Izrada simulacijske računalne igre preživljavanja u alatu Unity

Baštek, Boris

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Boris Baštek

IZRADA SIMULACIJSKE RAČUNALNE IGRE PREŽIVLJAVANJA U ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Boris Baštek

Matični broj: 0016135501

Studij: Poslovni sustavi

IZRADA SIMULACIJSKE RAČUNALNE IGRE PREŽIVLJAVANJA U ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Mario Konecki

Varaždin, rujan 2022.

Boris Baštek

Izjava o izvornosti

Izjavljujem da je moj završni 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 će biti obrađena tema izrade simulacijske računalne igre preživljavanja u alatu Unity. U radu će biti prikazan proces izrade igre od početka, koristeći dostupne i besplatne pakete sa Unity asset store-a. Izrada videoigre je kompleksan proces, osim programiranja uključuje i matematiku, fiziku, obradu zvuka, animacije itd. Počevši s osnovnim konceptima i postupnim nadograđivanjem cilj je obuhvatiti postupke koje je potrebno provesti kako bi se riješio određeni problem i kako bi se implementirala određena značajka u samu igru. Na kraju rada cilj je imati jednostavnu igru koja sadržava i demonstrira neke osnovne koncepte i sustave igre preživljavanja ali nudi i mogućnost daljnjeg proširivanja i nadogradnje postojećih sustava i značajki.

Ključne riječi: Unity, igra preživljavanja, 3D, akcijska igra, C#

Sadržaj

1. Uvod

Izrada video igre je često kompleksan proces, pogotovo za početnike koji nemaju iskustva. Danas se najčešće u industriji koristi program za razvoj video igara (eng. *game engine*) od kojih su trenutno najpoznatiji i Unity i Unreal 5. U ovom radu koristit će se Unity. Razvoj video igre u početku zahtjeva puno strpljenja, istraživanja i učenja kako bi se upoznao game engine i njegove mogućnosti a najčešće je najbolji put učenja kroz pokušaje i pogreške. Ovaj rad bavit će se prikazivanjem procesa razvoja video igre preživljavanja od početka do određene bazične razine na koju se kasnije može dodavati još sustava koji bi igru dodatno poboljšali.

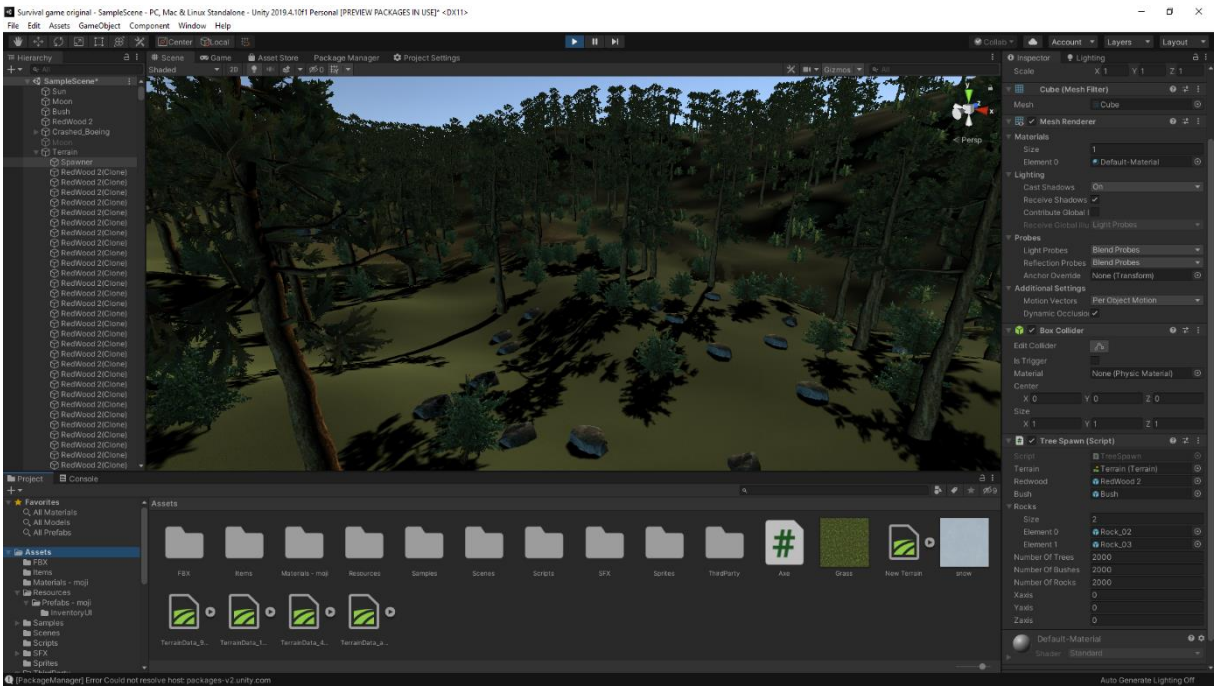
Cilj rada je izraditi igru preživljavanja koristeći besplatne alate, modele i pakete uz samostalnu izradu nekih modela. Igra je rađena u prvom licu i vodi se idejama i mehanikama karakterističnim za igre preživljavanja. Kao uzor i inspiraciju uzeo sam igru „Minecraft“ koja je objavljena 2011. godine koja je uvelike popularizirala žanr preživljavanja ali i postavila temelje za razvoj novih naslova a i dan danas slovi kao jedna od najpopularnijih igrara u žanru. Igru je prvotno razvio jedan developer, Markus Persson poznat i pod nazivom „Notch“ [1]. Sličan trend se nastavio i sa ostalim igrama u žanru te često vidimo da ih razvijaju manji timovi. Neke od igara koje spadaju u tu kategoriju su: „The Forest“, „Scum“ i „Valheim“. Razvoj video igara me je uvijek zanimao i glavna inspiracija za ovu temu mi je bilo upravo to da jedna osoba ili čak manji tim može razviti igru preživljavanja te odlučio sam s ovim radom istražiti to područje i razviti svoju igru.

2. Metode i tehnike rada

Razvoj igre rađen je unutar programskog alata za razvoj igara Unity. Koristeći programsko okruženje Microsoft Visual Studio 2019 i programski jezik C# programirane su funkcionalnosti igre koje su se onda povezivale sa 3D objektima unutar Unity-a. Za potrebe igre korišteni su i određeni besplatni paketi sa službene Unity trgovine. U manjem obujmu, za razvoj 3D modela potrebnih za igru, korišten je programski alat Blender. Za izradu prilagođenih grafičkih elemenata korišten je Gimp koji će ukratko biti obrađen u nastavku kao i ostali korišteni alati.

