Izrada videoigre žanra JRPG temeljene na upravljanju s više likova

Babović, Tomislav

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:102431

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Tomislav Babović

IZRADA VIDEOIGRE ŽANRA JRPG TEMELJENE NA UPRAVLJANJU S VIŠE LIKOVA

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Babović

Matični broj: 0036480922

Studij: Informacijsko i programsko inženjerstvo

IZRADA VIDEOIGRE ŽANRA JRPG TEMELJENE NA UPRAVLJANJU S VIŠE LIKOVA

DIPLOMSKI RAD

Mentor/Mentorica :

Izv. prof. dr. sc. Mario Konecki

Varaždin, Travanj 2024.

Tomislav Babović

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Sažetak

U ovom radu se opisuje razvoj „japanske igre uloga(JRPG)“, temeljene na upravljanu s više likova, video igre „Shadow Tomb Valley“ u Unity razvojnom alatu. Koncept igre podsjeća na starije JRPG-ove za gameboy ili Sony playstation. Igra se sastoji od „heroja“ s kojim igrač upravlja te pokušava doći do „ShadowTomba“. U igri postoje sigurne zone i zone u kojima su neprijatelji. Borba je napravljena tako da svatko ima svoj potez tokom kojeg može izvoditi akciju. Prilikom pobjede u borbi igrača se nagrađuje zlatom i iskustvom te u rijetkim slučajevima i predmetom. Igrač sakupljanjem iskustva povećava svoj nivo. Na taj način otključava jednu od dvije klase te vještine. U jednoj od sigurnih zona se nalaze prodavač i gostioničar kod kojih je moguće kupiti, prodati predmete te prespavati noć te tako u potpunosti obnoviti zdravlje i manu.

Ključne riječi: Unity, JRPG, više likova, korutina, event, borba, dinamičnost

Sadržaj

1. Uvod

Igre su pratile čovječanstvo od pradavnih vremena kada su se predmeti za igru izrađivali od kostiju. S razvojem tehnologije, došlo je i do evolucije igara, sve do pojave video igara. Prve video igre su se pojavile 1950-ih, ali zbog visokih troškova računalne tehnologije, postale su popularne tek u 1970-ima. Jedna od prvih igara koje su zarazile svijet bila je "Pong", koja je nastala 1972. godine pod okriljem Atarija. Ova igra je donijela veliku popularnost, potičući razvoj novih igara.[1]

Danas, preko 3 milijarde ljudi uživa u svijetu video igara[2], a njihova popularnost je dovela do procvata i razvoja gaming turnira. Nagradni fondovi za takve turnire mogu doseći nevjerojatnih 40 milijuna dolara[3].

Video igre dolaze u mnoge žanrove, a za razliku od filmova i knjiga, žanr video igre ne ovisi nužno o svijetu u kojem se odvija radnja. Neki od popularnih žanrova uključuju akcijske igre, pucačke igre, sportske igre i RPG-ove (Role-Playing Games). JRPG (Japanese Role-Playing Games) je podžanr igara uloga koji je poznat po borbi u potezima(eng. turn-based) stilu[4]. Među najprodavanijim naslovima u ovom podžanru su "Pokemon", koji je prodan u više od 300 milijuna primjeraka. Tu su i drugi naslovi poput "Final Fantasy" i "Dragon Quest", koji su nastali tijekom "zlatnog doba" igara uloga od 1990-ih do sredine 2000-ih.

Jedan od razloga za popularnost JRPG-a je naglasak na priči, što ih čini privlačnima za igrače. Međutim, tijekom tog zlatnog doba, JRPG-i su bili manje pristupačni za zapadno tržište zbog visokih troškova prevođenja japanskog teksta na engleski[5].

U ovom radu cilj je osmisliti JRPG igru pod nazivom "Shadow Tomb Valley" i prikazati njen razvoj kroz alat Unity i programski jezik C#. Igra je namijenjena za jednog igrača i igra se isključivo putem tipkovnice. Sadrži osnovne elemente JRPG igara poput borbe temeljene na potezima, vještina, više igrivih likova, različitih klasa, raznih vrsta predmeta, slučajnih borbi te borbi sa šef(eng. boss) neprijateljima.

2. Unity razvojno okruženje

Unity je softver za razvoj 2D i 3D video igara, virtualne stvarnosti i simulacija, koji radi na različitim platformama. Osim u video igračkoj industriji, koristi se i u drugim sektorima poput filmske i arhitektonske industrije. Razvila ga je kompanija Unity Technologies, a prvi put je predstavljen 2005. godine na Worldwide Developers konferenciji. Početno je bio dostupan samo za OS X platformu, no danas podržava preko 25 različitih platformi[6].

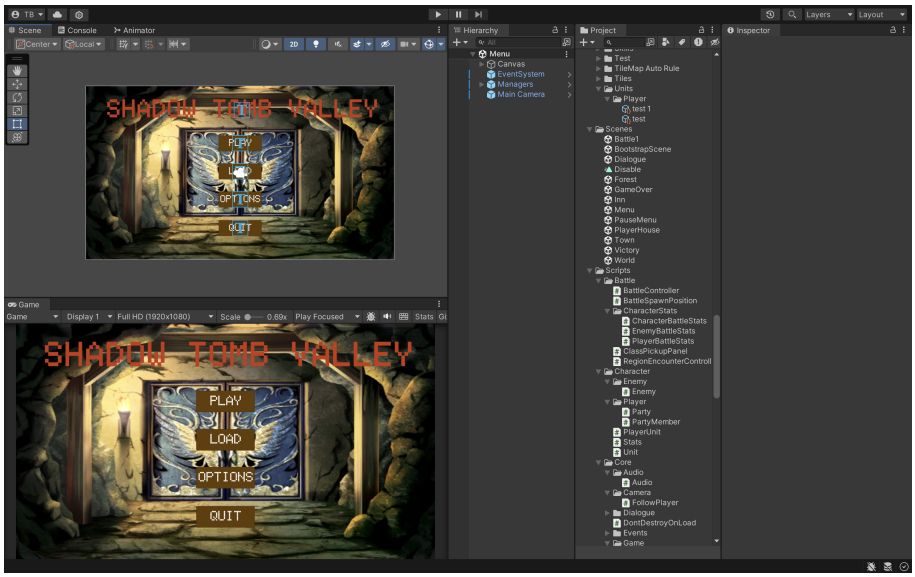
Unity podržava C# programski jezik, koji je korišten za pisanje skripti u razvoju video igre "Shadow Tomb Valley" o kojoj je riječ u ovom radu. Osim C#, Unity podržava i JavaScript i Python. Korisnici mogu besplatno koristiti Unity, no također ima svoj "Assets store" gdje se mogu kupiti razni dodaci i resursi za razvoj igre. U ovom projektu, korištena verzija Unityja je bila 2022.3.5f1.

2.1. Unity sučelje

Prilikom pokretanja prikazuje se sučelje koje se može prilagođavati. Primjer sučelja se nalazi na slici 1.

Neki od prozora na sučelju su:

- Hierarchy
- Scene
- Game
- Inspector
- Project
- Console



Slika 1: Početno sučelje (autorski rad)

Početno sučelje se može razlikovati od verzije do verzije te se može prilagoditi u izborniku prozor(eng. window). Osim toga postoji nekoliko rasporeda(eng. layouta), te je moguće napraviti vlastiti. Na taj način se može prilagoditi izgled sučelja potrebama korisnika.

2.1.1. Hijerarhija

Hijerarhija (eng. Hierarchy) je sučelje unutar Unityja koje prikazuje strukturu i organizaciju svih objekata u trenutnoj sceni. To je popis svih objekata koji se nalaze u vašoj sceni, prikazan kao stablo hijerarhije, gdje svaki objekt može imati roditeljske i dječje objekte[7].

Glavne komponente Hierarchy prozora uključuju:

1. Objekti u sceni: Svaki objekt u sceni, kao što su igrač, neprijatelji, dekorativni elementi ili kamere, prikazani su u Hierarchyju kao stavke u stablu. Objekti se mogu grupirati i organizirati prema potrebi.

2. Roditeljski i dječji objekti: Objekti mogu imati roditelje i djecu, što omogućuje hijerarhijsko organiziranje. Kada objekt služi kao roditelj, svi njegovi dječji objekti nasljeđuju transformacije (poziciju, rotaciju, veličinu) od roditeljskog objekta.

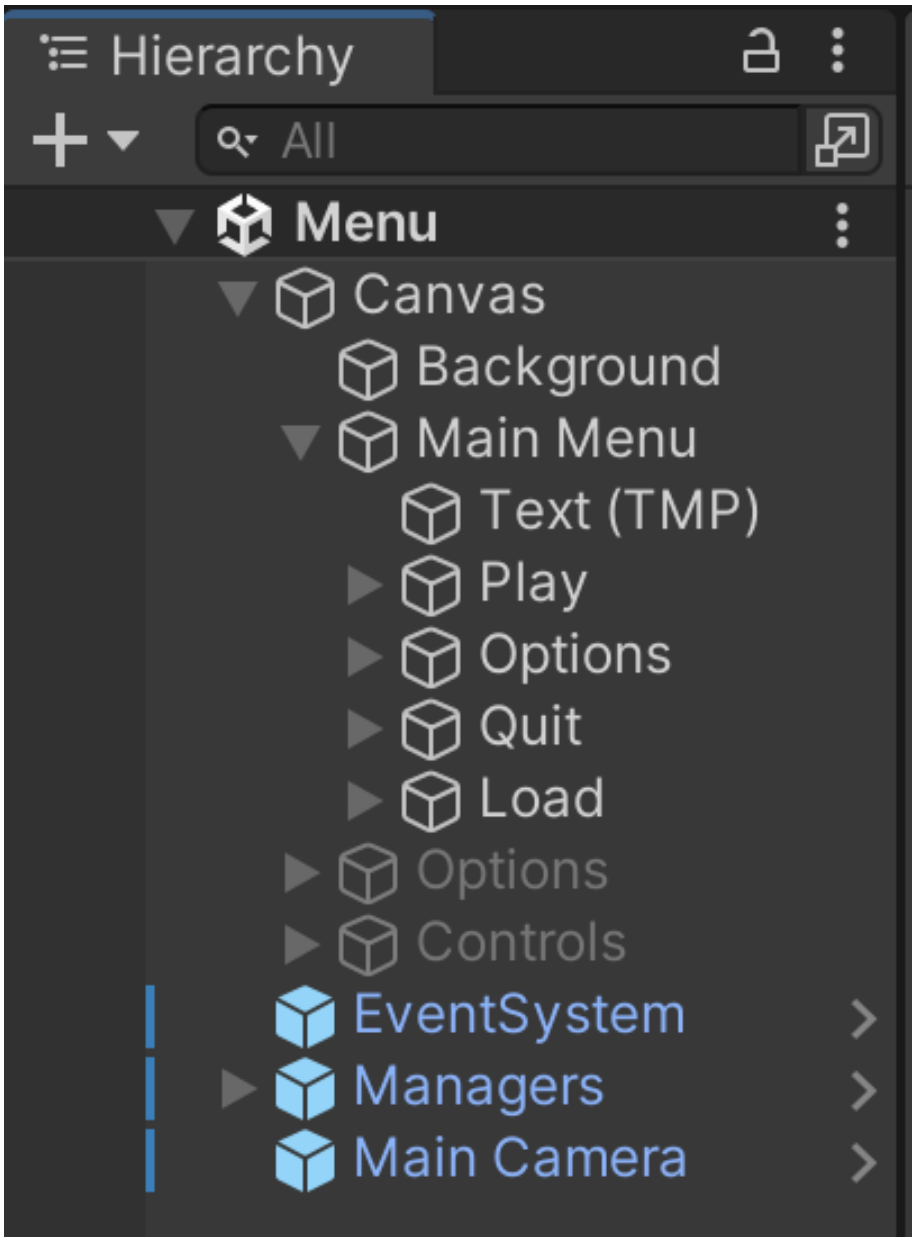
3. Ikonice i oznake: Hierarchy prikazuje ikonice i oznake koje pomažu u identifikaciji vrste objekata ili njihovog stanja. Na primjer, kamere se označavaju ikonicom kamere, a objekti se mogu označiti različitim bojama ili oznakama radi lakše identifikacije.

4. Redoslijed slojeva: Objekti su organizirani u Hierarchyju prema njihovom redoslijedu slojeva u sceni. Objekti na višim slojevima renderiraju se preko objekata na nižim slojevima, što utječe na njihov prikaz u igri.

5. Manipulacija objektima: Iz Hierarchy prozora možete jednostavno manipulirati objektima u sceni, kao što su pomicanje, rotiranje ili mijenjanje njihovih svojstava. Klikom na odre-

đeni objekt u Hierarchyju omogućuje vam se pristup njegovim svojstvima u Inspector prozoru.

Hijerarhija je ključni dio Unityjevog sučelja jer omogućuje organizaciju i upravljanje svim objektima u sceni. Pravilno korištenje Hierarchy prozora olakšava strukturiranje scene i omogućuje učinkovit rad prilikom razvoja video igara u Unityju.



Slika 2: Hijerarhija (autorski rad)

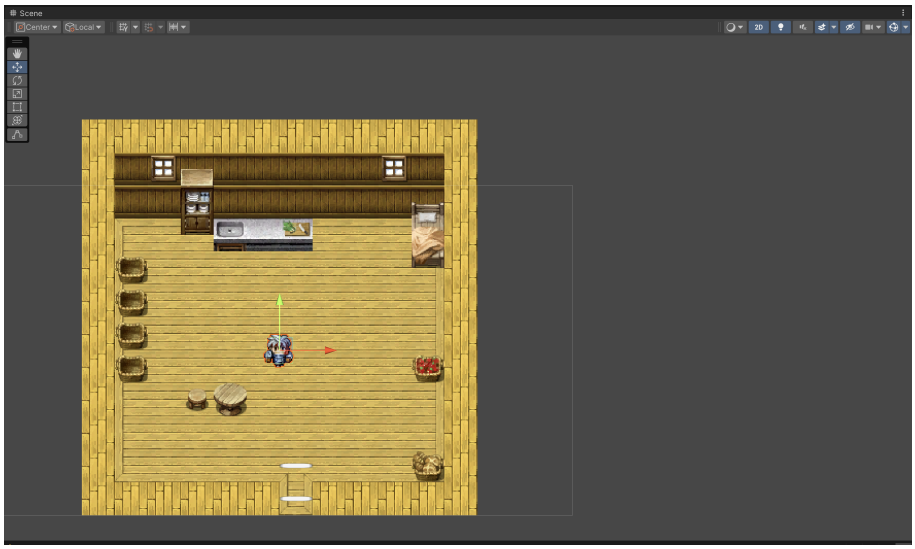
2.1.2. Scene

Scena (eng. Scene) u Unityju predstavlja prostor u kojem se odvija igra ili aplikacija. To je okruženje u kojem se postavljaju i organiziraju objekti, svjetla, kamere i ostali elementi koji čine interaktivno iskustvo korisnika[8].

Glavne karakteristike Scene u Unityju uključuju:

1. Prikaz prostora: Scene pružaju vizualni prikaz prostora u kojem će se igra odvijati. Možete postaviti pozadinu, krajolik, zgrade i druge elemente koji će definirati okolinu igre.
2. Postavljanje objekata: U sceni možete postaviti sve objekte koji će sudjelovati u igri, kao što su likovi, neprijatelji, prepreke, resursi i drugi elementi.
3. Organizacija elemenata: Objekti u sceni mogu se organizirati u hijerarhijsku strukturu, što olakšava njihovo upravljanje i manipulaciju. Roditeljski i dječji odnosi omogućuju grupiranje srodnih objekata.
4. Postavljanje svjetla i kamere: Scene omogućuju postavljanje svjetla i kamera kako bi se definirao osvjetljenje i perspektiva igre. Osvjetljenje i kamera ključni su za stvaranje atmosfere i doživljaja igre.
5. Pregled kroz Scene prozor: Kroz Scene prozor možete pregledavati i uređivati trenutno odabranu scenu. Možete pomicati, rotirati i zumirati prikaz scene radi preciznog pozicioniranja objekata.
6. Prebacivanje između scena: U Unityju možete imati više scena koje se mogu međusobno povezati. Prebacivanje između scena omogućuje prikaz različitih dijelova igre ili prijelaz između različitih stanja aplikacije.

Scene su temeljna strukturna jedinica u Unityju i ključne su za stvaranje interaktivnih iskustava. Pravilno upravljanje i organizacija scena ključni su za uspješan razvoj igara i aplikacija u Unity okruženju.



Slika 3: Scena (autorski rad)

2.1.3. Igra

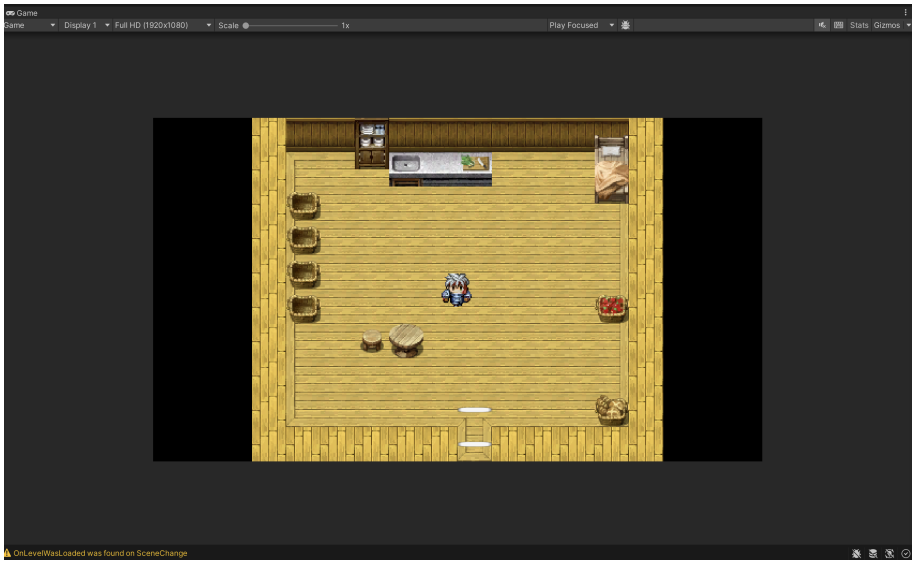
Prozor igra (eng. Game) u Unityju predstavlja prikaz igre u stvarnom vremenu tijekom izvođenja ili testiranja. To je prozor koji omogućuje programerima i dizajnerima da vide kako će

njihova igra izgledati i funkcionirati kada se pokrene[9].

Evo nekoliko ključnih karakteristika "Game" prozora:

1. Prikaz igre u stvarnom vremenu: "Game" prozor prikazuje trenutno stanje igre dok se izvodi u Unityju. To uključuje sve vizualne i interaktivne elemente igre kao što su likovi, okoliš, efekti, interakcije i ostalo.
2. Debugiranje igre: Programeri koriste "Game" prozor za praćenje i provjeru rada igre u stvarnom vremenu. Mogu pratiti kretanje likova, provjeravati kolizije, testirati funkcionalnosti i otklanjati greške.
3. Podešavanje kamere: Kroz "Game" prozor možete postaviti kamere kako biste vidjeli igru iz različitih perspektiva. To vam omogućuje da provjerite kako izgleda igra iz različitih kuteva i poboljšate korisničko iskustvo.
4. Interaktivnost: Dok se igra izvodi u "Game" prozoru, možete interaktivno sudjelovati u testiranju igre. To uključuje kretanje likova, izvođenje akcija, testiranje kontrola i provjeru funkcionalnosti.
5. Podešavanje performansi: "Game" prozor omogućuje vam praćenje performansi igre kako biste osigurali da radi glatko i bez zastoja. Možete pratiti FPS (frames per second), provjeriti opterećenje resursa i identificirati moguće probleme s performansama.

"Game" prozor je važan alat u razvoju igara u Unityju jer pruža uvid u to kako će igra izgledati i funkcionirati iz perspektive krajnjeg korisnika. Omogućuje programerima da testiraju i poboljšaju igru dok je još u razvoju, što rezultira kvalitetnijim krajnjim proizvodom.



