Razvoj računalne igre Trial of Cronos

Papić, Filip

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Filip Papić

Razvoj računalne igre Trial of Cronos

Završni rad

Mentor: izv. prof. dr. sc. Miran Pobar

Rijeka, 2024.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Rijeka, 2.5.2024.

Zadatak za završni rad

Pristupnik: Filip Papić

Naziv završnog rada: Razvoj računalne igre Trial of Cronos

Naziv završnog rada na engleskom jeziku: Development of Trial of Cronos computer game

Sadržaj zadatka: Proučiti i opisati faze razvoja računalnih igara. Izraditi i opisati vlastitu računalnu igru prema prethodno izrađenom dokumentu dizajna računalne igre. Opisati implementaciju igre, s naglaskom na glavnim sustavima i dijelovima funkcionalnosti, poput dućana i inventara, kretanja likova i neprijatelja.

Mentor
Izv. prof. dr. sc. Miran Pobar

Voditelj za završne radove
Izv. prof. dr. sc. Miran Pobar

Zadatak preuzet: 2.5.2024.

FPapić
(potpis pristupnika)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sadržaj

Sažetak

Tema ovog završnog rada je razvoj računalne igre Trial of Cronos, čiji će se proces detaljno opisati u nastavku rada. Prikazuje kako su pojedini koncepti i elementi igre preneseni u kod koristeći programski jezik C# i alat za razvoj igara Unity.

Na samom početku, kratko ćemo se osvrnuti na industriju video igara i njen povijesni razvoj, kako bismo pružili kontekst u kojem se ova igra nalazi. U nastavku ćemo se detaljnije posvetiti Unityju, opisu njegovih ključnih značajki i njegovoj ulozi u modernom razvoju igara. Posebna pozornost bit će posvećena alatima i tehnikama korištenim u razvoju Trial of Cronos, kao i izazovima koji su se pojavili tijekom programiranja igre.

Uz detaljan opis razvoja igre, rad se osvrće i na specifične dijelove gameplaya, mape, neprijatelja te tehničke implementacije, uključujući sustave animacija i interakcija s igračkim svijetom.

Ključne riječi: završni rad; Unity; C#; Razvoj računalne igre

1. Uvod

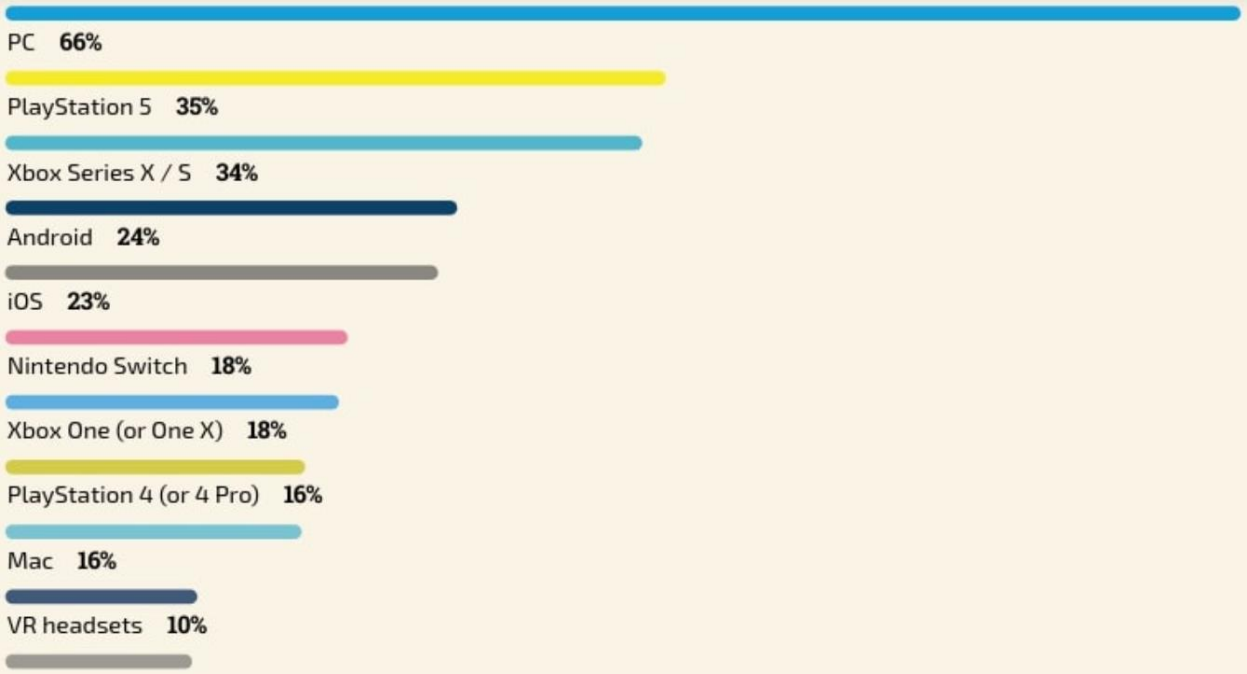
Trial of Cronos je 2D top-down shooter igra s naglaskom na elemente preživljavanja i taktičkog upravljanja resursima. Igra je smještena u grčkoj mitologiji, a igrač preuzima ulogu titana Cronosa koji se bori protiv grčkih bogova kako bi preuzeo kontrolu nad Olimpom. Posebnost igre je u mehanici manipulacije vremenom, koja omogućava stvaranje resursa potrebnih za napredak kroz igru. Uz detaljan opis razvoja igre, rad se osvrće i na specifične dijelove gameplaya, mape, neprijatelja te tehničke implementacije, uključujući sustave animacija i interakcija s igračkim svijetom.

Cilj ovog završnog rada je pružiti uvid u proces razvoja računalne igre te objasniti kako su teorijska znanja iz područja programiranja i razvoja igara primijenjena na stvarni projekt. Rad će također istaknuti ključne elemente i tehnike korištene u stvaranju igre Trial of Cronos, čime će se pokazati sveobuhvatnost razvoja igre od ideje do konačnog proizvoda.

Početak video igara bio je 1950. – 1960. kada su programeri počeli stvarati jednostavne igre i simulacije od kojih je jedna od prvih takvih igara bila „Spacewar!“ 1962. koja je napravljena za tada predstavljeno računalo PDP-1 (Programmed data Proccesor – 1).

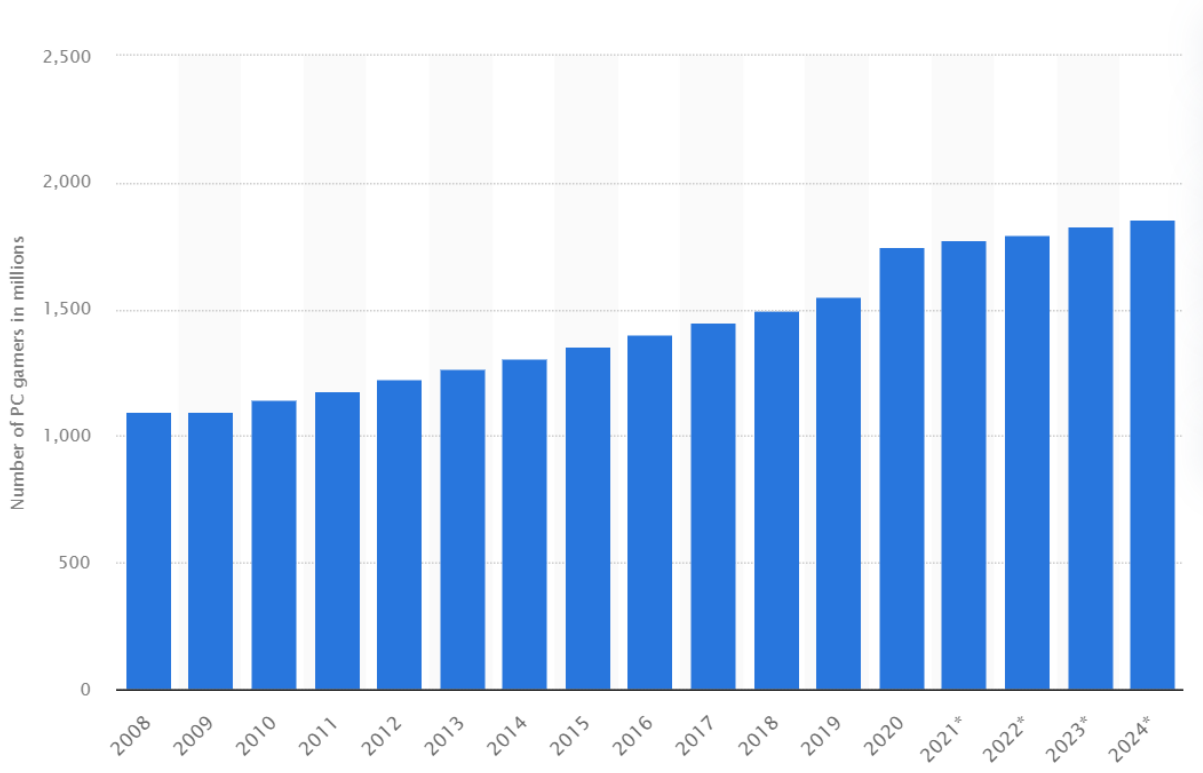
Prva igrača konzola bila je Magnavox Odyssey i prva arkadna igra popularni Pong. Igraća konzola je elektronički uređaj koji izbacuje video signal za prikaz video igre koju se može igrati nekim kontrolerom. Konzole mogu biti „kućne“ konzole koje su većinom stavljene na određeno mjesto ili mogu biti „ručne“ konzole kojima je i prikaz igre i kontroler ugrađen u jedan uređaj i mogu se igrati bilo gdje.

Početkom 1990. godine računalne igre gube na popularnosti zbog brzog rasta igraćih konzola pete generacije kao što su Nintendo 64 i PlayStation. Sredinom 2000-ih računalne igre vraćaju popularnost kroz digitalnu distribuciju istih poslije čega su kontinuirano na prvome mjesto što se tiče razvoja i igranosti igara.



Slika 1. Graf razvoja igara po platformi, [2]

Na Slici 4. vidimo graf koji prikazuje u kojem se postotku za koju platformu razvija najviše igara u 2024. godini.



Slika 2. Graf PC igrača po godini, [5]

Na Slici 5. vidimo graf koji prikazuje broj ljudi koji igra računalne igre od 2008. do 2024. godine prema „Statista“ stranici.

Možemo vidjeti kontinuiran rast od čak 800 milijuna ljudi u proteklih 16 godina.

Velik doprinos ovome broju ide popularnim „digitalnim dućanima“ kao što su Steam i Epic Games zbog kojih su razne igre dostupnije ljudima diljem svijeta nego što bi inače bile.

Isto tako jedan od ključnih razloga za povećanje broja igrača je širenje internetske povezanosti što je omogućilo razvoj eSportova što je privuklo milijune novih igrača.

Stavka koja nije zanemariva je i to da su besplatne igre zapravo jedne od najvećih i najboljih igara kao što su: League of Legends, Fortnite, Dota2, Counter-Strike: Globa Offensive i druge. Samim time što su besplatne igre će doprijeti do puno više ljudi od onih koje se plaćaju.

Velik rast u popularnosti streaming platformi kao što su Youtube i Twitch također su doprinijele popularnosti igara tako da osim što mogu igrati igru sada mogu i gledati kako drugi ljudi igraju igre te možda i sami htjeli zaigrati.

Tipičan razvoj računalne igre u može se podijeliti na nekoliko ključnih koraka:

- **Planiranje i dizajn:** U ovoj fazi definiraju se koncept, priča, likovi i osnovne mehanike igre. Kreira se dokument za dizajn igre (GDD) prema kojem se izvodi cijeli razvojni proces. Ova faza neće biti opisana u radu, ona je napravljena prije same implementacije igre.

- **Prototipiranje:** Razvojni tim stvara rane verzije igre kako bi testirao osnovne mehanike i koncept. Ova faza omogućava brzo identificiranje i ispravljanje potencijalnih problema te sprječavanje budućih, nije direktno opisana u nastavku za igru Trial of Cronos.

- **Implementacija:** U ovoj fazi razvojni tim koristi razne alate za izradu scena, postavljanje i programiranje objekata te različitih interakcija pomoću koda napisanog u odabranome programskom jeziku, te implementaciju zvuka i grafike. Ova faza razvoja računalne igre će biti detaljno opisana u nastavku rada za igru Trial of Cronos.

- **Testiranje i optimiziranje:** Igre se temeljito testiraju kako bi se osigurala stabilnost, ispravnost mehanika i uklanjanje grešaka. Mnogi alati za razvoj računalnih igara nude resurse za debugiranje koje pomažu u optimizaciji performansi igre. Ova faza razvoja se ukratko opiše u nastavu rada za igru Trial of Cronos.

- **Izvoz i distribucija:** Kada je igra završena, igra se izvozi na željene platforme. Razvojni tim može objaviti igru putem različitih distribucijskih kanala kao što su Steam, App Store, Google Play ili konzolne trgovine. Ova faza razvoja neće biti opisana u radu.

2. Opis igre Trial of Cronos

U ovoj igri igrač preuzima ulogu titana Cronosa koji se brani od grčkih bogova i vojnika kako bi preuzeo kontrolu Olimpa i postao ultimativno stvorenje. Kako bi preživio koristi moći koje je ukrao Zeusu za manipuliranje vremenom da bi strateški sakupljao resurse (kristale) koji mu trebaju koje će se koristiti za naoružanje koje će koristit tokom borbi.

2.1. Gameplay

Trial of Cronos je 2D top down shooter na principu survival gameplaya, što bi značilo da igrač gleda igru iz ptičje perspektive, shooter bi značilo da igrač koristi oružja kako bi prolazio porazio neprijatelje i prolazio razine, dok je jedan od ciljeva igre preživjeti (survival) boreći se protiv neprijatelja i skupljajući resurse.

Igrač započinje sa tutorialom (Lvl 0) koji počinje po noći, gdje se susreće s osnovnim mehanikama igre i borbom protiv nekoliko slabih protivnika. Nakon što uspješno završi početnu noć, dolazi dan, a igraču se objašnjava kako skupljati resurse tijekom dana čime završava tutorial. Postoji Help knjiga u kojoj su prikazani kristali koji se stvaraju tokom određenog vremena.