2.1. Unity

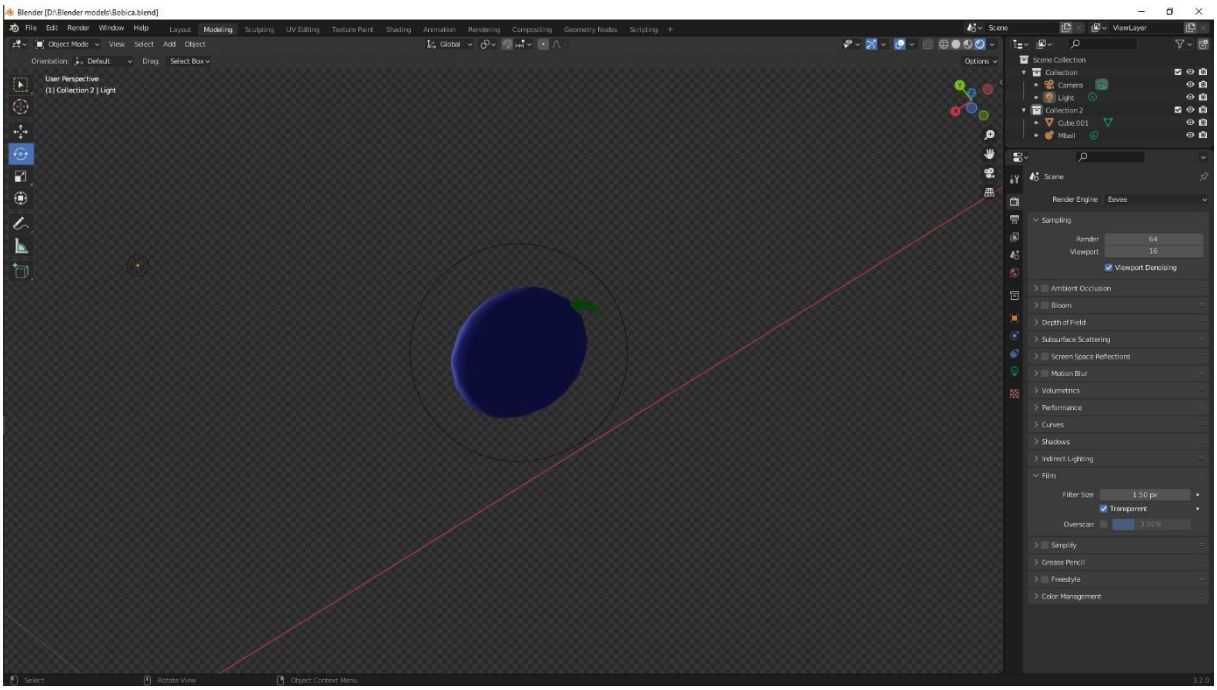
Unity je razvojni programski alat koji osim za izradu video igara omogućava korištenje u arhitekturi, auto i filmskoj industriji. Za razvoj video igara koristi programski jezik C#, što je bio jedan od bitnijih kriterija prilikom odabira ove teme. Pogodnosti koje dolaze uz Unity su između ostalog i velika zajednica kreatora koji putem već spomenute Unity trgovine (eng. Unity Asset Store) objavljuju ali i prodaju svoje radove u obliku paketa. Ti paketi su uglavnom kolekcije objekata ili kompletnih sustava npr. sustav dijaloga, sustav kontrole vremena, paketi 3D modela određene tematike (srednjovjekovni alati, dvorci, životinje) itd. Veliki dio tih paketa je besplatan što nudi priliku za kreativnost i brzi razvoj prototipa. Moguć je razvoj igara za razne platforme (stolna računala, konzole, mobilne) što korisnik može odabrati prilikom kreiranja novog projekta isto kao i hoće li razvijati 2D ili 3D igru. Osim toga Unity nudi i neke već gotove jednostavne projekte kako bi se početnici upoznali sa programskim alatom a u tu svrhu na internetu postoji veliki broj instruktivnih materijala koji također olakšavaju razvoj i učenje samoga alata. Najveći dio razvoja igre će se odvijati upravo u Unity-u te krećem sa praznim projektom.



Slika 1: Unity

2.2. Blender

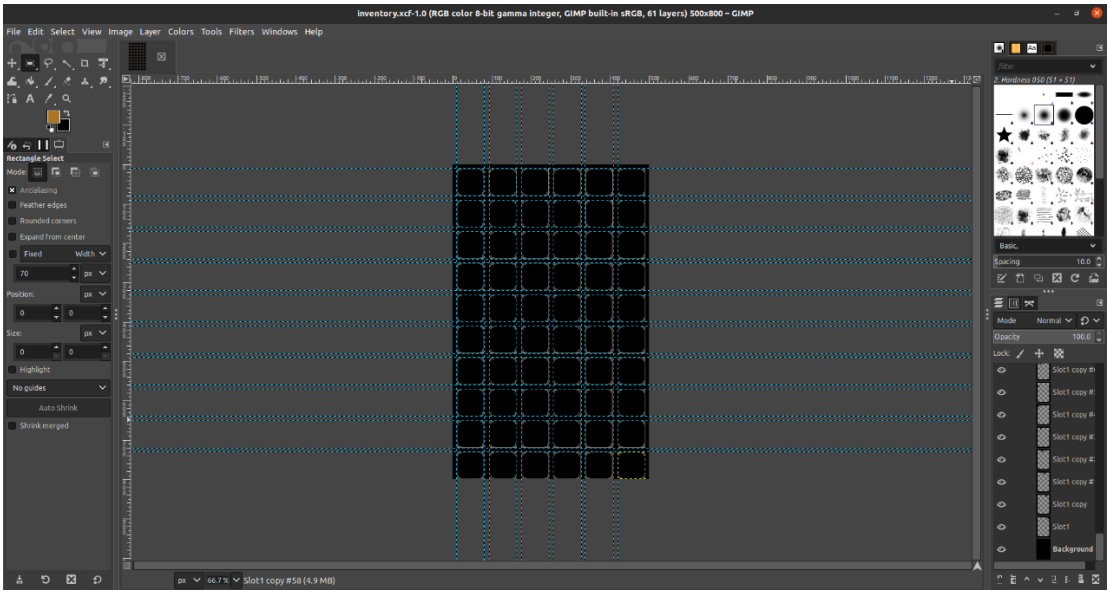
Blender je besplatni programski alat za izradu 3D modela te animiranje te je otvorenog koda (eng. open source). Korisnici mogu pristupiti kodu te ga izmijeniti i dodatno prilagoditi pa tako i izraditi pomoćne alate, doprinijeti razvoju, prijaviti greške te ih sami i ispraviti. U ovom radu blender je korišten za izradu jednostavnijih modela za predstavljanje objekata u igri te njihovo prikazivanje kao stavki u inventaru. Za primjer je uzet model bobice koju igrač može pokupiti nakon interakcije sa grmom. Korištenjem blendera moguće je kreirati sve potrebne modele za igru, što veći studiji uglavnom rade kako bi izbjegli plaćanje licenci i ovisnost o vanjskim paketima ali i kako bi mogli ponovno koristiti te modele i po potrebi ih prilagoditi. Osim 3D modela u blenderu je moguće kreirati animacije, provoditi simulacije, uređivati video itd. Zbog toga je Blender pogodan i za izradu animacija, promotivnih materijala pa čak i animiranih filmova. Ipak, u ovom radu fokus će biti samo na izradu pojedinih 3D modela dok će potrebne animacije biti implementirane kroz Unity.



Slika 2: Blender: Model bobice

2.3. Gimp

Gimp (GNU Image Manipulation Program) je besplatni programski alat otvorenog koda za uređivanje fotografija. Kao i Blender, dostupan je na Linux, Windows i Mac operacijskim sustavima. Isto tako nudi široki raspon mogućnosti te podržava velik broj dodataka ali i omogućava korisnicima da sami isprogramiraju dodatke i skripte te ga tako nadograde s dodatnim funkcionalnostima. Gimp je u ovom projektu korišten za izradu grafike za inventar.



Slika 3: Gimp

3. Razrada teme

U ovom poglavlju prikazan je proces izrade i razvoja igre i njezinih dijelova. Počevši od pripreme projekta i izrade terena ili „svijeta“ u kojem se igra odvija, bit će opisane mehanike igre kao što su kretnja igrača i njegove mogućnosti u igri, sustavi i skripte koje su korištene za implementiranje tih sustava i mehanika.

