VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS KATEDRA

Kursinis projektas

# Kompiuterinių žaidimų kūrimas naudojant Unity platformą

(Development of computer games using the Unity platform)

Atliko: 4 kurso, 2 grupės studentas

Andrius Svilys

Darbo vadovas:

Dr. Igor Katin

Vilnius

2023

## Turinys

[Turinys 1](#_Toc157326813)

[Įvadas 3](#_Toc157326814)

[Sąvokos 4](#_Toc157326815)

[1. Analizė 4](#_Toc157326816)

[1.1. Keliavimas tarp žaidimo lygių 4](#_Toc157326817)

[1.2. Žaidimo būsenų išsaugojimo ir atkūrimo sistema 4](#_Toc157326818)

[1.3. Meniu langai 5](#_Toc157326819)

[1.4. Durų uždangos 6](#_Toc157326820)

[1.5. Veikėjo aptikimas 6](#_Toc157326821)

[2. Metodika 7](#_Toc157326822)

[2.1. „ Unity“ savybės 7](#_Toc157326823)

[2.1.1. SceneManager klasė 7](#_Toc157326824)

[2.1.1. DontDestroyOnLoad metodas 7](#_Toc157326825)

[2.1.2. Time klasė 8](#_Toc157326826)

[2.1.3. Coroutine 8](#_Toc157326827)

[2.1.4. Physics.Raycast 9](#_Toc157326828)

[2.2. Kitos bibliotekos 9](#_Toc157326829)

[2.2.1. DOTween 9](#_Toc157326830)

[2.2.2. Newstonsoft.json 10](#_Toc157326831)

[3. Įgyvendinimas 10](#_Toc157326832)

[3.1. Keliavimas tarp žaidimo lygių 10](#_Toc157326833)

[3.2. Meniu langai 11](#_Toc157326834)

[3.3. Žaidimo būsenų išsaugojimo ir atkūrimo sistema 13](#_Toc157326835)

[3.3.1. Būsenų pastovumas tarp scenų 13](#_Toc157326836)

[3.3.2. Būsenų pastovumas tarp žaidimo sesijų 14](#_Toc157326837)

[3.4. Durų uždangos 17](#_Toc157326838)

[3.5. Veikėjo aptikimas 17](#_Toc157326839)

[4. Rezultatai ir išvados 18](#_Toc157326840)

[Priedai 20](#_Toc157326841)

## Įvadas

„DungeonBoy“ yra trimatis veiksmo žaidimas, kurio tikslas yra padėti žaidimo veikėjui pabėgti iš vaiduoklių apsėsto požemio.

Požemiuose patruliuojantys priešai ieško ir, radę, atakuoja veikėją. Veikėjas taip pat gali šaudyti į priešus. Kovos sistema pasižymi atributiniu žalos darymu: kovų dalyviai turi atsparumą arba silpnybę keturiems atributinės žalos tipams, kurie įvertinti koeficientais. Šaudmenys, naudojami kovose, turi bazinę žalos vertę ir atributinės žalos koeficientus. Galutinė žala apskaičiuojama bazinę šovinio žalą padauginus iš šovinio ir taikinio atributų koeficientų sumos. Skirtingi tipų priešai turi skirtingus atributinius koeficientus ir naudoja skirtingų tipų šaudmenis. Žaidėjas gali keisti veikėjo atributus jam uždėjus žaidimo pasaulyje randamas kepuraites. Ši sistema kovoms suteikia strategijos elementą, nes žaidėjas, teisingai pasirinkdamas šaudmenis ir kepuraites, gali didinti daromą ir mažinti patiriamą žalą. Panašios žalos darymo sistemos sutinkamos daugelyje žaidimų, pavyzdžiui „Pokémon“ ar „Shin Megami Tensei“.

Be šaudmenų ir kepuraičių, žaidėjas taip pat gali rasti gyvybes atstatančius eliksyrus bei spalvotus raktus, kurie atrakina atitinkančios spalvos duris. Veikėjas gali rinkti šiuos daiktus ir laikyti savo inventoriuje. Daiktų skaičius ribotas, o daikto panaudojimas pašalina jį iš inventoriaus. Tai sukuria žaidybinį resursų valdymo elementą.

Šis žaidimas kuriamas naudojant „Unity“ žaidimų kūrimo karkasą.­

Išvardytos žaidimo ypatybės buvo įgyvendintos pirmojo kursinio darbo metu, tačiau „DungeonBoy“ nėra baigtas. Šio kursinio projekto tikslas yra tęsti „DungeonBoy“ kūrimą ir sukurti tokį sistemų rinkinį, kurio pakaktų žaidimų dizaineriui kurti naujus žaidimo lygius naudojantis vien tik „Unity“ redaktoriumi ir, neatsiradus naujiems reikalavimams, nerašant naujo programinio kodo.

Šiam tikslui pasiekti keliami tokie uždaviniai:

* Sukurti būdą veikėjui keliauti iš vieno žaidimo lygio į kitą
* Sukurti žaidimo būsenų išsaugojimo ir atkūrimo sistemą
* Sukurti meniu langus žaidimo valdymui
* Sukurti uždangas, kurios slėps žaidimo erdvę kitapus užvertų durų
* Pataisyti priešų naudojamą veikėjo aptikimo algoritmą

Skyriuje *Analizė* detaliai išnagrinėti užsibrėžti uždaviniai. Kur įmanoma, pateikiami aktualūs pavyzdžiai iš žinomų žaidimų.

Skyriuje *Metodika* aprašytos „Unity“ savybės ir dvi kitų kūrėjų pateiktos bibliotekos, kurios bus naudojamos uždavinių įgyvendinimui.

Skyriuje *Įgyvendinimas* aprašyta kaip Analizėje detalizuoti projekto uždaviniai įgyvendinti naudojantis Metodikoje aprašytais įrankiais.

