Izrada igre preživljavanja u programskom alatu Unity

Štefičar, Dominik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Štefičar

Izrada igre preživljavanja u programskom alatu Unity

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Štefičar

Matični broj: 46446

Studij: Informacijski sustavi

Izrada igre preživljavanja u programskom alatu Unity

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2021

Dominik Štefičar

Izjava o izvornosti

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

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

_____

Sažetak

Tema ovog završnog rada bavi se razvojem 3D računalne igre sa temom preživljavanja, gdje se većinski i funkcionalni dio igre kreira u programskom alatu Unity pomoću C# programskog jezika. Opisani su također ostali korišteni alati i neki procesi pomoću kojih su pripremljeni modeli za igru. Igra započinje prikazom izbornika za pokretanje igre i pritiskom na gumb smješta igrača na napušteni otok s pogledom u prvom licu. Igraču se na ekranu prikazuju razina hrane i vode, te je zadatak pokušati istražiti otok, izraditi alate i logorsku vatru, spriječiti glad i žeđ, izgraditi kuću te generalno preživjeti što je duže moguće. Igrač posjeduje svoj inventar svih predmeta s kojima može baratati na razne načine te koristiti alate za interakciju sa okolinom. Cilj projekta je samostalni pokušaj kompletnog razvoja kvalitetne igre koristeći što veći raspon besplatnih pomagala.

Ključne riječi: Unity, preživljavanje, 3D modeliranje, igra u prvom licu, C#, animacije, izgradnja

Sadržaj

1. Uvod

Igre preživljavanja često se pojavljuju kao projekt malog indie tima. Danas postoje mnoge vodeće igre u tom žanru, kao što su „DayZ“, „ARK: Survival Evolved“, „The Forest“ , „Rust“ i mnoge druge. Uveliko su svojim razvojem pridonijele razvoju mehanika ostalih igra koje nisu istog žanra. Prva igra preživljavanja razvijena je i puštena u prodaju 1878. godine pod nazivom „Where's My Pouch“, ali u to vrijeme tehnologija nije bila dovoljno razvijena da bi igra privukla dovoljno pažnje te su popularnost ponajviše pridobile simulacijske arkadne igre. Prva moderna igra preživljavanja „Stranded“ izašla je 2003. godine gdje igrač preživljava na otoku i pokušava pronaći put kući. Karakter uloge u kojoj se igrač nalazi, gradnja baze i sloboda igre privukli su modernu publiku i dali poticaj industriji za razvoj sličnih igara [1] . Uz pomoć sve više razvijenih novih tehnologija, postoje mnogi alati koji danas olakšavaju da se kvalitetne igre razvijaju u što kraćem roku. Ovaj projekt bavi se razvijanjem osnovnih mehanika modernih igra preživljavanja koristeći se pritom raznim dostupnim alatima za što lakšu i kvalitetniju izradu nove igre.

Igra najveću značajnost ima u količini uloženog truda u realizam koji se predstavlja igraču. Danas koristeći se raznim videima i instrukcijama na internetu nije teško kreirati neke početne elemente za igru. Problem se nalazi u daljnjem razvoju kada se krenu razvijati i povezivati različiti sistemi unutar igre. Dodatnu prepreku predstavlja količina realizma u svakoj mehanici što traži detaljnije specifikacije načina prikazivanja i načina rada tih sistema. Pružajući dovoljno pažnje na kvalitetu i kvantitetu sadržaja igre daje korisniku bolje iskustvo prilikom igranja i veću podršku za daljnji razvoj.

Jedna od popularnijih igra preživljavanja nazivom „The Forest“ predstavlja se kao simulacijska horor igra preživljavanja iz prvog lica u otvorenom svijetu, te je prva verzija igre objavljena 2014. godine. Sadrži velik raspon razne vegetacije i mehanika sa kojom igrač može baratati, poput sječe raznih vrsta drva, izgradnje oružja i alata, izgradnje malih skloništa i velikih kuća, lova na životinje i borbe sa kanibalima [2] . Od kada sam pokrenuo prvu verziju pa sve do danas, fasciniralo me kako mali tim ljudi može izraditi ovakvu igru u Unity alatu, te me igra inspirirala da sam pokušam skupiti potrebne vještine i razviti nešto slično.

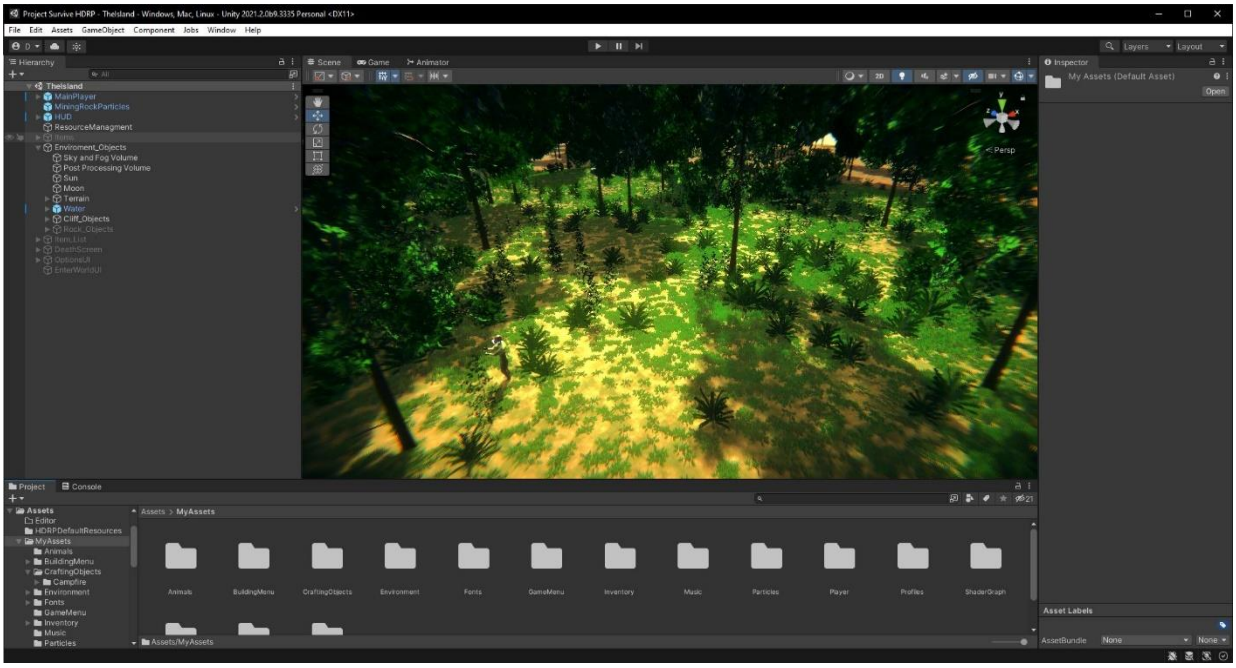
2. Metode i tehnike rada

Glavni dio razvoja igre smješten je unutar alata Unity, gdje se funkcionalnost izrađuje pomoću alata Microsoft Visual Studio i programskog jezika C# te se povezuju izrađeni 3D modeli i korak po korak razvija igra. Kako bi se izradili modeli, korišten je alat Blender za 3D modeliranje i animiranje. Za crtanje potrebnih tekstura korišten je Gimp i alat Materialize za modificiranje i izradu svojstva teksture.

2.1.1. Unity

Unity je alat za izradu 3D ili 2D igre. Kao pogonski sklop igre on je opremljen sa mnoštvo implementiranih pomagala za pomoć pri razvijanju, kao što su kolizije objekata, fizike, svijetlost i slično, te svake godine sve više i više pridonosi razvijanju svog seta unutarnjih alata kojim omogućava kvalitetan i lakši razvoj igara. Glavni jezik za programiranje je C# a najčešći korisnici su mali indie timovi i pojedinci. Postoje četiri verzije plana Unity alata od kojih je verzija bez ikakvih dodataka besplatna, a plaćaju se Unity Pro, Unity Enterprise i Unity Plus koji korisniku pridodaju neke pogodnosti tijekom razvoja projekata. Također Unity ima mogućnost razvoja igara na raznim platformama, te obuhvaća razvoj za stolna računala, konzole, mobilne platforme i web [3] .

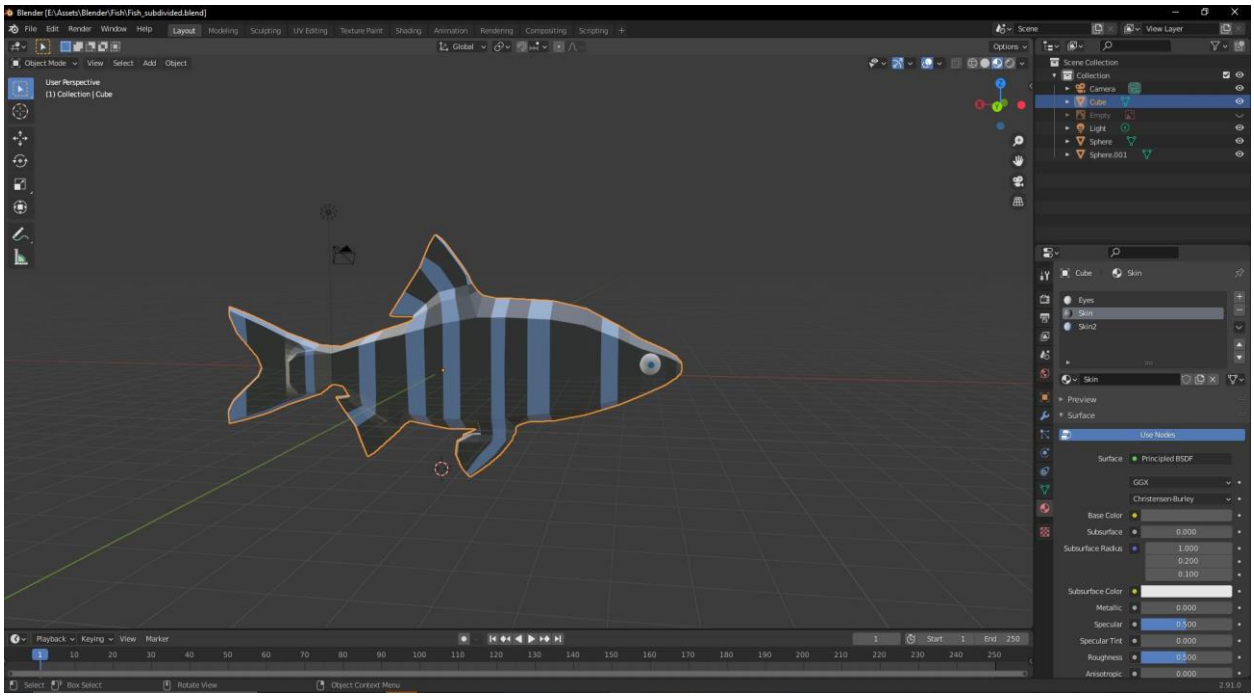
Unity Beta najnovija je eksperimentalna verzija alata Unity koja se koristi u svrhu ovog projekta i koja pruža veći opseg pomoćnih alata za izradu igre.