3.1. Priprema projekta

Za razvoj igre koristio sam Unity verziju 2019.4.10f1 koja je u vrijeme preuzimanja i početka rada na projektu bila najnovija. Kreiranjem novog projekta kreirana je glavna scena te osnovni elementi scene: „Main Camera“ i „Directional Light“. Kreirana scena će poslužiti kao glavna scena odnosno scena u koju će biti smještena igra. Ostale scene koje će poslužiti za izradu „Main menu“ i „Game Over“ ekrana bit će kreirane naknadno. Na početku je dobro odrediti neke pakete koji će se koristiti pa u tu svrhu sa Unity asset store stranice u projekt uključujem tri paketa:

- Standard Assets koji sadrži modele za vodu
- Dream Forest Tree koji sadrži nekoliko različitih modela stabala i grmova
- Rock Package koji sadži modele kamenja i stijena

Modeli iz navedenih paketa bit će sastavni elementi okruženja i prirode u igri.

3.2. Izrada terena

Izrada terena ili svijeta u kojem će se odvijati igra je dio koji nije potrebno odmah kreirati. Ipak, zbog nekih stvari kao npr. skaliranja i dobivanja osjeta za veličinu objekata i „svijeta“ dobro je napraviti barem prototip. Unity ima implementirane alate za kreiranje terena odnosno dodavanje 3D „Terrain“ objekta. Teren je izrađen pomoću alata za teren koji je sastavni dio 3D projekta u Unity-u. U scenu je dodan 3D Terrain objekt koji je onda moguće oblikovati koristeći navedeni alat za uređivanje terena. Kreirani objekt ima dodijeljene „Terrain“ i „Terrain Collider“ komponente. „Terrain“ komponenta omogućava uređivanje terena pomoću spomenutih alata dok „Terrain Collider“ nudi kolizijsku komponentu za detekciju kolizija s drugim objektima. Neki od ponuđenih alata za uređivanje terena su dodavanje novog susjednog terena, kist za bojanje terena, kistovi za dodavanje drveća i vegetacije te postavke terena gdje je moguće promijeniti niz ponuđenih postavki. Kist za bojanje je samo jedna od

opcija za uređivanje ali untar te opcije ponuđene su mogućnosti kao što su podizanje ili spuštanje terena, uglađivanje itd. koje je potrebno posebno odabrati.

3.2.1. Generiranje okoliša

Početna ideja bila je istražiti i implementirati proceduralno generiranje u svrhu kreiranja terena i okoliša kako bi se na početku igre svaki put dobio drugačiji svijet. Ideja je bila inspirirana igrom „Minecraft“ ali s obzirom na kompleksnost odlučio sam se za jednostavniji pristup. Kako bi osigurao da se pri pokretanju igre ipak dobije određeni stupanj raznolikosti odlučio sam napraviti skriptu koja će instancirati objekte i postaviti ih na nasumično mjesto na terenu. Teren je kreiran po postupku opisanom iznad. U skripti „TreeSpawn“ napravljene su tri metode, od kojih je svaka zadužena za instanciranje prefab-a željenog objekta na nasumično generiranoj lokaciji na terenu. Nakon instanciranja, objektu se dodaje pripadajuća komponenta, odnosno skripta koja određuje njegove karakteristike i ponašanje. Tako prilikom instanciranja stabla ono dobija skriptu „Drvo“, koja sadrži logiku i metode koje se odnose na stablo, njegovo ponašanje i interakciju s igračem. Isto vrijedi i za kamenje i grmove. Za generiranje pozicije koristim metodu GeneratedPosition() u kojoj se za vrijednosti X i Z generira nasumičan broj koji leži u rasponu dužine, odnosno širine terena (0-1000). Kako bi svi objekti bili generirani na terenu i ne bi lebdjeli u zraku ili bili „zakopani“ ispod tla, vrijednost Y koordinate se određuje korištenjem SampleHeight metode koja vraća relativnu visinu u odnosu na dani objekt terena. Tako generirana pozicija u obliku Vector3 objekta se vraća i koristi prilikom instanciranja objekata okoliša. U Start() metodi pozivaju se tri metode zadužene za instanciranje okoliša, PlaceTree(), PlaceBush() te PlaceRock() a za svaki objekt moguće je odrediti količinu koju je potrebno instancirati pomoću varijabli numberOfTrees, numberOfBushes i numberOfRocks.

```
Vector3 GeneratedPosition()
    {
        Vector3 temp;
        float x, y, z;
        x = UnityEngine.Random.Range(0, 1000);
        z = UnityEngine.Random.Range(0, 1000);
        temp.x = x;
        temp.y = 0f;
        temp.z = z;
        y = Terrain.activeTerrain.SampleHeight(temp);
        return new Vector3(x, y, z);
    }
```

```
void PlaceTree()
    {
        terrain.drawHeightmap = true;
        print("Terrain size: " + terrain.terrainData.size);
        for (int i = 0; i < numberOfTrees; i++)
        {
            GameObject newTree = Instantiate(Redwood, GeneratedPosition(),
Quaternion.identity, terrain.transform);

            newTree.AddComponent<Drvo>();
            newTree.gameObject.tag = "Tree";
        }
    }
private void PlaceBush()
    {
        terrain.drawHeightmap = true;
        for (int i = 0; i < numberOfBushes; i++)
        {
            GameObject newBush = Instantiate(Bush, GeneratedPosition(),
Quaternion.identity, terrain.transform);
            newBush.AddComponent<Bush>();
        }
    }
void PlaceRock()
    {
        terrain.drawHeightmap = true;
        for (int i = 0; i < numberOfRocks; i++)
        {

            GameObject newRock =
Instantiate(Rocks[UnityEngine.Random.Range(0, 2)], GeneratedPosition(),
Quaternion.identity, terrain.transform);
            newRock.AddComponent<Rock>();
        }

    }
void Start()
    {
        PlaceTree();
        PlaceBush();
        PlaceRock();
    }
```

3.3. Interakcija terena

Kako se radi o igri preživljavanja bitno je da igrač može imati interakciju sa terenom na kojem se nalazi. Iz okoliša će prikupljati resurse koje će koristiti za preživljavanje pa samim time bitno je da teren i stvari u okolišu nude neku vrstu interakcije koja će biti korisna igraču. U poglavlju prije je spomenuta opcija dodavanja drveća i vegetacije, međutim, tu se radi o bojanju drveća i vegetacije s kojima nije moguće ostvariti interakciju. Tako „obojana“ vegetacija je samo dekoracija. Takav način dodavanja okoliša je pogodan u slučaju izrade 2D igre ili oslikavanja dijela terena kojemu igrač neće moći pristupiti ali je i dalje vidljiv u igri. Kako bi vegetacija bila interaktivna potrebno je koristiti 3D objekte te im dodati „Collider“ komponentu i odgovarajuću skriptu koja će definirati njihovo ponašanje. Za primjer ću uzeti stablo. Kako bi igrač mogao srušiti stablo potrebno je da to stalbo na sebi ima (u ovom slučaju) „Capsule Collider“ i skriptu, u ovom slučaju nazvanu „Drvo“. Skripta se brine o tome da drvo ima određeni broj životnih bodova koje igrač oduzima interakcijom, kad se ti bodovi potroše drvo se počinje rušiti i igrač dobije određeni broj cjepanica koje može pokupiti i kasnije koristiti. Sličan postupak je potrebno napraviti za svaki objekt s kojima bi igrač trebao imati nekakvu interakciju. Za tu svrhu kreirana je apstraktna skripta „Interactable“ koju nasljeđuju sve skripte vezane za objekte s kojima bi igrač trebao imati interakciju kao npr. drvo ili kamen. Skripta objektu na koji je dodana dodjeljuje layer odnosno sloj „Interactable“ koji se kasnije koristi za detekciju objekta s kojim je moguće imati interakciju. Skripta definira 3 metode: OnInteract(), OnFocus() i OnLoseFocus() s kojima se za svaki objekt pojedinačno definira što će se dogoditi kada su pozvane.

```
public abstract class Interactable : MonoBehaviour
{
    public virtual void Awake()
    {
        // Any item that inherits Interactable will automatically be given
Interactable layer (layer 9)
        gameObject.layer = 9;
    }
    public abstract void OnInteract();
    public abstract void OnFocus();
    public abstract void OnLoseFocus();
}
```

