Razvoj akcijske igre „Survive It“ u Unity okruženju

Čabraja, Ivan

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Ivan Čabraja

RAZVOJ AKCIJSKE IGRE „SURVIVE IT“ U UNITY OKRUŽENJU

Završni rad

Pula, rujan 2023.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Ivan Čabraja

RAZVOJ AKCIJSKE IGRE „SURVIVE IT“ U UNITY OKRUŽENJU

Završni rad

JMBAG: 0303099380, izvanredan student

Studijski smjer: Informatika

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Kolegij: Programiranje

Mentor: Izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

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

Student
Ivan Čabraja

U Puli, 17.09.2023 godine

IZJAVA o korištenju autorskog djela

Ja, IVAN ČABRAJA dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Razvoj akcijske igre „Survive It“ u Unity okruženju

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice

Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

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

U Puli, 17.09.2023 godine

Potpis

Ivan Čabraja

SADRŽAJ

UVOD

Industrija igara je danas među najpopularnijim i najunosnijim industrijama. Pogotovo zahvaljujući lako dostupnim i besplatnim tehnologijama koje omogućuju da svatko da svoj doprinos. Unity razvojno okruženje je jedan primjer toga koji se razvijao i pratio rast tehnologije, a pritom olakšavao programerima bilo koje razine da razviju svoje projekte.

Industrija igara predstavlja jedan dinamičan krajolik koji se brzo razvija i neprestano pomiče granice tehnologije i kreativnosti. U srcu ovog univerzuma koji se stalno širi leži mnoštvo alata za razvoj igara, od kojih svaki doprinosi i oživljava virtualne svjetove. Među tim razvojnim okruženjima, Unity se ističe kao snažna i svestrana sila, koja pokreće inovacije i omogućuje programerima da osmisle impresivna i vizualno zapanjujuća iskustva igranja. U ovom radu pobliže ćemo pojasniti Unity razvojno okruženje, istražujući njegovu ključnu ulogu u oblikovanju igara.

Motivacija pisanja ovog rada je veliko zanimanje za game development područje, te želja za istraživanjem i razumijevanjem ovog uzbudljivog područja koje se stalno razvija. Tema ovog rada je action wave survival stila u top down 2D-u, što bi značilo da se igrica temelji na preživljavanju naleta čudovišta s pogledom kamere „top down“ odnosno da igrač gleda prema dolje igru kako se odvija. Također područje razvoja video igrica uključuje i kompleksnost rješavanja problema. Videoigre su postale značajan dio moderne kulture. Analiza njihovog utjecaja i alata korištenih za njihovo stvaranje može pružiti uvid u kulturne fenomene.

Cilj ovog rada je pobliže opisati i prikazati razvoj akcijske igre „Survive it“. U radu je detaljno opisano planiranje projekta, nadalje razvoj prototipa igre te igrača. U konačnici su i opisane sve komponente igre poput kamera, dostupnog oružja, mapa, čudovišta, projektili i sl. Također kroz rad se opisuje kod koji je korišten kako bi se na što jednostavniji način prikazao proces izrade akcijske igre.

1. UNITY RAZVOJNO OKRUŽENJE

Unity je pokretač za 2D i 3D igre koji postoji od 2005. godine. Razvio ga je Unity Technologies, a napravljen je kako bi većem broju programera omogućio pristup alatima za razvoj igara, što je u to vrijeme bio nov pothvat. Tijekom svog dugog vijeka, Unity razvojno okruženje se dramatično mijenjao i proširivao, uspijevajući držati korak s najnovijom praksom i tehnologijama (Shardon, 2023).

Čak i danas, glavni fokus pokretača igara je pružanje najrobusnijeg mogućeg skupa alata za industriju razvoja igara, kao i što je moguće jednostavnije korištenje programerima igara bilo koje razine. Također ovo razvojno okruženje je proširio svoj doseg na druge industrije s velikim fokusom na 3D razvoj u stvarnom vremenu, što ga čini jednim od najmoćnijih dostupnih motora.

Unity se može koristiti za izradu igrica na platformama kao što su IOS, Android, operacijske sustave (Windows, Mac, Linux) i konzole (Playstation, Xbox). Unity pruža pristup Asset store-u. To je mjesto koje sadrži brojna svojstva za igricu kao što su teksture, modeli, skripte, animacije i slično, također programeri mogu kreirati svoja svojstva.

Kod Unity-a postoji sustav za izraditi 2D ili 3D animacije, uz to se mogu koristiti i drugi programi za ubaciti animacije, to uključuje i poziciju rotacije objekta, a mogu se i manipulirati kosti od objekta.

Glavne značajke Unity razvojnog okruženja su razne, a u nastavku su te značajke i pobliže objašnjene (Unity, 2023).

- **Podrška za 3D i 2D grafiku**

Unity pruža podršku za 3D i 2D grafiku – dopuštajući slobodu odabira umjetničkog stila za projekte. Svaka vrsta grafike dolazi s vlastitim specijaliziranim skupom alata (kao što je rezanje listova spritea za 2D grafiku) i čak ima vlastite API-je za skripte.

3D grafika također nudi iznimno otporan skup alata s mogućnošću stvaranja prilagođenih materijala, izrade shadera s Shader Graphom, prilagodbe osvjetljenja, korištenja efekata naknadne obrade i više. Može se čak generirati 3D tereni ili stvoriti 2D mape pločica izravno u motoru, tako da postoji dobro zaokružen skup alata za korištenje za bilo koju grafiku.

- **Lako razumljiva arhitektura**

Unity nudi vrlo transparentnu metodu za sastavljanje arhitekture igre. Svaka "razina" u projektu igre Unity podijeljena je na scenu, a svaka scena sadrži sve objekte igre potrebne igraču za prelaženje razine - bilo da je to pozadina, lik igrača, neprijatelj, metak ili nešto drugo.

Unity također nudi mogućnost uspostavljanja odnosa roditelj-dijete između objekata u hijerarhiji, što olakšava dodavanje više objekata (poput odjeće, pištolja ili sudarača za otkrivanje sudara) jednom roditeljskom objektu lika igrača. Osim toga, Unity također ima alat Inspector koji daje brzi pristup svim svojstvima objekta, što znači da se stvari mogu brzo mijenjati, u hodu.

- **Unity Scripting API**

Unity dolazi sa snažnim API-jem za skriptiranje koji nudi brzi pristup najčešće potrebnim značajkama. To uključuje i opće značajke igre, kao i specifične API pozive koji omogućuju pristup određenim značajkama i nijansama.

- **Podrška za međuplatformsku izgradnju**

Unity igre podržavaju izgradnju na ogromnom broju platformi. Sve dok razvojni programer preuzme odgovarajući komplet, mogu se izvesti igre za Android, iOS, Windows, MacOS, Linux, PS4, Xbox One i druge. Mogu se čak izvesti HTML5 igre ako programer želi staviti svoju igru na web (pod pretpostavkom da je izvedba optimalna).

- **Virtualna stvarnost i mogućnosti proširene stvarnosti**

Kada su u pitanju VR i AR, novije tehnologije, Unity je jedan od ključnih pobornika razvoja s njima. Za VR su dostupni brojni paketi koji podržavaju gotovo sve dostupne VR slušalice, te se stalno ažuriraju i održavaju fleksibilnost s ovom promjenjivom tehnologijom.

AR ne smije biti izostavljen, s brojnim paketima za ARCore i ARKit. Unity također nudi AR Foundation, koju je Unity izradio kako bi omogućio programerima Unityja stvaranje AR aplikacija za Android i iOS u isto vrijeme, eliminirajući potrebu za posebnim projektima.

Unity također ima XR Interaction Toolkit kako bi razvoj VR i AR igara bio još lakši. Dakle Unity je jedan od najvećih pobornika XR tehnologija.

- **Prodavaonica velike imovine**

Unity nudi široku paletu grafičkih materijala, specifičnih predloška žanra igre, efekata čestica ili drugo. Njegovo neizmjerno veliko skladište sredstava dolazi s raznim plaćenim i besplatnim sredstvima koji se mogu koristiti za bilo koji projekt igre.

Iako Unity razvija neke od njih, mnoge od njih također je izradila zajednica. Osim toga, Unity olakšava dodavanje sredstava u kolekciju i njihovo instaliranje u projekt pomoću upravitelja paketa, što znači da nema petljanja s datotekama ručno.

- **Unity razvijeni paketi**

Unity nudi tonu vlastitih paketa i sredstava besplatno koji proširuju funkcionalnost na korisne načine. Na primjer, sredstvo Bolt nudi način implementacije vizualnog skriptiranja u Unity motor.

U međuvremenu, Unity Playground nudi okvir za 2D igre koji omogućuje učenje razvoja igara bez potrebe za kodiranjem od nule. Unity besplatno nudi sve, od besplatnih modela do raznih kompleta igara, dajući brzi pristup sredstvima odobrenim od strane Unityja za vježbanje.

- **Opcije cjevovoda renderiranja**

Renderiranje grafike na zaslonu nije lak podvig za računalo, a način na koji to postiže može značajno utjecati na izvedbu igara. Zbog toga je Unity ponudio nekoliko ugrađenih opcija za cjevovode renderiranja koji se mogu koristiti da se igra prebaci sa scene na zaslon za igranje. To programerima omogućuje da odaberu cjevovod renderiranja koji najbolje odgovara njihovim projektima i grafičkim potrebama tih projekata.

Dodatno, Unity također nudi Scriptable Render Pipeline API, koji programerima omogućuje izradu vlastitog cjevovoda ako to žele. Dakle, postoji puno slobode u pogledu načina na koji se igra prikazuje.

- **Alati za animaciju**

Unity nudi robustan skup alata za animaciju koji rade i za 3D i za 2D grafiku. Iako se animacija može uvesti iz drugog programa, kao što je Blender, Unity nudi mogućnost da se projekti animiraju izravno unutar samog engine-a. To uključuje podešavanje

položaja i rotacije cijelog objekta, do stvarne fizičke manipulacije kostima u 3D modelu. Unity čak nudi mogućnost dodavanja koštane opreme 2D slikama.

Uz to, njegov sustav Animator omogućuje jednostavno stvaranje stroja stanja animacije.

- **Prilagodljivost drugim industrijama**

Dok je Unity prije svega motor za igre, dodao je značajke kako bi motor bio koristan za druge industrije. Na primjer, zbog svojih opcija cjevovoda i alata za animaciju, Unity se zapravo može koristiti za CG filmove visoke kvalitete što su mnogi nezavisni filmaši iskoristili. S druge strane, Unity je također stvorio stvari kao što je Unity Reflect kako bi programerima zgrada pružio način da vizualiziraju svoje projekte i povežu ih s drugim CAD softverom.

- **Alati za analitiku**

Unity nudi nekoliko alata za praćenje problema s performansama i alate za jednostavno promatranje kako igrači stupaju u interakciju s igrom.

Osim toga, Unity također nudi brojne načine za poboljšanje otklanjanja pogrešaka s ovim alatima, pružajući robustan način za razumijevanje svakog aspekta igre.

U Unity-u postoje ugrađene metode i to iz klase MonoBehaviour koje kada se kreira nova skripta, onda automatski ta skripta se nasljeđuje iz monobeaviour-a. Primjerice postoji Awake metoda koja se poziva iako je objekt onemogućen, Metoda start se poziva odmah na početku, Metodom Update se poziva svaki frame igre, postoji još i FixedUpdate koji se poziva na svaki frame. OnCollisionEnter2D je metoda koja se poziva kad objekt u sceni imati kontakt s drugim objektom, OnTriggerEnter2D je slična metoda samo što treba na komponenti RigidBody2D označit IsTrigger.

Unity sadrži komponente koje se koriste za razvoj igre, svaki objekt je GameObject, svaka komponenta se sastoji od Transform komponente, za koju možemo reći da je to komponenta koja nam opisuje poziciju rotacije i veličinu objekta. BoxCollider2D je komponenta koja omogućuje da objekt ima fizički kontakt. RigidBody2D je komponenta koja omogućava fiziku na objektu. Sprite su komponente koje su zapravo slike, a AudioSource je komponenta koja omogućava da se reproducira zvuk u sceni.

GIMP je besplatan alat za izradu svojih vlastitih slika. GIMP je vrlo jednostavan alat koji pruža brojne funkcije koje imaju interakcije sa slikom i koji ima mogućnost stvaranja slojeva, a to je jedina funkcija koja nam prikazuje trenutne slojeve koje se nalaze na slici. GIMP je akronim za GNU Image Manipulation Program. To je besplatni program za zadatke kao što su retuširanje fotografija, kompozicija slike i autorska slika (About GIMP, 2023).

GIMP ima mnogo mogućnosti. Može se koristiti kao jednostavan program za slikanje, program za retuširanje fotografija stručne kvalitete, sustav za mrežnu serijsku obradu, renderer slika za masovnu proizvodnju, pretvarač formata slike itd.

GIMP je proširiv. Osmišljen je tako da se može nadopuniti dodacima i proširenjima za gotovo sve. Napredno skriptno sučelje omogućuje jednostavno skriptiranje svega, od najjednostavnijih zadataka do najsloženijih postupaka manipulacije slikama.

GIMP je napisan i razvijen pod X11 na UNIX platformama, ali u osnovi isti kod radi i na Windowsima i macOS-u.

Audacity je alat koji je vrlo koristan za izradu svojih vlastitih zvukova, a može se čak koristit za modifikaciju zvukova, dodavanje efekata i dodavanje zvukova na postojećim zvukovima. Audacity je softver koji korisnicima omogućuje besplatno snimanje i uređivanje audio zapisa. Sučelje Audacityja je jednostavno i vrlo lako za korištenje. Također nudi kompatibilnost s više platformi i podržava višestruke dodatke i biblioteke za poboljšanu funkcionalnost. Može glatko raditi na operativnim sustavima Windows, Apple macOS i Linux. Sve te značajke učinile su Audacity jednim od najpopularnijih softvera za uređivanje zvuka dostupnih danas (What is Audacity, 2023).