Slika 4: Sučelje igre (autorski rad)

2.1.4. Inspektor

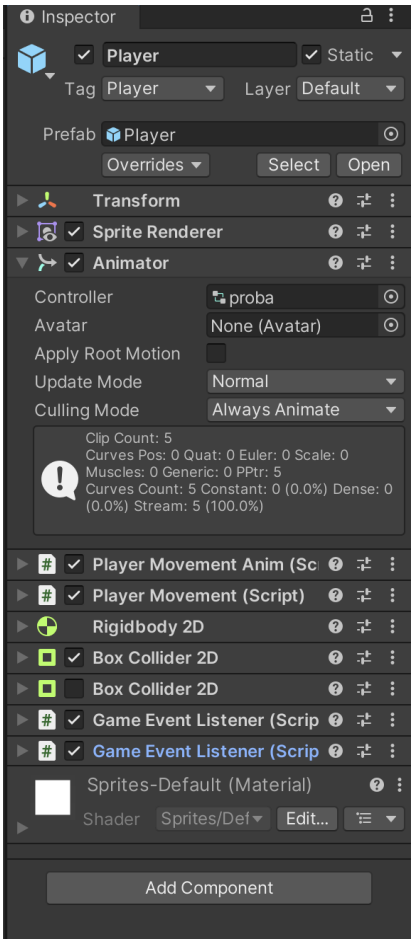
Inspektor (hrv. Inspector) je sučelje unutar Unityja koje omogućuje detaljan pregled i uređivanje svojstava i komponenti odabranog objekta ili resursa u projektu. Kada odaberete

određeni objekt u sceni ili resurs u prozoru projekta, njegove karakteristike, komponente i mogućnosti postaju vidljive u inspektoru[10].

Glavne komponente inspektora uključuju:

1. Svojstva objekta: Prikazuje sve osnovne podatke o odabranom objektu, kao što su pozicija, rotacija, veličina, ime, slojevi i tagovi. Možete mijenjati ove vrijednosti izravno u inspektoru.
2. Komponente: Ako je odabrani objekt opremljen komponentama, inspektor prikazuje sve te komponente i njihova svojstva. Na primjer, ako je objekt igrača, možete vidjeti komponente kao što su Collider, Rigidbody, skripte itd., s opcijama za uređivanje njihovih parametara.
3. Skripte: Ako je objekt povezan s određenim skriptama, inspektor prikazuje te skripte i njihove metode i varijable. Omogućuje vam brz pristup i uređivanje koda direktno iz Unityja.
4. Prefab opcije: Ako je odabrani objekt instanca prefaba, inspektor prikazuje dodatne opcije povezane s prefabom, kao što su veze s izvornim prefabom ili opcije za primjenu promjena na sve instance prefaba.
5. Opcije za kolaboraciju: inspektor također može sadržavati opcije za upravljanje verzijama i kolaboracijom ako koristite Unityjeve alate za verzioniranje i suradnju.

Inspektor je ključni alat u razvoju igara u Unityju jer omogućuje detaljno prilagođavanje i uređivanje svih aspekata vaših objekata i resursa. Pravilno korištenje inspektora omogućuje vam precizno podešavanje svojstava vaših objekata i komponenti te olakšava iterativni proces razvoja.



Slika 5: Inspektor (autorski rad)

2.1.5. Projekt

Prozor projekta (eng. project window) u Unityju je sučelje koje prikazuje sve datoteke i mape koje čine trenutni projekt. To je poput radnog prostora gdje možete organizirati, pregledavati i manipulirati resursima vaše igre. U ovom prozoru možete pronaći sve slike, zvukove, skripte, prefabe i druge resurse koji se koriste u vašoj igri[11].

Glavne komponente Project windowa uključuju:

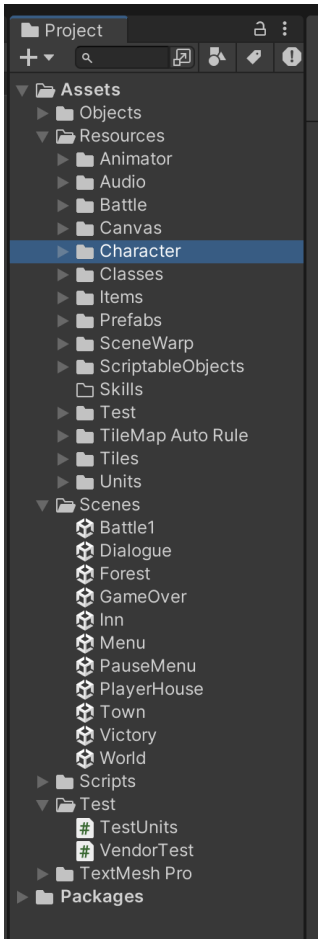
1. Direktoriji i datoteke: Project window prikazuje strukturu direktorija i sve datoteke koje su dio vašeg projekta. Možete organizirati svoje datoteke u direktorije kako biste ih lakše pronašli i upravljali njima.

2. Filter i pretraživanje: Omogućuje vam brzo pronalaženje određenih datoteka ili resursa u projektu pomoću funkcije pretraživanja. To vam štedi vrijeme jer ne morate ručno pregledavati cijelu strukturu projekta.

3. Kategorije i oznake: Datoteke se često kategoriziraju ili označavaju kako bi se olakšalo njihovo upravljanje. Možete dodavati oznake datotekama kako biste ih grupirali ili označili za određene svrhe.

4. Prikaz resursa: Kada kliknete na određenu datoteku ili mapu, Project window prikazuje detalje o tom resursu, uključujući i njezine metapodatke kao što su veličina, tip datoteke, zadnje uređivanje i slično.

Project window je važan alat za organizaciju vašeg projekta i olakšava vam navigaciju i upravljanje resursima. Pravilno korištenje ovog prozora može poboljšati učinkovitost vašeg razvojnog procesa i olakšati suradnju unutar tima.



Slika 6: Projekt (autorski rad)

2.1.6. Konzola

Prozor konzole (eng. Console) u Unityju je koristan alat koji služi za prikazivanje različitih poruka, upozorenja i grešaka tijekom izvođenja igre ili rada u Unity okruženju. Ova konzola pruža važne informacije programerima i dizajnerima o stanju njihove igre i mogućim problemima koji se mogu pojaviti[12].

Evo nekoliko ključnih značajki konzole:

1. Prikaz poruka: konzola prikazuje različite vrste poruka, uključujući informacije, upozorenja i greške. Ove poruke pružaju uvid u različite aspekte izvođenja igre i mogu pomoći u pronalaženju i rješavanju problema.

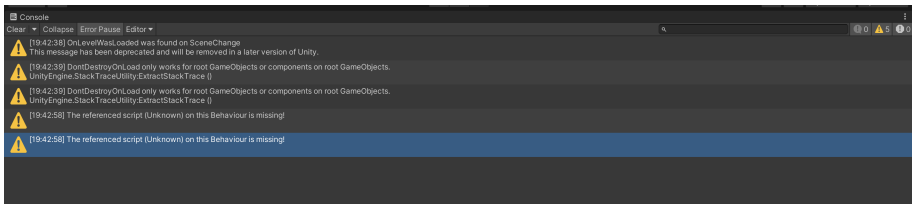
2. Debugiranje grešaka: Kada se pojave greške u kodu ili tijekom izvođenja igre, konzola ih prikazuje korisnicima. Ovo omogućuje programerima da identificiraju i rješavaju greške kako bi osigurali ispravno funkcioniranje igre.

3. Praćenje upozorenja: konzola također prikazuje upozorenja koja ukazuju na potencijalne probleme ili nedostatke u kodu. Ova upozorenja omogućuju programerima da prepoznaju i isprave potencijalne probleme prije nego što postanu ozbiljniji.

4. Prikazivanje informacija: Osim grešaka i upozorenja, konzola prikazuje i informativne poruke koje pružaju korisne informacije o izvođenju igre ili procesu razvoja.

5. Filtriranje poruka: konzola omogućuje filtriranje poruka prema vrsti (informacije, upozorenja, greške) kako bi se olakšalo praćenje i rješavanje određenih problema.

Kroz konzolu, programeri mogu pratiti stanje svoje igre, identificirati probleme i greške te ih riješiti kako bi osigurali stabilno i ispravno funkcioniranje konačne verzije igre.



Slika 7: Konzola (autorski rad)

2.2. MonoBehaviour

MonoBehaviour je ključna klasa u Unityju koja se koristi za stvaranje skripti koje upravljaju ponašanjem objekata u igri. Evo nekoliko bitnih informacija o MonoBehaviour klasi:

1. Osnovna klasa: MonoBehaviour je bazna klasa iz koje proizlaze sve ostale skripte u Unityju. To znači da sve skripte koje se koriste kao komponente objekata na sceni u igri moraju biti izvedene iz MonoBehaviour klase.

2. Funkcionalnosti: MonoBehaviour pruža različite metode koje se automatski pozivaju tijekom izvođenja igre. Ove metode omogućuju programerima da upravljaju različitim aspektima ponašanja objekata, kao što su inicijalizacija, ažuriranje, detekcija sudara i interakcija s korisnikom.

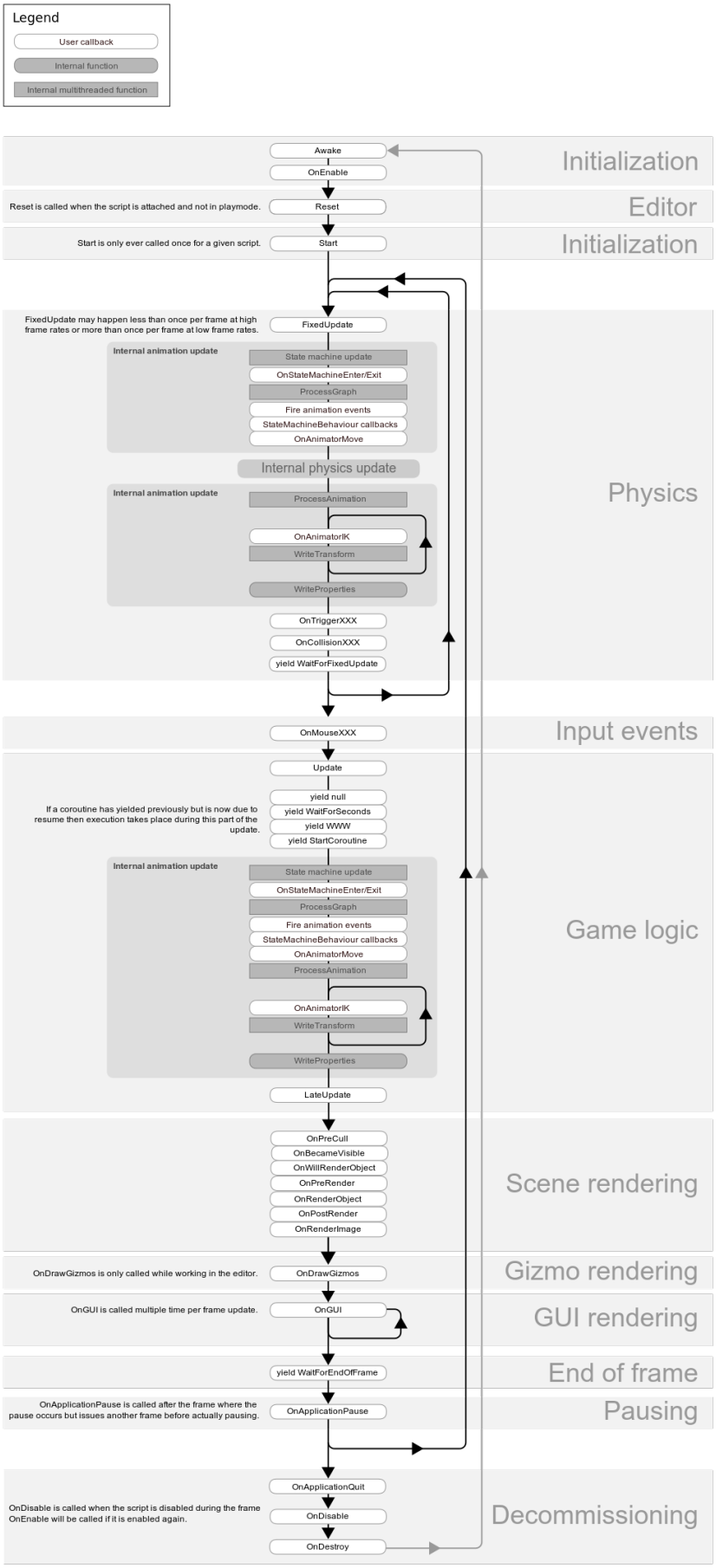
3. Automatsko izvršavanje: MonoBehaviour metode se automatski pozivaju u određenim trenucima tijekom životnog ciklusa objekta na koji su povezane. Na primjer, metoda "Start" se poziva jednom kada se objekt prvi put aktivira, dok se metoda "Update" poziva svaki frejm tijekom izvođenja igre.

4. Skripte za komponente: Skripte koje se dodaju kao komponente objektima u Unityju moraju biti izvedene iz MonoBehaviour klase. To omogućuje programerima da prilagode

ponašanje objekata pomoću skripti koje se lako povezuju i konfiguriraju putem Unity sučelja.

5. Fleksibilnost: Zbog svoje fleksibilnosti i moći, MonoBehaviour omogućuje programerima da implementiraju različite funkcionalnosti i interakcije u svojoj igri. To uključuje kretanje objekata, detekciju sudara, upravljanje animacijama, interakciju s korisnikom i još mnogo toga.

Ukratko, MonoBehaviour je osnovna klasa u Unityju koja omogućuje programerima da stvaraju dinamično ponašanje objekata u svojim igrama pomoću skripti[13].



Slika 8: MonoBehaviour (Izvor: Unity)

Neke od važnijih metoda u klasi MonoBehaviour su "Start", "Awake", "OnEnable", "Update", "FixedUpdate" i "OnCollisionEnter".

"Start" i "Awake" imaju sličnu funkcionalnost, a izvršavaju se samo jednom. Razlika je u tome što se "Awake" izvršava kada se objekt instancira, dok se "Start" izvršava tek kada se objekt aktivira prvi put, tj nakon awake. Obje metode se većinom koriste za inicijalizaciju skripti i postavljanje referenci na duge objekte.

"OnEnable" se razlikuje od "Start" po tome što se izvršava svaki put kad se objekt aktivira, tj. kad se objekt postavi u aktivno stanje.

"Update" i "FixedUpdate" se razlikuju u tome što se "Update" poziva svaki frame, dok se "FixedUpdate" poziva za svaki fiksni vremenski interval. Iz tog razloga se "FixedUpdate" često koristi za simulaciju fizike.

"OnCollisionEnter" je samo jedna od metoda iz familije OnCollision/OnTrigger i koristi se za izvršavanje koda kada se dva objekta sudare.

3. Razvoj igre

U procesu razvoja video igre "Shadow Tomb Valley", inspiracija je pronađena u video igri "Final Fantasy" koja je nastala 1987. godine. Slično kao u "Final Fantasy" igri, u "Shadow Tomb Valley" igrač se suočava s slučajnim borbama u kojima sakuplja iskustvo i zlato. Kroz prikupljeno iskustvo, igrač otključava nove klase i vještine kako bi bio sposoban poraziti glavnog neprijatelja, "Shadow Kinga". Ova igra naslanja se na klasični RPG model u kojem je napredovanje kroz borbe ključno za razvoj lika i napredovanje u priči. Također, igrač ima mogućnost otključavanja novih likova koji mu pomažu u borbi protiv "Shadow Kinga".

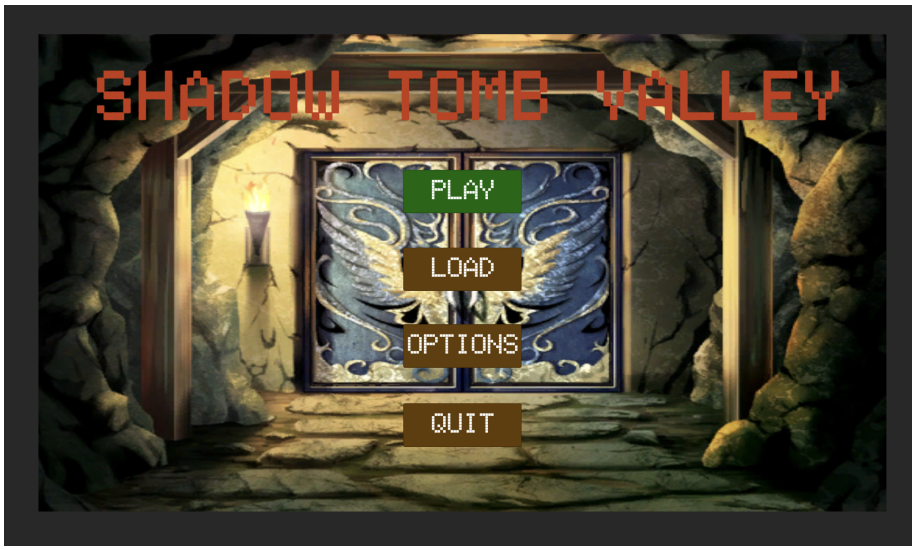
U nastavku će se prikazati proces razvoja igre "Shadow Tomb Valley". Počet će se s opisom različitih scena i načinom njihovog postavljanja. Zatim će se detaljnije opisati bitne skripte, kao što je "GameManagement", koja služi kao središnja skripta za upravljanje igrom. Nakon toga, prikazat će se kako su skripte povezane s scenama i na koji način komuniciraju međusobno. Ovaj proces omogućit će bolje razumijevanje arhitekture i funkcioniranja same igre.

3.1. Scene

U ovom dijelu istražit će se različite scene prisutne u igri te detaljno analizirati njihova funkcionalnost i način integracije unutar cjelokupnog igračkog iskustva. Osvrnut će se na različite aspekte implementacije, uključujući grafički prikaz, interakciju s korisnikom te funkcionalnosti koje svaka scena pruža unutar igre.

3.1.1. Glavni izbornik

Glavni izbornik(eng. Main menu) scena je bitan dio svake video igre. U glavnom izborniku igrači imaju mogućnost prilagoditi postavke igre, pregledati kontrolnu shemu, podešavati glasnoću zvuka, promijeniti rezoluciju, pokrenuti prethodno spremljenu igru te izaći iz igre. Na slici 9 je prikazan glavni izbornik igre koji je implementiran pomoću interaktivnih gumba. Gumbi(eng. button) su unaprijed generirani objekti koji dolaze s komponentom "Button" te djetetom "Text" za prikazivanje teksta na gumbu. Ovo omogućuje igračima intuitivno interagiranje s glavnim izbornikom kako bi prilagodili postavke prema svojim željama.



Slika 9: Glavni izbornik (autorski rad)

Button komponenta se sastoji od tri bitna dijela: postavki za grafički prikaz, postavki za navigaciju te postavki za pritisak na gumb.

- Postavke grafičkog prikaza:

  1. Slika na gumbu: Definira sliku koja se prikazuje na gumbu, poput ikone ili teksta.
  2. Promjena boje: Omogućuje promjenu boje gumba ovisno o trenutnom stanju, kao što je odabran ili kliknut.

