  {  // Loop melalui semua sentuhan yang ada  for (int i = 0; i < Input.touchCount; i++)  {  Touch touch = Input.GetTouch(i);  // Cek apakah sentuhan berada di atas elemen UI  bool isOverUI = EventSystem.current.IsPointerOverGameObject(touch.fingerId);  // Jika tidak ada sentuhan yang aktif dan sentuhan ini tidak berada di atas elemen UI, buat sentuhan ini menjadi sentuhan yang aktif  if (activeTouchID == -1 && !isOverUI)  {  activeTouchID = touch.fingerId;  }  // Jika sentuhan ini adalah sentuhan yang aktif, lakukan rotasi  if (touch.fingerId == activeTouchID)  {  if (touch.phase == TouchPhase.Began)  {  isTouching = true;  }  else if (touch.phase == TouchPhase.Moved && isTouching)  {  // Menghitung perubahan posisi sentuhan  float touchDeltaX = touch.deltaPosition.x;  float touchDeltaY = touch.deltaPosition.y;  // Menggunakan perubahan posisi untuk mengubah rotasi  rotationX += -touchDeltaY \* lookSpeed \* Time.deltaTime;  rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);  playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);  transform.rotation \*= Quaternion.Euler(0, touchDeltaX \* lookSpeed \* Time.deltaTime, 0);  }  else if (touch.phase == TouchPhase.Ended)  {  isTouching = false;  // Reset sentuhan yang aktif  activeTouchID = -1;  }  }  }  }  }  }  } |
| --- |

#### Cole

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /\*  \* Created by: Joshua Cole  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(CharacterController))]  public class FirstPersonCole : MonoBehaviour  {  [Header("Camera Settings")]  public Camera playerCamera;  [Header("Movement Settings")]  public float walkingSpeed = 7.5f;  public float runningSpeed = 11.5f;  public float jumpSpeed = 8.0f;  public float gravity = 20.0f;  public float lookSpeed = 2.0f;  public float lookXLimit = 45.0f;  CharacterController characterController;  Vector3 moveDirection = Vector3.zero;  float rotationX = 0;  [HideInInspector]  public bool canMove = true;  void Start()  {  characterController = GetComponent<CharacterController>();  // Lock cursor  Cursor.lockState = CursorLockMode.Locked;  Cursor.visible = false;  }  void Update()  {  // We are grounded, so recalculate move direction based on axes  Vector3 forward = transform.TransformDirection(Vector3.forward);  Vector3 right = transform.TransformDirection(Vector3.right);  // Press Left Shift to run  bool isRunning = Input.GetKey(KeyCode.LeftShift);  float curSpeedX = canMove ? (isRunning ? runningSpeed : walkingSpeed) \* Input.GetAxis("Vertical") : 0;  float curSpeedY = canMove ? (isRunning ? runningSpeed : walkingSpeed) \* Input.GetAxis("Horizontal") : 0;  float movementDirectionY = moveDirection.y;  moveDirection = (forward \* curSpeedX) + (right \* curSpeedY);  if (Input.GetButton("Jump") && canMove && characterController.isGrounded)  {  moveDirection.y = jumpSpeed;  }  else  {  moveDirection.y = movementDirectionY;  }  // Apply gravity. Gravity is multiplied by deltaTime twice (once here, and once below  // when the moveDirection is multiplied by deltaTime). This is because gravity should be applied  // as an acceleration (ms^-2)  if (!characterController.isGrounded)  {  moveDirection.y -= gravity \* Time.deltaTime;  }  // Move the controller  characterController.Move(moveDirection \* Time.deltaTime);  // Player and Camera rotation  if (canMove)  {  rotationX += -Input.GetAxis("Mouse Y") \* lookSpeed;  rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);  playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);  transform.rotation \*= Quaternion.Euler(0, Input.GetAxis("Mouse X") \* lookSpeed, 0);  }  }  } |
| --- |

#### 

#### Devy

| using System;  using UnityEngine;  /\*  \* Created by: Dani Devy  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody))]  public class FirstPersonDevy : MonoBehaviour  {  [Header("Camera Settings")]  public Transform playerCamera;  public Transform orientation;  //Other  private Rigidbody rb;  //Rotation and look  private float xRotation;  private float sensitivity = 50f;  private float sensMultiplier = 1f;  [Header("Movement Settings")]  //Movement  public float moveSpeed = 4500;  public float maxSpeed = 20;  public bool grounded;  public LayerMask whatIsGround;  public float counterMovement = 0.175f;  private float threshold = 0.01f;  public float maxSlopeAngle = 35f;  //Crouch & Slide  private Vector3 crouchScale = new Vector3(1, 0.5f, 1);  private Vector3 playerScale;  public float slideForce = 400;  public float slideCounterMovement = 0.2f;  //Jumping  private bool readyToJump = true;  private float jumpCooldown = 0.25f;  public float jumpForce = 100f;  //Input  float x, y;  bool jumping, sprinting, crouching;  //Sliding  private Vector3 normalVector = Vector3.up;  private Vector3 wallNormalVector;  void Awake()  {  rb = GetComponent<Rigidbody>();  }  void Start()  {  playerScale = transform.localScale;  Cursor.lockState = CursorLockMode.Locked;  Cursor.visible = false;  }  private void FixedUpdate()  {  Movement();  }  private void Update()  {  MyInput();  Look();  }  /// <summary>  /// Find user input. Should put this in its own class but im lazy  /// </summary>  private void MyInput()  {  x = Input.GetAxisRaw("Horizontal");  y = Input.GetAxisRaw("Vertical");  jumping = Input.GetButton("Jump");  crouching = Input.GetKey(KeyCode.LeftControl);  //Crouching  if (Input.GetKeyDown(KeyCode.LeftControl))  StartCrouch();  if (Input.GetKeyUp(KeyCode.LeftControl))  StopCrouch();  }  private void StartCrouch()  {  transform.localScale = crouchScale;  transform.position = new Vector3(transform.position.x, transform.position.y - 0.5f, transform.position.z);  if (rb.velocity.magnitude > 0.5f)  {  if (grounded)  {  rb.AddForce(orientation.transform.forward \* slideForce);  }  }  }  private void StopCrouch()  {  transform.localScale = playerScale;  transform.position = new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z);  }  private void Movement()  {  //Extra gravity  rb.AddForce(Vector3.down \* Time.deltaTime \* 10);  //Find actual velocity relative to where player is looking  Vector2 mag = FindVelRelativeToLook();  float xMag = mag.x, yMag = mag.y;  //Counteract sliding and sloppy movement  CounterMovement(x, y, mag);  //If holding jump && ready to jump, then jump  if (readyToJump && jumping) Jump();  //Set max speed  float maxSpeed = this.maxSpeed;  //If sliding down a ramp, add force down so player stays grounded and also builds speed  if (crouching && grounded && readyToJump)  {  rb.AddForce(Vector3.down \* Time.deltaTime \* 3000);  return;  }  //If speed is larger than maxspeed, cancel out the input so you don't go over max speed  if (x > 0 && xMag > maxSpeed) x = 0;  if (x < 0 && xMag < -maxSpeed) x = 0;  if (y > 0 && yMag > maxSpeed) y = 0;  if (y < 0 && yMag < -maxSpeed) y = 0;  //Some multipliers  float multiplier = 1f, multiplierV = 1f;  // Movement in air  if (!grounded)  {  multiplier = 0.5f;  multiplierV = 0.5f;  }  // Movement while sliding  if (grounded && crouching) multiplierV = 0f;  //Apply forces to move player  rb.AddForce(orientation.transform.forward \* y \* moveSpeed \* Time.deltaTime \* multiplier \* multiplierV);  rb.AddForce(orientation.transform.right \* x \* moveSpeed \* Time.deltaTime \* multiplier);  }  private void Jump()  {  if (grounded && readyToJump)  {  readyToJump = false;  //Add jump forces  rb.AddForce(Vector2.up \* jumpForce \* 1.5f);  rb.AddForce(normalVector \* jumpForce \* 0.5f);  //If jumping while falling, reset y velocity.  Vector3 vel = rb.velocity;  if (rb.velocity.y < 0.5f)  rb.velocity = new Vector3(vel.x, 0, vel.z);  else if (rb.velocity.y > 0)  rb.velocity = new Vector3(vel.x, vel.y / 2, vel.z);  Invoke(nameof(ResetJump), jumpCooldown);  }  }  private void ResetJump()  {  readyToJump = true;  }  private float desiredX;  private void Look()  {  float mouseX = Input.GetAxis("Mouse X") \* sensitivity \* Time.fixedDeltaTime \* sensMultiplier;  float mouseY = Input.GetAxis("Mouse Y") \* sensitivity \* Time.fixedDeltaTime \* sensMultiplier;  //Find current look rotation  Vector3 rot = playerCamera.transform.localRotation.eulerAngles;  desiredX = rot.y + mouseX;  //Rotate, and also make sure we dont over- or under-rotate.  xRotation -= mouseY;  xRotation = Mathf.Clamp(xRotation, -90f, 90f);  //Perform the rotations  playerCamera.transform.localRotation = Quaternion.Euler(xRotation, desiredX, 0);  orientation.transform.localRotation = Quaternion.Euler(0, desiredX, 0);  }  private void CounterMovement(float x, float y, Vector2 mag)  {  if (!grounded || jumping) return;  //Slow down sliding  if (crouching)  {  rb.AddForce(moveSpeed \* Time.deltaTime \* -rb.velocity.normalized \* slideCounterMovement);  return;  }  //Counter movement  if (Math.Abs(mag.x) > threshold && Math.Abs(x) < 0.05f || (mag.x < -threshold && x > 0) || (mag.x > threshold && x < 0))  {  rb.AddForce(moveSpeed \* orientation.transform.right \* Time.deltaTime \* -mag.x \* counterMovement);  }  if (Math.Abs(mag.y) > threshold && Math.Abs(y) < 0.05f || (mag.y < -threshold && y > 0) || (mag.y > threshold && y < 0))  {  rb.AddForce(moveSpeed \* orientation.transform.forward \* Time.deltaTime \* -mag.y \* counterMovement);  }  //Limit diagonal running. This will also cause a full stop if sliding fast and un-crouching, so not optimal.  if (Mathf.Sqrt((Mathf.Pow(rb.velocity.x, 2) + Mathf.Pow(rb.velocity.z, 2))) > maxSpeed)  {  float fallspeed = rb.velocity.y;  Vector3 n = rb.velocity.normalized \* maxSpeed;  rb.velocity = new Vector3(n.x, fallspeed, n.z);  }  }  public Vector2 FindVelRelativeToLook()  {  float lookAngle = orientation.transform.eulerAngles.y;  float moveAngle = Mathf.Atan2(rb.velocity.x, rb.velocity.z) \* Mathf.Rad2Deg;  float u = Mathf.DeltaAngle(lookAngle, moveAngle);  float v = 90 - u;  float magnitue = rb.velocity.magnitude;  float yMag = magnitue \* Mathf.Cos(u \* Mathf.Deg2Rad);  float xMag = magnitue \* Mathf.Cos(v \* Mathf.Deg2Rad);  return new Vector2(xMag, yMag);  }  private bool IsFloor(Vector3 v)  {  float angle = Vector3.Angle(Vector3.up, v);  return angle < maxSlopeAngle;  }  private bool cancellingGrounded;  /// <summary>  /// Handle ground detection  /// </summary>  private void OnCollisionStay(Collision other)  {  //Make sure we are only checking for walkable layers  int layer = other.gameObject.layer;  if (whatIsGround != (whatIsGround | (1 << layer))) return;  //Iterate through every collision in a physics update  for (int i = 0; i < other.contactCount; i++)  {  Vector3 normal = other.contacts[i].normal;  //FLOOR  if (IsFloor(normal))  {  grounded = true;  cancellingGrounded = false;  normalVector = normal;  CancelInvoke(nameof(StopGrounded));  }  }  //Invoke ground/wall cancel, since we can't check normals with CollisionExit  float delay = 3f;  if (!cancellingGrounded)  {  cancellingGrounded = true;  Invoke(nameof(StopGrounded), Time.deltaTime \* delay);  }  }  private void StopGrounded()  {  grounded = false;  }  } |
| --- |

#### Vagsether

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /\*  \* Created by: Hakon Vagsether  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(Rigidbody))]  public class FirstPersonVagsether : MonoBehaviour  {  [Header("Camera Settings")]  public Transform playerCamera;  [Header("Movement Settings")]  public float walkSpeed = 10.0f;  public float jumpForce = 25.0f;  public float mouseSensitivityX = 1.0f;  public float mouseSensitivityY = 1.0f;  public float lookXLimit = 60.0f;  Vector3 moveAmount;  Vector3 smoothMoveVelocity;  float verticalLookRotation;  Rigidbody rigidbodyR;  bool grounded;  public LayerMask groundedMask;  bool cursorVisible;  // Use this for initialization  void Start()  {  rigidbodyR = GetComponent<Rigidbody>();  LockMouse();  }  // Update is called once per frame  void Update()  {  // rotation  transform.Rotate(Vector3.up \* Input.GetAxis("Mouse X") \* mouseSensitivityX);  verticalLookRotation += Input.GetAxis("Mouse Y") \* mouseSensitivityY;  verticalLookRotation = Mathf.Clamp(verticalLookRotation, -lookXLimit, lookXLimit);  playerCamera.localEulerAngles = Vector3.left \* verticalLookRotation;    // movement  Vector3 moveDir = new Vector3(Input.GetAxisRaw("Horizontal"), 0, Input.GetAxisRaw("Vertical")).normalized;  Vector3 targetMoveAmount = moveDir \* walkSpeed;  moveAmount = Vector3.SmoothDamp(moveAmount, targetMoveAmount, ref smoothMoveVelocity, .15f);    // jump  if (Input.GetButtonDown("Jump"))  {  if (grounded)  {  rigidbodyR.AddForce(transform.up \* jumpForce \* 10);  }  }    Ray ray = new Ray(transform.position, -transform.up);  RaycastHit hit;    if (Physics.Raycast(ray, out hit, 1 + .1f, groundedMask))  {  grounded = true;  }  else  {  grounded = false;  }    /\* Lock/unlock mouse on click \*/  if (Input.GetMouseButtonUp(0))  {  if (!cursorVisible)  {  UnlockMouse();  }  else  {  LockMouse();  }  }  }    void FixedUpdate()  {  rigidbodyR.MovePosition(rigidbodyR.position + transform.TransformDirection(moveAmount) \* Time.fixedDeltaTime);  }    void UnlockMouse()  {  Cursor.lockState = CursorLockMode.None;  Cursor.visible = true;  cursorVisible = true;  }    void LockMouse()  {  Cursor.lockState = CursorLockMode.Locked;  Cursor.visible = false;  cursorVisible = false;  }  } |
| --- |