Nakon toga, igrač se suočava s prvim valom neprijatelja, označavajući početak prve razine. Igra sadrži 16 razina.

Igrač može mijenjati vrijeme između sunčanog, kišovitog i snježnog te ih kombinirati.

Kada padne noć igrača napadaju val neprijatelja koji napada iz svih smjerova, napreduje kroz razine, a svako peta razina sadrži drugi (dodatni) val gdje je glavni neprijatelj koji može biti Zeus, Poseidon ili Hades. Na posljednjoj razini, igrač se suočava sa svim glavnim neprijateljima.

2.2. Inspiracija za igru Trial of Cronos

Trial of Cronos uz originalnost da mijenjanje vremena i njihovo kombiniranje utječe na stvaranje kristala također ima i već neke dobro poznate elemente svijetu igara. Tematika grčke mitologije korištena je mnogim igrama ali odakle je došlo najviše inspiracije za nju je iz rogue-like dungeon crawler igre Hades čije su velike oznake proceduralno generiranje razina što znači da svaki nova sesija daje novi raspored prostorija i njihovih elemenata te neprijatelja, kada igrač umre gubi napredak razina što znači da počinje ispočetka ali dobije određene ili napretke u atributima, opremi ili slično. Igra se još i temelji na istraživanju tamnica i sakupljanju itema. Snažno se pokazuje tematike grčke mitologije prvo time što je glavni lik Zagreus sin boga podzemlja Hadesa koji je ujedno i glavni neprijatelj. Uz njih se još pojavljuju mnogi bogovi Olimpa kao što su: Zeus, Posejdon, Atena, Aresa i ostali. Na Slici 6. vidimo kako izgleda primjer gameplay-a igre Hades.

Slika 3. Primjer gameplay-a igre Hades

Jedna od većih inspiracija za igru je bullet hell dungeon crawler igra Enter the Gungeon, čiji je primjer gameplay-a prikazan na Slici 7. koja također ima top down perspektivu. Odavde dolazi i inspiracija za mehaniku Zeusa kada stoji na mjestu i baca munje sa vrha mape kao i za zadnju razinu gdje igrač ne treba ubiti neprijatelje već preživjeti određeno vrijeme kao što je u igri Enter the Gungeon fokus na izbjegavanja raznih projektila i brzo i pametno kretanje igrača po mapi. Jedan od ciljeva Trial of Cronos je da igra bude teška kako bi se igračeve vještine i pametno taktiziranje došlo što više do izražaja što daje dojam i da je igra Enter the Gungeon pokušala postići istu stvar.

Slika 4. Primjer gameplay-a Enter the Gungeon

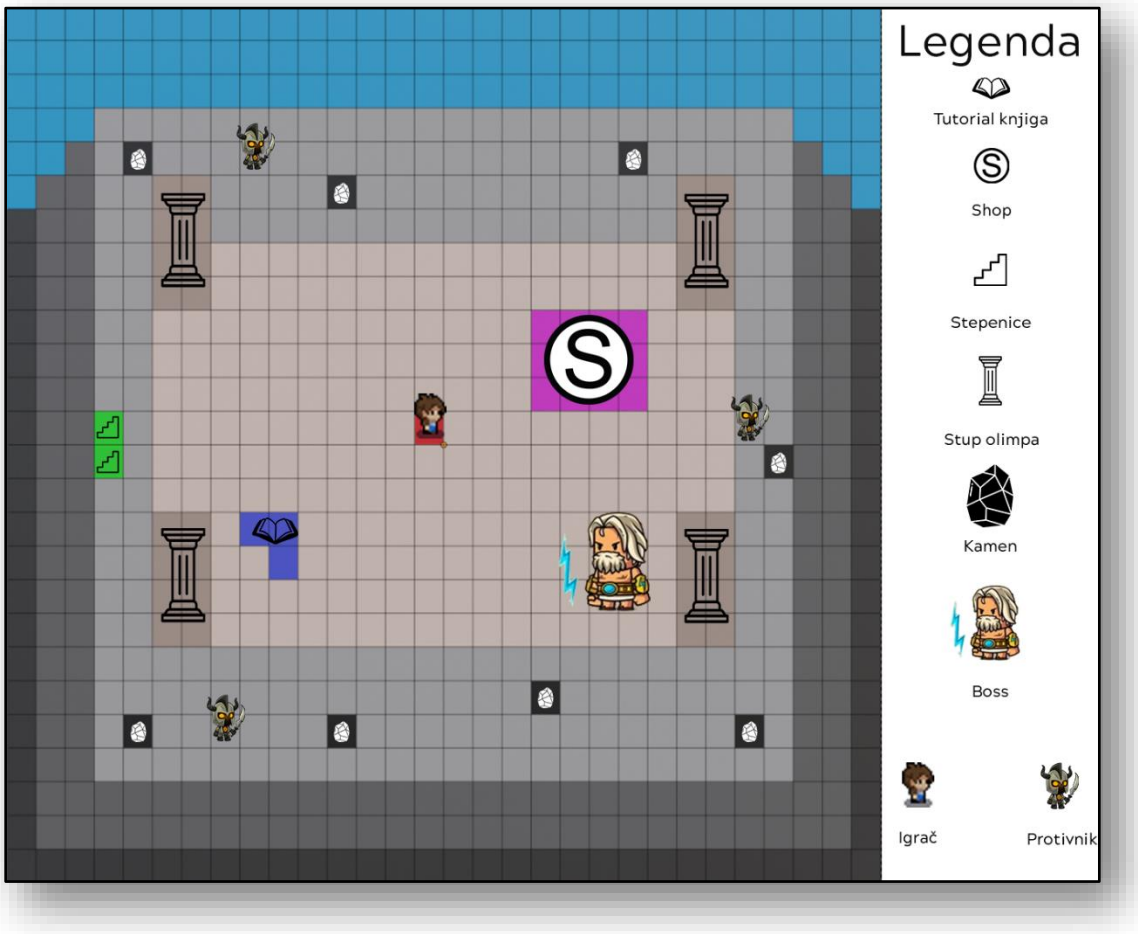
2.3. Mapa

Prostor u kojem se odvija igra se sastoji od 2 ravna dijela, načinjenih od kvadratnih polja, od kojih je prvi dio vrh planine Olimp koji sadrži hram te njegov okolni prostor. Veličina prvog dijela je 30x26 (780 polja) od kojih je 24x20 (480 polja) igrivo.

Na Slici 8. vidimo da se hram se nalazi u centru te se okolni prostor mijenja ambijentalno ovisno o vremenu, npr. Za snježno vrijeme bit će snijeg na stupovima hrama, za kišovito lokve vode, etc.

Hram služi više kao estetski element, igrač može pristupiti glavnom dućanu koji se nalazi u hramu, gdje može začarati i kupovati oružja i magije te poboljšavati svoj oklop. Dućan se koristi tako da kada igrač dođe blizu njega klikne npr. tipku E te mu se prikaže novi prozor dućana.

Rubovi prvog dijela mape su na rubovima vrha Olimpa (označeni crnom debelom linijom prikazanom na skici) koji su igraču nedostupni, te neprijatelji dolaze na mapu tako što se „penju“ na te iste rubove planine.

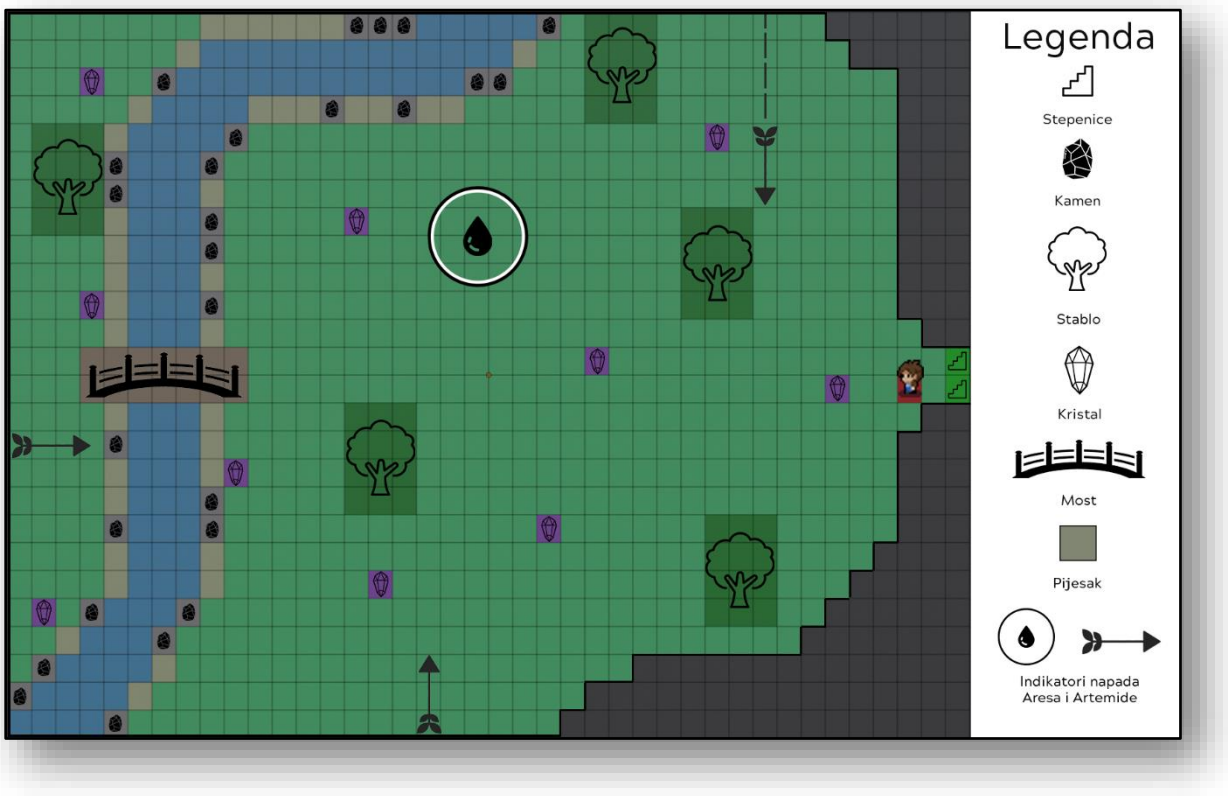


Slika 5. Skica gornje mape

Veličina drugog dijela mape je 40x26 (1040 polja) od kojih je 917 polja igrivo.

Drugom dijelu mape, kao što vidimo na Slici 9. se pristupa interakcijom sa stepenicama koji su na lijevom rubu planine Olimp. Taj dio sadrži rijeku, most, šumu i slično (čiji se dekor također mijenja ambijentalno, npr kada padne snijeg bude bijela i slično) te sa nje sakuplja kristale koji se stvaraju ovisno o kombinaciji vremena postavljenog nasumično za taj dan i vremena koje je prizvao na Olimpu prije nego što se spusti.

U rijeku igrač ne može upast te se preko nje može preći samo prelaskom mosta kako bi došao do kristala koji su se stvorili na drugoj strani što mu istovremeno otežava izbjegavati napade Artemide/Aresa jer ima manje mjesta.



Slika 6. Skica donje mape

3. Unity

Unity je jedan od najpopularnijih i najmoćnijih alata za razvoj računalnih igara. Omogućava stvaranje 2D i 3D igara za razne platforme uključujući računala, konzole, mobilne uređaje i virtualnu stvarnost.

Prva verzija je objavljena 2005. godine, a od tada je postao ključni alat u industriji igara s rasponom upotreba od velikih profesionalnih studija do manjih indie projekata.

Unity pruža niz značajki koje ga čine idealnim za razvoj mnoštva igara:

- **Multiplatformska podrška**: Unity omogućava razvoj za 20ak platformi koje su:
    - **Desktop**: Windows, Mac, Universal Windows Platform, Linux Standalone
    - **Mobilna**: IOS, Android
    - **Proširena stvarnost**: ARkit, ARcore, Microsoft HoloLens, Windows Mixed Reality, Magic Leap, Oculus, PlayStation VR2
    - **Konzole**: PS5, PS4, Xbox One, Xbox Series X|S, Nintendo Switch
- **Napredno grafičko 3D i 2D okruženje**: Unity podržava napredne grafičke tehnike, uključujući fiziku, sjenčanje, post-processing efekte i animaciju, omogućavajući visokokvalitetne vizualne prikaze.
- **Simulacija fizike:** Unity koristi NVIDIA PhysX engine za realistične simulacije fizike, što je ključno za igre koje zahtijevaju preciznu fiziku
- **C#:** Programiranje u Unityju temelji se na C# jeziku, koji je poznat po svojoj jednostavnosti i fleksibilnosti. Skripte se koriste za definiranje ponašanja objekata i interakcija unutar igre.
- **Asset Store:** Unity ima bogatu trgovinu resursa gdje razvojni timovi mogu kupiti ili preuzeti besplatne modele, teksture, zvukove, alate i skripte, čime se ubrzava proces razvoja.
- **Integrirani razvojni alat:** Unity Editor je integrirano razvojno okruženje koje omogućava vizualno uređivanje scena, manipulaciju objektima, testiranje i debugiranje igara unutar jedinstvenog sučelja.

Slika 7. Windows i Oculus logo

Slika 8. iOS i PS5 logo

Na Slikama 2. i 3. prikazan je logo za neke od platformi koje Unity podržava

Korištenje Unityja donosi brojne prednosti. Unity Personal verzija daje besplatan pristup mnogim potrebnim alatima za razvoj igara. Tokom razvoja igre Trial of Cronos koristila ova verzija Unitya koja je omogućila da se dobije gotov proizvod bez financijskih poteškoća. Unity također nudi i napredniju opciju Unity Pro koju koriste profesionalni timovi kojima je mnogo važnija kolaboracija među developerima koji rade na igri. Jedna od velikih prednosti bi bila zajednica i podrška pri izradi projekta. Unity ima jako veliko zajednicu gdje je na skoro svako pitanje moguće pronaći odgovor jer je neko prije imao isti problem, ali sve i da ne odgovor već nije objavljen, dobit će se jako brzo. Kroz razvoj igre Trial of Cronos korišteni su razni forumi kada bi bilo nedoumica ili ako samog Unity editora ili oko C# koda.