- Navigacija:

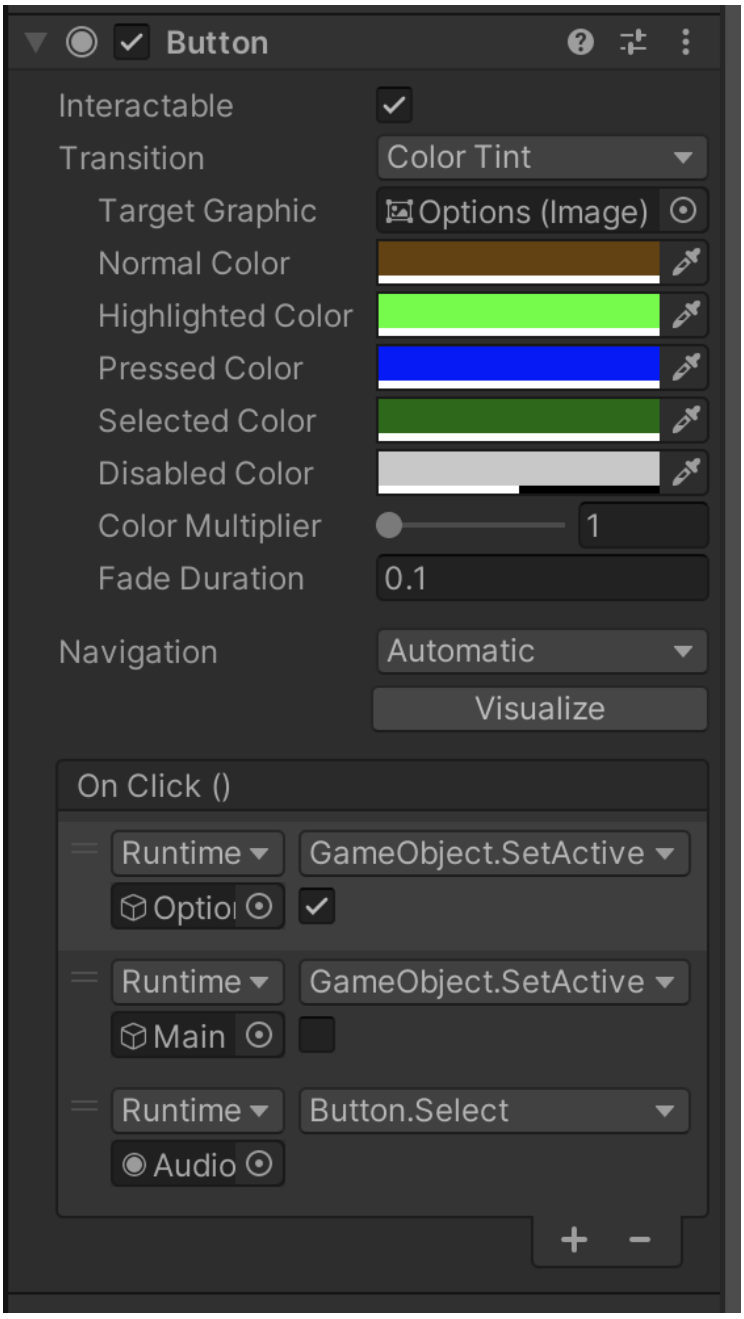
  1. Automatska navigacija: Unity preuzima kontrolu nad navigacijom i automatski postavlja kako se gumbi fokusiraju i kreću među njima, ovisno o njihovom položaju na ekranu.
  2. Eksplicitna navigacija: Omogućuje ručno postavljanje navigacije između gumba.

  U ovom radu koristi se oba načina postavljanja navigacije ovisno o potrebama scene. Na trenutnoj sceni, navigacija je postavljena automatski s obzirom da je sama navigacija na sceni jednostavna i Unity uspješno automatski može postaviti navigaciju.

- Postavke za pritisak na gumb:

  1. Događaji: Definiraju se događaji koji se izvršavaju kada je gumb pritisnut. Ti događaji mogu biti jednostavni, poput aktivacije/deaktivacije objekta, ili složeniji, poput izvođenja funkcije u povezanoj skripti.

  Primjerice, gumb "Options" može izvršiti deaktivaciju početnih objekata i aktivaciju potrebnih objekata za prikaz opcija. S druge strane, tipka "Play" može izvršiti funkciju iz povezane skripte koja započinje igru i postavlja inicijalne vrijednosti za početak igre. Ovo omogućuje interaktivnost korisnika s igrom te izvođenje različitih akcija ovisno o odabiru gumba.



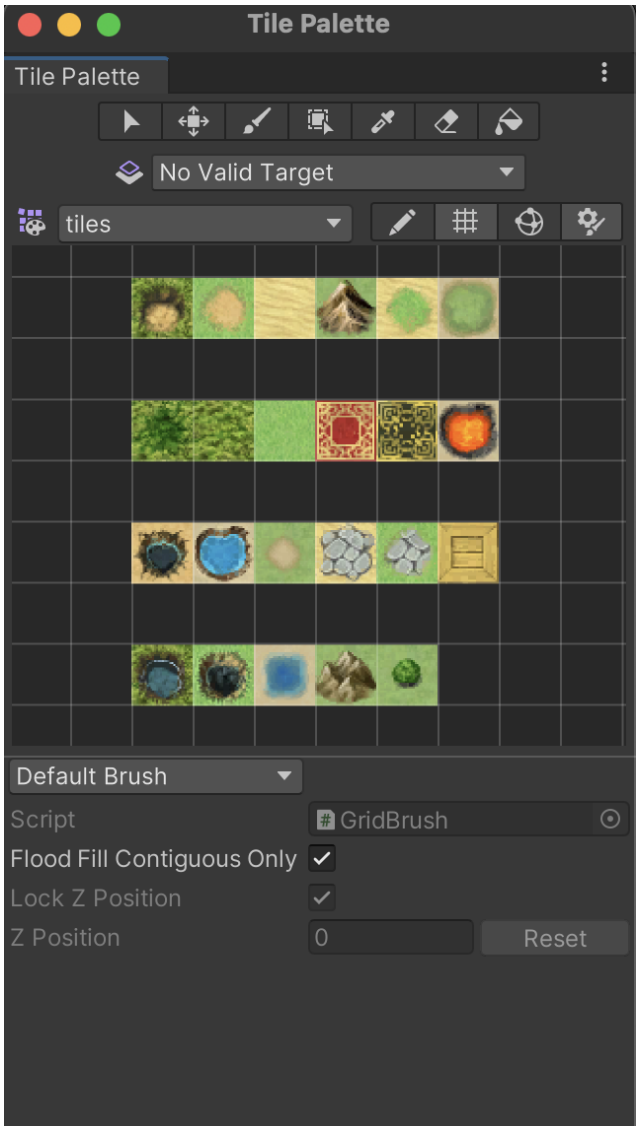
Slika 10: Tipka opcija (autorski rad)

3.1.2. Scene svijeta

Igra "Shadow Tomb Valley" se sastoji od 11 scena, od kojih su pet scena po kojima se igrač može kretati. Tih pet scena nazivaju se "svijet" scene. Sve su ove scene realizirane korištenjem objekta "Grid" koji služi kao roditelj za objekt "Tilemap". Objekt "Grid" koristi se za prikaz koordinatnog sustava podijeljenog u kvadratiće veličine 1x1. Objekt "Tilemap" je dijete "Grida" i služi za popločavanje "Grida". Ova dva objekta su važna za razvoj 2D igara jer omogućuju brzo i jednostavno uređivanje izgleda scene.

Objekti "Grid" i "Tilemap" ne dolaze u standardnoj instalaciji Unity platforme te je potrebno preuzeti paket "2D Tilemap Editor" putem "Package Manager"-a. Da bi se mogao koristiti

sistem popločavanja, također je potrebno napraviti ploče (engl. "tiles") koje se žele postaviti na mapu. Prvi korak je stvaranje "Tile Palette"-a, kao što je prikazano na Slici 11. U "Tile Palette" se dodaju spriteovi (slike) koje će se koristiti za popločavanje mape. Ovaj proces omogućuje dizajnerima da brzo i jednostavno stvore raznolike mape koristeći pripremljene spriteove.



Slika 11: Tile Palette (autorski rad)

Također, osim običnih ploča, moguće je koristiti i ploče s pravilima, poznate kao "Rule Tile" (hrv. Ploča s pravilima). To su posebne vrste ploča koje imaju određena pravila ponašanja. Prilikom postavljanja "Rule Tilea" u Unityju, automatski će se mijenjati izgled ploča prema pravilima koja su postavljena. Na slici 12 je prikazano nekoliko pravila jednog takvog "Rule Tilea" korištenog u izradi igre.

Pravila "Rule Tilea" sastoje se od kvadrata podijeljenog u 9 manjih kvadratića. Svaki od tih manjih kvadrata služi za vizualizaciju mreže, pri čemu središnji kvadrat označava poziciju na kojoj se postavlja ploča, dok ostali kvadrati označavaju susjedne pozicije. Susjedni kvadrati mogu biti:

- Prazni
- Sadržavati zelenu strelicu
- Sadržavati crveni x

Da bismo bolje razumjeli kako pravila funkcioniraju, možemo pogledati primjer prvog pravila na slici. Strelica označava da mora postojati gornji susjed, dok x-ovi označavaju da ti susjedi ne smiju postojati. Prazni kvadrati označavaju da za izgled ploče nije bitno što se nalazi na tim pozicijama.

Za generiranje potpunog "Rule Tilea" za 2D igru, potrebno je koristiti 47 spriteova. Međutim, "Rule Tile" omogućava postavljanje mnogo više pravila. Nakon što se postavi svih 47 pravila, dovoljno je otvoriti "Tile Palette", odabrati napravljeni "Rule Tile", te pritiskom miša na željenu lokaciju postaviti ploču. Ako se zadovolje pravila definirana u "Rule Tileu", spriteovi na tim pozicijama će se automatski mijenjati u spriteove postavljene unutar "Rule Tilea". Ovaj proces omogućuje jednostavno i efikasno popločavanje mape koristeći predefinirane setove pravila.

Slika 12: Rule tile (autorski rad)

U izradi igre "Shadow Tomb Valley" korišteno je nekoliko "Rule Tileova" i više "Tilemapova". Razlog korištenja više "Tilemapova" je taj što pruža dodatne opcije za uređivanje izgleda

mape. Na slici 13 se vidi primjer toga. Korištenjem odvojenih "Tilemapova" omogućuje se postavljanje različitih dijelova mape neovisno jednih o drugima. Na primjer, na jednom "Tilemapu" može se postaviti baza mape, dok se na drugom "Tilemapu" mogu postaviti stolovi i stolice. Korištenjem jednog "Tilemapa" za sve te elemente, morao bi se napraviti sprite koji sadrži bazu, stol i bocu kao jednu cjelinu. Korištenjem odvojenih "Tilemapova" omogućuje se veća fleksibilnost i lakše uređivanje izgleda mape, što je posebno korisno u kompleksnijim scenarijima gdje su dijelovi mape različiti po svojoj funkciji ili stilu.

Slika 13: Primjer korištenja više tile mapova (autorski rad)

Dodavanjem komponente "Tilemap Collider 2D" možemo stvoriti "Tilemap" s objektima s kojima se igrač sudara. Na primjeru slike 18, igrač može slobodno hodati po crvenoj površini, ali ne može proći kroz stolice i stol jer su ti dijelovi mape definirani kao objekti s kolizijom. Ovo omogućuje precizno definiranje prostora kroz koje igrač može prolaziti i prostora s kojim se sudara. Dodavanje "Tilemap Collider 2D" čini interakciju igrača s okolinom u igri vrlo intuitivnom i preciznom.

3.1.3. Dialog

Scena dijaloga u igri se aditivno učitava i omogućuje interakciju igrača s neigrivim likovima, kao i kupnju i prodaju predmeta. Ova scena se sastoji od skripte za prikaz teksta dijaloga i panela za trgovinu predmetima. Iako postoji mnogo paketa za dijaloge u Asset Sto-

reu, za potrebe ovog projekta razvijen je vlastiti sistem dijaloga. Ovaj dijalog omogućuje igraču komunikaciju s likovima unutar igre.

Slika 14: Primjer dijaloga sa trgovcem (autorski rad)

U ovom radu detaljno će biti opisano kako koristiti implementirani dijalog za postavljanje različitih neigrivih likova s različitim funkcionalnostima. Također, bit će prikazano kako se dijalog može koristiti za postavljanje opcija i kako na temelju odabrane opcije izvršiti odgovarajuću radnju.

3.1.4. Pauza

Scena pauze u igri, slično kao i dijalog, aditivno se učitava i omogućuje pauziranje igre te pristup pregledu podataka o igračevim likovima, predmetima itd. Na slici 15 možete vidjeti izgled scene pauze tijekom pregleda predmeta.

Slika 15: Pauza (autorski rad)

3.2. Skripte

U ovom dijelu će se detaljno analizirati ključne skripte i koncepte koji su korišteni u razvoju igre. Obradit će se kako ove skripte funkcioniraju, istražiti njihove funkcionalnosti te

objasniti na koji način dizajner može prilagoditi ili proširiti igru bez potrebe za dodatnim kodiranjem. Osim toga, istaknut će se mogućnosti koje omogućuju promjene u igri, poput dodavanja novih likova, prilagođavanja dijaloga ili mijenjanja događaja unutar dijaloga, što omogućuje fleksibilnost u prilagodbi igre prema specifičnim zahtjevima.

3.2.1. Scriptable object

Unity Scriptable Objects predstavljaju ključan koncept u Unity okruženju koji omogućuje pohranu podataka i logike unutar projekta. Ova tehnologija pruža sredstvo za organizaciju podataka i funkcionalnosti igre na način koji je fleksibilan, skalabilan i lako prilagodljiv potrebama razvoja igre. Scriptable Objects se ističu svojom neovisnošću o instanciranju, omogućujući njihovo stvaranje kao asseta unutar projekta i višestruku upotrebu u različitim kontekstima. Osim toga, oni se mogu koristiti za pohranu različitih tipova podataka, uključujući složene strukture podataka i logike igre. Primjena Unity Scriptable Objects proteže se na različite aspekte razvoja igre, kao što su pohrana konfiguracijskih podataka, definiranje ponašanja elemenata igre i implementacija sustava događaja i obavijesti unutar igre. Njihova jednostavna interakcija s Unity Editorom omogućuje laku izradu i uređivanje, čime se olakšava proces razvoja i iteriranje kroz različite verzije podataka.

Ukratko, Unity Scriptable Objects predstavljaju snažan alat za organizaciju i upravljanje sadržajem igre, čija je primjena ključna za ostvarivanje uspješnog i efikasnog razvoja igre u Unity okruženju.

U ovom projektu koncept scriptable objects-a koristi se u potpunosti - skoro svi objekti unutar igre implementirani su koristeći ovaj koncept. Primjeri obuhvaćaju likove igrača, neigrive likove, protivnike, dijaloge, događaje, predmete, klase, vještine, i mnoge druge. Ovaj pristup omogućuje dizajnerima da mijenjaju i dodaju elemente bez potrebe za programiranjem. Na primjer, prodavač je scriptable object koji sadrži listu predmeta koje prodaje. Budući da su i sami predmeti scriptable object-i, moguće je imati više različitih prodavača s različitim dijalozima i/ili predmetima. Na taj način, lako se može dodati novi predmet u prodaju, promijeniti dijalog prodavača ili čak zamijeniti cjelokupnu instancu prodavača s drugom.

3.2.2. GameManagement

GameManagement je ključna skripta u igri, djelujući kao centralno "skladište podataka" tijekom igranja. Sadrži informacije o likovima koje igrač upravlja, objektima koji se mogu spremiti tijekom igre te nekoliko važnih metoda za samu igru. Implementirana je kao singleton, što znači da postoji samo jedna instanca objekta. Ovo osigurava dosljedan pristup objektu GameManagement u cijelom kodu, čime se jamči konzistentnost podataka i njihova dostupnost drugim skriptama.

Skripta sadrži metode za dodavanje likova, postavljanje početnih postavki borbe te metode za spremanje i učitavanje igre. Također, sadrži listu objekata s kojima igrač može interaktirati i već je interaktao, a koji se ne trebaju prikazivati prilikom ponovnog učitavanja scene. Uz to, koristi se i uzorak dizajna "Composite" za spremanje i učitavanje igre. Composite uzorak

grupira objekte u strukturu stabla, što omogućuje skripti za spremanje da poziva metodu "Save" nad objektom GameManagement, a da svaki objekt unutar strukture obavlja potrebne radnje za spremanje podataka. Na taj način skripta za spremanje samo treba poznavati korijen stabla, dok ostatak logike ostaje unutar composite strukture.

```
public void Save(ref SaveData saveData)
{
    foreach (ISaveable saveable in saveables)
    {
        saveable.Save(ref saveData);
    }

    saveData.gameData.enemiesKilled = enemiesKilled;
    saveData.gameData.time = timePLayed + Time.realtimeSinceStartup;
    GameObject player = GameObject.FindGameObjectWithTag("Player");
    saveData.gameData.playerPositionX = player.transform.position.x;
    saveData.gameData.playerPositionY = player.transform.position.y;
    saveData.gameData.scene = SceneManager.Instance.GetMainSceneName();
    saveData.gameData.interactableIdsToNotRender =
        interactableIdsToNotRender;
}
```

3.2.3. SceneManager

SceneManager skripta djeluje kao omotač (eng. wrapper) nad Unityjevim SceneManagerom te je implementirana kao singleton. Svrha joj je olakšati rad sa scenama. Sastoji se od četiri metode koje služe za učitavanje nove scene, uklanjanje scene, dohvaćanje trenutno aktivne scene te dohvaćanje scene glavnog izbornika. Korištenjem ove skripte osigurava se korištenje standardnih metoda koje se mogu promijeniti unutar Unitya, a koje su se već mijenjale kroz starije verzije. U slučaju promjene u Unity SceneManageru, potrebno je samo na jednom mjestu mijenjati kod umjesto na svakom mjestu gdje se radi sa scenama.

```
public void LoadScene(int index, bool additive = false, bool setActive =
    false)
{
    UnityEngine.SceneManagement.SceneManager.LoadScene(
        index,
        additive ? UnityEngine.SceneManagement.LoadSceneMode.Additive :
            UnityEngine.SceneManagement.LoadSceneMode.Single
    );

    if (setActive)
    {
        CallAfterDelay.Create(0, new Action(() =>
        {
            UnityEngine.SceneManagement.SceneManager.SetActiveScene(
                UnityEngine.SceneManagement.SceneManager.
                    GetSceneByBuildIndex(index));
        })
        );
    }
```

```
}
```

U gornjem kodu primjećujemo da se učitavanje scene izvršava korištenjem Unityjeve "SceneManagement" skripte, uz dodatak dijela koda koji postavlja novu scenu kao trenutno aktivnu scenu. Taj dio je bitan prilikom aditivnog učitavanja scene, gdje želimo da nova učitana scena postane aktivna scena.

3.2.4. Event i EventListener

Unity već ima ugrađene mehanizme za izvršavanje akcija temeljenih na događajima, ali za potrebe ovog projekta bilo je potrebno implementirati oblik klasičnih "Event" i "EventListener" koncepata. U projektu su definirane dvije osnovne klase za oba objekta. Svaka od njih ima baznu klasu koja koristi druge objekte kao parametre te klasu koja ne zahtijeva parametre prilikom izvršavanja. Korištenjem ovih klasa i klasa koje ih nasljeđuju, postiže se odvajanje klasa i eliminira potreba da jedna klasa poznaje objekte različitih klasa na sceni kako bi s njima radila. Na primjer, za kretanje igrača, jedna skripta provjerava kretanje igrača i pokreće odgovarajući događaj za kretanje, obavještavajući tako druge skripte da se igrač kreće i da trebaju izvršiti odgovarajuću logiku. Klasa zadužena za animaciju kretanja ne provjerava kretanje igrača svaki frame, već reagira na događaj kretanja koji primi, pokrećući animaciju ovisno o smjeru kretanja igrača.

3.2.5. CallAfterDelay

Ova skripta ima ključnu ulogu u cijeloj igri jer omogućuje izvođenje određene akcije s odgodom. Time se kontrolira trenutak kada se nešto izvodi, omogućavajući izvršavanje koda prilikom aditivnog učitavanja scene bez potrebe da nova scena pozna prethodnu. Na primjer, u slučaju dijaloške scene, ne mora se znati s kojim likom je igrač stupio u interakciju, već se koristi ova klasa za postavljanje početnih komponenti dijaloške scene s odgovarajućim vrijednostima ovisno o liku. Korištenjem ove klase i event klase, kada igrač stupi u interakciju s likom, stvara se instanca ove klase koja pokreće događaj koji sadrži objekt lika s kojim je igrač stupio u interakciju. Nakon što se učita dijaloška scena, slušač na toj sceni postavlja tog lika kao lika s kojim je igrač stupio u kontakt, omogućujući sceni da prilagodi interakciju na temelju identiteta lika, primjerice trgovca ili gostioničara.

