Izrada igre borbene arene za više igrača u programskom alatu Unity

Kantoci, Mateo

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:556217

Rights / Prava: Attribution-NonCommercial-NoDerivs 3.0 Unported/Imenovanje-Nekomercijalno-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-03-29**

SVEUČILIŠTE U ZAGREBU FAKULTET ORGANIZACIJE I INFORMATIKE VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Kantoci

Izrada igre borbene arene za više igrača u programskom alatu Unity

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Kantoci

Matični broj: 0016107697

Studij: Informacijsko i programsko inženjerstvo

Izrada igre borbene arene za više igrača u programskom alatu Unity

DIPLOMSKI RAD

Mentor/Mentorica:

Dr. sc. Mladen Konecki

Varaždin, studeni 2020.

Mateo Kantoci

Izjava o izvornosti

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

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

_____

Sažetak

U ovom radu opisan je teorijski i praktičan dio izrade borbene igre koristeći programski alat *Unity*. Prije detaljne razrade razvoja igre borbene arene za više igrača, upoznati će se programski alat *Unity* te najosnovnije komponente potrebne za rad u programskom alatu. Nakon upoznavanja sa alatom opisati će se funkcionalnost igre i način na koji će igra biti implementirana. Nadalje, razradit će se razvoj igre koji je podijeljen na dvije scene. Svaka scena biti će detaljno razrađena kao i komponente i objekti koji se nalaze u pojedinoj sceni. Osim komponenata i objekta prikazati će se i detaljno razjasniti skripta za svaki objekt što će olakšati čitanje programskog koda pisanog u C# programskom jeziku.

Ključne riječi: programski alat, skripta, objekt igre, igrač, heroj, minioni, toranj

Sadržaj

Sadržaj iii
1. Uvod 1
2. Unity 2
2.1. Osnovni dijelovi Unity-a 2
2.1.1. Scena 3
2.1.2. Objekt igre 3
2.1.3. Objekti za ponovnu upotrebu 3
2.1.4. Svijetla 4
2.1.5. Kamere 4
2.1.6. Fizika 4
2.1.7. Čvrsto tijelo 4
2.1.8. Zid objekta 4
2.1.9. Skripte 4
2.1.10. Navigacija 5
3. Razvoj borbene arene 6
3.1. Scena glavnog izbornika 7
3.1.1. Objekti igre glavnog izbornika 8
3.1.1.1. Predvorje 8
3.1.1.2. Soba 10
3.1.1.3. Kontrola izbornika, postavke više igrača i informacija igrača 15
3.1.1.4. Mrežni igrač 17
3.2. Scena igre 19
3.2.1. Teren 20
3.2.1.1. Tornjevi 21
3.2.2. Glavni igrač 25
3.2.3. Ponovno oživljavanje AI miniona 43
3.2.4. Glavna kamera 54
3.2.5. Kamera života živih objekata 55
3.2.6. Korisničko sučelje heroja 56
3.2.7. Kraj igre 56
4. Zaključak 58
Popis literature 59
Popis slika 61

1. Uvod

Izrada igre borbene arene za više igrača inspirirana je od strane najpoznatijih online igara poput Dota 2 i League of Legends. Diplomski rad podijeljen je na teorijski i praktičan dio. Teorijski dio odnosi se na programski alat *Unity* i komponente i objekte koji su temelj za razvoj video igre, dok se praktični dio odnosi na sam razvoj igre. Ovim radom želim opisati, naučiti i podijeliti znanje kodiranja koristeći C# programski jezik, te ukazati na što jednostavniji način koliko se zapravo jednostavno može izraditi video igra koja na prvu zvuči veoma kompliciranо. Najnovije tehnologije pojednostavljuju rad sa više igrača te se sve više i više koriste proširenja za *Unity*. U ovome projektu koristiti će se *Photon* proširenje.

U prvome dijelu pojasniti će se što je to zapravo *Unity*, za što se koristi te platforme koje su trenutno podržane. Nakon upoznavanja potrebno je razumjeti osnovne komponente i objekte podržane u ovom programskom alatu, na koji način funkcioniraju te kada i u koju svrhu je potrebno odabrati odgovarajuću komponentu.

Slijedi detaljni opis borbene arene koji će pojasniti kako i na koji način pristupiti igri te koja je zapravo svrha i cilj. Nakon pojašnjenja i upoznavanja sa borbenom arenom postepeno će se opisati svaki objekt unutar igre prema prioritetu. Objekti, komponente i skripte su međusobno povezane pa će se također postepeno razjasniti pristup i međusobni odnos kako bih bilo lakše shvatiti zašto je baš na ovaj način igra implementirana.