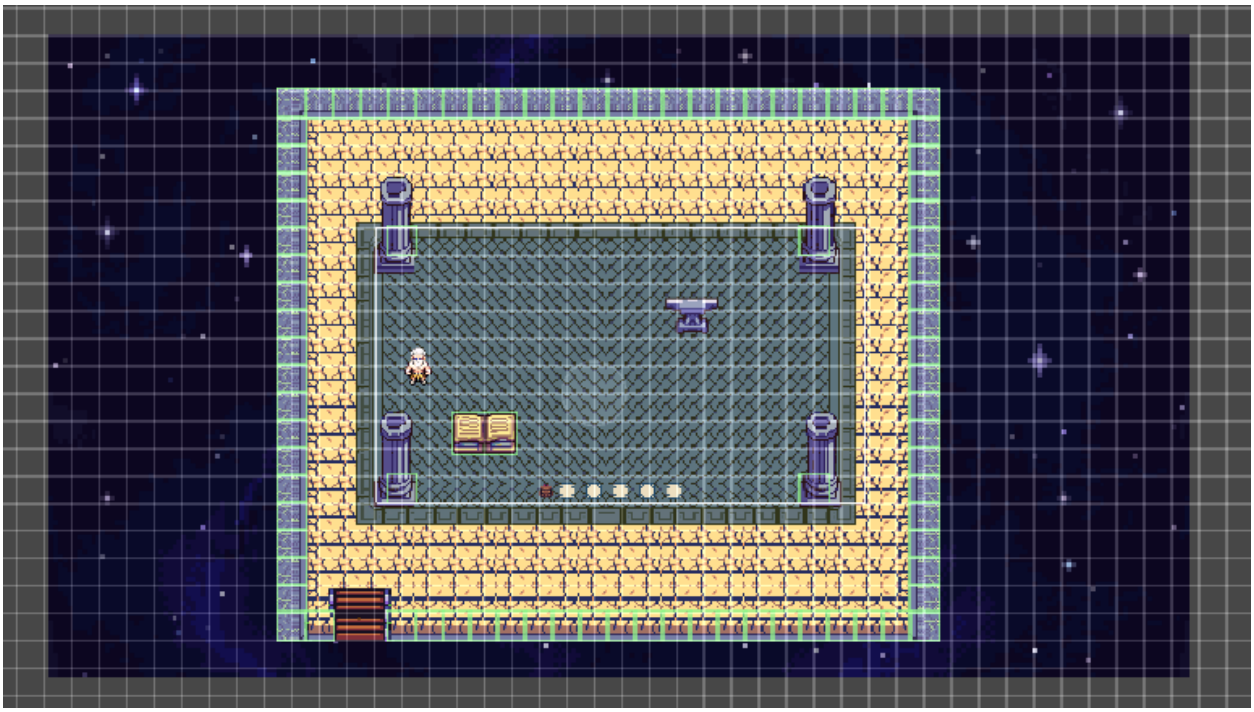
Još jedna velika prednost je lakoća i brzina testiranja igre. Unity nudi brzo i efikasno postavljanje objekata na scenu kao i testiranje istih kod implementacije Trial of Cronos promjena stvari kao što su kako se kreću neprijatelji i koliko brzo strijela treba ići je išlo jako lagano i brzo jer se sve varijable, ako su u kodu postavljene na public ili [SerializeField] (ovaj način se koristi kada varijabli želimo pristupi iz Unity editora ali ne želimo da bude public tj. da bude dostupna drugim skriptama), mogu mijenjati u Unity editoru.

Proceduralno generiranje sadržaja je još jedna od mogućnosti koja se može iskoristiti razvojem igre u Unity-u koje se često koristi za stvaranje svijetova i mapa što ubrzava i olakšava cijeli proces. Proceduralno generiranje znači da se dinamički kreira neki element predodređen nekim parametrima i pravilima. Većinom se koristi kod izrade elemenata, na primjer ako je potrebno 20 vrsta stabla napravit će se samo jedno te će se po tome jednome generirati automatski ostalih 19 s nekim promjenama (veličina, pozicija, struktura). U igri Trial of Cronos proceduralno generiranje se koristilo na sličan način, već su napravljeni prije svi kristali ali se svaki puta kada igrač dođe na donju mapu nasumično generiraju na različita mjesta.

4. Razvoj igre Trial of Cronos

4.5.1 Kreiranje Mape

Unity ima jako dobar „Tilemap sistem“ gdje se jako efikasno i precizno mogu izrađivati 2D mape i razine bez velikih poteškoća. Pomoću „Tile Pallete-a“ pojednostavljeno je korištenje tile-ova (tile je jedno polje na tilemapi) gdje se može odabrati jedan tile ili njih više koji trenutno trebaju i samo se postavljaju na tilemapu koja se prethodno postavila u scenu. Razlika između mape i tilemape je u tome što je mapa generalni naziv za područje igre koja sadrži razne objekte ili po kojoj se objekti kreću, a pomoću tilemape se zapravo izrađuje mapa polje po polje te se mogu izrađivati i elementi na mapi pogotovo u pixel art igrama gdje je točno jasno kakvo je koje polje. Mapa se također može izraditi pomoću više tilemap-a ako se sa određenim poljima treba odnositi na drugi način ili staviti određene komponente koje bi smetale ostatku polja u tilemapi. Grid je komponenta koja definira strukturu mreže za 2D prostor što znači da postavlja okvir za organizaciju polja te grid sam po sebi nema sloj na kojega se postavlja, a zatim se pomoću jedne ili više tilemapa postavljaju tile-ovi unutar tog prostora.



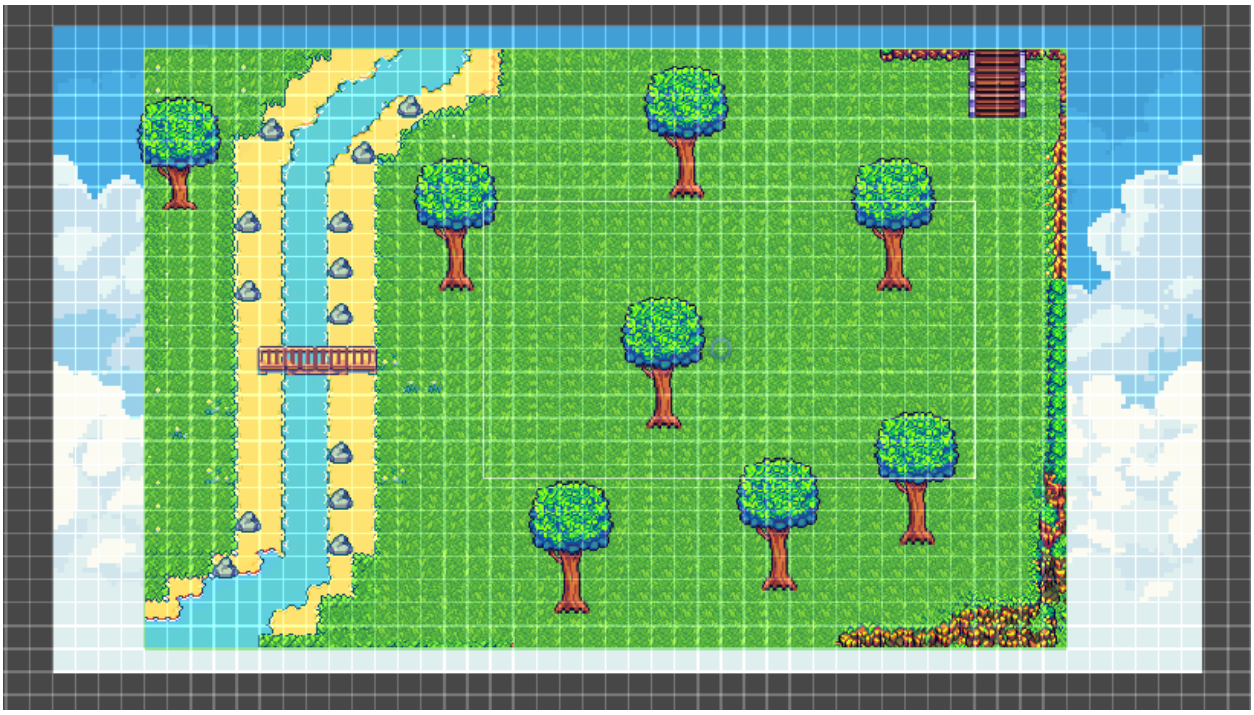
Slika 9. Gornja mapa

Na Slici 10. vidimo gornju mapu, tj. mapu gdje se igrač bori protiv neprijatelja i prelazi razine implementirana je tako što je stavljen objekt „Grid“ u kojeg su postavljene dvije Tilemape (jedna korištena za tile-ove koji trebaju collision a druga za one koje koji ne trebaju, objašnjeno u tekstu ispod) te je postavljena slika u objekt „Canvas“ koja predstavlja noć. Kako se koristi ista mapa za sve razine napravljen je prefab grida (koji sadrži dvije tilemape) čime se olakšava stvaranje scena za nove razine tako da umjesto da se ispočetka postavlja grid i tilemape može se samo postaviti prefab cijeloga grida na svaku scenu gdje je potrebno. Još jedna prednost prefab-a je to što ako treba promijeniti određeni tile ili dodati neku komponentu na tilemapu umjesto da se na svaku scenu svake razine posebno mijenja ili dodaje što treba, može se jednostavno na prefabu napraviti promjena nakon čega se klikne na „override“ i promjene se spreme na svim scenama gdje je korišten prefab.

Prva tilemapa sadrži sve tile-ove preko kojih igrač i protivnici mogu prelaziti (oni u koje se neće zabijati). Order in layer tilemape je postavljen na broj manji nego igrača i protivnika kako bi se ti objekti mogli vidjeti iz razloga što ako su na sloju većem od njih onda će prekrivati igrača jer su „bliže kameri“. Igra je 2D a ne 3D pa ne možemo samo pomaknuti jedan objekt ispred drugoga zbog čega se koriste slojevi kako bi se dobio dojam da je tilemapa iza igrača po kojoj se on kreće.

Druga tilemapa sadrži sve tile-ove sa collisionom (što znači da su u drugoj tilemapi sva polja tj. tile-ovi u koje je predviđeno da se igrač zabije kao što su stupovi) u koje će se zabijati svi objekti koji također imaju collision u ovome slučaju to su: igrač, neprijatelji i određeni projektili. To se postiglo tako što je postavljen Tilemap Collider 2D koji generira zasebni collider za svaki tile u toj tilemapi, nakon čega je postavljen i Composite Collider 2D kako bi se smanjio broj collidera kojeg Unity treba procesirati, svi spojeni u jedan što dovodi do bolje otpimizacije.

Ovdje se nalaze: stupovi, knjiga te krajnji tile-ovi mape.



Slika 10. Donja mapa

Na slici 11. vidimo donju mapu, tj. mapu „Dan“ gdje igrač dolazi preko dana i skuplja kristale koje koristi za kupovinu opreme kako bi se spremio za noć i borbu protiv neprijatelja.

Također napravljena pomoću i grida ali u ovome slučaju postoje tri tilemape i postavljena slika za pozadinu kako bi se dobio bolji dojam dana.

Prva tilemapa kao i prije služi za tile-ove koji nemaju collision tako da se njima igrač može slobodno kretati. Order in layer je postavljen na broj manji od igrača kako bi se objekt mogao vidjeti.

Druga tilemapa sadrži sve tile-ove na kojima treba biti collision te kao i prije postavljen je Tilemap Collider 2D i Composite Collider 2D

Ovdje se nalaze tile-ovi za: kamenja, donji dio deblo stabla, i krajnji tile-ovi mape. Treća tilemapa sadrži samo krošnje stabla koji su postavljen na Order in layer broj veći nego što ima igrač kako bi se dobio dojam da igrač prolazi iza krošnje a ne preko nje, te na njima nema collisiona kao na donjem dijelu debla stabla.

4.5.2. Implementacija glavnog lika Cronosa

Cronos je glavni objekt igre kojim upravlja igrač kako bi porazio valove neprijatelja i prešao na druge razine.

Osnovne komponente koje ima su:

- RigidBody 2D koja objektu daje mogućnost interakcije s drugim objektima te ako je postavljen na „Dynamic“ što u ovome slučaju je na kretanje mogu utjecati i sile kao što su: „Linear drag“, „Angular drag“ i „Gravity scale“

- Box Collider 2D koji objektu omogućava koliziju s drugim objektima koji također imaju ovu komponentu te je veličina postavljena tako da točno odgovara spriteu objekta

Skripte:

1. Implementacija kretanja igrača

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

public class Movement : MonoBehaviour
{
    [SerializeField] private float speed;
    private float horizontalAxis;
    private float verticalAxis;
    [SerializeField] LayerMask groundLayerMask;
    [SerializeField] Animator animator;
    private Rigidbody2D body;
    private Collider2D boxCollider2d;
    public WeaponParent weaponParent;



    void Start()
    {
        body = GetComponent<Rigidbody2D>();
        boxCollider2d = GetComponent<Collider2D>();
    }

    void Update()
    {
        horizontalAxis = Input.GetAxis("Horizontal");
        verticalAxis = Input.GetAxis("Vertical");

        if (horizontalAxis < 0) transform.localRotation = Quaternion.Euler(0, 180, 0);
        if (horizontalAxis > 0) transform.localRotation = Quaternion.Euler(0, 0, 0);

        animator.SetBool("running", Mathf.Abs(horizontalAxis) > 0.01f || Mathf.Abs(verticalAxis) > 0.01f);


    }

    void FixedUpdate()
    {
        body.velocity = new Vector2(horizontalAxis * speed, verticalAxis * speed);

    }
```

Slika 11. kod skripte za kretanje

Na Slici 12. vidimo [SerializeField] private float speed; varijabla je postavljena privatna što znači da se koristi samo unutar ove skripte ali SerializeField daje mogućnost da se vrijednost mijenja iz unity ediotra bez da se mijenja direktno u kodu.

Funckija Start() poziva se samo jednom i to kada scena na kojoj je ovaj objekt postane aktivna. Uzima se RigidBody2D komponenta objekta kao i Collider komponenta.

Funkcija Update() poziva se jednom po frameu.
U varijable horizontalAxis i verticalAxis se stalno sprema pozicija igrača po x osi (horizontalno) i y osi(vertikalno) te se provjerava ukoliko se igrač kreće da se pomoću animatora promjeni vrijednost bool varijable „running“ što zatim pokreće animaciju trčanja.

Funkcija FixedUpdate() za razliku od funkcije Update() ne ovisi o frameovima nego je postavljena da se poziva nakon određenog vremena što je otprilike 50 puta po sekundi. Bolje ju je koristiti nego funkciju Update() u slučaju kada se radi o upravljanju fizikom i kretanjima objekta kako bi se što točnije prikazale fizičke simulacije.
U ovome slučaju linija koda koja se nalazi u funkciji služi za brzinu kretanja lika, tj. stvara novi vektor ovisno o tome u kojem se smjeru igrač kreće i kolika je postavljena brzina.

2. Implementacija igračevog zdravlja