3.2.6. Dialogue

Kao što je već spomenuto, postoje mnogi paketi za implementaciju dijaloga, ali u ovom projektu dijalog je razvijen bez korištenja tih paketa. Glavni elementi su klasa Dialogue, koja interagira sa Scriptable Object klasom Dialogue i NPC klasom, koja predstavlja interaktivne likove s "Dialogue" klasom. Scriptable Object "Dialogue" sadrži listu "DialogueLine" objekata, dok se svaka "DialogueLine" sastoji od teksta, liste "DialogueChoice" objekata, te "UnityEvent" objekt za upravljanje događajima tijekom dijaloga. "DialogueChoice" obuhvaća tekst i "UnityEvent" koji upravlja događajima odabira. Kroz ovu strukturu, klasa Dialogue nije usko povezana s bilo

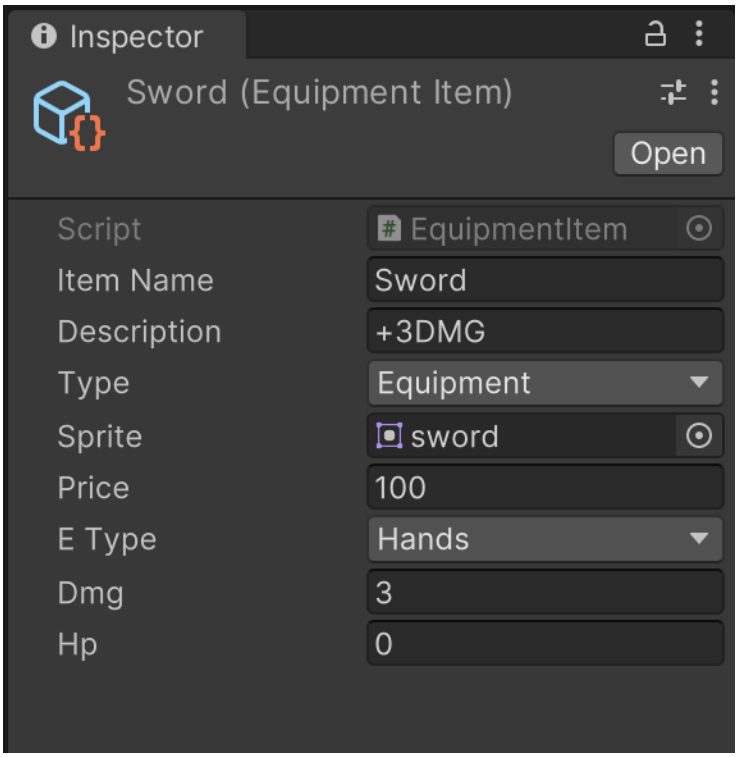
kojim objektom osim s UI elementima, već poznaje nadređene klase svih elemenata i može s njima interaktivno komunicirati. Ova skripta omogućuje prolazak kroz dijalog lika s kojim je igrač interaktirao i prikaz trgovine unutar igre.

3.2.7. BattleController

Klasa "BattleController" je odgovorna za simulaciju borbe u igri. Unutar ove klase nalaze se brojne reference na različite UI objekte na sceni borbe kako bi se mogli prikazati svi potrebni podaci. Također, sadrži statičke varijable koje se koriste za postavljanje borbe, uključujući zvučne zapise za borbu i pozadinu, kao i informacije o protivnicima. Borba se odvija potezno, pri čemu igrač prvi igra, a zatim protivnici. Važno je napomenuti da igra koristi koncept scriptable objects-a, pa ova skripta ne sadrži informacije o protivnicima, već se oni postavljaju kroz GameManagement skriptu prije početka borbe.

3.2.8. Predmeti

Klasa "ItemObject" je temeljna klasa koja predstavlja predmete unutar igre. Sadrži osnovne podatke kao što su ime, opis, slika i cijena predmeta. Osim toga, postoje dva glavna tipa predmeta: obični predmeti koji služe za liječenje igrača i predmeti koji predstavljaju opremu. Oprema, osim navedenih atributa, također sadrži podatke o zdravlju i napadu koji se dodaju liku igrača kada ih postavi na odgovarajući lik. Oprema je dalje podijeljena u četiri tipa: glavu, prsa, noge i ruke, pri čemu svaki lik može nositi po jedan komad opreme od svake vrste. Na slici 16 vidi se primjer mača koji je tip opreme koji se postavlja na ruke lika.



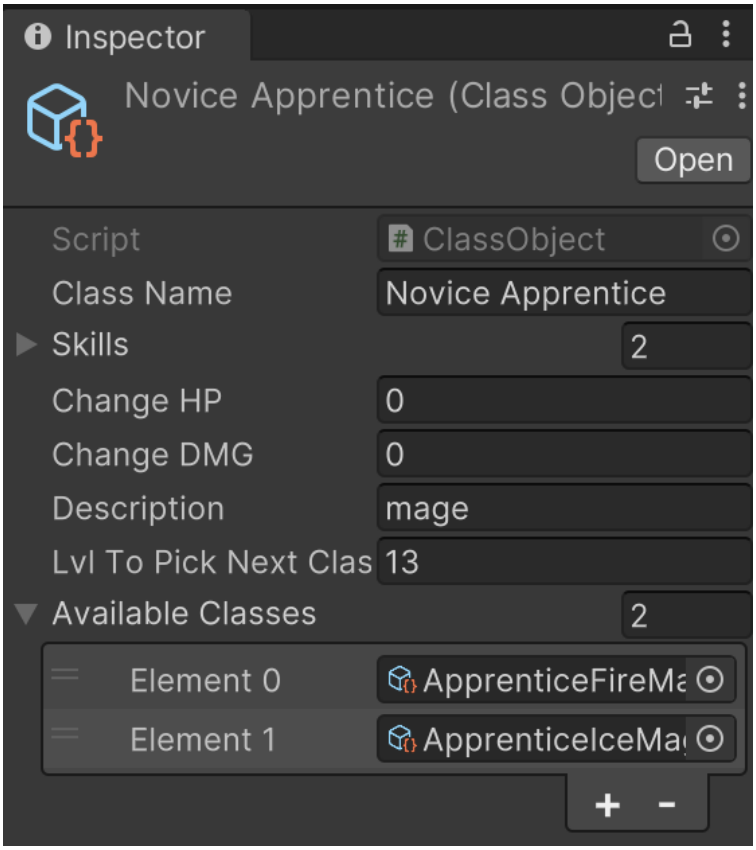
Slika 16: Primjer predmeta (autorski rad)

3.2.9. Igrivi likovi

Igrivi likovi u igri se sastoje od više klasa potrebnih za njihovo predstavljanje. Početna klasa je "BaseUnit", koja je apstraktna i sadrži strukturu podataka kao što su ime, napad, nivo, zdravlje, mana i iskustvo. Osim ovih osnovnih podataka, klasa također uključuje dvije slike koje predstavljaju likove na mapi i u borbi. Na primjer, slika koja predstavlja igračev lik na mapi gleda prema igraču, dok je slika u borbi orijentirana prema desnoj strani.

Igračevi likovi, osim općih podataka, također sadrže informacije o njihovoj klasi (npr. "Wizard"), vještinama (npr. "FireBall") i predmetima koje nose. Svi ovi podaci implementirani su korištenjem koncepta scriptable objecta. Na primjer, klasa koja predstavlja igračevu klasu sadrži informacije poput imena klase, vještina koje lik može naučiti odabirom te klase, kao i promjena u zdravlju ili napadu. Svaka klasa također definira nivo koji igračev lik mora dostići kako bi mogao odabrati tu klasu, te koje klase postaju dostupne kasnije u igri.

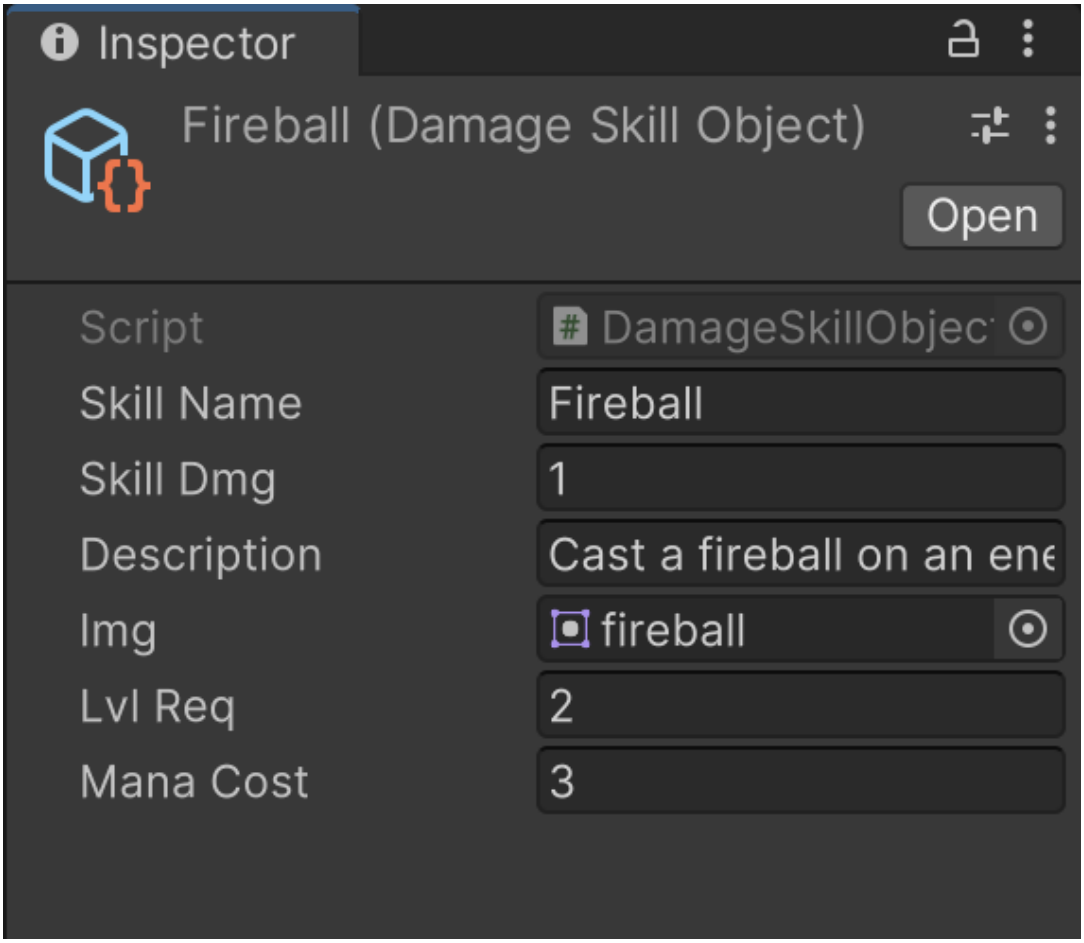
Primjerice, na slici 17 možemo vidjeti kako klasa "Novice Apprentice" sadrži podatke o sljedećim dvjema klasama - "Apprentice Ice Mage" i "Apprentice Fire Mage". Korištenjem scriptable objecta moguće je kreirati više početnih klasa za različite likove te na taj način granati njihovu progresiju i razvoj prema potrebama igre, bez potrebe za dodatnim kodiranjem.



Slika 17: Primjer klase (autorski rad)

Na slici 18 prikazani su atributi koji definiraju vještine, kao što su ime, šteta koju nanose, opis, slika, minimalni nivo lika potreban da bi se naučila vještina te trošak mane za korištenje vještine. Ovaj pristup omogućuje prilagodbu svakog aspekta vještine prema zahtjevima igre,

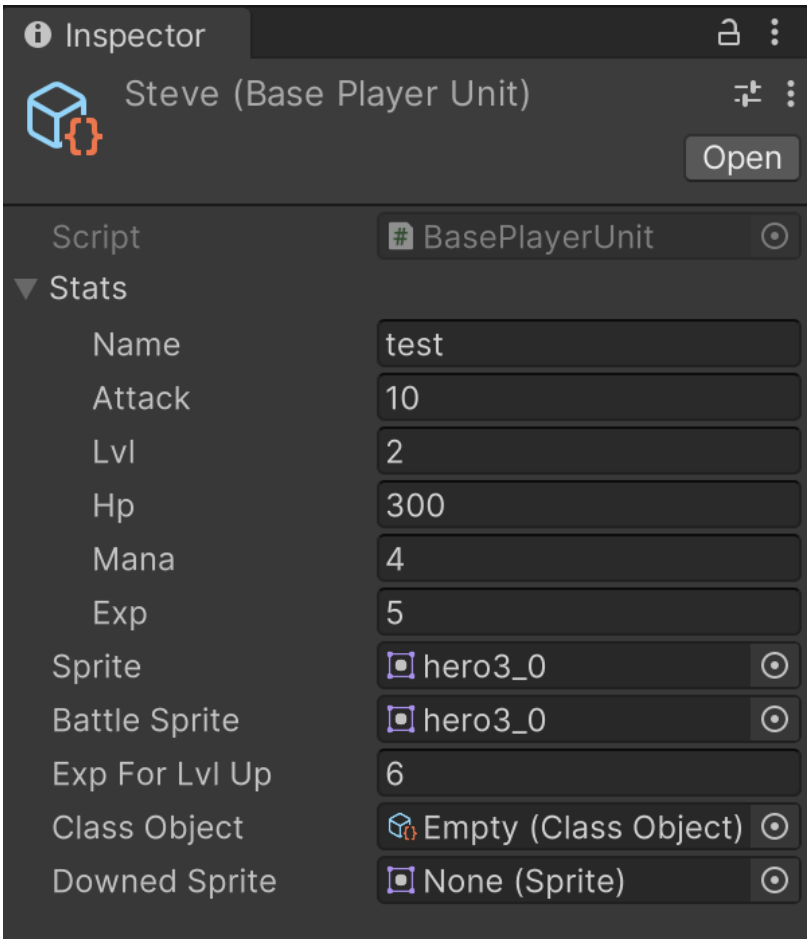
uz mogućnost jednostavnog ažuriranja podataka u slučaju potrebe.



Slika 18: Primjer vještine (autorski rad)

U igri su implementira dva tipa vještina, a to su napadačke vještine i vještine podrške. Napadačke vještine kao što ime kaže služe za napadanje protivnika, dok vještine podrške služe za liječenje igrača.

Uz korištenje scriptable object klasa, u igri se koriste i obične klase koje sadrže podatke o likovima kroz igru. Ova praksa je uspostavljena zbog toga što scriptable object objekti bilježe svaku promjenu podataka i služe samo za prikaz inicijalnih vrijednosti. Dakle, za svakog lika, bilo da je igračev ili protivnik, postoji skripta zadužena za njihovo predstavljanje tokom same igre. Na slici 19 prikazan je scriptable object koji predstavlja igrivog lika.



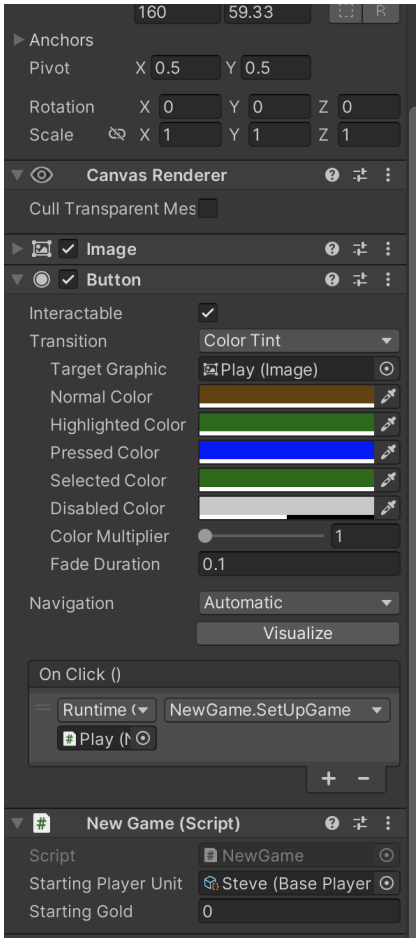
Slika 19: Primjer igrivog lika (autorski rad)

4. Igra

Sada kada smo razmotrili scene i ključne skripte, pogledat ćemo kako se one međusobno povezuju i funkcioniraju. Proći ćemo kroz postavljanje scena te prikazati primjere metoda koji su važni za igranje kroz određene scene. Također, upoznat ćemo se s dodatnim klasama koje su korištene, a nisu bile spomenute u prethodnom poglavlju.

4.1. Početak igre

Početna scena koja se prikazuje prilikom pokretanja igre je glavni izbornik. Glavni izbornik, kao što smo već opisali, sastoji se od različitih tipki, od kojih svaka ima svoju funkcionalnost. Na slici 20 možemo vidjeti strukturu tipke "Play", koja se sastoji od nekoliko komponenata i jedne skripte. Ta skripta sadrži metodu za postavljanje početnih vrijednosti igre i pokretanje same igre.



Slika 20: Play tipka (autorski rad)

```
public void SetUpGame()
{
    PartyMember partyMember = new(startingPlayerUnit);
    GameManagement.Instance.AddMember(partyMember);
```

```
        Inventory.Inventory.Instance.GoldAmount = startingGold;
        Time.timeScale = 1;
        PauseGame.canPause = true;

        SceneManager.Instance.LoadScene(0);
    }
```

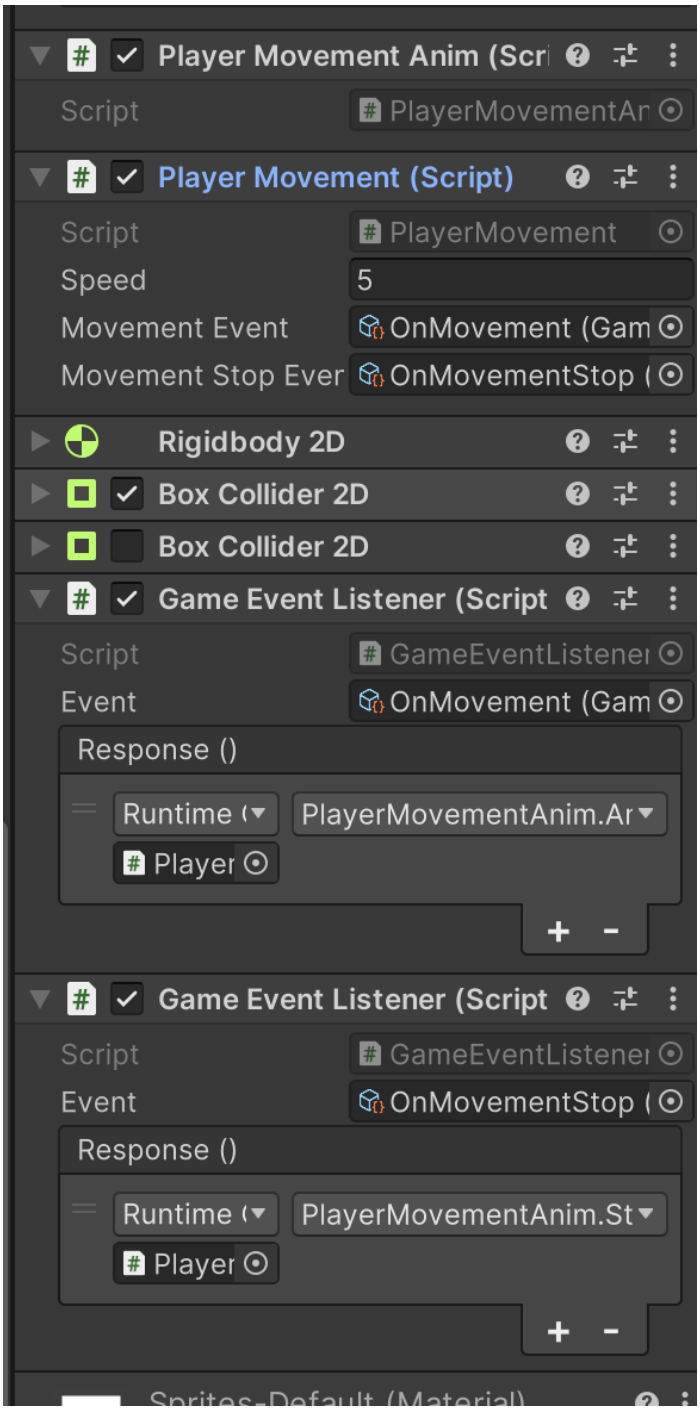
Prikazana skripta postavlja početnog lika i određenu količinu zlata koju igrač posjeduje na početku igre. I početni lik i početno zlato se mogu mijenjati kroz inspektor kako bi se mogli postaviti ovisno o potrebama igre.