Slika 1. Unity sučelje

2.1.2 Blender

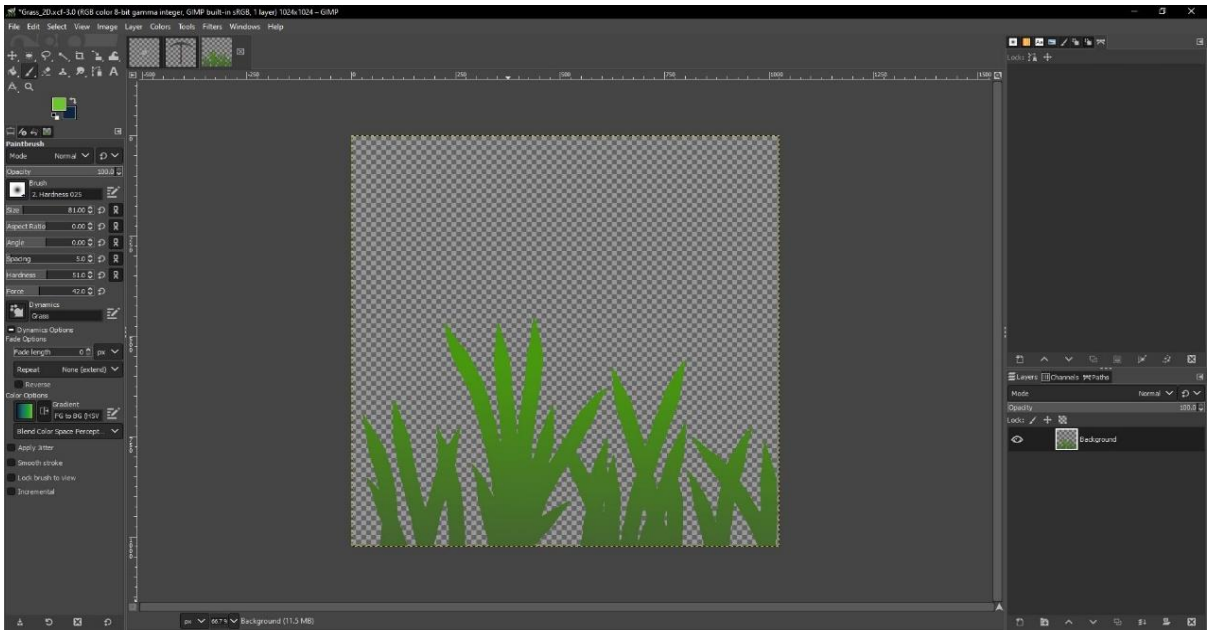
Blender je 3D alat dostupan za Linux, Windows i Mac sustave te primjenjiv za široku namjenu izrade 3D modela. Besplatan je za komercijalne i edukacije svrhe. Otvorenog je koda te organizacija potiče korisnike na male i velike izmjene u svrhu dodavanja novih funkcionalnosti, ispravljanja grešaka i poboljšanog baratanja alatom [4] . Dijelovi korišteni u realizaciji projekta su modeliranje, postavljanje kostiju, animiranje i projektiranje 2D slika na 3D modele za mapiranje tekstura.



Slika 2. Blender sučelje

2.1.3. Gimp

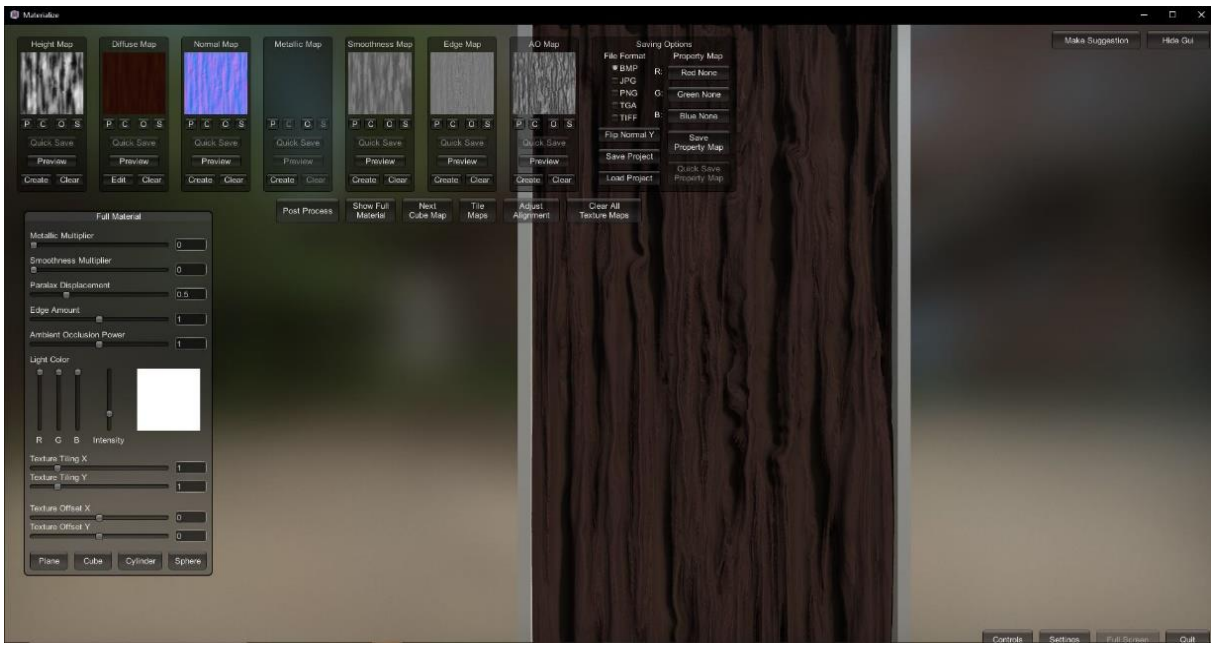
Gimp je alat za manipulaciju i izradu slika dostupan za Linux, OS, Windows i Mac sustave. Besplatan je za korištenje i otvorenog je koda. Pruža veliki raspon dodataka kreiranih od strane korisnika za olakšano uređivanje. Često se koristi u svrhu izrade grafičkih elemenata, ikona, ilustracija i maketa [5] . Pomoću ovog alata izrađene su razne ikone i teksture korištene u projektu.



Slika 3. Gimp sučelje

2.1.4. Materialize

Materialize je samostalan besplatan alat otvorenog koda za proces stvaranja materijala za modele proizvedeni iz slika. U alat se učitava tekstura te se korak po korak izrađuju mape zadane teksture [6] . Mape tekstura se koriste kako bi se na površini 3D modela kreirale ponavljajuće teksture, uzorci i specijalni vizualni efekti poput načina osvjetljavanja i odsjaja od površine modela. U igrama se najčešće koriste za prikaz slike na fizičkoj osnovi(eng. Physical based rendering - PBR) kako bi se tekstura bila foto realistična i reagirala na svijetlost [7] .



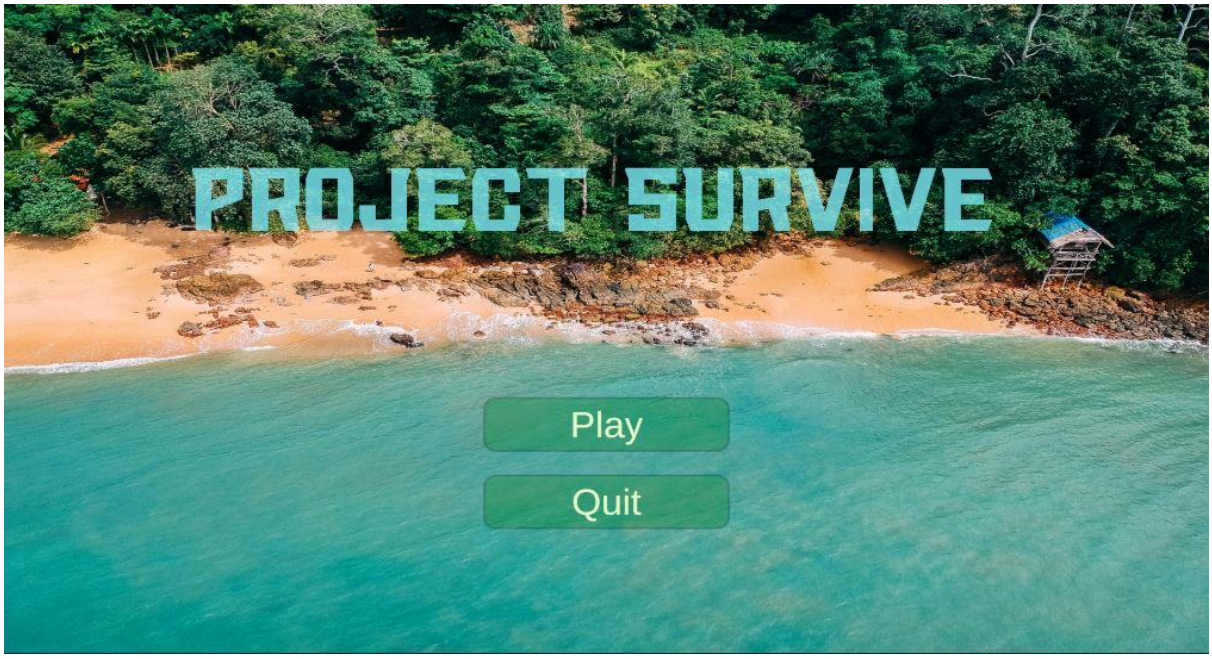
Slika 4. Materialize sučelje

3. Razrada teme

U ovom poglavlju prolazi se kroz proces razvoja igre, počevši od pripreme svih glavnih dijelova za početak razvoja. Opisat će se kako je pripremljen sam projekt, njegovi materijali za razvoj, te grafičke postavke. Nakon pripreme prelazi se na osnovne mehanike i svojstva igrača te njegove animacije nakon čega slijedi opis interakcije između igrača i okoliša. Nadalje se prikazuju svi aspekti upravljanja resursima, građa kuće i mehanike riba.

3.1. Priprema projekta

Proces razvoja igre počinje odabirom radne verzije Unity alata koja će se koristiti. Preporučeno je korištenje zadnje stabilne verzije, no u ovom slučaju koristi se Unity 2021 beta verzija koja nudi posebne mogućnosti za razvoj projekta. Također, postoje tri cjevovoda za prikaz slike (eng. Render pipeline) nazivom Standard, High-Definition i Universal putem kojih se upravlja grafikom scene. Svaki od njih nudi različite pogodnosti za postavljanje grafike projekt, te se generalno razlikuju po utjecaju na performanse igre. Projekt koristi High-Definition (HDRP) koji omogućuje korištenje najnovijih i zahtjevnijih Unity tehnologija za vizualno poboljšavanje igre. Postavljamo dvije scene u igri, od kojih će prva služiti za početni meni igre, a druga za ostatak igre. U meniju postavljamo gumb za početak koji nas prebacuje na sljedeću scenu i gumb za kraj preko kojeg se izlazi iz igre.



