SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

SEMINARSKI RAD

OT EJ A - Razvoj 2D igre u Unity game engineu

Jerko Ćurković

Split, Siječanj 2024.

SADRŽAJ

1 UVOD

Videoigre, kao oblik zabave i umjetnosti, prošle su kroz značajan evolucijski put od svojih skromnih početaka do današnjih tehnološki naprednih i vizualno zadivljujućih dostignuća. Različiti žanrovi, platforme i stilovi igara pridonose raznolikosti ovog medija, a među njima 2D videoigre zadržavaju posebno mjesto. 2D videoigre, iako manje tehnički kompleksne od svojih 3D kolega, nude posebnu estetiku i jednostavnost koja ponekad može biti nostalgična, ali istovremeno i inovativna. U 2D svijetu, igrači istražuju ravne razine, koristeći se perspektivom koja im omogućuje fokus na klasične elemente poput platformi, skakanja i rješavanja zagonetki. Njihova jednostavnost i istovremena dubina čine ih privlačnim oblikom digitalnog stvaralaštva, odajući počast klasičnim igrama i istražujući nove inovacije unutar poznatog formata.

Ovaj seminar istražuje ključne aspekte razvoja 2D igre, pružajući uvid u postavljanje projekta, osnove Unityja za 2D igre, skriptiranje u C#, upravljanje inputom, primjenu fizike, grafičke i zvučne elemente. Rad je podijeljen u 5 cjelina. Drugo poglavlje rada opisuje Unity *game engine* kao *cross-platform* i daje uvid u sve njegove bitne karakteristike i funkcionalnosti. Sljedeće poglavlje detaljnije opisuje Microsoft Visual Studio. Četvrto poglavlje je praktični dio koji detaljno opisuje razvoj 2D igre korištenjem Unity *game enginea* kroz osnovne faze razvoja. Zadnje, peto poglavlje donosi kratki zaključak cijelog seminarskog rada. Kroz ovaj pregled, čitatelj će dobiti dublje razumijevanje procesa stvaranja 2D igre i izazova koji prate njezin razvoj, uz naglasak na praktičnom iskustvu kroz primjer implementacije u Unityju i Visual Studiju.

2 UNITY

Unity predstavlja jedan od najmoćnijih i najrasprostranjenijih game enginea u svijetu razvoja videoigara kojeg je razvila tvrtka Unity Technologies 2005. godine. Koristi se za razvoj video igara, za kreiranje simulacija i drugog interaktivnog sadržaja. Game engine je napisan u programskom jeziku C++, što je programski jezik niske razine odličan za upravljanje memorijom. Budući da je *cross-platform game engine,* njegova svestranost omogućava programerima da kreiraju igre za različite platforme, uključujući PC, igraće konzole, mobilne uređaje, VR (*engl. Virtual Reality*) i AR (*engl. Augmented Reality*). Neke od danas izrazito popularnih igrica su izrađene korištenjem Unity *game enginea*. To su: Angry Birds 2, Pokemon Go i Temple Run[1].

2.1 Korisničko sučelje

Unityjevo je sučelje osmišljeno kako bi omogućilo programerima i dizajnerima da jednostavno upravljaju svim aspektima razvojnog procesa. Sučelje je sastavljeno od nekoliko dijelova.

2.1.1 Hierarchy

Hijerarhija prikazuje strukturu objekata u trenutačnoj sceni. Svaki objekt je naveden prema svojem položaju u hijerarhiji, omogućujući jednostavno organiziranje i navigaciju kroz elemente igre. Smještena je u gornjem lijevom kutu[2].

2.1.2 Scene

Scena je najveći prozor u sredini Unity softvera. On vam pruža pogled na trenutačnu razinu, izbornik ili svijet igre s kojim se trenutno radi. Ovaj prozor je mjesto gdje se može slobodno povlačiti, ispuštati, povećavati i smanjivati objekte, manipulirati kamerama i prilagođavati pozicije, rotacije i skale[2].

2.1.3 Game

Ovo je obično skriveno iza prozora Scene i može se pristupiti pritiskom na karticu duž vrha. Prikaz igre pruža pogled na scenu onako kao što bi ga vidio igrač. To znači da pruža istu perspektivu kao i kamera i ne može se pomicati stvari. Ovo je također mjesto gdje se igra izvodi kada se testira[2].

2.1.4 Inspector

Inspector se nalazi skroz desno u korisničkom sučelju i pruža detaljan prikaz odabranog objekta ili resursa. Omogućuje promjenu svojstava i parametara, poput pozicije, rotacije, skale, materijala, komponenata i skripti[2].

2.1.5 Project

Projekt se nalazi na samom dnu korisničkog sučelja i prikazuje sve resurse koji se koristite u projektu. To uključuje teksture, modele, zvukove, skripte i druge datoteke. Struktura ovog prozora odražava organizaciju resursa unutar projekta[2].

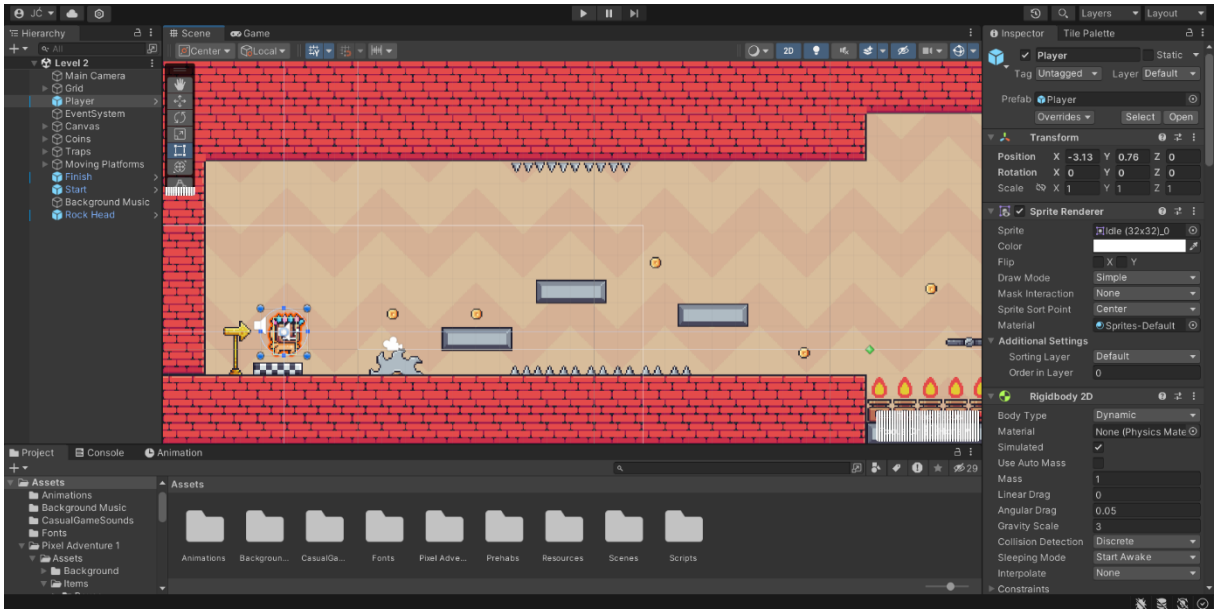
2.1.6 Console

Konzola prikazuje izlaz iz igre, uključujući poruke o pogreškama, upozorenjima i druge informacije. Ovdje programeri mogu pratiti stanje aplikacije i identificirati potencijalne probleme[2].

2.1.7 Animation

Animacijski prozor omogućuje stvaranje i uređivanje animacija. Programeri mogu definirati pokrete, prijelaze između stanja i upravljati animacijama likova i objekata.

Slika 2.1 prikazuje korisničko sučelje Unityja.



Slika 2.1 Korisničko sučelje Unityja

3 MICROSOFT VISUAL STUDIO

Microsoft Visual Studio je integralno razvojno okruženje (*engl. Integrated Development Environment*-IDE) koje je razvio Microsoft Corporation i predstavlja kamen temeljac za programere diljem svijeta. Prva verzija, nazvana Visual Studio 97, predstavljena je 1997. godine s ciljem pružanja sveobuhvatnog okruženja za razvoj aplikacija na platformi Windows[3]. Visual Studio je projektiran kako bi omogućio programerima stvaranje raznovrsnih softverskih rješenja, uključujući desktop, web, mobilne aplikacije, servise i igre. Tijekom godina, doživio je niz evolucijskih promjena i unaprjeđenja. Jedno od ključnih obilježja Visual Studio-a je podrška za različite programerske jezike. C++, C#, Visual Basic, F#, Python i mnogi drugi jezici podržani su unutar ovog razvojnog okruženja, pružajući programerima fleksibilnost u odabiru jezika koji najbolje odgovara njihovim projektima[3].

3.1 Povijest

Microsoft je prvi put izdao Visual Studio 1997. godine, prvi put kombinirajući mnoge svoje alate za programiranje. Visual Studio 97 dolazio je u dvije verzije: Visual Studio Professional i Visual Studio Enterprise. Profesionalno izdanje sadržavalo je tri CD-a, dok je Enterprise izdanje sadržavalo četiri CD-a. Visual Studio 97 bio je pokušaj korištenja istog razvojnog okruženja za više jezika. Visual J++, InterDev i MSDN Library koristili su isto okruženje nazvano Developer Studio. Visual Studio se također prodavao kao paket s odvojenim IDE-ima koji su se koristili za Visual C++, Visual Basic i Visual FoxPro[4]. Microsoft je objavio Visual Studio .NET u veljači 2002. godine. Ključna promjena dolazi s uvođenjem .NET Frameworka, pružajući programerima novi okvir za izradu sofisticiranih aplikacija. Fokus se širi na podršku za web razvoj i XML web servise. Verzije 2005 i 2008 unaprijedile su podršku za Windows Forms, Windows Presentation Foundation te timski rad putem Visual Studio Team Systema. Visual Studio 2010 uveo je Windows Presentation Foundation Designer i proširio podršku za razvoj aplikacija za SharePoint. Izdanje 2012. fokusiralo se na podršku za razvoj aplikacija za Windows 8, predstavljajući nove alate i okoline za programiranje. Verzija 2013. usredotočena na mobilni razvoj unaprijedila je alate za izradu aplikacija za Windows 8.1 i unijela poboljšanja za rad s Xamarinom. U 2015. godini dolazi do ključnog trenutka s uvođenjem Visual Studio Codea, lagane okoline prilagođene raznim platformama. Izdanja 2017. i 2019. naglašavaju brzinu i fleksibilnost razvojnog procesa te donose podršku za .NET Core, Azure i umjetnu inteligenciju kroz IntelliCode. Budući da je za

praktični dio ovog seminarskog rada korištena verzija iz 2019. godine, u idućem se djelu opisuje korisničko sučelje Microsoft Visual Studija 2019.

