Razvoj Video igara (tehnike i alati)

Sever, Luka

Sveučilište Jurja Dobrile u Puli

Naziv sastavnice

Luka Sever

Razvoj Video igara (Tehnike i Alati)

Završni rad

Pula, 3. rujna, 2022. godine.

Sveučilište Jurja Dobrile u Puli

Naziv sastavnice

Luka Sever

Razvoj Video igara (Tehnike i Alati)

Završni rad

JMBAG:0303092150, redoviti student

Studijski smjer: Računarstvo

Predmet: Multimedijalni sustavi

Znanstveno područje:

Znanstveno polje:

Znanstvena grana:

Mentor: doc.dr.sc. Željka Tomasović

Pula, 3. rujna, 2022. godine.

Tehnički fakultet u Puli

Ime i prezime studenta/ice Luka Sever

JMBAG 0303092150

Status: ☒ redoviti ☐ izvanredni

PRIJAVA TEME ZAVRŠNOG RADA

Doc. dr. sc. Željka Tomasović
Ime i prezime mentora

Računarstvo
Studij

Multimedijalni Sustavi
Kolegij

Potvrđujem da sam prihvatio/la temu završnog/diplomskog rada pod naslovom:

Razvoj Video igara (Tehnike i Alati)

(na hrvatskom jeziku)

Computer Game Development (Techniques and Tools)

(na engleskom jeziku)

Datum: 26.1.2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Luka Sever, kandidat za prvostupnika inženjera računarstva ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Luka Sever

U Puli, 3.9.2022.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Luka Sever dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom Razvoj video igara kroz godine (tehnike i alati)

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 3.9.2022.

Potpis

Luka Sever

Sadržaj

1. Uvod

Ne čudi da je ljudska ljubav prema igrama dovela do širenja granica iz stvarnog u virtualni svijet. Inovacija, tehnologija i individualci koji gaje veliku ljubav prema video igrama, udružuju se kako bi one postale stvarnost. U današnje vrijeme ljudi su okruženi svakojakom tehnologijom, što im omogućuje sudjelovanje u igranju videoigara. Prve igre razvile su se zbog ljudske radoznalosti i težnje za novim iskustvima, a time i za novim i svježim oblicima zabave. Video igre, od začetka pa do danas sve više dobivaju na značaju, što dalje razvija i novu industriju – industriju videoigara.

Zadatak ovog rada bio je upoznavanje s poviješću razvoja videoigara i djelomična izrada vlastite video igre unutar jednog od alata za razvoj videoigara. Izrađena igra konstruirana je u okruženju softvera Unity. Cilj igre je upoznavanje s metodologijom izrade videoigara, kao i upoznavanje s radom u jednom od poznatijih alata za izradu igara.

2. Povijesni razvoj videoigara

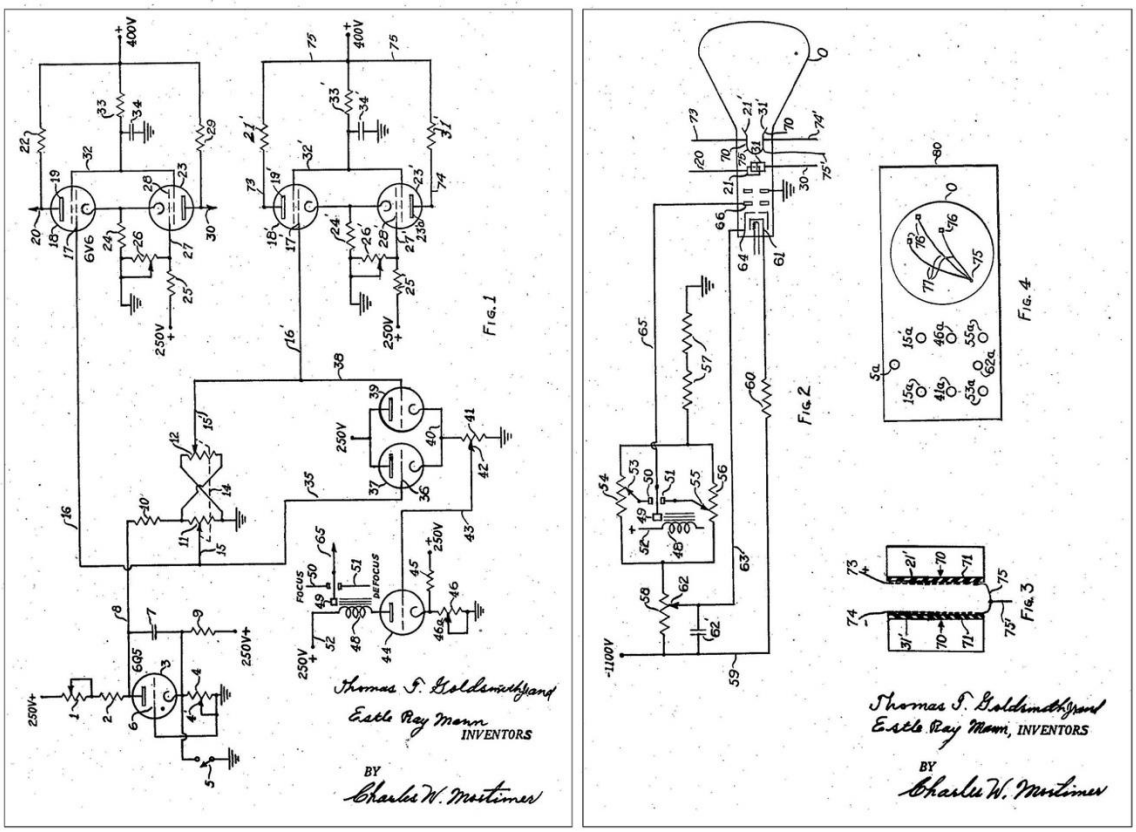
2.1 Rana povijest (1948 – 1970)

Pojam „video igra“ evoluirao je tijekom godina od pukog koncepta do novog i revolucionarnog sredstva zabave. U današnje vrijeme, kako bi se nešto smatralo video igrom, mora odašiljati video signal sredstvu za reprodukciju (računalni monitor, televizijski zaslon, itd.). Navedena definicija ne obuhvaća starije igre koje su kao sredstvo reprodukcije koristile pisač ili teleprinter - telegrafski pisaći stroj koji je služio primopredaji pisanih poruka [1].

Danas, pojam video igre obuhvaća bilo koju igru koja se može pokrenuti uz pomoć hardvera koji sadrži nužne logičke sklopove i koji omogućuju korisniku interakciju s unutarnjim svijetom same igre. Također, prikazuju i rezultat na temelju igračevih odabira. Ako se u obzir uzme definicija video igre, prva interaktivna igra

konstruirana je pedesetih godina dvadesetog stoljeća i nosila je naziv „cathode-ray tube amusement device“. Slika 1 prikazuje patent prve elektroničke igre [2].

Navedeni patent realiziran je samo kroz jedan ručno izrađeni prototip, dok industrijska proizvodnja nije bila pokrenuta[2].



Slika 1. Patent logičkog sklopovlja Thomasa Tolivana Goldsmitha Jr. (preuzeto s internetske stranice https://en.wikipedia.org/wiki/Cathode-ray_tube_amusement_device)

Ubrzo nakon objave patenta, Alan Turing i David Champernowne pokreću proizvodnju prve pisane računalne igre zvane „Turochamp“, koja slavu stječe pod drugim imenom - „Turbochamp“. „Turbochamp“ je šahovski program izrađen u svrhu predviđanja poteza pomoću računalnog algoritma koji odabire optimalan sljedeći potez. No, algoritam je bio u mogućnosti predvidjeti samo 2 poteza unaprijed. Turing je svoj algoritam pokušao pretvoriti u program koji bi se pokretao na računalu Ferranti Mark 1 (komercijalna verzija računala u Manchesteru). No navedeno računalo, zbog nedostatnih mogućnosti, nije bilo u stanju pokrenuti program zbog kompleksnosti koda [3].