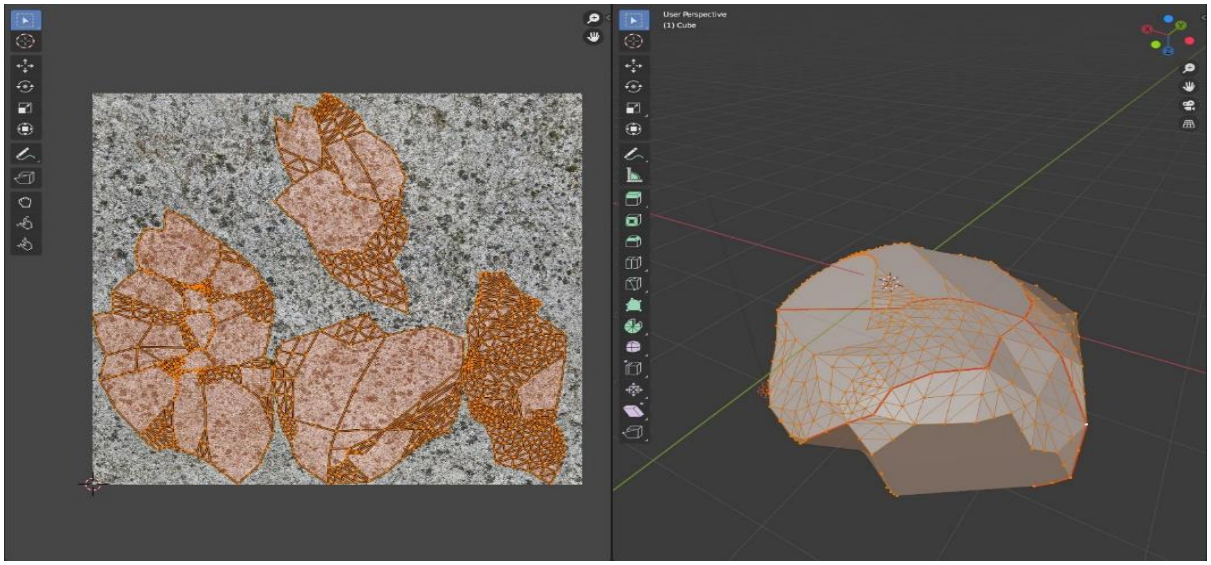
Slika 5. Meni igre

3.1.1. Priprema modela

Prije same implementacije igrača u igru, kao i svi ostali vlastito izrađeni modeli u projektu, izrađeni su u Blenderu gdje prolaze određen proces bi bili spremni za igru. Taj proces započinje izradom modela pomoću namještanja vrhova koordinatnih točaka u prostoru te njihovo spajanje u poligone. Za olakšanje posla često se koriste neke slike za reference koje olakšavaju posao postavljanja lokacija točaka.

Slika 6. Proces razvoja modela

Nakon što je 3D model gotov, kreiraju se materijali zaslužni za boju modela. Ako se nanose posebne teksture, prolazi se kroz proces UV odmotavanja (eng. UV unwrapping) kojim se projektira 2D slika na 3D model. Označuju se crvene linije kroz vrhove točka te se 3D model pokušava projektirati na 2D sliku određene veličine tako da se čitav model podijeli na četiri ili više strana i raširi po slici.

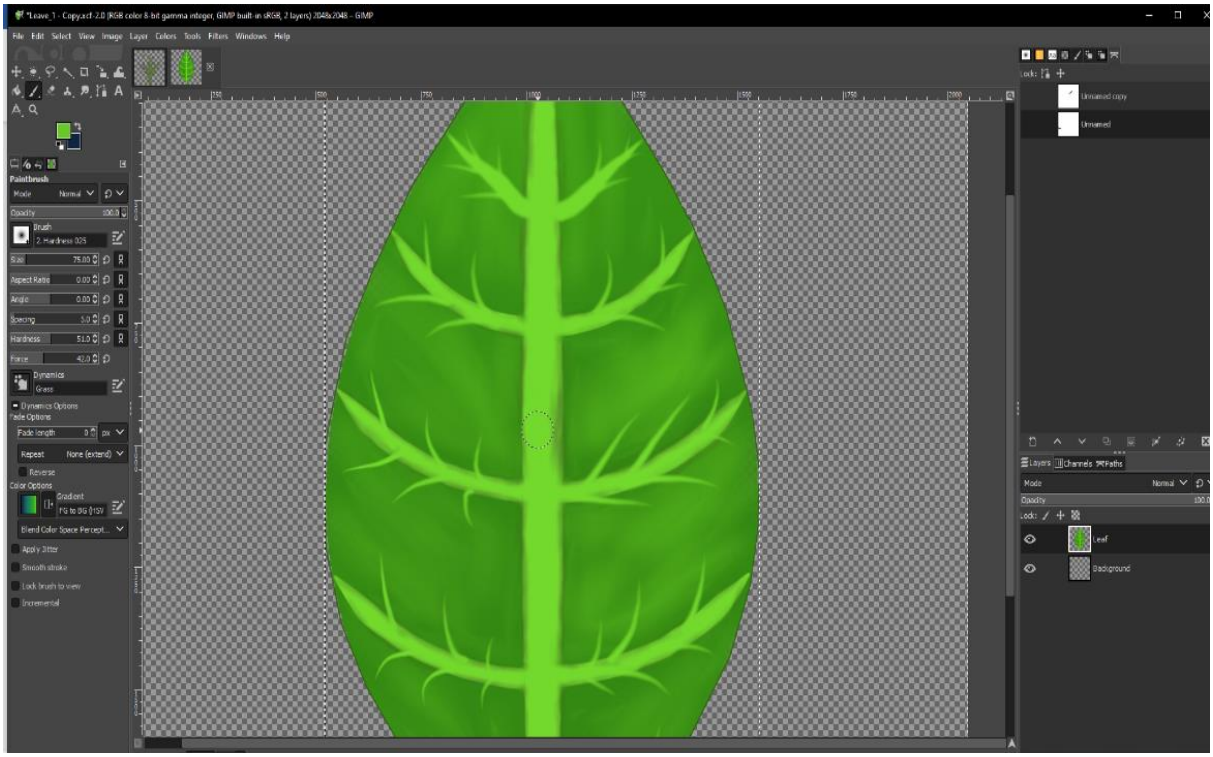


Slika 7. UV odmotavanje

U slučaju da su za model prihvatljivi materijali direktno kreirani iz blendera, moguće je preskočiti ovaj korak i zasebno odabrati poligone koji će poprimiti određeni materijal, kao što je to napravljeno sa modelom igrača.

3.1.2. Priprema tekstura

Nadalje kako bi uopće mogli dodati teksturu na model, potrebno ju je dobaviti ili kreirati. U projektu postoje teksture direktno preuzete sa web stranica koje nude besplatne bešavne (eng. Seamless) teksture, što znači da kada se tekstura ponavlja, ima kontinuirani uzorak bez vidljivog mjesta spajanja te bezbroj ponovljenih tekstura izgleda kao jedna bez obzira na njenu veličinu i smjer. Dio tekstura napravljen je u alatu Gimp gdje se pomoću raznih pomagala za crtanje, od kojih je glavni kist različitih oblika i svojstva, stvara slika.

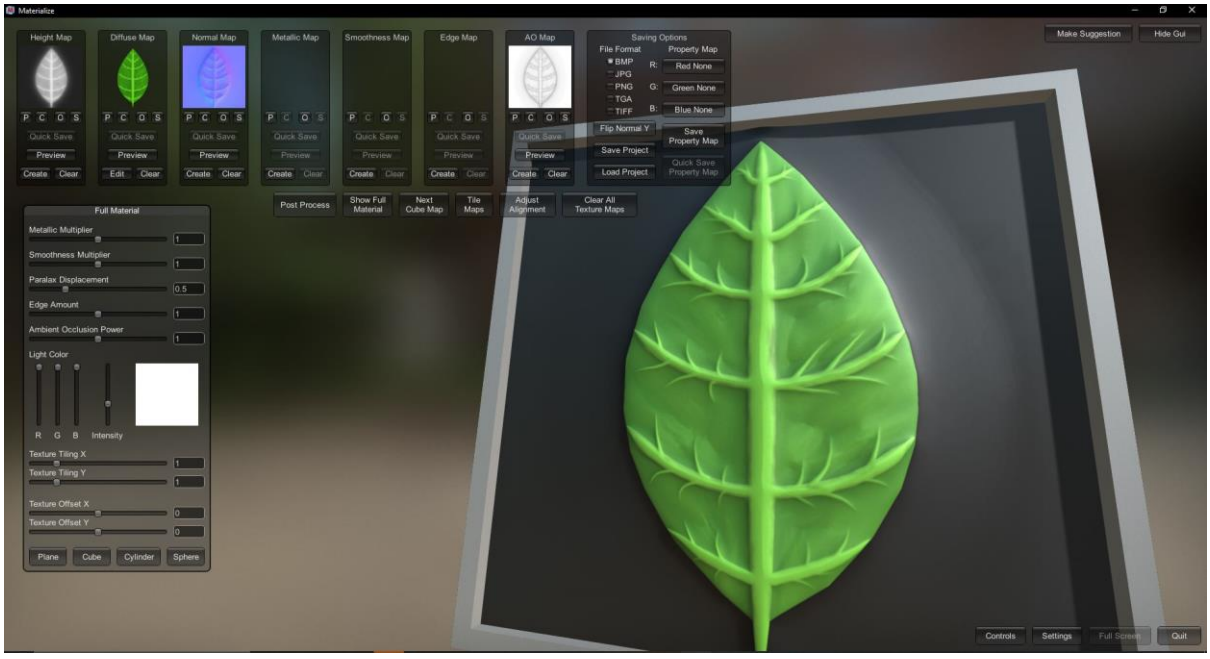


Slika 8. Crtanje lista u alatu Gimp

Ostatak tekstura unutar projekta koje većinom služe kao ikone koje reprezentiraju svoje 3D modele, poslikane su kamerom s pogledom na objekt direktno u Blenderu ili su preuzete sa web stranica koje nude besplatne ikone.

Za svaku teksturu koja će imati posebna svojstva poput vlastite refleksije svijetlosti na modelu te količinu izloženosti ambijentalnom svijetlu, potrebno je izraditi mape teksture, za što se često koristi neki od grafičkih alata kako bi se slici pridodali određeni tonovi i boje s obzirom na svojstva neke od mapa. Postoje i zasebni programi koji olakšavaju proces kreiranja mapa, te je jedan od njih besplatan alat Materialize koji je korišten za svrhe projekta.

Proces započinje dodavanjem Diffuse mape koja određuje boju materijala kod upijanja svijetlosti. Kako se projekt bavi prikazom slike na fizičkoj osnovi, bit će potrebna Albedo mapa umjesto Diffuse mape gdje je razlika u tome što Albedo mapa daje samo osnovnu boju predmetu, dok Diffuse mapa također zasjenjuje predmet. Zbog toga se Diffuse mapa ne sprema već se koristi samo kao izvorna slika za daljnju obradu. Sljedeće mape koje se kreiraju su Height i Normal mapa, te obje daju sličan rezultat, gdje Normal mapa daje teksturi dubinu bez mijenjanja ili dodavanje ekstra geometrije samog modela. Osnovna boja je ljubičasta te sve ostale boje i njihove nijanse predstavljaju geometriju modela i način na koji svijetlost djeluje na površinu modela dajući mu pritom prividne obline. Metallic mapa određuje jeli materijal metal, i ako jeste, kojim postotkom. Smoothness mapa definira kako se svijetlost raspršuje preko materijala, gdje najveća vrijednost znači da je predmet potpuno gladak i točka svijetlosti se dalje širi po materijalu, kao što je slučaj kod mramornih pločica. Kod najmanje vrijednosti je situacija suprotna i nema svjetlosne refleksije kao kod npr. kore drveta. Zadnja mapa u procesu je Ambient Occlusion mapa koja služi kao tehnika bacanja slabih sijena oko modela u svrhu realističnog izgleda objekta [8] .