### 3.4 Third Person Controller

Third person shooter adalah genre permainan video di mana pemain mengendalikan karakter dari sudut pandang pihak ketiga, yang berarti karakter terlihat dari belakang atau samping. Dalam konteks pengembangan game menggunakan Unity, third person shooter biasanya melibatkan penggunaan kamera yang dapat diatur untuk mengikuti karakter pemain dari belakang atau samping, sambil memberikan pemain kontrol atas senjata dan gerakan karakter. Pemain biasanya berinteraksi dengan lingkungan, berperang melawan musuh, dan menyelesaikan misi dalam permainan ini. Unity menyediakan berbagai alat dan fitur untuk memudahkan pengembangan game third person shooter, termasuk sistem animasi, fisika, dan pengelolaan karakter.

| **Unity Technology** |
| --- |
| **Third Person Controller :**  <https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-urp-196526> |

| **Zetcil Framework** |
| --- |
| **Avatar Controlling:**  <https://drive.google.com/drive/u/2/folders/18MVqZkfTwz8_fM-Fy0AHgBVEB8uHFf9y> |

| **Cara Membuat Pergerakan Third Person (FBX)** |
| --- |
| **Tutorial:** <https://youtu.be/pcopsgOXlPU> |

| **Cara Membuat Pergerakan Third Person (VRM)** |
| --- |
| **Tutorial:** <https://youtu.be/zrA6ZTqRlUg> |

| **Cara Membuat Pergerakan Third Person (GLB)** |
| --- |
| **Tutorial:** <https://youtu.be/yhJSAR12IqU> |

#### Cole

| using UnityEngine;  /\*  \* Created by: Joshua Cole  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(CharacterController))]  public class ThirdPersonCole : MonoBehaviour  {  [Header("Camera Settings")]  public Transform CameraParent;  [Header("Movement Settings")]  public float speed = 7.5f;  public float jumpSpeed = 8.0f;  public float gravity = 20.0f;  public float lookSpeed = 2.0f;  public float lookXLimit = 60.0f;  CharacterController characterController;  Vector3 moveDirection = Vector3.zero;  Vector2 rotation = Vector2.zero;  [HideInInspector]  public bool canMove = true;  void Start()  {  characterController = GetComponent<CharacterController>();  rotation.y = transform.eulerAngles.y;  }  void Update()  {  if (characterController.isGrounded)  {  // We are grounded, so recalculate move direction based on axes  Vector3 forward = transform.TransformDirection(Vector3.forward);  Vector3 right = transform.TransformDirection(Vector3.right);  float curSpeedX = canMove ? speed \* Input.GetAxis("Vertical") : 0;  float curSpeedY = canMove ? speed \* Input.GetAxis("Horizontal") : 0;  moveDirection = (forward \* curSpeedX) + (right \* curSpeedY);  if (Input.GetButton("Jump") && canMove)  {  moveDirection.y = jumpSpeed;  }  }  // Apply gravity. Gravity is multiplied by deltaTime twice (once here, and once below  // when the moveDirection is multiplied by deltaTime). This is because gravity should be applied  // as an acceleration (ms^-2)  moveDirection.y -= gravity \* Time.deltaTime;  // Move the controller  characterController.Move(moveDirection \* Time.deltaTime);  // Player and Camera rotation  if (canMove)  {  rotation.y += Input.GetAxis("Mouse X") \* lookSpeed;  rotation.x += -Input.GetAxis("Mouse Y") \* lookSpeed;  rotation.x = Mathf.Clamp(rotation.x, -lookXLimit, lookXLimit);  CameraParent.localRotation = Quaternion.Euler(rotation.x, 0, 0);  transform.eulerAngles = new Vector2(0, rotation.y);  }  }  } |
| --- |

| using UnityEngine;  /\*  \* Created by: Joshua Cole  \* Modified by: Rickman Roedavan  \*/  public class ThirdCameraCole : MonoBehaviour  {  [Header("Camera Settings")]  public float collisionOffset = 0.3f; //To prevent Camera from clipping through Objects  public float cameraSpeed = 15f; //How fast the Camera should snap into position if there are no obstacles  Transform CameraParent;  Vector3 defaultPos;  Vector3 directionNormalized;  Transform parentTransform;  float defaultDistance;  // Start is called before the first frame update  void Start()  {  CameraParent = transform.parent;  defaultPos = transform.localPosition;  directionNormalized = defaultPos.normalized;  parentTransform = transform.parent;  defaultDistance = Vector3.Distance(defaultPos, Vector3.zero);  //Lock cursor  Cursor.lockState = CursorLockMode.Locked;  Cursor.visible = false;  }  // LateUpdate is called after Update  void LateUpdate()  {  Vector3 currentPos = defaultPos;  RaycastHit hit;  Vector3 dirTmp = parentTransform.TransformPoint(defaultPos) - CameraParent.position;  if (Physics.SphereCast(CameraParent.position, collisionOffset, dirTmp, out hit, defaultDistance))  {  currentPos = (directionNormalized \* (hit.distance - collisionOffset));  transform.localPosition = currentPos;  }  else  {  transform.localPosition = Vector3.Lerp(transform.localPosition, currentPos, Time.deltaTime \* cameraSpeed);  }  }  } |
| --- |

#### Debdas

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /\*  \* Created by: Seemanta Debdas  \* Modified by: Rickman Roedavan  \*/  [RequireComponent(typeof(CharacterController))]  public class ThirdPersonDebdas : MonoBehaviour  {  [Header("Movement Settings")]  [SerializeField] float speed;  [SerializeField] float rotationSmoothTime;  [SerializeField] float gravity = 9.8f;  [SerializeField] float gravityMultiplier = 2;  [SerializeField] float groundedGravity = -0.5f;  [SerializeField] float jumpHeight = 3f;  float velocityY;  CharacterController controller;  float currentAngle;  float currentAngleVelocity;  private void Awake()  {  //getting reference for components on the Player  controller = GetComponent<CharacterController>();  }  private void Update()  {  HandleMovement();  HandleGravityAndJump();  }  private void HandleMovement()  {  //capturing Input from Player  Vector3 movement = new Vector3(Input.GetAxisRaw("Horizontal"), 0, Input.GetAxisRaw("Vertical")).normalized;  if (movement.magnitude >= 0.1f)  {  //compute rotation  float targetAngle = Mathf.Atan2(movement.x, movement.z) \* Mathf.Rad2Deg;  currentAngle = Mathf.SmoothDampAngle(currentAngle, targetAngle, ref currentAngleVelocity, rotationSmoothTime);  transform.rotation = Quaternion.Euler(0, currentAngle, 0);  //move in direction of rotation  Vector3 rotatedMovement = Quaternion.Euler(0, targetAngle, 0) \* Vector3.forward;  controller.Move(rotatedMovement \* speed \* Time.deltaTime);  }  }  void HandleGravityAndJump()  {  //apply groundedGravity when the Player is Grounded  if (controller.isGrounded && velocityY < 0f)  velocityY = groundedGravity;  //When Grounded and Jump Button is Pressed, set veloctiyY with the formula below  if (controller.isGrounded && Input.GetKeyDown(KeyCode.Space))  {  velocityY = Mathf.Sqrt(jumpHeight \* 2f \* gravity);  }  //applying gravity when Player is not grounded  velocityY -= gravity \* gravityMultiplier \* Time.deltaTime;  controller.Move(Vector3.up \* velocityY \* Time.deltaTime);  }  } |
| --- |

| using UnityEngine;  using System.Collections;  /\*  \* Modified for SeemantaDebdas by Rickman Roedavan  \*/  public class ThirdCameraDebdas : MonoBehaviour  {  public GameObject target;  public float rotateSpeed = 5;  Vector3 offset;  void Start()  {  offset = target.transform.position - transform.position;  }  void LateUpdate()  {  float horizontal = Input.GetAxis("Mouse X") \* rotateSpeed;  target.transform.Rotate(0, horizontal, 0);  float desiredAngle = target.transform.eulerAngles.y;  Quaternion rotation = Quaternion.Euler(0, desiredAngle, 0);  transform.position = target.transform.position - (rotation \* offset);  transform.LookAt(target.transform);  }  } |
| --- |

### 

#### Zamojc

| using UnityEngine;  using System.Collections;  /\*  \* Created by: Ian Zamojc  \* Modified by: Rickman Roedavan  \*/  public class ThirdPersonZamojc : MonoBehaviour  {  [Header("Movement Settings")]  public float movementSpeed = 10;  public float turningSpeed = 60;  void Update()  {  float horizontal = Input.GetAxis("Horizontal") \* turningSpeed \* Time.deltaTime;  transform.Rotate(0, horizontal, 0);  float vertical = Input.GetAxis("Vertical") \* movementSpeed \* Time.deltaTime;  transform.Translate(0, 0, vertical);  }  } |
| --- |

| using UnityEngine;  using System.Collections;  /\*  \* Created by: Ian Zamojc  \* Modified by: Rickman Roedavan  \*/  public class ThirdCameraZamojc : MonoBehaviour  {  [Header("Player Settings")]  public GameObject playerObject;  [Header("Movement Settings")]  public float damping = 1;  Vector3 offset;  void Start()  {  offset = playerObject.transform.position - transform.position;  }  void LateUpdate()  {  float currentAngle = transform.eulerAngles.y;  float desiredAngle = playerObject.transform.eulerAngles.y;  float angle = Mathf.LerpAngle(currentAngle, desiredAngle, Time.deltaTime \* damping);  Quaternion rotation = Quaternion.Euler(0, angle, 0);  transform.position = playerObject.transform.position - (rotation \* offset);  transform.LookAt(playerObject.transform);  }  } |
| --- |

### 3.5 Isometric Controller

Permainan isometrik adalah genre permainan video yang menggunakan tampilan isometrik, yang memberikan sudut pandang tiga dimensi pada dunia permainan, meskipun sebenarnya berjalan pada bidang datar. Dalam konteks pengembangan game menggunakan Unity, permainan isometrik memanfaatkan tampilan grafis yang memudahkan pemain untuk menjelajahi dan berinteraksi dengan lingkungan serta karakter dalam sudut pandang yang unik. Unity menyediakan alat-alat untuk menciptakan dunia isometrik, mengatur animasi karakter, dan mengendalikan logika permainan, sehingga pengembang dapat membuat pengalaman permainan yang menarik dengan perspektif isometrik yang khas.

|  |
| --- |

| **Cara Membuat Pergantian Karakter Ala Genshin Impact (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/WgWi0uTAyM0> |

| **Cara Membuat Pergantian Karakter Ala Genshin Impact (Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/0QPhl5klVYw> |

## 4. Entity Statistic

Mechanic game 'Entity Statistic' berhubungan dengan pengelolaan dan manipulasi dari berbagai statistik yang terkait dengan entitas dalam game, seperti karakter, monster, atau objek. Statistik ini bisa mencakup hal-hal seperti kesehatan, energi, kekuatan, kecepatan, dll. Pemrograman untuk pengelolaan statistik entitas melibatkan penyimpanan nilai-nilai statistik, pembaruan nilai-nilai ini berdasarkan tindakan dalam game, dan penggunaan nilai-nilai ini untuk pengambilan keputusan dalam game. Misalnya, jika sebuah karakter menerima kerusakan, nilai kesehatan karakter tersebut harus diperbarui, dan jika nilai kesehatannya mencapai nol, karakter tersebut harus dianggap mati. Pemrograman ini bisa melibatkan penggunaan struktur data untuk menyimpan nilai-nilai statistik, fungsi untuk memperbarui nilai-nilai ini, dan logika kondisional untuk mengambil keputusan berdasarkan nilai-nilai ini. Package lengkap dapat di download pada link: [4. Entity Statistic - Google Drive](https://drive.google.com/drive/u/0/folders/1tfw7n1eYwO43FWd8ORhzPsadJUF7FAkt)

### 4.1 Console Debugger

|  |
| --- |

### 4.2 Web Request

|  |
| --- |

## 5. Tactical Maneuvering

Mechanic game 'Tactical Maneuvering' berfokus pada penggunaan taktik dan manuver untuk mengatasi tantangan dalam game. Ini bisa mencakup penggunaan posisi, gerakan, dan penggunaan sumber daya atau kemampuan khusus. Dari sudut pandang pemrograman, ini bisa melibatkan penciptaan sistem yang memungkinkan pemain untuk memilih dan menggunakan berbagai taktik dan manuver, pengembangan algoritma untuk mengontrol gerakan dan perilaku entitas dalam game, dan pengembangan sistem untuk mengelola sumber daya atau kemampuan yang dapat digunakan pemain. Misalnya, jika sebuah game memungkinkan pemain untuk menggunakan manuver pengelakan untuk menghindari serangan musuh, pemrograman untuk ini bisa melibatkan pengembangan logika untuk mendeteksi dan menghindari serangan, pengembangan sistem untuk mengontrol gerakan pemain, dan pengembangan logika untuk mengelola penggunaan manuver tersebut. Package lengkap dapat di download pada link: [5. Tactical Manuevering - Google Drive](https://drive.google.com/drive/u/0/folders/18LM7xCGw_sJbqanEQTQz7CCKEEz-t9Ia)

### 5.1 Strategic Map Controller

|  |
| --- |

### 5.2 Gizmo Controller

|  |
| --- |

## 6. Physics Calculation

Mechanic game 'Physics Calculation' adalah aspek penting dari banyak game modern, terutama game yang menampilkan simulasi dunia 3D realistis. Ini mencakup perhitungan tentang bagaimana objek bergerak, bagaimana mereka bertabrakan, dan bagaimana gaya lain seperti gravitasi atau gesekan mempengaruhi mereka. Dari sudut pandang pemrograman, ini melibatkan penggunaan dan pengembangan algoritma fisika untuk menghitung perilaku objek dalam game. Misalnya, dalam game balap, perhitungan fisika digunakan untuk menghitung bagaimana kecepatan dan arah kendaraan berubah sebagai respons terhadap pengendalian pemain, gravitasi, dan gesekan dengan permukaan jalan. Ini bisa melibatkan pengembangan model matematika yang kompleks dan penggunaan pustaka fisika khusus untuk membantu melakukan perhitungan yang diperlukan. Package lengkap dapat di download pada link: [6. Physics Calculation - Google Drive](https://drive.google.com/drive/u/0/folders/19W7zQFVE5PD3ew9i0pzwWZr16cMJx7sZ)

### 6.1 Gravity System

|  |
| --- |

### 6.2 Liquid Simulation

|  |
| --- |

## 7. Gameplay Balancing

Mechanic game 'Gameplay Balancing' merupakan aspek kunci dalam pengembangan game untuk memastikan bahwa game tersebut menyenangkan dan menantang untuk dimainkan, tanpa terasa terlalu mudah atau terlalu sulit. Dari perspektif pemrograman, ini bisa melibatkan penyesuaian berbagai parameter dalam game, seperti kekuatan musuh, ketersediaan sumber daya, atau tingkat kesulitan rintangan, untuk menciptakan pengalaman bermain yang optimal. Selain itu, algoritma khusus mungkin perlu dikembangkan untuk mengadaptasi tingkat kesulitan game secara dinamis berdasarkan performa pemain. Misalnya, jika sistem mendeteksi bahwa pemain mengalami kesulitan dalam mengatasi rintangan tertentu, maka bisa saja secara otomatis menyesuaikan tingkat kesulitan rintangan tersebut untuk menjadikannya sedikit lebih mudah, atau sebaliknya, meningkatkan kesulitan jika pemain terlalu mendominasi. Package lengkap dapat di download pada link: [7. Gameplay Balancing - Google Drive](https://drive.google.com/drive/u/0/folders/1tN5Yf_2Fr1bGOBPxkDMPbjY5_yGllYjk)