3.2 Korisničko sučelje

Korisničko sučelje Microsoft Visual Studija 2019 temelji se na pažljivoj analizi potreba programera i fokusira se na stvaranje radnog okruženja koje potiče intuitivan i učinkovit rad. Ključni elementi korisničkog sučelja zajedno tvore jedinstveno iskustvo koje olakšava svaki korak u razvojnom procesu. Slijedi uvid u neke ključne sastavnice.

3.2.1 Izbornik i Traka Alata

Izbornik se nalazi na vrhu glavnog prozora i sadrži niz izbornika poput *File*, *Edit, View, Debug* i *Help*. Ovdje korisnici mogu pristupiti različitim opcijama, alatima i postavkama projekta. Traka Alata je smještena ispod izbornika, sadrži ikone koje predstavljaju često korištene funkcije, omogućujući brzi pristup akcijama poput spremanja projekta, pokretanja debug moda i slično.

3.2.2 Solution Explorer

Ovaj prozor prikazuje strukturu trenutnog projekta, uključujući datoteke, mape i druge resurse. Programeri mogu organizirati, dodavati i brisati resurse izravno iz ovog prozora.

3.2.3 Editor Prozor

Sastoji se od Code Editor i IntelliSense dijela. Code Editor je glavni prozor za pisanje, uređivanje i pregledavanje koda. Ovdje programeri pišu svoje skripte, a Visual Studio pruža napredne značajke poput automatskog dovršavanja i praćenja pogrešaka. IntelliSense je kontekstno osjetljiva značajka koja predviđa i prikazuje moguće opcije dok programer piše kod, što ubrzava proces pisanja.

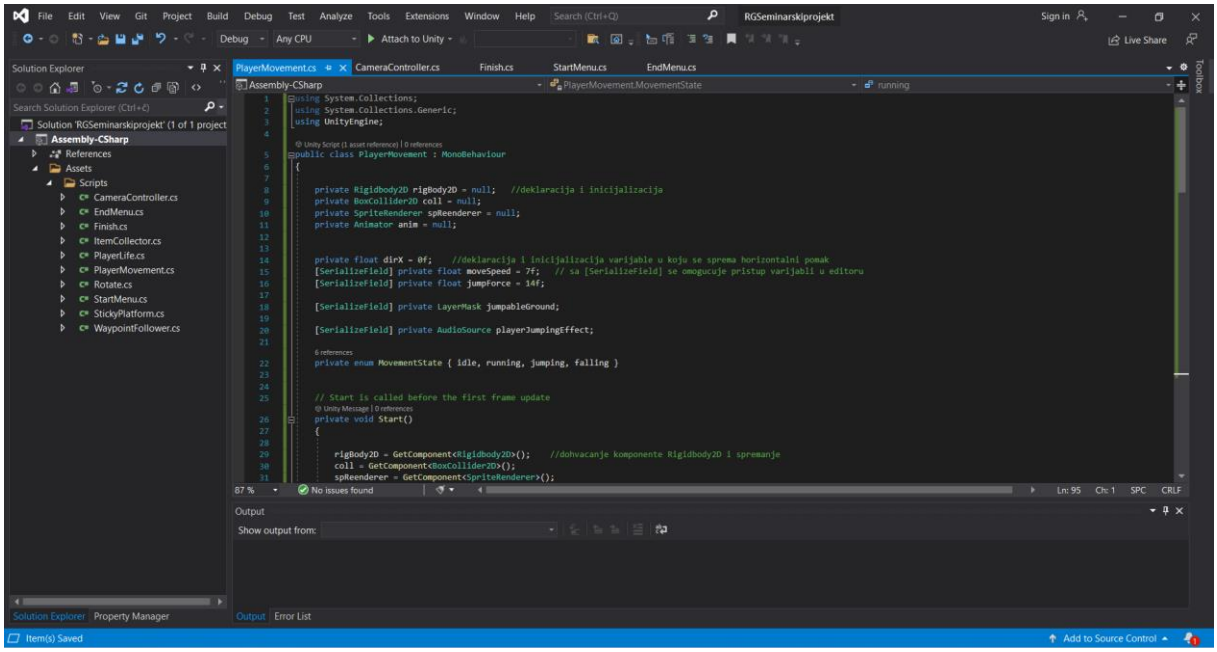
3.2.4 Output Prozor

Ovaj prozor prikazuje rezultate raznih procesa, kao što su rezultati kompilacije, izlaz iz programa ili informacije o greškama. Programeri koriste ovaj prozor za praćenje stanja njihove aplikacije.

3.2.5 Statusna Traka

Smještena je na dnu prozora te pruža informacije o trenutnom stanju aplikacije. Ovdje se mogu prikazivati informacije o trenutnoj radnji (npr., izgradnja projekta), status debugiranja ili druge obavijesti o sustavu. Boje statusne trake u Microsoft Visual Studiju 2019 mogu varirati ovisno o trenutnom stanju i aktivnostima u okruženju razvoja. Zelena boja označava uspješno izvršenu operaciju, crvena označava da je došlo do greške, a žuta/narančasta ukazuje na problem ili obavijest.

Slika 3.1 prikazuje korisničko sučelje Microsoft Visual Studija 2019.



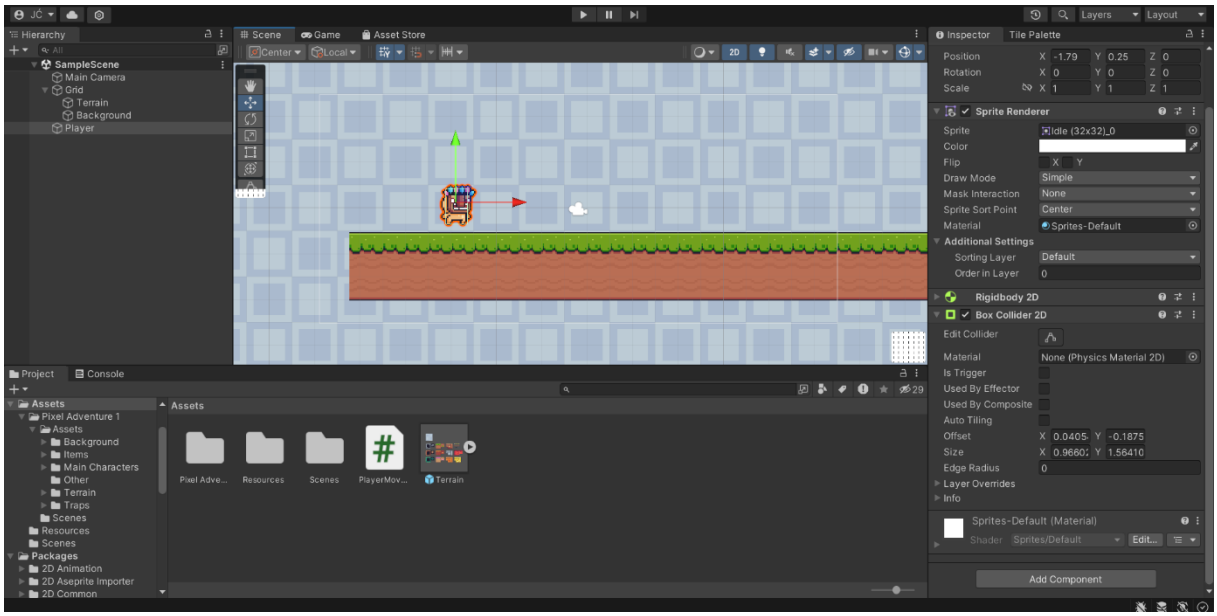
Slika 3.1 Korisničko sučelje Microsoft Visual Studija 2019

4 PRAKTIČNI DIO

Nakon svih prethodnih odlomaka u kojima su stečeni teorijski preduvjeti za izradu 2D videoigre, ovaj dio završnog rada se sada fokusira na izradu konkretne igre. Igra je implementirana korištenjem Unity *game enginea* verzije 2022.3.11f1. Programski kod pisan je u C# programskom jeziku korištenjem Visual Studio 2019 editora. Prva razina igre je izrađena uz pomoć tutoriala s YouTubea uz par sitnih preinaka, dok je druga razina izrađena samostalno. Prvi korak u razvoju igre je kreiranje projekta u Unityju. Inicijalno se podešavaju svojstva same scene. Primjer toga je podešavanje svojstava kamere, zajedno sa kutom gledišta i veličinom prozora igre. Igra je nazvana „OT EJ A“ jer je za pozadinsku glazbu korištena glazba iz crtanog filma „A JE TO“.