2.2 Dvadeseto stoljeće

Tijekom šezdesetih i sedamdesetih godina dvadesetog stoljeća proizvedeno je nekoliko računalnih igara koje su bile relativno neuspješne na tržištu zbog slabih računalnih resursa dostupnih krajnjim korisnicima [4]. Krajem sedamdesetih godina dvadesetog stoljeća situacija se značajno mijenja. Jezici korišteni u svrhu programiranja, kao FORTRAN i COBOL, bili su vrlo tehnički zahtjevni, stoga cijela industrija razvoja videoigara prelazi na programske jezike visoke razine, kao što su BASIC i C. Navedeni jezici nudili su veći raspon značajki, od dinamičke alokacije memorije do rekurzije [5]. S novim jezicima počinje se razvijati tehnologija zvana dijeljenja resursa (eng. *time-sharing)* koja dopušta dijeljenje resursa jednog sustava na više korisnika koji se povezuju preko terminala. Mogućnost povezivanja na udaljeni sustav rješava problem dostupnosti računala. Naime, tada je jedno računalo dijelilo više različitih osoba unutar iste ustanove.[6]

Ubrzo, razvojem portabilnog UNIX operativnog sustava, stvaraju se standardizirana programska okruženja koja su omogućavala prenošenje programa između više sustava. Time se doskočilo problemu *de novo* pisanja programa za svaki sustav. Osnivanje časopisa posvećenih temama vezanim uz računala, pr. časopis „Byte“ te izdavanjem knjige „101 BASIC Computer Games“ koja je sadržavala računalni kod koji bi korisnik sam upisivao i pokretao, uvelike je povećala interes javnosti za industrijom videoigara. Isto tako, time se omogućilo lakše dijeljenje novosti vezanih uz industriju, kao i ponuda novih programa krajnjim korisnicima [7].

S naglim porastom interesa za video igrama, raste i masovnost industrije videoigara, što vodi hiperprodukciji velikog broja nekvalitetnih proizvoda plasiranih na tržište od strane nepouzdanih tvrtki [8]. Konzola pod imenom „Atari VCS“, proizvod tvrtke Atari, svrstana je među najpopularnije konzole tog vremena, zbog čega je imala i velik broj njoj specifično izrađenih igara. Većina tih igara bila je lošije kvalitete, ili je jednostavno prerano plasirana na tržište što je rezultiralo gubitkom povjerenja u kvalitetu same industrije. Svi navedeni faktori rezultirali su padom industrije videoigara 1983. godine [8].

Pojavom novih sustava, industrija počinje dobivati više pažnje zbog boljeg marketinga. Jedan od najznačajnijih ulagača u promociji računalne industrije je British Broadcasting Corporation (BBC), koji je aktivno promovirao računalnu edukaciju i poticao razvoj BBC Micro računala. BBC Micro je serija mikroračunala izrađena od strane „Acorn Computers“ za BBC namijenjena poboljšanju obrazovanja u britanskim domovima i školama [9]. Zbog novih sustava počinje se pojavljivati i nova vrsta programera zvana „bedroom coders“, koji su prodavali svoje vlastite softvere dostupne na navedenim novijim sustavima. Manje izdavačke kuće, kao Acornsoft i Mastertronic, osnovane su s ciljem pružanja potrebnih resursa manjim proizvođačima kako bi spremnije mogli distribuirati svoj proizvod široj javnosti. Ubisoft je, također, započeo kao distribucijska kuća na francuskom tržištu, no sredinom osamdesetih godina proširuje se na proizvodnju i izdavanje vlastito-proizvedenih igara. Igre proizvedene u to doba pomogle su u oživljavanju industrije videoigara koja je bila u padu proteklih nekoliko godina. [10]

Devedesete godine dvadesetog stoljeća predstavljaju godine inovacije u industriji videoigara. Igre se prebacuju s dvodimenzionalne (**2D**) grafike na trodimenzionalnu (**3D**), što je dovelo do pojave novih žanrova igara, kao što su *first person shooter* (FPS), *real time strategy* (RTS) i *massively multiplayer online game* (MMO). Arkadne igre doživjele su svoj pad s pojavom privatnih kućnih konzola i računalnih sustava te se počinju zatvarati mjesta koja su nudila takav oblik zabave, pr. igrače arkade (eng. *arcade*).[11]

Krajem osamdesetih godina dvadesetog stoljeća igre za konzole isključivo postaju dostupne u obliku *read-only memory* (ROM) kazeta, a računalne igre kao diskete. Takvi mediji za pohranjivanje imali su veoma limitirano mjesto za pohranu koje nije omogućavalo spremanje većih animiranih segmenta u igre. Rješenje tog problema pronađeno je u optičkim medijima, specifično u mediju *compact disk-read-only memory* (CD-ROM). CD-ROM je sredinom osamdesetih godina predstavljen kao medij za distribuciju glazbe. No, ranih devedesetih godina dvadesetog stoljeća, zbog veće memorije pohrane i niske cijene, počinje zamjenjivati staromodne kazete. [12]

Prebacivanjem igara na optičke uređaje industrija se također prebacuje s 2D grafike na 3D grafiku. Pojedini arkadni sustavi još su se uvijek koristili jednostavnom vektorskom grafikom kako bi simulirali 3D okolinu.

Još je jedna tehnika korištena u svrhu simulacije 3D grafike, a to je paralaksno prikazivanje (eng. *parallax rendering*) različitih slojeva i skaliranje objekata kako se pomiče kamera. Paralaksno prikazivanje točno simulira dubinu nekog objekta, npr. zidova, bez da ima preveliki utjecaj na performanse igre [13]. Navedeni trikovi nisu bili uspješni u mimici prave 3D grafike, već su bili bliže 2.5D [14]. Prava 3D grafika koristila je poligone. Poligoni se koriste za sastavljanje 3D konstrukata koji, kada se povežu u jedno, čine pravi 3D model. Takav način prikazivanja modela popularizirala je igra zvana „Virtua Fighter“ (Slika 2.) [15].

Slika 2. Video igra „Virtua Figter“, primjer prvog korištenja 3D grafike u igrama (preuzeto s internetske stranice https://www.arcade-history.com/images/game/3298_2.png)

Pošto su računala bila naprednija od tadašnjih igraćih konzola, napreci u grafici bili su drastičniji zbog dostupnosti boljeg hardvera. Tvrtka pod imenom id Software izbacuje prvi pravi *game engine* tog vremena koji dijeli sadržaj u razne slojeve i olakšava izradu igara. *Game engine* je razvojna okolina, također poznata pod nazivom arhitektura igre, koja sadrži postavke za optimiziranje i pojednostavljenje razvoja videoigara [16].

Korist *game enginea* očituje se u činjenici da njegovom pojavom tvrtke više nisu morale same programirati 3D okruženje *de novo*, nego se ono već nalazilo unutar njega. Prvi *game engine* nije bio u potpunosti 3D, pošto su pojedine komponente, kao objekti i mape, bile još uvijek 2D. No, 1996. godine, id Software na tržište stavlja novu igru pod imenom „Quake“ koja je koristila potpuno novi *game engine* pod nazivom *Quake engine*. *Quake engine* je bio prvi pravi 3D *engine* koji je koristio 3D modele objekata i likova. Nova tehnologija *game enginea* dovela je do znatnog porasta u popularnosti i kvaliteti FPS igara [17].

2.3 Dvadeset i prvo stoljeće

Napreci u tehnologiji, popraćeni rastućom zainteresiranošću svijeta za video igrama, obilježavaju dvadeset i prvo stoljeće kao stoljeće najdrastičnijih promjena u industriji videoigara [18]. Potrošači, željni novih ideja, aktivno pomažu i grupno financiraju (eng. *crowdfunding)* proizvođače. Upravo pomoću grupnog financiranja, individualcima i manjim proizvođačima postaju dostupna sredstva uz pomoć kojih mogu proizvoditi kvalitetne proizvode konkurentne onima velikih proizvođača. Veliki proizvođači, unatoč povećanoj popularnosti manjih, održavaju konkurentnost unutar industrije osiguravajući veće budžete za izradu igara i softvera. Igre koje su nastale pod većim budžetom svrstavaju se u „AAA“ kategorizaciju. Kategorizacija AAA u industriji videoigara predstavlja igru koju su na tržište plasirale srednje ili velike tvrtke, kao npr. tvrtka Ubisoft. Igre AAA tvrtki mnogo su lakše postizale individualno zadane financijske ciljeve i generalno su bolje zarađivale [19]. Igra pod imenom „Final Fantasy VII“, jedna je od prvih igara za čiju je izradu bio dostupan budžet koji iznosi otprilike 40 do 45 milijuna dolara [20].

Proizvodi manjih tvrtki i individualaca poprimaju naziv *indie*. Pojam *indie* potječe od engleske riječi *independent* što znači samostalno. Igre koje spadaju u kategoriju *indie* karakterizira samostalnost i neovisnost o velikim tvrtkama i izdavačima [21]. Za razliku od AAA igri, koje počinju gubiti unikatnost zbog recikliranja starih igara, poput „Call of Duty“ i „Medal of Honor“ [22], *indie* igre, zbog svoje slobode, pružaju veću unikatnost i često su mnogo zanimljivije [23].