### 7.1 Configuration System

|  |
| --- |

### 7.2 AI Pathfinding

|  |
| --- |

### 7.3 Enemy Patroling

|  |
| --- |

## 8. Internal Economy

'Mechanic game' yang berkaitan dengan 'Internal Economy' melibatkan simulasi dari ekonomi internal dalam dunia game, yang bisa mencakup aspek-aspek seperti pengumpulan sumber daya, perdagangan, dan pengelolaan uang atau mata uang virtual. Dari perspektif pemrograman, ini bisa melibatkan pembuatan algoritma untuk mengendalikan harganya, supply dan demand dari berbagai item, serta tingkah laku dari NPC (Non-Player Characters) terkait dengan ekonomi. Sebagai contoh, jika sebuah item menjadi langka, harga jualnya bisa meningkat, dan NPC mungkin akan mulai menawarkan quest atau misi yang berhubungan dengan item tersebut. Juga, sistem perdagangan antar pemain mungkin perlu diprogram untuk memungkinkan pertukaran barang dan mata uang secara aman dan adil. Selain itu, algoritma kecerdasan buatan mungkin diperlukan untuk mengendalikan perilaku pedagang NPC, seperti menyesuaikan harga barang berdasarkan keadaan ekonomi saat ini atau berdasarkan interaksi sebelumnya dengan pemain. Package lengkap dapat di download pada link: [8. Internal Economy - Google Drive](https://drive.google.com/drive/u/0/folders/1j5oL7ZM9GFhJf4nxggnv2_gBkK_DTEqh)

### 8.1 Inventory Controller

|  |
| --- |

### 8.2 Item Mall System

|  |
| --- |

## 9. Social Interaction

Mechanic game yang berkaitan dengan 'Social Interaction' melibatkan interaksi antara pemain dalam game, baik secara langsung maupun tidak langsung. Dari perspektif pemrograman, ini bisa melibatkan pembuatan sistem komunikasi, seperti chat atau pesan suara, sistem pertemanan atau gilda, serta sistem untuk berbagi sumber daya atau informasi. Selain itu, sistem ini juga perlu mempertimbangkan keamanan dan moderasi, untuk mencegah pelecehan dan memastikan pengalaman bermain yang positif untuk semua pemain. Pemrograman juga melibatkan pembuatan sistem untuk interaksi tidak langsung, seperti berbagi item atau informasi melalui papan pengumuman, atau berpartisipasi dalam event multiplayer besar. Selain itu, interaksi sosial juga bisa melibatkan interaksi dengan NPC, yang memerlukan algoritma kecerdasan buatan untuk mengendalikan tingkah laku dan respons mereka terhadap tindakan pemain. Package lengkap dapat di download pada link: [9. Social Interaction - Google Drive](https://drive.google.com/drive/u/0/folders/1F4nnzUb9-0gVg4Oq169IYV9L5x065fz4)

### 9.1 Visual Novel Dialog

|  |
| --- |

### 9.2 Multiplayer System

|  |
| --- |

## 10. User Generated Content

Mechanic game yang berkaitan dengan 'User Generated Content' adalah salah satu mekanik yang paling kompleks dari segi pemrograman. Ini melibatkan pembuatan alat dan antarmuka yang memungkinkan pemain untuk membuat, mengedit, dan berbagi konten mereka sendiri dalam game. Hal ini bisa meliputi level, karakter, objek, atau bahkan cerita. Pemrograman untuk mekanik ini melibatkan pengembangan editor level atau objek, sistem untuk menyimpan dan memuat konten yang dibuat pengguna, serta sistem untuk berbagi dan menilai konten tersebut. Selain itu, penting untuk memastikan bahwa konten yang dibuat pengguna tidak mengganggu keseimbangan game atau mengandung materi yang tidak pantas, jadi sistem validasi dan moderasi juga perlu diprogram. Selain itu, penting untuk memastikan bahwa konten yang dibuat pengguna dapat diintegrasikan dengan baik ke dalam game, baik dari segi gameplay maupun grafis. Package lengkap dapat di download pada link: [10. User Generated Content - Google Drive](https://drive.google.com/drive/u/0/folders/1sfkQdOT2CTZbJMhfplmWqihGPZ74M14Q)

### 10.1 Avatar System Creation

|  |
| --- |

### 10.2 Item Crafting System

|  |
| --- |

# 

# 

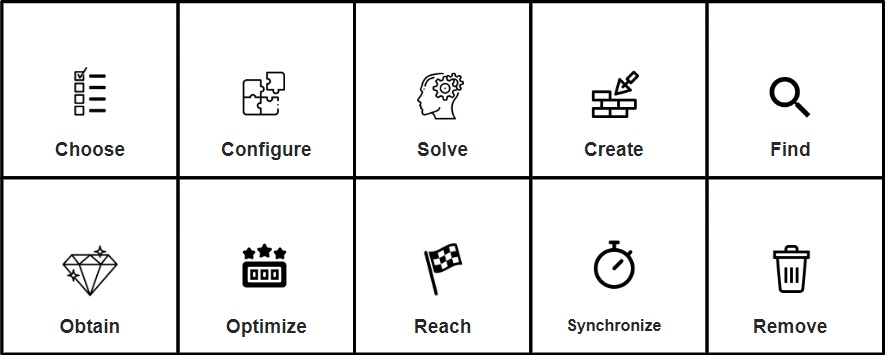
# Chapter 7 Game Goal Typology

Game Goal Typology Classification mengacu kepada gambaran tentang berbagai tujuan atau 'goals' yang umum ditemui dalam sebuah game. Tujuan-tujuan ini dikelompokkan ke dalam sepuluh kategori utama: Choose, Configure, Solve, Create, Find, Obtain, Optimize, Reach, Synchronize, dan Remove. ‘Choose', merupakan salah satu aspek yang sangat penting dalam permainan adalah kemampuan untuk membuat keputusan. Ini dapat berkisar dari keputusan sederhana seperti memilih karakter yang akan dimainkan, sampai ke keputusan yang lebih kompleks seperti memilih strategi untuk mengalahkan musuh. Dalam beberapa kasus, pilihan yang dibuat pemain dapat mempengaruhi alur cerita atau hasil dari game tersebut. Dalam pengembangan game, pemrograman untuk fitur 'Choose' dapat melibatkan implementasi logika cabang (branching logic) untuk menangani berbagai pilihan yang dibuat pemain dan menghasilkan hasil yang sesuai.

Sedangkan 'Configure', ini adalah elemen kunci dalam banyak game yang memungkinkan pemain untuk menyesuaikan aspek-aspek tertentu dari game tersebut. Ini bisa berupa pengaturan kendaraan, penyesuaian karakter, atau bahkan mengubah tata letak level. Fitur ini memungkinkan pemain untuk memiliki pengalaman yang lebih personal dan unik saat bermain game. Dari sudut pandang pengembangan, 'Configure' dapat melibatkan implementasi antarmuka pengguna yang memungkinkan pemain untuk melakukan penyesuaian, serta logika di belakang untuk menerapkan perubahan tersebut dalam game.

Sementara itu, 'Solve' merupakan komponen utama dalam banyak game teka-teki atau petualangan. Ini bisa berupa menyelesaikan teka-teki, menemukan jalan keluar dari labirin, atau menyelesaikan tantangan lain yang membutuhkan pemikiran kritis. Dalam pengembangan game, ini mungkin melibatkan penggunaan algoritma khusus untuk membuat teka-teki yang menantang atau logika untuk mengatur labirin dan jalan keluar. Pada bagian 'Create', ini adalah aspek yang sangat menarik dari banyak game modern yang memungkinkan pemain untuk membuat sesuatu yang baru dalam game, apakah itu bangunan, karya seni, atau bahkan level game sendiri. Fitur ini memungkinkan pemain untuk mengekspresikan kreativitas mereka dan berbagi kreasi mereka dengan pemain lain. Dalam pengembangan game, ini bisa melibatkan penggunaan alat pembuat (creation tools) yang memungkinkan pemain untuk merancang dan membangun kreasi mereka.

Selanjutnya, 'Find' adalah elemen yang sering muncul dalam game petualangan atau RPG. Ini bisa melibatkan mencari item tertentu, menemukan lokasi rahasia, atau menemukan karakter tertentu dalam game. Fitur ini menambahkan tingkat kedalaman dan eksplorasi ke dalam game, memberi pemain rasa pencapaian saat mereka menemukan sesuatu yang baru atau penting. Dalam pengembangan game, ini mungkin melibatkan penggunaan algoritma pencarian atau peta yang terperinci untuk membantu pemain menemukan apa yang mereka cari. 'Obtain' berfokus pada mendapatkan atau mengumpulkan benda-benda, seperti mata uang atau item kunci. 'Optimize' berkaitan dengan meningkatkan atau meningkatkan sesuatu, seperti kinerja atau statistik karakter. 'Reach' mengacu pada mencapai tujuan atau lokasi tertentu, seperti titik akhir level. 'Synchronize' melibatkan koordinasi tindakan atau pergerakan, seperti menari atau bermain musik. Dan terakhir, 'Remove' berkaitan dengan menghapus atau menghilangkan sesuatu, seperti musuh atau rintangan.

****

**Gambar 8. Game Goal Typology**

## 1. Choose

'Choose' adalah salah satu game goal typology yang sangat penting dalam permainan. Dalam hal pengembangan dan pemrograman, 'Choose' melibatkan implementasi logika cabang (branching logic) untuk menangani berbagai pilihan yang dibuat pemain dan menghasilkan hasil yang sesuai. Misalnya, dalam game RPG, pemain mungkin harus memilih untuk bersekutu dengan salah satu dari dua faksi yang bersaing. Keputusan ini bisa mempengaruhi jalan cerita, misi yang tersedia, dan bahkan hasil dari game tersebut. Dari sudut pandang pemrograman, ini bisa melibatkan penulisan kode yang memeriksa pilihan yang dibuat pemain dan kemudian memodifikasi keadaan game sesuai dengan pilihan tersebut. Ini bisa melibatkan pengubahan variabel, memulai atau menghentikan misi tertentu, atau bahkan mengubah perilaku karakter non-pemain (NPC) dalam game.

| **Cara Membuat Kuis Di Unity (Typology Choose Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/Xgjx6d6B2CY> |

| **Cara Membuat Kuis Di Unity (Typology Choose Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/e9ArG59Lw2o> |

| **Cara Membuat Kuis Di Unity (Typology Choose Part 2)** |
| --- |
| **Tutorial:** <https://youtu.be/L0lfp20Klok> |

## 2. Configure

'Configure' adalah game goal typology yang mengacu pada kemampuan pemain untuk mengubah atau mengkonfigurasi elemen dalam game, seperti karakter, item, atau lingkungan. Misalnya, pemain mungkin dapat menyesuaikan penampilan karakter mereka, mengatur strategi tim, atau membangun struktur. Dalam hal pemrograman, 'Configure' memerlukan implementasi dari berbagai antarmuka pengguna (UI) yang memungkinkan pemain untuk membuat pilihan dan membuat perubahan. Selain itu, sistem yang memungkinkan pengubahan dan pengelolaan objek perlu diprogram. Ini bisa melibatkan penggunaan sistem berbasis data untuk menyimpan dan mengambil informasi tentang objek yang dikonfigurasi, serta kode untuk mengubah properti objek tersebut secara real-time saat pemain membuat perubahan. Misalnya, jika pemain mengubah warna baju karakter mereka, sistem harus menyimpan informasi ini dan menggambarkan karakter dengan baju baru tersebut dalam game.

## 3. Solve

'Solve' dalam game goal typology merujuk pada tujuan permainan yang melibatkan pemecahan masalah atau teka-teki. Dalam pengembangan game, ini melibatkan pembuatan teka-teki atau tantangan yang menantang pemain untuk mencari solusi atau jalan keluar. Pemrograman untuk 'Solve' mungkin melibatkan pengembangan logika dan algoritma yang rumit untuk menghasilkan teka-teki yang bervariasi tingkat kesulitannya. Misalnya, dalam game teka-teki, pemrograman dapat melibatkan implementasi mekanika teka-teki, seperti mengatur urutan tindakan yang benar untuk menyelesaikan teka-teki atau mengembangkan sistem deteksi solusi yang akurat. Dalam game dengan elemen fisika, pemrograman mungkin melibatkan mengembangkan algoritma simulasi yang memungkinkan objek-objek dalam teka-teki berinteraksi dengan benar sesuai dengan hukum fisika.

## 4. Create

'Create' adalah game goal typology yang berfokus pada tujuan permainan yang memungkinkan pemain untuk merancang, membuat, dan membangun sesuatu yang baru dalam dunia game. Dalam pengembangan game, 'Create' melibatkan pembuatan alat atau antarmuka yang memungkinkan pemain untuk melakukan ini dengan mudah. Pemrograman untuk 'Create' mungkin melibatkan pengembangan editor yang memungkinkan pemain untuk merancang objek atau karakter dengan menggunakan komponen yang sudah ada, seperti pemilihan tekstur, bentuk, dan atribut lainnya. Dari sisi pemrograman, ini mungkin melibatkan implementasi antarmuka yang intuitif, pengembangan algoritma untuk menghasilkan objek yang sesuai dengan pilihan pemain, serta penyimpanan data objek yang telah dibuat oleh pemain. Selain itu, sistem 'undo' atau 'redo' mungkin diperlukan untuk memungkinkan pemain mengubah keputusan mereka selama proses penciptaan.