```
public void TakeDamage(int damageAmount)
{
    audioManager.PlaySFX(audioManager.damageTaken);
    currentHealth -= damageAmount;
    if (currentHealth <= 0)
    {
        Die();
    }
    else
    {
        OnHealthChanged.Invoke(currentHealth);
    }
}
```

Slika 12. Funkcija TakeDamage()

Na Slici 13. vidimo funkciju pomoću koje igrač prima štetu koja prima varijablu koliko zdravlja igraču treba skinuti. Poziva se u svim skriptama gdje protivnici rade štetu igraču npr. kada se zabiju u igrača ili kada ga pogode projektilom.

Postavljen je audio svaki put kada se igraču smanji zdravlje.

Kada se igračev „currentHealth“ smanji na 0 ili niže poziva se funkcija Die() gdje se poziva „EndGameScene“ scena te se prestane brojat vrijeme koliko je igrač bio živ.

OnHealthChanged je UnityEvent na koje se druge funkcije mogu pretplatiti te svaki put kada se pozove OnHealthChanged pozivaju se i sve druge funkcije koje su pretplaćene na ovaj događaj. U ovome slučaju pretplaćena je funkcija UpdateHealthUI() u skripti PlayerHealthUI koja smanjuje broj srca igrača kada primi štetu.

Slika 13. Puna srca

Slika 14. Prazno srce

Na Slici 14. vidimo srca kada su puna Na Slici 15. vidimo igrač primi štetu (-1)

Funkcija samo zamijeni sprite punog srca sa sprite-om praznoga i tako se igraču prikaže da je primio štetu.

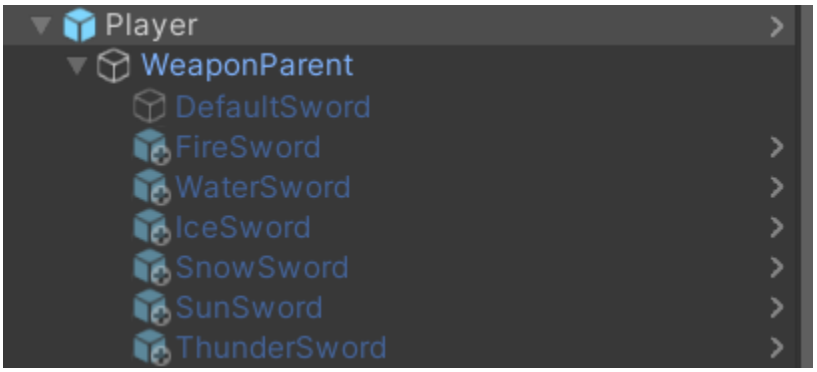
3. Implementacija zamjene oružja

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class WeaponSwitch : MonoBehaviour
6   {
7       public GameObject Sword;
8       public GameObject Bow;
9
10
11      private void Update()
12      {
13          if(Input.GetKeyDown(KeyCode.Alpha1))
14          {
15              Sword.SetActive(true);
16              Bow.SetActive(false);
17          }
18
19          if(Input.GetKeyDown(KeyCode.Alpha2))
20          {
21              Sword.SetActive(false);
22              Bow.SetActive(true);
23          }
24      }
25
26  }
27
```

Slika 15. Implementacija zamjene oružja

Na Slici 16. vidimo skriptu koja uzima dva objekta jedan za mač i drugi za luk te ako je pritisnuta tipka 1 aktivan je mač a deaktivira se luk, a ako je pritisnuta tipka 2 aktivira se luk a deaktivira mač.

Ova skripta nam je važna iz razloga što ne kontrolira jedan mač ili jedan luk nego više njih. Na objekt mača u Unity editoru postavi se objekt „WeaponParent“ pod kojim se nalaze svi mačevi u igri, kojeg vidimo na Slici 17.



Slika 16. Player u hijerarhiji

Na početku se aktivira objekt „DefaultSword“ a poslije u skriptama za inventory se aktivira mač ovisno o tome koji je od njih postavljen na prvome mjestu u inventoryu.

Slika 17. Prikaz InventoryManager skripte u unity editoru

Na slici 18. vidimo da na objektu „InventoryManager“ postoje „Sword names“ i polja za svaki objekt svakog mača. U njegovoj skripti napravljen je dictionary gdje svaki mač ima

ime i Game Object toga mača. Funkcija Update() poziva funkciju CheckFirstSlotForSword() koja zatim poziva funkciju ActivateSwordObject() koja aktivira objekt mača koji je trenutno na tome mjestu što vidimo na Slici 19.

```
void CheckFirstSlotForSword()
{
    Item firstSlotItem = GetItemInSlot(30);
    if (firstSlotItem != null && swordToObjectMap.ContainsKey(firstSlotItem.name))
    {
        Debug.Log(firstSlotItem.name + " detected in the first slot.");
        ActivateSwordObject(firstSlotItem.name);
    }
    else
    {
        Debug.Log("No recognized sword detected in the first slot.");
        DeactivateAllSwordObjects();
    }
}
```

Slika 18. Funkcija CheckFirstSlotForSword()

4. Implementacija dalekometnog napada

Ova se skripta koristi kada igrač napada lukom i strijelom, potreban je firepoint odakle će se instancirati strijele koje su napravljene kao prefab te imaju skriptu za projektil kojeg igrač ispaljuje koja strijeli daje štetu i provjerava kada dođe do kolizije sa objektom kojima ima tag „Enemy“ što vidimo na Slici 20.

```
void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.collider.CompareTag("Enemy"))
    {
        EnemyHealth enemyHealth = collision.collider.GetComponent<EnemyHealth>();
        if (enemyHealth != null)
        {
            enemyHealth.TakeDamage(damage);
            Destroy(gameObject);
        }
    }
}
```

Slika 19. OnCollisionEnter2D()

Na Slici 21. funkcija Attack() u skripti za dalekometni napad igrača služi za napad gdje se prati koliko je dugo igrač pritisnuo lijevi klik miša te se računa koliko će brzo strijela

letjeti. Ukoliko strijela nije pogodila objekt s tagom „Enemy“, uništava se nakon tri sekunde.

```
void Attack()
{
    float chargePercent = chargeTime / maxChargeTime;
    float projectileSpeed = Mathf.Lerp(minProjectileSpeed, maxProjectileSpeed, chargePercent);

    projectile = Instantiate(projectilePrefab, firePoint.position, firePoint.rotation);

    Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();

    rb.velocity = firePoint.right * projectileSpeed;
    Destroy(projectile, 3.0f);
}
```

Slika 20. Funkcija Attack()

5. Implementacija moći za heal

Ova skripta koristi kako bi si igrač svaki određeni broj sekundi mogao napuniti dio zdravlja tokom borbe.

```
void Update()
{
    if(Input.GetKeyDown(KeyCode.Alpha3))
    {
        if (Time.time - lastActivationTime >= cd)
        {
            if (playerhealth != null)
            {
                if(playerhealth.currentHealth + HealAmount <= playerhealth.maxHealth)
                {
                    audioManager.PlaySFX(audioManager.drinkPotion);
                    playerhealth.TakeDamage(-HealAmount);
                    GameObject Healspell = Instantiate(Healprefab, transform.position + offset, Quaternion.identity);

                    Healspell.transform.SetParent(transform);

                    Destroy(Healspell, 0.4f);
                    lastActivationTime = Time.time;
                }
            }
        }
    }

}
```

Slika 21. Funkija Update(), HealSpell

Na slici 22. vidimo da se većina skripte se nalazi u funkciju Update() gdje se provjerava je li pritisnuta tipka 3. Ukoliko je, provjerava se je li prošlo dovoljno vremena od zadnjeg puta kada je igrač aktivirao Heal spell. Radi tako što se pozove funkcija TakeDamge() koja inače smanjuje zdravlje igraču ali je poslana negativna vrijednost tako da mu zapravo puni zdravlje. Također se instancira prefab zelenog plusa iznad igrača kako bi i vidio da se spell aktivirao.

Postavljeno je da se puni za 1 zdravlje svako 7 sekundi.

4.5.3. Neprijatelji

Svi neprijatelji imaju istu skriptu za kretanje koja radi tako da u Start() funkciji odmah pronađe objekt na sceni koji ima tag „Player“ što vidimo na Slici 23.

```
private void Start()
{
    body = GetComponent<Rigidbody2D>();
    GameObject player = GameObject.FindGameObjectWithTag("Player");
    if (player != null)
    {
        playerTransform = player.transform;
    }
    else
    {
        Debug.LogError("Player not found! Ensure there is a GameObject tagged 'Player' in the scene.");
    }
}
```

Slika 22. Fukcija Start(), Neprijtelji

Zatim se u funkciji Update() što vidimo na Slici 24. računa udaljenost između objekta s ovom skriptom i objekta s tagom „Player“ te se enemy kreće prema neprijatelju. Postavljen je i Attack range svakome neprijatelju kako bi oni koji ispaljuju projektili mogli gađati sa veće udaljenosti što se koristi u kod kretanja gdje je namješteno da ako dođu u range napada da se više ne kreću prema objektu s tagom „Player“. Osigurano je da su neprijatelji uvijek okrenuti u pravome smjeru, tj. smjeru igrača tako što se gleda je li x koordinata varijable direction veća ili manja od nule. Ukoliko je koordinata x manja od nule neprijatelja se okreće za 180 stupnjeva kako bi mu sprite gledao u pravome smjeru, a ako je veće od nule rotacija mu se ne mijenja.

Svi neprijatelji imaju napravljenu animaciju za trčanje koja se aktivira na isti način kao i kod igrača (Cronosa). U slučaju kod neprijatelja animacija se gasi postavljanjem vrijednosti „running“ na false kada je neprijatelj došao dovoljno blizu igrača da ga može napadati gdje više nema potrebe za animacijom trčanja nego se pali animacija napada što je animirano kod oružja neprijatelja.

```
private void Update()
{
    if (playerTransform != null)
    {
        float distanceToPlayer = Vector2.Distance(transform.position, playerTransform.position);

        if (distanceToPlayer > attackRange)
        {
            Vector2 direction = (playerTransform.position - transform.position).normalized;
            body.velocity = direction * speed;

            if (direction.x < 0)
                transform.localRotation = Quaternion.Euler(0, 180, 0);
            else if (direction.x > 0)
                transform.localRotation = Quaternion.Euler(0, 0, 0);

            animator.SetBool("running", true);
        }
        else
        {
            body.velocity = Vector2.zero;
            animator.SetBool("running", false);
        }
    }
}
```

Slika 23. Funkcija Update(), Neprijatelji

Svi neprijatelji napravljeni su kao prefab-ovi kako bi se mogli koristiti objekti „EnemySpawner“ koji koriste za kontroliranje razina i koliko kojih neprijatelja po razini želimo.

Skripta za stvaranje neprijatelja radi tako što se prvo inicijalizira static lista u koju se spremaju svi neprijatelji kada krene razina. Bitno je da je lista static zbog toga što onda postoji samo jedna lista u koju se stavljaju svi neprijatelji, da lista nije static onda bi svaki spawner imao svoju listu što ovdje nije cilj.

```
private void HandleEnemyDeath(GameObject enemy)
{
    Debug.Log("Enemy died: "+ enemy.name);

    if (allActiveEnemies.Contains(enemy))
    {
        allActiveEnemies.Remove(enemy);
        Debug.Log(allActiveEnemies.Count);

    }

    if (allActiveEnemies.Count == 0)
    {
        AllEnemiesDestroyed();
    }
}
```

Slika 24. Funkcija HandleEnemyDeath()

Na Slici 25. vidimo da kada neprijatelj umre makne ga se iz static liste aktivnih neprijatelja te se provjerava postoji li jos neprijatelja u toj listi, u slučaju da ne postoji poziva se funkcija AllEnemisDestroyed().

```
private void AllEnemiesDestroyed()
{
    Debug.Log("All enemies are destroyed!");
    allActiveEnemies.Clear();
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);


}
```

Slika 25. Funkcija AllEnemiesDestroyed()

Na Slici 26. vidimo funkciju se koristi za prelazak između razina, kada su svi neprijatelji mrtvi pozove se iduća scena, a buildano je tako da su razine sve scene jedna za drugom tako da automatski ide na sljedeću razinu.

Za duplu provjeru jesu li svi neprijatelji mrtvi također postoji još jedna funkcija koja se poziva u funkciji Update()

```
private void Update()
{
    if (isSpawningActive)
    {
        CheckForRemainingEnemies();
    }
}
private void CheckForRemainingEnemies()
{
    GameObject[] remainingEnemies = GameObject.FindGameObjectsWithTag("Enemy");

    if (remainingEnemies.Length == 0)
    {
        AllEnemiesDestroyed();
    }
}
```

Slika 26. Funkcije Update(), CheckFroRemainingEnemies(); EnemySpawner

Na Slici 27. vidi se da funkcija CheckForRemainingEnemies() traži na sceni objekte s tagom „Enemy“ te ukoliko ih nema pozove se funkcija AllEnemiesDestroyed().

3.5.3.1. Melee/Tank/Range

Ove tri osnovne vrste neprijatelja sve dijele istu skriptu za kretanje prije navedenu te Range neprijatelj također koristi skriptu za dalekometni napad koja se razlikuje od skripte za igračev dalekometni napad u tome što kada neprijatelj ispali projektil napravljeno je tako da prati objekt s tagom „Player“.

```
public GameObject projectilePrefab;
public List<Transform> firePoints;
public float attackCooldown = 1.0f;
public float projectileSpeed = 10f;
public float disableTime = 10f;
private float timeUntilNextAttack = 0.0f;
public float TimeUntilDestroy = 3.0f;
private GameObject projectile;
```

Slika 27. Popis varijabli korištenih za neprijatelje

Na Slici 28. vidimo koje sve varijabla skripta koristi, prvo se uzima prefab za projekti, zatim lista u slučaju da neprijatelj ima više fire pointa što se koristi poslije za neke Boss-eve. AttackCooldown što određuje svako koliko će neprijatelj ispaliti projektil, brzina projektila, disableTime je varijabla koju također koristi određeni boss u slučaju da treba ugasiti određenu mehaniku. TimeUntiDestroy isto kao i prije ukoliko ne pogodi metu da se nakon nekog vremena projektil uništi i varijabla projectile u koju se sprema novo nastali projektil.