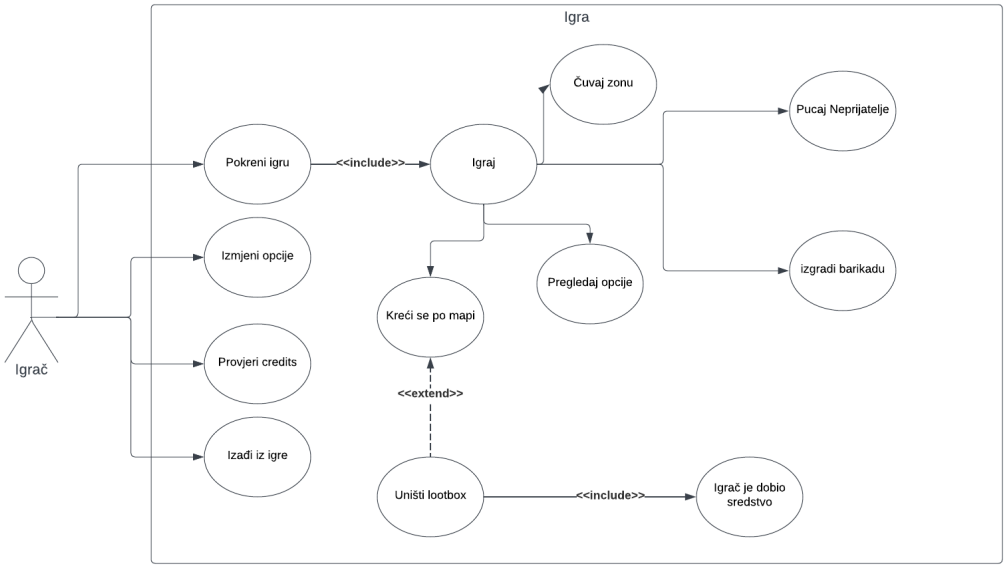
Za praćenje promjena koda i upravljanje verzijama projekta, korišten je GitHub (Github,2023). To je popularna platforma za razvoj softvera koja omogućuje programerima da surađuju, dijele i pregledavaju izmjene u kodu. Sve promjene ovog projekta "SurviveIT" nalaze se pohranjene na autorovom GitHub repozitoriju.

Poveznica do izvornog koda je:

https://github.com/Cabraja8/SurviveIT

2. PLANIRANJE PROJEKTA I PROCES IZRADE PROTOTIPA

Dijagram slučaja uporabe modelira ponašanje sustava i pomažu u hvatanju zahtjeva sustava. Opisuju funkcije visoke razine i opseg sustava. Ovi dijagrami također identificiraju interakcije između sustava i njegovih aktera. Akteri u dijagramu slučaju uporabe opisuju što sustav radi i kako ga akteri koriste, ali ne i kako sustav interno funkcionira. Oni ilustriraju i definiraju kontekst i zahtjeve bilo cijelog sustava ili važnih dijelova sustava. Moguće je modelirati složeni sustav s jednim dijagramom slučaje uporabe ili stvoriti mnogo dijagrama za modeliranje komponenti sustava. Obično se use case dijagram razvija u ranim fazama projekta i poziva se na njih tijekom procesa razvoja. Na početku igre, igrač ima više opcija koje predstavljaju glavni izbornik. Mogućnosti su „Izađi iz igre“ odnosno da se igra prekine. „Credits“ je izbornik koji prikazuje tko je autor igre i igrač može vidjeti opcije dok je u glavnom izborniku. Mogućnost „Pokreni igru“ označava da igrač pokreće igru i kreće s igranjem. Igra ima više mogućnosti, a to su Čuvaj zonu koja sadrži funkcionalnosti „Pucaj Neprijatelje“ ili „Izgradi barikadu“. Također usred igranja, igrač može otvori izbornik i pregledati opcije, igrač se može kretati po mapi i ako želi može pronaći kutije odnosno lootboxove i dobit potrebna sredstva. Kao što je i prikazano na slici 1, dijagram slučaja uporabe akcijske igre „Survive it“.



Slika 1. Dijagram slučaja uporabe akcijske igre „Survive it"

Ako se projekt želi uspješno dovršiti, i planiranje i izvedba moraju biti pravilno provedeni. Loše planiranje neće omogućiti odgovarajuće procese izvršenja i kontrole ili postizanje ciljeva projekta. Planiranje pomaže odlučiti kako najbolje iskoristiti resurse (ljude, vrijeme, novac, informacije, opremu) tako da daju najznačajniji doprinos postizanju cilja. Planiranje predstavlja temelj za učinkovitu procjenu i evaluaciju postignuća. Vrlo je bitno prije samog početka projekta, zadati si ciljeve i očekivanja, te si jasno naznačiti kako želimo da tijek projekta izgleda.

Prije početka projekta, moramo odrediti njegovo ime. U ovom slučaju ime projekta je razvoj akcijske igre „Survive it“ u Unity okruženju. Nakon određivanja imena, potrebno je proučiti koje su tehnologije dostupne i u konačnici odrediti koje će se tehnologije koristiti, te na koji način.

Glavna zamisao ove igre je preživljavanje naleta neprijatelja. Igrač je na misiji, na kojoj ga je poslao glavni kapetan, s ciljem obrane određenog mjesta na karti. Ako obrani to određeno mjesto, igrač je spašen. Tijekom igre, igrač ima mogućnost korištenja oružja, kojeg može naći u raznim kutijama, koje se nasumično pojavljuju na mapi, a igrač pomoću miša može koristiti oružje i pucati na neprijatelja. Postoje razna moguća sredstva koje igrač može koristiti poput oružja, zdravlja i dasaka. Daske se koriste za izradu barikada koje služe za obranu od neprijatelja. Neprijatelj se također stvara nasumično na 4 mjesta, a postoje tri vrste neprijatelja, od kojih svatko ima različite atribute.

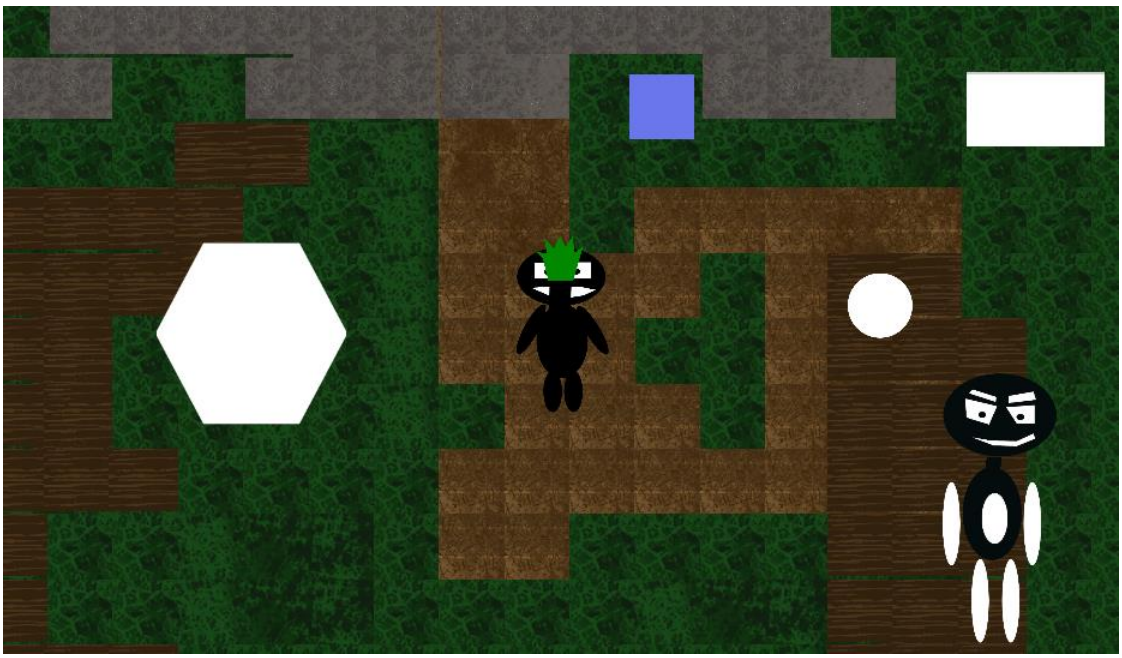
3. PROCES RAZVOJA AKCIJSKE IGRE „SURVIVE IT“ U UNITY OKRUŽENJU

3.1. PROTOTIP

Prije same izrade igre, bilo je potrebno izraditi nekakav prototip sa svim važnim funkcionalnostima igre. Prototip se sastoji od jednostavne animacije i prve skice igrača, neprijatelja i sredstava koje se nalaze u igri. Također bitno je bilo izraditi skripte u C# programskom jeziku. Nakon što je prototip bio izrađen, kreirana je nova scena u Unityu, i samo su dodani potrebni game object-i te su ubačene već gotove skripte. Izrada prototipa je omogućila testiranje i usavršavanje mehanike igre, osiguravajući da je sama igrica funkcionalna i zabavna za igrača.

Izrada prototipa je sastavni dio nacrta dizajna koji prethodi razvoju izvornog izgleda dizajna. Namijenjen je ne samo prikazu strukture budućeg web-mjesta, već i karti web-mjesta, međusobnom odnosu njegovih glavnih stranica. Izrada prototipa je proces osmišljen kako bi se značajno smanjilo vrijeme potrebno za razvoj samog proizvoda. (The importance of prototyping in design, 2019).

Na slici 2 je prikazan izgleda prototipa igrice, tj. prikazana je zamisao kako bi igrica trebala izgledati, njezini osnovni likovi i izgled mape.



Slika 2. Izgled prototipa

3.2. IGRAČ

Dizajn lika je kreiran u GIMP-u. Igrač ima komponente za fiziku, koristi se BoxColider2D koji služi za interakciju s ostalim objektima u sceni i Rigidbody2D koji se koristi da se primjene svojstva fizike na objektu i uz to ima i komponentu za zvuk Audio Source koji služi za puštanje zvuka te komponentu Animator koji služi za pokretanje animacija kod Igrača. Na slici 3 je prikazana ikona igrača.

Slika 3. Igrač

Programski kod 1 prikazuje varijable za igrača. Speed označava brzinu kretanje igrača, rb je varijabla koja se inicijalizira na RigidBody2D komponentu preko metode start, a tako vrijedi isto i za varijablu anim za komponentu Animator. Health je varijabla koja označava zdravlje lika, MoveAmount je varijabla koja zapravo služi da se zna koliko se lik može kretati u 2D prostoru. Audiosource služi da bi se dohvatio izvor zvuka, a pickupsound je varijabla koja ima zvuk koji se reproducira svaki put kada igrač uzme nešto. Healthbar je varijabla koja sadrži komponentu healthbar koja je zapravo zdravlje igrača, ali prikaz u sučelju i zadnja varijabla je planks koja označava koliko igrač ima dasaka za izgraditi barikade. Metoda start inicijalizira varijablu anim i rb tj. zapravo dohvaća komponente koje su na objektu. Metoda IncreasePlanks služi igraču da se opremi s daskama. Metoda update prati unos tipki te da li se igrač kreće ili ne.

```
public class Player : MonoBehaviour
{
    public float speed;
    public Rigidbody2D rb;
    private Animator anim;
    public int health;
    private Vector2 MoveAmount;
    public AudioSource audiosource;
    public AudioClip pickupsound;
    public HealthBar healthBar;
    public float planks=3f;

   private void Awake() {
    Time.timeScale =1f;
   }
     void Start() {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }
    public void IncreasePlanks(float plankAmount){
        planks = planks + plankAmount;
        FindObjectOfType<BuildBarricade>().UpdateDisplay();

        if(planks >3f){
            planks =3f;
        }
    }
     void Update() {
        Vector2 MoveInput = new Vector2(Input.GetAxisRaw("Horizontal"),Input.GetAxisRaw("Vertical"));
        MoveAmount = MoveInput.normalized * speed;

        if (MoveInput != Vector2.zero){
            anim.SetBool("IsRunning",true);
        }
        else{
            anim.SetBool("IsRunning",false);
        }
    }
```

Programski kod 1. Klasa Player

Programski kod 2 prikazuje kretanje, dohvat svojstva, ranjavanje igrača te vraćanje zdravlja. Metoda FixedUpdate je funkcija koja zapravo pokreće igrača. Metoda TakeDamage služi da bi se zdravlje igrača smanjilo od strane neprijatelja. Metoda PickupItem je funkcija koja reproducira zvuk kad igrač uzme nešto. Metoda RecoverHealth je funkcija koja služi da bi se vratilo zdravlje igrača, a to se poziva kad igrač uzme Medkit.

```
    private void FixedUpdate() {
        rb.MovePosition(rb.position + MoveAmount * Time.fixedDeltaTime);
    }

    public void TakeDamage(int damageAmount){

        health = health - damageAmount;

        healthBar.SetHealth(health);

        if(health <=0){
            healthBar.SetHealth(health);
            Destroy(this.gameObject);
            FindObjectOfType<Zone>().GameOver();
        }
    }


    public void PickupItem(){
        audiosource.PlayOneShot(pickupsound);
    }


    public void RecoverHealth(int healthAmount){
        health = health + healthAmount;
        if(health > 500){
            health = 500;
        }
        healthBar.SetHealth(health);

    }
}
```

Programski kod 2. Kretanje, Dohvat svojstva, Ranjavanje igrača, Vraćanje zdravlja

Programski kod 3 prikazuje Slider koji označava varijablu koja sadrži komponentu slider, Metoda SetHealth je funkcija koja služi za Healthbar da bi se prilagodio.

```
public class HealthBar : MonoBehaviour
{

    public Slider slider;

    public void SetHealth(int health){

        slider.value = health;
    }
```

Programski kod 3. Klasa HealthBar

Programski kod 4 prikazuje varijablu displayPlanks koja sadrži tekst za sučelje, Awake metoda se odmah prije prve slike igre pozove, UpdateDisplay ažurira tekst dasaka koji igrač može vidjeti.

```
public class BuildBarricade : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI displayPlanks;
     void Awake() {
         UpdateDisplay();
     }
      public void UpdateDisplay(){
         displayPlanks.text = FindObjectOfType<Player>().planks.ToString();
     }
}
```

Programski kod 4. Klasa BuildBarricade

Programski kod 5 prikazuje upute za igrača, informacije o oružju i prikaze pokreta te sadrže canvas-e. Metoda start se pokreće i zove ShowWeaponControl koja prikazuje kontrole oružja, te poziva Funkcija ShowMovmentAndWeapon koja prikazuje druge upute i onda na kraju se poziva Metoda RemoveAll koja miče sve s ekrana.

```
public class ShowTutorial : MonoBehaviour
{
    public GameObject weaponinfo;
    public GameObject showmovement;

    void Start()
    {
        ShowWeaponControl();
    }
     public void ShowWeaponControl(){
        weaponinfo.SetActive(true);
        StartCoroutine(ShowMovementAndWeapon());
    }
    public IEnumerator ShowMovementAndWeapon(){
        yield return new WaitForSeconds(2.5f);
        weaponinfo.SetActive(false);
        showmovement.SetActive(true);
        StartCoroutine(RemoveAll());
    }

    public IEnumerator RemoveAll(){
        yield return new WaitForSeconds(2.5f);
        weaponinfo.SetActive(false);
        showmovement.SetActive(false);
    }
}
```