3.4. Ciklus dana i noći

Za upravljanje svjetlom Unity koristi posebnu komponentu „Light“ koju je potrebno dodijeliti objektu koji će služiti kao izvor svjetlosti. Prilikom kreiranja 3D scene, u hijerarhiji objekata generira se „Directional Light“ objekt koji služi kao sunce. Za ostvarivanje ciklusa dana i noći potrebno je kreirati i objekt za mjesec te mu dodijeliti „Light“ komponentu i u njoj namjestiti svojstva kako bi se dobio efekt mjeseca. Za kontrolu sunca i mjeseca koristit ćemo skriptu „DayNightController“. Postavit ćemo sunce i mjesec tako da leže na kutu od 180 stupnjeva i da rotiraju zajedno istom brzinom. U Update() metodi kalkulirat ćemo prolazak vremena bazirano na duljini dana i množitelju vremena. Rezultat kalkulacije će biti trenutno vrijeme u rasponu od 0 – 1 koje ćemo množiti sa 360 kako bi dobili broj stupnjeva za rotaciju sunca i mjeseca, odnosno položaj u kojem se trebaju nalaziti u trenutnom vremenu.

```
void Start()
    {
        timeOfDay = startingTime;
    }
void Update()
    {
        SunRotation();
        MoonRotation();

        timeOfDay += (Time.deltaTime / dayDuration) * timeMultiplier;
        if(timeOfDay >= 1f)
        {
            timeOfDay = 0f;
        }
    }
void SunRotation()
    {
        // only rotating around X axis (0 and 180 can be changed)
        sun.transform.localRotation = Quaternion.Euler((timeOfDay * 360f) -
0f, 0f, 0f);
    }
void MoonRotation()
    {
        moon.transform.localRotation = Quaternion.Euler((timeOfDay * 360f)
- 180f, 0f, 0f);
    }
```

Slika 4: Dan

Slika 5: Noć

3.5. Igrač

3.5.1. Kontroler kretanja igrača

Unity ima implementiranu komponentu pod nazivom „Character Controller“ koja ima određena implementirana svojstva za kretanje igrača. Moguće je promijeniti postavke kao što su najviši kut strmine uz koju se igrač može penjati, razmak koraka, visina igrača, najmanja duljina pomaka i tako dalje, ali potrebno je samostalno implementirati mehaniku kretnje. Unity nudi i besplatni paket koji sadrži modele, skripte i već implementirane sustave za jednostavan razvoj prototipa i učenje u kojemu postoji osnovna skripta koja implementira kretanje igrača. Ipak, za potrebe ovog rada odlučio sam istražiti i kreirati svoju skriptu za kretnju. U ovom radu potrebno je da se igrač može kretati u svim smjerovima, skakati, čučnuti, penjati se i spuštati niz teren. Za implementaciju kretnje kreirana je skripta „FirstPersonController“ koja je dodijeljena 3D objektu igrača. Skripta objedinjuje funkcionalnosti za kretanje, skakanje, čučanj ali i neke druge koje će biti opisane u sljedećim poglavljima.

3.5.1.1. FirstPersonController

FirstPersonController skripta je primarno zadužena za upravljanje kretanjem igrača ali vodi bridu i o stvarima kao što su parametri i metode zadužene za statistiku igrača. Skripta je rađena prema serijalu tutorijala [5] te naknadno izmijenjena i dopunjena dodatnim funkcionalnostima. Smisao povezivanja svega u jednu skriptu je taj da sve što ima direktne veze sa igračem bude dotupno na jednom mjestu iako bi se neki njeni dijelovi mogli odvojiti u zasebne skripte. Na ovaj način igrač ima samo jednu skriptu na sebi te nudi sve opcije. Skripta sadrži metode za kretanje igrača, postavke tipki za kretanje, zadane vrijednosti varijabli kretanja, metodu za rukovanje interakcijama te dijelove vezane uz produkciju zvuka u igri i metode zadužene za praćenje statistike igrača.

```
void Update()
    {
        if (CanMove)
        {
            HandleMovementInput();
            HandleMouseLook();

            if (canJump)
                HandleJump();
```

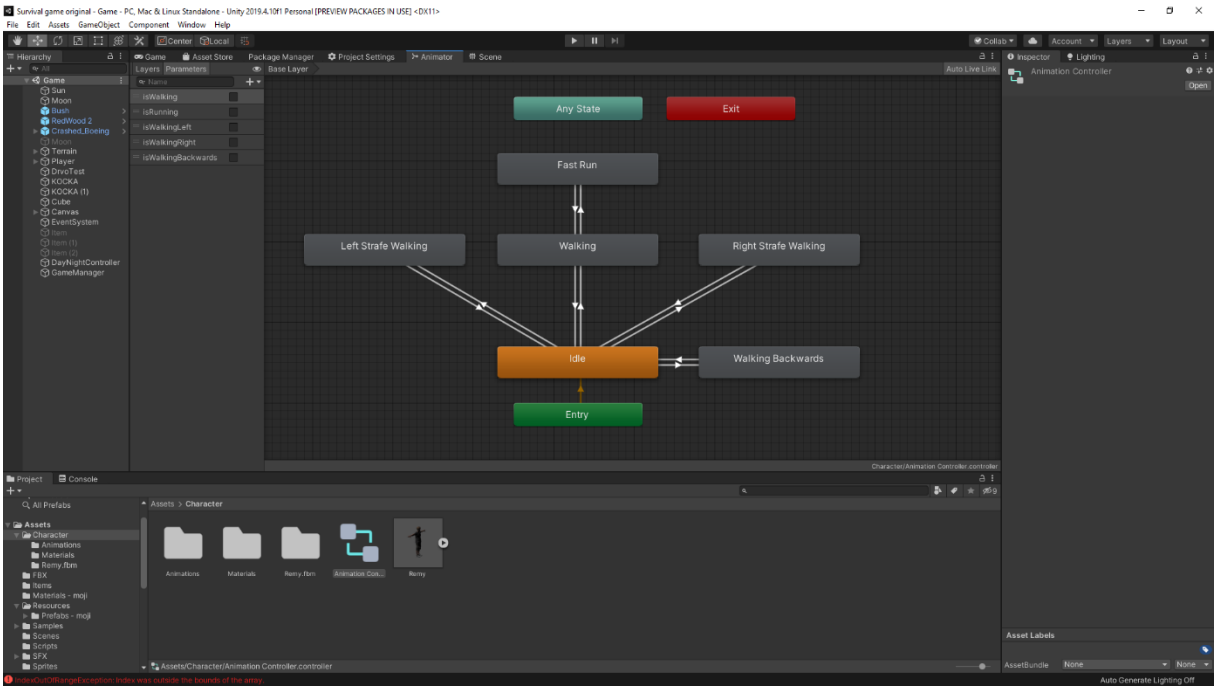
```
            if (canCrouch)
                HandleCrouch();

            if (useFootsteps)
                HandleFootsteps();

            if (canInteract)
            {
                HandleInteractionCheck();
                HandleInteractionInput();
            }

            if (useStamina)
                HandleStamina();

            ApplyFinalMovement();
            HandleHealth();
            HandleThirst();
        }
    }
```