U funkciji Update() se poziva funkcija Attack() koju vidimo na Slici 29.

```
void Attack()
{
    GameObject player = GameObject.FindGameObjectWithTag("Player");
    if (player != null)
    {
        if (firePoints.Count > 0)
        {
            Transform selectedFirePoint = firePoints[Random.Range(0, firePoints.Count)];
            Vector2 direction = (player.transform.position - selectedFirePoint.position).normalized;

            projectile = Instantiate(projectilePrefab, selectedFirePoint.position, selectedFirePoint.rotation);

            Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();

            rb.velocity = direction * projectileSpeed;
            Destroy(projectile, TimeUntilDestroy);
        }
        else
        {
            Debug.LogWarning("No fire points available. Please assign fire points in the inspector.");
        }
    }
    else
    {
        Debug.LogWarning("Player not found. Make sure the Player object is tagged correctly.");
    }
}
```

Slika 28. Funkcija Attack(), Neprijatelji

Prvo pronalazi objekt s tagom „Player“, ukoliko ga pronađe i ako u listi postoji više od 0 fire pointa prvo se uzima nasumično jedan od fire pointa kako se ne bi uvijek ponavljao isti uzorak napada ukoliko neprijatelj ima više fire pointa.
Nakon toga se gledaju pozicija igrača i pozicija odabranog fire pointa te se projektil instancira na temelju svih prije određenih varijabli i prefaba koji je postavljen u Unity editoru.

Sva tri osnovna neprijatelja imaju skriptu za sudar s igračem koja se služi tome da ako se igrač zabije u neprijatelja da gubi zdravlje što vidimo na Slici 30.

```
if (collision.collider.CompareTag("Player"))
{
    PlayerHealth playerHealth = collision.collider.GetComponent<PlayerHealth>();
    if (playerHealth != null)
    {
        playerHealth.TakeDamage(damageAmount);
    }
    else
    {
        Debug.LogError("PlayerHealth component not found on Player.");
    }
}
```

Slika 29. Logika kolizije s igračem

Ukoliko je objekt sa skriptom ušao u collision s objektom koji ima tag „Player“ te ukoliko zdravlje nije nula, poziva se funkcija TakeDamage.

Za sva tri neprijatelja postoji tri vrste za akt 1, akt 2 i akt 3 koji se mijenjaju svako 5 razina:

- **Akt 1 (1 – 5):**
  Melee (100 Život, 1 Napad, 2 brzina), Range (60 Život, 1 Napad, 3 brzina), Tank (180 Hp, 2 Napad, 1 brzina)
- **Akt 2 (6 – 10)**
  Melee (Život Hp, 1 Napad, 2 brzina), Range (160 Život, 1 Napad, 4 brzina), Tank (400 Život, 2 Attack, 1.5 brzina)
- **Akt 3 (11 – 15)**
  Melee (400 Život, 1 Napd, 2.5 brzina), Range (320 Život, 1 Napad, 5 brzina), Tank (700 Hp, 2 Napad, 1.5 brzina)

4.5.3.2. Posejdon

Posejdon je prvi Boss igre napravljen da ne bude previše zahtjevan već da ukaže igraču na to da te razine neće biti iste kao i prošle.

Prva verzija Posejdona koristi sve iste skripte kao i osnovni neprijatelji samo što njegov projektil kojeg ispaljuje ima skriptu pomoću koje se odbija od svega što može aktivirati istTrigger kao što vidimo na Slici 31.

```
void OnTriggerEnter2D(Collider2D other)
{
    PlayerHealth playerHealth = other.GetComponent<PlayerHealth>();

    if (playerHealth != null)
    {
        playerHealth.TakeDamage(damageAmount);
        Destroy(gameObject);
    }
    else if (other.tag != "Player" && other.isTrigger == false)
    {
        BounceBack();
    }
}

void BounceBack()
{
    if (rb != null)
    {
        rb.velocity = -rb.velocity * bounceSpeedMultiplier;
    }
    else
    {
        Debug.LogWarning("Rigidbody2D not found on projectile.");
    }
}
```

Slika 30. Funkcija BounceBack()

Radi tako što provjerava je li se zabio u objekt s tagom „Player“, ako je onda igrač prima određenu štetu i projektil se uništava, a ako nije poziva se funkcija BounceBack() koja mijenja predznak brzini što ga zapravo ispaljuje u drugu stranu i množi sa brojem ukoliko se želi da projektil postane brži kada se odbije.

Postoji i druga vrsta, to je Posejdon koji se pojavljuje na razini 16 kada se igrač bori protiv sva 3 boga. Funkcija drugog Posejdona je da stoji na mjestu i gađa igrača valovima tako da mu trebaju samo dvije skripte, jedna za napad i jedna skripta da nakon određenog vremena nestane s obzirom da na zadnjoj razini igrač ne može ubiti bogove nego treba preživjeti određeno vrijeme kako bi pobijedio.

Koristi se ista skripta za RangeAttack neprijatelja te se koristi skripta Zeus2Off koja samo uništi postavljen GameObject nakon određenoga vremena koju vidimo na Slici 32.

```
public class Zeus2Off : MonoBehaviour
{
    public float timeTillRemoval = 10.0f;
    public GameObject Zeus3;
    public Vector3 offset = new Vector3(-5, 0, 0);

    void Start()
    {
        StartCoroutine(Zeus2Death());
    }


    IEnumerator Zeus2Death()
    {
        yield return new WaitForSeconds(timeTillRemoval);
        if (Zeus3!=null)
        {
            Instantiate(Zeus3, transform.position + offset, transform.rotation);
        }
        Destroy(gameObject);

    }
}
```

Slika 31. Funkcija Zeus2off()

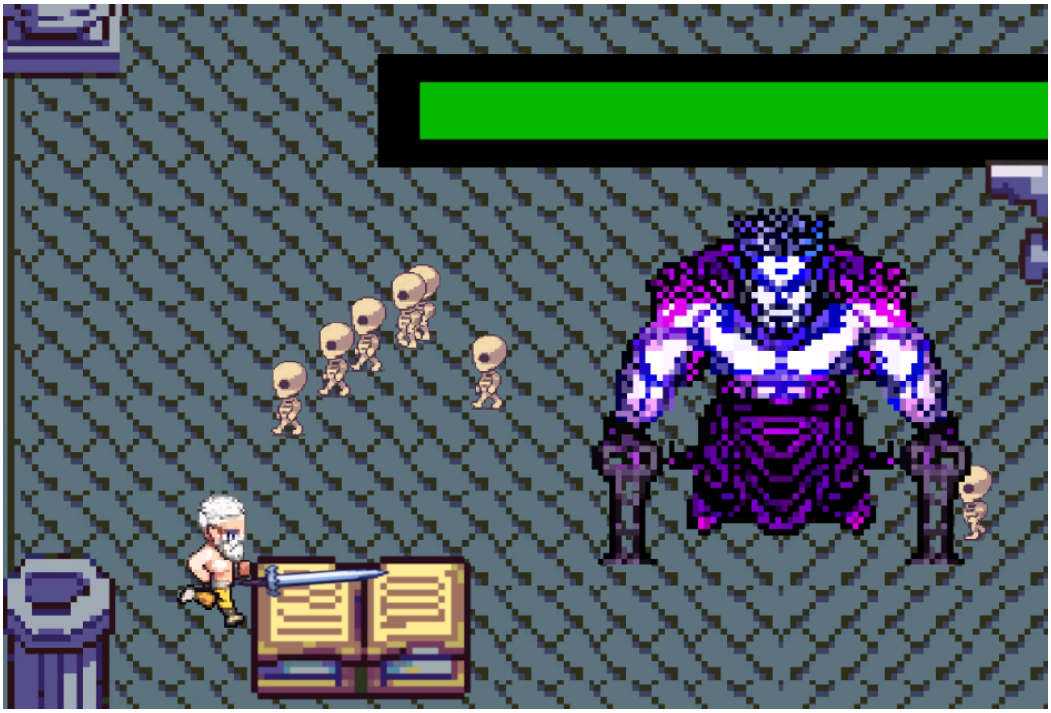
4.5.3.3. Hades

Hades je drugi Boss koji dolazi na razini 10 gdje igrač nailazi na nove mehanike.
Postoje tri verzije Hadesa, prva verzija napada igrača tako što stoji na mjestu ali po mapi stvara skeletone koji se bore za njega. Lako se ubijaju ali su brzi i stvara ih se mnogo sa puno mjesta kako igrač ne bi mogao znati s koje strane će mu dolaziti što se može vidjeti na Slici 33.

Za njihovo stvaranje koristi se već spomenuta skripta RangeAttack ali ovaj put fire points su raspoređeni po mapi. Stvoreni skeletoni imaju skripta EnemyHealth i EnemyCollision te brzina 3 i život 10 kako bi ih se jednoga lako moglo ubiti ali ideja je da ih bude previše tako da igrač mora dobro odigrati kako bi preživio.

Druga verzija Hadesa je dolazi kada prvoj verziji igrač skine pola zdravlja, prikazuje se animacija kako razbija lance sa sebe te prestane stvarati skeletone ali krene gađati igrača lancima koje je razbio. Skripte sve spomenute već RangeAttack, EnemyHealth, EnemyCollision i Movement.

Treća verzija Hadesa je ista kao i Posejdeonova druga, također na 16. razini stoji na mjestu i stvara skeletone koji hvataju igrača dok on pokušava izbjegavati projektile.

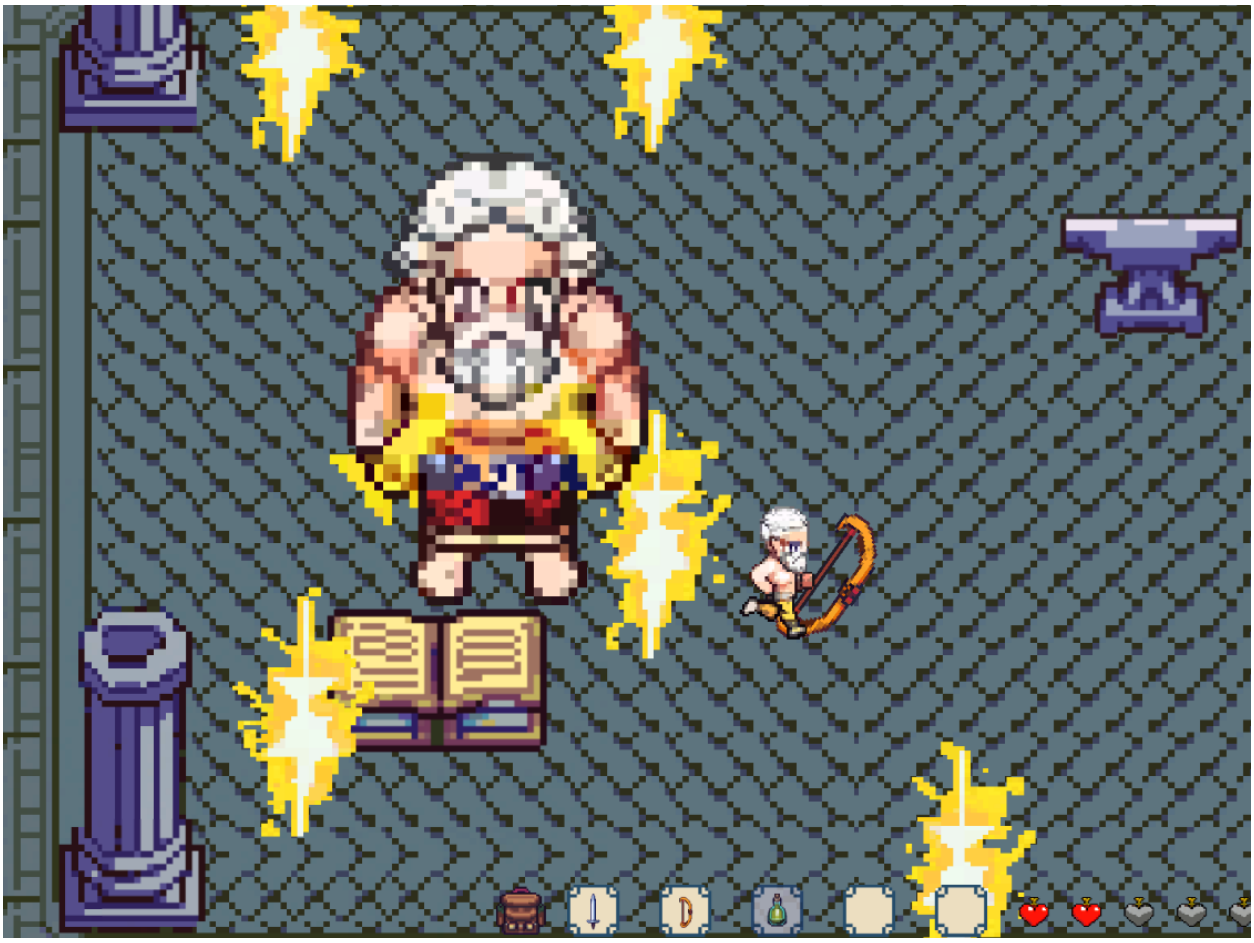


Slika 32. Primjer gameplaya prve faze Hadesa

4.5.3.4. Zeus

Zeus je treći i zadnji boss igre koji se pojavljuje na razini 15, ima dvije verzije u prvoj koristi munje kako bi gađao igrača te koristi već sve poznate skripte za RangeAttack, Movement i EnemyHealth.

Druga verzija Zeusa je kada se igrač prvi put susreće s mehanikom da bog stoji na mjesto i ne može ga se ubit dok ispaljuje nasumično projektile, ova faza dolazi odmah nakon što igrač ubije prvu verziju Zeusa i pobjeđuje drugu fazu tako što preživi 30 sekundi koja se može vidjeti na Slici 34.