Programski kod 5. Klasa ShowTutorial

Programski kodovi 6 i 7 imaju varijable koje sadrže sredstvo o municiji, te sadrže i varijable za prikaz na sučelju. Metoda UpdateDisplay ažurira svu municiju, a za metode GetCurrentBullets, GetCurrentShells i GetCurrentHandgunBullets dohvaćaju se trenutne municije. Metode IncreaseShells, IncreaseBullets i Increase HandGunBullets dodaju municiju za svaku kategoriju, a metode DecreaseShells, DecreaseBullets i DecreaseHandGunBullets smanjuju municiju.

```
public class WeaponAmmoDisplay : MonoBehaviour
{
    public int bullets;
    public int shells;
    public int HandgunBullets;
    [SerializeField] TextMeshProUGUI displayBullets;
    [SerializeField] TextMeshProUGUI displayShotgunShells;
    [SerializeField] TextMeshProUGUI displayHandGunBullets;
     public void UpdateDisplay(){
        displayBullets.text = bullets.ToString();
        displayShotgunShells.text = shells.ToString();
        displayHandGunBullets.text = HandgunBullets.ToString();
    }
    public int GetCurrentBullets(){
        return bullets;
    }
    public int GetCurrentShells(){
        return shells;
    }
    public int GetCurrentHandgunBullets(){
        return HandgunBullets;
    }

    public void IncreaseShells(int shellAmount){
        shells=shells + shellAmount;
        UpdateDisplay();
    }
    public void DecreaseShells(int shellAmount){

        shells= shellAmount;
        if(shells<1){
            shells=0;
        }
        UpdateDisplay();
    }
    public void IncreaseHandgunBullets(int BulletAmount){
        HandgunBullets = HandgunBullets + BulletAmount;
        UpdateDisplay();
     }
```

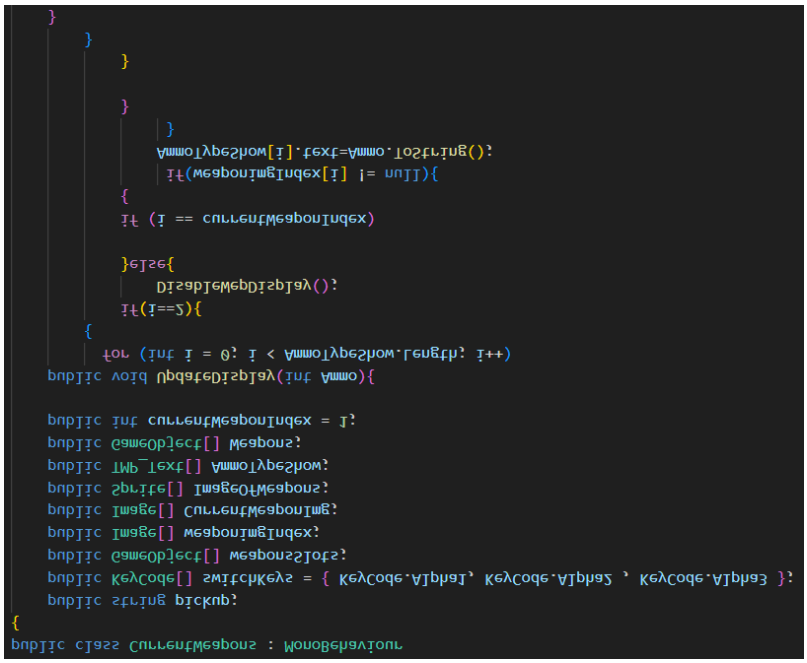
Programski kod 6. Klasa WeaponAmmoDisplay

```
    public void DecreaseHandgunBullets(int BulletAmount){
        HandgunBullets = BulletAmount;
        if(BulletAmount<1){
            BulletAmount=0;
        }
        UpdateDisplay();
    }


    public void IncreaseBullets(int BulletAmount){
        bullets=bullets + BulletAmount;
        UpdateDisplay();
    }
    public void DecreaseBullets(int BulletAmount){
        bullets= BulletAmount;
        if(bullets<1){
            bullets=0;
        }
    UpdateDisplay();
    }
}
```

Programski kod 7. Prikaz municije

Programski kod 8 ima pickup varijablu koja sadrži naziv oružja, switchKeys sadrži tipke za koje igrač koristi od 1-3, weaponSlots varijabla je polje koja sadrži trenutna oružja koje igrač ima u posjedu, weaponimgIndex je varijabla koja sadrži indeks od slike od oružja, CurrentWeaponImg je trenutni indeks od oružja, ImageOfWeapons varijabla sadrži slike od svih mogućih oružja, AmmoTypeShow je varijabla koja prikazuje igraču trenutni Ammo koji ima na oružju, Weapons varijabla je polje koje ima sva moguća oružja u igri i current weaponIndex sadrži trenutni indeks oružja.

Metoda UpdateDisplay ažurira zaslon i prikazuje za oružje koje je trenutno stanje municije.



Programski kod 8. Sredstva za trenutna oružja i ažuriranje sučelja

Programski kod 9 sadrži metodu start koja inicijalizira indeks 1 za oružje. Metoda Update nam omogućuje promjenu oružja kada se pritisnu tipke 1,2 ili 3, ako je oružje MeleeWeapon kao što su Knife i Crowbar onda ne prikazuje municiju jer za MeleeWeapon nema municije.

```
private void Start()
{
    for (int i = 2; i < weaponsSlots.Length; i++)
    {    if(weaponsSlots[i] != null){
        weaponsSlots[i].SetActive(false);
    }
    }
    SetCurrentIndex(1);
    SetCurrentText(1);

}
 private void Update()
{
    for (int i = 0; i < switchKeys.Length; i++)
    {
        if(weaponsSlots[i] != null){
        if (Input.GetKeyDown(switchKeys[i]))
        {
            currentWeaponIndex = i;
            SetCurrentWeapon(currentWeaponIndex);
            SetCurrentIndex(currentWeaponIndex);
        }
    }
    }
    if(  FindObjectOfType<MeleeWeapon>()){
       DisableWepDisplay();
    }
    else{
        SetCurrentText(currentWeaponIndex);
        SetCurrentIndex(currentWeaponIndex);
    }
```

Programski kod 9. Promjena oružja

Programski kod 10 sadrži SetCurrentIndex koji namješta indeks oružja, a SetCurrentText prikazuje trenutni Ammo.

```
public void SetCurrentIndex(int weaponIndex){
    for (int i = 0; i < weaponimgIndex.Length; i++)
    {
        if (i == weaponIndex)
        {
            if(weaponimgIndex[i] != null){
            weaponimgIndex[i].enabled=true;
            }
        }
        else
        {
            if(weaponimgIndex[i] != null){
            weaponimgIndex[i].enabled=false;
            }
        }
    }

}
public void SetCurrentText(int weaponIndex){
    for (int i = 0; i < AmmoTypeShow.Length; i++)
    {
        if (i == weaponIndex)
        {
            if(AmmoTypeShow[i] != null){
            AmmoTypeShow[i].enabled=true;
            }
        }
        else
        {
            if(AmmoTypeShow[i] != null){
            AmmoTypeShow[i].enabled=false;
            }
        }
    }
}
```

Programski kod 10. Indeksiranje oružja

Programski kod 11 sadrži DisableWebDisplay metodu koja makne trenutni ammo od oružja, ali to jedino funkcionira ako je trenutno oružje MeleeWeapon kao što su crowbar i knife. SetCurrentWeapon je metoda koja namješta trenutno oružje. To funkcionira na način da omogući oružje od igrača, ima 3 utora gdje spremi svoje oružje od 1-3.

```
public void DisableWepDisplay(){
        for (int i = 0; i < AmmoTypeShow.Length; i++)
    {
            if(AmmoTypeShow[i] != null){
            AmmoTypeShow[i].enabled=false;

        }
    }
}
public void SetCurrentWeapon(int weaponIndex)
{
    for (int i = 0; i < weaponsSlots.Length; i++)
    {
        if (i == weaponIndex)
        {
            if(weaponsSlots[i] != null){
            weaponsSlots[i].SetActive(true);
            }
        }
        else
        {
            if(weaponsSlots[i] != null){
            weaponsSlots[i].SetActive(false);
            }
        }
    }
}
```

Programski kod 11. Namještanje oružja i micanje teksta ako je oružje MeleeWeapon

Programski kod 12 sadrži metodu ChangeWeapon koja mijenja oružje ovisno o kojem oružju se radi, znači kad igrač pronađe oružje i kad se opremi onda mu se promjeni trenutni inventory tj. WeaponSlots, za svaki pickup vrijedi isto. Svaki put kad igrač pokupi oružje onda skripta provjeri o kojem se oružju radi i onda ga namjesti igraču u ruci.

```
public void ChangeWeapon(GameObject weapon){
    int weaponIndex = -1;
    for (int i = 0; i < weaponsSlots.Length; i++)
    {
        if(weaponsSlots[i] != null){

            weaponsSlots[i].SetActive(false);

     if(pickup=="ak47"){
        if(currentWeaponIndex==0){
            weaponsSlots[0].SetActive(false);
            CurrentWeaponImg[0].sprite = ImageOfWeapons[1];
            weaponsSlots[0] = null;
            currentWeaponIndex = 0;
        }
        SetCurrentIndex(0);
        CurrentWeaponImg[0].sprite = ImageOfWeapons[1];
        currentWeaponIndex = 0;
        weaponsSlots[0] = null;
        weaponsSlots[0] =weapon;
        weaponsSlots[0].SetActive(true);
        weaponIndex = i;
        weaponsSlots[0] = Weapons[1];
        SetCurrentWeapon(0);
         }
     else if(pickup=="glock-18"){
        if(currentWeaponIndex==1){
            weaponsSlots[1].SetActive(false);
            CurrentWeaponImg[1].sprite = ImageOfWeapons[0];
            weaponsSlots[1] = null;
            currentWeaponIndex = 1;
            SetCurrentIndex(1);
        }
        SetCurrentIndex(1);
        CurrentWeaponImg[1].sprite = ImageOfWeapons[0];
        currentWeaponIndex=1;
        weaponsSlots[1] = null;
        weaponsSlots[1] =weapon;
        weaponsSlots[1].SetActive(true);
        weaponIndex = i;
         weaponsSlots[1] = Weapons[0];
        SetCurrentWeapon(1);
         }
```

Programski kod 12. Promjena oružja

3.3. KAMERA

Kamera je komponenta u Unity-u koja prikazuje igru te je za kameru izrađena skripta CameraFollow. Slika 4 prikazuje izgled kamere koja se prikazuje na ekranu. Vidljivo je na slici broj 4 da se na ekranu prikazuje dostupna municija te dostupno oružje, također i numerirani su naleti neprijatelja npr. wave 1. Također dostupan je izvještaj o zdravlju te dostupne su barikade za borbu protiv neprijatelja.

Slika 4. Kamera

Programski kod 13 sadrži varijable playertransform, a ta varijabla sadrži vrijednosti pozicije igrača, speed je brzina kamere, za minX, maxX, minY, maxY to su vrijednosti koliko daleko može ići kamera po X i Y. Metoda Start zapravo samo dohvaća poziciju igrača. Kod metode Update ova funkcija zapravo računa poziciju za kameru.

```
public class CameraFollow : MonoBehaviour
{
    public Transform playerTransform;
    public float speed;

    public float minX;
    public float maxX;
    public float minY;
    public float maxY;


    // Start is called before the first frame update
    void Start()
    {
        transform.position = playerTransform.position;
    }

    // Update is called once per frame
    void Update()
    {
        if(playerTransform !=null) {
        float clampedX = Mathf.Clamp(playerTransform.position.x, minX,maxX);

        float clampedY = Mathf.Clamp(playerTransform.position.y, minY,maxY);


        transform.position = Vector2.Lerp(transform.position, new Vector2(clampedX,clampedY), speed);
        }
    }
}
```

Programski kod 13. Klasa CameraFollow

3.4. ORUŽJA

3.4.1. PUŠKE

Dizajn oružja je kreiran preko alata Logomakr. Korištene su njihove gotove ikone od oružja. Za dodavanje funkcionalnosti svakom oružju, koristila se ista skripta koja se naziva "Weapon skripta". Unutar te skripte su se modificirale specifikacije za svako oružje.