4.2. Početno selo

U ovom dijelu ćemo istražiti kretanje igrača i proces prijelaza između različitih scena. Zatim ćemo istražiti implementaciju dijaloga, predmeta te kako su oni povezani s različitim scenama i kako funkcioniraju u igri.

4.2.1. Kretanje igrača

Početna scena igre predstavlja kuću igračevog lika. Cilj igrača je izaći iz kuće. Scena je strukturirana s gridom koji definira izgled same lokacije, objektom igrača te objektom za "teleportiranje" igrača na novu scenu. Na slici 21 prikazano je kako je objekt igrača opremljen s dvije skripte i komponentama tijela Unityja za simulaciju njegovog ponašanja.



Slika 21: Objekt igraca (autorski rad)

```
void FixedUpdate()
{
    if (GameManagement.Instance.Paused == true
    || GameManagement.Instance.PlayerCanMove == false
    || (Input.GetAxisRaw("Horizontal") == 0 && Input.GetAxisRaw("Vertical")
        == 0)
    )
    {
        if (rb2D.velocity.magnitude != 0)
        {
```

```
            movementStopEvent.Raise();
        }

        rb2D.velocity = new Vector2();

        return;
    }

    Vector2 movement = new()
    {
        x = Input.GetAxisRaw("Horizontal"),
        y = Input.GetAxisRaw("Vertical")
    };

    movement.Normalize();
    // move the RigidBody2D instead of moving the Transform
    rb2D.velocity = movement * speed;
    movementEvent.Raise();
}
```

Primjer skripte "PlayerMovement" prikazuje kako se provjerava mogućnost kretanja igrača i sam proces kretanja. Ako je igrač u pokretu, na njega se primjenjuje sila kako bi se simuliralo kretanje. Tijekom kretanja, skripta pokreće događaj koji signalizira kretanje igrača. Kada se događaj kretanja aktivira, skripta "PlayerMovementAnim" analizira smjer kretanja igrača i pokreće odgovarajuću animaciju kretanja za taj smjer.

```
public void Animate()
{
    if (Input.GetAxisRaw("Vertical") < 0)
    {
        animator.SetTrigger("Down");
    }
    else if (Input.GetAxisRaw("Vertical") > 0)
    {
        animator.SetTrigger("Up");
    }
    else if (Input.GetAxisRaw("Horizontal") < 0)
    {
        animator.SetTrigger("Left");
    }
    else if (Input.GetAxisRaw("Horizontal") > 0)
    {
        animator.SetTrigger("Right");
    }
}

public void StopAnimation()
{
    AnimationHelper.RessetTrigger(animator);
    animator.SetTrigger("Stop");
}
```

Na ovaj način postiže se efikasnost jer nema potrebe za dva zasebna provjera statusa kretanja

igrača. Umjesto toga, jedna provjera se izvršava svaki frame kako bi se utvrdilo kretanje igrača. Sve akcije povezane s kretanjem, poput animacija, aktiviraju se samo kada se primi signal da se igrač kreće. Ovaj pristup omogućuje bolje upravljanje resursima i smanjuje nepotrebno izvršavanje koda, što doprinosi optimizaciji performansi igre.

4.2.2. Prijelaz između scena

Osim toga, na sceni se nalazi i objekt za teleportiranje igrača. Iako objekt može služiti za teleportiranje unutar same scene, u igri se koristi isključivo za teleportiranje između scena. Ovaj objekt koristi klasu "SceneWarpPoint" i implementiran je pomoću scriptable objekata, pri čemu svaki scriptable objekt sadrži informaciju o sceni na kojoj se nalazi te referencu na drugi objekt iste klase na koji se igrač teleportira. Pored toga, koristi se i klasa "SceneWarp" koja se aktivira kada igrač stupi u koliziju s "SceneWarpPoint" objektom, pokrećući sljedeći kod.

```
void OnTriggerEnter2D(Collider2D collider)
{
    if (collider.CompareTag("Player"))
    {
        SceneChange.onSignal += PositionPlayer;

        SceneManager.Instance.LoadScene(sceneWarpPoint.WarpTo.
            SceneBuildIndex);
    }
}

private void PositionPlayer()
{
    foreach (SceneWarp warp in GameManagement.Instance.warps)
    {
        if (warp.SceneWarpPoint.name == sceneWarpPoint.WarpTo.name)
        {
            var player = GameObject.FindGameObjectWithTag("Player");

            if (player != null)
            {
                player.transform.position = warp.transform.GetChild(0).
                    transform.position;
            }

            break;
        }
    }

    GameManagement.Instance.warps = new();

    SceneChange.onSignal -= PositionPlayer;
}
```

U ovom kodu vidimo da kada igrač stupi u koliziju s "SceneWarpPoint" objektom, skripta učitava novu scenu ovisno o objektu koji predstavlja odredište teleportacije. Nadalje, delegira se me-

toda "PositionPlayer" u novoj sceni. Ova metoda se poziva tek nakon što se učita nova scena i pronalazi sve objekte koji sadrže komponentu "SceneWarp". Zatim dohvaća referencu na objekt "SceneWarpPoint" i pozicionira igrača iznad djeteta tog objekta, koje predstavlja odredište teleportacije. Ovo je važno kako bi se igrač pozicionirao ispravno i kako bi se izbjeglo ponovno teleportiranje među scenama u beskonačnoj petlji.

Nakon što igrač izađe iz kuće, scena početnog sela se učitava. U početnom selu se nalazi gostionica, koja obuhvaća trgovca, gostioničara i još jedna osoba, koja predstavlja novog lika igrača. Ovdje se prvi put susrećemo s likom koji, osim fizičkih komponenti, sadrži i komponentu "PartyMember". Kako bismo postupno uveli sve elemente interaktivnih objekata, objekt koji predstavlja novog lik sadrži samo referencu na scriptable object koji opisuje lika koji se pridružuje igraču. Na taj način je moguće promijeniti lika ili postavke lika bez potrebe za kodiranjem. Prilikom interakcije igrača s likom, novi lik se pridružuje igraču, te igrač tada posjeduje dva lika s kojima se bori u bitkama. Klasa za prikaz lika nasljeđuje klasu "Destroyable", koja predstavlja interaktivne objekte koji se ne prikazuju ponovno nakon što igrač s njima interagira.

Slika 22: Početno selo (autorski rad)

```
protected override void Interact()
{
    GameManagement.Instance.AddMember(new Character.PartyMember(partyMember)
        );
    base.Interact();
}
```

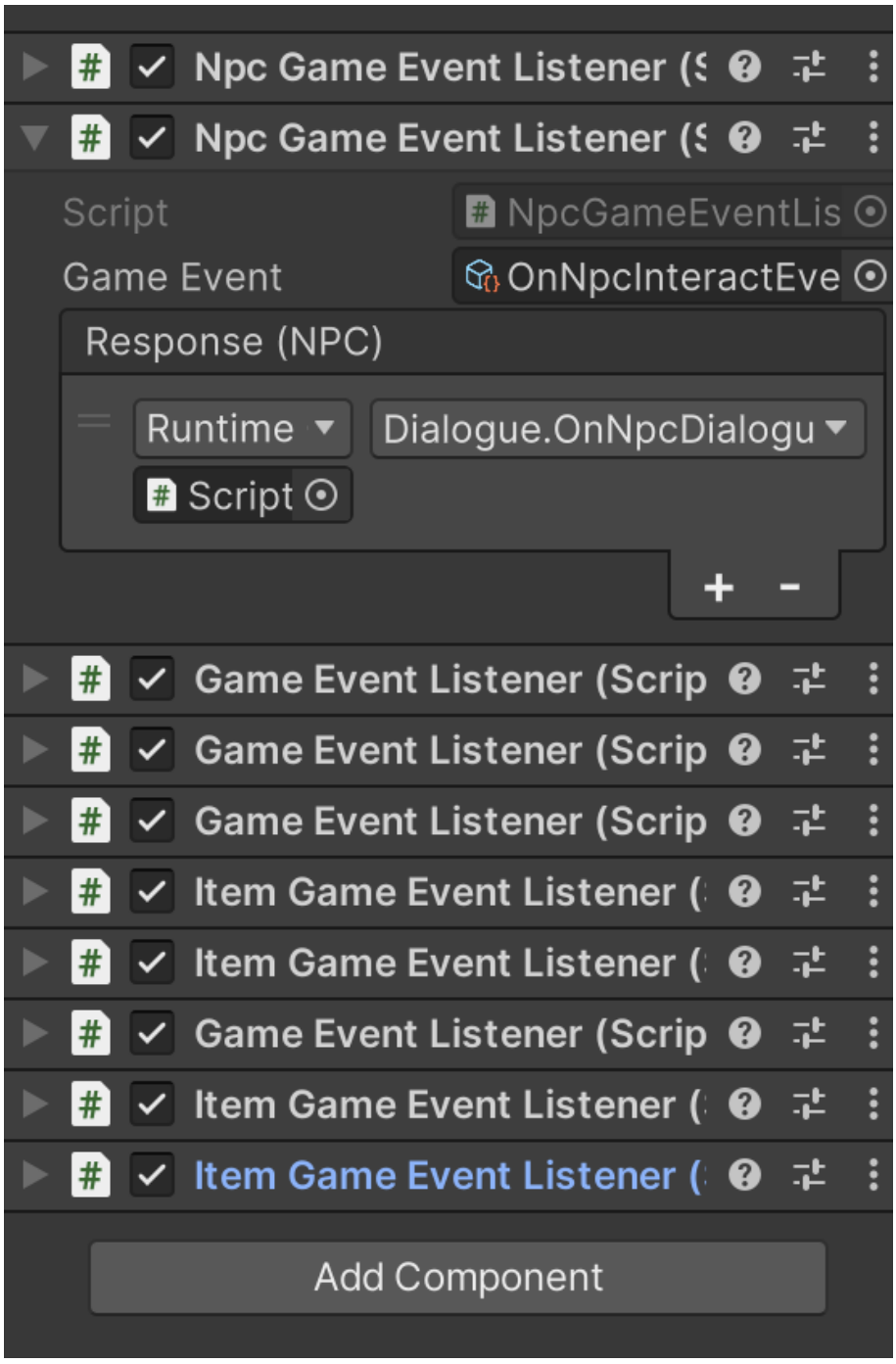
4.2.3. Dijalog

Prilikom ulaska u gostionicu, igrač ima mogućnost interakcije s dvije kutije koje sadrže iteme. Kutije imaju komponentu "BoxContent" koja sadrži referencu na item i količinu itema u kutiji. Kako bi se spriječilo ponovno skupljanje itema prilikom ponovnog učitavanja scene, objekt kutije nasljeđuje klasu "Destroyable". Osim toga, na sceni se nalaze trgovac i gostioničar, a oba objekta sadrže komponentu "NpcGameObject", koja proizlazi iz klase "Interactable". Ovdje se

prvi put susrećemo s konceptima eventa i dijaloga. Prilikom interakcije igrača s jednim od ova dva lika, primjerice trgovcem, pokreće se donji kod koji kreira objekt za odgođeno izvršavanje koda te aditivno učitava scenu dijaloga. Nakon učitavanja scene dijaloga, izvršava se odgođena logika koja podiže event interakcije s neigrivim likom, pri čemu se kao parametar šalje referenca na scriptable object neigrivog lika.

```
protected override void Interact()
{
    if (!interactable)
    {
        return;
    }

    interactable = false;

    CallAfterDelay.Create(0.1f, new Action(() =>
    {
        onNpcTrigger.Raise(npc);
        PauseGame.Instance.StopTime();
        PauseGame.canPause = false;
    })
    );
    SceneManager.Instance.LoadScene(10, true, true);
}
```

U nastavku će se opisati funkcionalnost dijaloga kroz primjer trgovca. Scena sadrži dva objekta opremljena komponentama potrebnim za rad s dijalogom: jedan objekt sadrži skriptu "Dialogue", dok drugi objekt sadrži sve listenere potrebne za izvršavanje logike dijaloga. Kao što je već rečeno, pokretanjem scene aktivira se postavljanje reference na lika s kojim igrač interagira. Na slici 23 prikazan je listener, a ispod njega se nalazi kod koji obavlja potrebne radnje.



Slika 23: Primjer dijaloga sa trgovcem (autorski rad)

```
public void OnNpcDialogue(NPC npc)
{
    this.npc = npc;
    SetDialogue(npc.Dialogue);
}
public void SetDialogue(Core.Dialogue dialogue)
{
    StartCoroutine(PlayDialogue(dialogue));
}
```

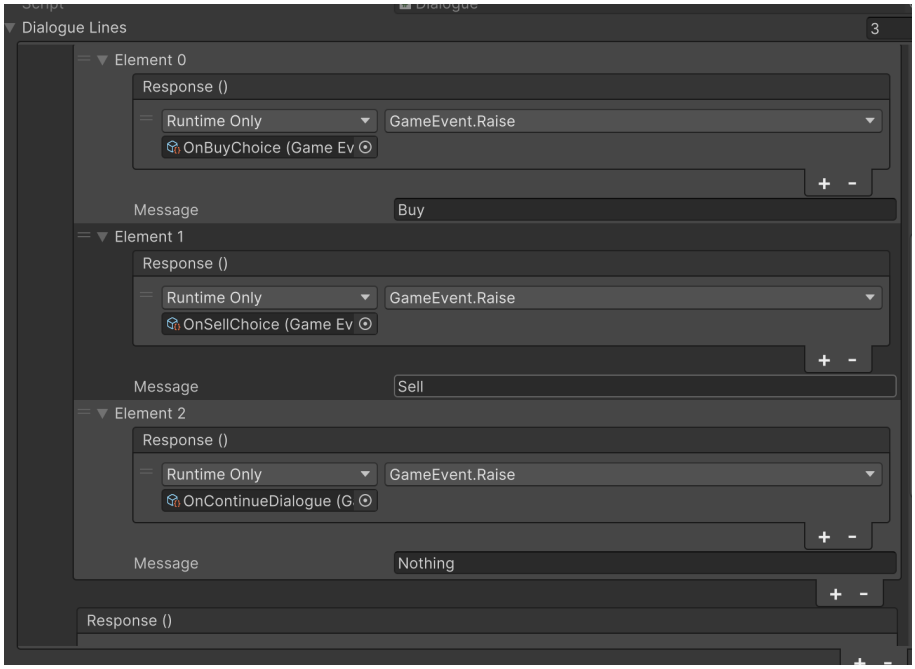
U prikazanom kodu jasno je da se logika dijaloga odvija asinkrono korištenjem korutina. Prili-

kom postavljanja dijaloga, prvo se aktivira objekt koji prikazuje tekst, a zatim se resetira tekst dijaloga. Ovaj korak resetiranja nužan je jer je dijalog implementiran korištenjem koncepta scriptable object, što znači da asset pamti na kojoj smo liniji svaki put kad promijenimo trenutnu liniju dijaloga. Nakon toga, trenutni "DialogueLine" iz dijaloga dohvaća se i postavlja tekst, zajedno s opcijama ako postoje. Tekst ostaje prikazan sve dok igrač ne pritisne razmaknicu ili ne odabere jednu od ponuđenih opcija. Nakon pritiska razmaknice, odvijaju se događaji postavljeni na trenutnoj liniji ili opciji, zatim se prelazi na sljedeću liniju i proces se ponavlja sve dok postoje linije u dijalogu.

```
public void SetDialogue(DialogueLine dialogueLine)
{
    continueDialogue = false;

    foreach (Transform child in choicePanel.transform)
    {
        Destroy(child.gameObject);
    }

    choicePanel.transform.DetachChildren(); // Destroy is not fast enough so
         we detach the children while they are being destroyed to avoid
        problems
    dialoguePanel.GetComponentInChildren<TMP_Text>().text = dialogueLine.
        Message;

    if (dialogueLine.HasChoices)
    {
        choicePanel.SetActive(true);
    }
    else
    {
        choicePanel.SetActive(false);
        return;
    }

    foreach (DialogueChoice dialogueChoice in dialogueLine.DialogueChoices)
    {
        var choice = Instantiate(choicePreflab, choicePanel.transform);
        choice.GetComponentInChildren<TMP_Text>().text = dialogueChoice.
            Message;
        choice.SetActive(true);
        choice.GetComponent<Button>().onClick.AddListener(dialogueChoice.
            Response.Invoke);
    }

    choicePanel.GetComponentInChildren<Button>()?.Select();
}
```

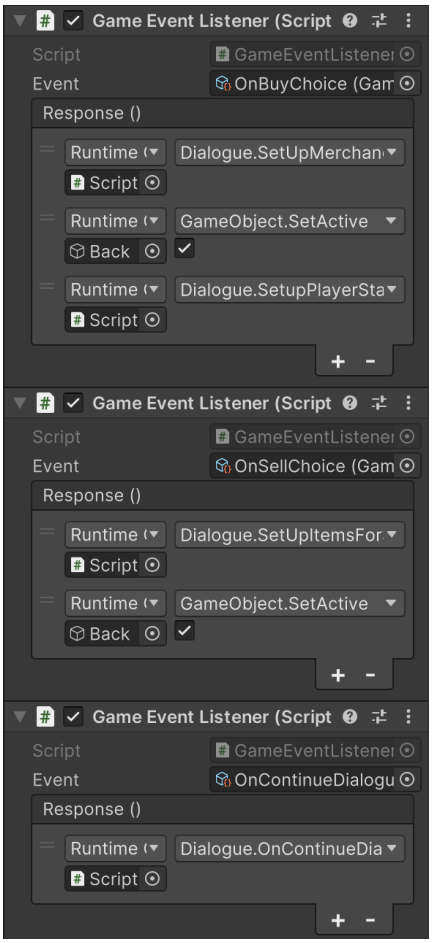
Ovaj dio koda prikazuje postavljanje teksta i opcija u dijalogu. Prvo se uništavaju opcije koje su bile prikazane prije, ako postoje. Zatim se određuje aktivnost panela ovisno o tome postoje li opcije u trenutnoj liniji dijaloga. Ukoliko ne postoje, panel se deaktivira i izvršavanje se prekida. Ako postoje opcije, za svaku od njih se instancira tipka koja predstavlja tu opciju. Na svaku

tipku se dodaju događaji koji su dodani na opciju kako bi se izvršili prilikom pritiska na tipku. Nakon toga se odabire prva opcija kao aktivna tipka na sceni.



Slika 24: Primjer opcija u dijalogu sa trgovcem (autorski rad)

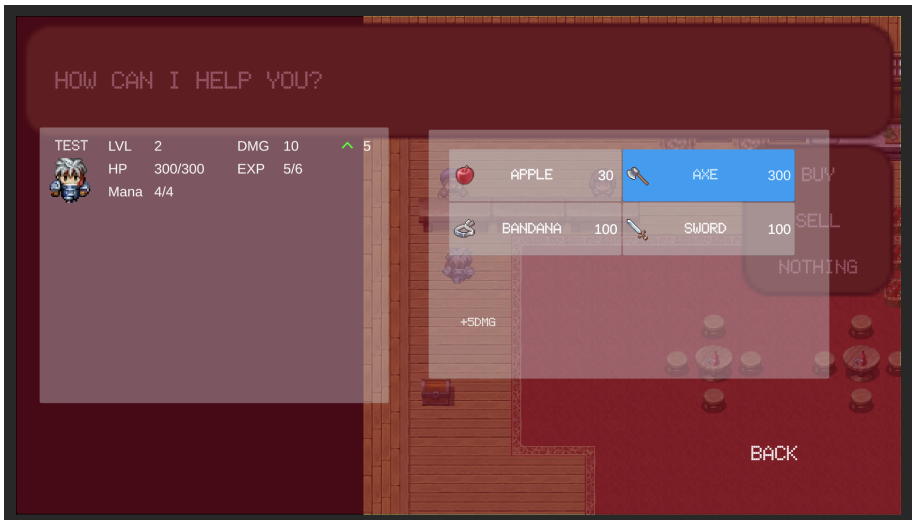
Na slici 14 su vidljive opcije tokom dijaloga sa trgovcem. Prva opcija podiže event za postavljanje kupnje, druga za postavljanje prodaje, a treća za nastavkom dijaloga. Na slici 25 su vidljivi listeneri zaduženi za svaki od tih evenata.