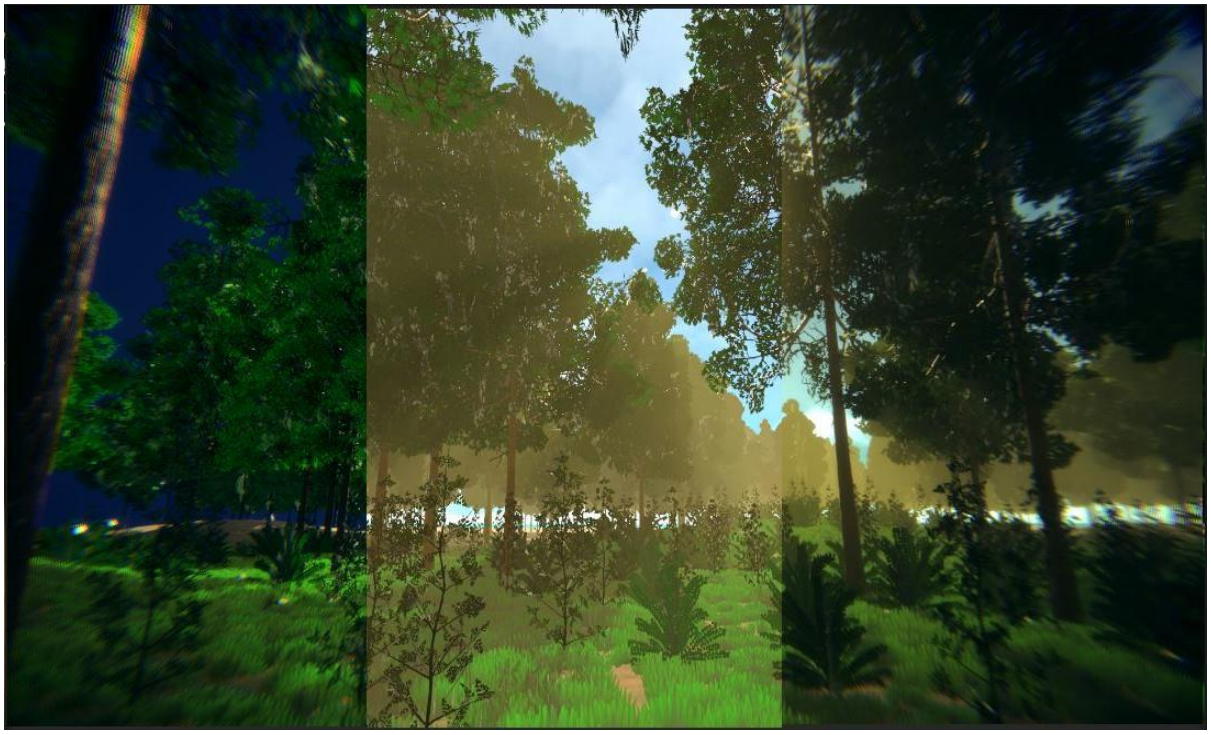


Slika 9. Materialize proces mapiranja lista

3.1.3. Priprema grafičkih postavka

Za vizualno unaprjeđivanje slike Unity HDRP nudi postavljanje postavka profila za naknadno obrađene efekte (eng. Post processing effects) te postavka profila za nebo i maglu. Počevši od profila za nebo i maglu, postavljamo specifikacije za realistično nebo i pridodajemo nebu sloj zvijezda i oblaka te postavljamo razinu, jačinu i boju volumetrijske magle.

Nadalje postavljamo profil efekata, pridodajemo sceni efekt sjaja (eng. Bloom) koji povećava blistavost izvora svijetlosti i pridonosi realizmu izgleda svijetlosti [9] . Sljedeći efekt je kromatska aberacija (eng. Chromatic Aberration) koji se u stvarnom svijetu stvara kada leća ne uspije fokusirati sve boje u jednu točku pa pritom ovisno o svjetlini dolazi do prijeloma boja na nekim objektima. U igri pridodaje realizmu slike te stvara efekt kao da je igra snimana kamerom [10] . Postavljanjem vrijednosti namještamo boju i jačinu sjena te naposlijetku dodajemo efekt Vinjete (eng. Vignette) koji služi za potamnjivanje obruba kamere.



Slika 10. Postavljanje profila grafike

3.2 Igrač

3.2.1. Mehanike kretnje igrača

Kako bi mogli vidjeti igru izvan pogleda Unity Editora, odnosno smjestiti se u pogled same igre, potrebno je u scenu implementirati glavnu kameru. Kamera će se prilijepiti na poziciju oči prijašnje nadodanog modela igrača te se ponašati kao pogled iz prvog lica. Također kamera ima set integriranih postavka za modifikaciju kamere, te su potrebne neke izmjene. Postavka „Clipping planes“ postavlja na vrijednost 0.2 što će pomoći da kamera nakon namještanja ne prikazuje sadržaj udaljen do 0.2 vrijednosti koordinatnog sustava, te se time sprječava nametanje modela igrača ispred kamere nakon izvođenja neke animacija igrača. Post Anti-aliasing postavljen je na TAA, što pomaže kod odstranjivanja nazubljenih rubova objekata u sceni. Uključeni „Occlusion Culling“ pomoći će u održavanju stabilnih slika po sekundi isključujući sve objekte koji se trenutno ne nalaze u širini pogleda kamere.

Slika 11. Clipping planes i TAA

Koncept interakcije između kamere i modela igrača bazira se na prepoznavanju pomaka miša te s obzirom jeli pomak po X ili Y osi, odnosno jeli miš pomaknut u smjeru gore-dolje ili lijevo-desno. Ako je u pitanje pomak lijevo-desno, potrebno je rotirati model igrača u željenom smjeru za toliku vrijednost koliki je pomak miša po X osi. Ako se miš pomakne po Y osi, potrebno je središnju kost armature igrača rotirati po X osi za toliku vrijednost koliki je pomak miša po Y osi. Pri tom potrebno je obratiti pažnju da ljudsko tijelo nema mogućnost pomicanja kostiju leđa za puni krug te je potrebno ograničiti igrača da u slučaju prelaska rotacije leđne kosti u vrijednosti od 90 ili 270 stupnjeva zaključa rotaciju na toj poziciji. Ovdje se nailazi na problem nametanja, gdje animacija modela pokušava konstanto promijeniti

rotaciju i poziciju modela, dok s druge strane to isto radi skripta za pogled igrača i često rezultira beskonačnom vrtnjom ili deformacijom modela. Problem je moguće razriješiti ako se za animacije koristi Update() metoda a za transformaciju modela LateUpdate(), te je razlika da se Update() metoda poziva jednom po slici, a LateUpdate() se poziva tek nakon što se cijela Update() metoda kompletno procesira. Za soluciju je kreirana skripta te rezultat ovakvog pristupa je poboljšani realizam igrača gdje se van igračevog pogleda može dobro raspoznati u kojem smjeru igrač trenutno gleda te realistična pratnja kamere igračevih animacija i pokreta.

```
    public void CameraRotation(){
        mouseX = Input.GetAxis(mouseXInputName) * mouseSensitivity *
Time.deltaTime;
        mouseY = Input.GetAxis(mouseYInputName) * mouseSensitivity *
Time.deltaTime;
        xAxisCheck -= mouseY;
        if(xAxisCheck > 90.0f)
        {
            xAxisCheck = 90.0f;
        }
        else if(xAxisCheck < -90.0f)
        {
            xAxisCheck = -90.0f;
        }
        playerRotationPosition.Rotate(Vector3.up * mouseX);
    }
    private void LateUpdate()
    {
        playerCameraPosition.Rotate(Vector2.left * xAxisCheck);

    }
```

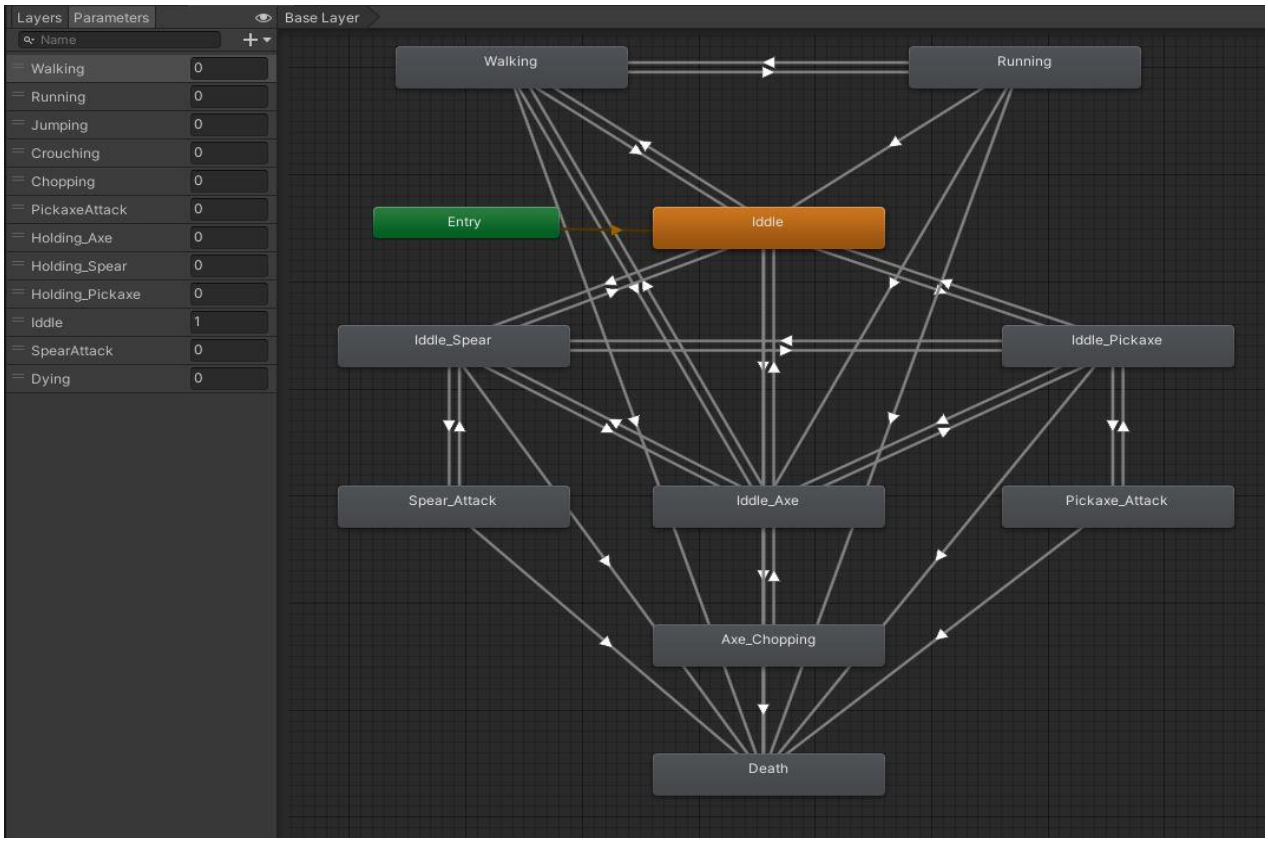
Za rješenje kretnje igrača izrađena je skripta preko koje se prilikom registracije „W,A,S,D“ i „Space“ tipke na tipkovnici izvrši pomak igrača. Potrebno je na model igrača nadodati komponentu “Character Controller” koja olakšava pomicanje modela s obzirom na druge kolizije u sceni. Pritiskom tipke „W“ i „S“ registrira se vertikalni pomak te se popunjava trodimenzionalni vektor sa vrijednosti pomaka naprijed pomnoženog sa duljinom puta te u pozitivnoj ili negativnoj vrijednosti s obzirom na pomak naprijed ili natrag. Ista stvar se odvija i kod horizontalnog pomaka. Tada se poziva metoda navedene komponente i unašaju se vrijednosti pomaka pomnožene sa određenom brzinom kretnje igrača. Tijekom pomaka šalje se nevidljiva zraka od sredine komponente do određene udaljenosti, ako je refleksija te zrake