Programski kod 14 prikazuje "Weapon skriptu" koja se sastoji od nekoliko ključnih elemenata:

Sredstva koje opisuju oružje su projectile, koje sadrži projektil za oružje, weapon sadrži oružje, nadalje postoji i shotPoint,shotPoint2 i shotPointHelper koji su za flash puške, rend sadrži Sprite o pušci i uz to postoji Fire1 i Fire2 koje sadrže flash, Ammo varijabla prikazuje trenutno stanje municije puške, bullets prikazuje municiju od Assault RIfle Bullets, shells prikazuje za ShotgunShells bullets i postoji još za handgunbullets, Max ammo je granica za svaku pušku koliko može imati u spremniku za oružje, timeBetweenShots je varijabla koja opisuje vrijeme između pucnjave, shotTime je vrijeme koliko treba čekati igrač da puca, audiosource sadrži izvor zvuka, ReloadSound sadrži zvuk punjenja oružja, ShotSound je zvuk pucnjave, reloadSign je prikaz na sučelju da igrač mora puniti oružje jer je prazno, a ammofull je kad igrač pokušava napuniti oružje, a spremnik za oružje pun i Reloading varijabla samo služi da bi mogli vidjeti da li se Oružje puni ili ne.

```
public class Weapon : MonoBehaviour
{
    public GameObject projectile;
    public Weapon weapon;
    public Transform shotPoint;
    public Transform shotPoint2;
    public Transform shotPointHelper;
    public SpriteRenderer rend;
    public SpriteRenderer Fire1;
    public SpriteRenderer Fire2;
    public int Ammo;
    public int bullets;
    public int shells;
    public int HandgunBullets;
    public int MaxAmmo;
    public float timeBetweenShots;
    private float shotTime;
    public AudioSource audiosource;
    public AudioClip ReloadSound;
    public AudioClip ShotSound;
    public TMP_Text reloadSign;
    public TMP_Text ammofull;

    public bool Reloading= false;

    private void Awake() {
    Time.timeScale =1f;
   }
```

Programski kod 14. Klasa Weapon

Na programskom kodu 15 koji sadrži metodu Update, vidljivo je da kad igrač pritisne R da se napuni spremnik za oružje. Update metoda provjerava ako je spremnik za oružje pun, dohvaća trenutno stanje puške tj. koliko municije ima trenutno oružje, uz to provjerava i smjer oružje dakle ako igrač pomakne mišem u neku stranu onda oružje se rotira po mišu, ako se pritisne lijevom tipkom miša onda igrač puca sa oružjem i onda igrač vidi i animacije od puške.

```
void Update()
{
    if(Input.GetKeyDown(KeyCode.R) && Ammo == MaxAmmo){
        Debug.Log("ammo full");
        ammofull.enabled = true;
        StartCoroutine(RemoveAmmo());
    }
    GetAmmo();
    GetCurrentWeapon();
    if(!PauseMenu.GameIsPaused){
        WeaponControl();
    }
     UpdateAmmo();
if(Ammo >0){
    if (Input.GetMouseButtonDown(0) && !PauseMenu.GameIsPaused){
        Fire1.enabled = false;
        Fire2.enabled = false;
    }

    if(Input.GetMouseButton(0) && !PauseMenu.GameIsPaused){
        if(Time.time >= shotTime){
            if(!Reloading){
            Shoot();
            }
        }
    }
    else{
        Fire1.enabled = false;
        Fire2.enabled = false;
    }
}
if(Ammo == 0){
    Fire1.enabled = false;
    Fire2.enabled = false;
    reloadSign.enabled=true;
}
if(Reloading){
    reloadSign.enabled=false;
```

Programski kod 15. Provjera za pocanje i napunit oružje

Programski kod 16 provjerava koje vrste oružje ima igrač i onda napuni spremnik za to oružje, prvo kad igrač pritisne tipku R skripta provjerava dali se radi od napadnoj pušci, sačmarici ili pištolju i ovisno o tipu puške onda će se pozvati pojedina funkcija i uz to će reproducirati zvuk od puške. Reloading je bool varijabla koja se namješta na true ako se pozvalo tj. ako je igrač pritisnuo tipku R.

```
if(Input.GetKeyDown(KeyCode.R) && !PauseMenu.GameIsPaused){
    if(Ammo == MaxAmmo){
        Debug.Log("ammo full");
    }
    if(gameObject.tag == "Assault Rifle"){
    if(Ammo != MaxAmmo && bullets !=0){
    Reloading = true;
    audiosource.PlayOneShot(ReloadSound);
    Invoke("ReloadGun",1.5f);
    }
    }
    else if(gameObject.tag == "Shotgun") {
    if(Ammo != MaxAmmo && shells !=0){
    Reloading = true;
    audiosource.PlayOneShot(ReloadSound);

    Invoke("ReloadShotgun",2.0f);
        }
    }else if(gameObject.tag =="HandGun"){
    if(Ammo != MaxAmmo && HandgunBullets !=0){
    Reloading = true;
    audiosource.PlayOneShot(ReloadSound);

    Invoke("ReloadHandgun",1.0f);
        }
    }
}
```

Programski kod 16. Provjera za napunit oružje ovisno o tipu oružja

Programski kod 17 prikazuje metodu RemoveAmmo koja makne tekst s ekrana. Weapon Control metoda služi za kretanje oružja od Igrača pomoću miša, GetCurrentWeapon metoda dohvaća trenutno oružje i GetAmmo Metoda dohvaća sve moguće municije.

```
public IEnumerator RemoveAmmo(){
    yield return new WaitForSeconds(1.0f);
    ammofull.enabled = false;
}
void WeaponControl(){
    Vector2 direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;

    float angle= Mathf.Atan2(direction.y,direction.x)* Mathf.Rad2Deg;

    Quaternion rotation = Quaternion.AngleAxis(angle -90f, Vector3.forward);

    transform.rotation = rotation;

    if(angle >=90 || angle<= -90){
        rend.flipX = true;
        shotPoint = shotPoint2;
    }
    else{
        rend.flipX = false;
        shotPoint=shotPointHelper;
    }
}

public Weapon GetCurrentWeapon(){

   return FindObjectOfType<Weapon>();
}

void GetAmmo(){
    bullets = FindObjectOfType<WeaponAmmoDisplay>().GetCurrentBullets();
    shells = FindObjectOfType<WeaponAmmoDisplay>().GetCurrentShells();
    HandgunBullets = FindObjectOfType<WeaponAmmoDisplay>().GetCurrentHandgunBullets();
}
```

Programski kod 17. Upravljanje oružjem, dohvat municije i micanje teksta s ekrana

Programski kod 18 Shoot metoda služi za pucanje. Nadalje postoje još i 3 metode ReloadShotgun, Reloadgun i ReloadHandgun koje su slične, služe za napuniti spremnik oružja ovisno o tipu oružja.

```
void ReloadShotgun(){
    Reloading = false;
   int reloadAmount=MaxAmmo - Ammo;
    reloadAmount = (shells - reloadAmount)>= 0 ? reloadAmount : shells;
    Ammo += reloadAmount;
    shells -=reloadAmount;
     FindObjectOfType<WeaponAmmoDisplay>().DecreaseShells(shells);
     FindObjectOfType<WeaponAmmoDisplay>().UpdateDisplay();
}
void ReloadGun(){
    Reloading = false;
    int reloadAmount=MaxAmmo - Ammo;
    reloadAmount = (bullets - reloadAmount)>= 0 ? reloadAmount : bullets;
    Ammo += reloadAmount;
    bullets -=reloadAmount;
     FindObjectOfType<WeaponAmmoDisplay>().DecreaseBullets(bullets);
     FindObjectOfType<WeaponAmmoDisplay>().UpdateDisplay();
}

void ReloadHandgun(){
    Reloading = false;
    int reloadAmount=MaxAmmo - Ammo;
    reloadAmount = (HandgunBullets - reloadAmount)>= 0 ? reloadAmount : HandgunBullets;
    Ammo += reloadAmount;
    HandgunBullets -=reloadAmount;
     FindObjectOfType<WeaponAmmoDisplay>().DecreaseHandgunBullets(HandgunBullets);
     FindObjectOfType<WeaponAmmoDisplay>().UpdateDisplay();
}

void Shoot(){
    Ammo--;
    audiosource.PlayOneShot(ShotSound);
    Instantiate(projectile ,shotPoint.position, transform.rotation );
    if(!rend.flipX){
    Fire1.enabled = true;
    }else if(rend.flipX){
    Fire2.enabled = true;
    }
    shotTime = Time.time + timeBetweenShots;
}
```

Programski kod 18. Metode za napunit spremnik oružja i metoda za pucanje

Programski kod 19 prikazuje Projektil koji je zapravo gameobject kojeg Igrač ispaljuje iz svoje puške te se sastoji od par varijabli. Speed je brzina projektila, lifeTime je vrijeme života projektila, explosion je gameobject i ponaša se kao Particle effect, damage predstavlja koliko štete radi projektil.

Metoda start poziva funkcije za uništavanje projektila i to je metoda DestroyProjectile, Metoda Kill uništava objekt, OnTriggerEnter2D je funkcija koja kad dođe u kontakt s objektom onda se poziva, u ovom slučaju postoje 3 objekta to su zid, neprijatelj i Lootbox.

```
public class Projectile : MonoBehaviour
{
    public float speed;
    public float lifeTime;
    public GameObject explosion;
    public int damage;
    void Start(){
        Invoke("DestroyProjectile",lifeTime);
        Destroy(gameObject,lifeTime);
    }
    void Update(){
        transform.Translate(Vector2.up * speed * Time.deltaTime);
    }
   private void OnTriggerEnter2D(Collider2D other) {
         if(other.tag=="Enemy"){
            other.GetComponent<Enemy>().TakeDamage(damage,other);
            other.GetComponent<Enemy>().ChangeTarget();
            DestroyProjectile();
            Kill();
            }
         if(other.tag=="LootBox"){
            other.GetComponent<LootBox>().DestroyBox();
            DestroyProjectile();
             }
         if(other.tag=="Wall"){
            DestroyProjectile();
            }
   }
   public void DestroyProjectile(){
        Instantiate(explosion,transform.position,Quaternion.identity);
        Destroy(gameObject);
    }
    void Kill(){
         Destroy(gameObject,lifeTime);
    }
}
```

Programski kod 19. Klasa Projectile

3.4.2. ORUŽJA NA BLIZINU

Programski kod 20 prikazuje da rend varijabla sadrži Sprite, anim sadrži animator komponentu. AttackTime je varijabla koja opisuje vrijeme potrebno za napad, timeBetweenAttacks je vrijeme između svakog napada, damage opisuje koliko radi štete, attackPoint je pozicija za napad, attackRange varijabla je varijabla koja opisuje domet koliko ima oružje za napad, enemyLayers je varijabla koja sadrži sve slojeve neprijatelja, audioSource je za zvuk izvor i AttackSound je zvuk za napad. Update metoda je metoda koja služi najčešće za kontrolu oružja i za napad s oružjem. WeaponControl je metoda koja se poziva preko Update-a, glavna funkcija, te metoda je da kontrolira oružje pomoću miša tj. rotira u željenu poziciju pomoću miša.

```
public class MeleeWeapon : MonoBehaviour
{

    public SpriteRenderer rend;
    public Animator anim;
    float AttackTime;
    public float timeBetweenAttacks;
    public int damage;
    public Transform attackPoint;
    public float attackRange = 0.5f;
    public LayerMask enemyLayers;
    public AudioSource audiosource;
    public AudioClip AttackSound;
   void Update()
   {
       if(!PauseMenu.GameIsPaused){
           WeaponControl();
       }
   if(Input.GetMouseButton(0) && !PauseMenu.GameIsPaused){
       if(Time.time >= AttackTime){
       Attack();
       }
   }
   }

   void WeaponControl(){
        Vector2 direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;

       float angle= Mathf.Atan2(direction.y,direction.x)* Mathf.Rad2Deg;

       Quaternion rotation = Quaternion.AngleAxis(angle -90f, Vector3.forward);

       transform.rotation = rotation;
         if(angle >=90 || angle<= -90){
           rend.flipX = true;

       }
       else{
           rend.flipX = false;


       }

   }
```

Programski kod 20. Klasa MeleeWeapon

Programski kod 21 Attack metoda je funkcija koja služi za napad s oružjem. Metoda OnDrawGizmosSelected je metoda iz MonoBehaviour-a, ona je više kao pomoćna funkcija koja služi za testiranje dometa oružja. Petlja foreach nam govori sadržaju čudovišta ako se nalazi u polju Collider2D, poziva funkciju TakeDamage, gdje funkcija TakeDamage uzima zdravlje od čudovišta tj. oštećuje ga. Ako igrač slučajno udari kutiju onda se kutija razbije tj. poziva se funkcija DestroyBox.

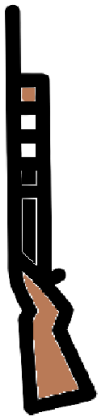
```
void Attack(){

    Debug.Log(gameObject.name);

    audiosource.PlayOneShot(AttackSound);
    if(gameObject.name=="Knife"){
    anim.SetTrigger("Knife");
    }
     if(gameObject.name=="crowbar"){
    anim.SetTrigger("Attack");
    }

    Debug.Log("attack");
    AttackTime = Time.time + timeBetweenAttacks;


   Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position,attackRange,enemyLayers);

   foreach (Collider2D enemy in hitEnemies)
   {
    Debug.Log("we hit" + enemy.name);


     if(enemy.name.Contains("Shadow Monster")){
    enemy.GetComponent<Enemy>().TakeDamage(damage,enemy);
     }
     if(enemy.name.Contains("Bat Monster")){
    enemy.GetComponent<Enemy>().TakeDamage(damage,enemy);
     }
       if(enemy.name.Contains("Brute Monster")){
    enemy.GetComponent<Enemy>().TakeDamage(damage,enemy);
     }

    if(enemy.name.Contains("LootBox")){
    enemy.GetComponent<LootBox>().DestroyBox();
    }


   }
}
private void OnDrawGizmosSelected() {
    if(attackPoint ==null){
        return;
    }
    Gizmos.DrawWireSphere(attackPoint.position,attackRange);
}
```

Programski kod 21. Metode za napad s oružjem i metoda za prikaz dometa oružja

3.4.3.DIZAJN ORUŽJA

Remington

Dizajn je kreiran pomoću logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata. Remington shotgun koristi shotgun shells kao municiju i ima kapacitet od 7 metaka. Vrijeme između pucnjeve je 1 sekunda. Za kreiranje zvuka, resursi su pronađeni na YouTube-u i freesounds.com. Na slici broj 5 je prikazan izgled oružja naziva Remington, koji je napravljen koristeći alate u logomakr-u.

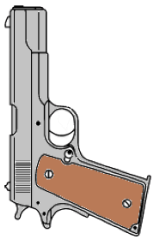


Slika 5. Oružje Remington (Logomakr, 2023)

Colt M1911

Na slici 6 je prikazano oružje naziva Colt M1911. Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

Colt m1911 pištolj koristi handgun bullets kao vrstu municije, ima 13 MaxAmmo-a, time between shots 0.4.



Slika 6. Oružje Colt M1911 (Logomakr, 2023)

Glock-18

Na slici 7 je prikazano oružje naziva Glock-18. Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

Glock-18 pištolj koristi handgun bullets kao vrstu municije, ima 15 MaxAmmo-a, time between shots 0.5

Slika 7. Oružje Glock-18 (Logomakr, 2023)

M16

Na slici 8 je prikazano oružje naziva M16. Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

M16 puška koristi Assault Rifle Bullets kao vrstu municije, ima 18 MaxAmmo-a, time between shots 0.3.

Slika 8. Oružje M16 (Logomakr, 2023)

M14

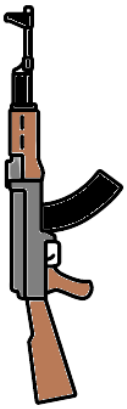
Na slici 9 je prikazano oružje naziva M14. Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

M14 puška koristi Assault Rifle Bullets kao vrstu municije, ima 18 MaxAmmo-a, time between shots 0.5.