Potražnja za inovativnim igrama potiče razvoj novih platformi koje omogućavaju igraču bolje iskustvo. Mobilni uređaji, koji su do tada drastično napredovali, predstavljaju još jednu u nizu novih platformi koje se ističu po mogućnosti igranja u pokretu. Proizvođači mobilnih uređaja, Apple i Google, proizvode i predstavljaju novu generaciju pametnih telefona na kojima je dostupna trgovina koja omogućuje kupovinu aplikacija i igara namijenjenih mobilnim uređajima [24][25]. Zbog velike popularnosti mobilnih igara, kao npr. „Bejeweled“ [26], proizvođači igara pronalaze nove načine financiranja nauštrb igrača, preko reklama ili dodatnih funkcija pod naplatom unutar igre [27].

Virtualna stvarnost (eng. *virtual reality,* ***VR****)*, nakon nekoliko neuspjelih pokušaja (npr. proizvod tvrtke Nintendo pod imenom „Virtual Boy“), počinje bolje kotirati na tržištu zbog svoje pristupačnije cijene [28]. Najava uređaja pod imenom „Oculus Rift“ kao prvog komercijalno dostupnog VR uređaja, poticajno djeluje na ostale tvrtke, poput tvrtke Sony i Valve koje počinju proizvoditi vlastite VR uređaje [29]. Igre proizvedene za VR tržište nisu zadovoljile očekivanja postavljena s najavom prvog VR uređaja, glavni razlog čemu biva nedostatak odgovarajućih igara [29].

3. Samostalan rad – izrada jednostavne igre pomoću softvera Unity

3.1 Odabir ideje

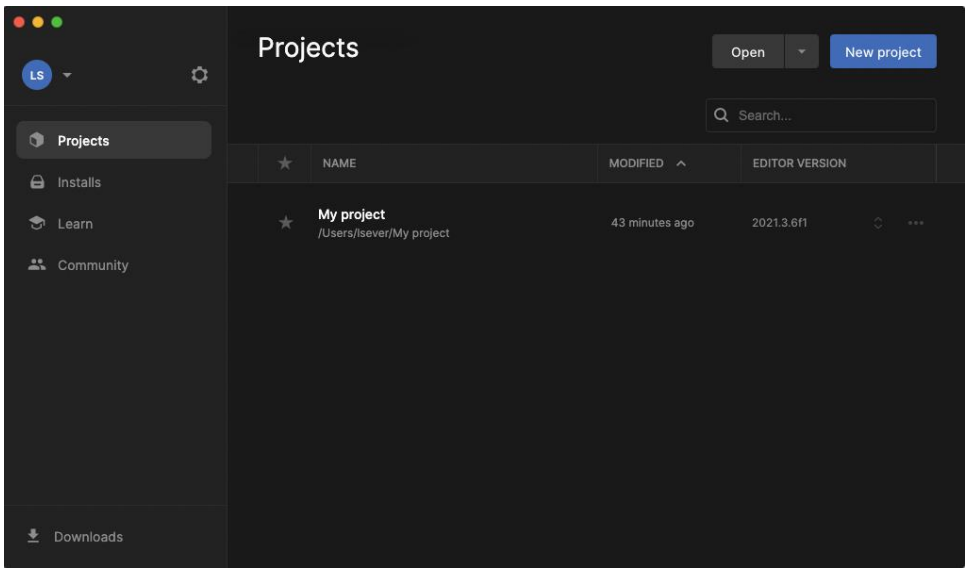
Za početak izrade prvo je potrebna ideja. U današnje vrijeme postoji mnogo žanrova videoigara, od *real time strategy* (RTS) igara do akcijsko avanturističkih igara, što pruža vrlo velik izbor mogućih puteva. Također, otežava odluku jer fokus ne mora uvijek padati samo na jedan stil, jer se kombinacijom različitih dostupnih stilova može postići unikatniji proizvod.

Odabrana tema u sklopu samostalnog rada je izrada video igre tipa „*2D Farming Game*“. Tema je odabrana zbog nekoliko razloga. Prvi razlog odabira je kompleksnost sadržaja. Igra sadrži nekoliko sustava koji se odvijaju u pozadini te sustave s kojima igrač može vršiti interakciju. Drugi razlog je dostupnost resursa (eng. *assets*). Resursi čine najvažniji dio igre. Bez njih ona jednostavno ne bi pravilno funkcionirala. Objekti, zvukovi, pozadine, likovi, entiteti, itd... sve su to materijali koji se koriste za prikazivanje događaja unutar igre, bez kojih bi igrač vidio samo prazan prozor.

3.1 Instalacija potrebnog softvera i priprema računala za izradu

Prije početka izrade potrebno je proučiti različite alate s kojima se može najbolje ostvariti vizija željenog ishoda. Na tržištu je dostupno mnogo različitih alata za izradu igara, poput softvera Unity, Unreal Engine, RPG Maker i Godot, od kojih se dio koristi bez naknade (Unity, Godot, Unreal Engine), a za ostale je potrebna licenca (RPG Maker). Mnogi softveri dostupni su studentima po promotivnim uvjetima i nude upoznavanje s radom samog softvera bez kupnje licence, kao i pristup tečajevima („tutorijalima“) preko kojih se mogu savladati osnovne vještine potrebne za daljnje korištenje.

Softver Unity, kao jedan od takvih softvera, besplatan je za korištenje i nudi širok raspon značajki, no relativno je kompliciran za uporabu. Jedna od prednosti softvera Unity leži u činjenici da se proizvedena igra može isporučiti na više različitih platformi te se logika programira u C# programskom jeziku, bez potrebe za savladavanjem novog programskog jezika. Nakon preuzimanja softvera glavno korisničko sučelje (eng. *editor*) ne pokreće se istodobno sa softverom, već se prvo mora kreirati projekt u prozoru „Unity Hub“ (Slika 3.).



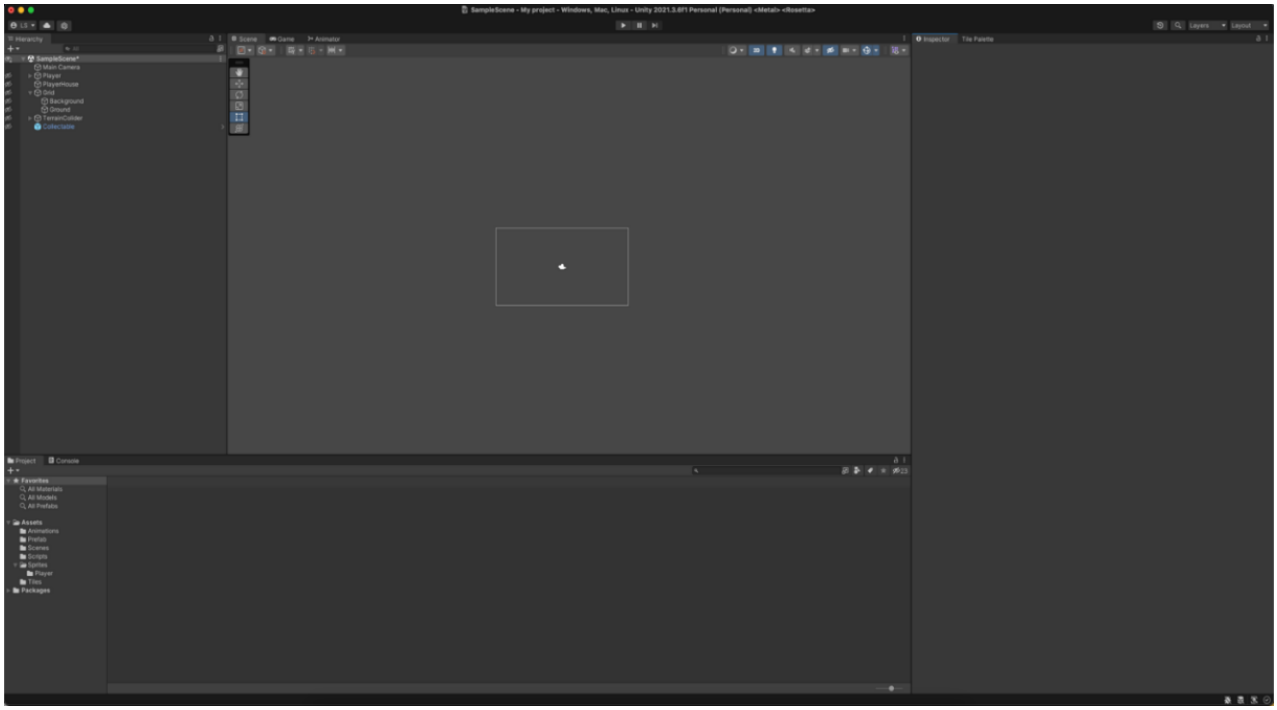
Slika 3. Prozor „Unity Hub“ unutar softvera Unity (slika zaslona, izradio Sever, L.)