Slika 25: Listeneri za opcije dijaloga sa trgovcem (autorski rad)

4.2.4. Predmeti

Prilikom odabira kupnje, listener poziva metodu za postavljanje predmeta koji se mogu kupiti kod tog trgovca, aktivira se tipka back, te se poziva metoda za prikazivanje podataka o likovima igrača. Slika 12 prikazuje izgled scene prilikom odabira opcije za kupnju predmeta.



Slika 26: Primjer prozora trgovine (autorski rad)

4.2.5. Trgovina

Na slici se vidi prozor za kupovinu s igračevim likovima s lijeve strane i ponuđenim predmetima s desne strane. Pomoću pomoćne klase "ScrollViewHelper" postavljaju se predmeti u pomični prozor. Da bismo bolje razumjeli postavljanje predmeta i tipki koje ih predstavljaju, potrebno je detaljnije pogledati strukturu same tipke.

Tipka koja predstavlja predmet sastoji se od istih podataka kao i predmet te dodatno sadrži podatak o količini predmeta, ako se tipka koristi u pregledu igračevih predmeta, te referencu na sam predmet. Također, sadrži tri događaja koji predstavljaju odabir, pritisak ili poništenje odabira same tipke, a ti događaji se aktiviraju prilikom tih radnji.

Prefab koji predstavlja predmet također sadrži dijete koje je identično samoj tipki. Razlog tome je što Unity, ako se koristi automatska navigacija među tipkama, ne može odabrati neinteraktivnu tipku. Stoga se mora koristiti eksplicitna navigacija. Međutim, korištenjem eksplicitne navigacije nailazi se na problem da ako se odabere tipka koja nije interaktivna, ona ne mijenja boju. Zbog toga, kada se neki predmet ne može kupiti, prodati ili iskoristiti, glavna tipka se postavlja kao neinteraktivna i inicijalizira se kopija kao dijete koja sadrži iste podatke, ali bez događaja na pritisak tipke.

Sada kada smo upoznati sa samim tipkama predmeta, možemo nastaviti s postavljanjem u igri. Metoda za postavljanje predmeta prima objekt koji treba sadržavati prikazane objekte kao djecu, listu predmeta, prefab tipke predmeta, klasu za određivanje interaktivnosti tipki predmeta te tipku za povratak s kupnje.

Prvo se uništavaju objekti koji su djeca roditeljskog objekta, ako postoje. Zatim se prolazi kroz listu predmeta i inicijalizira se tipka koja predstavlja svaki predmet. Za svaki predmet se korištenjem prosljeđene klase za određivanje interaktivnosti postavlja interaktivnost tipke. Trenutno postoje tri strategije određivanja interaktivnosti koje ovise o situaciji u kojoj se predmeti prikazuju. Strategija za kupnju provjerava cijenu predmeta i ako igrač nema dovoljno zlata, postavlja predmet kao neinteraktivan. Strategija za borbu postavlja interaktivno samo predmete

koji pomažu igraču, dok se zadnja strategija koristi u prodaji predmeta i prikazu predmeta u pauzi, postavlja sve predmete kao interaktivne.

```
public static void SetUpItems(GameObject contentWrapper, List<ItemObject>
    items, Button itemButtonPrefab, IInteractableStrategy
    interactableStrategy, Button backButton = null)
{
    foreach (Transform child in contentWrapper.transform)
    {
        Transform.Destroy(child.gameObject);
    }

    contentWrapper.transform.DetachChildren(); // Destroy is not fast enough
         so we detach the children while they are being destroyed to avoid
        problems

    foreach (var item in items)
    {
        Button button = Object.Instantiate(itemButtonPrefab, contentWrapper.
            transform);
        button.gameObject.SetActive(true);
        ItemButton itemButton = button.GetComponent<ItemButton>();

        itemButton.Init(item);

        interactableStrategy.SetInteractable(itemButton);

        if (button.interactable == true)
        {
            continue;
        }

        Button button1 = button.GetComponentsInChildren<Button>(true)[1];
        button1.gameObject.SetActive(true);
        button1.interactable = true;
        ItemButton itemButton1 = button1.GetComponent<ItemButton>();

        itemButton1.Init(item);
    }

    SetNavigation(contentWrapper, backButton);

    backButton?.Select();
    Button buttonToSelect = contentWrapper.GetComponentsInChildren<Button>()
        .FirstOrDefault(b => b.interactable == true);
    buttonToSelect?.Select();
}
private static void SetNavigation(GameObject contentWrapper, Button
    backButton = null)
{

    for (int i = 0; i < contentWrapper.transform.childCount; i++)
    {
```

```
Transform child = contentWrapper.transform.GetChild(i);
Button button = child.GetComponentsInChildren<Button>().First(b => b
    .interactable == true);

Navigation navigation = button.navigation;
navigation.mode = Navigation.Mode.Explicit;
Button navButton;

if (i == 0 && backButton != null)
{
    Navigation backNavigation = backButton.navigation;
    navigation.selectOnLeft = backButton;
    backNavigation.selectOnRight = button;
    backButton.navigation = backNavigation;

}

if (i == contentWrapper.transform.childCount - 1 && backButton !=
    null)
{
    Navigation backNavigation = backButton.navigation;
    navigation.selectOnRight = backButton;
    backNavigation.selectOnLeft = button;
    backButton.navigation = backNavigation;
}

if (i - 1 >= 0)
{
    navButton = contentWrapper.transform.GetChild(i - 1).
        GetComponentsInChildren<Button>().First(b => b.interactable
        == true);
    navigation.selectOnLeft = navButton;
}

if (i - 2 >= 0)
{
    navButton = contentWrapper.transform.GetChild(i - 2).
        GetComponentsInChildren<Button>().First(b => b.interactable
        == true);
    navigation.selectOnUp = navButton;
}

if (i + 1 <= contentWrapper.transform.childCount - 1)
{
    navButton = contentWrapper.transform.GetChild(i + 1).
        GetComponentsInChildren<Button>().First(b => b.interactable
        == true);
    navigation.selectOnRight = navButton;
}

if (i + 2 <= contentWrapper.transform.childCount - 1)
{
    navButton = contentWrapper.transform.GetChild(i + 2).
```

```
                GetComponentsInChildren<Button>().First(b => b.interactable
                == true);
            navigation.selectOnDown = navButton;
        }

        button.navigation = navigation;
    }
}
```

Nakon što su svi predmeti instancirani, potrebno je postaviti navigaciju među njima. Budući da su predmeti raspoređeni u dva stupca, navigacija se postavlja tako da prvi i zadnji predmet pokazuju na tipku za povratak. Za sve tipke provjerava se postoji li tipka prije ili nakon trenutne koja se obrađuje. Ukoliko postoje, postavlja se navigacija na sljedeći način: tipka koja je dva mjesta prije trenutne postavlja se na pritisak "gore", tipka koja je jedno mjesto prije postavlja se na pritisak tipke "lijevo", tipka nakon postavlja se na pritisak "desno", a tipka koja je dva mjesta nakon postavlja se na pritisak "dolje".

Prilikom postavljanja navigacije za svaku tipku, korištena je Unityjeva metoda "GetComponentsInChildren" kako bi se pronašla prva interaktivna tipka. Na taj način samo ta tipka sudjeluje u navigaciji. Dakle, ako igrač nema dovoljno zlata za kupnju određenog predmeta, glavna tipka će biti prikazana sivom bojom, ali neće sudjelovati u navigaciji. Umjesto toga, dohvaća se dijete tipke koje je interaktivno kako bi se odabirom te tipke promijenila boja i igraču prikazalo da je odabrana.

Prilikom odabira jednog od predmeta podiže se event odabira predmeta. Ovisno o scenama i potrebama moguće je imati različite funkcionalnosti koje se izvršavaju tada. Na trenutnoj sceni nakon što se odabere predmet mijenja se tekst koji prestavlja opis predmeta, te klasa "PlayerStats" koja je zadužena za prikaz podataka likova izvršava provjeru o kakvom se predmetu radi. Ukoliko je predmet tipa opreme onda se prikazuje promjena u zdravlju i šteti koju bi pojedini lik imao ukoliko se opremi sa odabranom opremom.

```
public void OnItemButtonSelect(ItemObject item)
{
    if (item.type == ItemObject.ItemType.Default)
    {
        return;
    }

    EquipmentItem equipmentItem = (EquipmentItem)item;
    EquipmentItem equipedItem = partyMember.GetEquipedItem(equipmentItem.
        EquipmentType);

    ShowStatDifferences(equipmentItem, equipedItem);
}

public void ShowStatDifferences(EquipmentItem newItem, EquipmentItem oldItem
    )
{
    int hpChangeValue = oldItem != null ? newItem.Hp - oldItem.Hp : newItem.
        Hp;
```

```
        if (hpChangeValue != 0)
        {
            hpChange.SetActive(true);
            hpChange.GetComponent<TMP_Text>().text = hpChangeValue.ToString();
            Image hpArrow = hpChange.GetComponentInChildren<Image>();
            hpArrow.color = hpChangeValue > 0 ? positiveStatChangeColor :
                negativeStatChangeColor;
            hpArrow.transform.rotation = hpChangeValue > 0 ? Quaternion.Euler
                (180, 0, 0) : Quaternion.Euler(0, 0, 0);
        }

        int dmgChangeValue = oldItem != null ? newItem.Dmg - oldItem.Dmg :
            newItem.Dmg;
        if (dmgChangeValue != 0)
        {
            dmgChange.GetComponent<TMP_Text>().text = dmgChangeValue.ToString();
            dmgChange.SetActive(true);
            Image dmgArrow = dmgChange.GetComponentInChildren<Image>();
            dmgArrow.color = dmgChangeValue > 0 ? positiveStatChangeColor :
                negativeStatChangeColor;
            dmgArrow.transform.rotation = dmgChangeValue > 0 ? Quaternion.Euler
                (180, 0, 0) : Quaternion.Euler(0, 0, 0);
        }
    }
```

Prilikom pritiska na tipku, generira se događaj koji obavještava sve slušatelje da je tipka pritisnuta, a kao parametar tog događaja šalje se predmet koji predstavlja pritisnutu tipku. Tijekom procesa kupnje, prvo se uklanja događaj za pomicanje odabira s predmeta kako bi se spriječilo ponovno pokretanje tog događaja prilikom odabira tipke za potvrdu kupnje. Na taj način se sprječava ne prikazivanje opisa predmeta i promjena na likovima u slučaju kada je predmet oprema. Kada igrač pritisne tipku za kupnju, pojavljuje se izbornik koji sadrži gumbe za potvrdu i odustajanje od kupnje. Ako igrač kupi određeni predmet, ponovno se postavljaju predmeti kako bi se osigurala ponovna interaktivnost tipki, uzimajući u obzir trenutno stanje zlata igrača. U slučaju da igrač odustane od kupnje određenog predmeta, izbornik se deaktivira, a predmet koji je bio odabran ponovno postaje trenutno odabrana tipka.

```
    private IEnumerator OnItemBuy(ItemObject item)
    {
        while (!confirmAction)
        {
            yield return null;
        }

        if (item.price <= Inventory.Inventory.Instance.GoldAmount)
        {
            Inventory.Inventory.Instance.GoldAmount -= item.price;
            Inventory.Inventory.Instance.AddItem(item);
        }

        confirmAction = false;
        SetUpMerchandise();
    }
```

4.3. Pauza

U sceni pauze, osim već opisanih koncepta, postoje neki noviteti. Scena se sastoji od tri dijela: u lijevom prozoru su prikazani igračevi likovi i njihovi podaci, na dnu su tipke koje predstavljaju funkcionalnosti implementirane na pauzi, a desno se pojavljuje prozor ovisno o odabranoj tipci.

Podaci o likovima obuhvaćaju informacije poput zdravlja, mane, razine, imena, napada i sakupljenog iskustva. Osim toga, sam objekt koji prikazuje te podatke sadrži komponentu "Button" kako bi se prilikom korištenja predmeta mogao odabrati lik na kojem se želi upotrijebiti predmet. Dodatno, osim tipke za odabir lika, klasa za prikaz podataka o likovima čeka događaj odabira predmeta kako bi prikazala promjene na liku ako se oprema koristi. Likovi također imaju događaj koji se aktivira kada ih se odabere, omogućujući prikaz opremljenih predmeta koje lik posjeduje.

Objekt predmeta sadrži metodu "Use" koja prima odabranog lika. Ukoliko se radi o "DefaultItemObject" predemtu, poziva se metoda "UseItem" nad likom te se liku liječi zdravlje i mana ovisno o predmetu. Ukoliko se radi o "EquipmentItemObject" predmetu se poziva metoda "Equip" nad likom koja služi za postavljanje predmeta na lika.

```
public void Equip(EquipmentItem newEquip)
{
    if (EquipedItems.ContainsKey(newEquip.EquipmentType.ToString()))
    {
        Attack -= EquipedItems[newEquip.EquipmentType.ToString()].Dmg;

        var itemHp = EquipedItems[newEquip.EquipmentType.ToString()].Hp;
        maxHp -= itemHp;
        Hp -= itemHp;

        var previousEquipedItem = EquipedItems[newEquip.EquipmentType.
            ToString()];
        EquipedItems.Remove(newEquip.EquipmentType.ToString());
        Inventory.Inventory.Instance.AddItem(previousEquipedItem);
    }

    EquipedItems.Add(newEquip.EquipmentType.ToString(), newEquip);
    Attack += newEquip.Dmg;
    maxHp += newEquip.Hp;
    Hp += newEquip.Hp;

    Inventory.Inventory.Instance.RemoveItem(newEquip);
}
```

Desni dio scene sastoji se od dva različita prozora koji se aktiviraju ovisno o tome koju tipku igrač pritisne. Prvi prozor je prozor za prikaz predmeta. Funkcionalnost je gotovo ista kao tokom kupnje, koja je već opisana. Jedina razlika je što prilikom pritiska na tipku predmeta nema potvrdnog izbornika, već igrač odabire svog lika i može odabrati na kojem liku želi iskoristiti ili opremiti predmet.

Slika 27: Prikaz predmeta na pauzi (autorski rad)

Drugi prozor je prozor koji prikazuje opremljene predmete za odabranog lika. Nakon odabira lika, podiže se događaj koji označava da je lik odabran, te se na osnovu odabranog lika ažuriraju prikazani podaci.

Slika 28: Prikaz opreme lika na pauzi (autorski rad)

```
public void ShowPlayerEquipment1()
{
    SelectPartyMember();

    StartCoroutine(ShowPlayerEquipment());
}

private IEnumerator ShowPlayerEquipment()
{
    while (selectedPartyMember == null)
    {
        yield return null;
```

```
        }

        equipmentPanel.SetActive(true);
        headEquipment.text = selectedPartyMember.GetEquipedItem(EquipmentItem.
            EquipType.Head)?.itemName;
        chestEquipment.text = selectedPartyMember.GetEquipedItem(EquipmentItem.
            EquipType.Chest)?.itemName;
        legEquipment.text = selectedPartyMember.GetEquipedItem(EquipmentItem.
            EquipType.Legs)?.itemName;
        handEquipment.text = selectedPartyMember.GetEquipedItem(EquipmentItem.
            EquipType.Hands)?.itemName;
    }
```

Na dnu se nalaze tipke koje služe za prikaz tih prozora, poput onih za prikaz podataka o likovima i njihovih predmeta. Osim njih, tu su još tipke za povratak u glavni izbornik, kao i tipke za spremanje i učitavanje igre.

4.4. Spremanje i učitavanje sprmeljene igre

Kako je već spomenuto, spremanje igre se izvodi korištenjem obrasca dizajna "composite". Pomoću primjera spremanja igre prikazat će se proces prolaska kroz strukturu i način na koji se podaci spremaju u datoteku. Klasa "SaveLoadData" preuzima odgovornost za spremanje i čitanje podataka.

```
    private SaveData CreateSaveGame()
    {
        SaveData save = new SaveData();

        GameManagement.Instance.Save(ref save);

        return save;
    }

    public void SaveGame()
    {
        SaveData save = CreateSaveGame();
        BinaryFormatter bf = new();

        FileStream file = File.Create(Application.persistentDataPath + "/gamesave.
            save");
        bf.Serialize(file, save);
        file.Close();
    }
```

U prikazanom kodu vidljivo je pozivanje metode "Save" na instanci objekta "GameManagement", nakon čega se pomoću binarnog formatera (BinaryFormatter) podaci spremaju u datoteku. Prilikom prikupljanja podataka za spremanje koristi se klasa "SaveData", koja sadrži sve relevantne podatke za igru. Ova klasa je strukturirana u tri glavne kategorije: "PartyData" za informacije o igračevim likovima, "InventoryData" za podatke o predmetima koje igrač posjeduje, te "GameData" za informacije o trenutnom stanju igre, kao što su trenutna scena i slično.

```
public void Save(ref SaveData saveData)
{
    foreach (ISaveable saveable in saveables)
    {
        saveable.Save(ref saveData);
    }

    saveData.gameData.enemiesKilled = enemiesKilled;
    saveData.gameData.time = timePLayed + Time.realtimeSinceStartup;
    GameObject player = GameObject.FindGameObjectWithTag("Player");
    saveData.gameData.playerPositionX = player.transform.position.x;
    saveData.gameData.playerPositionY = player.transform.position.y;
    saveData.gameData.scene = SceneManager.Instance.GetMainSceneName();
    saveData.gameData.interactableIdsToNotRender =
        interactableIdsToNotRender;
}
```

Ova metoda poziva metodu "Save" nad strukturom "saveable" koja predstavlja kompozitnu strukturu. Nadalje, postavlja opće podatke o igri u klasu za spremanje. Podaci za koje je "GameManagement" klasa zadužena uključuju: proteklo vrijeme igranja, broj pobijeđenih protivnika, trenutnu scenu igrača, poziciju na sceni te listu objekata tipa "Destroyable" s kojima je igrač već interaktirao i koji se ne bi trebali prikazivati pri učitavanju scene na kojoj se nalaze.