## 5. Find

'Find' adalah game goal typology yang melibatkan pencarian dan penemuan objek atau informasi dalam lingkungan permainan. Dalam pengembangan game, ini melibatkan penciptaan lokasi atau kondisi di mana pemain harus mencari sesuatu yang penting untuk melanjutkan permainan. Pemrograman untuk 'Find' mungkin melibatkan pengembangan algoritma pencarian yang mengatur lokasi atau penempatan objek yang harus ditemukan oleh pemain. Dari sudut pandang pemrograman, ini bisa melibatkan pemberian indikasi atau petunjuk yang tepat kepada pemain untuk membantu mereka menemukan objek yang dicari. Sistem pelacakan objek, seperti penanda atau tanda petunjuk, mungkin juga diperlukan untuk membantu pemain menavigasi dan menemukan objek dengan lebih mudah. Selain itu, pemrograman juga dapat melibatkan pengaturan tingkat kejadian objek yang muncul, sehingga pemain merasakan tantangan dalam mencari objek tersebut.

| **Cara Membuat Game Typology Find** |
| --- |
| **Tutorial:** <https://youtu.be/DQGQGu-ab84> |

## 6. Obtain

'Obtain' adalah game goal typology yang fokus pada tujuan permainan untuk mendapatkan atau mengumpulkan benda-benda penting dalam game. Dalam pengembangan game, ini melibatkan penempatan objek-objek yang dapat diambil oleh pemain di berbagai lokasi. Pemrograman untuk 'Obtain' mungkin melibatkan pengembangan sistem yang mengelola distribusi objek, baik itu item, mata uang, atau sumber daya lainnya. Dari sudut pandang pemrograman, ini dapat melibatkan pembuatan mekanika untuk menentukan probabilitas kemunculan objek di berbagai tempat, serta pengembangan sistem inventaris yang memungkinkan pemain menyimpan dan mengelola objek yang telah diperoleh. Selain itu, pemrograman juga harus mempertimbangkan bagaimana objek tersebut dapat digunakan atau diperdagangkan oleh pemain, serta bagaimana mereka dapat berinteraksi dengan objek-objek tersebut dalam game.

| **Cara Membuat Game Typology Obtain** |
| --- |
| **Tutorial:** <https://youtu.be/97z39adCNkg> |

## 7. Optimize

'Optimize' dalam game goal typology merujuk pada tujuan permainan yang mendorong pemain untuk meningkatkan atau mengoptimalkan kinerja, kemampuan, atau atribut dalam game. Dalam pengembangan game, 'Optimize' melibatkan pengembangan sistem yang memungkinkan pemain untuk memodifikasi atau meningkatkan aspek-aspek tertentu dalam game. Dari perspektif pemrograman, ini mungkin melibatkan pembuatan algoritma yang memungkinkan pemain untuk mengatur atribut atau karakteristik dalam game, seperti meningkatkan statistik karakter atau memodifikasi kemampuan. Selain itu, pemrograman juga harus mempertimbangkan dampak dari optimisasi ini pada permainan secara keseluruhan, untuk memastikan keseimbangan dan pengalaman yang adil bagi semua pemain. Misalnya, jika pemain dapat mengoptimalkan karakter mereka dengan cara yang tidak seimbang, ini dapat mengganggu kesenjangan antara pemain yang telah mengoptimalkan dan yang belum. Dalam hal ini, pemrograman harus mengatur batasan atau kriteria yang adil untuk mengoptimalkan aspek tertentu dalam game.

## 8. Reach

'Reach' dalam game goal typology mengacu pada tujuan permainan yang mengharuskan pemain mencapai lokasi atau titik tertentu dalam game. Dalam pengembangan game, 'Reach' melibatkan pembuatan kondisi atau tantangan yang harus dipenuhi oleh pemain agar mereka dapat mencapai tujuan tersebut. Dari perspektif pemrograman, ini bisa melibatkan pembuatan peta atau level yang dirancang sedemikian rupa sehingga pemain harus melewati rintangan atau tantangan tertentu untuk mencapai titik akhir. Pemrograman mungkin melibatkan implementasi mekanika pergerakan yang memungkinkan pemain berinteraksi dengan lingkungan dan melewati rintangan, seperti melompat, merayap, atau menghindar. Selain itu, pembuatan algoritma untuk menghitung jarak dan jalan tercepat dari titik awal ke titik akhir juga diperlukan. Dalam beberapa game, pemrograman juga dapat melibatkan pengembangan mekanika 'checkpoint' atau 'save point' yang memungkinkan pemain menyimpan kemajuan mereka dan mengulang tantangan jika diperlukan.

## 9. Synchronize

'Synchronize' dalam game goal typology merujuk pada tujuan permainan yang melibatkan koordinasi atau sinkronisasi tindakan atau pergerakan antara pemain atau elemen dalam game. Dalam pengembangan game, 'Synchronize' melibatkan pembuatan situasi di mana pemain harus bekerja sama atau mengatur tindakan mereka dengan cermat untuk mencapai tujuan tertentu. Pemrograman untuk 'Synchronize' mungkin melibatkan pengembangan sistem yang memungkinkan pemain berkomunikasi atau berinteraksi dalam waktu nyata. Ini bisa berupa fitur chat, mode kooperatif, atau sistem komunikasi visual seperti isyarat. Dari sisi pemrograman, ini juga mungkin melibatkan pembuatan algoritma yang mengoordinasikan tindakan pemain atau entitas lain dalam game. Misalnya, dalam permainan tim, pemrograman dapat mengatur bagaimana tindakan dari satu pemain dapat mempengaruhi tindakan yang diperlukan dari pemain lain untuk mencapai tujuan bersama. Selain itu, pemrograman juga harus mempertimbangkan waktu dan reaksi dalam mengoordinasikan tindakan, serta mengembangkan sistem umpan balik yang memungkinkan pemain untuk tahu apakah mereka telah berhasil melakukan sinkronisasi dengan benar.

## 10. Remove

'Remove' dalam game goal typology mengacu pada tujuan permainan yang melibatkan penghapusan atau eliminasi elemen tertentu dalam game. Dalam pengembangan game, 'Remove' melibatkan penciptaan tantangan atau situasi di mana pemain harus menghilangkan objek atau karakter tertentu untuk mencapai tujuan. Pemrograman untuk 'Remove' mungkin melibatkan pembuatan mekanika yang memungkinkan pemain berinteraksi dengan objek atau karakter dan menghapusnya dari lingkungan permainan. Dari sisi pemrograman, ini bisa melibatkan implementasi deteksi tabrakan atau interaksi yang memungkinkan pemain untuk memilih target dan menghapusnya dari permainan. Pemrograman juga harus mempertimbangkan konsekuensi dari penghapusan ini pada permainan secara keseluruhan, untuk memastikan bahwa tindakan ini sesuai dengan tujuan permainan dan tidak mengganggu keseimbangan atau pengalaman pemain. Misalnya, dalam permainan puzzle, pemrograman dapat melibatkan pengembangan algoritma yang menentukan bagaimana penghapusan objek akan mempengaruhi struktur level dan apakah tujuan pemain dapat tercapai setelah objek tersebut dihapus.

# 

# 

# Chapter 8 Augmented Reality

# 

Pengembangan Augmented Reality (AR) berbasis Unity telah memungkinkan penjelajahan yang mendalam dalam dunia teknologi yang semakin maju. AR adalah teknologi yang menggabungkan elemen dunia nyata dengan elemen virtual, menciptakan pengalaman yang unik dan memperkaya realitas sekitar. Dalam konteks Unity, berbagai jenis teknik AR telah dikembangkan untuk menciptakan pengalaman yang semakin menarik dan imersif bagi pengguna.

Salah satu teknik AR yang populer adalah marker-based AR. Teknik ini melibatkan penggunaan marker atau citra khusus yang dikenali oleh kamera perangkat untuk menentukan posisi dan orientasi objek virtual dalam ruang nyata. Misalnya, dengan mengarahkan kamera perangkat ke gambar marker pada majalah atau papan iklan, pengguna dapat melihat objek 3D atau informasi tambahan yang terhubung dengan marker tersebut. Seiring dengan perkembangan teknologi, markerless AR juga semakin penting. Teknik ini tidak memerlukan marker fisik, melainkan memanfaatkan fitur-fitur lingkungan seperti permukaan rata atau pola unik untuk menentukan posisi objek virtual. Markerless AR sering memanfaatkan teknologi pemrosesan citra dan deteksi objek yang canggih.

Selain itu, interaksi tombol virtual adalah elemen penting dalam pengalaman AR. Ini memungkinkan pengguna untuk berinteraksi dengan objek virtual dengan menekan tombol-tombol yang muncul di layar perangkat mereka. Ini menciptakan tingkat interaktivitas yang tinggi dan memungkinkan pengguna untuk mengendalikan objek AR. Selain teknik-teknik dasar, Unity juga mendukung pengembangan AR berbasis lokasi dengan menggunakan sensor perangkat seperti GPS dan accelerometer. Ini memungkinkan pengembang untuk menciptakan pengalaman AR yang terkait dengan lokasi geografis pengguna, seperti tur panduan kota atau permainan berbasis lokasi.

Keuntungan besar dalam mengembangkan AR dengan Unity adalah ketersediaan beragam asset dan plugin yang dapat mempercepat pengembangan. Unity memiliki toko asset yang besar, yang berisi berbagai objek 3D, efek visual, dan komponen AR yang dapat digunakan oleh pengembang untuk membuat pengalaman AR yang kaya.Selain itu, Unity juga mendukung platform multiplatform, artinya pengembang dapat menciptakan pengalaman AR yang kompatibel dengan berbagai perangkat, termasuk smartphone, tablet, dan perangkat AR khusus seperti HoloLens atau Magic Leap.

Dalam konteks AR, visualisasi 3D adalah aspek penting lainnya. Unity memungkinkan pengembang untuk menciptakan objek 3D yang realistis dan animasi yang memukau, sehingga pengguna merasa benar-benar terlibat dalam pengalaman AR. Selain itu, Unity juga mendukung pengembangan berbasis cloud. Ini berarti data AR dapat disimpan di cloud, memungkinkan pengguna untuk mengakses pengalaman AR mereka dari berbagai perangkat dan lokasi yang berbeda.

Terakhir, integrasi dengan ARCore dan ARKit membuat Unity menjadi platform yang sangat kuat dalam pengembangan AR di perangkat Android dan iOS. Ini membuka peluang besar untuk menciptakan pengalaman AR yang seru di perangkat seluler. Secara keseluruhan, Unity telah memainkan peran besar dalam pengembangan teknologi AR yang inovatif dan menarik. Dengan berbagai teknik, alat, dan kemampuan yang tersedia, para pengembang memiliki potensi tak terbatas untuk menciptakan pengalaman AR yang memukau bagi pengguna mereka.

## 1. Native AR

NativeAR berbasis WebRequests adalah konsep pengembangan dalam Unity yang menggabungkan teknologi Augmented Reality (AR) dengan penggunaan WebRequests (permintaan ke server web) untuk mengambil data dinamis dari internet. Dalam konteks ini, "NativeAR" merujuk pada pengembangan pengalaman AR yang dapat diakses secara langsung dari perangkat bergerak Anda, seperti ponsel atau tablet, tanpa perlu mengunduh aplikasi tambahan.

## 

## 2. Vuforia

Vuforia adalah platform pengenalan gambar dan Augmented Reality (AR) yang dikembangkan oleh PTC (Parametric Technology Corporation). Platform ini memungkinkan pengembang perangkat lunak untuk membuat pengalaman AR yang interaktif dengan menggunakan pengenalan gambar dan tracking berbasis kamera. Vuforia dapat digunakan untuk menggabungkan objek virtual dengan dunia nyata, sehingga pengguna dapat melihat dan berinteraksi dengan konten digital yang tumpang tindih pada objek fisik melalui perangkat mobile atau AR glasses.

Kombinasi Vuforia dan Unity sangat populer dalam pengembangan Augmented Reality. Unity menyediakan lingkungan pengembangan yang kuat dan intuitif, sementara Vuforia memungkinkan integrasi mudah pengenalan gambar dan tracking berbasis kamera. Dengan menggabungkan keduanya, pengembang dapat menciptakan aplikasi AR yang menarik dengan lebih mudah, seperti aplikasi pemasaran, edukasi, permainan AR, dan banyak lagi. Ini adalah alat yang sangat berguna bagi mereka yang ingin memanfaatkan teknologi AR dalam proyek pengembangan perangkat lunak mereka.

| **Vuforia** |
| --- |
| <https://developer.vuforia.com/> |

| **Cara Install Vuforia Di Unity** |
| --- |
| **Tutorial:** <https://youtu.be/TD_EirnlaqY> |

### 2.1 Marker Based

Marker-based augmented reality (AR) dengan Vuforia adalah teknologi yang memungkinkan konten digital, seperti model 3D, animasi, atau informasi, ditampilkan di atas objek fisik atau gambar, yang disebut marker, dalam dunia nyata. Vuforia, sebuah platform pengembangan AR yang populer, menggunakan algoritma visi komputer untuk mengenali dan melacak marker-marker ini secara real-time melalui kamera perangkat. Setelah marker terdeteksi, Vuforia dengan akurat menentukan posisi dan orientasinya dalam ruang 3D, memungkinkan pengembang untuk secara mulus mengintegrasikan elemen-elemen virtual ke dalam pandangan pengguna, menciptakan pengalaman AR yang interaktif dan mendalam. Pendekatan berbasis marker ini banyak digunakan dalam berbagai aplikasi, termasuk pendidikan, permainan, pemasaran, dan pelatihan industri, memberikan pengguna interaksi yang lebih menarik dengan lingkungan fisik mereka.

| **Cara Membuat Target Marker Di Vuforia** |
| --- |
| **Tutorial:** <https://youtu.be/wIFVE6qIHnc> |

| **Cara Build Aplikasi Vuforia Ke Android** |
| --- |
| **Tutorial:** <https://youtu.be/-GEMABknH-4> |

### 2.2. Object Based

Object-Based Augmented Reality (AR) berbasis Vuforia di Unity adalah metode pengembangan AR yang memungkinkan objek fisik dunia nyata menjadi titik awal untuk menambahkan objek virtual dan interaksi digital. Metode ini berfokus pada penggunaan objek fisik dalam dunia nyata sebagai elemen kunci dalam pengalaman AR. Daripada mengandalkan penanda khusus, seperti gambar atau logo, Object-Based AR memanfaatkan objek fisik yang ada sebagai "penanda" alami. Objek Fisik: Objek fisik dalam dunia nyata yang akan digunakan sebagai penanda tidak perlu memiliki pola visual khusus. Ini bisa berupa benda sehari-hari seperti mainan, alat, atau bahkan produk yang dikenali secara alami oleh perangkat.

### 2.3. Marker Less

Marker-Less Augmented Reality (AR) berbasis AR Foundation di Unity adalah pendekatan dalam pengembangan AR di mana tidak diperlukan penanda fisik atau objek khusus untuk memicu objek virtual dan pengalaman AR. Sebaliknya, teknik ini memanfaatkan pemahaman komputer dan pemrosesan citra untuk mengenali dan melacak permukaan objek dunia nyata secara real-time.

AR Foundation adalah kerangka kerja (framework) dalam Unity yang memungkinkan pengembang untuk membuat pengalaman AR yang kompatibel dengan berbagai perangkat, termasuk iOS dan Android. Ini memberikan alat dan fungsionalitas penting untuk mengembangkan aplikasi AR yang responsif dan dinamis.

Visual SLAM (Simultaneous Localization and Mapping): Salah satu teknologi kunci dalam Marker-Less AR adalah Visual SLAM. Ini adalah metode pemetaan dan pelacakan real-time yang memungkinkan perangkat untuk memahami posisinya dalam ruang dan memetakan objek dunia nyata saat kamera bergerak. Dengan Visual SLAM, objek virtual dapat ditempatkan dengan akurat di lingkungan fisik.