Gumb „New Project“ otvara prozor s ponuđenim predlošcima koje korisnik slobodno odabire, a služe lakšem početku izrade. Svaki od predložaka podudara se s različitim popularnim žanrovima igara na tržištu i znatno skraćuje potrebno vrijeme uspostave novog projekta. Odabrani predložak za izradu igre je *2D Core* koji postavlja 2D okruženje i podešava postavke kamere.

3.2 Upoznavanje s korisničkim sučeljem

Pokretanjem novokreiranog projekta pokreće se prozor „Unity editor“ (Slika 4.). Glavno sučelje Unity Editora sastoji se od nekoliko osnovnih elemenata: 1. u središtu prikaza nalazi se „Scene“ i „Game view“, 2. nalijevo od središnjeg prozora „Scene“ nalazi se prozor „Hierarchy“, 3. na dnu zaslona podno prozora „Scene“ nalazi se prozor „Project“ i konzola, 4. nadesno od prozora „Scene“ nalazi se prozor „Inspector“.

Prozor „Hierarchy“ prikazuje sve elemente koji se nalaze u igri. Preko njega je moguće stvoriti različite objekte čije se podešavanje vrši u prozoru „Inspector“. Bez ovog prozora precizan odabir objekata za podešavanje ne bi bio moguć, već bi se, nasumičnim pritiskom na određene elemente, moralo "pogoditi" željeni objekt u sceni.



Slika 4. Glavno korisničko sučelje softvera Unity – „Unity Editor“ (slika zaslona, izradio Sever, L.)

Prozori „Scene view“ i „Game view“ prikazuju događanja u igri, ali su fundamentalno različiti. Prozor „Scene view“ omogućava vizualnu reprezentaciju pozicije objekata u igri i njihovo pomicanje. Prozor „Game view“ omogućava

isprobavanje igre u stvarnome vremenu bez da se igra sastavi (eng. *compile*) i pokreće zasebno svaki put.

Prozor „Project“ na dnu zaslona služi za interakciju, stvaranje, podešavanje i prikazivanje svih datoteka i mapa koje se nalaze unutar projekta. Preko njega se uvoze (eng. *importing*) resursi koji se koriste za kreiranje objekata unutar igre i okoline u kojoj se igrač nalazi. Preporučena podjela mapa unutar projekta je na glavne kategorije: *scripts*, *sprites*, *animations*, *tiles* i *scenes*. Unutar svake mape nalaze se pripadajući resursi, kao npr. unutar *sprites* mape nalaze se *Sprites*. Sprites su dvodimenzionalne slike u sceni i koriste se za prikazivanje objekta unutar igre [30].

Prozor „Console“ nalazi se gdje i prozor „Project“ i on prikazuje postoji li ikakva pogreška u projektu. Pogreška može biti samo upozorenje, ali također i problem u zapisanim skriptama koji softver Unity prepoznaje i sprječava sastavljanje igre dok se problem ne razriješi.

Prozor „Inspector“ pruža mogućnost interakcije i podešavanja postavki svakog objekta. Također omogućava i nadgledanje promjena nekog objekta prilikom igranja što olakšava utvrđivanje mogućih pogrešaka i problema.

3.3 Stvaranje lika igrača – lik *player*

Velik postotak igara dostupnih na tržištu sadrži jednog ili više glavnih likova preko kojih igraču omogućuju direktan utjecaj na ishod igre i potpuni doživljaj njihove priče (npr. protagonist igre „Assassins Creed“ - Ezio Auditore ili protagonist igre „God of War“ - Kratos). Također, postoje određene igre (npr. „Gothic“ ili „Stalker“) u kojima igrač nije protagonist priče, već samo jedna osoba koja se nalazi u svijetu i koja indirektno utječe na priču i događaje. Zbog navedenog razloga stvara se lik „Player“. Lik *Player* sastoji se od više različitih funkcija, ali glavne od njih su pomicanje po svijetu i interakcija s objektima.

Za stvaranje lika kojeg će igrač kontrolirati potrebno je pronaći resurse koji će se koristiti za prikazivanje lika na zaslonu. Odabrani resursi mogu biti samo jedna slika ili doći u obliku *sprite sheeta*. *Sprite sheet* predstavlja jednu sliku koja u sebi sadržava više manjih slika koje predstavljaju više objekata ili animaciju određenog objekta.

Odabrani *sprite* ili *sprite sheet* potrebno je uvesti u softver Unity. To se čini desnim klikom na prozor „Project“ i pritiskom na „Import New Asset…“. Također, može se vršiti povlačenjem datoteke u prozor „Project“ na dnu zaslona. Uvezeni *sprite sheet* potrebno je razdvojiti na zasebne elemente, čemu softver Unity doskače nudeći svoj alat za uređivanje *spriteova* zvan „Sprite Editor“. Za razdvajanje *spriteova* potrebno je podesiti nekoliko postavki *sprite sheeta*. Pritiskom na *sprite sheet* u prozoru „Project“ otvaraju se postavke u prozoru „Inspector“ s njegove desne strane. Postavka potrebna za podešavanje zove se „Sprite Mode“. Postavka „Sprite Mode*“* određuje ako određeni resurs sadrži samo jednu sliku ili sadrži više slika unutar sebe. Ako resurs sadrži više slika potrebno je prebaciti postavku sa „Single“ u „Multiple“. Prozor „Sprite Editor“ dopušta odabir načina rezanja, koji može biti automatski ili prema veličinama koje se upišu. U prozoru „Project“ na dnu zaslona, ako se proširi izrezani *sprite sheet*, bit će prikazani svi *spriteovi* razdvojeni u svoje zasebne slike. Povlačenjem odabrane slike u scenu kreirat će se zasebni objekt u kojem se nalazi *sprite* koji predstavlja igrača.

Kako bi se igrač mogao pomicati po svijetu potrebno je programirati logiku za pomicanje objekta na zaslonu. Korištena važeća logika glasi da igrač predstavlja točku koordinatnog sustava i za pomicanje u desnu ili lijevu stranu potrebno je točku pomicati po X-osi (pozitivno po osi = desno; negativno po osi = lijevo). Poznata je trenutačna lokacija objekta, no potrebno je izračunati i njegovu novu poziciju, a to se radi pomoću izraza (1):

$$V_n = V_c + V_d * s * \Delta Time \quad (1)$$

za izračun vektora nove pozicije, gdje je: $\mathbf{V_n}$ = vektor nove pozicije, $\mathbf{V_c}$ = vektor sadašnje pozicije, $\mathbf{V_d}$ = vektor smjera kretanja, **s** = brzina kretanja, **Time** = vrijeme. [31]

S ustanovljenom logikom i formulom stvara se novi C# *script file* u mapi „Scripts“ pod nazivom „Movement“. Unutar skripte, u funkciji „Update“, stvaraju se varijable „speed“ i „direction“. Varijabla „speed“ tipa „float“ predstavlja brzinu kretanja objekta, a varijabla „direction“ tipa „Vector3“ predstavlja smjer kretanja objekta. Varijable „horizontal“ i „vertical“ koriste funkciju softvera Unity zvanu „Input.GetAxisRaw()“, a rezultat koji funkcija „vraća“ može biti u vrijednosti od 1 do -1 što predstavlja smjer kretanja na određenoj osi.