Slika 33. Primjer gameplaya druge faze Zeusa

Također dok ispaljuje munje prikazuje se mala animacija na gromovima u rukama kako nestaju i dolaze nazad kako bi se dobio dojam da zapravo on baca te munje.

Zadnja, 16. razina, što zapravo predstavlja Zeusovu treću fazu gdje je pozvao druga dva boga jer sam nije mogao poraziti igrača.
Hades stvara skeletona, Posejdon napada odbijajućim valovima i Zeus ispaljuje munje, igrač sve to treba preživjeti 30 sekundi kako bi prešao igru.

4.5.4. Vrijeme/Kristali

Ono što čini ovu igru posebnom je njen jedinstven način sakupljanja resursa. Igrač koristi moći mijenjanja vremena u nadi kako će stvoriti točno one kristale koji mu trebaju da bi kupio opremu i prešao igru.

Da bi se to ostvarilo prvo treba napraviti sistem kiše, sunca i snijega, njihovo nasumično stvaranje na početku svake razine te miješanje nasumično stvorenog i onog kojeg igrač odabere kako bi se na drugoj mapi stvorili točno ti kristali.

U skripti za promjenu scene (kod prelaska sa gornje na donju mapu) se postavlja inicijalno vrijeme na početku sve razine što je čini jednu od najvažnijih skripti.

```
private void Awake()
{
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
    }
    else
    {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
    Player = GameObject.Find("Player");

}
```

Slika 34. Funkcija Awake(), SceneChange

Na početku skripte kao što se vidi na Slici 35. u funkciju Awake() koja se poziv odmah kada igrač pokrene igru za razliku od funkcije Start() koja se poziva tek kada ta scena postane aktivna.

DontDestroyOnLoad služi za to da se objekt na kojem je ova skripta ne uništi tokom mijenjanja scena. Također se pronalazi igrač na početku igre.

```
private void Start()
{

    if (weatherChangeCanvas == null)
    {
        Debug.LogError("WeatherChangeCanvas reference is missing!");
        return;
    }

    initialCrystalType = GetRandomCrystalType();

    kButton.onClick.AddListener(() => OnWeatherChangeClick("K"));
    sButton.onClick.AddListener(() => OnWeatherChangeClick("S"));
    snButton.onClick.AddListener(() => OnWeatherChangeClick("SN"));


    SceneManager.activeSceneChanged += OnSceneChanged;
}
```

Slika 35. Funkcija Start(), SceneChange

U funkciji Start(), kao što se vidi na Slici 36. se početni tip kristala, tj. početno vrijeme postavlja na vrijednost funkcija GetRandomCrystalType(), a na gumbove se dodaje listener koji ih pretplati na događaj tako da kada se kliknu svaki pozove funkciju OnWeatherChangeClick() s oznakom svoga vremena (K – kišovito, S – sunčano, SN – snježno). Izgled gumba vidimo na Slici 40.

```
public void OnWeatherChangeClick(string selectedType)
{
    if (weatherChangeCanvas != null)
    {
        weatherChangeCanvas.gameObject.SetActive(true);
    }
    selectedCrystalType = selectedType;
    Debug.Log("OnWeatherChangeClick called with type: " + selectedType);

    if (Player != null)
    {
        Player.transform.position = coordinates;
    }

    SceneChangeUtility.ChangeScene("Dan");

}
```

Slika 36. Funkcija OnWeatherChangeClick()

selectedCrystalType se postavi na vrijednost koju je korisnik odabrao klikom na jedan od tri gumba pomoću čega se zabilježi njihov odabir. Igraču se mijenjaju koordinate koje su postavljena tako da kada se promijeni scena igrač bude na sceni „Dan“ odmah ispred skala kako bi se dobio dojam da je prešao s jedne mape na drugu doslovno skalama. Zadnja linija koda mijenja trenutnu scenu na scenu „Dan“ što se vidi na Slici 37.

```
private string GetRandomCrystalType()
{
    int randomIndex = Random.Range(0, 3);
    switch (randomIndex)
    {
        case 0: return "K";
        case 1: return "S";
        case 2: return "SN";
        default: return "K";
    }
}
```

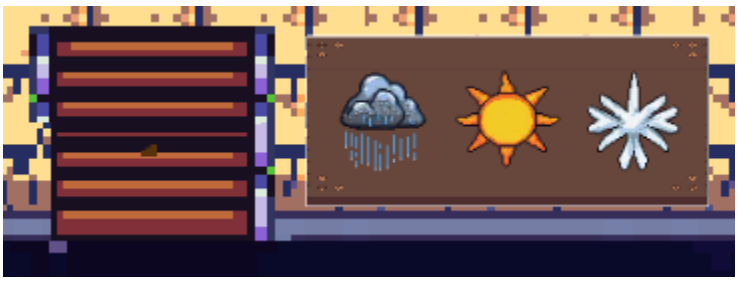
Slika 37. Funkcija GetRandomCrystalType()

Na Slici 38. se vidi jednostavna funkcija koja uzima nasumičan broj koji odlučuje vrijeme koje će biti postavljeno svakoga dana tako što uzme broj u switch gdje ovisno o slučaju postavi prikladno vrijeme.

```
private void OnSceneChanged(Scene previousScene, Scene newScene)
{

    if (newScene.name == "Dan")
{
    Debug.Log("Scene changed to 'Dan'. Crystal type remains: " + initialCrystalType);
    return;
}
    initialCrystalType = GetRandomCrystalType();

}
```

Slika 38. Funkcija OnSceneChanged

Na Slici 39. vidimo funkciju OnSceneChanged(), jako važna funkcija koja svaki put kada se scena promijeni provjerava ime trenutne scene. Ukoliko je scena „Dan“ ne dogodi se ništa ali ako je bilo koja druga scena postavlja se novo vrijeme kao što je napravljeno u funkciji Start().



Slika 39. Izgled gumba za odabir vremena

Skripta CrystalSpawner služi za stvaranje kristala po mapi dan ovisno o početnome vremenu i vremenu kojeg je igrač odabrao.

```
private void Start()
{
    if (SceneChange.Instance != null)
    {
        string initialType = SceneChange.Instance.InitialCrystalType;
        SpawnCrystals(initialType);

        string selectedType = SceneChange.Instance.SelectedCrystalType;
        if ((initialType == "K" && selectedType == "S") || (initialType == "S" && selectedType == "K"))
        {
            SpawnRainbowCrystals();
        }
        if ((initialType == "K" && selectedType == "SN") || (initialType == "SN" && selectedType == "K"))
        {
            SpawnMuddyCrystals();
        }
        if ((initialType == "S" && selectedType == "SN") || (initialType == "SN" && selectedType == "S"))
        {
            SpawnSunFrostCrystals();
        }
        if ((initialType == "S" && selectedType == "S"))
        {
            SpawnSunCrystals();
        }
        if ((initialType == "K" && selectedType == "K"))
        {
            SpawnThunderCrystals();
        }
        if ((initialType == "SN" && selectedType == "SN"))
        {
            SpawnIceCrystals();
        }
```

Slika 40. Funkcija Start(), CrystalSpawner

Na Slici 41. vidimo da se na početku skripte u varijablu initalType se sprema vrijeme koje je postavljeno na početku dana koji se odmah stvaraju pomoću funkcije SpawnCrystals() te se u varijablu selectedType sprema vrijeme koje je igrač odabrao nakon čega slijedi provjera vremena.
Napravljene su if petlje za svaki mogući slučaj na temelju kojih se stvaraju određeni kristali.

```
private void SpawnCrystals(string type)
{
    GameObject crystalPrefab;

    switch (type)
    {
        case "K":
            crystalPrefab = crystalKPrefab;
            break;
        case "S":
            crystalPrefab = crystalSPrefab;
            break;
        case "SN":
            crystalPrefab = crystalSNPrefab;
            break;
        default:
            Debug.LogError("Unknown crystal type: " + type);
            return;
    }

    SpawnCrystalPrefab(crystalPrefab, numberOfCrystals);
}
```

Slika 41. Funkcija SpawnCrystals()

Na Slici 42. vidi se funkcija SpawnCrystals() služi za provjeru osnovnih vrsta kristala putem switch petlje koja uzima string varijablu tipa kristala te se poziva funkcija SpawnCrystalPrefab koja ih stvara koju vidimo na Slici 43.

```
private void SpawnCrystalPrefab(GameObject prefab, int count)
{
    int spawnedCount = 0;
    while (spawnedCount < count)
    {
        Vector3 spawnPosition = GenerateValidSpawnPosition();

        Instantiate(prefab, spawnPosition, Quaternion.identity);
        spawnedCount++;
    }
}
```

Slika 42. Funkcija SpawnCrystalPrefab()

Funkcija prima dva argumenta, jedan je prefab kristala kojeg treba stvarati a drugi je broj kristala koliko će ih stvoriti koji je postavljen na 10 za osnovne vrste kristala.

```
private Vector3 GenerateValidSpawnPosition()
{
    Vector3 spawnPosition;
    int maxAttempts = 10000; // Maksimalan broj pokušaja
    int attempts = 0;

    do
    {
        spawnPosition = new Vector3(
            Random.Range(boundaryBounds.x, boundaryBounds.x + boundaryBounds.size.x * boundaryTilemap.cellSize.x),
            Random.Range(boundaryBounds.y, boundaryBounds.y + boundaryBounds.size.y * boundaryTilemap.cellSize.y),
            0);

        attempts++;

    } while ((IsPositionOnRestrictedTilemap(spawnPosition) || !IsPositionWithinBoundary(spawnPosition)) && attempts < maxAttempts);

    if (attempts >= maxAttempts)
    {
        Debug.LogWarning("Couldn't find a valid spawn position after " + maxAttempts + " attempts.");
    }

    return spawnPosition;
}
```

Slika 43. Funkcija GenerateValidSpawnPosition()

Funkcija koju vidimo na Slici 444. generira polje na koje se kristal može stvoriti, ukoliko se nasumično uzme polje sa tilemape koja je postavljena na restricted varijablu traži drugo nasumično polje dok ne nađe polje koje je unutar granica tj. tilemape postavljene na boundary varijablu.
Uzima nasumično koordinatu x i koordinatu y što se vrti u do-while petlji sve dok se ne dobije dozvoljeno polje.

Skripta Crystal je još jedna skripta koja upravlja sakupljanjem kristala samo iz drugačije perspektive.

```
public void Update()
{
    if (Input.GetKeyDown(KeyCode.R))
    {
        if (Vector3.Distance(playerTransform.position, transform.position) <= collectRadius)
        {
            GameManager.Instance.CollectCrystal(crystalType);
            InventoryManager.instance.AddItem(itemReference);
            audioManager.PlaySFX(audioManager.pickupSound);
            Debug.Log(crystalType);

            randomCrystalCount = Random.Range(1, 4);

            switch (crystalType)
            {
                case "Fire":
                    InventoryManager.instance.AddCrystals(CrystalType.Fire, randomCrystalCount);
                    int fireCount = InventoryManager.instance.GetCrystalCount(CrystalType.Fire);
                    Debug.Log("Fire crystals count: " + fireCount);
                    break;
```

Slika 44. Funkcija Update(), Crystal

U funkciji Update(), kao što vidimo na Slici 45. se provjerava je li pritisnuta tipka R, ukoliko je pritisnuta onda se gleda ako je igrač unutar zadanog radijusa za skupljanje kristala kako bi se ostvarilo to da igrač treba biti blizu kristalu kako bi ga skupio. Pomoću funkcije AddItem() iz skripte InventoryManager kristal se dodaje igraču u inventory te pomoću audioManager-a svaki put kada se skupi kristal čuje se zvuk skupljanja.
Zatim se uzima nasumičan broj od 1 do 4 kako ne bi uvijek bio isti broj kristala kojeg igrač dobije nego se doda i sreća u igru kako ne bi igraču igra izgledala isto svaki put kada je igra. Switch petlja služi za provjeru kristala kojeg treba dodati u inventory, postoje slučajevi za sve kristale ovdje vidimo primjer za Fire kristal koji se pomoću funkcije AddCrystals() iz skripte InventoryManager dodaje u inventory što se ostvaruje putem dva argumenta, jedan koji šalje tip kristala i drugi koji šalje koliko je kristala skupljeno.

4.5.5 Tutorial

Slika 45. Izgled tutoriala u hijerarhiji

Tutorial, kao što vidimo na Slici 46. je napravljen od objekta Canvas u kojeg je stavljena slika koja predstavlja sprite lika te još 6 slika od kojih svaka ima i objekt Text koji predstavljaju tekstualni oblak gdje igrač čita upute. Skripta Tutorial postavljena je na Canvas(Guide) kojom se upravlja kada je koja slika aktivna.

```
void Start()
{
    previousPosition = Player.transform.position;

    ActivateSpeechBubble(0);
}
```

Slika 46. Funkcija Start(), Tutorial

Na početku scene aktivira se prva slika pomoću funkcije ActivateSpeechBubble() kao što vidimo na Slici 47.

```
void Update()
{
    if (Player.transform.position != previousPosition && count1 == 0)
    {
        previousPosition = Player.transform.position;
        ActivateSpeechBubble(1);
        count1 = 1;
    }

    if ((Input.GetKeyDown(KeyCode.Alpha1) || Input.GetKeyDown(KeyCode.Alpha2)) && count1 == 2)
    {
        ActivateSpeechBubble(3);
        count1 = 3;

    }

    if (Input.GetMouseButtonDown(0) && IsNear(Player, "Stairs") && count1 == 3)
    {
        ActivateSpeechBubble(4);
        count1 = 4;
    }

    if (Book.activeSelf && count1 == 1)
    {
        ActivateSpeechBubble(2);
        count1 = 2;

    }

    if (Input.GetKeyDown(KeyCode.R) && IsNear(Player, Anvil) && count1 == 4)
    {
        ActivateSpeechBubble(5);
        count1 = 5;
    }

}
```

Slika 47. Funkcija Update(), Tutorial

Na Slici 48. vidimo funkciju Update() gdje se aktiviraju ostale slike ovisno o uvjetu te kolika je vrijednost varijable count1 koja se koristi kako ne bi trebalo raditi nove funkcije za gašenje slika jer će prvi dio svakoga if-a uvijek biti ispunjen kada se ispuni prvi put ali varijabla count1 sprječava aktivaciju već viđenih slika. Prvi if provjerava je li se igrač maknuo s mjesta, ako je i ako je count1 jednak nuli aktivira se sljedeća slika i povećava varijabla count1. Druga slika igraču govori o promjeni oružja pomoću tipki 1 i 2 te kada to obavi prelazi se na iduću sliku koja govori igraču da klikne na skale i odabere jedno od tri vremena i skupi kristale. Kada je to obavio nova slika mu kaže da klikne na knjigu gdje su upute o igri. Kada je sve to obavio treba kliknuti na shop tipkom R te nakon svega toga igraču se izbaci poruka da kada je spreman za borbu da pritisne tipku B.