pod kutom, prepoznaje se da je igrač na neravnom terenu te se poziva druga metoda koja konstantno vuče igrača prema dolje do kad se ne vrati na ravnicu. Što se tiče skakanja, namješta se vrijednost u obliku grafa kako će objekt putovati dok je u zraku te vrijednost jačine skoka. Provjerava se istinitost varijable „isJumping“ te u istinom slučaju poziva se metoda koja pomiče igrača u zrak te ga spušta sve do kad igrač ne udari neku koliziju ili se spusti na tlo.

```
    private void PlayerMovement() {
        float horizInput = Input.GetAxis(horizontalInputName);
        float vertInput = Input.GetAxis(verticalInputName);
        Vector3 forwardMovement = transform.forward * vertInput;
        Vector3 rightMovement = transform.right * horizInput;

charController.SimpleMove(Vector3.ClampMagnitude(forwardMovement +
rightMovement, 1.0f) * movementSpeed);


        if ((vertInput != 0 || horizInput != 0) && OnSlope())
        {
            charController.Move(Vector3.down * charController.height
/ 2 * slopeForce * Time.deltaTime);
        }
    }
    public IEnumerator JumpEvent()
    {
        charController.slopeLimit = 90.0f;
        float timeInAir = 0.0f;


        do
        {
            float jumpForce = jumpFallOff.Evaluate(timeInAir);
            charController.Move(Vector3.up * jumpForce *
jumpMultiplier * Time.deltaTime);
            timeInAir += Time.deltaTime;
            yield return null;
        } while (!charController.isGrounded &&
charController.collisionFlags != CollisionFlags.Above);
        charController.slopeLimit = 90.0f;
        isJumping = false;}
```

3.2.2. Animacije

Prilikom dodavanja izvornih modela u mapu projekta potrebno je namjestiti postavke animacija i označiti izvodi li se animacija u petlji ili samo jednom. Potrebno je kreirati novi objekt „Animation Controller“ koji koristeći stanja strojeva daje mogućnost postavljanja animacija modela kao stanja između kojih kostur modela prelazi na temelju jednakosti između postavljenih varijabli i brojeva.

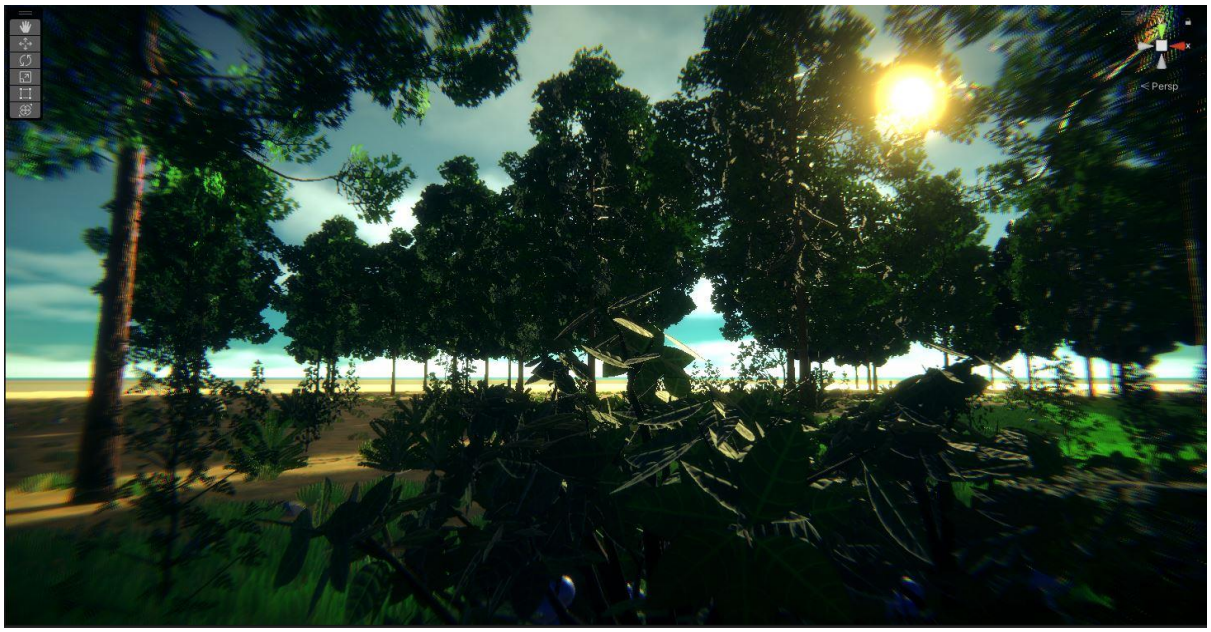


Slika 12. Animation Controller

Kostur modela igrača prvobitno ulazi u stanje mirovanja (eng. Idle state) gdje je zadana animacija držanja ruka u zraku. Mreža poveznica sa strelicama sastoji se od pojedinačnih jednosmjernih veza gdje se jednakošću prelazi iz jednog u drugo stanje. Unity automatizira pokret kostura iz jednog mjesta na drugo i obavlja prijelaz animacije. Kako bi kontroler mijenjao animacije, potrebno ga je povezati skriptom, te ovisno o pritisku određenih tipki tipkovnice skripta mijenja vrijednost varijable. Koristeći ova stanja, igraču je omogućeno da u normalnom stanju drži ruke u zraku a prilikom odabira kreiranih alata drži koplje, sjekiru ili kramp u ruci od kojih svaki ima animaciju za obradu resursa. Držanjem jedne od tipka za hodanje izvodi se animacija hodanja te pritiskom tipke „Shift“ prelazi se u animaciju za trčanje, te isto tako pritiskom tipke „Space“ vrijedi i za skakanje. Naposlijetku prilikom gubitka vode ili hrane prelazi se u animaciju rušenja igrača na pod.

3.3. Okoliš i vegetacija

Unity nudi implementirani set alata za dodavanje terena u projekt, te se stvaranjem novog objekta „Terrain“ koji sadrži komponentu „Terrain“ i „Terrain collider“ stvara početni krajolik. Pomoću „Terrain collider“ komponente dodaje se kolizija koju igrač može detektirati. Koristeći komponentu „Terrain“ mijenjamo svojstva koja se dijele na kratice. Koristeći razna svojstva kratica poput kistova za podizanje i spuštanje dijelova terena, kistova za postavljanje veće količine modela vegetacije i kistova za postavljanje detalja poput trave i cvijeća, kreiran je okoliš u obliku otoka na kojemu igrač preživljava.



Slika 13. Okoliš

3.3.1. Interaktivni okoliš

Kako bi igrač ostvario interakciju sa stablom, kamenom, bobicama i granama, trebamo navedenim objektima pridružiti neka svojstva, za što će nam poslužiti par skripta. Da bi pridodali skripte objektima, nužno je proći listu svih drveća zapisanih u komponenti „Terrain“. Komponenta ne dozvoljava da objekti koji se postavljaju na teren sadrže skripte kako bi se postigle optimalne performanse. Umjesto direktne komunikacije sa objektima vegetacije, moguće je proći kroz listu svih drveća zapisano u komponenti, za svaku instancu objekta pročitati njegovu lokaciju na terenu i kreirati kolizijsku kapsulu na njegovom mjestu. Svaki dodani objekt na terenu ima svoj prototip izvornog objekta i svoj indeks u redu, time je moguće odrediti kakva će svojstva oblika poprimiti kreirana kolizijska kapsula, skripta koje će se

nadodati, njena oznaka i naziv. Sve kapsule smjestiti će se u poseban objekt u hijerarhiji scene. Sve objekte vegetacije s kojima želimo interakciju spremamo u listu drveća terena.

```
    private void SpawnTreeColliders()    {
        _originalTrees = terrainForSpawn.terrainData.treeInstances;
        for (int i = 0; i < 
terrainForSpawn.terrainData.treeInstances.Length; i++)
        {
            TreeInstance treeInstance = 
terrainForSpawn.terrainData.treeInstances[i];
            GameObject capsule = 
GameObject.CreatePrimitive(PrimitiveType.Capsule);
            CapsuleCollider capsuleCollider = 
capsule.GetComponent<CapsuleCollider>();
            if (treeInstance.prototypeIndex == 0) {
                capsule.name = "Tree";
                capsule.tag = "LongwoodTree";
                capsuleCollider.center = new Vector3(0, 10, 0);
                capsuleCollider.height = 20;
                capsuleCollider.radius = 0.5f;
                TreeStats tree = capsule.AddComponent<TreeStats>();
                tree.treeIndex = i;
                capsule.transform.position = 
Vector3.Scale(treeInstance.position, terrainForSpawn.terrainData.size);
                capsule.transform.parent = WorldColliders.transform;
            }
              if (treeInstance.prototypeIndex == 1) {
            … }
            capsule.transform.parent = WorldColliders.transform;
            rend = capsule.GetComponent<Renderer>();
            rend.enabled = false;
            capsuleList.Add(capsule); } }
```