Funkcija „FixedUpdate()“ koristi se kada nad objektom treba primjeniti zakonitosti fizike, stoga se unutar te funkcije svrstava formula za kretanje objekta (Slika 5.).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movement : MonoBehaviour
{
    public float speed;
    private Vector3 direction;

    private void Update(){
        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");

        direction = new Vector3(horizontal, vertical);

    }

    public void FixedUpdate(){
        this.transform.position += direction * speed * Time.deltaTime;
    }


}
```

Slika 5. Prikaz skripte „Movement“ (slika zaslona, izradio Sever, L.)

Nakon kreiranja i programiranja skripte potrebno ju je dodijeliti objektu, a to je objekt „Player“, kako bi se on mogao pomicati. Pritiskom na objekt „Player“ u prozoru „Hierarchy“ prikazuju se podaci o objektu u prozoru „Inspector“ unutar kojeg se dodjeljuje skripta pritiskom na gumb „Add Component“.

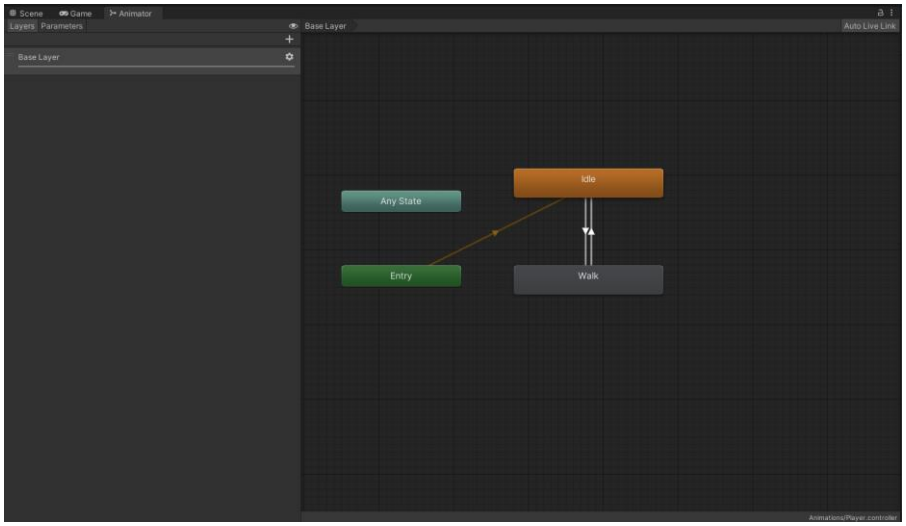
Ispitivanje funkcionalnosti napisane skripte vrši se pritiskom na gumb „Play“ pri vrhu zaslona koji pokreće igru.

3.4 Dodavanje animacije objektu

Dodavanjem pokreta objektu, igraču se omogućuje kretanje po svijetu. Zbog unaprjeđenja tog dojma potrebno je dodati i animacije. Animiranje u 2D igrama veoma je slično starijim crtanim filmovima napravljenih od niza slika koje, kad se spoje, stvaraju dojam glatkih pokreta.

Prije početka izrade animacije potrebno je prvo pronaći *sprite sheet* koji sadrži odgovarajuće slike koje čine animaciju pokreta. Pronalaskom *sprite sheeta* i podešavanjem u *sprite editoru*, odabiru se slike koje čine jednu animaciju (npr. hodanje udesno).

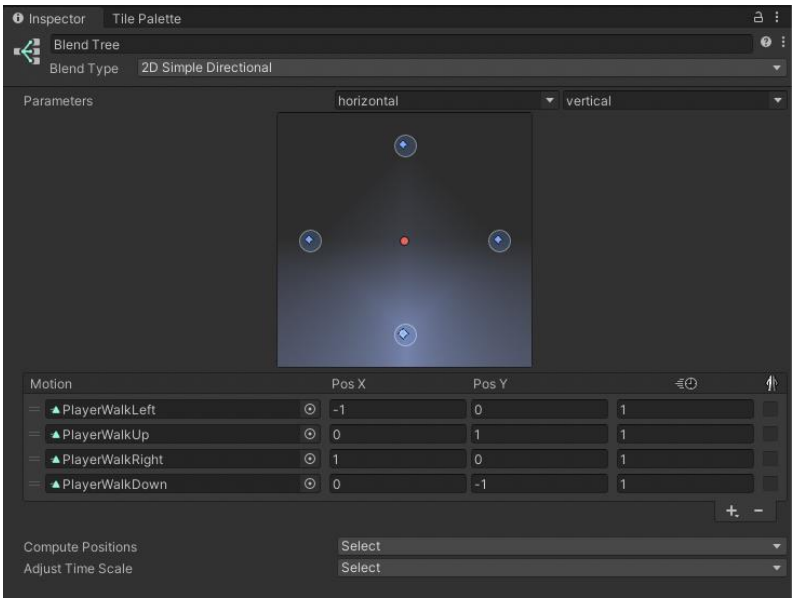
Odabrane slike povlače se u prozor „Hierarchy“ što predstavlja stvaranje animacije. Softver Unity za svaku generiranu animaciju stvara njoj popratni kontroler. Kontroler animacije služi uspostavljanju isječka animacije za određene uvjete, poput hodanja ulijevo kada se promjeni vrijednost smjera kretanja u negativnu vrijednost. Pritiskom na kontroler otvara se novi prozor zvan „Animator“ u kojem se podešava animacija samog objekta (Slika 6.).



Slika 6. Prikaz prozora „Animator“ unutar softvera Unity (slika zaslona, izradio Sever, L.)

U prozoru „Animator“ stvara se novo stanje pod imenom „Walk“ i spaja se sa stanjem „Idle“ koje je prethodno generirano. Tranzicije se stvaraju obostrano jer se igrač može prebacivati iz pozicije „Idle“ u poziciju „Walk“ kad se pomiče. No, kada se prestane pomicati, mora se vratiti u poziciju „Idle“ jer inače animacije traju beskonačno dugo. Otvaranjem prozora „Parameters“ stvaraju se tri nova parametra pod nazivom „isWalking“, „vertical“ i „horizontal“. Parametar „isWalking“ podatkovnog tipa „bool“, gdje vrijednost može biti istinita ili neistinita, predstavlja pomicanje objekta. Pritiskom na „Walk“ otvara se prozor „Blend Tree“ u kojem se postavlja željena animacija za određeni uvjet. Svrha prozora „Blend Tree“ je spajanje više animacija u jednu potpunu animaciju što omogućuje prethodno spomenutu glatkoću pokreta [32]. U prozoru „Inspector“ nalaze se postavke za „Blend Type“.

Te postavke predstavljaju način izračunavanja pokreta, npr. „2D Simple Directional“, koji se koristi kada pokreti predstavljaju različite smjerove, ali ne dopušta više animacija za isti smjer kretanja (Slika 7.).



Slika 7. Primjer postavljene animacije za pomicanje lika (slika zaslona, izradio Sever, L.)

S postavljenim kontrolerom animacije potrebno je pridodati kontroler objektu. Pritiskom na objekt *Player* dodaje se komponenta zvana „Animator“ i kao „Controller“ odabire se konfigurirani kontroler. Dodani kontroler sadrži uvjete za prebacivanje između različitih stanja, no objekt ne može utjecati na stanje varijabli potrebnih za animiranje. Podešavanjem skripte upravljanja kretanjem rješava se problem dostavljanja potrebnih varijabli kontroleru. Unutar postojeće skripte stvara se javna varijabla „animator“ tipa „Animator“. Stvara se nova funkcija pod imenom „AnimateMovement()“ koja prima varijablu „direction“ tipa „Vector3“. Stvorena funkcija provjerava postoji li referenca na komponentu „Animator“ unutar objekta. Zatim provjerava veličinu vektora koji predstavlja pomak objekta. Ako su izvršene provjere istinite, onda se pozivaju funkcije za postavljanje vrijednosti unutar kontrolera animacije. Ako je provjera veličine vektora neistinita, vrijednost za „isWalking“ postavlja se na „false“ što vraća objekt u njegovu mirujuću poziciju (eng. *Idle state)* (Slika 8.).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movement : MonoBehaviour
{
    public float speed;
    public Animator animator;
    private Vector3 direction;

    private void Update(){
        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");

        direction = new Vector3(horizontal, vertical);

        AnimateMovement(direction);

    }

    public void FixedUpdate(){
        this.transform.position += direction * speed * Time.deltaTime;
    }

    void AnimateMovement(Vector3 direction){
        if(animator != null){
            if(direction.magnitude > 0){
                animator.SetBool("isWalking",true);
                animator.SetFloat("horizontal", direction.x);
                animator.SetFloat("vertical", direction.y);
            }else{
                animator.SetBool("isWalking", false);
            }
        }
    }

}
```

Slika 8. Prikaz skripte „Movement“ s pripadajućom animacijom lika (slika zaslona, izradio Sever, L.)

3.5 Izrada okoline unutar igre

Igrač ima mogućnost slobodnog kretanja, ali ne postoji svijet u kojem on tu mogućnost može iskoristiti. Bez svijeta, igrač nema nikakve smjernice za kretanje niti posjeduje informaciju s kojim objektima može vršiti interakciju. Zbog tog razloga potrebno je kreirati okolinu, ili tzv. svijet igre. Prije izrade svijeta valja osmisliti raspored (eng. *layout*) svijeta koji se planira izraditi, a za izradu je moguće koristiti papir ili software za crtanje, kao npr. Adobe Photoshop, CorelPainter, Microsoft Paint i slično. Inicijalni dizajn znatno olakšava izradu svijeta pošto se prije same kreacije usklađuju nove ideje s već postojećim dizajnima i tako sprječava neusklađenost svijeta s igrom.