4.5.6. Inventory

Cijeli inventory se nalazi pod hud canvasom, a sastoji se od više dijelova koji su:

- Gumb koji je uvijek vidljiv na kojega kada igrač klikne lijevim klikom ili pritisne tipku I otvori se MainInventory objekt
- MainInventory je empty object pod kojim se nalaze jedan canvas koji predstavlja tamnu pozadinu kada se otvori inventory i canvasa koji za pozadinu ima sliku praznog inventorya na kojeg je dodan Grid Layout Group kako bi se dobila zasebna mjesta u inventoryu za svaki item čije sprite-ove možemo vidjeti na Slici 49.

Slika 48. Prikaz sprite-a za Pozadinu Inventory-a i za jedno polje u inventory-u

- Empty objekt na kojem se nalazi skripta koja upravlja inventoryem Inventory Manager

Inentory Manager skripta služi za dodavanje itema u inventory, manipulaciju kristalima (njihovo dodavanje i provjera ima li ih igrač dovoljno kada kupuje stvar iz shopa) i za aktivaciju i deaktivaciju mačeva ovisno o tome koji je na predodređenome mjestu za njih,

```
private void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject);
        InitializeCrystals();
    }
    else
    {
        Destroy(gameObject);
    }
    swordToObjectMap.Add("FireSword", fireSwordGameObject);
    swordToObjectMap.Add("WaterSword", waterSwordGameObject);
    swordToObjectMap.Add("IceSword", iceSwordGameObject);
    swordToObjectMap.Add("SunSword", sunSwordGameObject);
    swordToObjectMap.Add("SnowSword", snowSwordGameObject);
    swordToObjectMap.Add("ThunderSword", snowSwordGameObject);
    swordToObjectMap.Add("DefaultSword", defaultSwordGameObject);

}
```

Slika 49. Funkcija Awake(), InventoryManager

Na Slici 50. vidimo da se na početku skripte u funkciji Awake() inicijaliziraju se svi kristali i dodaju svi mačevi u dictionary prije inicijaliziran koji ima string varijablu ovdje ime mača i gameobject koji se dodao putem Unity editora za svaki mač.

```
private void Start()
{
    if (startItems.Length > 0) AddItemToSlot(startItems[0], 30);
    if (startItems.Length > 1) AddItemToSlot(startItems[1], 31);
    if (startItems.Length > 2) AddItemToSlot(startItems[2], 32);

    ChangeSelectedSlot(30);
}
```

Slika 50. Funkcija Start(), InventoryManager

Na Slici 51. vidimo da se u funkciji Start() putem funkcije AddItemToSlot() se dodaju tri itema koja će igrač imati odmah na početku igre a to su početni mač, luk i Heal Spell. Funkcija ChangeSlelectedSlot() služi za prikaz igraču ka kojem se mjesto u inventoryu trenutno nalazi.

Poziva se i u funkciji Update() gdje se igraču omogućuje da mijenja prvih pet mjesta u inventoryu pomoću tipki od 1 do 5. CheckFirstSlotForSword() se također poziva u funkciji Update() koja je već opisana prije u radu.

```
public bool AddItem(Item item)
{
    for (int i = 0; i < inventorySlots.Length; i++)
    {
        InventorySlot slot = inventorySlots[i];
        if (slot != null)
        {
            InventoryItem itemInSlot = slot.GetComponentInChildren<InventoryItem>();
            if (itemInSlot != null && itemInSlot.item == item && itemInSlot.count < maxStackedItems
            && itemInSlot.item.stackable == true)
            {
                itemInSlot.count = itemInSlot.count + RandomValue;
                itemInSlot.RefreshCount();
                return true;
            }
        }
    }

    for (int i = 0; i < inventorySlots.Length; i++)
    {
        InventorySlot slot = inventorySlots[i];
        if (slot != null)
        {
            InventoryItem itemInSlot = slot.GetComponentInChildren<InventoryItem>();
            if (itemInSlot == null)
            {
                SpawnNewItem(item, slot);
                return true;
            }
        }
    }

    return false;
}
```

Slika 51. Funkcija AddIem()

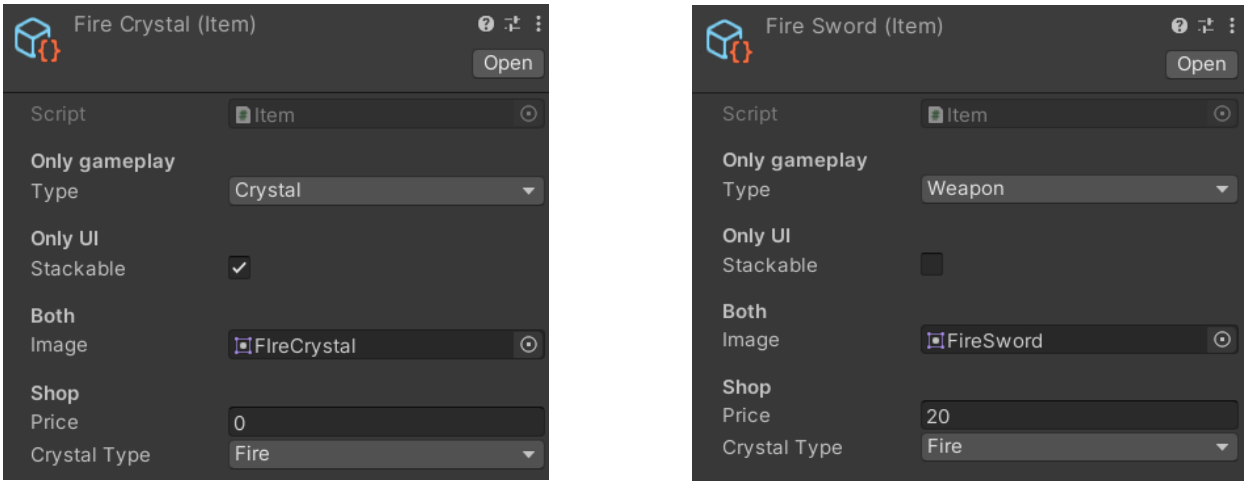
Funkcija AddItem(), koja se vidi na Slici 52. služi za dodavanje novih ili već postojećih itema u inventory.
Prva for petlja ide kroz sva mjesta u inventoryu te ukoliko postoji već neki item na tome mjestu, ako je item stackable što znači da može imate više istih itema na istome mjestu (npr. kristali mogu dok mačevi ne mogu) i ako broj tog itema nije dosegao broj zadan kao maksimalan broj toga itema dodaje na count toga itema varijabla RandomValue koja sadrži vrijednost koliko je kristala igrač dobio (od 1 do 4) kada je pokupio kristal.
Druga for petlja također prolazi kroz sva mjesta u inventoryu se poslije gleda ako je mjestu prazno te se poziva funkcija SpawnNewItem() za dodavanje novoga itema u inventory.

Za prepoznavanje je li item stackable ili ne i općenito svi itemi su napravljeni kao scriptable objekti gdje su definirani razni podaci za svaki item koji se nalaze u skripti Item.

```
[CreateAssetMenu(menuName = "Scriptable object/Item")]
public class Item : ScriptableObject
{
    [Header("Only gameplay")]
    public ItemType type;

    [Header("Only UI")]
    public bool stackable = true;

    [Header("Both")]
    public Sprite image;

    [Header("Shop")]
    public int price;
    public CrystalType crystalType;
}

public enum ItemType {
    Crystal,
    Weapon
}

public enum CrystalType {
    Fire,
    Ice,
    Mud,
    Rainbow,
    SunFrost,
    Snow,
    Sun,
    Thunder,
    Water
}
```

Slika 52. Skripta Item

Na Slici 53. vidimo da postoje 4 polja prvo se odnosi samo na gameplay i gleda se koji je tip itema za što se ispod u kodu vidi da je napravljena enumeracija koja određuje njihov tip između kristala i oružja.
Drugo polje provjerava ako je item stackable što je koristilo prije u skripti Inventory Manager pomoću čega se odlučivalo može li ići više istih itema na isto mjesto. Sljedeće polje je za sliku itema. U zadnjem polju Shop se određuje cijena itema te tip kristala kojim se plaća za taj item.



Slika 53. Primjer stackable itema(Fire Crystal) i ne stackable itema (Fire Sword)

Na slici 54. vidimo kako to izgeda u Unity editoru, prvo što vidimo je primjer za Fire kristal gdje mu je postavljeno da je tip kristal, da je stackable te u shopu nema cijene jer se oni skupljaju po mapi a ne kupuju. Drugo što vidimo je primjer itema koji je oružje zbog čega nije postavljen da je stackable te mu je cijena postavljena na 20 fire kristala.

Zadnja skripta za inventory koja omogućuje „kretanje“ itema između mjesta je InventoryItem koja sadrži pet funkcija gdje su tri osnovne funkcije za premještanje itema po mjestima u inventoryu, a dvije jako važne funkcije koje se koriste stalno su InitialiseItem() i RefreshCount().

```
public void InitialiseItem(Item newItem) {
    item = newItem;
    image.sprite = newItem.image;
    RefreshCount();
}

public void RefreshCount() {
    countText.text = count.ToString();
    bool textActive = count > 1;
    countText.gameObject.SetActive(textActive);
}
```

Slika 54. Funkcije IntialiseImte() i RefreshCount()

Na Slici 55. vidimo dvije funkcije.

Prva funkcija stvara novi objekt s novim itemom koji se poslao funkciji kao argument te za sliku itema uzima sliku koja se nalazi u poslanome objektu poslije čega poziva funkciju RefreshCount(). Svaki put kada se poziva neka funkcija koja dodaje item na mjesta u inventoryu ta funkcija će pozivati i ovu funkciju.

Druga funkcija prikazuje koliko toga itema ima na jednome mjestu, a tekst je vidljiv samo ako je broj veći od 1, ukoliko tekst nije veći od 1 onda nema potrebe za prikazivanjem broja. Ova funkcija se poziva svaki put kada se skupe kristali kako bi igrač uvijek znao točan broj.

Slika 55. Primjer inventoya s itemima

Na Slici 56. vidimo primjer kako izgledaju itemi u inventoryu i oni koji su stackable kao prva dva koji su kristali i piše ispod njih broj koliko ih igrač ima te mač koji nije stackable tako da ni ne piše ništa ispod njega.

4.5.7. Shop

Na Slici 57. vidimo shop u kojem igrač može kupovati različite iteme ako ima dovoljan broj kristala čija cijena piše ispod svakog itema.

Shop je jedan od glavnih dijelova igre pomoću kojeg igrač može smišljati taktike za odabir vremena ovisno o tome koje iteme želi kupiti.

Slika 56. Izgled Shop-a

U hijerarhiji u Unity-u postavljen je slično kao i inventory, nalazi se u hud canvasu gdje također ima tamnu pozadinu kada se shop otvori i canvas ShopScreen na kojeg je postavljen Grid Layout Group.

Kao što inventory ima skriptu Inventory Manger koja upravlja njime, tako i shop ima skritpu Shop Manager koja služi za upravljanje radnjama vezanim uz shop.

```
private void InitializeShop()
{
    if (itemsForSale.Length != shopSlots.Length)
    {
        Debug.LogError("Number of items does not match number of slots!");
        return;
    }

    for (int i = 0; i < shopSlots.Length; i++)
    {
        shopSlots[i].SetupSlot(itemsForSale[i], this);
    }
}
```

Slika 57. Funkcija InitializeShop()

Na Slici 58. vidimo funkciju InitializeShop() koja postavlja iteme u shop tako što prođe kroz sva shop mjesta te pomoću funkcije SetupSlot() postavi svaki item posebno.

```
public void SetupSlot(Item item, ShopManager manager)
{
    this.item = item;
    this.shopManager = manager;

    if (item != null)
    {
        if (itemImage != null)
        {
            itemImage.sprite = item.image;
        }
        if (itemPrice != null)
        {
            itemPrice.text = item.price.ToString();
        }
        if (crystalImage != null)
        {
            crystalImage.sprite = GetCrystalSprite(item.crystalType);
        }
    }
}
```

Slika 58. Funkcija SetupSlot()

Na Slici 59. vidimo funkciju SetupSlot() koja od itema koji je proslijeđen uzima sliku i cijenu itema.

Kada korisnik ide kupiti item provjerava se ima li dovoljno kristala za kupovino putem funkcije HasEnoughCrystals() koja se nalazi u skripti Inventory Manager a poziva se u funkciji CanAfford() koju vidimo na Slici 60. u skripti Shop Manager gdje se prvo provjerava postoji li uopce objekt sa skriptom Inventory Manager.

```
public bool CanAfford(Item item)
{
    if (InventoryManager.instance != null && item != null)
    {
        return InventoryManager.instance.HasEnoughCrystals(item);
    }
    return false;
}
```

Slika 59. Funkcija CanAfford()

Kada korisnik klikne na item poziva se funkcija iz skripte ShopSlot OnPointerClick() koja poziva funkciju PurchaseItem() iz skripte Shop Manager kada korisnik klikne na neki od itema u shopu. Zatim funkcija PurchaseItem() koju vidimo na Slici 61. prvo poziva funkciju CanAfford() za provjeru kristala nakon čega se pozivaju funkcije ReduceCrystal() i AddItem() iz Inventory Managera kako bi se korisniko smanjio broj kristala u inventoryu za onoliko koliko je koštao item i da mu se doda item na prvo slobodno mjesto u inventory.

```
public void PurchaseItem(Item item)
{
    if (CanAfford(item))
    {
        InventoryManager.instance.ReduceCrystals(item);
        InventoryManager.instance.AddItem(item);
        Debug.Log("Item purchased: " + item.name);
        audioManager.PlaySFX(audioManager.pickupSound);


    }
    else
    {
        Debug.Log("Not enough crystals to purchase: " + item.name);
    }
}
```

Slika 60. PurchaseItem()

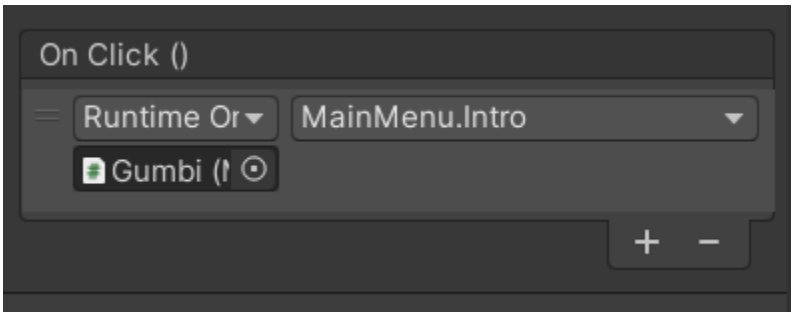
4.5.8. Početna i zadnja scena

Početna scena je napravljena pomoću dva canvasa, u jednome canvasu se nalazi pozadinska slika i slika za pozadinu izbornika, a u drugome canvasu se nalaze sve gumbi.