## Sąvokos

**Scena**: aplinka, kurioje egzistuoja žaidimo objektai

**Žaidimo objektas**: scenoje egzistuojantis objektas, kuriame talpinami komponentai

**Komponentas**: programinis kodas, kuris gali būti priskirtas žaidimo objektui

**Skriptas**: žaidimo kūrėjo sukurtas komponentas

**Žaidimo sesija**: laiko tarpas tarp žaidimo aplikacijos paleidimo ir išjungimo

**Įvykių funkcijos**: funkcijos, kurios sudaro konvejerį naujo kadro apskaičiavimui

**Žaidimo lygis**: scena, kurioje žaidėjas valdo veikėją ir gali sąveikauti su žaidimo pasaulio aplinka

1. Analizė
   1. Keliavimas tarp žaidimo lygių

Kompiuterinio žaidimo veikimą galima palyginti su filmo veikimu: keičiantis kadrams, sukūriamas tolygaus veiksmo įspūdis. Tačiau priešingai nei filmo, kompiuterinio žaidimo kadrai nėra iš anksto žinomi ir keičiasi atitinkamai pagal žaidėjo įvestis. Kompiuterinio žaidimo kadro sukūrimą galima suskirstyti į du etapus: skaičiavimą (objektų būsenų kitimas priklausomai nuo praėjusio laiko ir žaidėjų įvesčių) ir objektų bei vaizdo efektų atvaizdavimą ekrane (ekrano pikselių spalvų skaičiavimas).

Scenoje egzistuojančių objektų skaičius tiesiogiai įtakoja kadro apskaičiavimo greitį — kuo objektų daugiau, tuo ilgiau užtrunka skaičiavimas. Net jei objektas nepatenka į kameros matymo lauką ar yra užstojamas sienos, jo veikimo logika vistiek apdorojama.

Piešimo etape, objektai, kurie nepatenka į kameros prizmės erdvę, nėra įtraukiami į atvaizdo skaičiavimą. Ši ypatybė yra vadinama kameros prizmės okliuzija. Reguliuojant atstumą tarp priekinės ir galinės kameros prizmės ribojamas objektų, kurie gali patekti į ją, skaičius. Kuo mažiau objektų patenka į kameros prizmę, tuo mažesnis vaizdo apskaičiavimo sudėtingumas. Dėl grafinio procesoriaus pajėgumo trūkumo, daugelis ankstyvųjų trimačių žaidimų pasižymėjo labai maža kameros prizme.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| *Iliustracija 1*. Kameros prizmės okliuzija. Apatiniame dešiniajame kampe matoma scenos projekcija. Kadagi dvi raudonos sferos nepatenka į kameros prizmės ribojamą erdvę, jos nėra atvaizduotos projekcijoje. |

Be kameros prizmės okliuzijos, taip pat verta paminėti objektų okliuziją. Įprastai, visi objektai, kurie patenka į kameros prizmę yra įtraukiami į vaizdo skaičiavimą, net jei jie nėra matomi, pavyzdžiui yra už sienos ar pilnai užstojami kelių priešais stovinčių objektų. Objektų okliuzija yra technika, kuri leidžia piešimo konvejeriui apdoroti tik tuos objektus, kurie patenka į kameros prizmę.

Daugelis modernių žaidimų kūrimo karkasų, tame tarpe ir „Unity“, pateikia įrankius objektų okliuzijai. Yra keletas skirtingų algoritmų, įgyvendinančių šią techniką, tačiau verta žinoti, kad objektų okliuzijos skaičiavimas ne visada atsiperka. Pavyzdžiui, jei scenoje nėra pakankamai daug užstojamų objektų arba užstojantis objektas juda scenoje, objektų okliuzijos skaičiavimas gali pareikalauti daugiau resursų, nei jos nenaudojant.

|  |
| --- |
| A screenshot of a computer generated image  Description automatically generated |
| *Iliustracija 2*. Objektų okliuzija. Nors projekcijoje matoma tik viena sfera, vaizdo apskaičiavimo trukmei įtakos turėjo viskas, kas patenka į kameros prizmę. |

Dar vienas būdas sumažinti vaizdo skaičiavimų sudėtingumą yra detalumo lygių (angliškai LOD, Level of Detail) technika. Kamerai artėjant prie objekto, objekto tinklelis ir tekstūros keičiamos iš mažiau detalių į labiau detalius. Ši technika yra naudinga, kai žaidimas pasižymi aukštos raiškos tinkleliais ir tekstūromis. „DungeonBoy“ atveju ši technika nebus aktuali.

Kameros prizmės mažinimas, objektų okliuzija ir LOD yra technikos, naudojamos optimizuoti GPU skaičiavimų sudėtingumą. Tačiau kadro skaičiavimams taip pat reikalingi CPU ir RAM resursai.

Žaidimo pasaulio skaidymas į lygius yra būdas, kuris sumažina žaidimo reiklumą visiems kompiuterio resursams. Vienu metu užkraunama mažesnės apimties scena, taigi tiek loginių, tiek vaizdo skaičiavimų sudėtingumas sumažėja ir tampa lengviau išvengti delsos tarp kadrų atvaizdavimo. Be to, žaidimo suskirtymas į lygius žaidėjui leidžia lengviau pajusti progreso jausmą.

„Unity“ galimi du scenos užkrovimo būdai: vienetinis arba sudėtinis. Vienetinis būdas sunaikina aktyvią sceną ir užkrauną sekančią. Sudėtinis būdas leidžia prijungti vieną sceną prie kitos. Šis būdas yra parankus, kai norima pratęsti lygį žaidėjui nepastebint. Toks scenų užkrovimo būdas labai tinka žaidimams, kurie vyksta uždarose patalpose. Norint užtikrinti, kad pridedant vis naujas scenas žaidimas veiks be delsos, reikalingas būdas sekti ir naikinti anksčiau užkrautas scenas, o tai šiek tiek komplikuoja šio būdo naudojimą. „DungeonBoy“ bus naudojamas vienetinis scenų užkrovimas. Iš vieno lygio į kitą veikėjas bus perkeliamas žaidėjui aktyvavus portalo objektą arba pasirinkus atitinkamas parinktis iš meniu langų.

* 1. Žaidimo būsenų išsaugojimo ir atkūrimo sistema

Žaidimo būsenos išsaugojimas ir atkūrimas yra viena dažniausiai sutinkamų žaidimų savybių, būdinga pačių skirtingiausių tipų žaidimuose. Ši savybė užtikrina progreso tęstinumą žaidėjui nusprendus baigti žaidimo sesiją ar ištikus veikėjo žūčiai (ar net žaidimo aplikacijai susidūrus su kodo klaida). Be to, žaidimo būsenos išsaugojimas leidžia tą patį žaidimo segmentą sužaisti kitaip, eksperimentuoti žaidimo pasaulyje.