Za početak izrade potrebno je stvoriti novi 2D objekt zvan „Tilemap“, koji čini komponentu koja pohranjuje i obrađuje ploče (eng. *tiles),* u smislu da prenosi potrebne informacije od ploča na zaslonu do drugih komponenti [33]. Stvaranjem *tilemape* u prozoru „Hierarchy“ stvara se objekt rešetke (eng. *grid)* u kojem se nalazi novo stvoreni *tilemap*. Rešetka dopušta programeru lakše postavljanje i usklađivanje pozicije ploča na zaslonu [34].

Kao pod-objekt rešetke postavljen je stvoreni „tilemap“ kojeg je potrebno duplicirati. Duplikacijom objekta „tilemap“ razdvaja se pozadina u dva sloja: „Background“ i „Ground“. Pozadinski (eng. *background)* sloj služi popunjavanju rupa i neusklađenosti u svijetu, a sloj zemlja (eng. *ground*) stvaranju puteva, planina, kuća i ostalih objekata koji se nalaze u sceni. Za stvaranje paleta ploča (eng. *tile pallets)* potrebni su resursi koji najbolje ispunjavaju ideju osmišljenog svijeta. Otvaranjem prozora „Tile Pallete“ i pritiskom na gumb „Create New Pallete“ otvara se prozor za stvaranje nove palete ploča. Paleta ploča (eng. *tile pallete)* služi kao mjesto pronalaska i spremanja svih ploča koje se mogu koristiti u sceni. Pritiskom na gumb „Create“ otvara se prozor gdje se odabire mjesto spremanja paleta. Povlačenjem resursa iz prozora „Project“ unutar prozora „Tile Pallete“, prebačeni resursi se pretvaraju u ploče koje se spremaju u odabranu datoteku unutar projekta.

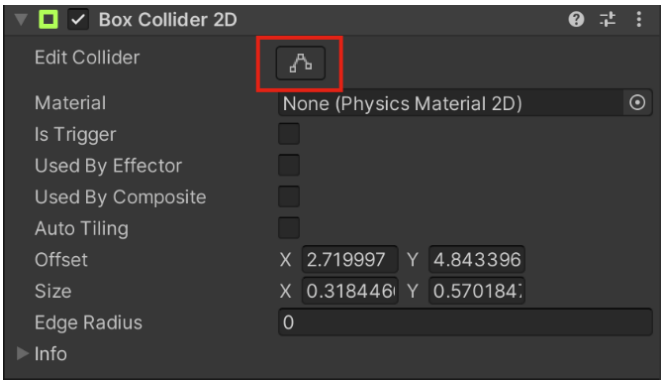
Odabirom ploče na paleti i pritiskom rešetke u sceni postavlja se ploča na određenu poziciju. Spajanjem više ploča u jednu cjelinu stvara se pozadina i okolina igre (Slika 9.).

Slika 9. Primjer kreirane razine ili *levela* unutar igre (slika zaslona, izradio Sever, L.)

Hodajući svijetom, do izražaja dolazi da kroz stvorene planine, kuće i slično, igrač može slobodno prolaziti.

Razlog tomu je što objekti ne sadrže komponentu koja procesira dodire između njih. *Collider* je način na koji Unity prepoznaje i upravlja dodirima između objekata.

Prvo se programiraju zakonitosti fizike koji djeluju na igrača. To se čini dodjelom komponente „Rigidbody 2D“. Zajedno s komponentom „Rigidbody 2D“ treba se dodati i komponenta „Box Collider 2D“ koja će definirati područje oko igrača ili takozvani *hitbox*. Softver Unity sam procjenjuje veličinu objekta i po njima kreira *hitbox* definiran zelenim rubovima. Ako se zeleni rubovi ne podudaraju s likom, podešavaju se pritiskom na gumb sa slike 10.



Slika *10*. Prikaz gumba za podešavanje područja oko igrača – hitbox (slika zaslona, izradio Sever, L.)

Za dodavanje granica unutar svijeta i rubova brda potrebno je stvoriti novi objekt zvan „TerrainCollider“ unutar kojeg će se stvoriti pod-objekti za svako područje u kojem se želi ograničiti kretanje objekta. U kontekstu projekta igrač ne smije prolaziti kroz kuću i penjati se uzbrdo ako ne koristi stepenice. Pošto je kuća interaktivna, kreira se kao poseban objekt, a ne kao dio pozadine i dodaje joj se komponenta „Polygon Collider 2D“. Za ostale dodane površine kao pod-objekte objekta „TerrainCollider“ dodaje se komponenta „Edge Collider 2D“ i prilagođava tako da je na rubovima površina igrač ograničen, dok se po ostatku svijeta može slobodno kretati.

3.6. Stvaranje objekta kojeg igrač može pokupiti

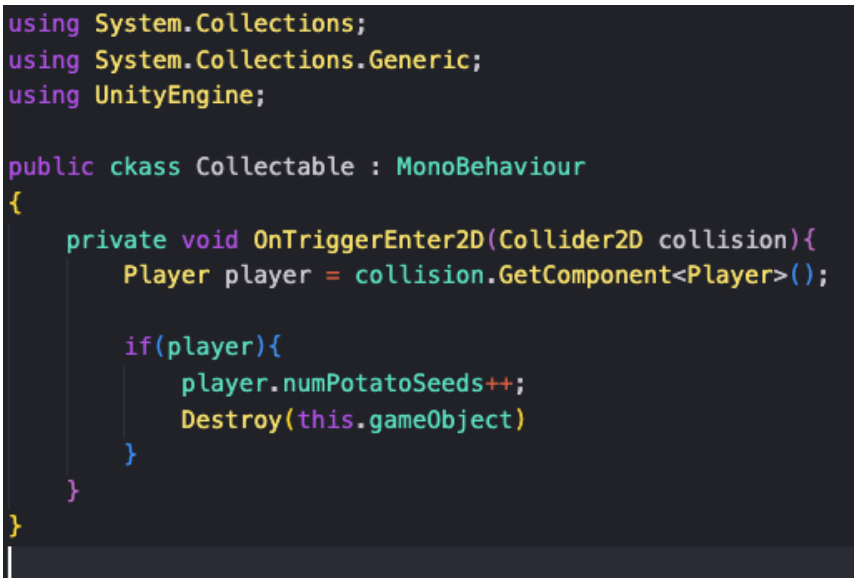
Kako bi igrač mogao ispunjavati svoju funkciju treba imati mogućnost uzimanja stavki postavljenih u svijetu, kao npr. sjemenke krumpira *(*eng. *potato seeds)*. Objekti koje igrač može pokupiti nazivaju se *collectables.*

Prije početka izrade potrebno je pronaći odgovarajuće resurse koji predstavljaju stavku izrade. Odgovarajuće resurse potrebno je podesiti i, ako je potrebno, izrezati u zasebne slike. Povlačenjem odabranog resursa unutar prozora „Scene“, stvara se objekt unutar igre koji predstavlja stavku. Stvoreni objekt ne posjeduje komponentu koja prepoznaje trenutak kada objekt igrača prođe preko njega, stoga je potrebno dodati komponentu pod imenom „Box Collider 2D“. Dodana komponenta posjeduje mogućnost okidanja (eng. *trigger)* ako igrač dođe u kontakt sa stavkom, što omogućava izvršenje odabrane skripte (Slika 11.).

Slika 4. Prikaz stvorene stavke s komponentom "Box Collider 2D" (slika zaslona, izradio Sever, L.)

Potrebno je stvoriti skriptu pod imenom „Collectable“ koja se dodjeljuje objektu stavke kako bi se ona mogla pokupiti. Unutar stvorene skripte stvara se privatna funkcija pod imenom „OnTriggerEnter2D()“ koja prima varijablu „collision“ tipa „Collider2D“. Stvorena funkcija služi prepoznavanju ulaska objekta unutar polja okidača [36]. Funkcija nije sposobna ustanoviti je li objekt igrača ušao unutar okidača, stoga je potrebno stvoriti novu skriptu pod imenom „Player“ koja se dodaje objektu igrača. Unutar funkcije „OnTriggerEnter2D()“, unutar skripte „Collectable“, stvara se lokalna varijabla pod imenom „player“ tipa „Player“ koja provjerava je li se interakcija uistinu dogodila između objekta igrača i stavke (Slika 12.).