Metoda "Save" u klasi "Party" prolazi kroz listu likova i za svakog stvara instancu objekta "PartyMemberData", koju dodaje u listu podataka o igračevim likovima. Klasa "PartyMemberData" sadrži sve potrebne podatke o pojedinom liku, a prilikom kreacije objekta, njezin konstruktor prima instancu "PartyMember" klase kako bi preuzeo sve relevantne informacije o liku. Vještine i predmeti koje lik posjeduje spremaju se kao liste imena tih objekata. Prilikom učitavanja, imena tih objekata se koriste za pronalazak i učitavanje odgovarajućeg scriptable objecta koji predstavlja taj objekt.

```
public void Save(ref SaveData saveData)
{
    PartyMemberData partyMemberData = new(this);

    saveData.partyData.partyMembersData.Add(partyMemberData);
}

public PartyMemberData(PartyMember partyMember)
{
    stats.name = partyMember.Name;
    stats.attack = partyMember.Attack;
    stats.mana = partyMember.Mana;
    stats.hp = partyMember.Hp;
    stats.exp = partyMember.Exp;
    stats.lvl = partyMember.Lvl;
    maxHp = partyMember.MaxHp;
    maxMana = partyMember.MaxMana;
    expForLvlUp = partyMember.ExpForLvlUp;
    className = partyMember.PlayerClass.name;

    foreach (SkillObject skill in partyMember.Skills)
```

```
            {
                skills.Add(skill.name);
            }

            foreach (var equipmentItemequip in partyMember.EquipedItems)
            {
                equipedItems.Add(equipmentItemequip.Value.itemName);
            }
        }
```

Slično kao podaci o likovima, i podaci o igračevim predmetima se spremaju. "InventoryData" klasa sadrži podatke o igračevom zlatu, te listu objekata klase "InventoryItemData". Prilikom spremanja podataka o predmetima, instancira se nova instanca te klase koja u konstruktor prima ime predmeta i količinu koju igrač posjeduje.

```
        public void Save(ref SaveData saveData)
        {
            saveData.playerInventory.gold = goldAmount;

            foreach (InventoryItem item in playerInventory)
            {
                saveData.playerInventory.items.Add(new InventoryItemData(item.Item.
                    name, item.NumberOfItems));
            }
        }
```

Učitavanje igre funkcionira na sličan način kao i spremanje. Metoda "Load" poziva istoimenu metodu nad instancom objekta "GameManagement", koja je zadužena za učitavanje podataka. Ova metoda zatim poziva metodu "Load" nad composite strukturom koja čita spremljene podatke i instancira potrebne objekte koji ih predstavljaju. Glavna razlika u odnosu na spremanje podataka je što se prvo učitava spremljena scena, a zatim se poziva metoda "Load". Na taj način se osigurava da postoji objekt igrača i da mu se može postaviti pozicija na spremljenu vrijednost.

```
    public void LoadGame()
    {
        if (File.Exists(Application.persistentDataPath + "/gamesave.save"))
        {
            BinaryFormatter bf = new();
            FileStream file = File.Open(Application.persistentDataPath + "/gamesave.
                save", FileMode.Open);
            SaveData saveData = (SaveData)bf.Deserialize(file);
            file.Close();

            this.saveData = saveData;

            SceneChange.onSignal += Load;

            SceneManager.Instance.LoadScene(saveData.gameData.scene);
        }
    }
```

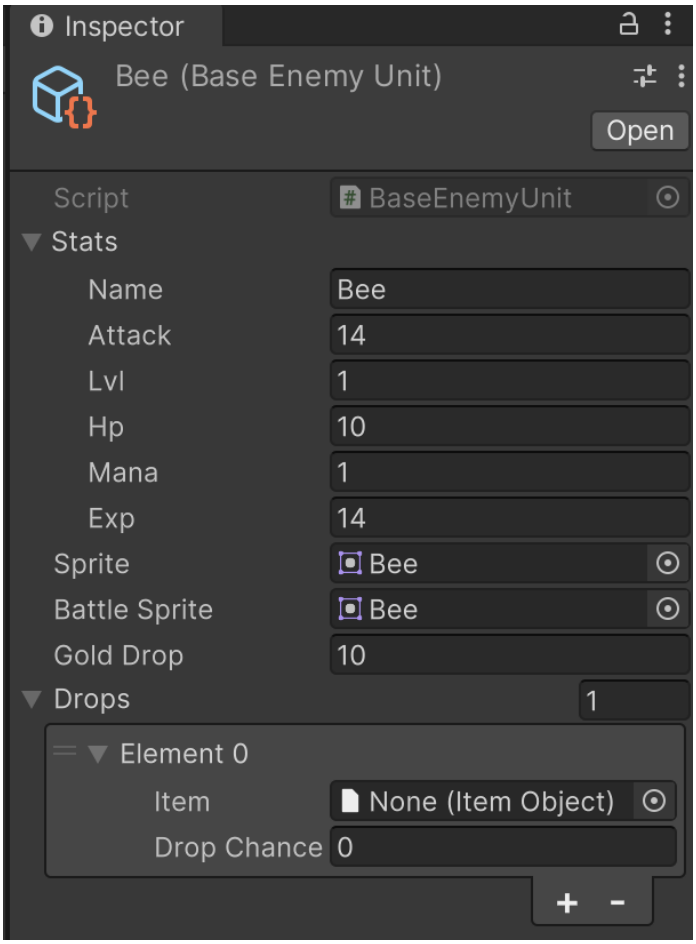
```
private void Load()
{
    GameManagement.Instance.Load(saveData);
}
```

4.5. Zone sa borbama

Borbe u igri funkcioniraju na osnovu prilagodljivih zona. Klase za protivnike, poput "BaseEnemyUnit", nasljeđuju apstraktnu klasu "BaseUnit" i predstavljaju scriptable object protivnika. Slično kao i kod "PartyMember" klasa, sadrže podatke o imenu, zdravlju, šteti i drugima. Osim toga, sadrže i informaciju o tome koliko zlata igrač dobije pobjedom, te listu struktura "DropItem".

Struktura "DropItem" sadrži referencu na objekt klase "ItemObject" i šansu da igrač dobije taj predmet nakon pobjede nad tim protivnikom. Prilikom pobjede u borbi, ovisno o šansi za svaki predmet, igrač može dobiti jedan ili nijedan predmet sa liste.

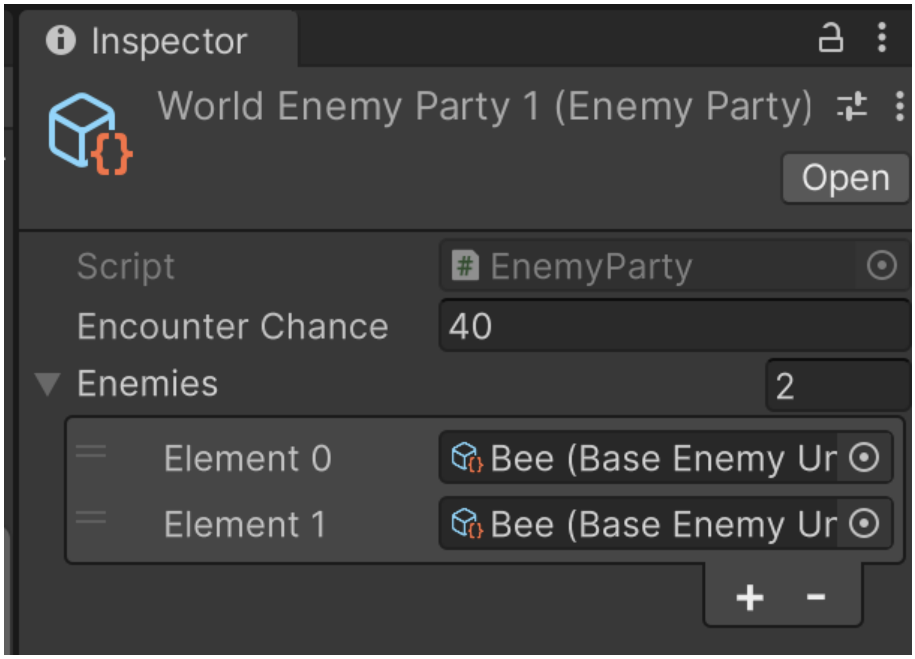
Ovaj sustav omogućava dinamično kreiranje i podešavanje zona s borbama, gdje se protivnici mogu prilagoditi različitim okruženjima i razinama težine igre.



Slika 29: Primjer protivnika (autorski rad)

```
public ItemObject GetDroppedItem()
{
    UnityEngine.Random.InitState((int)Time.time);

    int i = UnityEngine.Random.Range(1, 101);

    foreach (DropItem dropItem in drops)
    {
        if (i <= dropItem.dropChance)
        {
            return dropItem.item;
        }
    }

    return null;
}
```

Klasa "EnemyParty" omogućava prilagođavanje mogućih sukoba s protivnicima u borbi. Ova klasa sadrži listu protivnika i omogućava postavljanje šanse za sukob s tim protivnicima. Na taj način se može prilagoditi koji protivnici i u kojoj kombinaciji mogu napasti igrača, kao i odrediti vjerojatnost sukoba s svakom skupinom protivnika. Ovaj prilagodljivi pristup omogućava raznolikost u izazovima s kojima se igrač može susresti tijekom igre, omogućujući dinamično iskustvo i prilagodljivost prema različitim verzijama igre.



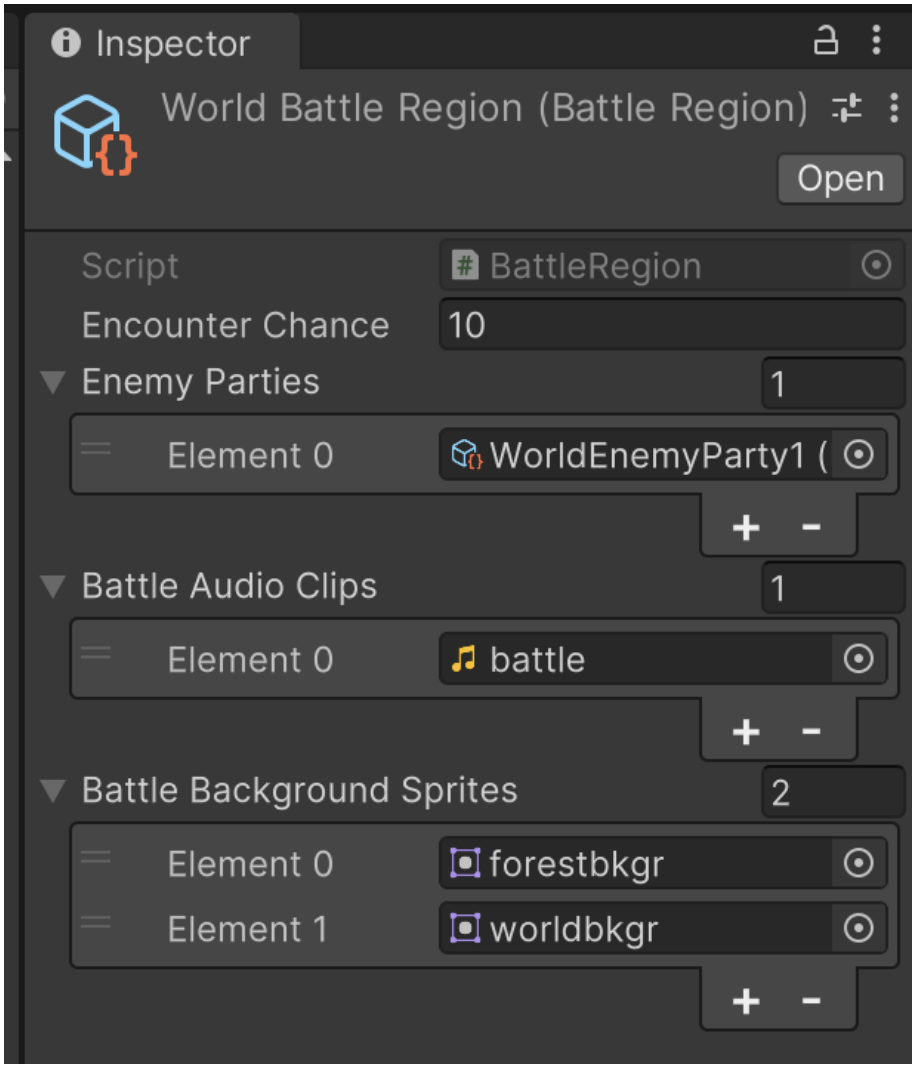
Slika 30: Primjer grupe protivnika (autorski rad)

Klasa "BattleRegion" omogućava prilagodbu borbenih zona u igri. Ova klasa sadrži listu objekata "EnemyParty" i omogućava postavljanje šanse za sukob s protivnicima u toj zoni. Na taj način različite scene svijeta mogu imati različite borbene zone, s većom ili manjom vjerojatnošću sukoba. Osim toga, klasa sadrži listu zvučnih zapisa koji se mogu reproducirati tijekom borbi u toj zoni, kao i listu pozadina koje se mogu koristiti kao pozadina u borbi. Zvučni

zapis i pozadina se nasumično biraju za svaku borbu kako bi se postigao dojam dinamičnosti.

Klasa "RegionEncounterController" prima obavijesti o kretanju igrača i kontrolira učestalost borbi u određenoj zoni. Ova skripta sadrži podatke o minimalnom vremenu između borbi i promjeni šanse za sukob u toj zoni. Ove varijable pružaju dodatnu kontrolu nad učestalosti sukoba s protivnicima u igri.

Kada se borba pokrene, koristi se metoda "StartEncounter" u klasi "GameManagement" koja postavlja statičke varijable o protivnicima, zvučnom zapisu i pozadini borbe prije nego što učita scenu za borbu. Ovim se osigurava da su svi potrebni parametri postavljeni prije početka borbe.



Slika 31: Primjer borbene zone (autorski rad)

4.6. Borba

Na slici 32 se vidi izgled scene borbe. Ova scena predstavlja okruženje u kojem se odvija borba između igračevih likova i protivnika. Ovdje su prikazani različiti elementi potrebni za vođenje borbe, uključujući likove, njihove podatke, sučelje i moguće akcije koje igrač može

poduzeti tijekom borbe. Osim toga, može se primijetiti i pozadina koja dodaje atmosferu borbi.

Slika 32: Borba (autorski rad)

Scena se može podijeliti na dva dijela. Na donjem dijelu se nalaze podaci o akterima u borbi. S lijeve strane su podaci o igračevim likovima, dok se s desne strane nalaze podaci o protivnicima. Svaki lik sadrži informacije poput imena, razine, zdravlja i mane, dok se protivnici prikazuju samo s imenom, razinom i zdravljem. Ispod igračevih podataka nalaze se tipke koje predstavljaju različite radnje u borbi. Klasa koja upravlja simulacijom borbe naziva se "BattleController". S obzirom na to da je proces borbe složen, ova klasa sadrži mnoge reference na objekte na sceni koji se neće nabrajati. Umjesto toga, u opisu borbe ćemo ih spominjati i objasniti njihovu svrhu na sceni. Borba se sastoji od 6 stanja: "START", što je početno stanje u kojem se postavlja sama scena; "PLAYERTURN" i "ENEMYTURN", koja su stanja za prikaz trenutnog djelovanja u borbi, bilo da je to igrač ili protivnik, te "WON", "LOST" i "RUN", koja označavaju ishod borbe - pobjedu, poraz ili bijeg iz borbe.

4.6.1. Postavljanje borbe

Postavljaju se zvučni zapis i pozadina borbe, te slijedi učitavanje igračevih likova i protivnika.

```
var i = 0;
foreach (PartyMember partyMember in GameManagement.Instance.Party.
    PartyMembers)
{
    GameObject playerStats = Instantiate(playerStatsPrefab,
        playerStatsParent.transform);
    playerStats.transform.position = new Vector3(playerStats.transform.
        position.x, playerStats.transform.position.y - (playerStats.
        transform.localPosition.y * i), 0);
    PlayerBattleStats playerBattleStats = playerStats.GetComponent<
        PlayerBattleStats>();
    playerBattleStats.Init(partyMember);
```

```
            GameObject playerPosition = Instantiate(playerPositionPrefab,
                playerPositionParent.transform);
            playerPosition.transform.position = new Vector3(playerPosition.
                transform.position.x - (playerPosition.transform.localPosition.x
                 * i), playerPosition.transform.position.y, 0);
            BattleSpawnPosition battleSpawnPosition = playerPosition.
                GetComponent<BattleSpawnPosition>();
            battleSpawnPosition.Init(partyMember);

            battlePositions.Add(battleSpawnPosition);
            turnOrder.Add(battleSpawnPosition);
            i++;
        }
```

Pri učitavanju likova u borbu, prvo se instancira objekt koji sadrži podatke o liku korištenjem prefab "playerStats". Ovaj prefab opremljen je komponentom "PlayerBattleStats" koja pohranjuje osnovne informacije o liku kao što su ime, razina, zdravlje i mana, te referencu na samog lika. Zatim se instancira vizualni objekt lika u borbi koristeći odgovarajući prefab. Ovaj prefab posjeduje komponentu "BattleSpawnPosition" koja uključuje referencu na lika, sliku koja ga predstavlja u borbi, te boje koje označavaju koji lik je trenutno na potezu. Također, ova komponenta uključuje i animator za prikaz animacija u borbi. Nakon postavljanja vizualnih elemenata, bitno je osigurati navigaciju među tipkama. Likovi igrača se prikazuju s desna na lijevo, dok su protivnici s lijeva na desno.

```
for (int i = 0; i < partyBattlePositions.Count; i++)
        {
            var partyMember = partyBattlePositions[i];
            var button = partyMember.GetComponent<Button>();
            Navigation navigation = button.navigation;
            navigation.mode = Navigation.Mode.Explicit;
            Button navButton;

            if (i == 0 && cancelActionButton != null)
            {
                Navigation cancelButtonNavigation = cancelActionButton.
                    navigation;
                cancelButtonNavigation.mode = Navigation.Mode.Explicit;
                navigation.selectOnRight = cancelActionButton;
                cancelButtonNavigation.selectOnLeft = button;
                cancelActionButton.navigation = cancelButtonNavigation;
            }

            if (i == partyBattlePositions.Count - 1 && cancelActionButton !=
                null)
            {
                Navigation cancelButtonNavigation = cancelActionButton.
                    navigation;
                cancelButtonNavigation.mode = Navigation.Mode.Explicit;
                navigation.selectOnLeft = cancelActionButton;
                cancelButtonNavigation.selectOnRight = button;
                backButton.navigation = cancelButtonNavigation;
            }
```

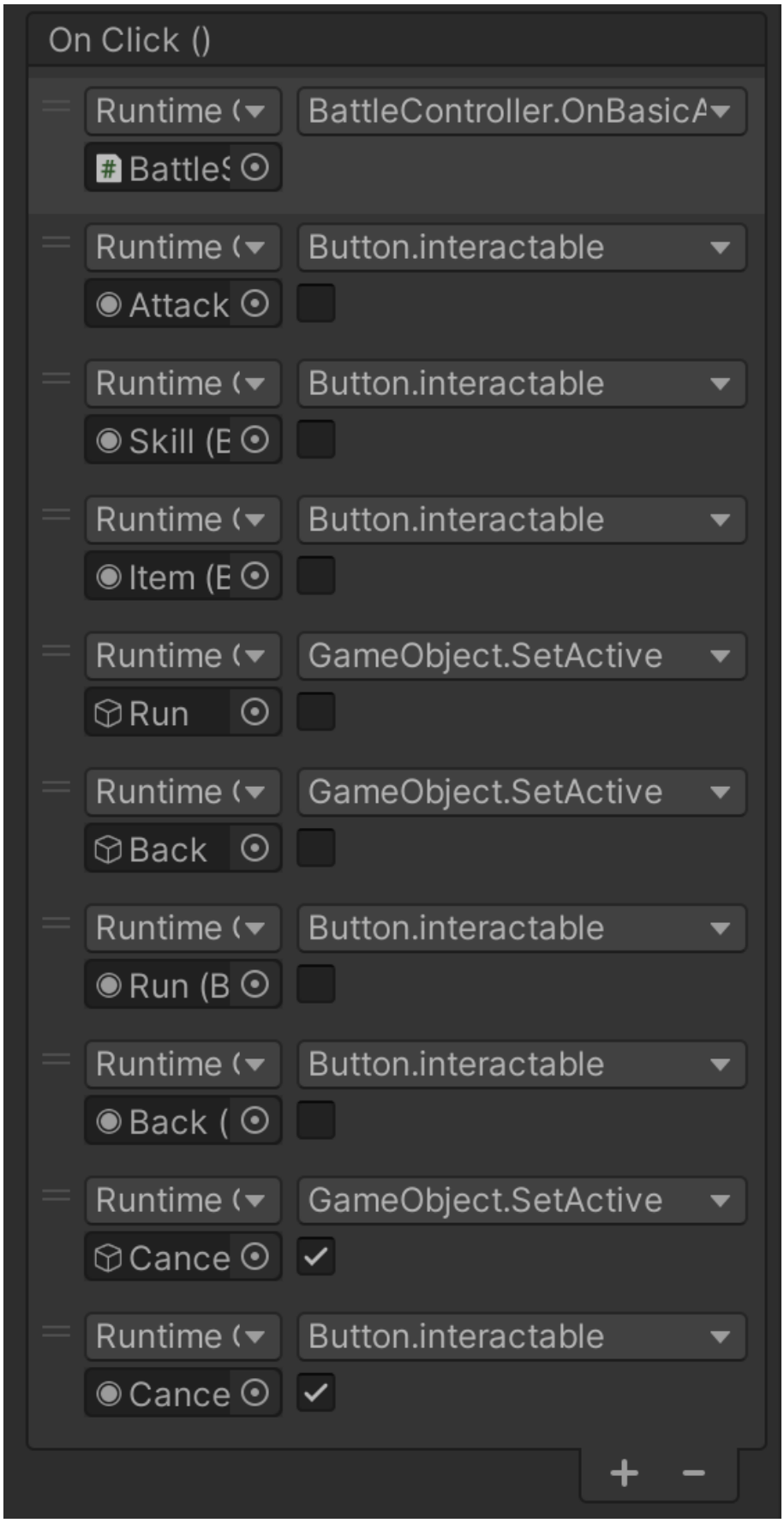
```
        if (i - 1 >= 0)
        {
            navButton = partyBattlePositions[i - 1].GetComponent<Button>();
            navigation.selectOnRight = navButton;
        }

        if (i + 1 <= partyBattlePositions.Count - 1)
        {
            navButton = partyBattlePositions[i + 1].GetComponent<Button>();
            navigation.selectOnLeft = navButton;
        }

        button.navigation = navigation;
    }
```

Navigacija se postavlja tako da likovi igrača međusobno povezuju tako da trenutni lik u nizu pokazuje desno na prethodnog, a lijevo na sljedećeg. Prvi lik u nizu pokazuje desno na tipku za odustajanje od akcije, dok zadnji lik pokazuje lijevo na istu tu tipku. Navigacija za protivnike je slična, osim što su smjerovi navigacije za prethodnog i sljedećeg, kao i za tipku za odustajanje, obrnuti.