Žaidimo būsenų išsaugojimas žaidimuose gali būti įgyvendintas keliais būdais. Kai kurie žaidimai būsenas išsaugo automatiškai, kas tam tikrą laiko tarpą, arba pasiekus tam tikrą tašką žaidimo lygyje. Dažnai sutinkamas rankinis išsaugojimo būdas, kurio metu žaidėjas atveria meniu langą ir pasirenka išsaugojimo pasirinkimą. Tokie meniu gali būti atidaromi bet kuriuo metu, arba tik tam tikrose žaidimo vietose. Pavyzdžiui, „Resident Evil“ žaidimuose rankinis išsaugojimas leidžiamas žaidėjui radus rašymo mašinėlę, be to, žaidėjas inventoriuje privalo turėti rašalinę juostą, kuri sunaudojama atlikus išsaugojimą, taip apribojant galimų išsaugojimų kiekį.

Žaidimo būsenos išsaugomos jas įrašant į dokumentus, ir atkuriamos juos nuskaitant. Žaidimai, naudojantys rankinį būsenų išsaugojimo būdą dažnai leidžia sukurti keletą žaidimo būsenos dokumentų. Taip sukuriamas žaidimo progreso archyvas, kuris žaidėjui suteikia pasirinkimo laisvę, be to, apsaugo nuo situacijų, kai progresas tampa per sunkus ar neįmanomas.

„DungeonBoy“ tinkamas hibridinis sprendimas. Žaidimo būsena bus automatiškai išsaugoma žaidėjui keliaujant į sekantį lygį. Rankinis išsaugojimas įmanomas žaidėjui sąveikaujant su būsenos išsaugojimo objektais (išsaugojimo taškais). Siekiant apriboti projekto apimtį, žaidimo būsenos išsaugojimas įmanomas tik viename dokumente.

„DungeonBoy“ veikėjo veiksmai gali keisti scenoje esančių daiktų (šaudmenų, eliksyrų, raktų, kepuraičių) skaičių, durų būsenas (atrakinta, atverta), priešų pozicijas. Visa ši informacija turi būti išsaugota. Be to, reikia išsaugoti veikėjo būseną kiekvienoje aplankytoje scenoje: veikėjo poziciją, turimų daiktų inventorių, gyvybių būseną, dėvimą kepuraitę. Taip pat reikia išsaugoti paskutinės aplankytos scenos identifikatorių.

* + 1. Paketų palyginimas

„Unity Asset Store“ galima rasti net keletą paketų, įgyvendinačių būsenų išsaugojimo funkcionalumą. Žemiau aptariami trys populiariausi paketai.

**Moodkie „Easy Save“**

Populiariausias iš trijų aptariamų paketų. Pirma versija sukurta dar 2011 metais ir yra nuolatos atnaujinamas, todėl turi gerą stabilumo garantiją. Naudoja specialiai šiam paketui sukurtą JSON serializatorių, kuris buvo specialiai pritaikytas „Unity“ karkasui ir geba serializuoti įvairius duomenų tipus (ne tik primityvus). Turi duomenų kriptografinės apsaugos ir kompresijos galimybes. Integracijos su debesyje esančia duomenų baze galimybė gali būti naudinga, norint užtikrinti, kad žaidėjai negalėtų patys keisti būsenų dokumento turinio. Ši technika dažnai naudojama tinklo žaidimams, kur norima, kad žaidėjai neturėtų nesąžiningais būdais įgyto pranašumo prieš kitus. Kadangi šis paketas siūlo didelį funkcionalumų pasirinkimą, nenuostabu, kad jis nėra nemokamas. Teksto rašymo metu jo kaina yra 69 USD.

**KROIA-Productions „Flexible save system“**

Sukurtas 2022. Iki šiol turi tik vieną atnaujinimą, kuris buvo paleistais tais pačiais metais, taigi galima manyti, kad nėra aktyviai prižiūrimas. Kaip ir „Easy Save“, leidžia serializuoti sąrašus ir struktūras, ne tik primityvus. Šio paketo kūrėjai išskiria galimybę sukurti ir užkrauti Prafab resursus iš automatiškai redaktoriuje užpildomo sąrašo. Ši galimybė patraukli jei numatoma, kad objektai žaidimo pasaulyje gali atsirasti dinamiškai. Paketas yra nemokamas ir gerai dokumentuotas.

**New Game Studio „Extendable Save System“**

Sukurtas 2019 ir nuo to laiko nebuvo atnaujintas. Šio paketo pagrindinė savybė yra plečiamumas. Su paketu pateikiamas apmokomasis video, iš kurio matyti, kad serializuoti galima tik primityvus. Dokumentacija gan skurdi, bet paketas nemokamas.

Visi aprašyti paketai turi galimybę saugoti duomenis JSON format. Tai svarbi savybė, nes JSON yra žmogui skaitomas formatas, taigi bus lengviau įžvelgti ir taisyti galimas klaidas sistemos pritaikymo metu.

Atsižvelgiant į aukščiau išvardintus argumentus, „DungeonBoy“ labiausiai tiks „Flexible save system“. Gera dokumentacija ir serializuojamų duomenų tipų įvairovė yra būtent tai, ko reikia. Be to, galimybė instancijuoti objektus iš sąrašo gali būti naudinga ateityje. „Easy Save“ mažiau patrauklus, nes dalis siūlomų funkcionalumų nėra aktualūs kuriamam žaidimui ir sąlyginimai aukštos kainos. Akivaizdu, kad šis paketas skirtas komercinio lygio žaidimams. „Extendable Save System“ pasirodė mažiausiai patraukliu pasirinkimu dėl prastos dokumentacijos ir serializuojamų duomenų tipų įvairovės trūkumo.