Slika 61. Hijerarhija scene MainMenu

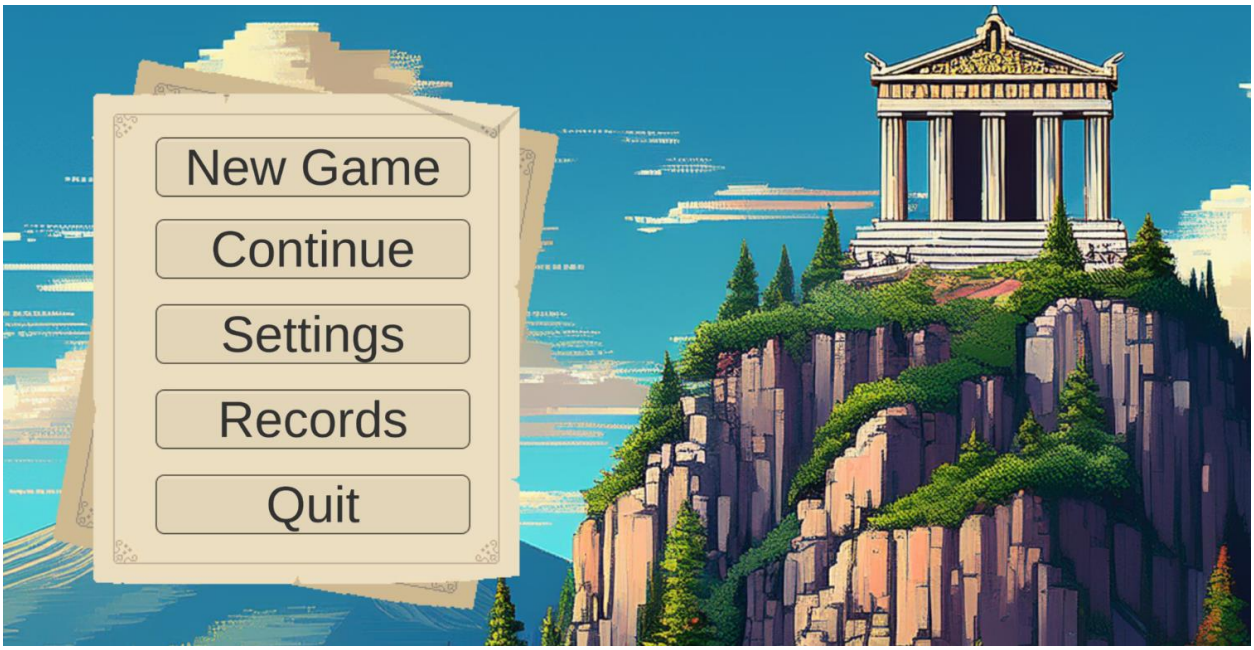
Na Slici 62. prikazana je hijerarhija MainMenu scene.

Na canvas „Gumbi“ postavljena je jednostavna skripta u kojoj su funkcije od kojih svaka vodi na drugu scenu ovisno o tome koju opciju korisnik odabere. Scene se mijenjaju pomoću linije: SceneManager.LoadScene(„ImeScene“), a aplikacija se gasi linijom Application.Quit().



Slika 62. Dodana funkcija Intro() iz skripte MainMenu na gumb

Na Slici 63. prikazan je primjer kako se dodaje određena funkcija iz skripte MainMenu koja je postavljena na canvas Gumbi na određeni gumb. Klikne se na padajući izbornik gdje je sada MainMenu.Intro, odabere se skripta MainMenu gdje se prikaže novi izbornik za odabir željene funkcije.



Slika 63. Izgled početne scene u igri

Na Slici 64. je prikazan izgled početne scene

Nakon što igrač odabere opciju New Game prikaže se slika na kojoj je ispisana priča igre te je psotavljen AI voice over koji čita tekst.

Zadnja scena se prikaže kada korisnik umre u borbi gdje mu se prikaže koliko je dugo vremena bio živ.
Jednostavna scena gdje ima samo jedan canvas u kojem su slika koja služi za pozadinu, tekst koji se ispisuje na pozadini i gumb koji služi da igrača vrati na prvu razinu.

Slika 64. Death Screen

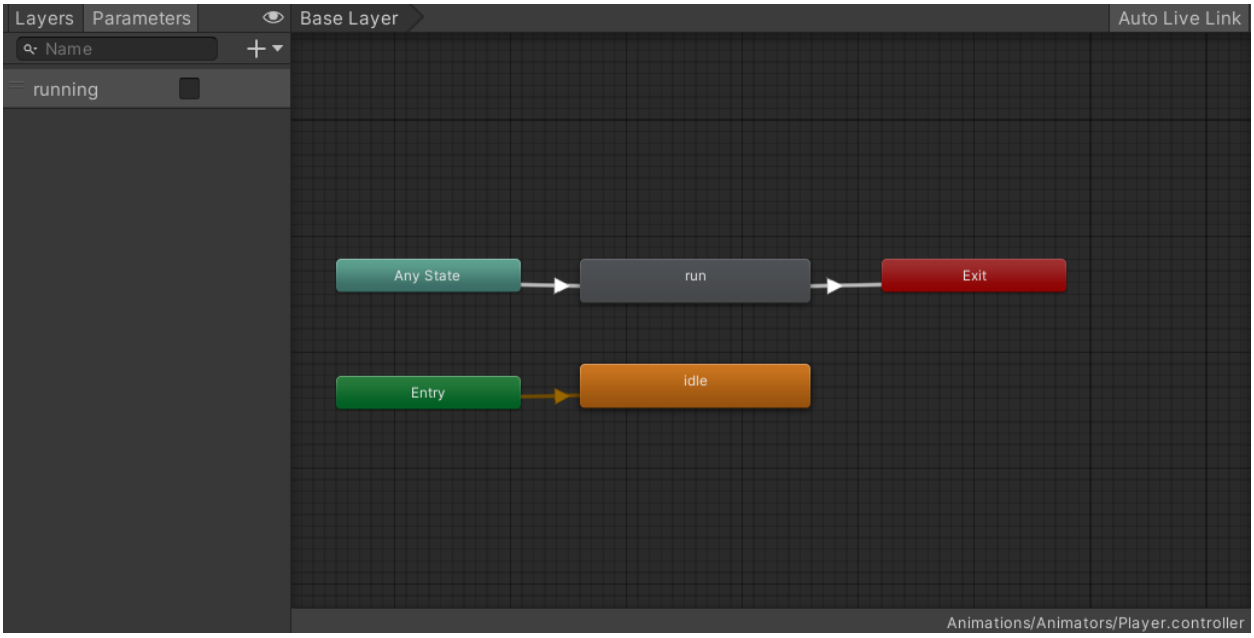
Na Slici 65. se prikazuje Death Screen kada igrač umre
Vrijeme koje se ispisuje broji se u pozadini otkad igrač završi sa introm pa sve dok ne umre nakon čega se resetira vrijeme i počne brojati ispočetka.

Ukoliko igrač uspješno prijeđe igru neće dobiti Death Screen nego Victory Screen koji radi na isti način samo što mu nije ispisano da je mrtav nego dobije tekst koji opisuje njegov uspjeh skupa s vremenom koliko dugo mu je trebalo da prijeđe igru. Kada se klikne na gumb na Victory ekranu ne vrati igrača na prvu razinu nego na MainMenu scenu.

4.5.9. Animatori i animacije

Unity animator služi za animaciju raznih likova i objekata na način da izgleda prirodno i dobro složeno. Omogućuje korisniku kontroliranje prijelaza između stanja dodavanjem različitih varijabli i uvjeta kada će određena animacija biti aktivna i kada će se određena animacija ugasiti. Zadanim varijablama se jako lagano upravlja iz skripti gdje možemo mijenjati njihove vrijednosti kada se ispuni potreban uvjet.

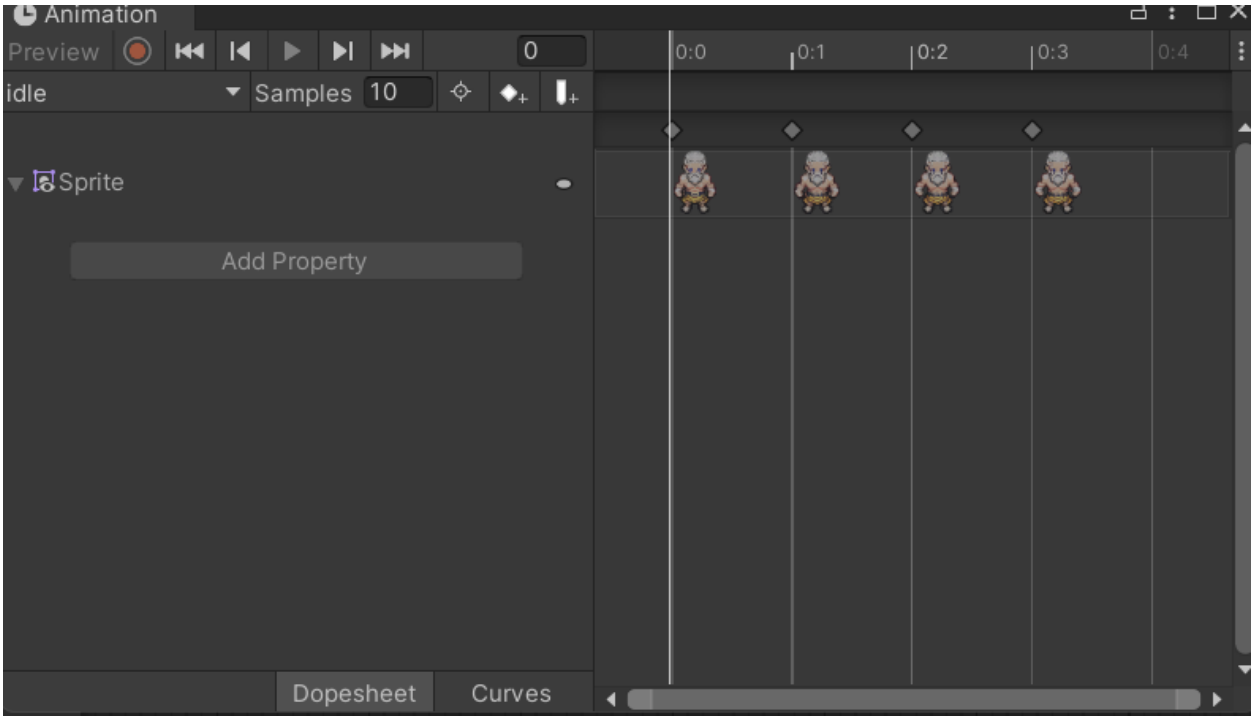
Za igru Trial of Cronos korišteno je mnogo animatora, točnije za kretanje Cronosa i svih neprijatelja (range, melee, tank, Posejdon, Zeus, Hades, skeleton), također je korišten za napade svih bogova i prijelaze između faza (Hadesov prijelaz između prve i druge faze te Zeus kada baca munje s vrha mape).



Slika 65. Izgled animatora objekta "Player"

Na Slici 66. prikazan je primjer animatora igrača kojem je početno stanje postavljeno na Idle animaciju. Na prijelazu između Any State i animacije run postavljen je uvjet da varijabla running bude jednaka true što se postiglo u kodu tako što je postavljeno da kada se korisnik kreće running je true. Na strelici koja označava prijelaz između run i exit stanja postavljen je uvjet da je varijabla running jednaka false a to je uvijekt kada se igrač ne kreće što nije nigdje definirano u kodu ali je na početku automatski postavljena na false a s obzirom da je linija za running u

kodu postavljena u funkciji Update() u trenutku kada se prestane kretate varijabla se vraća na false.



Slika 66. Idle animacija Cronosa

Na Slici 67. je prikazan primjer animacije Cronosa kada miruje. Ako želimo da animacija bude brža ili sporija možemo promijeniti Samples koji je zapravo određuje koliko frame-ova animacija koristi u sekundi.

4. Zaključak

Tijekom izrade računalne igre Trial of Cronos, istražen je cjelovit proces razvoja računalne igre u alatu za njihovu izradu Unity te programskom jeziku C#. Kroz implementaciju različitih sustava igre poput inventara, dućana, izrade mapa i neprijatelja prikazan je tehnički aspekt igre. Detaljno je opisan i sustav skupljanja resursa i njihovo stvaranje na čiji utjecaj ima vanjsko vrijeme koje igrač kontrolira i stvara nove taktike za što brži prelazak igre.
Kombinirajući teorijska znanja s praktičkim izazovima razvoja računalnih igara, Trial of Cronos prikazuje cjelokupan proces od dizajna početne igre do njene implementacije i gotovog proizvoda.

5. Popis slika

6. Literatura

[1] Documentation, Unity Documentation, https://docs.unity.com/

[2] Dmitriy Byshonkov, „Platforms and engines“ Gamedev Reports https://gamedevreports.substack.com/p/gdc-and-game-developer-the-state

[3] Community, Unity Discussions, https://discussions.unity.com/

[4] Materijali s vježbi. Merlin https://moodle.srce.hr/2023-2024/course/view.php?id=170206#section-3

[5] „Number of PC gaming users worldwide from 2008 to 20024“ Statista https://www.statista.com/statistics/420621/number-of-pc-gamers/