Priložena je Update metoda koja se izvršava na svaku sličicu. Ona poziva gore navedene metode ovisno o tome ispunjavaju li uvjete za njihovo izvršavanje. Sustav kretanja je rađen na način da se omogući jednostavno uključivanje ili isključivanje dijelova koda pomoću varijabli kako bi bilo lakše testirati pojedine dijelove ali i kako bi se pružila mogućnost uklanjanja određenih dijelova iz igre bez brisanja ili komentiranja dijelova koda. HandleMovementInput, HandleMouseLook, HandleJump, HandleCrouch i ApplyFinalMovement su metode koje upravljaju kretanjem igrača te se iz njihovog naziva može zaključiti na koji dio kretnje se odnose. ApplyFinalMovement metoda služi za definiranje kretanja igrača ako se nalazi na velikoj strimini. U tom slučaju igrač će početi klizati, odnosno padati niz strminu. HandleStamina, HandleHealth i HandleThirst su metode koje se odnose na statistike igrača te pokreću korutine za njihov rad. HandleStamina smanjuje razinu energije kada igrač trči te pokreće korutinu za regeneraciju kada igrač miruje. HandleHealth metoda pokreće korutinu za smanjivanje zdravlja ili životnih bodova te ukoliko dođe do nule ubija igrača i prikazuje mu Game Over ekran. HandleThirst radi na isti princip kao i HandleHealth. Pokreće korutinu za povećavanje žeđi odnosno smanjivanje vrijednosti. Ukoliko vrijednost žedi dođe na nulu, ubija igrača i prikazuje mu Game Over ekran. HandleInteractionCheck provjerava gleda li igrač u objekt s kojim je moguća interakcija te definira što će se dogoditi kada igrač pogleda u objekt

i od objekta. Rezultat toga je moguće vidjeti kada igrač pogleda u stablo i na ekranu mu iskoči tekst „Press Mouse 1 to interact“ te se ukloni kada igrač više ne gleda u stablo. HandleInteractionInput definira što će se dogoditi ako igrač provede interakciju s objekom. U slučaju stabla, svaki put kad igrač „udari“ u stablo skida mu životne bodove. Stablo definira svoje ponašanje u tom slučaju u skripti „Drvo“ te u slučaju da su mu životni bodovi nula, stablo se sruši i izbaci materijale koje igrač može pokupiti. HandleFootsteps je metoda koja brine o tome da se puštaju odgovarajući audio zapisi u odnosu na materijal po kojem igrač hoda. Ukoliko igrač hoda po travi puštat će se određeni audio zapis hodanja po travi itd.

3.5.2. Animacije i model igrača

Radi jednostavnosti model igrača i animacije kretanja su preuzete sa stranice Mixamo.com [7]. Preuzeti model je Remy te animacije za hodanje naprijed, natrag, lijevo, desno, animacija trčanja i idle animacija. Unity ima implementirani sustav za izradu i upravljanje animacija. Za korištenje sustava potrebno je kreirati Animator Controller komponentu. Klikom na Animation Controller komponentu otvara se Animator u kojemu je moguće dodati animacije te upravljati njihovim interakcijama.



Slika 6: Animator

Prilikom importanja modela i animacija, dobra je praksa napraviti određenu strukturu unutar hijerarhije kako bi se animacije i materijali mogli razdvojiti i kasnije lakše pronaći. U tu svrhu kreirana je nova mapa „Character“ koja sadrži 3D model igrača i sve animacije koje se

odnose na taj model a bit će spremljene unutar posebne mape „Animations“. Za sve materijale koji će se koristiti, odnositi na 3D model i njegov prikaz u igri kreirana je mapa „Materials“. Preuzete animacije jednostavno prevučemo u Animator gdje je svaka animacija prikazana kao zaseban element odnosno stanje. Kreiranjem tranzicija između tih stanja određujemo iz kojeg stanja u koje su prijelazi dozvoljeni. U postavkama tranzicije moguće je podesiti određene postavke kao što su vremenska duljina prijelaza, minimalna duljina animacije koja mora biti prikazana prije promjene stanja, postavljanje uvjeta i druge. U svrhu animiranja kretnje igrača odnosno njegovog 3D modela kreirano je pet parametara koji su onda postavljeni kao uvjeti na tranzicije između animacija. Tako iz „Idle“ animacije možemo preći u animaciju „Walking“ te kada igrač više ne hoda prelazi natrag iz animacije „Walking“ u animaciju „Idle“. U skripti „PlayerAnimationScript“ implementirana je logika za prijelaz iz jednog stanja u drugo na način da se prati unos igrača, odnosno kontrole kretanja. „Idle“ animacija je postavljena kao zadana. Kada igrač pritisne tipku „W“ na tipkovnici, u igri se kreće prema naprijed i napravi se tranzicija iz „Idle“ animacije u animaciju „Walking“. Kada je tipka puštena aktivira se tranzicija iz „Walking“ animacije natrag u „Idle“. Prema tom principu ostvareni su i svi ostali prijelazi između animacija. Zbog problema sa promjenom centra, rotacije i pozicije 3D modela prilikom izvršavanja animacija, napravljena je metoda ResetPlayerModelPosition() koja se poziva nakon svake tranzicije između animacija i vraća postavke centriranja, rotacije i pozicije 3D modela na početne.

```
void Start()
    {
        animator = GetComponent<Animator>();
        initialPosition = gameObject.transform.localPosition;
        initialRotation = gameObject.transform.localRotation;
        characterController = GetComponentInParent<CharacterController>();
        initialCenter = characterController.center;
    }


    private void ResetPlayerModelPosition()
    {
        gameObject.transform.localPosition = initialPosition;
        gameObject.transform.localRotation = initialRotation;
        realPlayerControllerCenter =
gameObject.transform.parent.gameObject.GetComponent<CharacterController>();
        realPlayerControllerCenter.center = initialCenter;
    }
    public void WalkingAnimation(bool isWalking)
```

```
{
    animator.SetBool("isWalking", isWalking);
    ResetPlayerModelPosition();
}

public void WalkingLeft(bool isWalkingLeft)
{
    animator.SetBool("isWalkingLeft", isWalkingLeft);
    ResetPlayerModelPosition();
}

public void WalkingRight(bool isWalkingRight)
{
    animator.SetBool("isWalkingRight", isWalkingRight);
    ResetPlayerModelPosition();
}

public void WalkingBackwards(bool isWalkingBackwards)
{
    animator.SetBool("isWalkingBackwards", isWalkingBackwards);
    ResetPlayerModelPosition();
}

public void RunningAnimation(bool shouldSprint = true)
{
    animator.SetBool("isRunning", shouldSprint);
    ResetPlayerModelPosition();
}


private void Update()
{
    // setup movement controls
    bool forward = Input.GetKey(KeyCode.W);
    bool sprint = Input.GetKey(KeyCode.LeftShift);
    bool left = Input.GetKey(KeyCode.A);
    bool right = Input.GetKey(KeyCode.D);
    bool backwards = Input.GetKey(KeyCode.S);
    bool crouch = Input.GetKey(KeyCode.C);
```

```
        WalkingAnimation(forward ? true : false);
        WalkingLeft(left ? true : false);
        WalkingRight(right ? true : false);
        WalkingBackwards(backwards ? true : false);

        //works but the if statement is more readable
        //RunningAnimation(forward && sprint ? true : !forward || !sprint ?
false : false);
        if (forward && sprint)
        {
            RunningAnimation();
        }

        if (!forward || !sprint)
        {
            RunningAnimation(false);
        }

        if (crouch)
        {
            if (isCrouching < 1)
            {
                isCrouching++;
            }
            else
            {
                isCrouching--;
                ResetPlayerModelPosition();
            }
        }
    }
```