U slučaju da je varijabla „player“ istinita, vrijednost unutar objekta igrača povećava se za jedan. Metoda pod imenom „Destory“ koja prima referencu na objekt stavke uništava sami objekt, što predstavlja sakupljanje.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public ckass Collectable : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision){
        Player player = collision.GetComponent<Player>();

        if(player){
            player.numPotatoSeeds++;
            Destroy(this.gameObject)
        }
    }
}
```

Slika 5. Prikaz skripte „Collectable“ (slika zaslona, izradio Sever, L.)

3.6 Spremanje skupljenih stavki u spremište

Stvorene stavke koje igrač sakuplja ne spremaju se, nego povećavaju vrijednost jedne varijable unutar objekta igrača. Spremanje stavki unutar spremišta (eng. *inventory)* igrača pruža mogućnost spremanja raznih postavljenih objekata unutar igre s određenim ograničenjima (npr. broj identičnih stavki). Spremište igrača sastoji se od jednog ili više utora koji mogu sadržavati objekte. Objekt stavke nakon sakupljanja sprema se u utor unutar spremišta. Izrada spremišta započinje stvaranjem skripte pod imenom „Inventory“. Unutar klase „Inventory“ stvara se nova javna klasa pod imenom „Slot“ koja predstavlja jedan utor. Klasa „Slot“ sadrži javne varijable „count“ tipa „int“ i „maxAllowed“ tipa „int“. Varijabla „count“ služi numeričkoj reprezentaciji broja stavki unutar utora.

Varijabla „maxAllowed“ služi za određivanje maksimalnog broja istih stavki unutar jednog utora. Kako bi utor znao koja vrsta stavke se nalazi na toj poziciji potreban je način identifikacije objekta skupljene stavke. Unutar skripte „Collectable“ stvara se javni enum pod nazivom „CollectableType“. Enum je vrsta vrijednosti koja je definirana brojem konstanti kojima je pridodana numerička vrijednost [38].

Primjer takve vrijednosti bila bi vrijednost „NONE“ pridodana vrijednost „0“. Unutar enuma „CollectableType“ postavljaju se vrijednosti mogućih stavki unutar igre.

Zatim se unutar klase „Collectable“ stvara javna varijabla tipa „CollectableType“ pod imenom „type“. Varijabla „type“ služi dodjeljivanju vrste stavke objektu stavke (npr. sjemenke krumpira ili sjemenke mrkve) (Slika 13.).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Collectable : MonoBehaviour
{

    public CollectableType type;
    void OnTriggerEnter2D(Collider2D collision){
    Player player = collision.GetComponent<Player>();

    if(player){
        player.inventory.Add(this);
        Destroy(this.gameObject);
    }
   }
}

public enum CollectableType{
    NONE, POTATO_SEED
}
```

```
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class Inventory {
7      public class Slot{
8          public CollectableType type;
9          public int count;
10         public int maxAllowed;
11     }
12     public List<Slot> slots = new List<Slot>();
13 }
```

Slika 6. Prikaz skripte „Collectable“(lijevo) i skripte „Inventory“ (desno) (slika zaslona, izradio Sever, L.)

Stvorena lista pod imenom „slots“ služi za pohranu stvorenih utora spremišta. Stvara se javni konstruktor pod imenom „Inventory“ koji prima parametar tipa „int“ pod imenom „numSlots“ koji služi za postavljanje utora spremišta.

Unutar konstruktora stvara se petlja koja se ponavlja onoliko puta koliko je određeno varijablom „numSlots“. Unutar petlje stvara se *instance* utora, te se stvoreni utor dodaje listi utora. Unutar javne klase „Slot“ stvara se javni konstruktor pod imenom „Slot“ koji služi postavljanju početnih vrijednosti unutar utora, poput vrste stavke unutar utora. Unutar skripte „Player“ stvara se nova varijabla tipa „Inventory“ pod imenom „invenotry“ koja igraču omogućava pristup inventaru. Stvara se privatna funkcija pod imenom „Awake“. Funkcija „Awake“ se u softveru Unity poziva kada se objekt unutar igre, koji sadrži skriptu, učita na scenu prije nego što se aplikacija pokrene [39]. Unutar funkcije „Awake“, varijabli pod imenom „inventory“ dodjeljuje se instanca inventara s odabranim brojem utora.

3.7 Vizualni prikaz inventara

Igrač skupljanjem stavki unutar igre zapaža nestanak objekata na zaslonu, ali ne posjeduje način vizualne reprezentacije skupljenih stavki. Vizualna reprezentacija spremišta igrača ostvaruje se stvaranjem *user interface* (**UI**) elementa koji sadrži prozor sa svim skupljenim stavkama. *User Interface* unutar igre omogućuje igraču izvršavanje zadataka unutar igre, poput provjeravanja spremišta kroz izravnu interakciju sa sučeljem, ili prikaza informacija na zaslonu [40].

Unutar prozora „Hierarchy“ stvara se UI objekt pod imenom „Canvas“ unutar kojega se stvara prazni objekt pod imenom „Inventory“. Unutar objekta „Inventory“ stvara se UI objekt vrste „Panel“ pod imenom „Background“. Pritiskom na stvoreni objekt otvara se prozor koji omogućuje podešavanje izgleda UI sučelja. Unutar objekta „Background“ stvara se prazni objekt pod imenom „Slots“ unutar kojeg se stvaraju UI objekti vrste „Image“ pod imenom „Slot“. Objekt „Slot“ predstavlja jedan utor u sučelju koji sadrži stavku. Unutar objekta „Slot“ stvaraju se UI objekti vrste „Image“ pod imenom „Icon“ i „TextMeshPro“ pod imenom „Quantity“ (Slika 14.).

Slika 7. Prikaz UI sučelja spremišta unutar (slika zaslona, izradio Sever, L.)

Stvoreno UI sučelje ne posjeduje način otvaranja (prikazivanja na zaslonu), stoga se stvaranjem skripte pod nazivom „Inventory_UI“ omogućuje programiranje logike UI elementa.

Unutar skripte „Invenotry_UI“ stvara se varijabla tipa „GameObject“ pod imenom „inventoryPanel“. Zatim se izvršava provjera unutar funkcije „Update“ u svrhu provjere određene tipke na tipkovnici, u kontekstu projekta tipka „Tab“. Ako je provjera istinita, izvršava se funkcija pod imenom „ToggleInventory“ koja otvara prozor spremišta unutar igre. Stvara se javna funkcija pod imenom „ToggleInventory“ unutar koje se obavlja provjera postojanja aktivnog sučelja na zaslonu. Stvorena skripta dodjeljuje se objektu pod imenom „Inventory“ unutar objekta „Canvas“.

UI sučelje posjeduje mogućnost otvaranja i zatvaranja pritiskom na određenu tipku, ali ne posjeduje mogućnost prikaza stavki unutar spremišta. Unutar skripte „Invenotry_UI“ stvara se javna varijabla tipa „Player“ pod imenom „Player“, te funkcija pod imenom „Setup“. Stvara se nova skripta pod imenom „Slot_UI“. Unutar skripte „Slot_UI“ stvaraju se javne varijable tipa „Image“ pod imenom „itemIcon“ i „TextMeshProUGUI“ pod imenom „quantityText“. Stvara se javna funkcija pod imenom „SetItem“ koja prima parametar vrste „Inventory.Slot“ pod imenom „slot“.

Unutar funkcije obavlja se provjera vrijednosti varijable „slot“. Ako je istinita, vrijednosti unutar utora postavljaju se na vrijednosti koje se nalaze u spremištu igrača.

Unutar skripte „Inventory_UI“ stvara se lista tipa „Slot_UI“ pod imenom „slots“ koja predstavlja broj utora koji se nalaze na sučelju inventara. Unutar funkcije „Setup“ izvršava se provjera podudaraju li se brojevi utora sa sučeljem spremišta. Ako je provjera istinita, izvršava se petlja koja prolazi kroz utore u sučelju spremišta (Slika 15.).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class Slot_UI : MonoBehaviour
{
    public Image itemIcon;
    public TextMeshProUGUI quantityText;

    public void SetItem(Inventory.Slot slot){
        if (slot != null)
        {
            itemIcon.sprite = slot.icon;
            itemIcon.color = new Color(1,1,1,1);
            quantityText.text = slot.count.ToString();

        }
    }
    public void SetEmpty(){
        itemIcon.sprite = null;
        itemIcon.color = new Color(1,1,1,0);
        quantityText.text = "";
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Inventory_UI : MonoBehaviour
{
    public GameObject inventoryPannel;
    public Player player;
    public List<Slot_UI> slots = new List<Slot_UI>();