Slika 9. Oružje M14 (Logomakr,2023)

AK-47

Na slici 10 prikazano je oružje naziva AK.47. Dizajn je kreiran pomoću Logomakr-a,, AK-47 puška koristi Assault Rifle Bullets kao vrstu municije, ima 20 MaxAmmo-a, time between shots 0.3.

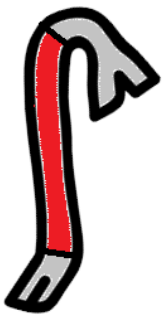


Slika 10. Oružje AK-47 (Logomakr,2023)

CROWBAR

Na slici 11 prikazano je oružje naziva Crowbar. Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

služi za se boriti na blizinu, ima 34 damage-a i time between attacks 1.

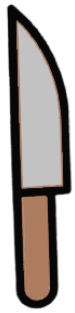


Slika 11. Oružje Crowbar (Logomakr,2023)

KNIFE

Na slici 12 prikazano je oružje-nož.Dizajn je kreiran pomoću Logomakr-a, alata za brzo i jednostavno stvaranje logotipa i vizualnih elemenata.

Služi za se borit na blizinu ima 46 damage-a i time between attacks 0.5.



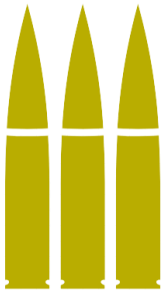
Slika 12. Oružje Knife (Logomakr,2023)

3.4.4. MUNICIJA

Kod municije postoje 3 vrste, a to su: Assault Rifle Bullets, Shotgun shells i Handgun Bullets koje igrač može pokupit iz lootbox-eva.

Assault Rifle Bullets

Na slici 13 je prikazana municija jedne vrste naziva assault rifle bullets. Assault Rifle Bullets su metci koji se koristi kod puška, inače se stvaraju kada igrač razbije lootbox, igrač može uzet da si napuni municiju.



Slika 13. Assault Rifle Bullets (Logomakr,2023)

Programski kod 22 prikazuje kada igrač pokupi metke onda ih dobije, bulletAmount je varijabla koja sadrži koliko će igrač dobiti kada pokupi. Metoda OnTriggerEnter2D je metoda iz MonoBehaviour-a, ona pomaže kod interkacije objekata, ako se dotakne objekt onda se pozove.

```
public class ARAmmoPickup : MonoBehaviour
{

    int BulletAmount=20;



   private void OnTriggerEnter2D(Collider2D other) {

    if(other.gameObject.tag=="Player"){

        FindObjectOfType<Player>().PickupItem();
        Destroy(gameObject);
        FindObjectOfType<WeaponAmmoDisplay>().IncreaseBullets(BulletAmount);
    }
   }
}
```

Programski kod 22. Klasa ARAmmoPickup

Shotgun shells

Na slici 14 prikazana je municija nazvana shotgun shells. Shotgun shells su metci koji se koristi kod sačmarice, inače se stvaraju kada igrač razbije lootbox, igrač može uzet da si napuni municiju.

Slika 14. Shotgun Shells (Logomakr,2023)

Programski kod 23 prikazuje da kada igrač pokupi metke onda ih dobije. ShellAmount je varijabla koja sadrži koliko će igrač dobiti kada pokupi, metoda OnTriggerEnter2D je metoda iz MonoBehaviour-a, ona pomaže kod interakcije objekata, ako se dotakne objekt onda se pozove.

```
public class SGAmmoPickup : MonoBehaviour
{
    int shellAmount=10;

   private void OnTriggerEnter2D(Collider2D other) {

    if(other.gameObject.tag=="Player"){
      FindObjectOfType<Player>().PickupItem();
        Destroy(gameObject);
        FindObjectOfType<WeaponAmmoDisplay>().IncreaseShells(shellAmount);
    }
   }
}
```

Programski kod 23. SGAmmoPickup

Handgun bullets

Na slici 15 prikazana je municija nazvana handgun bullets. Handgun Bullets su metci koje se koriste kod pištolja, inače se stvaraju kada igrač razbije lootbox, igrač može uzet da si napuni municiju.

Slika 15. Handgun Bullets (Logomakr,2023)

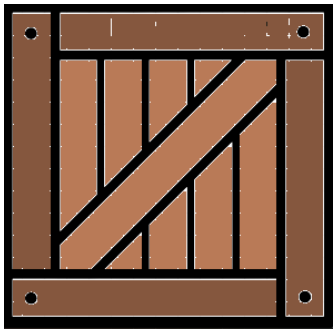
Programski kod 24 prikazuje da kada igrač pokupi metke onda ih dobije. BulletAmount je varijabla koja sadrži koliko će igrač dobiti kad pokupi, metoda OnTriggerEnter2D je metoda iz MonoBehaviour-a, ona pomaže kod interkacije objekata, ako se dotakne objekt onda se pozove.

```
public class HGAmmoPickup : MonoBehaviour
{
    int BulletAmount=20;

   private void OnTriggerEnter2D(Collider2D other) {

    if(other.gameObject.tag=="Player"){
        FindObjectOfType<Player>().PickupItem();
        Destroy(gameObject);
        FindObjectOfType<WeaponAmmoDisplay>().IncreaseHandgunBullets(BulletAmount);
    }
   }
}
```

Programski kod 24. HGAmmoPickup

3.4.5. LOOTBOX

Na slici 16 je prikazan lootbox. Lootbox je kutija koji igrač može uništiti da dobije potrebna sredstva protiv čudovišta, u sredstva se ubrajaju oružja, municije, daske za izradu barikada i MedKit. Inače se lootbox stvara po mapi nasumično tako da igrač mora istražiti mapu kako bi pronašao gdje se nalazi kutija, te je malo teže za sredstva jer se stvaraju nasumično da igra može imati više izazova za igrača.



Slika 16. LootBox (Logomakr,2023)

Programski kod 25 sadrži skriptu koja služi za lootbox. Audisource je varijabla koja sadrži izvor zvuka, BreakSound je zvuk za razbijanje kutije, Drop je varijabla koja sadrži polje, ona je zaslužna za sve stvari koje mogu biti u dropu. Start metoda inicijalizira audisource, Destroybox metoda je metoda gdje kada igrač pukne u kutiju

onda se poziva ta funkcija. DeleteBox metoda je funkcija koja stvara gameobject iz polja i to nasumično, spawnPoint je pomoćna varijabla, metoda SetSpawnPoint je funkcija koja postavlja spawnpoint i metoda OnDestroy je metoda iz MonoBehaviour-a koja kad se uništi objekt se pozove i ovom slučaju metoda nam služi da bi oslobodili mjesto za spawnanje.

```
public class LootBox : MonoBehaviour
{
    public AudioSource audiosource;
    public AudioClip BreakSound;
    public GameObject[] Drop;

    private void Start() {
        audiosource = FindObjectOfType<Player>().GetComponent<AudioSource>();
    }
    public void DestroyBox(){
            audiosource.PlayOneShot(BreakSound);
            Invoke("DeleteBox",.3f);

    }
    public void DeleteBox(){
            int randomIndex = Random.Range(0, Drop.Length);
                GameObject randomObject = Drop[randomIndex];
                Instantiate(randomObject, transform.position, Quaternion.identity);

            GameObject remove = gameObject;
            Destroy(remove);
            Debug.Log("LootBox");

    }
    private LootBoxSpawner.LootBoxSpawnPoint spawnPoint;

    public void SetSpawnPoint(LootBoxSpawner.LootBoxSpawnPoint point)
    {
        spawnPoint = point;
    }

    private void OnDestroy()
    {
        if (spawnPoint != null){
            spawnPoint.isOccupied = false;
        }
    }
```

Programski kod 25. LootBox

3.4.6. BARIKADE

U igri, barikade služe kao sredstvo za obranu od čudovišta. Daju igraču veću obranu i mogu pomoći u samoobrani od neprijateljskih napada. Igraču je potrebna jedna daska ili daska za izgradnju barikade.

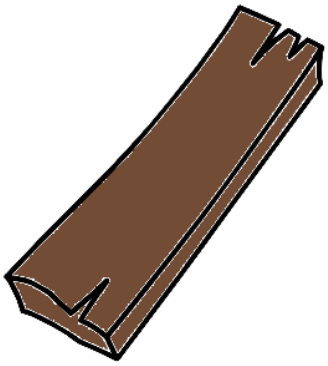
Igrač mora locirati daske u lootbox-evima koje se generiraju diljem karte kako bi izgradio barikadu. Spremnici koji se nazivaju lootboxes sadrže razne vrste nagrada, uključujući oružje, zdravlje, streljivo i druge praktične predmete.

Programski kod 26 plankAmount je varijabla koja sadrži vrijednost koliko ima dasaka. Planksfull je varijabla za sučelje koja samo prikazuje igraču da ima pun inventory , DestroyDelay je varijabla koja se koristi za vrijeme koliko treba da se uništi item, start metoda incijalizira varijable i poziva funkcije za uništavanje objekta i to je DestroyPickup. OnTrigger2D se poziva kad igrač uzme dasku, metoda disableText miče tekst nakon nekoliko sekundi.

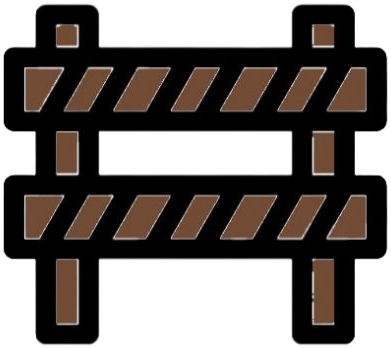
```
public class PlankPickup : MonoBehaviour
{
    int plankAmount = 1;
    public TMP_Text planksfull;
    public int DestoryDelay=10;

    private void Start() {
        planksfull = FindObjectOfType<DisableText>().already3;
        planksfull.enabled = false;
    StartCoroutine(DestoryPickup());
    }
    IEnumerator DestoryPickup(){
        yield return new WaitForSeconds(DestoryDelay);
        GameObject ObjToRemove=gameObject;

        Destroy(ObjToRemove);
    }
    private void OnTriggerEnter2D(Collider2D other) {
        if( FindObjectOfType<Player>().planks ==3){
            Debug.Log("Inventory full");
            planksfull.enabled = true;
            Invoke("disabletext",1.5f);
        }else{
        if(other.gameObject.tag=="Player"){
            FindObjectOfType<Player>().PickupItem();
            Debug.Log("planks pickedup");
            FindObjectOfType<Player>().IncreasePlanks(plankAmount);
            FindObjectOfType<BuildBarricade>().UpdateDisplay();
            Destroy(gameObject);
        }
        }
    }
   void disabletext(){
    planksfull.enabled = false;
    }
}
```

Programski kod 26. Klasa PlankPickup

Na slikama 17 i 18 su prikazani barikade.



Slika 17. Plank (Logomakr,2023)



Slika 18. Barricade (Logomakr,2023)

Programski kod 27 sadrži prikaz svih atributa od skripte, BreakSound i BuildSound su zvukovi, audiosource je izvor zvuka, eastSign i WestSign varijable služe za tekst, barricades polje sadrži sve objekte, isOccupied je bool varijabla koja sadrži da li je okupirano mjesto ili ne, plankCost je varijabla koja sadrži koliko iznosi 1 barikada. RebuildCoolDown je varijabla koja služi da bi postojao neki vremenski odmak od novog buildanja i renderer sadrži Sprite Renderer od barikade.

```
public class BarricadeSpot : MonoBehaviour
{

    public AudioClip BreakSound;
    public AudioClip BuildSound;
    public AudioSource audisource;
    public GameObject eastSign;
    public GameObject westSign;
    public GameObject[] barricades;
    public bool isOccupied;
    public float plankCost = 1f;
    public float rebuildCooldown = 2f;
    private float nextRebuildTime = 0f;
    private Renderer renderer;
```

Programski kod 27. BarricadeSpot

Programski kod 28 prikazuje metodu Start koja incijalizira varijable, NotOcupied služi kako bi se varijabla isOccupied postavila na false.

```
private void Start()
{
    renderer = GetComponentInChildren<Renderer>();
    defaultColor = renderer.material.color;

}
public void NotOcupied(){
    this.isOccupied=false;
}
```

Programski kod 28. Metoda start i NotOcupied

Programski kod 29 TakeDamage metoda služi za uništavanje barikada.

```
public void TakeDamage(BarricadeSpot barricadeSpot)
{

        barricadeSpot.isOccupied = false;

        barricadeSpot.nextRebuildTime = Time.time + rebuildCooldown;

        foreach (Transform child in transform)
        {
            if (child.CompareTag("Barricade"))
            {
                child.gameObject.SetActive(false);
            }
        }

}
```

Programski kod 29. Metoda TakeDamage

Programski kod 30 metoda Update služi da bi provjerili da li je dostupno mjesto za izgradit barikadu, a ako je, onda će se ona i izgradit.

```
private void Update()
{
    RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.mousePosition), Vector2.zero);
    if (hit.collider != null && hit.collider.gameObject == gameObject)
    {
        if (Input.GetKeyDown(KeyCode.F) && !isOccupied && Time.time >= nextRebuildTime && FindObjectOfType<Player>().planks >= plankCost)
        {
            audisource.PlayOneShot(BuildSound);

            foreach (GameObject barricade in barricades)
            {
                if (!barricade.activeSelf)
                {
                    barricade.SetActive(true);
                    barricade.transform.position = transform.position;
                    barricade.transform.rotation = transform.rotation;
                    break;
                }
            }
            isOccupied = true;
            FindObjectOfType<Player>().planks -= plankCost;
            FindObjectOfType<BuildBarricade>().UpdateDisplay();
        }
    }
```

Programski kod 30. Metoda Update

3.5. KORISNIČKO SUČELJE

3.5.1.Glavni izbornik

Glavni izbornik se sastoji od Play, Options, Credits i Quit gumbova.

Programski kod 31 prikazuje PlayGame metodu koja se poziva kad se pokreće igra. QuitGame metoda se poziva kad igrač klikne na gumb Quit.

```
public class MainMenu : MonoBehaviour
{


    public void PlayGame(){
        SceneManager.LoadScene(1);
        FindObjectOfType<AudioManager>().FirstAudio();
    }

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


}
```

Programski kod 31. Klasa MainMenu

3.5.2. Izbornik pauziranja

Programski kod 32 GameIsPaused je varijabla koja govori o tome da li je igra pauzirana ili ne, PauseMenuUI je varijabla koja sadrži sučelje za pauzu.