### 2.4. Gesture Interaction

Vuforia adalah sebuah platform augmented reality yang menawarkan berbagai jenis variasi interaksi yang dapat digunakan untuk memperkaya pengalaman pengguna. Salah satu jenis interaksi yang umum adalah deteksi marker, di mana pengguna dapat mengarahkan perangkat mereka ke suatu objek atau gambar yang telah ditandai sebelumnya untuk melihat konten tambahan seperti model 3D atau informasi tambahan. Selain itu, terdapat juga variasi interaksi berbasis deteksi wajah dan gestur, di mana pengguna dapat berinteraksi dengan aplikasi Vuforia melalui gerakan tangan atau ekspresi wajah mereka. Contohnya, pengguna dapat menggerakkan tangan mereka untuk memanipulasi objek virtual atau mengenali ekspresi wajah untuk mengaktifkan respon tertentu dalam aplikasi. Semua variasi ini memungkinkan pengembang untuk menciptakan pengalaman augmented reality yang lebih menarik dan interaktif.

| **Cara Membuat Interaksi Rotasi & Zoom di Vuforia** |
| --- |
| **Tutorial:** <https://youtu.be/V01mWwdbOL8> |

| **Cara Membuat Variasi Interaksi di Vuforia** |
| --- |
| **Tutorial:** <https://youtu.be/QNGBafStmHA> |

| **Cara Membuat Interaksi Joystick di Vuforia** |
| --- |
| **Tutorial:** <https://youtu.be/pdLRo2MBzDo> |

| **Cara Membuat Interaksi AR Interior Rumah (Part 1)** |
| --- |
| **Tutorial:** <https://youtu.be/vXey3HaeHhk> |

### 2.5. Virtual Button

Virtual Button di Vuforia adalah salah satu fitur yang memungkinkan interaksi yang menarik dengan marker kertas dalam konteks augmented reality. Dengan Virtual Button, pengguna dapat menciptakan tombol virtual yang terkait dengan marker fisik pada kertas. Ketika pengguna menekan marker kertas yang telah ditandai, tombol virtual ini dapat diaktifkan, memungkinkan aksi atau respons tertentu, seperti menampilkan objek 3D tambahan, memulai animasi, atau mengakses informasi tambahan. Hal ini menciptakan pengalaman yang lebih interaktif dan dinamis bagi pengguna ketika mereka berinteraksi dengan objek fisik yang telah ditandai dengan marker kertas. Virtual Button adalah salah satu alat yang kuat dalam pengembangan aplikasi augmented reality yang melibatkan penekanan marker kertas untuk menciptakan respons yang beragam dan menarik.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using Vuforia;  public class VuforiaVirtualButton : MonoBehaviour  {  [Header("Main Settings")]  public VirtualButtonBehaviour VirtualButton;  [Header("Event Settings")]  public UnityEvent OnPressedEvent;  public UnityEvent OnReleasedEvent;  // Start is called before the first frame update  void Start()  {  VirtualButton.RegisterOnButtonPressed(OnButtonPressed);  VirtualButton.RegisterOnButtonReleased(OnButtonReleased);  }  // Update is called once per frame  void Update()  {    }  public void OnButtonPressed(VirtualButtonBehaviour vb)  {  OnPressedEvent.Invoke();  }  public void OnButtonReleased(VirtualButtonBehaviour vb)  {  OnReleasedEvent.Invoke();  }  } |
| --- |

| **Cara Membuat Interaksi Virtual Button Di Vuforia** |
| --- |
| **Tutorial:** <https://youtu.be/8RzwVqnuGIw> |

### 2.6. Camera Focus

Camera focus dalam konteks Vuforia adalah salah satu fitur yang memungkinkan kamera ponsel atau perangkat lainnya untuk lebih fokus pada objek yang ditandai atau dikenali dalam pengalaman augmented reality. Dengan kemampuan ini, kamera dapat secara otomatis menyesuaikan fokusnya agar objek yang ditampilkan dalam aplikasi Vuforia menjadi lebih tajam dan jelas. Ini merupakan aspek penting dalam menciptakan pengalaman augmented reality yang realistis dan memuaskan, karena objek virtual yang lebih tajam dan fokus dapat lebih baik terintegrasi dengan lingkungan fisik sekitarnya. Fitur kamera focus dalam Vuforia membantu pengembang dan pengguna dalam mencapai hasil yang optimal dalam interaksi dengan dunia virtual yang dihadirkan melalui perangkat ponsel mereka.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using Vuforia;  public class VuforiaCameraFocus : MonoBehaviour  {  // Start is called before the first frame update  void Start()  {  VuforiaApplication.Instance.OnVuforiaStarted += OnVuforiaStarted;  VuforiaApplication.Instance.OnVuforiaPaused += OnPaused;  }  private void OnVuforiaStarted()  {  VuforiaBehaviour.Instance.CameraDevice.SetFocusMode(FocusMode.FOCUS\_MODE\_CONTINUOUSAUTO);  VuforiaBehaviour.Instance.CameraDevice.SetCameraMode(Vuforia.CameraMode.MODE\_DEFAULT);  }  private void OnPaused(bool paused)  {  if (!paused) // Resumed  {  // Set again autofocus mode when app is resumed  VuforiaBehaviour.Instance.CameraDevice.SetFocusMode(FocusMode.FOCUS\_MODE\_CONTINUOUSAUTO);  }  }  } |
| --- |

### 2.7. Multi Marker

Multi Marker dalam konteks Vuforia adalah salah satu fitur yang memungkinkan beberapa objek dikenali sekaligus untuk menghasilkan suatu event atau animasi tertentu. Dengan kemampuan ini, kemungkinan dari visualisasi AR menjadi lebih bervariasi karena ada kombinasi dari berbagai bentuk marker yang dapat di deteksi. Contoh umum dari bentuk multi marker adalah battle card yang memungkinkan animasi pertarungan muncul jika terdapat 2 kartu tertentu yang terdeteksi oleh kamera. Selain itu terdapat juga variasi animasi binatang makan, yang akan menghasilkan suatu animasi jika terdapat 2 kartu yang menunjukkan animasi tertentu ketika marker binatang dan makanannya terdeteksi secara bersamaan.

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using Vuforia;  using UnityEngine.Events;  public class VuforiaMultiMarker : MonoBehaviour  {  [System.Serializable]  public class CTargetMarker  {  public string MarkerName;  public bool MarkerStatus;  }  [Header("Target Markers")]  public List<CTargetMarker> TargetMarker;  [Header("Tracked Event")]  public UnityEvent OnTrackedFound;  public UnityEvent OnTrackedLost;  public void SetTrackedTrue(int aIndex)  {  TargetMarker[aIndex].MarkerStatus = true;  }  public void SetTrackedFalse(int aIndex)  {  TargetMarker[aIndex].MarkerStatus = false;  }  // Start is called before the first frame update  void Start()  {    }  bool AllTrackedFound()  {  bool result = true;  for (int i = 0; i < TargetMarker.Count; i++)  {  if (TargetMarker[i].MarkerStatus == false)  {  result = false;  }  }  return result;  }  bool AllTrackedLost()  {  bool result = false;  for (int i = 0; i < TargetMarker.Count; i++)  {  if (TargetMarker[i].MarkerStatus == true)  {  result = true;  }  }  return result;  }  // Update is called once per frame  void Update()  {  if (AllTrackedFound())  {  OnTrackedFound.Invoke();  }  else if (!AllTrackedLost())  {  OnTrackedLost.Invoke();  }  }  } |
| --- |

# 

# 

# Chapter 9 Virtual Reality

# 

Konsep Virtual Reality (VR) berbasis Unity adalah sebuah gebrakan revolusioner dalam dunia teknologi yang memungkinkan pengguna untuk sepenuhnya terbenam dalam pengalaman komputer yang dibuat dengan realisme tinggi. Unity, sebagai salah satu platform pengembangan permainan dan aplikasi terkemuka, telah menjadi landasan penting bagi pengembang VR untuk menciptakan dunia-dunia digital yang memukau. Saat menjelajahi dunia VR, ada beberapa jenis teknik yang digunakan untuk menciptakan pengalaman yang mengesankan dan imersif bagi pengguna.

Salah satu teknik VR yang paling umum adalah Head-Mounted Display (HMD), di mana pengguna mengenakan perangkat berupa headset khusus yang dilengkapi dengan layar yang meliputi seluruh pandangan pengguna. Ini memungkinkan pengguna untuk benar-benar merasa berada dalam dunia virtual, karena pandangan mereka sepenuhnya terbentuk oleh lingkungan VR.

Selain itu, teknik Room-Scale VR mengizinkan pengguna untuk bergerak bebas dalam ruang fisik yang cukup besar, dengan perangkat pelacak yang memungkinkan penyesuaian posisi objek virtual sesuai dengan pergerakan pengguna. Hal ini menciptakan pengalaman yang lebih imersif karena pengguna dapat berinteraksi dengan lingkungan VR seolah-olah mereka berada di dalamnya. 360-Degree Video VR adalah teknik lain yang menggabungkan video 360 derajat dengan elemen VR, memungkinkan pengguna untuk menjelajahi dan berinteraksi dengan video 360 derajat yang direkam dalam lingkungan nyata. Hal ini umumnya digunakan dalam aplikasi pemandu wisata VR atau konten VR berbasis video.

Dalam dunia VR yang semakin maju, teknik Hand Tracking juga semakin penting. Ini memungkinkan pengguna untuk berinteraksi dengan objek virtual menggunakan gerakan tangan mereka tanpa perlu menggunakan controller fisik, menciptakan pengalaman yang lebih alami dan mendalam. Selain teknik-teknik tersebut, ada juga konsep Mixed Reality (MR), yang menggabungkan elemen dunia nyata dengan objek virtual dalam waktu nyata, menciptakan pengalaman yang lebih kompleks dan menarik. Teknik VR ini sering digunakan dalam aplikasi pelatihan, simulasi industri, dan hiburan interaktif. Dengan berbagai teknik VR yang tersedia, Unity memberikan alat yang kuat bagi para pengembang untuk menciptakan pengalaman VR yang memukau dan bervariasi, membuka pintu menuju dunia digital yang semakin nyata dan menarik.

## 1. Google VR

Google VR adalah inisiatif dari Google yang bertujuan untuk mendukung dan memajukan pengembangan aplikasi Realitas Virtual (VR) di berbagai platform, termasuk Unity. Dalam konteks pengembangan aplikasi di Unity, Google VR menyediakan berbagai alat, sumber daya, dan platform yang memungkinkan pengembang untuk menciptakan pengalaman VR yang menarik dan beragam. Google VR menyediakan SDK (Software Development Kit) yang memudahkan pengembang untuk mengintegrasikan perangkat Google VR dengan aplikasi Unity mereka. SDK ini mencakup alat dan sumber daya untuk mengelola pelacakan gerakan, tampilan ganda, audio VR, dan fitur-fitur VR khusus lainnya.

## 2. Occulus VR

Oculus VR adalah merek dan platform yang signifikan dalam pengembangan aplikasi VR (Realitas Virtual) di Unity. Dalam konteks pengembangan aplikasi VR di Unity, Oculus VR adalah salah satu dari beberapa perangkat keras dan platform yang didukung oleh Unity untuk menciptakan pengalaman VR yang mengesankan. Oculus VR menyediakan "Oculus Integration for Unity," yang merupakan paket Unity yang memudahkan pengembang untuk mengintegrasikan perangkat Oculus dengan aplikasi mereka. Ini termasuk dukungan untuk kontroler Oculus, pelacakan gerakan, dan fitur-fitur VR khusus lainnya.

# 

# 

# 

# Chapter 10 Unity Third Party Plugins

Third-party plugins untuk Unity adalah alat yang tak ternilai dalam ekosistem pengembangan game. Dikembangkan oleh berbagai vendor independen, plugin-plugin ini menyediakan fitur tambahan yang memperkaya potensi Unity sebagai platform pengembangan game. Contoh plugin pihak ketiga meliputi pengaturan tampilan, alat visual scripting, integrasi sosial media, dan bahkan dukungan untuk perangkat realitas virtual dan augmented. Kemampuan untuk mengakses berbagai plugin ini memungkinkan pengembang untuk menghemat waktu dan upaya dalam mengembangkan beragam aspek game mereka.

Ketika datang ke pengembangan game dengan Unity, plugin pihak ketiga telah menjadi komponen penting dalam perangkat pengembang. Mereka memungkinkan pengguna Unity untuk mengakses alat dan teknologi yang mungkin tidak mereka miliki sendiri, seperti mesin fisika yang canggih, perangkat keras spesifik, atau bahkan algoritma kecerdasan buatan yang kuat. Dengan integrasi plugin pihak ketiga, pengembang dapat mempercepat pengembangan game mereka dan menjaga proyek mereka tetap up-to-date dengan teknologi terbaru, yang sering kali sulit atau tidak praktis untuk dibangun dari awal.

Salah satu keunggulan utama penggunaan plugin pihak ketiga dalam Unity adalah fleksibilitasnya. Seiring dengan perkembangan teknologi dan perubahan tren di industri game, pengembang dapat dengan mudah menyesuaikan proyek mereka dengan menggunakan berbagai plugin yang tersedia. Selain itu, komunitas pengembang yang luas sering berkontribusi dengan plugin-plugin open source, memperluas ekosistem Unity dan memberikan solusi kreatif yang dapat diadopsi oleh pengembang lain. Dengan kata lain, plugin pihak ketiga menjadi fondasi yang kuat untuk menciptakan game yang inovatif, efisien, dan berkualitas tinggi di Unity.

## 1. Dynamic Parkour System

Dynamic Parkour System buatan Eric Canela untuk Unity adalah sebuah terobosan yang revolusioner dalam pengembangan game ala Assassin's Creed. Sistem ini memungkinkan para pengembang game untuk dengan mudah mengintegrasikan elemen parkour yang dinamis dan realistis ke dalam permainan mereka. Dengan menggunakan Dynamic Parkour System, pemain dapat merasakan sensasi melompat, berlari, dan memanjat gedung-gedung dengan cara yang sangat mirip dengan karakter dalam Assassin's Creed. Sistem ini menyediakan berbagai fitur canggih seperti deteksi permukaan, animasi yang halus, dan kontrol yang responsif, sehingga pengembang dapat fokus pada aspek cerita dan desain level tanpa harus menghabiskan waktu yang banyak untuk mengimplementasikan mekanik parkour yang rumit. Dengan adanya Dynamic Parkour System ini, pengalaman bermain game ala Assassin's Creed dapat lebih mudah diwujudkan dalam berbagai proyek game yang berbasis Unity.

| **Dynamic Parkour System** |
| --- |
| <https://github.com/knela96/Dynamic-Parkour-System> |

## 2. Microsoft Playfab

Microsoft PlayFab adalah sebuah platform layanan cloud yang dirancang khusus untuk membantu pengembang permainan dalam mengelola, mengembangkan, dan meningkatkan permainan mereka. Ini merupakan bagian dari layanan cloud Microsoft Azure yang menyediakan berbagai alat dan fitur yang dapat digunakan oleh pengembang permainan untuk memudahkan pengembangan, distribusi, dan pengelolaan permainan mereka. Berikut ini beberapa poin penting tentang Microsoft PlayFab:

* Manajemen Pengguna dan Data: PlayFab menyediakan alat untuk manajemen pengguna, autentikasi, serta penyimpanan data. Ini memungkinkan pengembang untuk membuat sistem akun, menyimpan data pemain, dan melacak kemajuan permainan.
* Monetisasi: Platform ini juga mendukung strategi monetisasi permainan dengan fitur seperti pembelian dalam aplikasi, promosi, dan sistem mata uang dalam permainan.
* Analytics: PlayFab dilengkapi dengan alat analitik yang kuat, yang memungkinkan pengembang untuk melacak performa permainan mereka, memahami perilaku pemain, dan mengambil keputusan berdasarkan data.
* Pengembangan: Pengembang permainan dapat menggunakan berbagai SDK (Software Development Kit) yang disediakan oleh PlayFab untuk mengintegrasikan layanan ini dengan permainan mereka, yang mencakup dukungan untuk berbagai platform seperti PC, konsol, dan perangkat seluler.
* Multi-Platform: PlayFab mendukung pengembangan permainan lintas platform. Artinya, pengembang dapat membuat permainan mereka dapat diakses dan dimainkan di berbagai perangkat.
* Dukungan AI: PlayFab juga memiliki kemampuan integrasi dengan layanan kecerdasan buatan (AI) Microsoft seperti Azure Cognitive Services, yang dapat digunakan untuk meningkatkan pengalaman pemain dengan fitur-fitur seperti pengenalan suara atau analisis sentimen pemain.
* Komersialisasi: Microsoft PlayFab juga memiliki program komersialisasi yang memungkinkan pengembang permainan untuk menghasilkan pendapatan dengan cara menjual layanan tambahan atau berkolaborasi dengan mitra Microsoft.