4.1 Odabir avatara, terena i pozadine - 1. korak

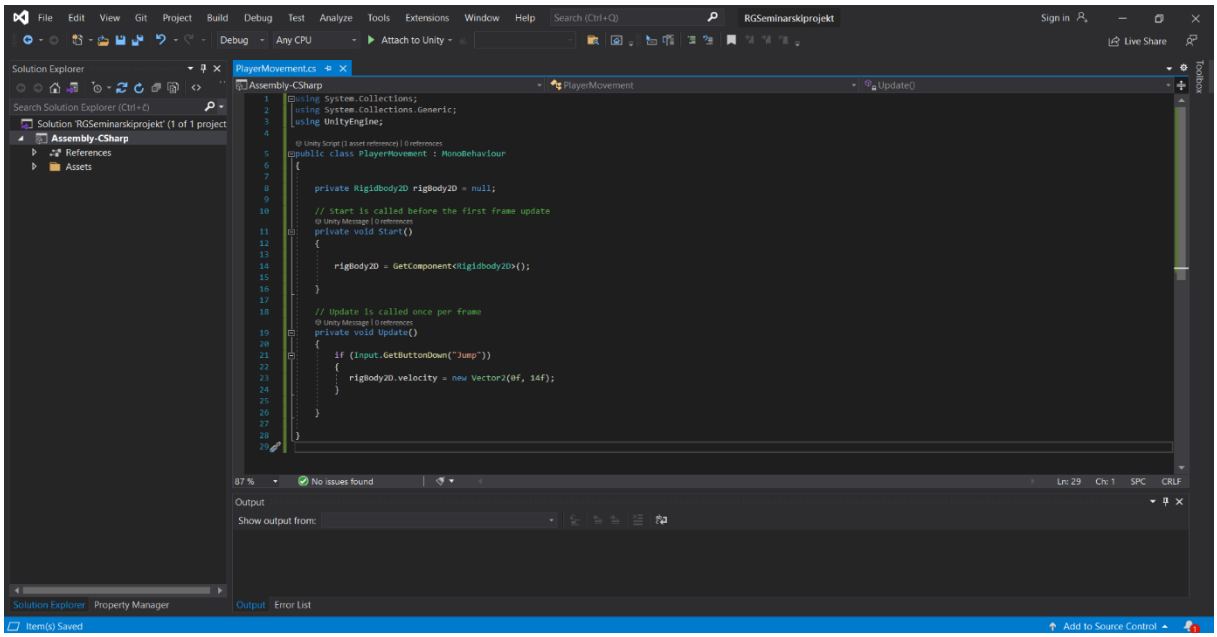
Kako je tema igre izrada pomalo nostalgične 2D igre u pixel svijetu prvo je potrebno s ogromnog Unity Asset Storea preuzeti mapu u kojoj se nalaze sva sredstva koja se koriste u igri (npr. likovi, animacije, pozadine, zamke) ili se to može i samostalno izraditi bez preuzimanja. U ovom slučaju je s Asset Storea preuzet besplatan paket Pixel Adventure 1. Za početak je potrebno kreirati *Tilemap* objekt koji služio kao okvir u koji slažemo dijelove terena i nazvati ga *Terrain*. Isto to je potrebno napraviti za pozadinu i nazvati ga *Background*. Kada je to napravljeno, onda je potrebno dodati još dva nova *Sorting Layera* (*Background* i *Terrain*) uz postojeći *Default* kako bi se moglo manipulirati s iluzijom dubine prostora i svaki od njih pridružiti odgovarajućem objektu (*Default* će se pridružiti objektu *Player* kada bude kreiran). Kada je sve to postavljeno, slijedi ispuna objekata sa odabranim terenom i pozadinom. Da teren bude što realniji, potrebno mu je omogućiti fiziku. To Unity radi preko komponenti. Objektu *Terrain* je potrebno dodati komponentu *Rigidbody 2D* koja dodaje dinamičku fiziku u 2D prostoru, *Tilemap Collider 2D* za dodavanje kolizija na elemente, *Composite Collider 2D* za kombiniranje više kolizija u jednu te *Platform Effector 2D* za kontrolu ponašanja objekata na platformama. U tom je trenutku sve spremno za dodavanje novog 2D objekta pod nazivom *Player*. Njemu je potrebno dodati sliku lika kojeg smo odabrali te dodati fiziku. Dodaje mu se također komponenta *Rigidbody 2D* te još *Box Collider 2D* koji je sličan kao i *Tilemap Collider 2D*. Za sve *Collidere* je potrebno urediti granice kako bi fizika bila što realnija. Slika 4.1 prikazuje stanje igre na kraju ovog koraka.



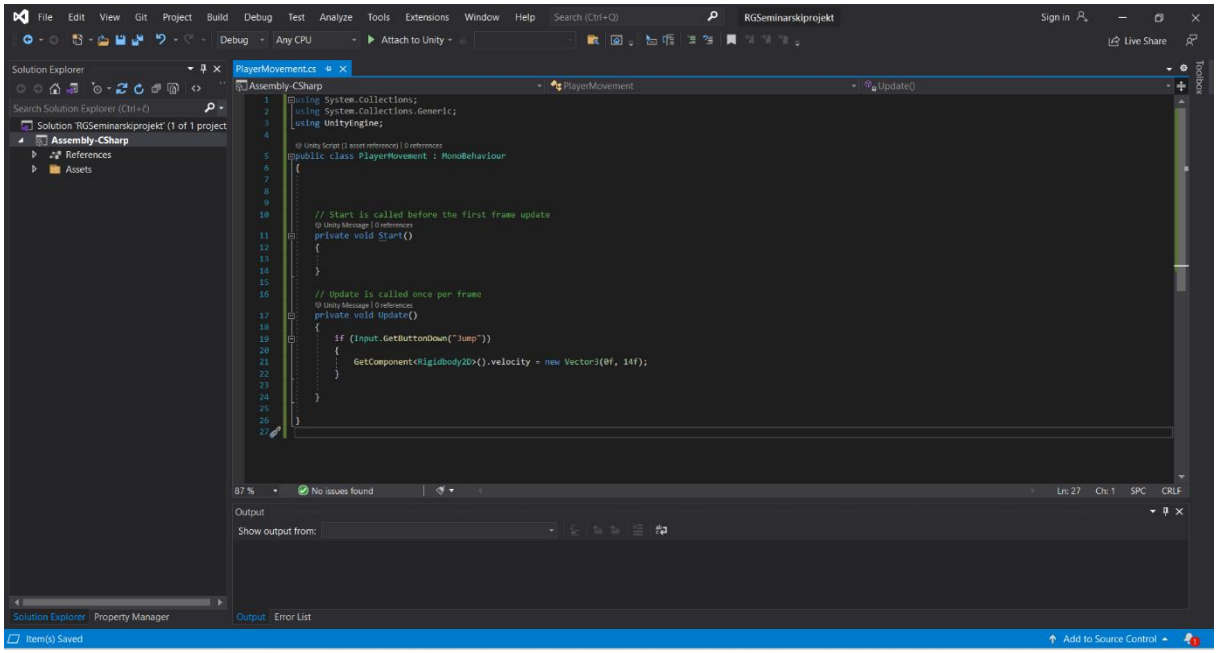
Slika 4.1 Stanje igre nakon 1. koraka

4.2 Kodiranje kretnji – 2. korak

U ovom koraku igra postaje igra jer omogućavanje igraču da upravlja likom je sama srž igre. Da bi to bilo moguće, potrebno je objektu Player dodati novu komponentu. Ta komponenta je C# skripta pod nazivom *PlayerMovement.cs* u kojoj je potrebno kodirati upravljanje likom. U skripti je potrebno prvo dohvatiti komponentu koja je odgovorna za kretanje, a to je *Rigidbody 2D*.Potrebno je omogućiti optimalno dohvaćanje komponente. Slika 4.2 prikazuje optimalno dohvaćanje, dok slika 4.3 to nije.



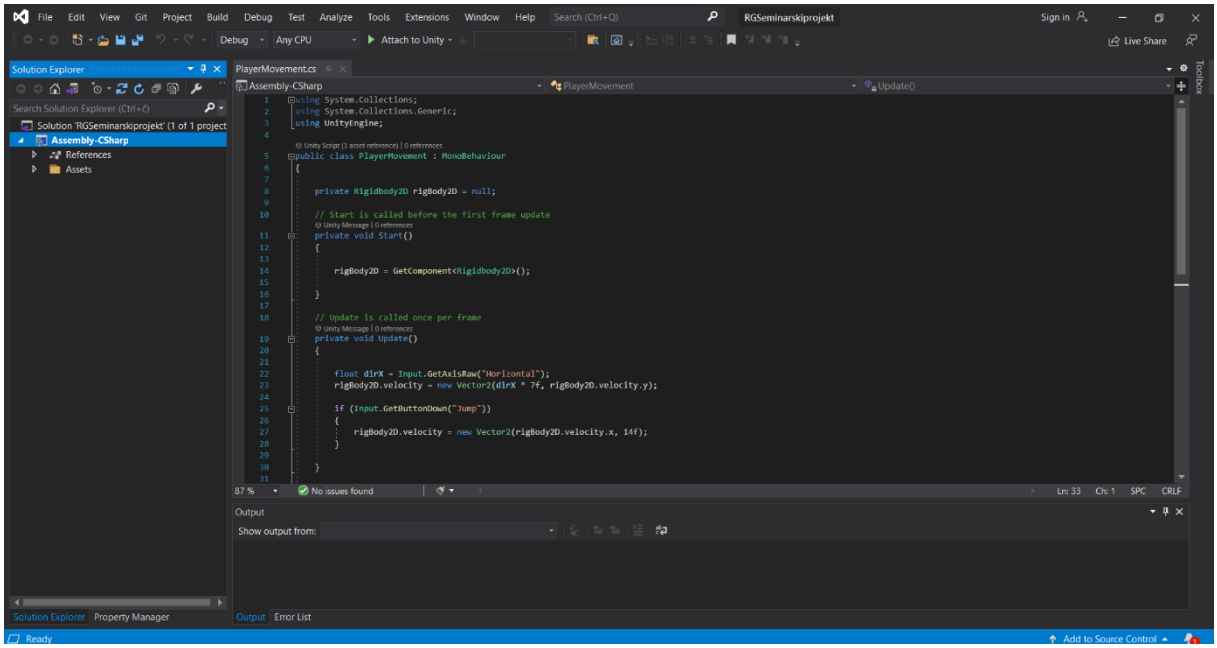
Slika 4.2 Optimalno dohvaćanje komponente