3.5.3. Statistika igrača

Kako bi preživio u igri, igrač mora pratiti tri parametra: glad, žeđ i izdržljivost. Sva tri parametra su u početku postavljeni na početnu vrijednost te se glad i žeđ vremenom konstantno smanjuju i tako forsiraju igrača da traži hranu i vodu kako bi preživio. Igrač hranu može naći rušenjem stabala (pri čemu postoji mogućnost da će osim drveta dobiti i 1 do 3 jabuke) ili skupljanjem bobica sa grmova. Žeđ može kontrolirati time što će pronaći vodu i spremiti je u bočicu te konzumirati po potrebi. Ukoliko vrijednost barem jednog od ova dva parametra padne na nulu, igra se završava i igraču se prikaže „Game Over“ ekran na kojem ima opciju početi ponovno ili se vratiti u glavni izbornik. Treći parametar, izdržljivost (eng. stamina) označava „umor“ ili razinu energije. Prilikom normalnog hodanja ili hodanja u čučnju izdržljivost se ne koristi dok se prilikom trčanja ili skakanja ta vrijednost smanjuje. Kada vrijednost dosegne nulu, igrač više ne može trčati ni skakati, odnosno vrijednosti za trčanje budu vraćene na vrijednosti za hodanje što čini igrača sporijim. Nakon tri sekunde „mirovanja“ ili normalnog hodanja razina izdržljivosti se regenerira te igrač može ponovno trčati.

Slika 7: Prikaz statistike igrača, glad, žeđ i izrdžljivost

Sva tri indikatora imaju zasebne skripte za upravljanje. Time se jednostavno može podesiti i nadograditi njihova funkcionalnost i način prikaza. U sljedećem kodu je prikazana skripta za upravljanje indikatorom gladi. Skripta sadrži referencu na sliku, odnosno grafiku i definira maksimalnu vrijednost gladi koja je onda postavljena prilikom pokretanja u metodi Start(). Za dobivanje informacija o promjenama koriste se akcije OnDamage i OnHeal iz FirstPersonController skripte. OnDamage akcija se aktivira kada igrač gubi na statusu gladi te se poziva metoda UpdateHealth koja postavlja indikator na vrijednost proslijeđenu kroz parametar. OnHeal radi na isti način te se poziva kada igrač nešto pojede i time podigne

vrijednost gladi. Ponovno se poziva UpdateHealth i indikator se postavi na novu vrijednost uvećanu za vrijednost dobivenu kroz hranu.

```
public class Healthbar : MonoBehaviour
{
    public Image healthbar;
    private float maxHealth = 100;

    private void OnEnable()
    {
        FirstPersonController.OnDamage += UpdateHealth;
        FirstPersonController.OnHeal += UpdateHealth;
    }

    private void OnDisable()
    {
        FirstPersonController.OnDamage -= UpdateHealth;
        FirstPersonController.OnHeal -= UpdateHealth;
    }

    private void UpdateHealth(float currentHealth)
    {
        healthbar.fillAmount = currentHealth / maxHealth;
    }

    private void Start()
    {
        UpdateHealth(maxHealth);
    }
}
```

3.6. Sustav inventara

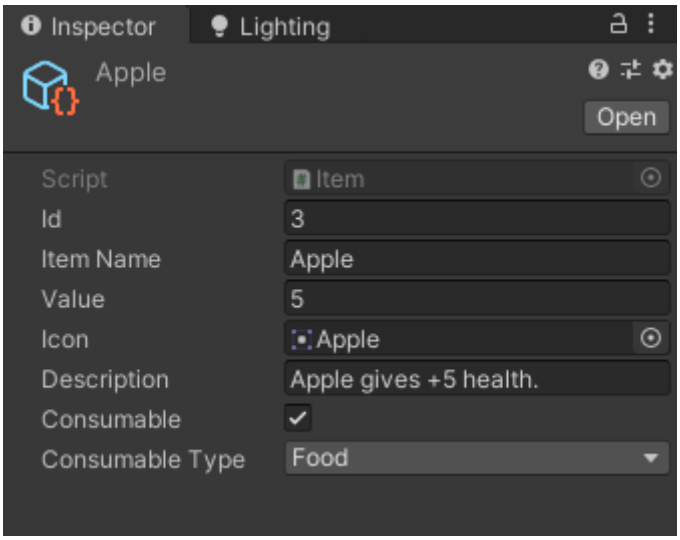
U igri preživljavanja inventar i upravljanje resursima su jako bitni. Implementacija inventara može uveliko utjecati na iskustvo igranja jer je inventar najčešće korištena značajka igre. Omogućuje igračima sakupljanje stvari i njihovo korištenje a u nekim situacijama i forsira igrača da prioritizira neke resurse ili koristi druge sustave igre s kojima je najčešće i povezan npr. crafting, chestovi, ruskak i slično.

3.6.1. Inventar

Inventar je ostvaren kroz nekoliko skripti i to ga čini najkompleksnijim elementom ovog rada. Za početak potrebno je odrediti što će inventar sve trebati pohranjivati. Odlučio sam se za jednostavan dizajn i sustav u kojem će inventar imati maksimalno 60 mjesta odnosno utora (eng. *slot*) za predmete (eng. *item*) gdje će jedan utor sadržavati jedan predmet. Za definiranje itema korišten je ScriptableObject. ScriptableObject je spremnik podataka koji omogućava spremanje velikih količina podataka neovisno o klasi i instanci objekta [6]. U ovom slučaju koristimo ga za kreiranje item prefaba, što znači da će svaki item imati svoje podatke i biti kreiran kao zaseban objekt samo jednom i spremljen kao takav u memoriju. Po potrebi onda možemo instancirati taj objekt iz memorije bez da ponovno konstuiramo novi objekt u runtime-u kada ga treba instancirati. U skripti „Item“ definirano je što svaki item treba imati.

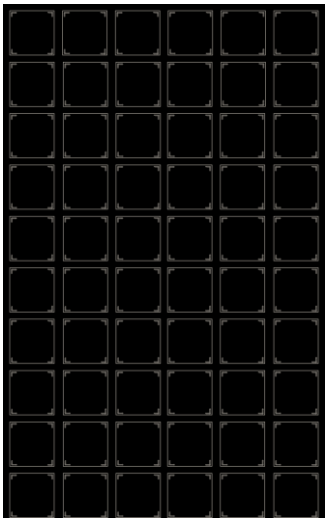
```
[CreateAssetMenu(fileName ="New Item", menuName ="Items")]
public class Item : ScriptableObject
{
    public int id;
    public string itemName;
    public int value;
    public Sprite icon;
    public string description;
    public bool consumable;
    public enum type
    {
        None,
        Food,
        Drink
    }
    public type ConsumableType;
}
```

CreateAssetMenu atribut omogućava da se item kreira kao ScriptableObject nakon čega je potrebno popuniti vrijednosti za taj item. Na slici 9 dan je primjer za item „Apple“.