* 1. Meniu langai

Meniu langas yra grafinė sąsaja, kuri žaidėjui pateikia žaidimo valdymo galimybes, pavyzdžiui pradėti naują ar tęsti anksčiau pradėtą žaidimą, sustabdyti ar baigti žaidimą, keisti žaidimo nustatymus, peržiūrėti žaidimo kūrėjų sąrašą ir panašiai. Visi šie pasirinkimai yra susiję su žaidėjo veiksmais už žaidimo pasaulio ribų, todėl neturi jo įtakoti.

Atsižvelgiant į „DungeonBoy“ savybes, reikalingi trys meniu langai:

* pagrindinis langas, kuriame galima pradėti naują arba tęsti išsaugotą žaidimą, bei išjungti žaidimo aplikaciją
* pauzės langas, kuris trumpam sustabdo žaidimą veiksmą. Šis langas turi leisti tęsti sustabdytą žaidimą, tęsti žaidimą nuo paskutinio išsaugojimo taško arba naviguoti į pagrindinį meniu
* veikėjo žūties langas atsiveria veikėjo gyvybėms pasiekus 0 arba mažiau. Šiame lange turi būti du pasirinkimai: tęsti žaidimą nuo paskutinio išsaugojimo taško, arba naviguoti į pagrindinį meniu

Apibendrinant šiuos poreikius, išskiriami penki pasirinkimai, kuriuos žaidėjui turi pateikti meniu langas:

* pradėti naują žaidimą
* tęsti sustabdytą žaidimą
* tęsti žaidimą nuo paskutinio išsaugojimo taško
* naviguoti į pagrindinį meniu
* išjungti žaidimo aplikaciją

Įsijungus bet kuriam langui, veiksmas žaidimo pasaulyje turi sustoti.

* 1. Durų uždangos

„DungeonBoy“ lygių dizainas yra atviro plano — neturi žaidėjui regimų sienų, išskyrus sienų fragmentus, kuriuose stovi durys. Siekiant padidinti žaidėjo sąveikų su žaidimo pasauliu įvairovę bei paįvairinti žaidimo lygių išvaizdą, erdvė už veikėjo neatvertų durų bus uždengta. Veikėjui atrakinus ir atvėrus duris, ši uždanga išnyks.

* 1. Veikėjo aptikimas

Šiuo metu priešai gali aptikti ir net atakuoti veikėją per sienas ir uždaras duris. Tai sumažina žaidėjo įsitraukimą į žaidimo pasaulį, kai kuriose situacijose priešams suteikia pranašumą. Ši savybė turi būti pašalinta.

1. Metodika
   1. „ Unity“ savybės
      1. SceneManager klasė

Žaidimo metu, „Unity“ gali turėti tik vieną aktyvią sceną. Keičiant vieną sceną kita, pirmoji yra sunaikinama, ir užkraunama antroji — būtent šia seka.

Kaip galima spręsti iš pavadinimo, SceneManager klasė yra skirta darbo su scenomis vykdymui. Naudojantis šios klasės metodais, galima dinamiškai kurti naujas scenas, sulieti kelias scenas į vieną ir kita. Projekto kontekste reikalingi bus scenos užkrovimo (LoadSceneAsync) ir nuorodos į aktyvią sceną gavimo (GetActiveScene) metodai. Be šių metodų taip pat pravartu žinoti apie OnSceneUnloaded (aktyvia buvusi scena buvo sunaikinta) ir OnSceneLoaded (nauja scena užsikrovė) ir įvykius, kurie iššaukiami SceneManager klasės įvykus scenų apsikeitimui. SceneManager yra statinė klasė, todėl jos metodai gali būti pasiekiami iš bet kurios klasės neturint nuorodos į SceneManager objektą.

LoadSceneAsync priima du argumentus: scenos, kuri turi būti užkrauta identifikatorių, ir nebūtiną scenos užkrovimo būdo identifikatorių. Scenos gali būti užkrautos dvejais būdais: Single arba Additive. Single būdas sunaikina aktyvią sceną, užkrauna naują ir ją padaro aktyvia. Additive būdas prijungia naują sceną prie šiuo metu aktyvios. Nepasirinkus užkrovimo būdo, naudojamas Single. Scenos identifikatorius yra būtinas argumentas. Tai gali būti arba scenos pavadinimas, arba scenos indeksas. Scenai vardą galima lasivai priskirti redaktoriuje. Scenos indeksas randamas atidarius Build Settings dialogo langą (File > BuildSettings). Priedas 1 pateikia šio lango ekrano nuotrauką.

* + 1. DontDestroyOnLoad metodas

Keičiantis scenoms, senoji scena sunaikinama, o kartu su ja ir visi joje esantys objektai. Visi scenoje egzistuojantys objektai yra GameObject klasės objektai. Ši klasė paveldi Object klasę, kuri pateikia DontDestroyOnLoad. Šis metodas užtikrina, kad jį iškvietęs objektas nebus sunaikintas scenos sunaikinimo atveju. Su viena sąlyga.

Sceną galima atvaizduoti kaip nejungų medžių grafą. DontDestroyOnLoad metodas taikomas tik šakninių scenos medžių viršūnių objektams; visi objektai, priklausantys medžiui kurio šakninės viršūnės objektas iškvietė DontDestroyOnLoad metodą taip pat bus išsaugoti tarp scenų pasikeitimo. Priedas 2 pateikia scenos grafo iliustraciją.

* + 1. Time klasė

Statinė Time klasė yra naudojama sekti ir kontroliuoti aplikacijos laikui. Time.deltaTime yra plačiai naudojamas šios klasės atributas, kuris saugo laiko tarpą, prabėgusį nuo paskutinio iki dabartinio kadrų. Time.scaleTime atributas naudojamas norint sulėtinti ar pagreitinti veiksmą scenose. Nustačius šio atributo reikšmę į 0, Update ir FixedUpdate įvykių funkcijos nebevykdomos, taigi tai yra geras būdas sustabdyti žaidimo veiksmą.