Metoda Update služi kako bi došli do izbornika, Resume funkcija je metoda koja nastavlja s igrom, Pause je metoda koja pauzira, a Menu metoda je funkcija koja pokreće drugu scenu u ovom slučaju je glavni Izbornik.

```
public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused=false;

    public GameObject pauseMenuUI;

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape)){
            if(GameIsPaused){
                Resume();
            }else{
                Pause();
            }
        }
    }
    public void Resume(){
        pauseMenuUI.SetActive(false);
        Time.timeScale =1f;
        GameIsPaused=false;

    }
    void Pause(){
        pauseMenuUI.SetActive(true);
        Time.timeScale =0f;
        GameIsPaused=true;

    }
    public void Menu(){
        SceneManager.LoadScene(0);
    }
}
```

Programski kod 32. Klasa PauseMenu

3.5.3. Zaslon za dovršenu igru

Programski kod 33 Metoda Setup namješta Zaslon za dovršenu igru, metoda RestartLevel pokreće ponovno nivo, ReturnToMenu se vraća u glavni izbornik.

```
public class GameOverScreen : MonoBehaviour
{

 public void Setup(){
    gameObject.SetActive(true);
    Time.timeScale=0;
    FindObjectOfType<CurrentWeapons>().gameObject.SetActive(false);
 }

 public void RestartLevel(){
    Time.timeScale=1;
    int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
    SceneManager.LoadScene(currentSceneIndex, LoadSceneMode.Single);
 }

 public void ReturnToMenu(){
    Time.timeScale=1;
    SceneManager.LoadScene(0);
 }



}
```

Programski kod 33. Klasa GameOverScreen

3.5.4. Pop up tekst

Na zaslonu postoji i pop up tekst koji pomažu igraču da dobije neki feedback od igre.

Programski kod 34 prikazuje AlreadyEquiped, already3, already100 varijable koje sadrže tekst, start metoda incijalizira tj. onemogućuje njihovu vidljivost na ekranu.

```
public class DisableText : MonoBehaviour
{
    public TMP_Text AlreadyEquiped;
    public TMP_Text already3;
    public TMP_Text already100;

    void Start()
    {
        AlreadyEquiped.enabled = false;
        already3.enabled = false;
        already100.enabled = false;

    }
}
```

Programski kod 34. DisableText

3.5.5. Prikaz brojač valova

Programski kod 35 prikazuje WaveDisplay varijablu koja sadrži tekst za sučelje, metoda UpdateDisplay je metoda koja ažurira valove kad se poziva funkcija.

```
public class WaveCounterDisplay : MonoBehaviour
{
    public TMP_Text WaveDisplay;

    public void UpdateDisplay(string num){
     WaveDisplay.text = "Wave: "+ num;
    }

}
```

Programski kod 35. Klasa WaveCounterDisplay

3.5.6. Audio

3.5.6.1. AudioManager

Programski kod 36 AudioManager je varijabla koja sadrži instancu tj. sebe, warmusic i ThemeMusic sadrže zvuk, MUSIC_KEY i SFX_KEY sadrže stringove o pojedinim zvukovima, Awake metoda je metoda koja pomaže da se iz jedne scene do druge prenosi instanca. Start metoda mijenja zvuk, LoadVolume je metoda koja učitava zvukove, FirstAudio je metoda koja mijenja izvor zvuka u warmusic zvuk i ChangeAudio mijenja zvuk.

```
public class AudioManager : MonoBehaviour
{
    [SerializeField] AudioMixer mixer;

    public static AudioManager instance;
    public AudioClip warmusic;
    public AudioClip ThemeMusic;
    public const string MUSIC_KEY = "musicVolume";
    public const string SFX_KEY = "sfxVolume";
    private void Awake() {
        if(instance== null){
            instance = this;
            DontDestroyOnLoad(gameObject);
        }else{
            Destroy(gameObject);
        }
        LoadVolume();
    }
    private void Start() {
        if(SceneManager.GetActiveScene().buildIndex==1){
            Debug.Log("fun");
            GetComponent<AudioSource>().clip=warmusic;
        }
    }
    void LoadVolume(){
        float musicVolume = PlayerPrefs.GetFloat(MUSIC_KEY,1f);
        float sfxVolume = PlayerPrefs.GetFloat(SFX_KEY,1f);

        mixer.SetFloat(VolumeSettings.MIXER_MUSIC,Mathf.Log10(musicVolume)*20);
        mixer.SetFloat(VolumeSettings.MIXER_SFX,Mathf.Log10(sfxVolume)*20);
    }

    public void FirstAudio(){
        GetComponent<AudioSource>().clip=warmusic;

    }

    public void ChangeAudio(){
        Debug.Log("change audio");
        GetComponent<AudioSource>().clip=ThemeMusic;
        GetComponent<AudioSource>().Play();
    }

}
```

Programski kod 36. Klasa AudioManager

3.5.6.2. VolumeSettings

Programski kod 37 mixer je varijabla koja sadrži AudioMixer komponentu, MusicSlider i SFXSlider su varijable koje sadrže komponente slider i postoje još i dva stringa, a to su MusicVolume i SFXVolume. Awake metoda je funkcija koja dohvaća slušatelje na muziku i sfx, Start metoda je funkcija koja sadrži vrijednosti muzike i SFX-a s time da dohvača one koje je igrač postavio, SetMusicVolume i SetSFXVolume su funkcije koje namještaju zvuk.

```
public class VolumeSettings : MonoBehaviour
{
   [SerializeField] AudioMixer mixer;
   [SerializeField] Slider MusicSlider;
   [SerializeField] Slider SFXSlider;
   public const string MIXER_MUSIC = "MusicVolume";
   public const string MIXER_SFX = "SFXVolume";

    private void Awake() {
        MusicSlider.onValueChanged.AddListener(SetMusicVolume);
        SFXSlider.onValueChanged.AddListener(SetSFXVolume);
    }

    private void Start() {
        MusicSlider.value= PlayerPrefs.GetFloat(AudioManager.MUSIC_KEY,1f);
       SFXSlider.value= PlayerPrefs.GetFloat(AudioManager.SFX_KEY,1f);

    }

    private void OnDisable() {
    PlayerPrefs.SetFloat(AudioManager.MUSIC_KEY,MusicSlider.value);
    PlayerPrefs.SetFloat(AudioManager.SFX_KEY,SFXSlider.value);

    }

    public void SetMusicVolume(float value){
        mixer.SetFloat(MIXER_MUSIC,Mathf.Log10(value)*20);
    }
    public void SetSFXVolume(float value){
        mixer.SetFloat(MIXER_SFX,Mathf.Log10(value)*20);
    }

}
```

Programski kod 37. Klasa VolumeSettings

3.5.6.3. UIClick

Programski kod 38 audiosource je varijabla za izvor zvuka, buttonClick je varijabla koja sarži zvuk za gumb, ButtonClick je metoda koja reproducira zvuk i poziva se svaki put kad se klikne gumb na sučelju.

```
public class UIClick : MonoBehaviour
{

    public AudioSource audiosource;
    public AudioClip buttonclick;


   public void ButtonClick(){
    audiosource.PlayOneShot(buttonclick);
   }
}
```

Programski kod 38. Klasa UIClick

3.5.7. VideoKontroler

Programski kod 39 myVideo sadrži komponentu videoPlayer, video sadrži video, hasPlayed sadrži vrijednost da li je video reproduciran ili ne, objectToEnable je varijabla koja sadrži polje gameobject-a, Awake metoda govori o tome da li je reproduciran video i reproducira ga, Update metoda se koristi za preskočiti video, Start metoda dohvaća komponentu VideoPlayer, Endvideo je metoda koja preskoči video kada se pritisne tipka ESC, EnableAllObjects je metoda koja sve objektu u polju omogući.

```
public class VideoControler : MonoBehaviour
{
    [SerializeField] VideoPlayer myvideo;
    public GameObject video;
    bool hasPlayed = false;
    public GameObject[] objectsToEnable;
    private void Awake() {
        myvideo.loopPointReached += EndVideo;
        if (hasPlayed)
        {
            myvideo.enabled = false;
        }
    }
    private void Update() {
        if (Input.GetKeyDown(KeyCode.Escape)) {

        EndVideo(myvideo);
}
    }
    void Start() {
        myvideo = GetComponent<VideoPlayer>();
    }
    public void EndVideo(VideoPlayer vp){
        video.SetActive(false);
        hasPlayed = true;
        EnableAllObjects();
        FindObjectOfType<PauseMenu>().Resume();
    }
    public void EnableAllObjects(){
        foreach (GameObject obj in objectsToEnable) {
            obj.SetActive(true);
        }
        FindObjectOfType<AudioManager>().ChangeAudio();
    }
}
```

Programski kod 39. Klasa VideoControler

3.5.8. Zona

Programski kod 40 health je varijabla koja opisuje zdravlje zone, slider je varijabla koja sadrži komponentu Slider, GameOverScreen je gameobject koji sadrži zaslon za dovršenu igru, TakeDamage metoda je funkcija koja se poziva svaki put kad netko ošteti zonu tj. neprijatelji, GameOver je funkcija koja se poziva kad je kraj igre tj. kad zona ima 0 zdravlja ili igrač dođe do 0 zdravlja, SetHealth funkcija mijenja zdravlje zone svaki put kad netko ošteti zonu.

```
public class Zone : MonoBehaviour
{
    public int health;
    public Slider slider;
    public GameOverScreen GameOverScreen;

   public void TakeDamage(int damageAmount){

        health = health - damageAmount;
        SetHealth(health);
        if(health <=0){
            SetHealth(health);
            GameOver();
        }
    }

       public void GameOver(){
            Debug.Log("Game Over you LOST");
            GameOverScreen.Setup();
        }

        public void SetHealth(int health){

        slider.value = health;
    }
}
```

Programski kod 40. Klasa Zone

3.5.9. Valni Spawner

Programski kod 41 klasa Wave se odnosi na val, enemies polje sadrži sve neprijatelje, count označava broj neprijatelja, timeBetweenSpawns označava vremenski koliko

između svakog neprijatelja koji se stvorio da se stvori, finishedSpawning je bool varijabla koja provjerava da li je gotovo stvaranje neprijatelja.

```
public class WaveSpawner : MonoBehaviour
{
    [System.Serializable]
        public class Wave {
            public Enemy[] enemies;
            public int count;
            public float timeBetweenSpawns;
        }

        private bool finishedSpawning;

        public Wave[] waves;

        public Transform[] spawnPoints;
        public float timeBetweenWaves;
    public GameOverScreen GameOverScreen;

    public Wave currentWave;
    public int currentWaveIndex;

    public Transform player;

    private void Start() {
        player = GameObject.FindGameObjectWithTag("Player").transform;
        StartCoroutine(StartNextWave(currentWaveIndex));
    }

    IEnumerator StartNextWave( int index){

        yield return new WaitForSeconds(timeBetweenWaves);
        StartCoroutine(SpawnWave(index));
    }
```

Programski kod 41. Klasa WaveSpawner

Programski kod 42 prikazuje SpawnWave metodu čija je funkcija da stvara valove neprijatelja. Update metoda je funkcija koja poziva stalno StartNextWave i kad je gotov onda je gotova igra.

```
    }
    IEnumerator SpawnWave(int index){
        currentWave = waves[index];
        for(int i =0; i<currentWave.count;i++){
            if(player==null){
                yield break;
            }else{
                Enemy randomEnemy=currentWave.enemies[Random.Range(0,currentWave.enemies.Length)];
                Transform randomSpot = spawnPoints[Random.Range(0,spawnPoints.Length)];
                Instantiate(randomEnemy,randomSpot.position,randomSpot.rotation);
                if(i == currentWave.count -1){
                    finishedSpawning = true;
                }else{
                    finishedSpawning = false;
                }
                yield return new WaitForSeconds(currentWave.timeBetweenSpawns);
            }
        }

    }
    private void Update() {
        int num = currentWaveIndex+1;
        FindObjectOfType<WaveCounterDisplay>().UpdateDisplay(num.ToString());

        if(finishedSpawning==true && GameObject.FindGameObjectsWithTag("Enemy").Length==0 ){
                finishedSpawning=false;
                if(currentWaveIndex +1 < waves.Length){
                    currentWaveIndex++;
                    StartCoroutine(StartNextWave(currentWaveIndex));
                }else{
                    Debug.Log("GAME FINISHED!!!");
                    GameOverScreen.Setup();
                }
        }
    }
```

Programski kod 42. Metoda SpawnWave i Update

3.5.10. LootBox Spawner

Programski kod 43 prikazuje klasu LootBoxSpawnPoint koja sadrži transform poziciju isOccupied koja opisuje poziciju kutije i da li je okupiran, lootBoxPrefab sadrži prefab od lootbox-a, maxLootBoxes sadrži vrijednost koliko kutija se može stvoriti odjednom, spawnDelay je koliko treba sačekati da se stvori kutija, spawnPoints sadrži mjesta za stvaranje kutija, spawnedLootBoxes su kutije koje su trenutno stvorene, start metoda pokreće stvaranje kutija, SpawnLootBoxes stvara kutije.

```
public class LootBoxSpawner : MonoBehaviour
{
    [System.Serializable]
    public class LootBoxSpawnPoint{
        public Transform transform;
        public bool isOccupied;
    }
    public GameObject lootBoxPrefab;
    public int maxLootBoxes = 4;
    public float spawnDelay = 5f;
    public LootBoxSpawnPoint[] spawnPoints;
    private List<GameObject> spawnedLootBoxes = new List<GameObject>();

    private void Start() {
        StartCoroutine(SpawnLootBoxes());
    }
    private IEnumerator SpawnLootBoxes()
    {
        while (true)
        {
            if (spawnedLootBoxes.Count < maxLootBoxes)
            {
                LootBoxSpawnPoint freeSpawnPoint = GetRandomFreeSpawnPoint();
                if (freeSpawnPoint.transform != null)
                {
                GameObject lootBoxObject = Instantiate(lootBoxPrefab, freeSpawnPoint.transform.position, Quaternion.identity);
                LootBox lootBox = lootBoxObject.GetComponent<LootBox>();
                lootBox.SetSpawnPoint(freeSpawnPoint);
                spawnedLootBoxes.Add(lootBoxObject);
                freeSpawnPoint.isOccupied = true;
                }
            }
            yield return new WaitForSeconds(spawnDelay);
        }
    }

    private LootBoxSpawnPoint GetRandomFreeSpawnPoint()
```

Programski kod 43. Klasa LootBoxSpawner

Programski kod 44 GetRandomFreeSpawnPoint je funkcija koja dohvaća sva moguća mjesta koja su slobodna.

```
private LootBoxSpawnPoint GetRandomFreeSpawnPoint()
{
    List<LootBoxSpawnPoint> freeSpawnPoints = new List<LootBoxSpawnPoint>();

    foreach (LootBoxSpawnPoint spawnPoint in spawnPoints)
    {
        if (!spawnPoint.isOccupied)
        {
            freeSpawnPoints.Add(spawnPoint);
        }
    }

    if (freeSpawnPoints.Count > 0)
    {
        int randomIndex = Random.Range(0, freeSpawnPoints.Count);
        return freeSpawnPoints[randomIndex];
    }

    return new LootBoxSpawnPoint();
}
```

Programski kod 44. Metoda LootBoxSpawnPoint

3.6. MAPA

Na slici 19 je prikan pregled mape. Za izradu mape u Unity-u, koristio se Tile Pallet alat i upotrijebljeno je Sprite Sheet koji je izrađena u GIMP-u. Mapa se sastoji od pločica poput trave, kamena, blata i zemlje. Kako bi pločice bile oblikovane, korišten je Tile Pallet alat u Unity-u. Pomoću ovog alata su dodavane i organizirane pločice koje čine mapu igre. Sprite Sheet je slika koja sadrži više malih sličica (spriteova). Na ovom Sprite Sheetu se nalaze elementi poput trave, kamena, blata i zemlje, koje je autor kreirao u GIMP-u. Koristeći Tile Pallet alat, dodavane se odgovarajuće pločice iz Sprite Sheeta na mapu. Pločice su pažljivo postavljene kako bi stvorile željeni izgled mape.