    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Tab)){
            ToggleInvenotry();
        }
    }

    public void ToggleInvenotry(){
            if(!inventoryPannel.activeSelf){
                inventoryPannel.SetActive(true);
                Setup();
            }else
            {
                inventoryPannel.SetActive(false);
            }
    }

    void Setup(){
        if(slots.Count == player.inventory.slots.Count){
            for(int i=0;i<slots.Count; i++){
                if(player.inventory.slots[i].type != CollectableType.NONE){
                    slots[i].SetItem(player.inventory.slots[i]);
                }else{
                    slots[i].SetEmpty();
                }
            }
        }
    }
}
```

Slika 8. *Prikaz skripte „Slot_UI“ (lijevo) i skripte „Inventory_UI“ (desno) (slika zaslona, izradio Sever, L.)*

Izvršava se provjera unutar petlje ako se unutar igračeva spremišta na istom mjestu nalazi stavka. Ako je provjera istinita, postavlja se određena stavka u odabrani utor na sučelju. Skripta pod imenom „Slot_UI“ dodaje se na objekt pod imenom „Slot“.

4. Zaključak

Softveri za izradu videoigara, poput softvera Unity, znatno olakšavaju izradu igara. Unity se pokazao kao softver koji pruža raznovrsne mogućnosti uz jednostavno i intuitivno sučelje. Velika prednost softvera je što omogućuje programiranje logike u jeziku C# koji je vrlo dobro dokumentiran i kojeg programeri rado koriste. U izradi igre prvo je bila potrebna ideja o vrsti i žanru igre. S odabranim žanrom bilo je potrebno pronaći pripadajuće resurse koji najbolje ostvaruju ideju.

Resursi korišteni za izradu igre pronađeni su na stranicama koje nude besplatne limitirane resurse (npr. internetske stranice itch.io i gameart2d.com).

Kao najzahtjevniji izazov tokom izrade igre svrstava se implementacija nekih od sustava i popravljanje grešaka koje bi taj sustav izazvao. Implementirani sustav sadržavao je nekoliko već stvorenih komponenti i poneke je bilo potrebno ponovno napisati i prepravljati kako bi uspješno funkcionirale.

Način na koji se igra stvorena u svrhu ovog rada može poboljšati je dodavanjem zvukova za određene događaje, kao npr. zvukovi pri hodanju igrača po svijetu.

Izrada igara pomoću softvera, kao softver Unity, znatno je jednostavnija, ali i dalje zahtjeva mnogo pažnje i vremena. Kroz izradu igre shvaća se važnost postojanja kolega i timova, podjele posla i razmjene ideja, sve u svrhu postizanja što boljeg krajnjeg proizvoda. Inače, kada bi pojedinac sve uloge preuzeo na sebe, izrada bi bila otežana. Isto tako, proizvod bi teško mogao biti komercijaliziran jer bi za njegovu kvalitetnu izradu bilo potrebno mnogo više vremena i resursa.

5. Literatura

[1] https://hjp.znanje.hr/index.php?show=search_by_id&id=f19mXBR%2F&keyword=teleprinter (Pristupljeno 29.8.2022)

[2] https://www.popularmechanics.com/culture/gaming/a20129/the-very-first-video-game/ (Pristupljeno 29.8.2022)

[3] https://www.history.com/news/in-1950-alan-turing-created-a-chess-computer-program-that-prefigured-a-i (Pristupljeno (29.8.2022)

[4] https://news.elearninginside.com/how-plato-changed-the-world-in-1960/ (Pristupljeno 30.8.2022)

[5] Ritchie, D. M. (1983). C reference manual. Programming Languages (Pristupljeno 30.8.2022)

[6] http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html (Pristupljeno 31.8.2022)

[7] Williams, G. (1981). New Games New Directions. *Byte*. 06-12, 8 (Pristupljeno 31.8.2022)

[8] Clark, P. (1982). The Play's the Thing. *Byte*. 07-12, 8 (Pristupljeno 31.8.2022)

[9] Hornby, T. (2010). Acorn and the BBC micro: From education to obscurity. (Pristupljeno 31.8.2022)

[10] https://www.gameinformer.com/b/features/archive/2011/12/06/ubi-uncensored.aspx (Pristupljeno 31.8.2022)

[11] https://www.theverge.com/2013/1/16/3740422/the-life-and-death-of-the-american-arcade-for-amusement-only (Pristupljeno 31.8.2022)

[12] Aoyama, Y., & Izushi, H. (2003). Hardware gimmick or cultural innovation? Technological, cultural, and social foundations of the Japanese video game industry. *Research policy*, *32*(3), 423-444. (Pristupljeno 31.8.2022)

[13] https://hmn.wiki/bs/Parallax_mapping (Pristupljeno 31.8.2022)

[14] Pile, J. (2013). *2D graphics programming for games*. CRC Press. (Pristupljeno 01.9.2022)

[15] https://www.arcade-history.com/?n=virtua-fighter&page=detail&id=3298 (Pristupljeno 01.9.2022)

[16] https://www.arm.com/glossary/gaming-engines (Pristupljeno 01.9.2022)

[17] https://www.techradar.com/news/gaming/the-evolution-of-3d-games-700995/2 (Pristupljeno 01.9.2022)

[18] Tripp, S., Grueber, M., Simkins, J., & Yetter, D. (2020). Video Games in the 21st Century: The 2020 Impact Report. (Pristupljeno 01.9.2022)

[19] Bernevega, A., Gekker, A. (2022). The Industry of Landlords: Exploring the Assetization of the Triple-A Game. Games and Culture, 17(1), 47-69. (Pristupljeno 01.9.2022)

[20] https://www.polygon.com/a/final-fantasy-7 (Pristupljeno 01.9.2022)

[21] https://www.eurogamer.net/what-is-indie (Pristupljeno 01.9.2022)

[22] https://www.escapistmagazine.com/the-medal-of-honor-curse/ (Pristupljeno 01.9.2022)

[23] https://www.techradar.com/news/gaming/is-indie-gaming-the-future-716500 (Pristupljeno 01.9.2022)

[24] https://appleinsider.com/articles/18/07/10/the-revolution-steve-jobs-resisted-apples-app-store-marks-10-years-of-third-party-innovation (Pristupljeno 02.9.2022)

[25] https://www.theverge.com/2012/3/6/2848223/google-play-store-rebranded-android-market (Pristupljeno 02.9.2022)

[26] https://www.forbes.com/sites/oliverchiang/2010/12/29/10-years-of-popcap-games-beyond-bejeweled/?sh=15b7e34723a4 (Pristupljeno 02.9.2022)

[27] https://techcrunch.com/2013/06/12/king-quits-advertising-since-it-earns-so-much-on-candy-crush-purchases/ (Pristupljeno 02.9.2022)

[28] https://asmedigitalcollection.asme.org/computingengineering/article/17/3/031013/370980/A-Review-of-the-Capabilities-of-Current-Low-Cost (Pristupljeno 02.9.2022)

[29] https://www.theverge.com/2015/12/8/9873840/htc-vive-headset-delay-april-2016/ (Pristupljeno 02.9.2022)

[30] https://remarkablecoder.com/sprites-in-games/ (Pristupljeno 05.9.2022)

[31] https://www.gamedeveloper.com/programming/movement-prediction (Pristupljeno 30.8.2022)

[32] https://docs.unity3d.com/Manual/class-BlendTree.html (Pristupljeno 1.9.2022)

[33] https://docs.unity3d.com/Manual/class-Tilemap.html (Pristupljeno 1.9.2022)
[34] https://docs.unity3d.com/Manual/class-Grid.html (Pristupljeno 1.9.2022)
[35]
[36] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html (Pristupljeno 3.9.2022)
[37]
[38] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum (Pristupljeno 3.9.2022)
[39] https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html (Pristupljeno 4.9.2022)
[40] https://ecampusontario.pressbooks.pub/gamedesigndevelopmenttextbook/chapter/user-interface/ (Pristupljeno 4.9.2022)

Sažetak

Cilj ovog rada bilo je upoznavanje s poviješću izrade videoigara i pokušaj izrade vlastite igre koristeći se modernim tehnikama.

Povijesni dio rada objašnjava najbitnija događanja unutar industrije videoigara. Također, objašnjava neke od tehnika korištenih u izradi videoigara tokom dvadesetog stoljeća.

Glavni dio rada objašnjava ideju igre kreirane unutar samostalnog rada i opisuje izradu igre koristeći softver Unity. Također je opisan i dokumentiran tok izrade video igre s objašnjenjima varijabli, objekata, funkcija i metoda, te njihovim korištenjem.

Ključne riječi: Unity, objekt, funkcija, varijabla, resurs, skripta

Abstract

The purpose of this bachelors thesis was to get to know the history of video game development, and an attempt to create a video game utilizing modern techniques.

The historical part of this work explains the most significant events in the video game industry. Also, it explains some of the techniques utilized during the making of video games in the twentieth century.

The main part of this work explains the idea behind the game created and further explains the making of the game using the software Unity. Furthermore, the making of the game is explained and documented in minute detail, including variables, objects, functions and methods and how they function in the context of this project.

Keyword: Unity, object, function, variable, asset, script