Slika 4.3 Neučinkovito dohvaćanje komponente

Razlika je u tome što se metoda *Update* poziva nakon svakog *framea* dok se *Start* poziva samo nakon prvog *framea*. Zato je učinkovitije komponentu dohvatiti samo jednom nakon prvog *framea*, nego je dohvaćati nakon svakog *framea*. Kada je komponenta dohvaćena i

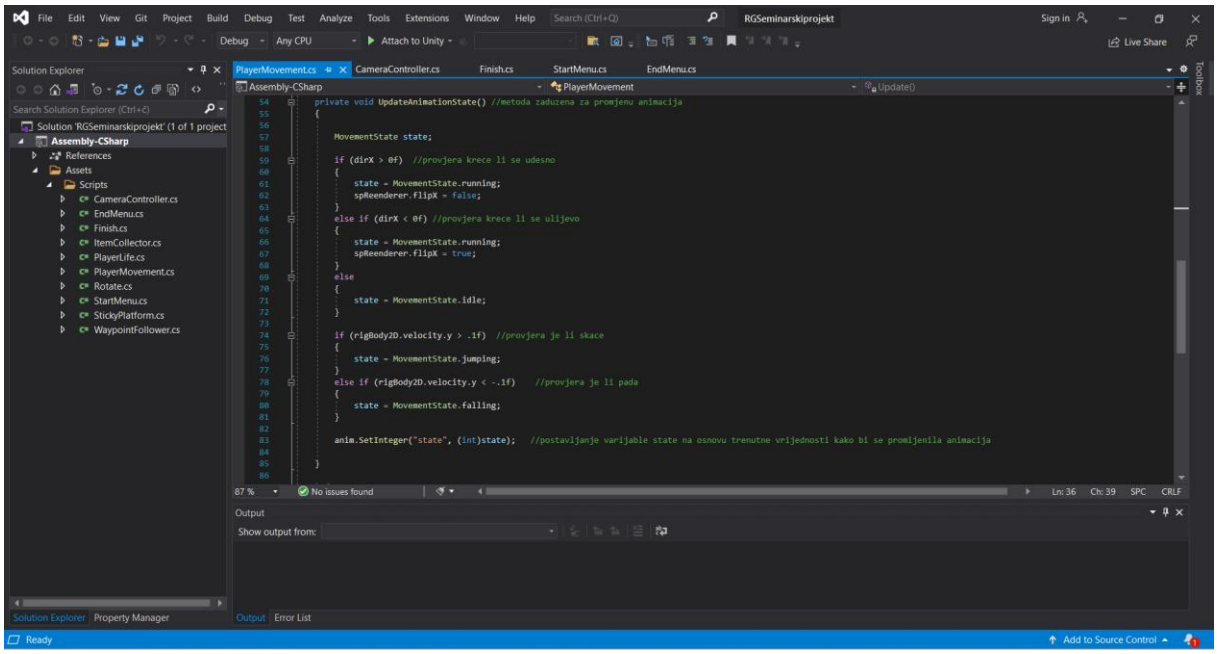
spremljena u objekt *rigBody2D* onda je potrebno manipulirati *velocity* propertyjem ovisno o tipki koju je igrač pritisnuo na tipkovnici. Slika 4.4 prikazuje kod na kraju ovog koraka.



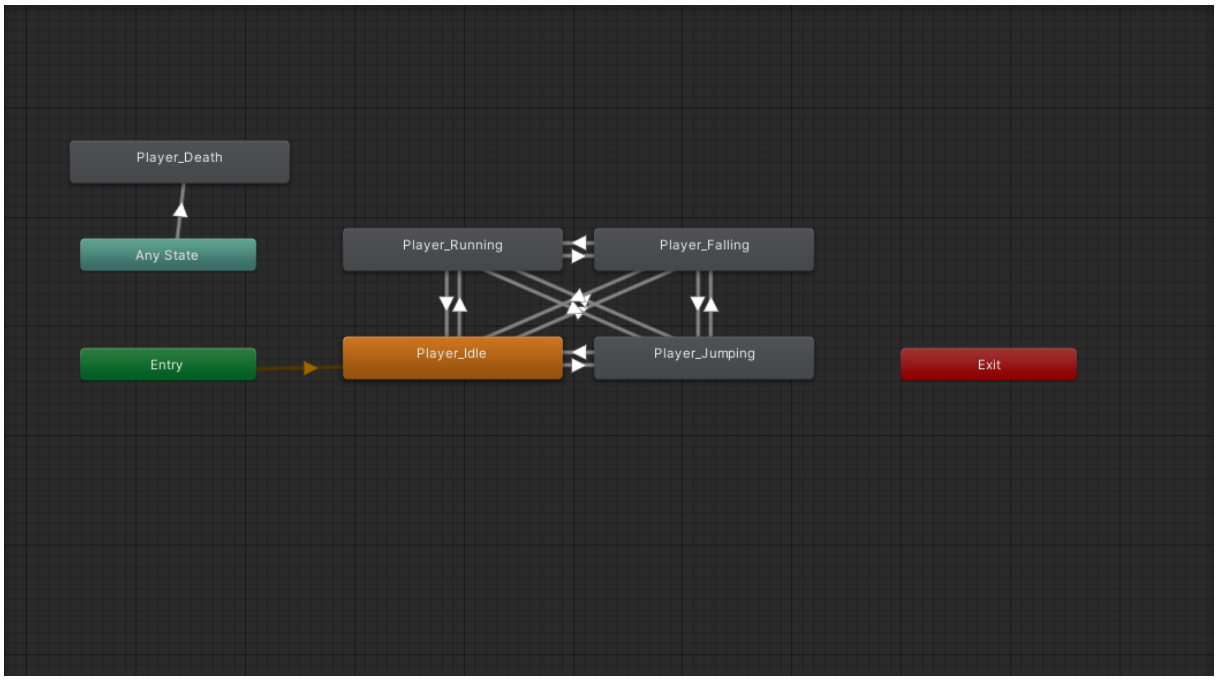
Slika 4.4 Kodiranje kretnji lika

4.3 Animacije – 3. korak

Kako bi igra bila što realnija tako je potrebno napraviti animacije koje se vežu uz neko stanje objekta, najčešće trčanje, skakanje, padanje i smrt. Animacije su zapravo prikazivanje određenog broja sličica u sekundi kako bi se stvorila iluzija pomak u ljudskom oku. Ovisno o vrsti animacije, neke je potrebno staviti u petlju, kao npr. animacija kada lik stoji u mjestu, a neke je potrebno izvršiti samo jednom, kao npr. smrt. U Unityju je prvo potrebno kreirati animaciju i odabrati slike koje će je činiti. Zatim se odabire broj sličica u sekundi kako bi animacija izgledala što realnije. Zatim se animacija doda objektu. Ukoliko objekt ima više animacija, tada je potrebno napraviti više stanja i svakom stanju dodati pripadajuću animaciju i omogućiti prijelaze između stanja. Lik u igri se može naći u stanju da stoji u mjestu, skače, pada, trči ili umire. Kako su stanja i animacije lika vezana uz njegovo kretanje, onda se promjena između njih obavlja u skripti *PlayerMovement.cs*. Slika 4.5 prikazuje promjenu animacija u skripti, a slika 4.6tranziciju između stanja u *Animatoru.*