Nakon postavljanja navigacije, borba započinje igračevim potezom. Prilikom početka poteza lika, njegova vizualna reprezentacija ističe se bojom koja je postavljena u komponenti "BattleSpawnPosition". Ovo isticanje omogućuje igraču jasno prepoznavanje trenutnog lika koji djeluje u borbi.

4.6.2. Igračev potez

Prilikom početka poteza, lik koji je trenutno na potezu ističe se, a na vrhu ekrana gdje obično piše "TURN" mijenja se tekst kako bi sadržavao ime tog lika. Ovaj dodatni vizualni prikaz jasno ukazuje igraču tko je trenutno na potezu i omogućuje mu lakše praćenje borbe. Kada je na redu igrač, ima nekoliko mogućnosti za akciju, uključujući obični napad, uporabu vještine ili predmeta, te opciju za pokušaj bijega.



Slika 33: Prikaz tipke za obični napad (autorski rad)

Kada igrač odabere obični napad, odnosno pritisne "ATTACK" tipku, podiže se odgovarajući događaj koji označava obični napad igrača. Nakon toga, automatski se odabire prvi protivnik iz liste kao meta napada. Osim toga, postavlja se navigacija tipke za odustajanje kako bi pokazivala na protivnike, omogućujući igraču jednostavno odustajanje od akcije ako to želi.

Korištenjem korutina, sustav čeka igračev odabir protivnika te se zatim poziva korutina u klasi "BattleSpawnPosition" pod nazivom "OnBaseAttacked". Ovoj korutini prosljeđuje se informacija o šteti koju igrač nanosi protivniku. Ovaj korak omogućuje prikazivanje animacija ili efekata povezanih s napadom igrača na odgovarajućem protivniku.

```
public void PlayerBasicAttack()
{
    battlePositions.First(bp => bp.Unit is Enemy).GetComponent<Button>().
        Select();
    SetCancelButtonNavigation();
    coroutineToStop = StartCoroutine(BasicAttackCoroutine());
}

private IEnumerator BasicAttackCoroutine()
{
    while (targetUnit == null)
    {
        yield return null;
    }

    EventSystem.current.SetSelectedGameObject(null);

    var enemyBattlePosition = battlePositions.First(e => e.Unit ==
        targetUnit);

    yield return enemyBattlePosition.OnBaseAttacked(turnTaker.Attack);

    targetUnit = null;
    turnEnded = true;
}
```

Korutina "OnBaseAttacked" prvo postavlja okidač na animatoru lika na "Attack" kako bi se prikazala odgovarajuća animacija koja signalizira da je lik napadnut. Nakon što se animacija izvrši, poziva se metoda "TakeDmg" nad samim objektom lika. U ovoj metodi se smanjuje zdravlje lika za vrijednost štete koju napadač nanosi.

Nakon što se primijeni šteta, podižu se odgovarajući događaji koji označavaju da je lik primio štetu i da se promijenilo njegovo zdravlje. Ukoliko zdravlje lika padne na 0 ili ispod toga, podiže se dodatni događaj koji označava smrt lika. Ovi događaji su ključni za upravljanje stanjem likova u borbi te omogućuju daljnje reakcije i animacije u skladu s promjenama u borbi.

Prilikom primanja štete, nad objektom "CharacterBattleStats" se poziva metoda "OnHpChange", koja ažurira prikaz zdravlja lika u donjem dijelu prozora. Ukoliko je podignut događaj smrti, isti objekt, kao i objekt "BattleSpawnPosition", pozivaju metodu "OnDeath". Ova metoda čeka izvršavanje animacije smrti te nakon toga uništava objekt lika na sceni. Ovaj proces omogućuje glatku animaciju smrti lika te njegovo uklanjanje sa scene nakon završetka animacije.

```
public void OnSkillUsed(SkillButton skillButton)
{
    lastSelectedButton = EventSystem.current.currentSelectedGameObject.
        GetComponent<Button>();
    if (skillButton.skill is SupportSkillObject)
    {
        battlePositions.First(bp => bp.Unit is PartyMember).GetComponent<
            Button>().Select();
    }
    else
    {
        battlePositions.First(bp => bp.Unit is Enemy).GetComponent<Button>()
            .Select();
    }
    SetCancelButtonNavigation();

    coroutineToStop = StartCoroutine(SkillUseCoroutine(skillButton.skill));
}
```

Ukoliko igrač odabere tipku "SKILLS", prikazuje se popis vještina koje lik posjeduje. Prilikom odabira vještine, prema gornjem kodu, odabire se igračev lik ili protivnik, ovisno o tome radi li se o vještini za podršku ili napadačkoj vještini. Osim toga, postavlja se navigacija tipke za odustajanje od akcije. Zatim se započinje korutina koja čeka na igračev odabir mete.

Nakon toga, na isti način kao kod običnog napada, poziva se metoda "OnSkillUsed" nad klasom "BattleSpawnPosition". Ova metoda postavlja okidač animatora na vrijednost imena vještine. Potrebno je napraviti animaciju za svaku vještinu koju želimo animirati i postaviti okidač na isti naziv kao i samu vještinu.

Također, kao kod običnog napada, poziva se metoda "TakeDmg" sa pozitivnom vrijednosti ako je napadačka vještina ili negativnom ako je vještina podrške, kako bi se oduzela ili dodala odgovarajuća količina zdravlja.

Ukoliko igrač odabere akciju za odustajanje od korištenja vještine, odabrana tipka koja predstavlja vještinu postavlja se kao trenutno odabrana tipka.

```
public void OnItemClicked(ItemObject itemObject)
{
    lastSelectedButton = EventSystem.current.currentSelectedGameObject.
        GetComponent<Button>();
    battlePositions.First(bp => bp.Unit is PartyMember).GetComponent<Button
        >().Select();
    SetCancelButtonNavigation();
    coroutineToStop = StartCoroutine(ItemUseCoroutine(itemObject));
}

private IEnumerator ItemUseCoroutine(ItemObject itemObject)
{
    while (targetUnit == null)
    {
        yield return null;
    }
```

```
        var item = Inventory.Inventory.Instance.PlayerInventory.First(i => i.
            Item == itemObject);

        item.Use(targetUnit);
        targetUnit = null;
        turnEnded = true;
    }
```

Ukoliko igrač odabere tipku "ITEMS", prikazuje se lista predmeta koje igrač posjeduje. Za prikazivanje predmeta koristi se već opisan proces postavljanja predmeta, a za interaktivnost tipki predmeta koristi se "BattleStrategy". Prilikom odabira predmeta kojeg igrač želi iskoristiti, odabire se prvi igračev lik kao trenutno odabrana tipka. Zatim se postavlja navigacija tipke za odustajanje od akcije.

Nakon što igrač odabere lika nad kojim želi iskoristiti predmet, poziva se metoda "Use" nad objektom predmeta, pri čemu se proslijeđuje lik nad kojim se predmet koristi. Važno je napomenuti da u borbi igrač može koristiti samo "DefaultItem" predmete, koji služe za liječenje igračevih likova.

```
    public void OnRunButton()
    {
        if (state != BattleState.PLAYERTURN)
            return;

        int rint = Random.Range(0, 3);
        if (rint == 2)
        {
            state = BattleState.RUN;
            StartCoroutine(EndBattle());
        }

        turnEnded = true;
    }
```

Ukoliko igrač odabere tipku "RUN", izvršava se kod koji mu daje šansu od 33% da pobjegne iz borbe.

4.6.3. Protivnički potez

Nakon što igrač završi s potezima svojih likova, na red dolaze protivnici. Prilikom protivničkog poteza, ističe se njihov lik kako bi se prikazalo koji protivnik je na potezu. Slučajnim odabirom se odabire meta napada, a zatim se poziva metoda "PlayTurn" nad klasom "BasicEnemyPlayTurn", koja nasljeđuje klasu "PlayTurn". Na taj način je moguće različitim borbenim zonama dodijeliti različite načine na koje protivnici napadaju igrača, dodati modifikatore na štetu koju nanose ili, ako se u budućnosti implementiraju novi napadi, dodati mogućnost odabira koje napade protivnik koristi. "BasicEnemyPlayTurn" koristi normalan napad za napadanje igrača, nakon čega se poziva metoda "OnBaseAttacked" nad objektom lika.

```
    IEnumerator EnemyTurn(Enemy enemy)
```

```
{
    if (state != BattleState.ENEMYTURN)
    {
        yield break;
    }

    backButton.gameObject.SetActive(false);

    turn.text = "Enemy turn";
    Random.InitState((int)Time.time);

    var partyBattlePositions = battlePositions.FindAll(bp => bp.Unit is
        PartyMember);
    int i = Random.Range(0, partyBattlePositions.Count);
    var partyBattlePosition = partyBattlePositions[i];

    yield return playTurn.PlayTurn(enemy, partyBattlePosition);

    turnEnded = true;
}
```

Prilikom završetka borbe, postoje tri mogućnosti: igrač je izgubio, igrač je pobijedio, ili je igrač pobjegao. U slučaju da je igrač izgubio, mijenja se zvučni zapis i učitava scena poraza. Ukoliko je igrač pobjegao ili pobijedio, postavlja se prethodni zvučni zapis kao trenutni, a zatim se uništava scena borbe i igrač se vraća na prethodnu scenu. U slučaju pobjede, prikazuje se prozor koji obavještava igrača o količini zlata i iskustva koje je osvojio, te o predmetima koje je dobio, ako je dobio neki. Ukoliko je jedan od likova dobio dovoljno iskustva za postizanje novog nivoa, nad objektom lika se poziva metoda "LevelUp", koja pojačava lika.

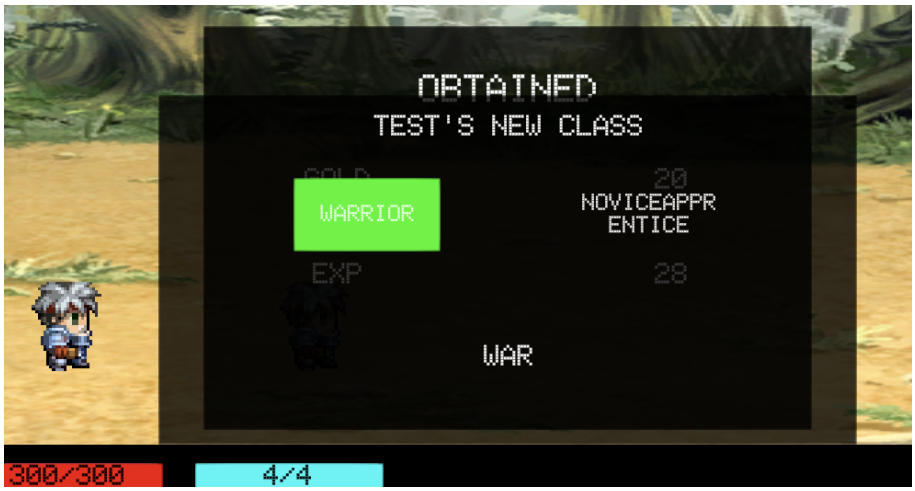
```
private void LevelUp()
{
    Lvl++;
    Attack += Random.Range(2, 5);
    var hp = Random.Range(4, 7);
    maxHp += hp;
    Hp += hp;
    var mana = Random.Range(3, 7);
    maxMana += mana;
    Mana += mana;

    foreach (SkillObject skillObject in playerClass.Skills)
    {
        if (skillObject.lvlReq <= Lvl)
        {
            AddSkill(skillObject);
        }
    }

    SetExpNeededForNextLvl();
}
```

Kada novi nivo lika omogućuje odabir sljedeće klase, prikazuje se prozor s tipkama

koje predstavljaju nove klase koje igrač može odabrati za lika, kao što je prikazano na slici 34. Prilikom odabira klase, provjeravaju se vještine koje igrač može naučiti u toj klasi. Ako igrač zadovoljava nivo potreban za vještinu, ta vještina se dodaje u vještine koje lik posjeduje.

```
private IEnumerator PickUpClass(PartyMember partyMember)
    {
        classPanel.SetActive(true);
        List<ClassButton> classes = new();

        classPanel.GetComponent<ClassPickupPanel>().Title = partyMember.Name + "
            's new class";

        var currentClass = partyMember.PlayerClass;
        var i = 0;
        var availableClassesCount = currentClass.AvailableClasses.Count;
        var classPanelWidth = classPanel.GetComponent<RectTransform>().sizeDelta
            .x;
        var buttonWidth = classButtonPrefab.GetComponent<RectTransform>().
            sizeDelta.x;
        var spaceBetween = (classPanelWidth - buttonWidth *
            availableClassesCount) / (availableClassesCount + 1);

        foreach (ClassObject newClass in currentClass.AvailableClasses)
        {
            GameObject classButtonObject = Instantiate(classButtonPrefab,
                classPanel.transform);
            ClassButton classButton = classButtonObject.GetComponent<ClassButton
                >();

            classButton.transform.position = new Vector3(classPanel.transform.
                position.x + spaceBetween * (i + 1) + (buttonWidth * i),
                classButton.transform.position.y, 0);
            classButton.Class = newClass;
            classes.Add(classButton);
            i++;
        }

        EventSystem.current.SetSelectedGameObject(classes[0].gameObject);

        while (classPickedUp is null)
        {
            yield return null;
        }

        partyMember.AddClass(classPickedUp);
        foreach (SkillObject skillObject in classPickedUp.Skills)
        {
            if (skillObject.lvlReq <= partyMember.Lvl)
            {
                partyMember.AddSkill(skillObject);
            }
        }
```

```
        classPickedUp = null;
        classPanel.SetActive(false);

        //Check for case when multiple lvls are gained so player can choose the
            next class
        if (partyMember.PlayerClass.LvlToPickNextClass <= partyMember.Lvl)
        {
            yield return PickUpClass(partyMember);
        }

        yield return new WaitForEndOfFrame();
    }
```



Slika 34: Odabir klase prilikom pobjede u borbi (autorski rad)

4.7. Pobjeda

Cilj igre je doći do lika "ShadowKinga" na sceni "Forest" i pobijediti ga u borbi. Za prikazivanje tog lika na sceni koristi se klasa "Enemy", koja predstavlja interaktivnog protivnika na mapi. Ova klasa također omogućuje prikazivanje dijaloga s likom. Dijalog ima događaj koji označava početak borbe, a kada se dijalog završi, podiže se događaj koji označava početak borbe.

Nakon što igrač pobijedi "ShadowKinga", podiže se događaj pobjede u borbi, što označava da je potrebno uništiti objekt koji ga predstavlja na mapi. Kada igrač pobijedi u borbi, može pristupiti "Shadow Tombu" i na taj način pobijediti u igri. Na slici 35 prikazana je scena pobjede, gdje se prikazuje koliko je protivnika igrač pobijedio i koliko je vremena proteklo od početka igre.

Slika 35: Scena pobjede (autorski rad)

5. Zaključak

Razvoj igre uloga unutar Unity platforme nije samo tehnički izazov, već i umjetnost stvaranja uronjenog i uzbudljivog svijeta za igrača. Kroz opisane metode, tehnike i alate, ovaj rad istražuje proces kreiranja jedinstvenog iskustva koje će igrači uživati istražujući. Od borbenih mehanika do upravljanja likovima i dijalozima, svaki element je pažljivo razrađen kako bi pružio zanimljivu i uronjavajuću igračku interakciju.

Korištenjem Unity platforme i programskog jezika C#, ostvareni su temelji za dinamičan i privlačan svijet. Integracija različitih sustava, poput borbe, inventara i dijaloga, pruža igračima širok spektar mogućnosti za istraživanje i napredovanje kroz priču. Kroz ovaj proces, naglašena je važnost detaljnog planiranja, implementacije i testiranja svake komponente kako bi se osigurala kvaliteta i funkcionalnost igre.

U konačnici, ovaj rad nije samo tehnički dokument, već priča o procesu stvaranja svijeta punog izazova, avanture i nezaboravnih iskustava za igrače diljem svijeta.

Popis literature

[1] t. F. E. Wikipedia. „History of games.” (bez dat.), adresa: https://en.wikipedia.org/wiki/History_of_games (pogledano 15. 4. 2024.).

[2] J. Howarth. „How Many Gamers Are There? (New 2024 Statistics).” (siječanj 2024.), adresa: https://explodingtopics.com/blog/number-of-gamers (pogledano 15. 4. 2024.).

[3] E. Earnigns. „Largest Overall Prize Pools in Esports.” (bez dat.), adresa: https://www.esportsearnings.com/tournaments (pogledano 15. 4. 2024.).

[4] t. F. E. Wikipedia. „Video game genre.” (bez dat.), adresa: https://en.wikipedia.org/wiki/Video_game_genre (pogledano 15. 4. 2024.).

[5] t. F. E. Wikipedia. „Role-playing video game.” (bez dat.), adresa: https://en.wikipedia.org/wiki/Role-playing_video_game (pogledano 15. 4. 2024.).

[6] t. F. E. Wikipedia. „Unity (game engine).” (bez dat.), adresa: https://en.wikipedia.org/wiki/Unity_(game_engine) (pogledano 15. 4. 2024.).

[7] Unity. „The Hierarchy window.” (bez dat.), adresa: https://docs.unity3d.com/Manual/Hierarchy.html (pogledano 15. 4. 2024.).

[8] Unity. „The Scene view.” (bez dat.), adresa: https://docs.unity3d.com/Manual/UsingTheSceneView.html (pogledano 15. 4. 2024.).

[9] Unity. „The Game view.” (bez dat.), adresa: https://docs.unity3d.com/Manual/GameView.html (pogledano 15. 4. 2024.).

[10] Unity. „The Inspector window.” (bez dat.), adresa: https://docs.unity3d.com/Manual/UsingTheInspector.html (pogledano 15. 4. 2024.).

[11] Unity. „The Project window.” (bez dat.), adresa: https://docs.unity3d.com/Manual/ProjectView.html (pogledano 15. 4. 2024.).

[12] Unity. „Console window.” (bez dat.), adresa: https://docs.unity3d.com/Manual/Console.html (pogledano 15. 4. 2024.).

[13] Unity. „MonoBehaviour.” (bez dat.), adresa: https://docs.unity3d.com/Manual/class-MonoBehaviour.html (pogledano 15. 4. 2024.).

[14] Unity. „Order of execution for event functions.” (bez dat.), adresa: https://docs.unity3d.com/Manual/ExecutionOrder.html (pogledano 15. 4. 2024.).

Popis slika

Popis tablica

1. Prilog 1

Uz ovaj rad je priložen Unity projekt koji sadržava sve komponente korištene u razvoju igre.