* + 1. Coroutine

Įprastai, „Unity“ numato, kad funkcijos vykdymas yra nedalomas kadro metu. Coroutine yra ypatingos formos „Unity“ funkcija, kurios veikimas gali tęstis nepertraukiamai kelių kadrų metu. Ši funkcija grąžina IEnumerator, o viduje privalo turėti bent vieną yield return sąlygą. Ši sąlyga nurodo, kad sekantis teiginys nebus vykdomas, kol nebus patenkinta sąlyga. Vienoje Coroutine funkcijoje gali būti daug yiel return sąlygų. Žemiau pateikiamas Coroutine funkcijos kodo pavyzdys, išspausdina pirmą žinutę, laukia 2 sekundes, ir išspausdina antrąją:

IEnumerator CoroutineFunkcija()

{

Debug.Log("Laukiama salygos ivykdymo");

    yield return new WaitForSeconds(2);

    Debug.Log("Salyga ivykdyta");

}

Norint pradėti Coroutine funkcijos veikimą, ji turi būti iškviesta kaip StartCoroutine funkcijos argumentas, t.y. StartCoroutine priima IEnumerator, kurį grąžina Coroutine funkcija:

StartCoroutine(CoroutineFunkcija())

* + 1. Physics.Raycast

„Unity“ fizikos posistemė suteikia galimybę sukurti spindulius, kurie tikrina spindulio kryptyje egzistuoja kolaideriai. Kai spindulys kerta kolaiderį, kolizija nėra fiksuojama, todėl išvengiama OnTrigger/OnCollision funkcijų, kurios iškviečiamos kolizijos atveju, vykdymo. Tai leidžia vykdyti logiką nekeičiant žaidimo būsenos.

Physics.Raycast metodas „iššauna“ spindulį nurodyta kryptimi. Šis metodas taip pat leidžia nurodyti spindulio ilgį, bei nurodyti sluoksnių kaukę, kuri tikrina tik tuos kolaiderius, kurie yra nuždengti kaukės. Jei spindulys kerta kolaiderį, Physics.Raycast grąžina reikšmę true.

* 1. Kitos bibliotekos

Žemiau aprašytos bibliotekos nepriklauso oficialių „Unity“ paketų rinkiniui, todėl turi būti parsisiųstos iš kūrėjų tinklapių ir įdiegtos kaip atskiru atveju pritaikyti paketi. Parsisiuntus šiuos paketus, į „Unity“ projektą jie įdiegiami naviguojant į Assets > Import Packages > Custom Package ir pasirenkant norimą paketą

.

* + 1. DOTween

DOTween yra „Unity“ kūrėjų tarpe populiari biblioteka, skirta interpoliavimo reikmėms. Ji gali būti naudojama objektų pozicijos ar rotacijos, spalvų ir tekstūrų keitimui per tam tikrą laiką.

Ši biblioteka „ apvelka“ interpoliacijai tinkamas klases ir papildo jas interpoliavimo metodais. Pavyzdžiui DOFade metodas keičia Image komponento spalvos alpha kanalo reikšmę. Žemiau pateikiamame pavyzdyje alpha reikšmė per 2,5 sekundės palaipsniui taps lygi 0.

Image myImage;

   myImage.DOFade(0, 2.5f);

DOTween metodai grąžina Tween klasės objektą. Šis objektas gali būti naudojamas norint sekti ar keisti interpoliacijos progresą, pavyzdžiui Tween.IsPlaying grąžina būlio reikšmę, kuri nurodo ar interpoliacija dar vyksta. Tween.Kill sustabdo interpoliaciją esamame taške, Tween.Complete sustabdo interpoliaciją ir jos reikšmę nustato į Tween objektą grąžinusio metodo vykdymui pateiktą galutinės reikšmės argumentą.

* + 1. Newstonsoft.json

Norint išsaugoti žaidimo būseną, į būsenų dokumentą reikia įrašyti žaidimo aplikacijos veikimo metu egzistuojančių objektų būsenas, t.y. objektų atributų reikšmes. Šis procesas vadinamas serializacija. Norint atkurti būsenas, iš būsenų dokumento turimų duomenų sukuriami objektai. Tai yra deserializacija.

Serializuotas kodas gali būti saugomas įvariais formatais, bet šio projekto metu pasirinktas Json formatas. Šis formatas parankus žaidimo kūrimo metu, nes yra lengvai skaitomas žmogui. Tai padeda aptikti klaidas ir įvertinti serializuojamų struktūrų tinkamumą siekiamam tikslui.

„Unity“ turi įrankį JsonUtility, tačiau projekte bus naudojama Newtonsoft.json biblioteka. Ši biblioteka yra populiari C# projektuose ir yra paprasta naudoti.

1. Įgyvendinimas
   1. Keliavimas tarp žaidimo lygių

Keliavimui tarp žaidimo lygių buvo sukurtas ScenePortal skriptas. Jis priskiriamas objektui žaidimo pasaulyje, ir veikėjas sąveikaudamas su šiuo objektu transportuojamas į naują sceną. Prieš keičiantis scenai, išsaugoma scenos ir veikėjo būsenos. Žaidimo būsenų išsaugojimas ir atkūrimas aprašytas poskyryje 3,4. ScenePortal klausosi žaidimo būsenos išsaugojimo patvirtinimo įvykių. Jų skaičiui pasiekus lygiai 2, pradedamas naujos scenos užkrovimas.

bool \_saving = false;

    int \_responseCount = 0;

    void OnEnable()

    {

        \_saveCompleteChannel.OnEventRaise += AcceptResponse;

    }

    void OnDisable()

    {

        \_saveCompleteChannel.OnEventRaise -= AcceptResponse;

    }

public override void Interact(ItemSO item = null)

    {

        if(\_saving)

            return;

        \_saving = true;

        \_saveSceneEvent.RaiseEvent(SceneManager.GetActiveScene().buildIndex);

        StartCoroutine(FinishInteract());

    }

    IEnumerator FinishInteract(){

        yield return \_responseCount == 2;

        GameManager.Instance?.LoadNextScene((int)\_nextScene);

    }

* 1. Meniu langai

Meniu langai sukurti naudojant Canvas komponentą skirtą 2D elementų atvaizdavimui. Pagrindinį meniu lango funkcionalumą atlieką objektas su Image komponentu, skirtu fono atvaizdavimui, mygtukai ir Menu skripte aprašyta klasė. Ši klasė turi metodus, kurie atlieka meniu lango aktyvavimą/deaktyvavimą ir mygtukų fukcionalumą paspaudimo metu. Meniu langui esant aktyviam, veiksmas žaidimo pasaulyje turi būti sustabdytas. Deaktyvavus meniu langą, veiksmą grįžta. Tai įgyvendinta keičiant „Unity“ Time klasės scaleTime atributo reikšmę.