Slika 8: ScriptableObject – Apple

Kada su tako definitani itemi možemo krenuti konstuirati inventar. Jednostavan dizajn inventara je kreiran u Gimpu i sadrži 60 slotova. Takav dizajn zapravo služi kao predložak preko kojega onda dolaze pravi slotovi koji imaju isti dizajn kao i na predlošku.


Slika 9: Dizajn inventara

Kreiran je prazan 3D objekt kojem je pridružena skripta „InventoryManager“. Skripta sadrži listu itema, metode za dodavanje i uklanjanje iz liste, metodu za prikaz itema u inventoriju te metodu za otvaranje i zatvaranje inventorija kada igrač pritisne tipku „Tab“. Prilikom pokretanja igre inventar je zatvoren a kad ga igrač otvori pokaže mu se kursor i zaključa se kretanje mišem što omogućava interakciju sa inventarom. Kada se inventar zatvori kursor se isključi i igraču se vrati kontrola nad kamerom, odnosno igrač ponovno dobije kontrolu kretanja mišem. 3D objektu inventory managera dodan je “Sroll view“ kojem su uklonjeni scroll elementi i dodan dizajn inventara kao slika u pozadini. Ispod njega je novi

objekt „Content“ koji će kao child elemente sadržavati slotove sa itemima koji će biti u inventoriju. Slot je napravljen kao prefab koji će se instancirati za svaki item, prikazivati informacije vezane za item te nuditi interakciju s njim. Prilikom prikaza itema u inventoriju uništavaju se svi slotovi ako postoje te se ponovno čita lista itema i instanciraju se slotovi koji su popunjeni sa itemima iz liste i njihovim podacima.

```
public void ShowItems()
    {
        foreach (Transform child in Content.transform)
        {
            GameObject.Destroy(child.gameObject);
        }

        for (int i = 0; i < Items.Count; i++)
        {
            if (Items[i] != null)
            {
                GameObject o = Instantiate(SlotItem, Content.transform);
                o.GetComponent<Slot>().AddItem(Items[i]);
                o.GetComponentInChildren<Text>().text = Items[i].itemName;

o.gameObject.transform.Find("ItemIcon").GetComponent<Image>().sprite =
Items[i].icon;
            }
            else continue;
        }
    }
```

Za ostvarivanje svega toga kreirana je nova skripta „Slot“ koja je pridružena prefabu Slota tako da svaki put kada je slot instanciran automatski na sebi ima i skriptu što olakšava dohvaćanje i mijenjanje njegovih svojstava za prikaz itema. Skripta sadrži metodu za dodavanje itema koji treba prezentirati, uklanjanje itema iz inventara kada igrač klikne na gumb X, metodu za konzumiranje itema, event metode i metodu za prebacivanje itema u hotbar koji će biti objašnjen u poglavlju u nastavku. Event metode OnPointerClick(), OnPointerEnter() i OnPointerExit() definiraju ponašanje kada igrač klikne na slot u inventoriju te kada prelazi mišem preko slota. Kada igrač desnim klikom klikne na item u inventoriju, ukoliko je item definiran kao consumable, pozvat će se metoda Consume() koja će „konzumirati“ item i dodijeliti određenu vrijednost itema na statistiku gladi ili žeđi ovisno o tipu. Zbog toga skripta sadržava referencu na FirstPersonController skriptu iz koje poziva metode Eat() ili Drink() te

samim time ažurira i korisničko sučelje (eng. *User Interface - UI*) za te elemente. Ako igrač klikne na item srednjim klikom poziva se metoda SwitchToHotbar() koja prebacuje item u hotbar i miče ga iz inventara. Te dvije stvari su implementirane u metodu OnPointerClick(). OnPointerEnter() metoda aktivira opis alata (eng. *Tooltip*), odnosno element za prikazivanje opisa itema kada je kursor miša na itemu odnosno slotu. Kada se kursor makne sa slota tooltip za taj slot se deaktivira metodom OnPointerExit().

Slika 10: Inventar i itemi

Na kraju da bi svaki item mogao bi ti pokupljen i spremljen u inventar koristi se skripta „Pickup“ koja je dodijeljena itemu pri instanciranju. Kada igrač klikne lijevim klikom na item on ga „pokupi“ i spremi u inventar.

```
public class ItemPickup : MonoBehaviour
{
    public Item item;

    void PickUp()
    {
```

```
            InventoryManager.Instance.Add(item);
            Destroy(gameObject);
        }

        private void OnMouseDown()
        {
            PickUp();
        }

}
```

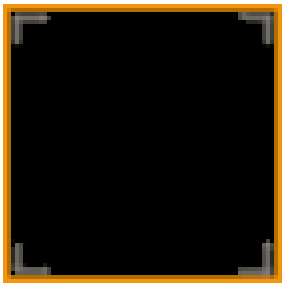
3.6.2. Hotbar

Hotbar je „manji inventar“ koji služi igraču za prikaz odmah dostupnih itema. Ima poseban prikaz i koristi zasebne slotove. Postupak izrade je sličan onome za inventar. Kreiran je novi prazni element kojem je dodijeljena skripta „HotBarManager“ koja ima slično ponašanje kao i „InventoryManager“. Kreiran je i novi prefab za slot koji je u principu isti onaj kao i slot za inventar ali su uklonjeni X gumb i tooltip. Kreirana je i nova skripta „Hotbar“ koja definira njegovo ponašanje. Hotbar skripta ima metode za dodavanje itema koji treba prikazati, njegovo brisanje te vraćanje u inventar ukoliko igrač pritisne srednji klik na njega. HotBarManager skripta sadrži listu itema koje treba prikazati, polje hotbar slotova koje treba instancirati te metodu za njihovo instanciranje, metodu za prikazivanje itema te metode za intanciranje itema u ruku igrača kada je slot aktivan i njegovo micanje iz ruke kada slot nije aktivan.

Slika 11: Dizajn hotbara

Grafika za hotbar je kreirana u Gimpu i služi kao predložak ali u ovom slučaju kreirani su svi slotovi na početku igre i postoje cijelo vrijeme. Razlika između slotova u inventaru i slotova u hotbaru je ta što se u inventaru stalno brišu i kreiraju novi kako bi prikazali iteme, tako da postoji samo onoliko slotova koliko i itema u inventaru. Kod hotbara, s obzirom da ima samo 5 slotova, ti slotovi se kreiraju na početku i ne brišu se. Umjesto liste za njih je korišteno polje a za prezentaciju itema mijenjaju se zadane postavke slota sa itemom koji treba prikazati. Prilikom brisanja itema iz tog slota, slot ostaje a njegove se vrijednost vraćaju na prvotno stanje čime on postaje slobodan. Igrač može pristupiti svakome od njih pritiskom na tipku 1-5 na

tipkovnici. U Gimpu je kreirana posebna sličica za aktivan slot. Aktivan slot je onaj koji igrač odabere pritiskom tipke a na početku predodređeni aktivni slot je slot 1.