Slika 4.5 Promjena animacija u skripti

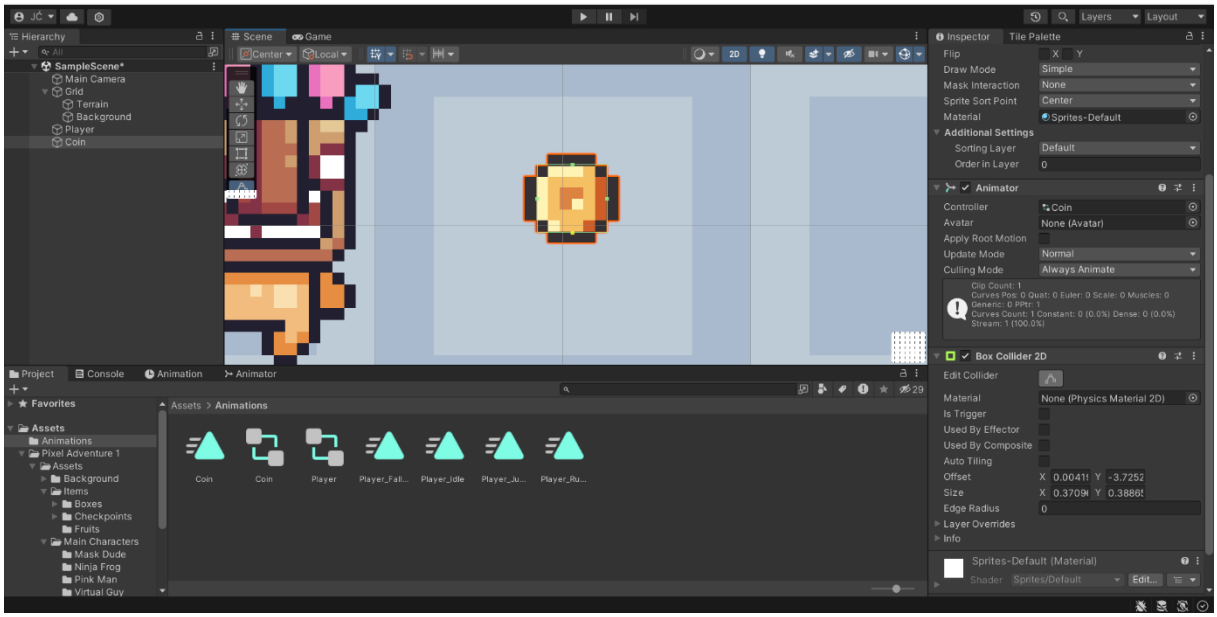


Slika 4.6 Tranzicija između stanja u Animatoru

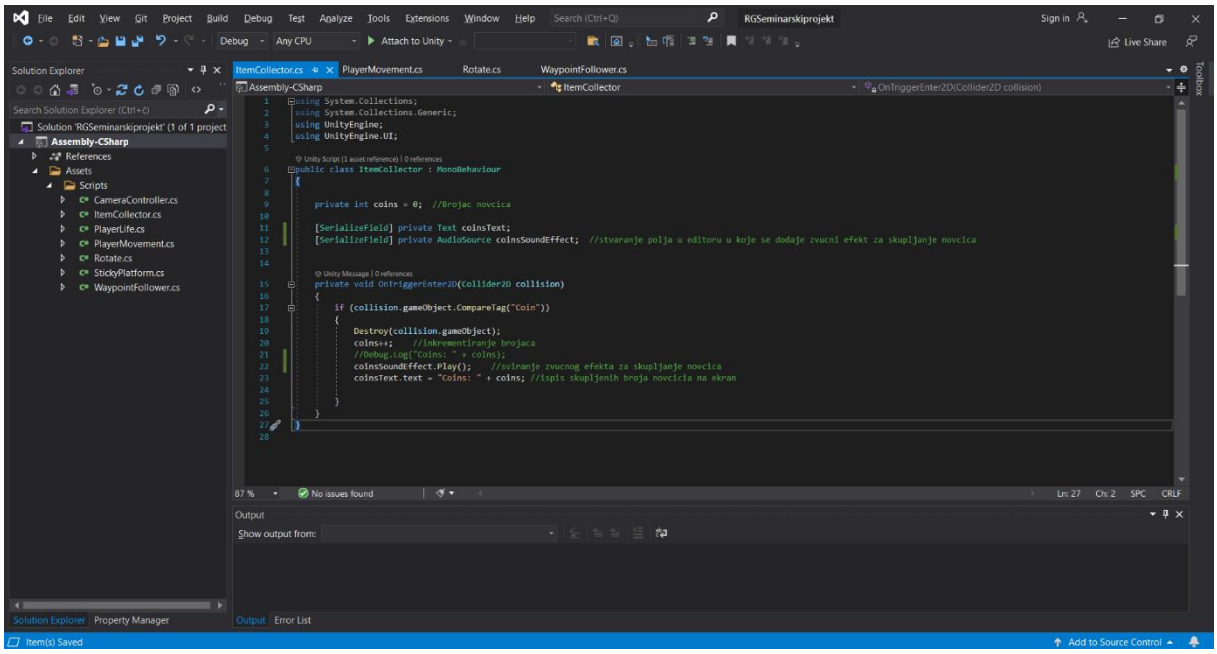
4.4 Coinsi – 4. korak

Jedan od glavnih razloga kretanja u igri je skupljanje nagrada kako bi igrač imao osjećaj ostvarenosti dok igra igru. U ovoj igri su nagrade *Coinsi* koje igrač skuplja dok se kreće 2D

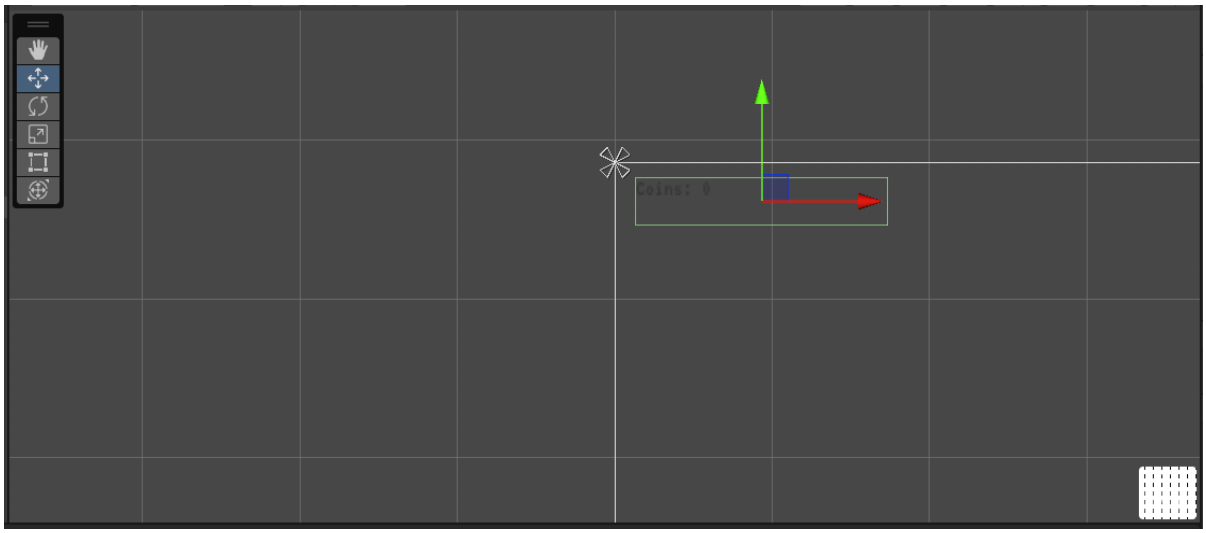
svijetom. Prvo je potrebno kreirati 2D objekt i nazvati ga *Coin* te dodati sliku i animaciju. Igrač skupi *Coin* tako da se sudari s njim pa je objektu *Coin* potrebno dodati komponentu *Box Collider 2D.* Isto tako je potrebno objektu dodati *tag* pod nazivom *Coin* kako bi igrač prilikom sudara znao da je riječ upravo o tom objektu. Prepoznavanje se napravi u skripti *ItemCollector.cs* koja se doda kao komponenta objektu *Player*. Prilikom sudara se provjerava je li se igrač sudario s objektom koji ima tag *Coin* te ukoliko je, onda se objekt s tim tagom uništi te se uveća stanje skupljenih novčića za jedan i ispiše se trenutno stanje novčića u gornjem lijevom prozoru. Slika 4.7 prikazuje dodavanje objekta Coin, 4.8 prikazuje kod koji obavlja detekciju sudara s novčićem, a 4.9prikazuje položaj ispisa trenutnog stanja novčića.