public void Enter(Action callback){

        Time.timeScale = 1;

        StartCoroutine(\_sceneTransition.FadeInScene(\_fadeDuration, () => {

            ActivateUI(true);

            StartCoroutine(\_sceneTransition.FadeOutScene(\_fadeDuration, () => {

                Time.timeScale = 0;

                callback();

            }));

        }));

    }

    public void Leave(Action callback){

        Time.timeScale = 1;

        StartCoroutine(\_sceneTransition.FadeInScene(\_fadeDuration, () => {

            ActivateUI(false);

            StartCoroutine(\_sceneTransition.FadeOutScene(\_fadeDuration, () => {

                callback();

            }));

        }));

    }

Meniu lango aktyvavimas/deaktyvavimas turi išnykimo animaciją, įgyvendintą naudojantis DOTween biblioteka ir Coroutine funkcijomis. Šios animacijos aprašytos SceneTransition skripte, ir taip pat bus naudojamos vykdant scenų pasikeitimą, taip išvengiant staigių vaizdo ekrane pasikeitimų ir suteikiant vientisumo.

  public IEnumerator FadeInScene(float duration, Action callback){

        PrepareTransition();

        Tween temp = FadeIn(duration);

        \_fadeInTween = temp;

        yield return \_fadeInTween.WaitForCompletion();

        callback();

    }

    public IEnumerator FadeOutScene(float duration, Action callback){

        PrepareTransition();

        Tween temp = FadeOut(duration);

        \_fadeOutTween = temp;

        yield return \_fadeOutTween.WaitForCompletion();

        callback();

    }

Visi trys analizėje apražyti meniu langai (pagrindinis, pauzės, veikėjo žūties) specializuoja Menu klasę. Pagrindinė šių specializacijų paskirtis yra klausytis tam tikrų įvykių žaidime, ir aktyvuoti meniu langą. Veikėjo žūties langas klausosi įvykio, kuris iššaukiamas iš žalą skaičiuojančios DamageHandler klasės priklausančios veikėjo objektui, pauzės langas iššaukiamas žaidėjui aktyvavus pauzės įvestį (klavišas „P“). Pagrindinis meniu yra aktyvuojamas žaidimo pradžioje, taip pat gali būti aktyvuotas iš kitų meniu langų.

* 1. Žaidimo būsenų išsaugojimo ir atkūrimo sistema
     1. Būsenų pastovumas tarp scenų

Žaidėjui keliaujant iš vienos scenos į kitą, pirmoji scena, kartu su visais joje esančiais objektais, yra sunaikinama, ir užkraunama antroji. Kadangi veikėjas yra scenos dalis, jis taip pat sunaikinamas, kartu su gyvybių būsena ir inventoriumi. Kitaip tariant, sunaikinamas žaidėjo progesas žaidime.

Be veikėjo, žaidime egzistuoja kitos esybės, kurios kartojasi kiekvienoje scenoje: kamera, grafinės sąsajos (UI) objektas Canvas, kuriame atvaizduojami meniu langai ir informacinės žinutės veikėjui susidūrus su interaktyviais scenų elementais, bei žaidėjo įvesčių valdymo sistema. Šias esybes galima pavadinti žaidimo branduoliu. Žaidimo branduolys neturi reikšmės žaidėjo progresui, tačiau instancijuoti ir susieti jam priklausančius objektus iš naujo kiekvienos scenos užkrovimo metu ne tik reikalautų papildomo kodo (kas kelia kodo klaidų, taigi ir žaidimo stabilumo sumažėjimo, riziką), bet ir turėtų neigiamos įtakos scenos užkrovimo trukmei.

Prasidėjus naujai sesijai, pirmiausia užkraunama inicializacijos scena. Joje egzistuoja tik vienas šakninis scenos objektas su GameManager skriptu. Šiam objektui priklauso žaidimo branduolys. GameManager Awake įvykių funkcija iškviečia DontDestroyOnLoad metodą, kuris užtikrina žaidimo branduolio pastovumą tarp scenų. GameManager Start įvykių funkcija aktyvuoja pagrindinį meniu langą. Žaidėjui pasirinkus tęsti arba pradėti naują žaidimą, sukuriamas veikėjo objektas ir yra priskiriamas objektui su GameManager skriptu. Taip užtikrinamas veikėjo būsenų pastovumas ir išvengiamas žaidimo branduolio sunaikinimo keičiantis scenoms.

* + 1. Būsenų pastovumas tarp žaidimo sesijų

Žaidėjo progreso išsaugojimas ir atkūrimas tarp žaidimo sesijų turi daugiau reikalavimų, nei tarp scenų. Norint atkurti žaidėjo progresą naujoje sesijoje, nepakanka išsaugoti veikėjo turimų daiktų inventorių ir gyvybių būseną. Taip pat reikia išsaugoti paskutinį žaidimo lygį, kuriame buvo išsaugotas žaidėjo progeresas. Dar didesnis iššūkis yra atkurti lygio scenos būseną. Kiekviename lygyje esančius objektus galima išskirti į dvi grupes: statinius ir dinamiškus. Statinei grupei priklauso lygio objektai, kurie nesikeičia žaidimo metu: sienos, grindys ir blokai, ant kurių gali šokinėti žaidėjas. Dinamiškus elementus galima išskirti į tris pogrupius: priešai, daiktai ir durys. Priešai juda savo maršrutais lygyje, be to, turi gyvybių būseną. Daiktai gali būti paimti (ir įdėti į veikėjo inventorių), tokiu atveju, atkūrus lygmenį, jie nebeturėtų jame egzistuoti. Durys gali būti atrakintos, užvertos arba atvertos: šias būsenas taip pat reikia išsaugoti.

Žaidimo būsenos dokumente saugoma ši informacija: paskutinio išsaugoto žaidimo lygio scenos indeksas ir scenų žodynas. Šio žodyno raktai yra scenos indeksai, o reikšmė — objektas, kuriame saugoma veikėjo būsena scenoje (gyvybės, inventorius, pozicija), ir daiktų, priešų ir durų sąrašai su atitinkamų objektų būsenomis. Priedas 3 pateikia šių struktūrų pseudo kodą.