Microsoft PlayFab memberikan solusi lengkap untuk pengembang permainan, mulai dari pengelolaan data pemain hingga pemantauan performa, monetisasi, dan manajemen platform yang berbeda. Ini membantu pengembang untuk fokus pada pengembangan inti permainan mereka, sementara PlayFab mengurus aspek-aspek teknis yang kompleks.

| **Microsoft Playfab** |
| --- |
| <https://playfab.com/> |
| <https://github.com/PlayFab/UnitySDK> |
| <https://learn.microsoft.com/en-us/gaming/playfab/sdks/unity3d/installing-unity3d-sdk#install-the-playfab-unity-editor-extensions-and-the-playfab-sdk> |

| **Cara Membuat Register User Baru Di PlayFab** |
| --- |
| **Tutorial:** <https://youtu.be/UcddtvJfv0A> |

| **Cara Membuat User Login Di PlayFab** |
| --- |
| **Tutorial:** <https://youtu.be/DEqFivyyGmI> |

| **Cara Membuat Leaderboard Di PlayFab** |
| --- |
| **Tutorial:** <https://youtu.be/jzgHd6ZExEU> |

| **Cara Mengeloa Data Di PlayFab** |
| --- |
| **Tutorial:** <https://youtu.be/FeybvKKBaSI> |

### 2.1 Playfab Anonymous

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabAnonymous : MonoBehaviour  {  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Success Settings")]  public UnityEvent OnLoginSuccessEvents;  [Header("Error Settings")]  public UnityEvent OnLoginErrorEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokeAnonymousLogin()  {  var request = new LoginWithCustomIDRequest  {  CustomId = SystemInfo.deviceUniqueIdentifier,  CreateAccount = true  };  PlayFabClientAPI.LoginWithCustomID(request, OnLoginSuccess, OnLoginError);  }  void OnLoginSuccess(LoginResult result)  {  Debug.Log("PlayFab Login: Success");  OnLoginSuccessEvents?.Invoke();  }  void OnLoginError(PlayFabError result)  {  Debug.Log("PlayFab Login: Error (" + result.GenerateErrorReport() + ")");  OnLoginErrorEvents?.Invoke();  }  } |
| --- |

### 2.2 Playfab Register

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabRegister : MonoBehaviour  {  [Header("PlayFab Settings")]  public string PlayFabID;  [Header("Register Settings")]  public InputField emailInput;  public InputField usernameInput;  public InputField displaynameInput;  public InputField passwordInput;  public InputField confirmpassInput;  public bool UpdateDisplayName;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Success Settings")]  public UnityEvent OnRegisterSuccessEvents;  public UnityEvent OnUpdateDisplayNameSuccessEvents;  [Header("Error Settings")]  public UnityEvent OnRegisterErrorEvents;  public UnityEvent OnUpdateDisplayNameErrorEvents;  public UnityEvent OnUsernameLengthEvents;  public UnityEvent OnPasswordLengthEvents;  public UnityEvent OnPasswordConfirmationEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokeRegister()  {  if (usernameInput.text.Length < 3)  {  Debug.Log("PlayFab Register: User Length Error");  OnUsernameLengthEvents?.Invoke();  }  if (passwordInput.text.Length < 6)  {  Debug.Log("PlayFab Register: Password Length Error");  OnPasswordLengthEvents?.Invoke();  }  if (passwordInput.text != confirmpassInput.text)  {  Debug.Log("PlayFab Register: Password Confirmation Error");  OnPasswordConfirmationEvents?.Invoke();  }  else  {  var request = new RegisterPlayFabUserRequest  {  Email = emailInput.text,  Username = usernameInput.text,  Password = passwordInput.text,  RequireBothUsernameAndEmail = true  };  PlayFabClientAPI.RegisterPlayFabUser(request, OnRegisterSuccess, OnRegisterError);  }  }  public void InvokeUpdateDisplayName()  {  string newDisplayName = displaynameInput.text;  var request = new UpdateUserTitleDisplayNameRequest  {  DisplayName = newDisplayName  };  PlayFabClientAPI.UpdateUserTitleDisplayName(request, OnDisplayNameUpdateSuccess, OnDisplayNameUpdateError);  }  void OnRegisterSuccess(RegisterPlayFabUserResult result)  {  Debug.Log("PlayFab Register: Success");  OnRegisterSuccessEvents?.Invoke();  if (UpdateDisplayName)  {  InvokeUpdateDisplayName();  }  }  void OnRegisterError(PlayFabError result)  {  Debug.Log("PlayFab Register: Error (" + result.GenerateErrorReport() + ")");  OnRegisterErrorEvents?.Invoke();  }  private void OnDisplayNameUpdateSuccess(UpdateUserTitleDisplayNameResult result)  {  Debug.Log("DisplayName updated successfully: " + result.DisplayName);  OnUpdateDisplayNameSuccessEvents?.Invoke();  // Refresh leaderboard or update UI as needed.  }  private void OnDisplayNameUpdateError(PlayFabError error)  {  Debug.LogError("Error updating DisplayName: " + error.ErrorMessage);  OnUpdateDisplayNameErrorEvents?.Invoke();  // Handle error, if needed.  }  } |
| --- |

### 2.3 Playfab Login

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabLogin : MonoBehaviour  {  [Header("PlayFab Settings")]  public string PlayFabID;  [Header("Register Settings")]  public InputField emailInput;  public InputField passwordInput;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Success Settings")]  public UnityEvent OnLoginSuccessEvents;  [Header("Error Settings")]  public UnityEvent OnLoginErrorEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokeLogin()  {  var request = new LoginWithEmailAddressRequest  {  Email = emailInput.text,  Password = passwordInput.text  };  PlayFabClientAPI.LoginWithEmailAddress(request, OnLoginSuccess, OnLoginError);  }  public void InvokePasswordReset()  {  var request = new SendAccountRecoveryEmailRequest  {  Email = emailInput.text,  TitleId = PlayFabID  };  PlayFabClientAPI.SendAccountRecoveryEmail(request, OnResetSuccess, OnResetError);  }  void OnLoginSuccess(LoginResult result)  {  Debug.Log("PlayFab Login: Success");  OnLoginSuccessEvents?.Invoke();  }  void OnLoginError(PlayFabError result)  {  Debug.Log("PlayFab Login: Error (" + result.GenerateErrorReport() + ")");  OnLoginErrorEvents?.Invoke();  }  void OnResetSuccess(SendAccountRecoveryEmailResult result)  {  Debug.Log("PlayFab Reset Password: Success");  OnLoginSuccessEvents?.Invoke();  }  void OnResetError(PlayFabError result)  {  Debug.Log("PlayFab Reset Password: Error (" + result.GenerateErrorReport() + ")");  OnLoginErrorEvents?.Invoke();  }  } |
| --- |

### 

### 2.4 Playfab Player

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabPlayer : MonoBehaviour  {  [Header("PlayFab Settings")]  public string PlayerID;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Result Settings")]  public UnityEvent OnSuccessEvents;  public UnityEvent OnErrorEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokePlayerInformation()  {  PlayFabClientAPI.GetPlayerProfile(new GetPlayerProfileRequest(), OnGetPlayerProfileSuccess, OnGetPlayerProfileError);  }  private void OnGetPlayerProfileSuccess(GetPlayerProfileResult result)  {  PlayerID = result.PlayerProfile.PlayerId;  Debug.Log("PlayerID: " + PlayerID);  OnSuccessEvents?.Invoke();  }  private void OnGetPlayerProfileError(PlayFabError error)  {  Debug.LogError("PlayFab Error: " + error.ErrorMessage);  OnErrorEvents?.Invoke();  }  } |
| --- |

### 2.5 Playfab Leaderboard

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabLeaderboard : MonoBehaviour  {  [System.Serializable]  public class LeaderboardUI  {  public Text positionText;  public Text playFabIdText;  public Text displayNameText;  public Text displayScoreText;  }  [Header("PlayFab Settings")]  public string PlayFabID;  [Header("Statistic Settings")]  public string LeaderboardName;  public int LeaderboardStartPosition;  public Text LeaderboardValue;  [Header("Display Settings")]  public LeaderboardUI[] LeaderboardDisplay;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Success Settings")]  public UnityEvent OnSaveSuccessEvents;  public UnityEvent OnLoadSuccessEvents;  [Header("Error Settings")]  public UnityEvent OnSaveErrorEvents;  public UnityEvent OnLoadErrorEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokeSaveLeaderboard()  {  var request = new UpdatePlayerStatisticsRequest  {  Statistics = new List<StatisticUpdate>() {  new StatisticUpdate()  {  StatisticName = LeaderboardName,  Value = int.Parse(LeaderboardValue.text)  }  }  };  PlayFabClientAPI.UpdatePlayerStatistics(request, OnSaveLeaderboardSuccess, OnSaveLeaderboardError);  }  public void InvokeLoadLeaderboard()  {  var request = new GetLeaderboardRequest  {  StatisticName = LeaderboardName,  StartPosition = LeaderboardStartPosition,  MaxResultsCount = LeaderboardDisplay.Length  };  PlayFabClientAPI.GetLeaderboard(request, OnLoadLeaderboardSuccess, OnLoadLeaderboardError);  }  void OnSaveLeaderboardSuccess(UpdatePlayerStatisticsResult result)  {  Debug.Log("PlayFab Leaderboard: Success");  OnSaveSuccessEvents?.Invoke();  }  void OnSaveLeaderboardError(PlayFabError result)  {  Debug.Log("PlayFab Leaderboard: Error (" + result.GenerateErrorReport() + ")");  OnSaveErrorEvents?.Invoke();  }  void OnLoadLeaderboardSuccess(GetLeaderboardResult result)  {  for (int i = 0; i < LeaderboardDisplay.Length; i++)  {  var item = result.Leaderboard[i];  LeaderboardDisplay[i].positionText.text = (item.Position + 1).ToString();  LeaderboardDisplay[i].playFabIdText.text = item.PlayFabId;  LeaderboardDisplay[i].displayNameText.text = item.DisplayName;  LeaderboardDisplay[i].displayScoreText.text = item.StatValue.ToString();  }  OnLoadSuccessEvents?.Invoke();  }  void OnLoadLeaderboardError(PlayFabError result)  {  Debug.Log("PlayFab Leaderboard: Error (" + result.GenerateErrorReport() + ")");  OnLoadErrorEvents?.Invoke();  }  } |
| --- |

### 2.6 Playfab Data Request

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.Events;  using PlayFab;  using PlayFab.ClientModels;  public class PlayFabDataRequest : MonoBehaviour  {  [System.Serializable]  public class CDataRequest  {  public string Name;  public Text Value;  }  [Header("PlayFab Settings")]  public string PlayFabID;  [Header("Data Settings")]  public List<CDataRequest> DataRequest;  [Header("Event Settings")]  public UnityEvent StartEvents;  public UnityEvent UpdateEvents;  [Header("Success Settings")]  public UnityEvent OnSaveSuccessEvents;  public UnityEvent OnLoadSuccessEvents;  [Header("Error Settings")]  public UnityEvent OnSaveErrorEvents;  public UnityEvent OnLoadErrorEvents;  // Start is called before the first frame update  void Start()  {  StartEvents?.Invoke();  }  // Update is called once per frame  void Update()  {  UpdateEvents?.Invoke();  }  // Update is called once per frame  public void InvokeSaveDataRequest()  {  Dictionary<string, string> tempData = new Dictionary<string, string>();  for (int i = 0; i < DataRequest.Count; i++)  {  tempData.Add(DataRequest[i].Name, DataRequest[i].Value.text);  }  var request = new UpdateUserDataRequest  {  Data = tempData  };  PlayFabClientAPI.UpdateUserData(request, OnSaveSuccess, OnSaveError);  }  public void InvokeLoadDataRequest()  {  PlayFabClientAPI.GetUserData(new GetUserDataRequest(), OnLoadSuccess, OnLoadError);  }  void OnSaveSuccess(UpdateUserDataResult result)  {  Debug.Log("PlayFab Save Data Request: Success");  OnSaveSuccessEvents?.Invoke();  }  void OnSaveError(PlayFabError result)  {  Debug.Log("PlayFab Save Data Request: Error (" + result.GenerateErrorReport() + ")");  OnSaveErrorEvents?.Invoke();  }  void OnLoadSuccess(GetUserDataResult result)  {  Debug.Log("PlayFab Load UserData");  if (result.Data != null)  {  for (int i = 0; i < DataRequest.Count; i++)  {  if (result.Data.ContainsKey(DataRequest[i].Name))  {  DataRequest[i].Value.text = result.Data[DataRequest[i].Name].Value;  }  }  }  }  void OnLoadError(PlayFabError result)  {  Debug.Log("PlayFab Load Data Request: Error (" + result.GenerateErrorReport() + ")");  OnLoadErrorEvents?.Invoke();  }  } |
| --- |

## 3. Tobii Eye Tracker

Tobii Developer SDK untuk Unity adalah alat bagi para pengembang untuk mengintegrasikan teknologi eye tracking Tobii ke dalam permainan dan aplikasi mereka. Dengan SDK ini, pengembang dapat membuat pengalaman yang lebih immersif dan responsif dengan melacak gerakan mata pengguna secara real-time. Ini memungkinkan untuk menciptakan interaksi yang lebih alami, seperti mengendalikan kursor atau objek dalam permainan hanya dengan melihatnya. Dengan Tobii Developer SDK untuk Unity, pengembang memiliki akses ke berbagai fitur dan fungsi eye tracking yang inovatif untuk meningkatkan pengalaman pengguna dalam bermain game dan aplikasi yang menarik.

| **Tobii Eye Tracker** |
| --- |
| <https://developer.tobii.com/> |

## 4. Leap Motion

Ultra Leap Motion SDK untuk Unity adalah alat yang memungkinkan pengembang untuk mengintegrasikan teknologi pengenalan gerakan tangan yang canggih ke dalam aplikasi dan permainan mereka. Dengan SDK ini, pengguna dapat mengendalikan aplikasi dengan gerakan tangan mereka, menciptakan pengalaman yang lebih intuitif dan interaktif. Ini membuka pintu untuk berbagai macam aplikasi, mulai dari game hingga aplikasi edukasi dan simulasi, yang memanfaatkan kemampuan pelacakan tangan yang akurat. Ultra Leap Motion SDK untuk Unity memberikan alat yang kuat bagi pengembang untuk menciptakan aplikasi yang lebih mendalam dan menghadirkan pengalaman interaktif yang menarik bagi pengguna.

| **Leap Motion** |
| --- |
| <https://developer.leapmotion.com/unity> |

# 

# 

# 

# 

# 

# Chapter 11 Unity Troubleshooting

Pembahasan troubleshooting dalam Unity sangat penting bagi pengembang pemula, karena mereka seringkali dihadapkan pada berbagai masalah atau error saat mengembangkan game pertama mereka. Salah satu langkah pertama dalam menyelesaikan masalah ini adalah dengan memahami pesan error yang muncul. Unity akan memberikan pesan yang cukup deskriptif tentang masalah yang terjadi, seperti "null reference exception" atau "missing script." Pengembang pemula harus belajar untuk membaca pesan error ini dengan seksama, karena ini adalah kunci untuk menemukan penyebab masalah.