Slika 4.7 Dodavanje objekta Coin



Slika 4.8 Skripta ItemCollector.cs

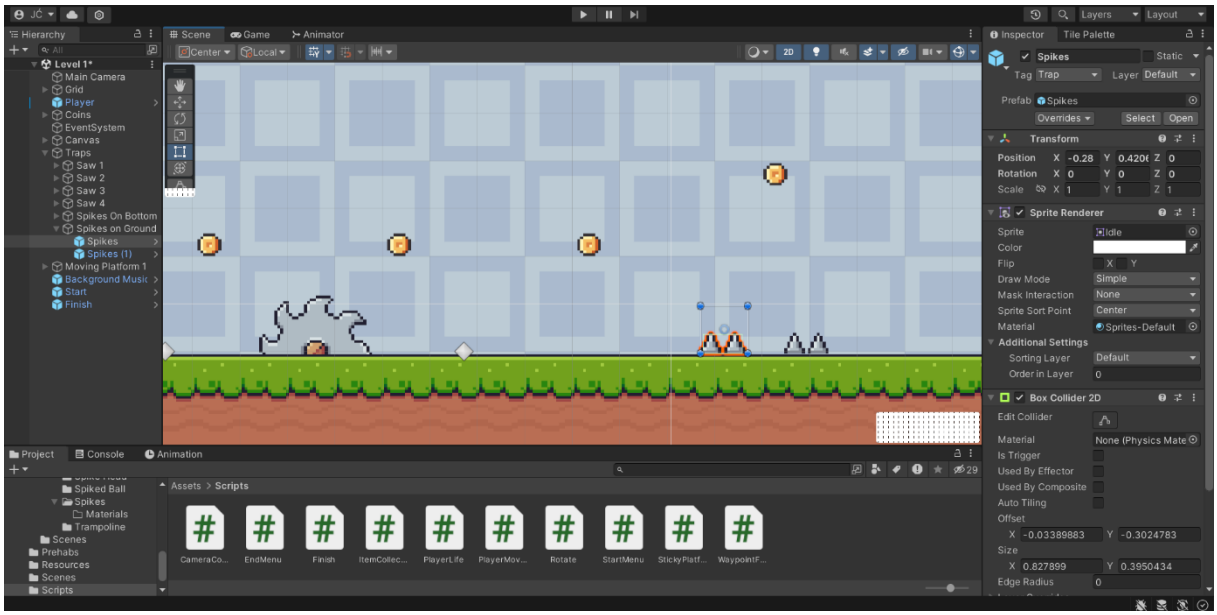


Slika 4.9 Ispis trenutnog stanja novčića

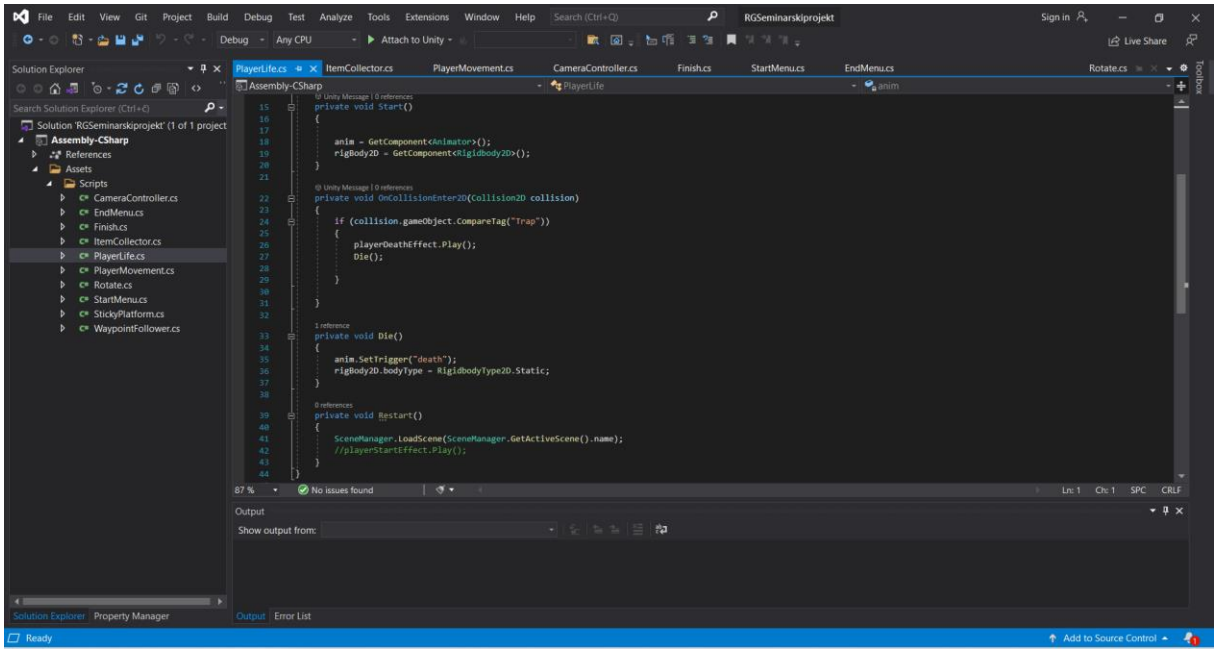
4.5 Zamke – 5. korak

Kako igrač ne bi samo lagodno šetao po svijetu i skupljao novčiće, potrebno je malo otežati stvari. Za to služe zamke. Zamke se stvore tako da se kreira 2D objekt, doda mu se komponenta *Box Collider 2D* i stvori se tag *Trap* tako da igrač zna da se sudario sa zamkom. Za prepoznavanje je kao i u prethodnom slučaju potrebno objektu *Player* kao komponentu dodati skriptu pod nazivom *PlayerLife.cs*. Prilikom sudara se provjerava je li se objekt *Player* sudario s objektom koji ima tag *Trap* te ukoliko je tada se igrač prebacuje u stanje *Death* i

pokreće se animacija za smrt i učitava se ponovno level (scena). Slika 4.10 prikazuje dodavanje objekta s tagom Trap, a slika 4.11prikazuje kod koji obavlja detekciju sudara sa zamkom.



Slika 4.10 Dodavanje objekta s tagom Trap

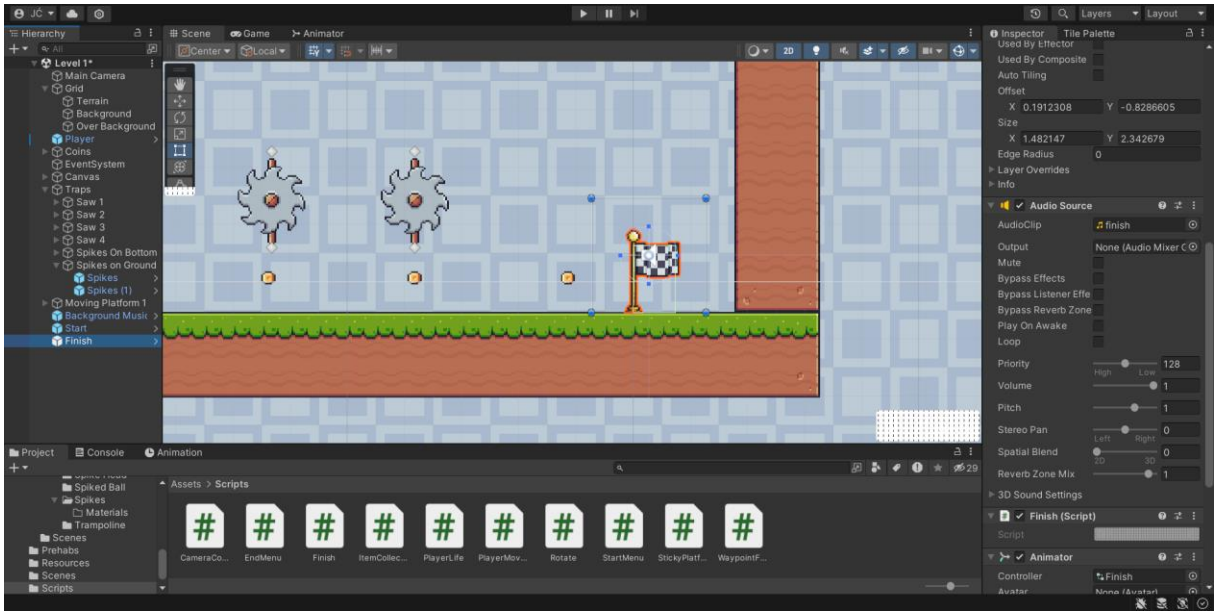


```
private void Start()
{
    anim = GetComponent<Animator>();
    rigBody2D = GetComponent<Rigidbody2D>();
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Trap"))
    {
        playerDeathEffect.Play();
        Die();
    }
}

private void Die()
{
    anim.SetTrigger("death");
    rigBody2D.bodyType = RigidbodyType2D.Static;
}

private void Restart()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    //playerStartEffect.Play();
}
```

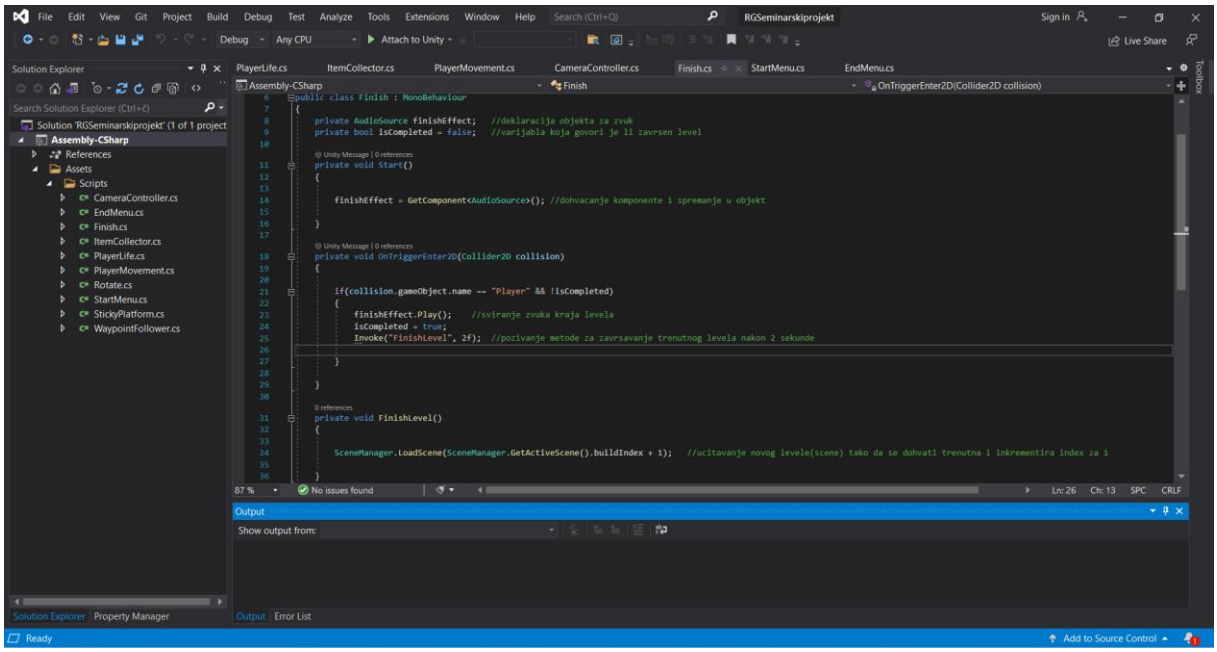
Slika 4.11 Skripta PlayerLife.cs

4.6 Zvučni efekti – 6. korak

Kako bi igra bila što realnija potrebno je dodati i zvučne efekte kao što su na primjer zvuk za skupljanje novčića, zvuk za smrt, zvuk za završetak levela, pozadinska glazba… Zvučni efekt se dodaje tako da se objektu doda komponentu *Audio Source* i odabrani zvučni efekt te unutar skripte, koja je već dodana kao komponenta objektu, se dohvati komponenta *AudioSource* te se spremi u objekt. Kada se želi pustit glazba onda se jednostavno nad tim objektom, u kojem je spremljena komponenta, pozove metoda *Play()*. Slika 4.12 prikazuje dodavanje komponente *Audio Source* objektu *Finish*, a slika 4.13prikazuje kod za puštanje glazbe u skripti *Finish.cs*.