Slika 19. SpriteSheet za mapu

3.7.ČUDOVIŠTA

Čudovišta ili Enemys su podijeljene u dvije skupine, a to su Melee Monster koji predstavlja čudovišta koji napadaju samo u bližem kontaktu i Ranged Monster koji napadaju iz daljine, tj u Unity-u je izrađena Enemys Skriptu i još dvije skripte koje nasljeđuju(inheritance) iz enemy-a.

Programski kod 45 prikazuje Health koja je varijabla za zdravlje. Target i Player su pozicije od mete i igrača, speed je brzina kretanja neprijatelja, timeBetweenAttacks je varijabla koja označava vrijeme između napada, damage je varijabla koja označava koliko neprijatelj može nanijeti štete, deathAnim je animacije za smrt neprijatelja, virtualna metoda start je funkcija koja dohvaća sve potrebne komponente.

```
public class Enemy : MonoBehaviour
{
    public int Health;
    public Transform Target;
    public Transform Player;
    public float speed;
    public float timeBetweenAttacks;
    public int damage;
    public Animator deathAnim;
     public virtual void Start() {
       Target = FindObjectOfType<Zone>().transform;
       Player = FindObjectOfType<Player>().transform;
       deathAnim= GetComponent<Animator>();

    }
```

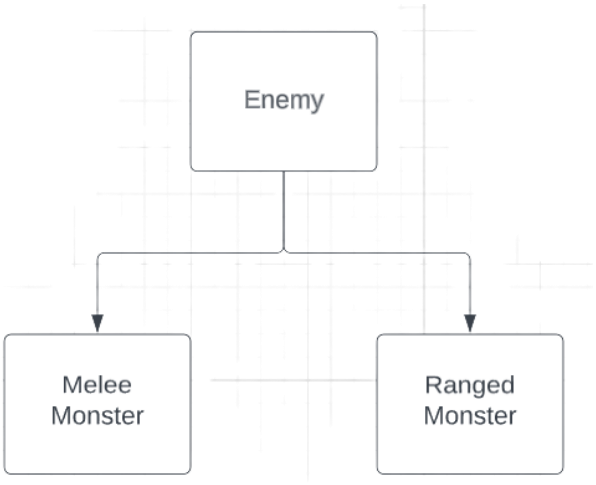
Programski kod 45. Klasa Enemy

Programski kod 46 prikazuje TakeDamage funkciju koja se poziva svaki put kad neprijatelj izgubi zdravlje, Death je metoda koja se poziva kad umre RangedMonster,ChangeTarget metoda je funkcija koja se poziva kad igrač napadne neprijatelja.

```
    public void TakeDamage(int damageAmount, Collider2D enemy){

        Health = Health - damageAmount;

        if(name == "Shadow Monster(Clone)" || name=="Brute Monster(Clone)"){

         if(Health <=0){
            enemy.GetComponent<MeleeMonster>().MonsterGrowl();
            enemy.GetComponent<MeleeMonster>().UpdateDisplay();
            enemy.GetComponent<Enemy>().speed=0;
            enemy.GetComponent<Enemy>().damage=0;
            deathAnim.SetBool("death",true);
         }
        enemy.GetComponent<MeleeMonster>().DamageTaken();
        }
        if(name=="Bat Monster(Clone)"){

            if(Health<=0){
                enemy.GetComponent<RangedMonster>().Death();
                enemy.GetComponent<Enemy>().speed=0;
            }
            enemy.GetComponent<RangedMonster>().DamageTaken();
        }
    }
        public void Death(){
            Debug.Log("dmg");
            Destroy(this.gameObject);

        }
    public void ChangeTarget(){
        Target = Player;
    }
}
```

Programski kod 46. Metoda TakeDamage

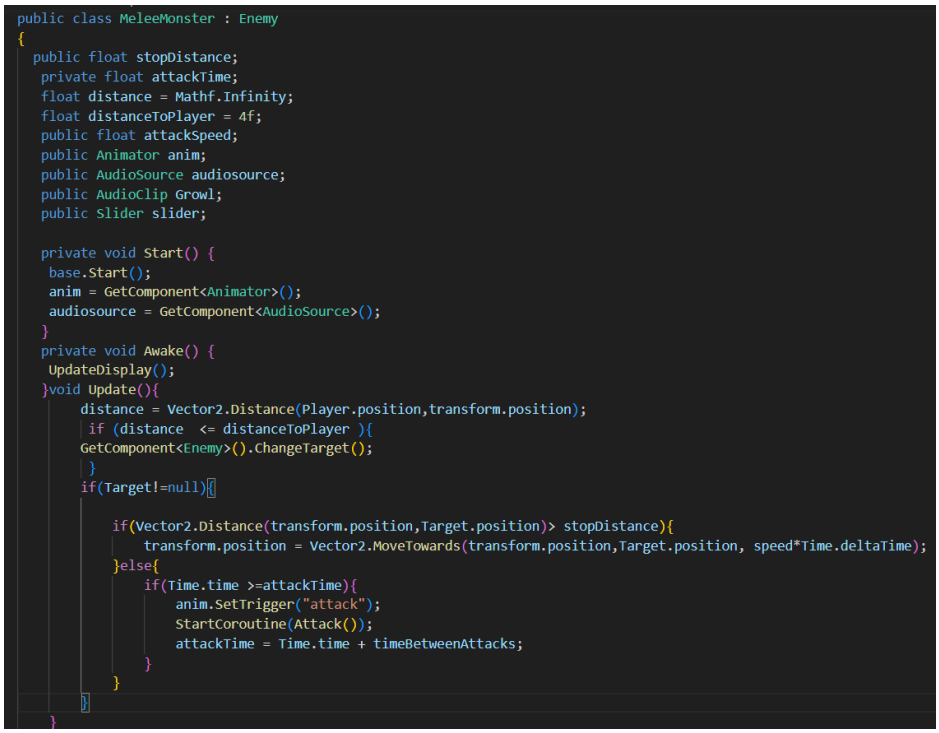
Na slici 20 je prikazano nasljeđivanje kod čudovišta.



Slika 20. Prikaz Nasljeđivanja

3.7.1.MELEE MONSTER

Programski kod 47 prikazuje stopDistance koja je distanca za stopiranje, attackTime je varijabla koja sadrži vrijeme napada, distance je distanca u igri, distanceToPlayer je distanca od objekta do igrača, attackSpeed je brzina napada, anim je varijabla koja sadrži animator, audiosource je varijabla koja sadrži izvor zvuka, Growl je zvuk i slider ja varijabla koja sadrži komponentu slider, Start metoda dohvaća sve potrebne komponente, Awake poziva funkciju UpdateDisplay, Update provjerava distancu od igrača i kreće se prema njemu i napada ako dođe dovoljno blizu.



```
public class MeleeMonster : Enemy
{
  public float stopDistance;
   private float attackTime;
   float distance = Mathf.Infinity;
   float distanceToPlayer = 4f;
   public float attackSpeed;
   public Animator anim;
   public AudioSource audiosource;
   public AudioClip Growl;
   public Slider slider;

   private void Start() {
    base.Start();
    anim = GetComponent<Animator>();
    audiosource = GetComponent<AudioSource>();
   }
   private void Awake() {
    UpdateDisplay();
   }void Update(){
        distance = Vector2.Distance(Player.position,transform.position);
        if (distance  <= distanceToPlayer ){
       GetComponent<Enemy>().ChangeTarget();
        }
       if(Target!=null){

           if(Vector2.Distance(transform.position,Target.position)> stopDistance){
               transform.position = Vector2.MoveTowards(transform.position,Target.position, speed*Time.deltaTime);
           }else{
               if(Time.time >=attackTime){
                   anim.SetTrigger("attack");
                   StartCoroutine(Attack());
                   attackTime = Time.time + timeBetweenAttacks;
               }
           }
       }
   }
```

Programski kod 47. Klasa MeleeMonster

Programski kod 48 OnCollisionEnter2D je metoda koja poziva DestroyTheBarricade koja uništava barikadu ako dođe u kontaktu, MonsterGrowl je funkcija koja reproducira zvuk, DamageTaken metoda je funkcija koja se poziva kad se ošteti neprijatelj, UpdateDisplay je funkcija koja ažurira healthbar od neprijatelja.

```
  private void OnCollisionEnter2D(Collision2D other)
{

    if (other.gameObject.tag=="Barricade") {

    anim.SetTrigger("attack");

    StartCoroutine(DestroyTheBarricade(other));

    }
}

public IEnumerator DestroyTheBarricade(Collision2D other){
    yield return new WaitForSeconds(2.0f);
        other.gameObject.GetComponentInParent<BarricadeSpot>().NotOcupied();
        other.gameObject.SetActive(false);

}

public void MonsterGrowl(){
    audiosource.PlayOneShot(Growl);
    Debug.Log("growl");
}

public void DamageTaken(){
    anim.SetTrigger("damaged");
    UpdateDisplay();
}

public void UpdateDisplay(){
    slider.value = GetComponent<Enemy>().Health;
}
```

Programski kod 48. Metoda OnCollisionEnter2D, DestroyTheBarricade, MonsterGrowl, DamageTaken i UpdateDisplay

Programski kod 49 prikazuje metodu Attack čija se funkcija poziva kad je neprijatelj kod igrača i napada ga.

```
    IEnumerator Attack(){
        if(Target.tag=="Player"){
        Target.GetComponent<Player>().TakeDamage(damage);


        }
        if(Target.tag=="Zone"){
        Target.GetComponent<Zone>().TakeDamage(damage);
        }

        Vector2 originalPosition = transform.position;
        Vector2 targetPosition = Target.position;
        float percent=0;

        if(!audiosource.isPlaying){

       MonsterGrowl();
        }

        while(percent<=1){
            percent +=Time.deltaTime * attackSpeed;
            float formula= (-Mathf.Pow(percent,2)+percent)*4;
            transform.position = Vector2.Lerp(originalPosition,targetPosition, formula);
            yield return null;
        }

    }
}
```

Programski kod 49. Metoda Attack

3.7.1.1.SHADOW MONSTER

Klasa neprijatelja "Melee Monster" uključuje tip "Shadow Monster". Ovaj neprijatelj udara igrača izravno fizičkim udarcima. Čudovištu iz sjene dostupno je 100 bodova zdravlja. Čudovište iz sjene ima vrijeme napada od jedne sekunde, tako da će čekati jednu sekundu nakon svakog napada prije nego što još jednom udariti igrača. Kada čudovište iz sjene napadne, nanosi igraču 14 bodova štete.

Zaustavna udaljenost za Čudovište iz sjene je 2. Ovo znači da kada se Čudovište iz sjene udalji 2 jedinice od igrača, prestat će ga napadati i krenuti iznova. Čudovište iz sjene može koristiti svoje napade na igrača dok mu je što je moguće bliže zahvaljujući ovoj blizini.

Brzina napada Čudovišta iz sjene (brzina napada) je 2. Ovo pokazuje sposobnost Čudovišta iz sjene da brzo izvrši napade. Zbog sporije brzine napada od ostalih neprijatelja, Čudovište iz sjene napadat će rjeđe i sveukupno sporije.

Na slici broj 21 je prikazan takozvani Shadow monster.

Slika 21. Shadow Monster

3.7.1.2.BRUTE MONSTER

Klasa neprijatelja „Melee Monster“ uključuje tip “ Brute Monster“. Ovaj protivnik koristi fizičke udarce kao izravne napade. Za Brute Monster dostupno je 100 bodova zdravlja.

Trajanje napada Brute Monstera je 3 sekunde, tako da će trebati 3 sekunde nakon svakog napada da ponovo pogodi igrača. Brute Monster napada i nanosi 24 boda štete igraču. Zaustavna udaljenost za Brute Monster je 2. Prema tome, Brute Monster će zaustaviti svoje napade na igrače i nastaviti s njima nakon što se udalje 2 jedinice. On može koristiti svoje snažne napade na igrača zbog te blizine.

Brzina napada (attack speed) Brute Monstera iznosi 5. To označava koliko brzo Brute Monster može izvršiti niz napada. Veća brzina napada znači da će Brute Monster biti agresivniji i može nanijeti više štete igraču u kraćem vremenskom razdoblju.

Na slici broj 22 je prikazan takozvani Brute Monster.