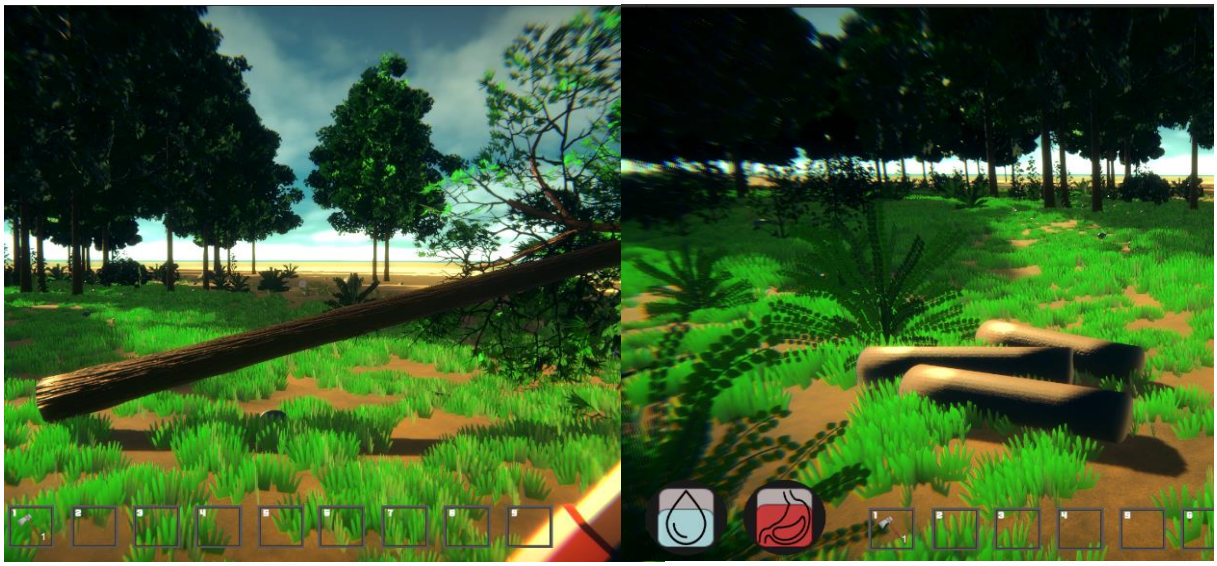
3.3.2. Prikupljanje resursa

Sada igrač može detektirati objekt ispred sebe i komunicirati sa skriptama. Nakon što igrač sruši drvo, ili iskopa kamen, kako bi spriječili incident nestanka materijala, svaka skripta zadužena za event uništenja objekta vegetacije treba pozivati metodu dodavanja objekta klase u listu za sve objekte koji se trebaju ponovno postaviti na teren nakon određenog vremena. Koristeći integriranu metodu InvokeRepeating() zadajemo koja metoda će se pozivati, u kojem početnom vremenu, i u kojem vremenskom razmaku. Tako će se svaku minutu pozivati metoda za ponovno postavljanje objekta na teren, i kod svakog poziva provjeravati proteklo vrijeme za svaki objekt vegetacije u listi. Ako je određeno vrijeme prošlo, kreira se nova instanca drva za

teren, inicijaliziraju se vrijednosti visine, širine, rotacije, pozicije i indeksa prototipa s obzirom na vrstu objekta. Instanca se dodaje u popis drveća terena, omogućuje se njezin kolizijski objekt i briše se iz liste objekata za ponovno postavljanje.

```
private void RespawnTree()
    {
        if (managedTrees.Count == 0) {
            return; }
        for (int cnt = 0; cnt < managedTrees.Count; cnt++)
        {
            if (managedTrees[cnt].respawnTime < Time.time)
            {
                TreeInstance treeTemp = new TreeInstance
                {
                    position = managedTrees[cnt].terrainPosition
                };
                if(managedTrees[cnt].treeName == "SmallRock")
                {
                    treeTemp.prototypeIndex = 3;
                    float treeRot = Random.Range(0f, 360f) * Mathf.Rad2Deg;
                    treeTemp.rotation = treeRot;
                    treeTemp.widthScale = Random.Range(0.5f, 1f);
                    treeTemp.heightScale = Random.Range(0.5f, 1f);
                }
                else if (managedTrees[cnt].treeName == "Branch")
                {
                    treeTemp.prototypeIndex = 4;
                    treeTemp.widthScale = 1f;
                    treeTemp.heightScale = 1f;
                }
                 …
                treeTemp.color = Color.white;
                treeTemp.lightmapColor = Color.white;
                managedTrees[cnt].terrain.AddTreeInstance(treeTemp);

managedTrees[cnt].marker.gameObject.GetComponent<CapsuleCollider>().enabled
= true;
                managedTrees.RemoveAt(cnt);} } }
```

Svako drvo, kamen, grana i grmlje bobica u svijetu igrača mora osim svojeg kolizijskog objekta imati i skriptu svojstva. Izvorna skripta nalazi kod stabla, a svi ostali objekti sadrže varijaciju te skripte. Želimo da igrač nakon određenog broja interakcije poput cijepanja drva uništi objekt vegetacije i pokupi resurse na određeni način. Kako bi to postigli, svakom objektu pridodaje se njegov broj života. Svaki puta kada naprimjer igrač udara sjekirom, poziva se metoda unutar skripte stabla koja za parametar uzima određenu vrijednost štete koju igrač šalje te koja se oduzima od ukupnog života stabla. Ako se vrijednost života spusti na nulu, poziva se metoda koja prvo pamti sve potrebne podatke o objektu poput indexa, terena, pozicije i slično. Kolizijski objekt se isključuje jer objekt više ne postoji, na njegovom mjestu se postavlja marker koji označuje mjesto, te se objekt dodaje u prethodno navedenu listu objekata za ponovno postavljanje i miče iz liste drveća terena. Na kraju ovisno o vrsti objekta zadajemo što će se desiti, naprimjer kada igrač isječe stablo, ono se razdijeli na dva dijela i krene padati na tlo, gdje se nakon određenog vremena stvori cjepanica koju igrač može pokupiti kao resurs. S obzirom da se radi o stvaranju objekta u sceni usred pokrenute igre, objekti koji će se stvoriti moraju biti spremljeni u posebnu mapu „Resources“, te ako objekt koristi zakone fizike i gravitacije potrebno je da u sebi sadrži podešenu komponentu „Rigidbody“. Kad se stablo isječe, zamjenjuje se objektom stabla podijeljeno na panj i drvo, te se stablu pridodaje masa i impulzivna sila koja započinje rušenje drva. Poziva se potprogram koji čeka 10 sekundi prije nego uništi srušeno drvo, te se na njegovom mjestu stvore 3 cjepanice.



Slika 14. Prikupljanje resursa drveća

Kako bi igrač mogao komunicirati sa okolnim objektima vegetacije, potrebna mu je funkcionalnost koja će se brinuti o baratanju informacija objekta komunikacije. Za to će poslužiti skripta koja kod interakcije poput udarca alatom ili prikupljanja resursa, ispušta

nevidljivu zraku koja prati postoji li kakav kolizijski objekt ispred igrača te ako postoji, ovisno o oznaci i komponenti koju objekt sadrži, poziva prikladnu metodu za interakciju. Metoda zapisuje sve potrebne informacije objekta u varijable, te poziva metodu iz skripte vegetacijskog objekta kojom objektu oduzima broj života. Ako se broj života smanji na nulu, pokreće se proces prethodno objašnjenog uništavanja objekta.

```
    public void DestroyTree(Terrain _terrain, int _treeIDX, Vector3
_treePos, Vector3 _treeScale,Vector3 _transformForce)
    {
        Vector3 treePos = _treePos;
        Vector3 transformForce = _transformForce;
        int treeIDX = _treeIDX;
        Terrain terrain = _terrain;
        gameObject.GetComponent<Collider>().enabled = false;
        GameObject marker = gameObject;


        marker.transform.position = treePos;
        Vector3 terrainPosition =
terrain.terrainData.treeInstances[treeIDX].position;
        rMgr.AddTerrainTree(terrain.name, treeIDX, Time.time +
respawnTimer, marker.transform, terrain, terrainPosition, "LongwoodTree");
        List<TreeInstance> trees = new
List<TreeInstance>(terrain.terrainData.treeInstances);
        trees[treeIDX] = new TreeInstance();
        terrain.terrainData.treeInstances = trees.ToArray();
        GameObject treeFall = Instantiate(fallTree,
this.transform.position, this.transform.rotation);
        treeFall.transform.localScale = _treeScale;
        GameObject treeStump = treeFall.transform.GetChild(0).gameObject;
        health = 100;
        treeStump.GetComponentInChildren<Rigidbody>().mass = 300;

treeStump.GetComponentInChildren<Rigidbody>().AddForce(transformForce *
200, ForceMode.Impulse);
        StartCoroutine(WaitForTreeFall(treeFall));
  }
```

3.3.3. Dan i noć

Unity HDRP verzija nudi poboljšanu implementacije alata za kontrolu svijetlosti i omogućuje izradu realističnog neba. Koristeći te alate možemo igraču izraditi sustav dana i noći. Kreiramo dva objekta i pridodajemo im komponentu „Light“ u kojoj postavljamo svojstva nalik suncu i mjesecu. Preko skripte ćemo upravljati objektima te profilom neba i magle. Na svakoj metodi Update() broji se vrijeme dana od 0 do 24 na temelju brzine orbite i vremena koje je prošlo. Tada se poziva metoda koja rotira mjesec i sunce oko terena, provjerava koje je doba dana i odabire koji objekt baca sjenu. Kada sunce padne ispod zemlje, polako se pojačava i intenzitet svijetlosti zvijezda preko noći pa do dana.

Slika 15. Sunce i mjesec

```
    private void UpdateTime()
    {
        float alpha = timeOfDay / 24.0f;
        float sunRotation = Mathf.Lerp(-90, 270, alpha);
        float moonRotation = sunRotation - 180;

        sun.transform.rotation = Quaternion.Euler(sunRotation, -150.0f, 0);
        moon.transform.rotation = Quaternion.Euler(moonRotation, -150.0f,
0);

        sky.spaceEmissionMultiplier.value = starsCurve.Evaluate(alpha) *
starsIntensity;

        CheckNightDayTransition();}
```

3.4. Upravljanje resursima

Temelj mehanika igre svodi se na prikupljanje resursa pomoću kojih igrač može izraditi potrebne alate, logorsku vatru i dijelove kuće. Za to će nam pomoći alat „Canvas“ koji se ponaša kao platno za prikazivanje 2D slika. Pomoću njega igrač će imati svoj inventar resursa, interaktivne ikone za izradu alata, inventar za opremanje alata, stanje gladi i žeđi, te popis zadataka igre.

3.4.1. Inventar resursa

Inventar se sastojati od pozadinske slike na koju se prilijepi objekt sa slikom okvira na njemu određeno mjesto u nizu. Postoji dvadesetak mjesta u nizu u koje igrač smješta resurse prilikom prikupljanja. Kako bi se svaki resurs mogao pravilno razvrstati, potrebno je kreirati skriptu za prihvat svakog resursa koja će provjeravati čitav niz i njihova stanja. Svaki puta kada se dodaje novi resurs, provodi se provjera u metodi AddItem(), gdje se prvo prolazi kroz sva mjesta u nizu i pregledava njihovo stanje popunjenosti. Ako je mjesto prazno, dodaje se u popis praznih mjesta, suprotno ako je mjesto popunjeno istim resursom i maksimalni kapacitet nije popunjen, mjesto se dodaje u listu za punjenje kapaciteta. Ako postoji mjesto za popuniti kapacitet, broj novih resursa nadodaje se postojećem sve dokle postoji kapaciteta te u slučaju ostatka on se sprema na prvo sljedeće prazno mjesto. U slučaju da resurs ne postoji od prije, dodaje se na prvo sljedeće slobodno mjesto u nizu. Skripta sadrži i metodu za provjeru količine zadanog resursa, te metodu za snižavanja kapaciteta resursa koju će koristiti drugi dijelovi sustava za upravljanje resursa kako bi igrač imao mogućnost trošenja resursa.