Slika 4.12 Dodavanje komponente Audio Source

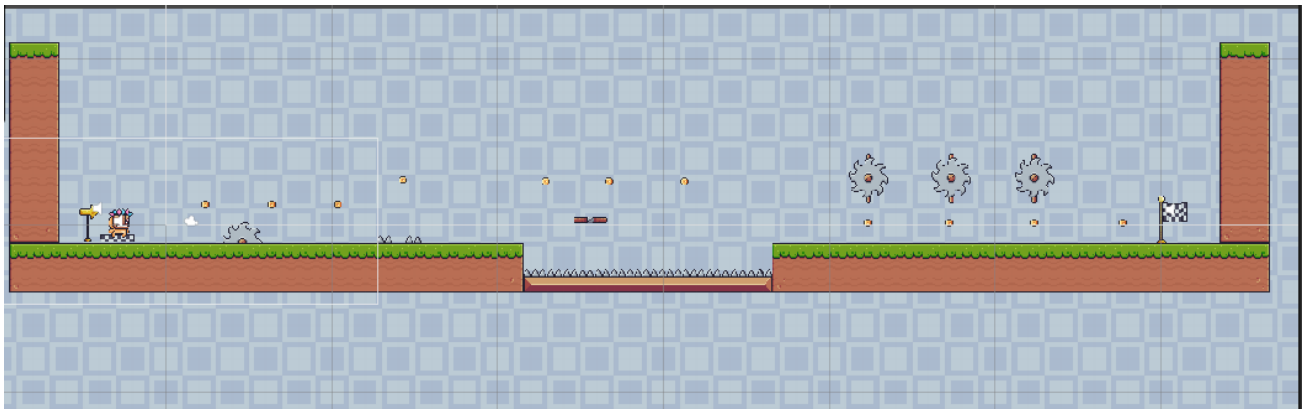


Slika 4.13 Skripta Finish.cs

Pozadinska se glazba dodaje na isti način. Doda se 2D objekt i nazove *Background Music* te mu se doda komponenta *Audio Source* i u postavkama ostavi *on awake*.

4.7 Level 2 – 7. korak

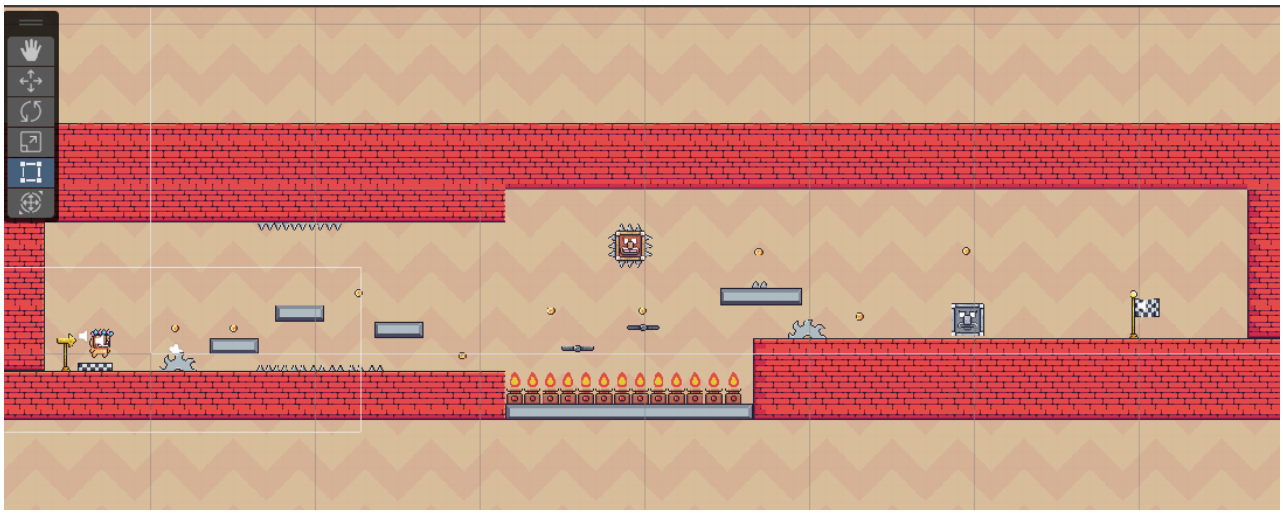
Kada je sve gotovo s prvom razinom, onda se kreće na iduću. Slika 4.14 prikazuje kompletnu prvu razinu u igri.



Slika 4.14 Level 1

Level je zapravo *Scena* u Unityju te je za izradu novog levela potrebno izraditi novu *Scena*. Najlakši način je jednostavno kopirat postojeću i raditi promjene. Isto tako je dobra praksa i napraviti mapu *Prefabs* i u nju spremati sve objekte koji će se ponovo koristit. Na razini 2 se

koriste zamke koje su napravljene na isti način kao i na prethodnoj razini (nove zamke su vatra i bodljikava glava). Također je promijenjena i pozadina i teren. Da bi se prešlo s jedne razine na drugu potrebno je dodati kod za učitavanje nove scene, a taj se kod nalazi u skripti Finish.cs i vidljiv je na slici 4.13. Slika 4.15 prikazuje kompletnu drugu razinu igre.



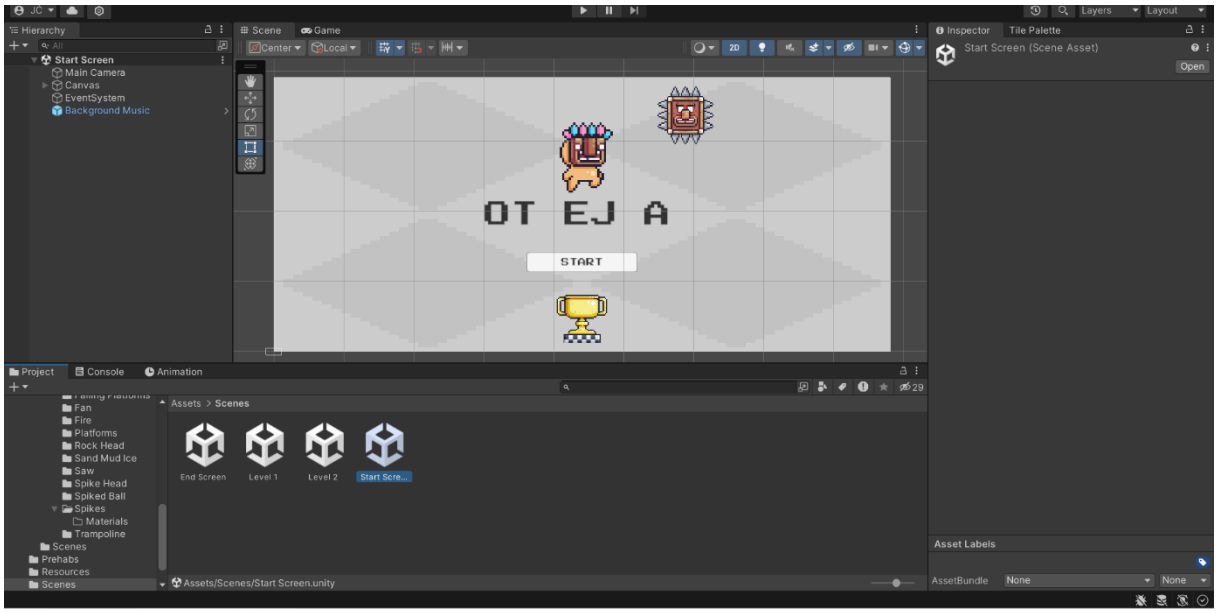
Slika 4.15 Level 2

4.8 Start i End Menu – 8. korak

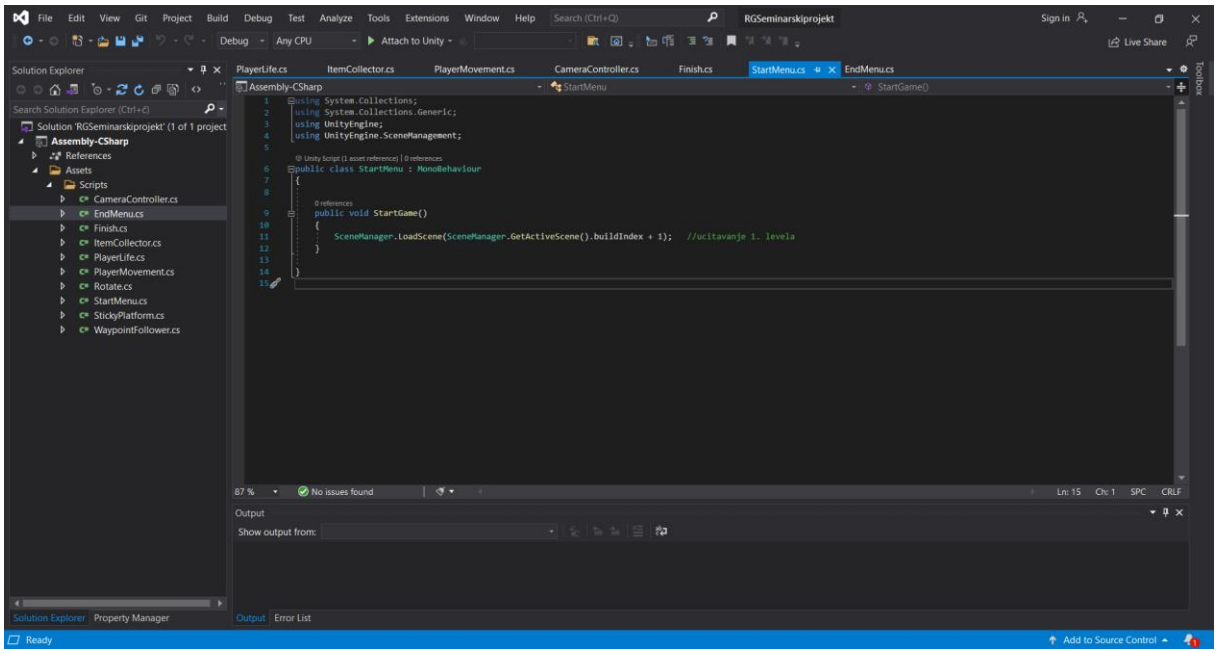
Na samom kraju, svakoj igrici je potreban izbornik. Izbornik se radio kao i scena. Potrebno je napraviti dvije nove scene za svaki izbornik. Izbornik se uredi da vizualno izgleda privlačno, odnosno, doda mu se tekst i slika/e.