Selanjutnya, pengembang pemula perlu menguasai keterampilan pencarian sumber masalah. Ini melibatkan menelusuri kode mereka dengan hati-hati untuk menemukan tempat di mana error terjadi. Debugger di Unity adalah alat yang sangat berguna dalam hal ini, karena memungkinkan pengembang untuk melihat nilai-nilai variabel saat runtime dan melacak jalur eksekusi program. Dengan menggunakan debugger, pengembang pemula dapat mengidentifikasi kesalahan logika, masalah pemrograman, atau kesalahan pengejaan yang mungkin terjadi dalam kode mereka.

Terakhir, pembahasan troubleshooting harus mencakup pemahaman tentang solusi umum untuk masalah yang sering muncul dalam pengembangan game Unity. Misalnya, jika pengembang mengalami masalah dengan "null reference," mereka mungkin perlu memeriksa apakah objek yang diberikan sebagai referensi telah diinisialisasi dengan benar. Jika ada masalah dengan grafis atau tampilan, mungkin perlu memeriksa pengaturan kamera atau materi yang digunakan. Sumber daya online seperti forum Unity, dokumentasi resmi, dan komunitas pengembang yang ada juga bisa menjadi sumber yang sangat berguna dalam menemukan solusi untuk masalah yang umumnya dihadapi pengembang pemula. Dengan demikian, pemahaman mendalam tentang troubleshooting dalam Unity adalah keterampilan yang sangat berharga bagi mereka yang ingin sukses dalam pengembangan game.

# 

# 

# Chapter 12 Special Game Courses

Special Course Game adalah elemen penting dalam berbagai metode belajar tentang pengembangan game. Ini memberikan kerangka kerja yang kuat untuk memahami dengan lebih dalam dunia kompleks pengembangan game. Ketika berbicara tentang variasi pembelajaran game, Special Course menonjol sebagai pilihan yang sangat menarik. Hal ini terutama disebabkan oleh kurikulum yang dapat disesuaikan sepenuhnya dengan kebutuhan penyelenggara dan beragam modul yang diajarkan.

Bayangkan Anda adalah seorang pemula yang memiliki impian untuk menjadi seorang pengembang game. Game Special Course memungkinkan Anda untuk memilih modul yang sesuai dengan minat dan tujuan Anda. Anda dapat mulai dengan modul dasar, seperti pemrograman dasar dalam Unity, Unreal, atau Godot, dan kemudian berkembang ke arah yang lebih spesifik, seperti animasi karakter atau desain level. Anda bahkan bisa mencoba modul-modul yang lebih lanjut, seperti pengembangan game VR atau AR. Semua pilihan ini memberikan fleksibilitas yang sangat besar dalam pengembangan keahlian Anda sesuai dengan apa yang Anda impikan dalam dunia game.

Selain itu, Special Game Courses juga menawarkan manfaat tambahan yang sangat berharga, seperti dukungan intensif dari pengajar yang berpengalaman dan akses ke komunitas pengembang yang dapat saling mendukung. Ini adalah tempat yang sempurna untuk bertanya, berdiskusi, dan berkembang bersama dengan orang-orang yang memiliki minat yang sama dalam pengembangan game. Dengan demikian, Special Game Courses bukan hanya menjadi elemen penting dalam berbagai metode pembelajaran game, tetapi juga merupakan pintu masuk untuk mencapai tingkat keahlian yang lebih tinggi dalam dunia pengembangan game yang kompetitif dan penuh tantangan.

# 

# 

# 

# Game Ideation Index

# 

## Game Premis

1. Seorang anak lelaki (S) berusaha mengalahkan (P) seorang pangeran kegelapan (O) yang membunuh kedua orang tuanya (K)
2. Seorang anak lelaki (S) berusaha menghancurkan (P) sebuah cincin keramat (O) di gunung terkutuk (K)
3. Seorang anak lelaki (S) mendapatkan tugas (P) untuk mendapatkan pedang suci (O) di suatu hutan belantara (K)
4. Seorang kapten kapal bajak laut (S) berusaha mencari harta karun legendaris (P) yang tersembunyi di pulau terlarang (O) yang dihuni oleh makhluk-makhluk laut ganas (K).
5. Seorang pemburu harta karun (S) berusaha menemukan (P) harta karun kapal bajak laut (O) yang hilang (K).
6. Seorang ahli arkeologi (S) berusaha mengungkap misteri (P) piramida kuno (O) di tengah padang pasir yang terpencil (K)
7. Seorang prajurit wanita (S) berusaha menyelamatkan (P) kerajaan yang terancam oleh seorang penyihir jahat (O) yang mengutuk keluarganya (K).
8. Seorang penjelajah luar angkasa (S) berusaha menemukan (P) artefak misterius (O) di asteroid terjauh dalam tata surya (K).
9. Seorang penyihir muda (S) diberikan misi (P) untuk mengembalikan (O) buku mantra kuno (K) yang dicuri oleh makhluk magis jahat.
10. Seorang agen rahasia (S) berusaha menghentikan (P) kelompok teroris internasional (O) yang mengancam keamanan dunia, sementara menjaga identitasnya (K).
11. Seorang detektif swasta (S) ditugaskan untuk memecahkan (P) serangkaian pembunuhan misterius (O) yang terjadi di sebuah kota terpencil (K).
12. Seorang penjelajah waktu (S) berusaha untuk menggagalkan (P) peristiwa bersejarah yang akan merusak aliran waktu (O) setelah perang besar (K).
13. Seorang penyihir penyembuh (S) berusaha menyembuhkan (P) penyakit misterius (O) yang menyebar dengan cepat di seluruh kerajaan (K).
14. Seorang penyelidik ilmiah (S) diberikan tugas untuk mengeksplorasi (P) fenomena paranormal (O) di sebuah laboratorium penelitian yang terpencil (K).
15. Seorang detektif cyber (S) berusaha melacak (P) seorang peretas jaringan yang sangat berbahaya (O) yang mencuri data pribadinya (K).
16. Seorang petualang (S) berusaha menemukan (P) harta terpendam (O) di dalam gua terlarang (K).
17. Seorang pemuda (S) mencoba membebaskan (P) kerajaannya yang direbut (O) oleh pasukan penjajah (K).
18. Seorang penyihir (S) berusaha menyelamatkan (P) desanya dari kutukan (O) yang dikenakan oleh seorang dukun jahat (K).
19. Seorang agen rahasia (S) disusupkan (P) ke dalam organisasi kriminal (O) untuk menggulingkan pemimpinnya yang korup (K).
20. Seorang prajurit (S) berusaha mengalahkan (P) naga legendaris (O) yang mengancam kerajaannya (K).
21. Seorang ahli biologi (S) melakukan eksperimen (P) dengan organisme eksotis (O) di laboratorium terpencil (K).
22. Seorang petualang luar angkasa (S) berusaha menghindari (P) asteroid berbahaya (O) yang mengancam stasiun ruang angkasa (K).
23. Seorang prajurit samurai (S) melindungi (P) kaisar dari usaha pembunuhan (O) oleh kelompok pemberontak (K).
24. Seorang insinyur mekanik (S) berusaha memperbaiki (P) mesin utama (O) di dalam kapal ruang angkasa yang terperangkap dalam lubang hitam (K).
25. Seorang penulis (S) mencari inspirasi (P) untuk penulisan novelnya (O) di sebuah rumah berhantu (K).
26. Seorang agen kesehatan masyarakat (S) berusaha menghentikan (P) wabah virus misterius (O) yang menyebar di seluruh kota (K).
27. Seorang detektif paranormal (S) berusaha menyelidiki (P) aktivitas supranatural (O) di sebuah hotel tua yang angker (K).
28. Seorang penjelajah waktu (S) berusaha menghindari (P) peristiwa bersejarah yang akan mengubah masa depan (O) setelah perang besar (K).
29. Seorang arkeolog (S) mencari (P) reruntuhan kota kuno (O) di dalam hutan yang belum terjamah (K).
30. Seorang penyelidik ilmiah (S) berusaha memahami (P) fenomena alam yang aneh (O) di suatu tempat terpencil (K).
31. Seorang penyihir muda (S) belajar (P) ilmu sihir (O) untuk menyelamatkan kerajaannya (K) dari serangan makhluk gelap.
32. Seorang perampok ulung (S) berusaha mencuri (P) berlian besar (O) dari museum terkenal (K).
33. Seorang pendaki gunung (S) berusaha mencapai (P) puncak gunung tertinggi (O) dalam ekspedisi berbahaya (K).
34. Seorang penyelam (S) mencoba mengeksplorasi (P) reruntuhan kota kuno (O) di bawah laut yang belum pernah dijelajahi (K).
35. Seorang kapten kapal (S) berusaha menghadapi (P) badai besar (O) di tengah samudra yang ganas (K).
36. Seorang prajurit ninja (S) berusaha menyusup (P) ke benteng musuh (O) untuk mengambil informasi rahasia (K).
37. Seorang pilot pesawat luar angkasa (S) harus mencegah (P) asteroid besar (O) menabrak planetnya (K).
38. Seorang arkeolog (S) berusaha mengungkap misteri (P) piramida melayang (O) di atas gunung misterius (K).
39. Seorang agen rahasia (S) berusaha menghentikan (P) serangan teroris (O) yang mengancam konferensi perdamaian dunia (K).
40. Seorang prajurit Viking (S) berusaha merebut kembali (P) desanya (O) dari pasukan penjajah (K).
41. Seorang petualang luar angkasa (S) berusaha menyelidiki (P) pesawat ruang angkasa hilang (O) di daerah antariksa yang belum dijelajahi (K).
42. Seorang penyelidik (S) berusaha memecahkan misteri (P) pencurian barang berharga (O) dari museum terkenal (K).
43. Seorang petualang (S) berusaha menemukan (P) kunci ke portal dimensi lain (O) di dalam labirin bawah tanah yang mematikan (K).
44. Seorang penyelam laut dalam (S) berusaha mengeksplorasi (P) reruntuhan kapal kuno (O) yang terdampar di dasar laut dalam (K).
45. Seorang agen antariksa (S) berusaha menyelamatkan (P) stasiun luar angkasa yang terancam (O) oleh gangguan teknis misterius (K).
46. Seorang pelancong waktu (S) berusaha mengubah (P) masa lalu untuk menyelamatkan (O) krisis global yang sedang terjadi (K).
47. Seorang penjelajah dunia (S) berusaha mencari (P) artefak hilang (O) di dalam gua es yang berbahaya (K).
48. Seorang detektif magis (S) berusaha mengungkap (P) pembunuhan misterius (O) yang melibatkan sihir gelap (K).
49. Seorang ksatria (S) berusaha menyelamatkan (P) putri kerajaan yang diculik (O) oleh naga besar (K).
50. Seorang agen rahasia (S) mendapatkan misi (P) untuk menghentikan (O) rencana teroris yang melibatkan senjata nuklir (K).
51. Seorang petualang (S) berusaha mencari (P) harta karun kuno (O) di dalam kuil yang tersembunyi di dalam hutan (K).
52. Seorang penyelidik ilmiah (S) berusaha memecahkan (P) misteri fenomena alam aneh (O) di sebuah pulau terpencil (K).
53. Seorang penyihir muda (S) berusaha melawan (P) serangan makhluk jahat (O) yang ingin menaklukkan dunia (K).
54. Seorang kapten kapal bajak laut (S) berusaha menghadapi (P) serangan armada musuh (O) di tengah laut yang badai (K).
55. Seorang ilmuwan luar angkasa (S) berusaha mengeksplorasi (P) planet asing yang belum pernah dijelajahi (O) dengan robot penjelajah (K).
56. Seorang pemburu monster (S) berusaha memburu (P) monster legendaris (O) yang menghantui desa (K).
57. Seorang agen mata-mata (S) mendapatkan tugas (P) untuk mengambil dokumen rahasia (O) dari kantor perusahaan korup (K).
58. Seorang penyelidik paranormal (S) berusaha menangkap (P) hantu yang meresahkan sebuah rumah tua (O) di kota kecil (K).
59. Seorang prajurit samurai (S) berusaha mempertahankan (P) desanya dari serangan tentara musuh (O) yang kuat (K).
60. Seorang peneliti biologi (S) berusaha mencari (P) obat untuk penyakit misterius (O) yang menyerang penduduk desa terpencil (K).
61. Seorang pemimpin pemberontak (S) berusaha menggulingkan (P) pemerintahan tirani (O) yang telah lama berkuasa (K).
62. Seorang penjelajah dimensi (S) berusaha untuk kembali (P) ke dunianya sendiri (O) setelah tersesat di dimensi paralel (K).
63. Seorang petualang (S) berusaha untuk melarikan diri (P) dari labirin maut (O) yang diciptakan oleh seorang jenius psikopat (K).
64. Seorang mahasiswa jurusan ilmu komputer (S) berusaha untuk membongkar (P) jaringan peretas yang mengancam keamanan data kampus (O).
65. Seorang mahasiswa seni (S) berusaha menyelidiki (P) hilangnya karya seni berharga (O) dari galeri seni kampus (K).
66. Seorang mahasiswa ilmu pengetahuan alam (S) diberi tugas (P) untuk menjelajahi hutan terpencil di sekitar kampus (O) untuk penelitian ilmiah (K).
67. Seorang mahasiswa kedokteran (S) berusaha menyelamatkan (P) teman-temannya yang terinfeksi (O) oleh virus misterius yang menyebar di kampus (K).
68. Seorang mahasiswa sastra (S) berusaha mengungkap (P) misteri tulisan tangan aneh (O) yang ditemukan di perpustakaan kampus (K).
69. Seorang mahasiswa olahraga (S) berusaha memimpin timnya (P) menuju kemenangan di turnamen nasional (O) untuk mendapatkan pengakuan bagi kampusnya (K).
70. Seorang mahasiswa psikologi (S) diberi tugas (P) untuk merawat dan membantu pasien dengan gangguan mental (O) di klinik psikologi kampus (K).
71. Seorang mahasiswa arkeologi (S) berusaha menggali (P) situs arkeologi kampus (O) untuk menemukan artefak bersejarah yang berharga (K).
72. Seorang mahasiswa musik (S) berusaha untuk menemukan (P) musik dari komposer legendaris yang hilang (O) di dalam kampus yang luas (K).
73. Seorang mahasiswa jurusan sosiologi (S) mencoba mengatasi (P) masalah konflik antar-kelompok di kampus (O) untuk menciptakan perdamaian (K).
74. Seorang mahasiswa hukum (S) berusaha membela (P) seorang teman yang salah dituduh (O) dalam kasus hukum kontroversial di kampus (K).
75. Seorang mahasiswa teknik (S) mencoba untuk merancang (P) mesin inovatif (O) yang dapat membantu meningkatkan efisiensi di kampus (K).
76. Seorang mahasiswa politik (S) berusaha untuk memenangkan (P) pemilihan ketua mahasiswa (O) dan memimpin perubahan positif di kampus (K).
77. Seorang mahasiswa jurnalistik (S) berusaha mengungkap (P) skandal besar yang melibatkan pejabat kampus (O) dalam penyelidikan jurnalisme investigasi (K).
78. Seorang mahasiswa kedokteran gigi (S) berusaha menyelidiki (P) serangkaian kasus misterius kehilangan gigi (O) di kampus (K).
79. Seorang mahasiswa ilmu politik (S) berusaha mengorganisir (P) protes mahasiswa besar-besaran (O) untuk memperjuangkan perubahan di kampus (K).
80. Seorang mahasiswa teater (S) berusaha mempertunjukkan (P) pertunjukan besar terakhir di aula besar kampus (O) sebelum dirobohkan (K).
81. Seorang mahasiswa kedokteran (S) berusaha untuk menemukan (P) obat untuk wabah penyakit misterius (O) yang menyerang kampus (K).
82. Seorang mahasiswa biologi (S) berusaha menyelamatkan (P) lingkungan alam kampus (O) dari perusakan oleh perusahaan konstruksi (K).
83. Seorang mahasiswa astronomi (S) mencoba mengungkap (P) misteri penampakan fenomena langit yang tak terduga (O) di atas kampus (K).
84. Seorang koki (S) mencoba untuk mengelola (P) restoran yang penuh dengan pelanggan aneh dan permintaan makanan yang aneh (O).
85. Seorang agen perjalanan (S) harus mengatur (P) liburan yang sangat kacau (O) untuk sekelompok pelanggan yang tidak kooperatif (K).
86. Seorang ilmuwan gila (S) menciptakan (P) perangkat aneh (O) yang menghasilkan efek samping komikal di seluruh kota (K).
87. Seorang aktor/aktris (S) berusaha untuk mendapatkan (P) peran utama dalam sebuah pertunjukan besar (O) yang dipenuhi dengan kesalahan panggung dan kebingungan (K).
88. Seorang penulis humor (S) berusaha mencari (P) inspirasi untuk bukunya (O) dengan berbagai situasi lucu dalam kehidupan sehari-hari (K).
89. Seorang agen perjalanan (S) harus mengatur (P) perjalanan rombongan yang sangat eksentrik (O) ke destinasi eksotis yang penuh dengan kejadian lucu (K).
90. Seorang pemilik restoran (S) mencoba untuk menyelamatkan (P) restorannya (O) dengan menyajikan menu yang unik dan menggelikan (K).
91. Seorang ilmuwan sosial (S) berusaha untuk memecahkan (P) teka-teki perilaku manusia yang aneh (O) dalam eksperimen sosial yang aneh (K).
92. Seorang pemain teater (S) harus berurusan dengan (P) drama dan kekacauan di balik panggung (O) selama produksi drama komedi yang besar (K).
93. Seorang penyelidik (S) terjebak dalam (P) konspirasi konyol (O) yang melibatkan tokoh-tokoh aneh dalam pemerintahan (K).
94. Seorang pengacara (S) harus mempertahankan (P) klien yang terlibat dalam kasus hukum yang sangat aneh (O) yang menghasilkan berbagai insiden komikal (K).
95. Seorang guru (S) mencoba mengelola (P) kelas penuh anak-anak yang sangat nakal (O) dengan metode pengajaran yang tidak biasa (K).
96. Seorang petugas kebersihan (S) harus membersihkan (P) kantor pusat perusahaan yang terkenal berantakan (O) setelah pesta besar (K).
97. Seorang ilmuwan (S) mencoba memecahkan (P) misteri makhluk luar angkasa (O) yang ternyata adalah makhluk aneh yang berperilaku konyol (K).
98. Seorang guru seni (S) berusaha menginspirasi (P) murid-muridnya dengan proyek seni yang aneh dan kreatif (O) di sekolah seni (K).
99. Seorang agen real estat (S) mencoba menjual (P) rumah yang penuh dengan hantu (O) kepada pembeli yang skeptis (K).