Slika 16. Inventar resursa

3.4.2. Inventar opreme

Princip ovog inventara sličan je prijašnje objašnjenom inventaru resursa. Izradit ćemo još 9 slobodnih mjesta u nizu te im dodijeliti broj tipke na tipkovnici od jedan do devet. Kada igrač pritisne jednu od navedenih tipka, određeno mjestu u nizu će se označiti te provjeriti postoji li alat na tom mjestu. Ako alat postoji, javlja se animatoru modela da prebaci kostur u drugo stanje namijenjeno za određeni alat, te se aktivira trenutno opremljen alat u sceni koji igrač drži u ruci. Potrebno je i napomenuti na koji način svako mjesto u inventarima radi. Prilikom prvog pokretanja skripte svakog mjesta, pamti se određena 2D slika i postavlja se broj dostupnih resursa. U svakoj Update() metodi konstantno se provjerava postoji li više od jednog objektnog djeteta objekta mjesta, s obzirom da svako mjesto već ima jedno objektno dijete grafike u koju se smješta ikona i tekst, te ako postoji čitaju se i postavljaju svojstva broja resursa i ikona resursa smještenog u određenom mjestu.

Slika 17. Inventar opreme

3.4.3. Izrada alata

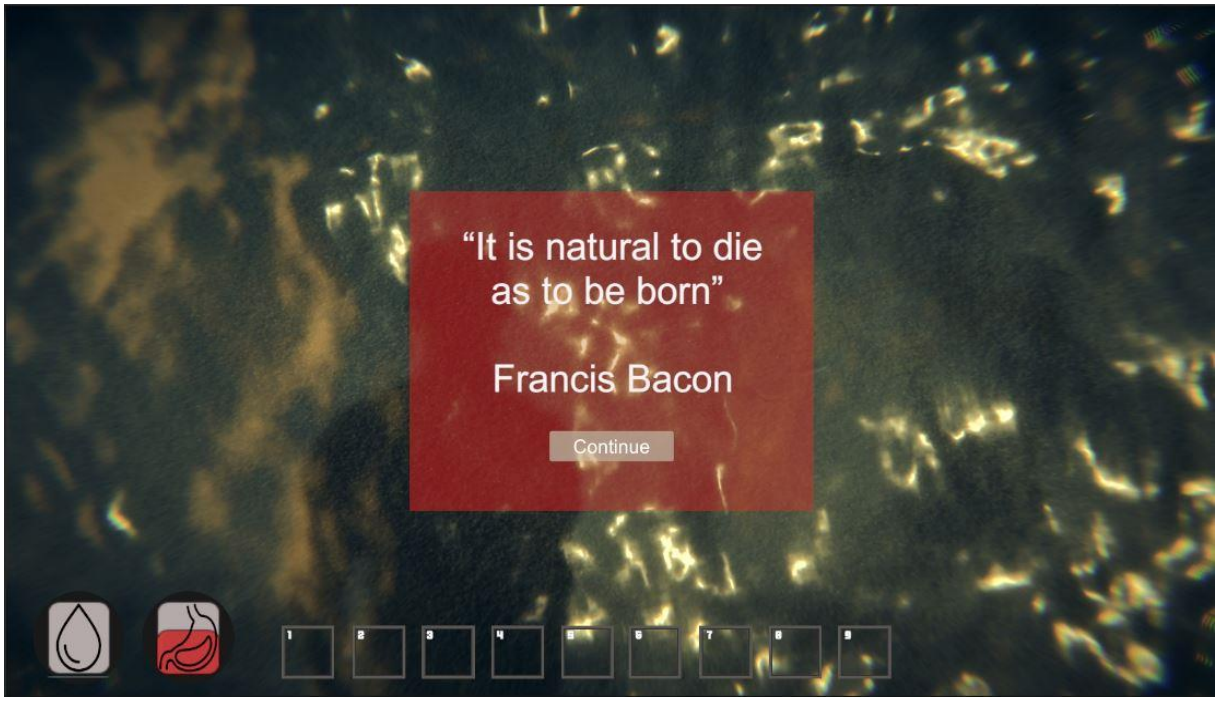
Ideja je da igrač u isto vrijeme vidi svoj inventar svih resursa, te desno od njega vidi listu svih stvari koje može izraditi potrošnjom tih resursa. Ta sekcija sastojat će se od pozadinske slike i interaktivnih ikona. Svaka ikona ima pridruženi tekst broja potrebnih resursa, te u slučaju klika na ikonu, provjerava se inventar igrača za sve potrebne resurse. Ako igrač ima sve što je potrebno, određeni broj resursa se miče iz inventara i dodaje se izrađena stvar.

Slika 18. Izrada alata

```
public void CraftItem()  {
        foreach(RequiredItem i in itemsNeeded) {
            if (inv.GetItemAmount(i.itemID) < i.amountNeeded)
                return; }

        foreach(RequiredItem i in itemsNeeded) {
            inv.RemoveItemAmount(i.itemID, i.amountNeeded); }

        Item z = Instantiate(craftedItem, Vector3.zero,
Quaternion.identity).GetComponent<Item>();
        z.amountInStack = craftedItemAmount;
        inv.AddItem(z,null,false,true);
    }
```

3.4.4. Život igrača

Tijekom preživljavanja na otoku igrač treba brinuti o svojem statusu gladi i žeđi koji ako padnu na nulu završavaju igru i igrač mora krenuti ispočetka. Da bi igrač to spriječio, može prikupljati bobice sa grma i loviti pa zatim ispeći ribu na logorskoj vatri. Prikupljene resurse može konzumirati desnim klikom miša u inventaru resursa i podići razinu statusa za određeni dio. Kako bi se to realiziralo, potrebno je na platno postaviti dva klizača objekta. Rotirat ćemo ih tako da je vrijednost klizača postavljena na sto i nalazi se gore, a vrijednost nula postavljena dolje. Postavimo im ikone asocijacije na trbuh i kap vode, zadamo plavu boju klizača za žeđ i crvenu za glad kako bi im igrač odmah mogao prepoznati namjenu. Oni će se ponašati kao trenutni status gladi i žeđi te će pridodana skripta upravljati za koliko se vrijednost smanjuje svaki trenutak. Ako igrač konzumira ribu ili bobice tada će se statusi povisiti i smanjiti za određenu vrijednost hrane ne prelazeći pritom maksimalnu vrijednost statusa. Također ako igrač krene trčati, status žeđi će se brže snižavati. Skripta u svakom trenutku za određeni broj smanjuje varijable statusa i provjerava prelaze li nulu. U slučaju prelaska animatoru se javlja da se prebaci u stanje igračeve smrti i pokrene zadanu animaciju. Na ekranu se prikazuje citat i gumb za povratak u glavi izbornik.



Slika 19. Završetak igre

```
void Update()
    {
        if(Hunger > 0)
        {
            Hunger -= HungerDrainer;
            if(Hunger < 0)
            {
                Hunger = 0;
                playerAnimator.PlayerDeath();
                this.gameObject.GetComponent<PlayerLook>().enabled = false;
                playerMovement.enabled = false;
                Cursor.lockState = CursorLockMode.None;
                Cursor.visible = true;
                DeathUI.SetActive(true);
            }
        }
              …
    }
   public void RestoreBar(float hunger,float thirst)
    {
        Hunger += hunger;
        Thirst += thirst;
        if(Hunger > 100)
        {
            Hunger = 100;
        }
        if (Thirst > 100)
        {
            Thirst = 100;
        }
    }
```

3.4.5. Riba

Glad je status igrača koji se može generalno održati samo putem konzumacije ribe. Ribe se nalaze oko otoka u vodi te ih igrač može uloviti koristeći koplje. Da bi riba bila realistična treba imati svoje jato te kretnje u raznim smjerovima. Za realizaciju je potrebno stvoriti objekt koji će se ponašati kao stvoritelj jata riba i implementirati umjetnu inteligenciju kretanja pojedine ribe. Svaki stvoritelj ima svoju skriptu u kojoj prvo postavljamo dimenzije prostora u kojem se ribe mogu stvoriti i kretati, vrijeme između stvaranja novih riba te objekt

ribe koji želimo stvoriti. Za svaku dodanu ribu nasumično ili ručno postavljamo limit ukupnog broja riba i maksimalan broj riba koje se stvaraju u trenutku. Skripta kod pokretanja poziva metodu za prikupljanje objekta svih pozicija prema kojima se mogu kretati ribe unutar zadanog prostora. Ako su postavljene nasumične kretnje ribe tada lista objekata ostaje prazna. Postavimo li nasumičnu generaciju brojeva za vrijednosti, ona se izvodi sljedeća. Nadalje kreiraju se grupe svih dodanih objekta riba te se za određeno vrijeme između stvaranja novih riba ponavlja metoda za stvaranje. Metoda SpawnNPC() pokreće petlju za svaki dodani objekt ribe i pretražuje koliko djece ima pripadajuća grupa uspoređujući broj sa vrijednosti limita broja riba. Dokle god nije postignut limit, stvara se maksimalni broj riba u trenutku pri čemu se pozivaju metode za nasumičnu poziciju ribe i nasumični smjer kretanja.

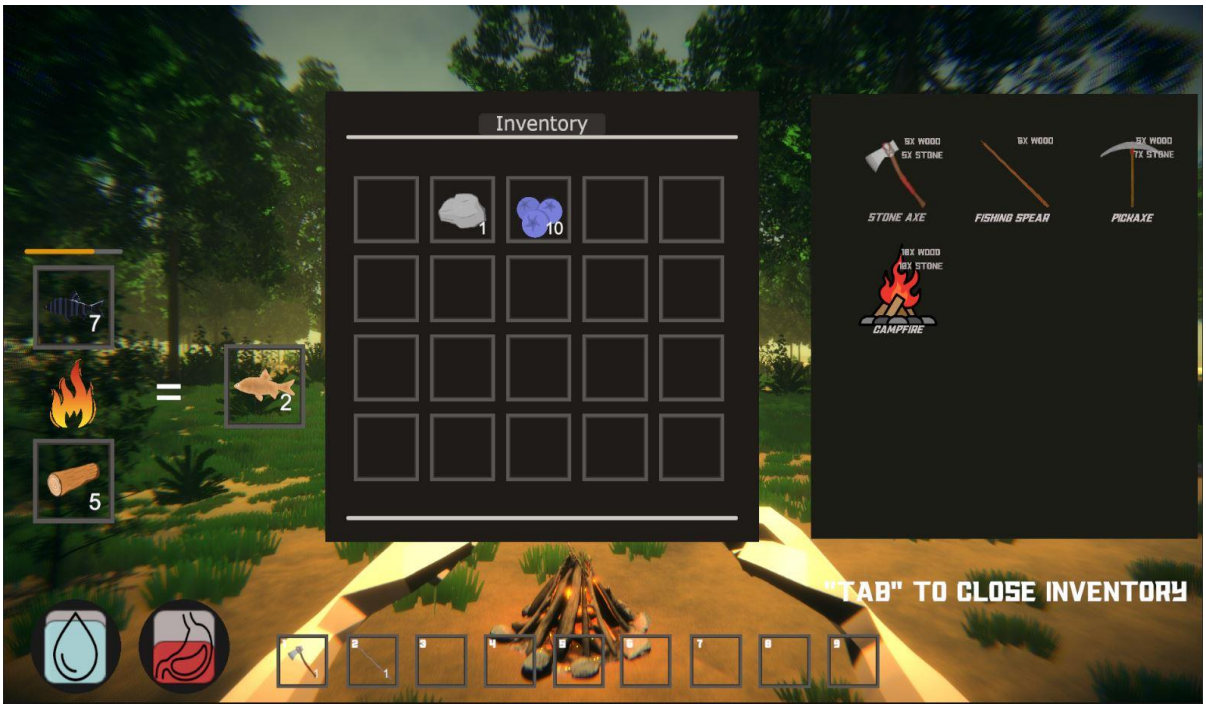
```
void SpawnNPC(){
        for(int i = 0; i < AIObject.Count(); i++) {
if(AIObject[i].enableSpawner && AIObject[i].objectPrefab != null){
                GameObject tempGroup =
this.transform.Find(AIObject[i].AIGroupName).gameObject;

if(tempGroup.GetComponentInChildren<Transform>().childCount <
AIObject[i].maxAI) {
    for(int y = 0; y < AIObject[i].spawnAmount; y++) {
        Quaternion randomRotation = Quaternion.Euler(Random.Range(-20, 20),
Random.Range(0, 360), 0);
        GameObject tempSpawn;
        tempSpawn = Instantiate(AIObject[i].objectPrefab, RandomPosition(),
randomRotation);
        tempSpawn.transform.parent = tempGroup.transform;
        tempSpawn.AddComponent<AIMove>();}}}}}


public Vector3 RandomPosition(){
        Vector3 randomPosition = new Vector3(Random.Range(-spawnArea.x,
spawnArea.x), Random.Range(-spawnArea.y, spawnArea.y), Random.Range(-
spawnArea.z, spawnArea.z));
        randomPosition = transform.TransformPoint(randomPosition * .5f);
        return randomPosition; }
```