4.8.1 Start Menu

Ovdje se doda još jedan objekt tipa *Button* i tom objektu se kao komponenta doda skripta *StartMenu.cs* u kojoj se prilikom klika na *Button* učitaje prva razina igre. Slika 4.16 prikazuje cijelu scenu, a slika 4.17 prikazuje skriptu *StartMenu.cs*.



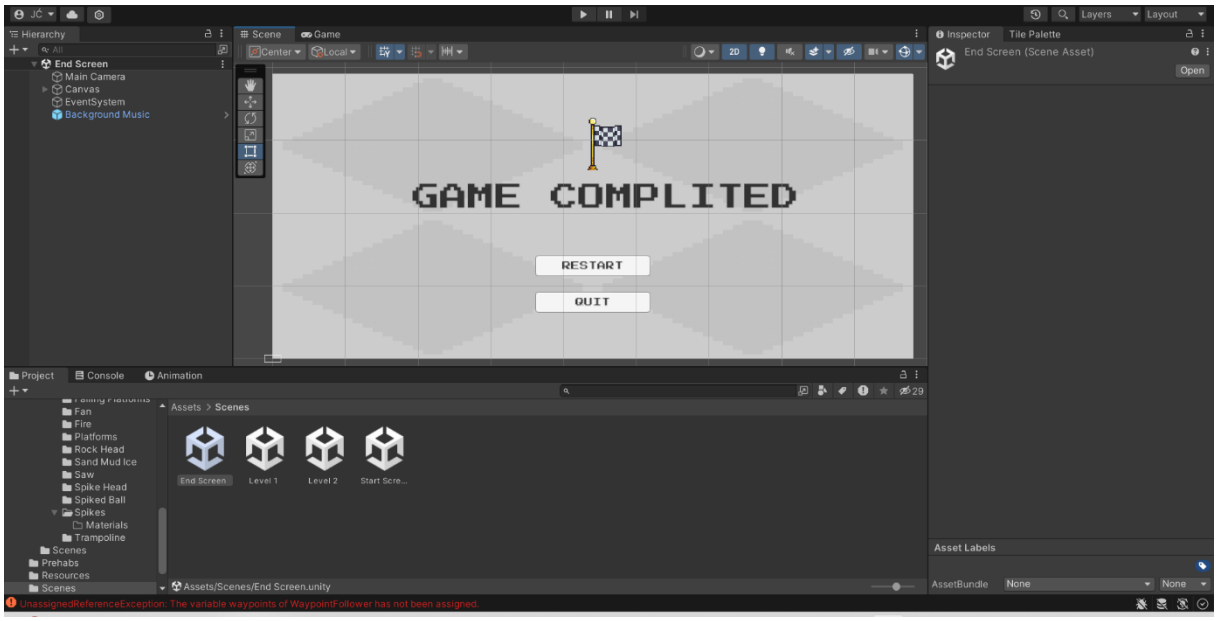
Slika 4.16 Start Menu



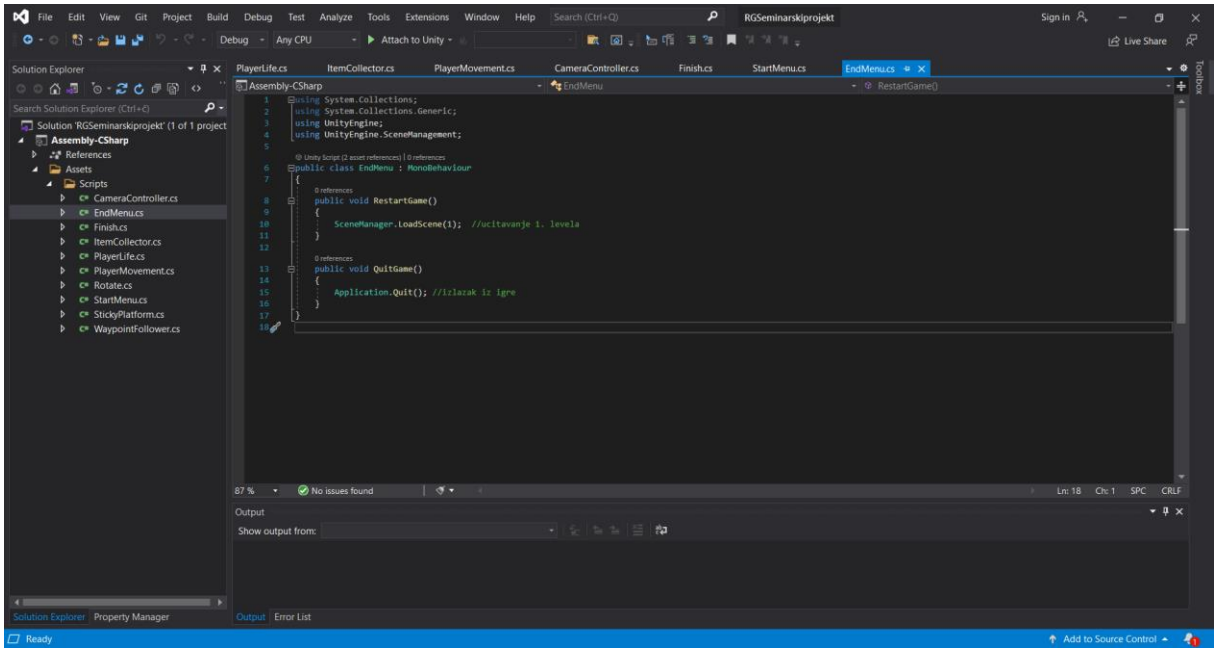
Slika 4.17 Skripta StartMenu.cs

4.8.2 End Menu

Slično kao i u prethodnom slučaju za *Start Menu*. Ovdje se dodaju dva objekt tipa *Button* (jedan za ponovno pokretanje igre, a drugi za izlaz iz igre) te im se kao komponenta doda skripta *EndMenu.cs*. Slika 4.18 prikazuje cijelu scenu, a slika 4.19 prikazuje skriptu *EndMenu.cs*.



Slika 4.18 End Menu



Slika 4.19 Skripta EndMenu.cs

5 ZAKLJUČAK

Industrija računalnih igara neprestano se razvija. Kao posljedica toga, razvijaju se alati za što bržu i jednostavniju implementaciju igara. Programeri se često okreću upravo *cross-platform* razvoju igara što pogoduje smanjivanju vremena isporuke. Sve navedene karakteristike pruža upravo Unity *game engine*. Unity je jednostavan za korištenje, a pogodan je i za *cross-platform* razvoj 2D i 3D igara.

Razvoj 2D igre u Unityju predstavlja dinamičan proces koji se temelji na kombinaciji moćnih alata, fleksibilnosti i kreativnosti. Unityjev 2D sustav pruža programerima jednostavan pristup manipulaciji spriteova, animacija i kolizija, čime se omogućava kreiranje vizualno privlačnih igara. Ključne komponente poput Box Collider 2D, Rigidbody2D, Tilemap Collider 2D i Composite Collider 2D omogućuju preciznu kontrolu kolizija i dinamike igre. Integracija *Sprite Renderera,* Unityjevog Animation sustava te zvuka pridonosi bogatstvu iskustva igrača. Unityjev Asset Store dodatno olakšava razvojni proces pružanjem širokog spektra resursa, od grafika do korisnih alata. Ova raznolikost ubrzava razvoj i omogućuje programerima da unaprijede svoje igre uz minimalni napor.

U zaključku, Unity se ističe kao svestran i pristupačan game engine za razvoj 2D igara. Raznolikost alata, zajedno s podrškom raznih resursa, čini Unity idealnim okruženjem za programere željne kreiranja visokokvalitetnih i zabavnih 2D igara. Razvojni proces u Unityju nije samo tehnički izazovan, već pruža i prostor za izražavanje kreativnosti te postizanje željenih igračkih iskustava.

LITERATURA

[1] „24 Great Games That Use The Unity Game Engine“, s Interneta, https://www.thegamer.com/unity-game-engine-great-games/, pristupljeno: 24.01.2024.

[2] „What is Unity? Everything you need to know“, s Interneta, https://www.androidauthority.com/what-is-unity-1131558/, pristupljeno: 24.01.2024.

[3] „Introduction to Visual Studio“, s Interneta, https://www.geeksforgeeks.org/introduction-to-visual-studio/, pristupljeno: 24.01.2024.

[4] Wikipedia, s Interneta, https://en.wikipedia.org/wiki/Visual_Studio#History, pristupljeno: 24.01.2024.

[5] „Build a 2D Platformer Game in Unity | Unity Beginner Tutorial“, s Interneta, https://www.youtube.com/watch?v=Ii-scMenaOQ&list=PLrnPJCHvNZuCVTz6lvhR81nnaf1a-b67U&index=2, pristupljeno: 24.01.2024.

PRILOZI

Kazalo slika

Popis oznaka i kratica

VR Virtual Reality

AR Augmented Reality

IDE Integrated Development Environment

SAŽETAK I KLJUČNE RIJEČI

Sažetak

U seminarskom radu opisana je realizacija 2D računalne igre korištenjem Unity game enginea. Unity je cross-platform game engine koji se koristi za razvoj video igara, simulacija i drugog interaktivnog sadržaja. Unity omogućuje vizualni editor sa širokim rasponom različitih funkcionalnosti. Microsoft Visual Studio je integralno razvojno okruženje koje je razvio Microsoft Corporation. Kombinirajući moćan Unity game engine s Visual Studiom, stvara se robusno okruženje za realizaciju kreativnih vizija. U procesu razvoja, koristi se Unityjev 2D sustav koji olakšava manipulaciju spriteova, animacija i kolizija. U radu je prikazano kako je jednostavno korištenjem Unity game izraditi zanimljivu, a istovremeno i zahtjevnu igru.

Ključne riječi

Videoigre, Unity, Microsoft Visual Studio