Dinamiškų lygio elementų būsenų išsaugojimui ir atkūrimui sukurta abstrakti SpawnerLoader klasė, kuri numato išsaugojimo (Save) ir atkūrimo (Load) metodus. Ši klasė turi atributą \_sceneList, kuris saugo vienos rūšies, pvz.: priešų, objektus. Vykdant išsaugojimo metodą, šis sąrašas iteruojamas ir sukuriamas naujas, serializacijai tinkamų struktūrų sąrašas. \_sceneList taip pat naudojamas atkuriant objektų būsenas. Scenoje esantys objektai turi unikalius vardus. Ši savybė išnaudojama atkuriant jų būsenas pagal deserializuotą būsenų dokumentą. \_sceneList sąraše esantys objektai, neradę atitikments būsenų struktūroje yra sunaikinami.

SpawnLoader Start įvykių funkcija iškviečia Load metodą, kuris aprašytas atskirai kiekvienoje SpawnLoader specializacijoje, atsižvelgiant į jai priklausančių dinamiškų lygio objektų savybes. Tai reiškia, kad kiekvienos rūšies dinamiški lygio elementai atkuria savo objektų būsenas nepriklausomai vieni nuo kitų. Start įvykių funkcija užtikrina, kad visi scenoje esantys objektai buvo užkrauti, todėl būsenų struktūroje neišsaugotus objektus galima saugiai sunaikinti.

Žaidimo lygiuose egzistuoja šakninio lygio objektas su GameSceneLoader skriptu. Šis objektas agreguoja visus objektus su SpawnerLoader specializacijomis ir laukia išsaugojimo įvykio, kuris gali būti iškviestas žaidėjui sąveikaujant su išsaugojimo taško arba scenos portalo objektais. Sulaukus šio įvykio, GameSceneLoader iteruoja per jam priklausančius objektus su SpawnerLoader skriptu ir iškviečia jų Save metodą (Priedas 4 pateikia šių objektų ekrano nuotraukas). Išsaugojimo įvykį iššaukiantys objektai klausosi būsenos išsaugojimo patvirtinimo įvykių, kuriuos iššaukia PlayerLoader ir GameSceneLoader objektai, kai išsaugojimo metodas pasiekia pabaigą. Jei tokių įvykių buvo užfiksuota lygiai du, tai reiškia, kad scenos ir veikėjo būsenos buvo sėkmingai išsaugotos. Scenos portalo atveju, sulaukus dviejų patvirtinimų, iššaukiamas naujos scenos užkrovimas.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| Iliustracija 2: GameSceneLoader objektų hierarchija su SpawnerLoader ir dinamiškais scenos objektais |

Veikėjo būsenos išsaugojimas ir atkūrimas remiasi tuo pačiu, įvykių architektūra grįstu, principu kaip ir SpawnerLoader. Inventoriaus atkūrimas yra išskirtinis atvejis. Jam atkurti reikia iš naujo sukurti objektus. Tuo tikslu PlayerLoader turi nuorodą į ScriptableObject, kuriame laikomas sąrašas visų žaidime egzistuojančių daiktų, kuriuos gali turėti inventorius. Inventoriaus būsenoje saugomų daiktų vardai atitinka daiktų vardus šiame sąraše.

Žaidimo būsenos serializavimas/deserializavimas yra įgyvendintas naudojantis metodikoje aprašyta Newtonstoft.json biblioteka. Projekte sukurtos specialios struktūros, kurios apibūdina žaidėjo ir dinamiškų lygio elementų būsenas. Šios struktūros įrašomos ir nuskaitomos iš dokumento naudojantis C# File sąsaja. „Unity“ numatyto standartinę duomenų pastovumo dokumentams skirtą lokaciją žaidėjo diske. Kelias į šia lokaciją prieinamas per statinį Application.persistentDataPath atributą. Loader skriptas aprašo metodus reikalingus struktūru serializacijai/deserializacijai ir įrašymui į/skaitymui iš būsenų dokumento. Objektai, kuriems reikalingas būsenų pastovumas komponuoja Loader klasės objektą.

using Newtonsoft.Json;

using System.IO;

public class Loader

{

    public string ConvertDataToJson<T>(T data){

        JsonSerializerSettings settings = new JsonSerializerSettings

        {

            ReferenceLoopHandling = ReferenceLoopHandling.Ignore

        };

        return JsonConvert.SerializeObject(data, settings);

    }

    public SaveData LoadData(string saveFilePath){

        if(File.Exists(saveFilePath)){

            string saveFileText = File.ReadAllText(saveFilePath);

            SaveData parsedSaveData = JsonConvert.DeserializeObject<SaveData>(saveFileText);

            return parsedSaveData;

        }

        return new SaveData();

    }

    public SaveData LoadData(SceneConfig sceneConfig){

        string saveFilePath = GetSaveFilePath(sceneConfig.saveFileName);

        return LoadData(saveFilePath);

    }

    public void WriteToFile(SaveData data, string saveFilePath){

        string jsonString = ConvertDataToJson(data);

        File.WriteAllText(saveFilePath, jsonString);

    }

    public string GetSaveFilePath(string saveFileName){

        return Application.persistentDataPath + "/" + saveFileName;

    }

}

Žaidėjui pradėjus naują sesiją, GameManager Start įvykių funkcijos vykdymo metu aktyvuojamas pagrindinis meniu. Žaidėjui pasirinkus tęsti žaidimą, GameManager komponuojamas Loader objektas nuskaito būsenų dokumentą ir deserializuoja šiame dokumente įrašytą struktūrą. Ši struktūra saugoma ir žaidimo metu atnaujinama statiniame GameState attribute. Užkraunama scena, kurioje paskutinį kartą buvo išsaugota žaidimo būsena. SpawnerLoader objektai atkuria savo būsenas naudodamiesi nuoroda į GameManager.GameState. GameManager atkuria veikėjo būseną.