Slika 22. Brute Monster

3.7.2.RANGED MONSTER

Programski kod 50 prikazuje stopDistance koja je distanca za stopiranje. AttackTime je vrijeme za napad, anim je varijabla koja sadrži animator, shotPoint je varijabla koja sadrži poziciju za stvaranje projektila, enemyBullet je gameobject projektil, distance je distanca u igri, distanceToPlayerje distanca između neprijatelja i igrača, audiosource je izvor zvuka, Growl je zvuk, death je Gameobject za smrt, slider je varijabla koja sadrži komponentu slider, start metoda dohvaća anim, update provjerava distancu i napada ako je udaljen od igrača

```
public class RangedMonster : Enemy
{
    public float stopDistance;
    private float attackTime;
    public Animator anim;
    public Transform shotPoint;
    public GameObject enemybullet;
    float distance = Mathf.Infinity;
    float distanceToPlayer = 4f;
    public AudioSource audiosource;
    public AudioClip Growl;
    public GameObject death;
    public Slider slider;
    public override void Start() {
     base.Start();
         anim = GetComponent<Animator>();
     }
     private void Update() {

         distance = Vector2.Distance(Player.position,transform.position);

          if (distance  <= distanceToPlayer ){

         FindObjectOfType<Enemy>().ChangeTarget();
          }

         if(Target != null){
         if(Vector2.Distance(transform.position,Target.position)>stopDistance){
             transform.position = Vector2.MoveTowards(transform.position,Target.position,speed*Time.deltaTime);
         }
         if(Time.time >=attackTime){
             attackTime = Time.time + timeBetweenAttacks;
             anim.SetTrigger("attack");
         }

         }
    }
```

Programski kod 50. Klasa RangedMonster

Programski kod 51 DamageTaken je metoda koja se poziva kad se ošteti neprijatelj, MonsterGrowl je metoda koja reproducira zvuk, UpdateDisplay se poziva svaki put kad se ažurira healthbar od neprijjatelja, RangedAttack se poziva kada se napada igrač.

```
}
 public void DamageTaken(){
    anim.SetTrigger("damaged");
    UpdateDisplay();
 }
 public void MonsterGrowl(){
    audiosource.PlayOneShot(Growl);
    Debug.Log("growl");
 }

public void UpdateDisplay(){
    slider.value = GetComponent<Enemy>().Health;
 }
 public void Death(){
    Debug.Log("dead");
    Instantiate(death,transform.position,transform.rotation);
    MonsterGrowl();
    Destroy(this.gameObject);
 }
 public void RangedAttack(){
    Vector2 direction = Target.position - shotPoint.position;
    float angle= Mathf.Atan2(direction.y,direction.x)* Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle -90, Vector3.forward);
    shotPoint.rotation = rotation;

    if(!audiosource.isPlaying){

   MonsterGrowl();
    }

    Instantiate(enemybullet,shotPoint.position,shotPoint.rotation);
}
```

Programski kod 51. Metoda DamageTaken, MonsterGrowl, UpdateDisplay, Death i RangedAttack

3.7.2.1.BAT MONSTER

Klasa neprijatelja "Ranged Monster" uključuje neprijatelje poput Bat Monstera. Ovaj protivnik svojim mecima gađa igrača. Čudovište Šišmiš ima ukupno 100 bodova zdravlja. Interval napada Bat Monstera je 3 sekunde, tako da će čekati 3 sekunde nakon svakog napada prije nego što još jednom puca u igrača. Čudovište Šišmiš napada i nanosi igraču 15 bodova štete (štete).

Zaustavna udaljenost za Čudovište Šišmiš je 4. Prema tome, kada je Čudovište Šišmiš udaljeno 4 jedinice od igrača, ono će zastati i nastaviti pucati na njega. Pritom može pokušati nanijeti štetu držeći se na određenoj sigurnoj udaljenosti od igrača.

Brzina metka neprijatelja (enemy bullet) kojeg Bat Monster puca iznosi 12. To označava brzinu kojom se metak kreće prema igraču. Brži metak može biti teže izbjeći, stoga igrač mora biti spreman reagirati brzo kako bi izbjegao takve napade.

Na slici broj 23 je prikazan takozvani Bat monster.

Slika 23. Bat Monster

3.8.PROJEKTIL

Programski kod 52 prikazuje TargetP koja je varijabla koja sadrži poziciju igrača, targetPosition je pozicija od mete, lifetTIme je koliko traje projektil, damageAmount je onoliko koliko radi štete, speed je varijabla za brzinu kretanja, Start metoda incijalizira neke komponente, Update je metoda koja provjerava distancu, OnTriggerEnter2D je metoda koja se poziva kad projektil dođe u kontakt s nekim objektima kao što su igrač zona ili neprijatelj.

```
public class EnemyProjectile : MonoBehaviour
{
    public Transform TargetP;
    public Vector2 targetPosition;
    public float lifeTime;
    public int damageAmount;
    public float speed;

    private void Start() {
        TargetP = FindObjectOfType<Enemy>().Target;
        targetPosition = TargetP.transform.position;
        Destroy(gameObject,lifeTime);
    }
     void Update() {
      if (Vector2.Distance(transform.position,targetPosition)>.1f){
        transform.position=Vector2.MoveTowards(transform.position,targetPosition,speed*Time.deltaTime);
      } else{
        Destroy(gameObject);
      }
    }
    private void OnTriggerEnter2D(Collider2D other) {
        if(other.tag == "Player"){
            FindObjectOfType<Player>().TakeDamage(damageAmount);
            Destroy(gameObject);
        }
        if(other.tag=="Zone"){
            FindObjectOfType<Zone>().TakeDamage(damageAmount);
        }
         if(other.tag=="Wall"){
            Destroy(gameObject);
        }
    }
}
```

Programski kod 52. Klasa EnemyProjectile

3.9. PLAYER KONTROLE

U igri, kretanje igrača se izvršava pomoću W, A, S i D tipki. Tipka W omogućava kretanje prema gore, tipka A omogućava kretanje u lijevom smjeru, tipka D omogućava kretanje u desnom smjeru, a tipka S omogućava kretanje prema dolje.

Pucanje se izvršava pomoću miša. Kada igrač cilja mišem, oružje će se okrenuti prema tom smjeru. Nakon što je igrač usmjerio oružje prema željenom cilju, može pritisnuti lijevu tipku miša kako bi izvršio pucanj.

Igrač ima mogućnost napuniti svoje oružje s municijom pritiskom na tipku R. To će omogućiti igraču da ponovno napuni svoje oružje ako je potrošio sve metke.

Kada igrač razbije lootbox, ima mogućnost pokupiti različite resurse kao što su municija, zdravlje (medkit), oružja i daske. Korištenjem tipke T, igrač može promijeniti trenutno oružje koje koristi ili ga pokupiti ako već nema takvo oružje.

Ovi mehanizmi omogućuju igraču dinamično kretanje, ciljanje i pucanje u igri. Također, prikupljanje resursa putem lootboxova dodaje element taktike i strategije igraču, pružajući mu različite opcije i prednosti tijekom igre.

ZAKLJUČAK

Game development područje je zaista uznapredovalo unatrag par godina, te se i dalje razvija. Postoje razni alati koji omogućavaju čak i običnom laiku da krene samostalno s izradom igra. Cilj ovog rada je bio stvoriti igru, kompletno od početka. Počevši od samog planiranja, do izrade prototipa i u konačnici izrade same igre. Potrebno je uložiti puno vremena, rada i truda za odabrati one aspekte koji su bitni za razvoj igre. U radu je prikazan i kod koji je korišten kako bi se igrica stvorila, te koji je ključ za realizaciju igre. Idejna zamisao ove igre je misija koju igrač mora preživjeti, pritom se mora obraniti od napada neprijatelja. Igrač ima mogućnost korištenja oružja uz kojih prevladava neprijatelja. Zamisao kao takva je vrlo jednostavna, no svakako postoji mogućnost razvoja igrice, uz dodavanja raznih razina ili proširenja spektra neprijatelja protiv kojih se bori igrač, ili čak osmisliti više vrsta igrača.

U zaključku, Unity razvojno okruženje se pokazao kao ključna sila u području razvoja igara, nudeći moćnu i pristupačnu platformu za programere da ožive svoje vizije. Međutim, kao i svaka tehnologija, Unity ima svoja ograničenja, a razumijevanje tih ograničenja ključno je za uspješno upravljanje projektima.

Prvo, ograničenja Unity projekta često se vrte oko izvedbe i skalabilnosti. Kako projekti postaju ambiciozniji i složeniji, pitanja poput optimizacije i upravljanja resursima mogu predstavljati izazove. Osim toga, 2D mogućnosti Unityja, možda neće zadovoljiti specifične potrebe svih projekata. Stvaranje određenih vrsta 2D igara, kao što su napredne fizičke simulacije ili složene interakcije može zahtijevati dodatnu prilagodbu ili integraciju rješenja trećih strana. Nadalje, uloga Unityja u industrijama izvan igara se širi. Njegova primjena u arhitekturi, obrazovanju, zdravstvu i drugim sektorima dokaz je njegove svestranosti. Kako se Unity nastavlja razvijati, možemo očekivati šire usvajanje u ovim područjima.

Unatoč ovim ograničenjima, budućnost osmišljavanja igara u Unity-u obećava, s nekoliko potencijalnih proširenja na tržište. Unityjeva kontinuirana predanost tehnologijama proširene stvarnosti (AR) i virtualne stvarnosti (VR) otvara vrata novim i impresivnim iskustvima igranja. Rast platformi za igranje u oblaku i prilagodljivost ekosustava Unity uslugama strujanja mogli bi dovesti do inovativnih metoda distribucije.

Iako Unity ima svoja ograničenja, on ostaje pokretačka snaga u industriji igara, s uzbudljivim mogućnostima za širenje. Kako tehnologija napreduje, a kreativni umovi nastavljaju pomicati granice, Unityjeva prilagodljivost i snažna podrška zajednice dobro ga pozicioniraju da oblikuje budućnost ne samo igranja već i brojnih drugih područja. Razumijevanje i snalaženje u njegovim ograničenjima uz istovremeno iskorištavanje njegovog potencijala ključno je za uspješan i inovativan razvoj igara.

LITERATURA

1. GIMP , About GIMP. Dostupno na: https://www.gimp.org/about/introduction.html, (01.07.2023)
2. GitHub. (01.07.2023) Dostupno na: https://github.com/
3. Logomakr (26.06.2023), Dostupno na : https://logomakr.com/
4. Schardon, L. (25.06.2023) What is Unity? A Guide for One of the Top Game Engines.
5. The Importance of Prototyping in Design. 2019 Dostupno na: https://uxdesign.cc/importance-of-prototyping-in-designing-7287c7035a0d (01.07.2023)
6. Unity. (25.06.2023) Dostupno na: https://unity.com/our-company
7. Autor nepoznat. (01.07.2023) What is Audacity? Dostupno na: https://www.educba.com/what-is-audacity/

Zvukovi:

Ak47 shot, dostupno na: https://freesound.org/people/LeMudCrab/sounds/163457/

Ak-47 reload sound, dostupno na: https://freesound.org/people/ser%C3%B8t%C5%8Dnin/sounds/674742/

Colt m1911 shot sound , dostupno na: https://freesound.org/people/Carmelomike/sounds/255216/

Colt m1911 reload sound, dostupno na: https://freesound.org/people/nioczkus/sounds/396331/

Crowbar sound, dostupno na: https://www.youtube.com/watch?v=UCYTRky4aWk

Glock-18 shot sound , dostupno na : https://freesound.org/people/aleksnascimento/sounds/161195/

Glock-18 reload sound , dostupno na: https://freesound.org/people/oneshotofficial/sounds/616758/

AudioMachine - Epica -> intro sound , dostupno na:
https://www.youtube.com/watch?v=i4oNphrwmqk

Knife stab sound , dostupno na: https://www.youtube.com/watch?v=AjJEhMtqDzE

Lootbox break sound & barricade break, dostupno na:

https://www.youtube.com/watch?v=oTiA8ZCL38E

Build barricade sound, dostupno na:
https://freesound.org/people/InspectorJ/sounds/406048/-

M14 shot sound , dostupno na: https://www.youtube.com/watch?v=Mp2CzLCWt7M

M14 reload sound , dostupno na: https://www.youtube.com/watch?v=ZoiSxoA-Qyg

M16 shot sound , dostupno na:
https://freesound.org/people/qubodup/sounds/162404/

M16 reload sound , dostupno na:
https://freesound.org/people/GFL7/sounds/276964/

Pickup sound, dostupno na: https://www.youtube.com/watch?v=xku7oOuJrQU

Remington shot sound, dostupno na:
https://freesound.org/people/seanmorrissey96/sounds/509432/

Remington reload sound , dostupno na:
https://www.youtube.com/watch?v=pTIJfrTxFbw

Music theme , dostupno na: https://www.youtube.com/watch?v=bpI2cFSmzsM

Click za UI gumbove , dostupno na:
https://freesound.org/people/Breviceps/sounds/448086/

Popis slika

Popis programskih kodova

SAŽETAK

Unity razvojno okruženje je vrlo jak alat za razvoj igara, no dolazi sa ograničenjima, pogotovo ako se kompleksnost samog projekta produbljuje. No njegova adaptivnost i jaka podrška zajednice čini ovaj alat odličan za razvoj jednostavnih igara poput ove opisane u radu. Razumijevanje ograničenja ovog alata, no istovremeno iskorištavanje njegovog potencijala je ključno u razvoju uspješnih i inovativnih projekata. U radu se opisuje proces kreiranja alata pomoću Unity platforme s dodatnim alatima koji su pomogli u izradi igre. Cilj igre je da igrač preživi napade neprijatelja, te se mora obraniti da bi pobijedio igru, kretanjem po mapi i pronalaženjem kutije pomoću kojih može preživjeti.

Ključne riječi: Unity razvojno okruženje, igra, 2D, GIMP, GitHub

ABSTRACT

Unity game engine is a very powerful tool for game development, but it comes with limitations, especially if the project’s complexity itself deepens. However, its adaptability and strong community support make this tool great for developing simple games like the one described in the paper. Understanding the limitations of this tool, while at the same time exploiting its potential, is crucial to the development of successful and innovative projects. The bachelor’s thesis describes the process of creating tools using the Unity platform, along with additional tools that helped create the game. The aim of the game is for the player to survive the attacks of the enemies, and the player has to defend himself to win the game by moving around the map and finding the boxes that he can use to survive.

Keywords: Unity game engine, game, 2D, GIMP, GitHub