## Visual Novel & Cutscene Dialog

1. Tampilan Cutscene Opening-Ending Skip
2. Tampilan Strip Comic-Skip
3. Tampilan Animation Comic-Skip
4. Tampilan Motion Comic-Skip
5. Tampilan Tutorial GamePlay
6. Tampilan Visual Dialog Per Trigger
7. Visual Dialog 2D Classic
8. Visual Dialog 2D-Live
9. Visual Dialog 3D-Modern
10. Visual Dialog Pilihan Berdasarkan Event

## Avatar Controlling

1. Character Manipulation (\*)
2. Bergerak 4Direction WASD
3. Bergerak 4Direction Point Click
4. Berlari-Berguling
5. Berjongkok-BerjongkokMerayap
6. Tiarap-TiarapMerayap
7. Charging kecepatan lari
8. Loncat1X-Loncat2X
9. Loncat-Dash di udara
10. Terjun payung-Melayang ke bawah
11. Loncat-Terbang ke atas
12. DashPendek-DashPanjang
13. Menendang Dash-di-tembok
14. Parkour antar 2 dinding
15. Memanjat tangga
16. Memanjat tepian batas
17. Memanjat bawah jembatan
18. Bergelantungan menggunakan akar pohon
19. Menggunakan equipment flying-hook
20. Sliding via bawah platform
21. Sliding via kemiringan platform
22. Berenang-Menyelam
23. Hampir jatuh di ujung platform
24. Bergantungan di ujung platform
25. Membidik kiri-kanan
26. Membidik atas-bawah
27. Membidik berdasarkan arah gerakan mouse
28. Menggunakan senter
29. Memukul-Terpukul
30. Menebas-Tertebas
31. Menembak-Tertembak
32. Bertahan dari serangan
33. Charging action
34. Bersembunyi dalam item
35. Menghancurkan benda di atas
36. Menginjak musuh dari atas
37. Melabrak musuh dari samping
38. Mendorong benda di samping
39. Menarik benda di samping
40. Terkena hambatan mematikan
41. Membidik suatu target lurus-lengkung
42. Mengambil-Menggunakan item
43. Membuka-Mengunci pintu
44. Membuka-Mengunci lemari
45. Mengambil item di bawah
46. Teleport menggunakan portal
47. Teleport berdasarkan proyektil
48. Menggunakan lift-benda floating
49. Berganti player untuk atasi rintangan platform
50. Berganti baju/alat untuk atasi rintangan platform
51. Area platform bisa berputar 360 derajat
52. Melontarkan diri ala bola pinball
53. Terdorong ke oleh angin
54. Terbakar oleh api
55. Basah oleh air
56. Terjebak dalam pasir hisap
57. Menciptakan platform yang bisa dipijak
58. Bergerak lambat oleh waktu
59. Menaiki binatang/kendaraan
60. Rewind kembali ke waktu sebelumnya
61. Serangan combo
62. Sekarat-Mati

## Calculation & Internal Economy

1. Kalkukasi-Visualisasi Score
2. Kalkukasi-Visualisasi Star Per Level
3. Kalkukasi-Visualisasi Koin
4. Kalkukasi-Visualisasi Timer
5. Kalkukasi-Visualisasi Health Point
6. Kalkukasi-Visualisasi Mana Point
7. Kalkukasi-Visualisasi Experience Point
8. Kalkukasi-Visualisasi Stamina Point
9. Kalkukasi-Visualisasi Str Point
10. Kalkukasi-Visualisasi Dex Point
11. Kalkukasi-Visualisasi Agi Point
12. Kalkukasi-Visualisasi Int Point
13. Kalkukasi-Visualisasi Wis Point
14. Kalkukasi-Visualisasi Luck Point
15. Kalkukasi-Visualisasi Inventory
16. Kalkukasi-Visualisasi Skill
17. Dashboard Analitik Playfab

## Gameplay Balancing & Optimalization

1. Addressable Loading dalam satu project
2. Occlusion Culling dalam satu scene
3. Frustum Culling dalam satu scene
4. Impostor Generator dalam satu scene
5. Frame Rate Acceleration
6. Level of Detail (LOD) Mechanism
7. Resources Loading Runtime
8. Shader Optimalization
9. Lightning Runtime vs Bake
10. Crowd Entity Animation
11. Memory Process Reducer
12. Particle & Feel Optimalization

## 

# 

# 

# Game Generation Index

# 

## Text SFX Creator

* CoolText: <https://cooltext.com/>
* FlammingText: <https://flamingtext.com/>
* TextStudio: <https://www.textstudio.com/>
* CoolTextGenerator: <https://maketext.io/>
* TextFX: <https://www.textfx.co/>

## 

## Character Creator

* VRoid Studio: <https://studio.vroid.com/>
* Ready Player Me: <https://readyplayer.me/>
* Daz 3D: <https://www.daz3d.com/>
* MakeHuman: <http://www.makehumancommunity.org/>
* Tafi Avatar: <https://www.tafi.co/avatar>
* Character Creator 3: <https://www.reallusion.com/character-creator/>
* Mixamo/Fuse: <https://www.mixamo.com/>
* ManuelbastioniLAB (plugin untuk Blender): <https://www.manuelbastioni.com/>
* Reallusion iClone: <https://www.reallusion.com/iclone/>

## 

## Text Generator

* ChatGPT: <https://chat.openai.com/>
* JasperAI: <https://www.jasper.ai/>
* RytrAI: <https://rytr.me/>
* WriteSonic: <https://writesonic.com/>
* ScaleNut: <https://www.scalenut.com/>
* SiderAI: <https://sider.ai/>
* PopAI: <https://www.popai.pro/>
* JenniAI: <https://jenni.ai/>
* CopyAI: <https://www.copy.ai/tools>
* Simplified: <https://simplified.com/ai-writer/>
* Piscart: <https://tools.picsart.com/text/ai-writer/>
* Smodin: <https://smodin.io/writer>
* Wepik: <https://wepik.com/ai-writer>
* Ahref: <https://ahrefs.com/writing-tools/paragraph-generator>

| **Prompt AI : Storytelling** |
| --- |
| A young boy's quest to defeat a Dark Prince who had murdered his parents. |

| **Prompt AI : Coding** |
| --- |
| Make a code for controlling camera orbit an object using mouse movement. |

## 

## Text To Speech Generator

* ProsaAI: [www.prosa.ai](http://www.prosa.ai)
* TTS Free: <https://ttsfree.com/text-to-speech>
* Free TTS: <https://freetts.com/>
* TTS Maker: <https://ttsmaker.com/>
* Narakeet: <https://www.narakeet.com/>
* Botika: <https://botika.online/TextToSpeech/index.php>
* MurfAI: <https://murf.ai/text-to-speech>
* Revoicer: <https://revoicer.com/>
* SpeechGen: <https://speechgen.io/>
* TeksKu: <https://teksku.com/>
* Resemble: <https://www.resemble.ai/>
* Synthesia: <https://www.synthesia.io/text-to-speech>
* ClimpChamp: <https://clipchamp.com/>
* Text2Speech: <https://www.text2speech.org/>
* Hearling: <https://hearling.com/>

| **Prompt AI : Storytelling** |
| --- |
| Once upon a time, in a peaceful village nestled between rolling hills and dense forests, lived a young boy named Ethan. He was known for his kind heart and gentle nature, but his life took a dark turn when a sinister figure known as the Dark Prince descended upon their village. The Dark Prince, shrouded in shadows, had a malevolent desire for power and had mercilessly slain Ethan's parents. |

## Image Generator

* Tensir Art: <https://tensor.art/>
* ClipDrop: <https://clipdrop.co/stable-diffusion>
* GenCraft: <https://gencraft.com/>
* HotpotAI: <https://hotpot.ai/art-generator>
* ClaidAI: <https://claid.ai/>
* WePikAI: <https://wepik.com/ai>
* PicsArt: <https://picsart.com/ai-image-generator/>
* DeepAI: <https://deepai.org/machine-learning-model/text2img>
* FreeImageGenerator: <https://freeimagegenerator.com/>
* Simplified: <https://simplified.com/ai-image-generator/>
* Crayon: <https://www.craiyon.com/>
* Fotor: <https://www.fotor.com/ai-image-generator/>
* TopspotAI: <https://topspotai.com/>
* GrooveAI: <https://groove.ai/>

| **Prompt AI : Character** |
| --- |
| A teenager using red news boy hat & red jacket infront of town plaza |

| **Prompt AI : Place** |
| --- |
| A town plaza with a lot of food court and bazaar arena |

## Video Generator

* HeyGen: <https://www.heygen.com/>
* LeilaPix: <https://convert.leiapix.com/>
* InvideoAI: <https://invideo.io/>
* HouroneAI: <https://hourone.ai/>
* VeedIO: <https://www.veed.io/>
* Kapwng: <https://www.kapwing.com/>
* SteveAI: <https://www.steve.ai/>
* DeepBrain: <https://www.deepbrain.io/>
* Elai: <https://elai.io/>

## 

# Daftar Pustaka

Djaouti, D., Alvarez, J., Jessel, J.-P., & Rampnoux, O. (2011). Origins of serious games. In M. Ma, A. Oikonomou, & L. C. Jain (Eds.), Serious Games and Edutainment Applications (pp. 25–43). London, UK: Springer London. doi:10.1007/978-1-4471-2161-9\_3

Hellerstedt, Andreas & Mozelius, Peter. (2019). Game-based learning - a long history.

Bilgin, Enes. (2020). A Brief Overview on Gamification History.

H. M. Chandler, "Game Production Handbook" (Book style), Sudbury: Jones and Bartletts Publishers, 2010.

Plass, Jan & Homer, Bruce & Kinzer, Charles. (2015). Foundations of Game-Based Learning. Educational Psychologist. 50. 258-283. 10.1080/00461520.2015.1122533.

Roedavan, Rickman et al. Zetcil: Game Mechanic Framework for Unity Game Engine. IJAIT (International Journal of Applied Information Technology), v. 3, n. 02, p. 96-105, July 2020. ISSN 2581-1223.

Roedavan, R., Pudjoatmodjo, B., Siradj, Y., Salam, S., & Hardianti, B. D. (2021). Serious Game Development Model Based on the Game-Based Learning Foundation. Journal of ICT Research and Applications, 15(3), 291-305. https://doi.org/10.5614/itbj.ict.res.appl.2021.15.3.6

Roedavan, R., Siradj, Y., & Stefany, S. (2023). Educational Game Scenario Model Based On Imperative Game Goal Typology. Journal of Games Game Art and Gamification, 8(1), 18-23. doi:10.21512/jggag.v8i1.9497