Slika 12: Dizajn aktivnog slota

Kada igrač odabere slot, ukoliko se u njemu nalazi item on će se instancirati ispred igrača kao da ga drži u ruci. Ako igrač odabere slobodni slot, odnosno slot u kojem ne postoji item, postojeći će se obrisati iz ruke i neće se instancirati novi. Ako igrač klikne srednjim mišem na item u hotbaru dok je inventar otvoren, item će se prebaciti natrag u inventar.

Slika 13: Igrač sa itemom u ruci

U Start() metodi instanciraju se slotovi pozivom LoadHotbarSlots() i postavljaju se početne vrijednosti. U Update() metodi svi slotovi su mapirani na tipke 1-5 kako je već spomenuto te se prema tipki koja je pritisnuta odrađuje logika za promjenu aktivnog slota i prikaz itema u ruci.

```
void Start()
    {
        Content = GameObject.Find("HotbarSlots");
        LoadHotbarSlots();
        hotbarSlots = GameObject.FindGameObjectsWithTag("HotbarItem");
        hotbarSlots[0].GetComponent<Image>().sprite = selectedItem;
    }

public void LoadHotbarSlots()
    {
        for (int i = 0; i < 5; i++)
        {
            GameObject o = Instantiate(HotbarItem, Content.transform);
            o.GetComponentsInChildren<Image>()[1].enabled = false;
            o.GetComponentInChildren<Text>().enabled = false;
        }
    }

void Update()
    {
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            try
            {
                GameObject[] hotbarslots =
GameObject.FindGameObjectsWithTag("HotbarItem");
                foreach (GameObject slot in hotbarslots)
                {
                    if(slot.GetComponent<Image>().sprite = selectedItem)
                    {
                        slot.GetComponent<Image>().sprite = defaultSprite;
                    }
                }
                hotbarslots[0].GetComponent<Image>().sprite = selectedItem;

                // remove item from hand if it exists
                RemoveItemFromHand();
                // place item in hand

SpawnItemInHand(hotbarSlots[0].GetComponent<Hotbar>().item);
            }
```

```
            catch (System.Exception)
            {
                Debug.Log("No hotbar slot");
            }
        }

private void SpawnItemInHand(Item item)
    {
        // right hand slot position
        Transform CameraPosition = Camera.transform;
        GameObject activeItem = Instantiate(Resources.Load(item.itemName),
CameraPosition.position, Quaternion.AngleAxis(0f, Vector3.right),
CameraPosition) as GameObject;
    }

public void RemoveItemFromHand()
    {
        try
        {
            GameObject itemInHand =
GameObject.Find("RightHandSlot").transform.GetChild(0).gameObject;
            Destroy(itemInHand);
        }
        catch (System.Exception)
        {
            Debug.Log("No item in hand");
        }
    }
```

3.7. Upravljanje scenama

Pri pokretanju igre igraču je potrebno prvo prikazati glavni izbornik. Za to koristimo novu scenu u kojoj ćemo postaviti dva gumba, jedan za pokretanje igre a drugi za izlazak iz igre. Isto to želimo napraviti i u slučaju završetka igre kada igrač „umre“, međutim u tom slučaju nema potrebe za kreiranjem nove scene. Umjesto toga kreiran je novi „Canvas“ koji će poslužiti kao „Game Over“ ekran. U njemu ćemo isto tako koristiti dva gumba od kojih će jedan nuditi igraču da ponovno pokrene igru ispočetka, a drugi će nuditi povratak na glavni izbornik. Za upravljanje scenama kreirane su dvije skripte: „GameManager“ i „MainMenu“. Za scenu igre kreiran je novi prazni 3D objekt i dodana mu je skripta „GameManager“ koja će u slučaju da igrač izgubi isključiti kontrole i prikazati „Game Over“ ekran. U slučaju da igrač želi ponovno

pokrenuti igru scena će biti ponovno učitana. Ako igrač odabere vratiti se na glavni izbornik učitat će se scena „Main Menu“. Istom tom logikom, „Main Menu“ scena u slučaju pokretanja igre učitava scenu igre, a opcija izlaska zatvara igru.

```
public class MainMenu : MonoBehaviour
{
    public void LoadGame()
    {
        SceneManager.LoadScene("Game");
    }

    public void QuitGame()
    {
        Application.Quit();
    }
      }

public class GameManager : MonoBehaviour
{
    GameObject gameOver;
    public void KillPlayer()
    {
        //show game over
        gameOver.SetActive(true);

        //disable FirstPersonController sript

GameObject.Find("Player").GetComponent<FirstPersonController>().enabled =
false;

GameObject.Find("InventoryManager").GetComponent<InventoryManager>().enable
d = false;
        //enable mouse cursor
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = true;
    }
    private void Awake()
    {
        gameOver = GameObject.Find("GameOverScreen");
        gameOver.SetActive(false);
    }
      }
```

Slika 14: Main Menu

Slika 15: Game Over ekran

4. Zaključak

Rad je pisan kronološki tako da vjerno opisuje cijeli postupak i redosljed razvoja igre. U početku je bilo teško usmjeriti se prema jednom cilju i podijeliti razvoj na više dijelova. Najteži dio je bio učenje i upoznavanje sa game engine-om i povezivanje funkcionalnosti kroz programiranje. Uz puno sati gledanja videa, debugiranja i čitanja dokumentacije polako sam usvajao znanje i postao sve komforniji i brži u korištenju alata i razvoju. Prošavši sve dijelove od programiranja, animacija i dizajna vlastitih materijala dobio sam dojam o trudu i vremenu koji je potrebno uložiti u igru da se dobiju zadovoljavajući rezultati. Samim time počeo sam više cijeniti rad profesionalnih developera i studija te razmišljati o načinu kako su određeni dijelovi implementirani i kako bi ih mogao i sam implementirati u svoju igru. Osim Unity-a, kroz rad sam se upoznao i sa osnovama rada u Blenderu i Gimpu koji su bili velika pomoć u izradi vlastitih materijala.

Ovaj projekt smatram osobnim postignućem i uvjerenjem da je moguće ostvariti ideje ukoliko se uloži dovoljno truda i vremena. Svaki početak je težak ali uz upornost i konstantno unaprjeđivanje moguće je postići zadovoljavajuće rezultate. Planiram nastaviti rad na ovom projektu kako bi stekao još iskustva i pokušao razviti i implementirati neke nove sustave te koristiti ovaj projekt kao bazu za potencijalne projekte u budućnosti.

Popis literature

[1] „Minecraft“. Minecraft Wiki. Dostupno: https://minecraft.fandom.com/wiki/Minecraft [pristupano 15.07.2022.]

[2] Unity. Dostupno: https://unity.com/ [pristupano 29.06.2022.]

[3] About. Dostupno: https://www.blender.org/about/ [pristupano 29.06.2022.]

[4] GIMP – GNU Image Manipulation Program. Dostupno: https://www.gimp.org/ [pristupano 12.08.2022.].

[5] Comp-3 Interactive, (04.06.2021.) „Detailed First Person Controller“, *Youtube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=2FTDa14nryI&list=PLfhbBaEcybmgidDH3RX_qzFM0mIxWJa21 [pristupano 12.12.2021.]

[6] Unity Manual. Dostupno: https://docs.unity3d.com/2019.4/Documentation/Manual/class-ScriptableObject.html

[7] „Remy“ (bez dat.) u Mixamo.com. Dostupno: https://www.mixamo.com/#/?page=1&type=Character [pristupano: 20.07.2022.]

Popis slika