* 1. Durų uždangos

Durų uždanga yra sudaryta iš keturių, kubo šonines sienas sudarančių, plokštumų. Durų sistema turi nuorodą į uždangą ir uždangos medžiagos matomumas yra valdomas durų būsenos. Jei durys dar nebuvo atvertos, jų atvėrimas aktyvuoja išnykimo animaciją, kuri buvo įgyvendinta pasinaudojus DOTween biblioteka. DOFade metodas interpoliuoja kubo tekstūros alpha reikšmę nuo 1 iki 0 per nurodytą laiką.

Uždangai sukurtas PlaneBoxController skriptas, kuris pateikia viešai prieinamą funkciją, keičiančią medžiagos matomumą.

public void Fade(float duration = 1.0f){

        foreach (var wall in \_walls)

        {

            wall.gameObject.GetComponent<Renderer>().material.DOFade(0.0f, duration);

        }

    }

Priedas 5 pateikia žaidimo vaizdus, kurie iliustruoja šios uždangos sukuriamą efektą.

* 1. Veikėjo aptikimas

Priešai žaidėją aptinka naudojant sferos formos kolaiderį. Veikėjui kirtus šio kolaiderio ribas, priešo būsena pasikeičia iš pasyvios į agresyvią. Priklausomai nuo distancijos tarp žaidėjo ir priešo, agresyvi būsena toliau skiriasi į vijimosi arba atakos būsenas.

Pirmojoje „DungeonBoy“ versijoje, priešai galėjo atakuoti veikėją per sienas. Užkirsti tam kelią atlikti du pakeitimai:

1. Priešų būsenos valdymui skirtas EnemyStateManager skriptas papildytas IsPlayerBehindWall metodu:

public bool IsPlayerBehindWall(Transform player){

        LayerMask mask = LayerMask.GetMask("Walls");

        Vector3 origin = \_projectileLauncher.transform.position - (Vector3.forward \* 0.4f);

        Vector3 target = player.position + Vector3.up;

        float distance = Vector3.Distance(origin, player.position);

        Ray ray = new Ray(origin, target - origin);

        if (Physics.Raycast(ray, distance, mask))

        {

            return true;

        }

        return false;

    }

Prieš priešui pradedant atakuoti veikėją, sukuriamas „Unity“ fizikos posistemės spindulys tarp priešo ir veikėjo. Šis spindulys kerta tik sienų sluoksnyje esančius kolaiderius. Tai užtikrinama pasinaudojus „Unity“ sluoksnių kauke (LayerMask klasės objektu). Jei spindulys kerta sienos kolaiderį, tai reiškia, kad veikėjas yra už sienos ir priešo būsena pasikeičia į vijimosi būseną. Ši būsena turi savo laikmatį. Jei veikėjas neišlenda iš už sienos, ši būsena pareina į pasyvią, ir priešas grįžta į savo patruliavimo maršrutą.

1. Šaudmenys susinaikina, jiems aptikus koliziją su sienų sluoksnyje esančiu objektu.
2. Rezultatai ir išvados

Rezultatai:

* ScenePortal skriptu įgyvendintas būdas veikėjui keliauti iš vieno žaidimo lygio į kitą
* Sukurta žaidimo būsenų išsaugojimo ir atkūrimo sistemą
* Sukurti meniu langai žaidimo valdymui
* Sukurta uždanga, slepianti žaidimo erdvę kitapus užvertų durų
* Priešai nebegali aptikti ir atakuoti veikėjo, jei jis už sienos
* Sukurtas naujas žaidimo lygis, naudojantis vien „Unity“ redaktoriumi. Į šį lygį galima pateikt žaidėjui aktyvavus scenos portalą pirmo lygio pabaigoje

Išvados:

* Projekto metu sukurti keliavimo tarp scenų, būsenų išsaugojimo, ir meniu langų sprendimai suteikia pakankamą pagrindą plėsti „DungeonBoy“ naudojantis vien tik „Unity“ redaktoriumi
* Durų uždangos ir pataisytas veikėjo aptikimas pagerina žaidėjo patirtį

## Priedai

|  |
| --- |
| **Priedas 1** |
| A screenshot of a computer  Description automatically generated |
| Scenų indeksai Build Settings dialogo lange (pažymėta geltonai) |

|  |
| --- |
| **Priedas 2** |
| A diagram of a company  Description automatically generated |
| Scenos grafo pavyzdys. Šakninės scenos medžių viršūnės pažymėtos žaliai |

|  |  |
| --- | --- |
| **Priedas 3** | |
| {      "metaData": {          "lastSavedSceneIndex": int      },      "scenes": {          "2": {              "player": {},              "enemies": {                  "list": []              },              "items": {                  "list": []              },              "doors": {                  "list": []              },          }      }  } | "player": {      "body": {          "position": Vector3,          "rotation": Quaternion      },      "hat": string,      "health": {          "currentHealth": float,          "maxHealth": float      },      "inventory": [          {              "name": string,              "count": int          }      ]  } |
| Būsenų struktūra | Veikėjo būsenos struktūra |
| "enemies": {      "list": [          {              "health": {                  "currentHealth": float,                  "maxHealth": float              },              "body": {                  "position": Vector3,                  "rotation": Quaternion              },              "waypoints": [Vector3],              "name": string,              "prefabName": string          }      ]  } | "items": {      "list": [          {              "body": {                  "position": Vector3,                  "rotation": Quaternion              },              "name": string,              "prefabName": string          }      ]  } |
| Priešų būsenų sąrašas su vienu įrašu | daiktų būsenų sąrašas su vienu įrašu |

|  |  |
| --- | --- |
| "doors": {      "list": [          {              "isLocked": bool,              "isOpen": bool,              "isJammed": bool,              "isFogEnabled": bool,              "name": string,              "prefabName": string          }      ]  } |  |
| durų būsenų sąrašas su vienu įrašu |  |

|  |  |
| --- | --- |
| **Priedas 4** | |
| A video game screen with a cartoon character  Description automatically generated | A video game screen with a cartoon character on a checkered floor  Description automatically generated |
| Išsaugojimo taškas | Scenos portalas |

|  |  |
| --- | --- |
| **Priedas 5** | |
| A video game screen of a video game  Description automatically generated | A video game of a cartoon character  Description automatically generated |
| Durų uždanga prieš atveriant duris | Durų uždanga atvėrus duris |