Svaka stvorena riba sadrži svojstva modela, kostur, animaciju plivanja koju animator pušta u petlji, kolizijski model i skriptu za umjetnu inteligenciju. Kod stvaranja objekta ribe nasumično se skalira njena veličina i traži se kolizijski objekt u djeci objekta ribe. U svakom trenutku provjerava se ima li riba svoju metu prema kojoj se kreće, ako nema provjera se jeli riba stigla do svog odredišta. U slučaju da je riba stigla na odredište, dodajemo joj novu metu prema kojoj će se kretati, suprotno ako nije, nasumično se postavlja brzina kretnje i određuje se odredište kao zadnja meta prema kojoj se riba kretala prije zastoja. Riba koja ima dodijeljeno odredište u svakom se trenutku rotira preko metode za sfernu interpolacije točaka između trenutne rotacije ribe i odredišta za omjer brzine okretanja pomnoženog vremenskim intervalom od zadnjeg okvira(eng. *Frame*). Ribi se konstantno dodaje nova pozicija preko metode Vector3.MoveTowards() gdje se od trenutne pozicije pomiče do odredišta za duljinu od brzine ribe pomnoženo sa vremenskim intervalom od zadnje okvira. Kada riba putuje ispred nje se baca nevidljiva zraka i prati nalazi li se ispred nje odredište. Ako objekt ispred ribe nije odredište, generira se nasumičan broj od jedan do sto, ako ispadne broj manji od četrdeset riba traži novu metu, te se isto desi ako riba naleti na teren. Navedenom logikom postiže se da kada stvorimo jato riba u moru, nasumično se kreću okolinom i pritom reagiraju na nju.

Slika 20. Lov na ribe

Kada igrač ulovi ribu ona se sprema u njegov inventar, no pojede li ju sirovu izgubit će na statusu žeđi i povisiti za mali udio status gladi. Prije konzumacije za puni potencijal ribu je potrebno ispeći na prethodno kreiranoj logorskoj vatri. Interakcijom sa logorskom vatrom

igraču se otvara inventar resursa i dio za kuhanje. Za kuhanje postoje tri potrebna mjesta za proces. Prvo mjesto služi kao ulazno mjesto za svježu ribu gdje igrač iz inventara resursa prenosi sakupljenu ribu. Na drugo mjesto se postavlja količina drva za vatru nakon čega drva počinju sagorijevati i logorska vatra počinje isijavati svijetlost vatre. Nakon što prođe određeno vrijeme smanjuje se broj dodanih drva za jedan. Nakon što ponestane dodanih drva proces kuhanja se zaustavlja i logorska vatra se gasi. Ako su oba mjesta popunjena započinje proces kuhanja. Nakon što vrijednost klizača iznad svježe ribe dođe od sto do nula, broj unesenih riba se smanjuje za jedan, te se na treće mjesto dodaje jedna kuhana riba.



Slika 21. Kuhanje ribe

3.5. Građevine

Gradnja skloništa izvodi se otvaranjem menija za gradnju pritiskom tipke „Q“ na tipkovnici. Igraču se na odabir nudi temelj, krov, stepenice i zid. Svaki objekt zahtijeva određeni broj drva kako bi se mogao postaviti. Igrač prvo mora izgraditi temelj da bi mogao na njega graditi ostale objekte i sagraditi sklonište. Objekt se može postaviti na sredinu ekrana pomakom miša ili ručno negativnim i pozitivnim pomakom kroz X, Y i Z dimenzije trodimenzionalnog prostora.

Slika 22. Meni za gradnju

Sustav gradnje sastoji se od dvije skripte. U skriptu zaduženu za postavljanje objekta učitavaju se svi dostupni objekti za gradnju. Kod pritiska tipke „Q“ igraču se prikazuje meni te odabire željena građevina. Poziva se metoda koja baca nevidljivu zraku od sredine ekrana u smjeru gdje igrač gleda i ako zraka dodiruje teren, zapisuje se lokacija dodira. Na mjestu dodira prikazuje se objekt pregleda na čijem mjestu se lijevim klikom miša gradi građevina. Kako ne bi došlo do intersekcije između dva modela objekta, postavlja se vrijednost koordinatne mreže gdje svaki objekt ima svoje dimenzije unutar skupa ćelija mreže te kolizijski model. Svaki objekt postavlja se na početak ćelije i plavom bojom indicira jeli moguća gradnja na toj poziciji. Ako objekt pregleda prepoznaje koliziju s drugim objektom na tom mjestu, postaje crvene boje i brani igraču da gradi objekt. Prilikom postavljanja građevine kada je to dozvoljeno, pregledava se igračev inventar resursa za zahtijevani broj resursa i ako se pronađe, postavlja se građevina

i smanjuje se broj resursa u inventaru. Druga skripta koristi se kod otvaranja menija za gradnju. Igraču se na sredini ekrana prikazuje krug gdje svaku četvrtinu kruga reprezentira jedna građevina. Zapisuje se trenutna pozicija miša u dvodimenzionalnom prostoru ekrana, te se X koordinata dijeli sa širinom ekrana a Y koordinata sa visinom ekrana. Dijeli se broj ukupnih objekata sa stupnjevima punog kruga i izračunava se kut između od početka kruga do pozicije miša. Dobivena vrijednost kuta se dijeli sa prethodno navedenom vrijednosti i pretvara u cijeli broj. Taj broj se zapisuje i reprezentira trenutni indeks odabranog objekta.

```
 public void GetCurrentMenuItem()
    {
        mousePosition = new Vector2(Input.mousePosition.x,
Input.mousePosition.y);
        toVector2M = new Vector2(mousePosition.x / Screen.width,
mousePosition.y / Screen.height);


        float angle = (Mathf.Atan2(fromVector2M.y - centerCircle.y,
fromVector2M.x - centerCircle.x) - Mathf.Atan2(toVector2M.y -
centerCircle.y, toVector2M.x - centerCircle.x)) * Mathf.Rad2Deg;


        if(angle < 0)
        {
            angle += 359.99f;
        }
        CurMenuItem = (int)(angle / (360 / menuItems));
        if(CurMenuItem != OldMenuItem)
        {
            buttons[OldMenuItem].sceneImage.color =
buttons[OldMenuItem].NormalColor;
            OldMenuItem = CurMenuItem;
            buttons[CurMenuItem].sceneImage.color =
buttons[CurMenuItem].HighlightedColor;
        }


    }
```

4. Zaključak

Zadatak završnog rada bio je razviti igru preživljavanja u alatu Unity, te je osobni cilj bio postići taj rezultat bez plaćanja bilo kakvih licenca ili paketa, vodeći se idejom kvalitetnog realizma unutar igre.

U početku je bilo teško pohvatati toliko široko područje razvoja uključujući ne samo glavni alat za izradu igre i programiranje, već i pronalazak te korištenje pomoćnih alata za razvoj modela i prikupljanje besplatnog materijala za igru. Stjecanjem znanja i vještina preko raznih videa, uputstva, foruma i dokumentacije, te mnogih neuspjelih pokušaja razvoj je postao sve lakši i razumljiviji. Mnoge stvari koje su se u početku činile nemogućim za izradu od strane jedne osobe postale su samo vremenski izazov te sam ostao uporan i postigao zadovoljavajuće finalne rezultate. Tako je su savladani alati Unity, Blender, Gimp i Materialize što većinski obuhvaća kompletan razvoj igre.

Uspješnim završetkom projekta dokazano je kako su danas besplatno dostupne sve potrebne tehnologije i dovoljno je znanja da svaki individualac može krenuti u svijet razvoja igara te uz dovoljno truda postići kvalitetne rezultate. Nije potrebno kupovati gomile raznih visoko kvalitetnih paketa modela, pomoćnih programa i kodova već samostalnim radom i prelaženjem preko neuspjeha moguće je realizirati bilo kakvu ideju. Izrazito sam zadovoljan rezultatom rada i planiram razvijati ovaj projekt i dalje, te potencijalno jednog dana izraditi i objaviti kompletnu igru na web.

Popis literature

[1] The History of Survival Games - From Archaic to ARK. Pristupljeno 10. rujna, 2021. https://survivethis.news/en/history-of-survival-games/

[2] The Forest | Endnight Games. Pristupljeno 10. rujna, 2021. https://endnightgames.com/games/the-forest

[3] What is Unity? Everything you need to know - Android Authority. Pristupljeno 12. rujna, 2021. https://www.androidauthority.com/what-is-unity-1131558/

[4] About — blender.org. Pristupljeno 10. rujna, 2021. https://www.blender.org/about/

[5] GIMP - GNU Image Manipulation Program. Pristupljeno 12. rujna, 2021. https://www.gimp.org/

[6] Bounding Box Software - Materialize. Pristupljeno 12. rujna, 2021. https://boundingboxsoftware.com/materialize/

[7] What is PBR and What a 3D Artist Should Know - CG Obsession. Pristupljenog 13. rujna, 2021. https://cgobsession.com/what-is-pbr-and-what-a-3d-artist-should-know/

[8] Texture Maps: The Ultimate Guide For 3D Artists. Pristupljeno 12. rujna, 2021. https://conceptartempire.com/texture-maps/

[9] What is BLOOM in video Games Graphics Settings? Pristupljeno 13. rujna, 2021. https://explainopedia.com/what-is-bloom-in-games/

[10] What is Chromatic Aberration? - Graphics Settings Explained. Pristupljeno 12. rujna, 2021. https://www.game-debate.com/news/28265/what-is-chromatic-aberration-graphics-settings-explained

Popis slika