Kroz korisničko sučelje i priložene slike dobit će se vizualna interpretacija implementirane igre te će se moći vrlo lako usporediti sa ostalim poznatim MOBA igrama. Diplomski rad će završiti zaključkom te će biti iznesena razmišljanja, rezultati kao i prednosti i nedostaci korištenja programskom alata *Unity.*

2. Unity

Unity programski je alat (eng. *Game engine*) na više platformi koji je razvijen sa strane kompanije *Unity Technolo*gies, a prvenstveno se koristi za razvoj video igara i simulacija za računala, konzola i mobilnih uređaja. Na *Apple*-ovoj svjetskoj konferenciji 2005. godine najavljen je samo za *OS X*, te je od tada proširen na 27 platformi.

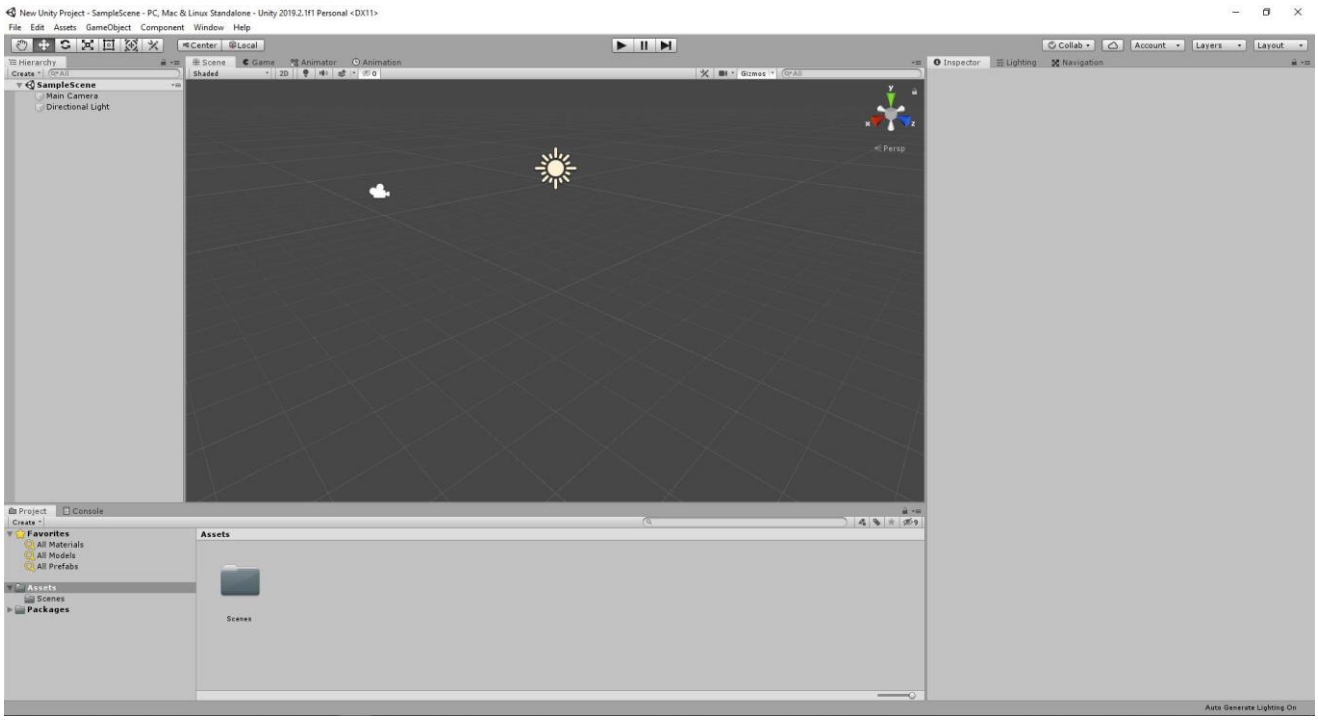
Sve namjenski je mehanizam za igre koji podržava 2D i 3D grafiku, „drag-and-drop“ funkciju i skriptiranje kroz *C#* programski jezik ili alternativno kroz *Javascript* programski jezik. Posebno je popularan u razvoju mobilnih igara te je najvećim dijelom fokusiran na mobilne platforme.

Unity također nudi usluge programerima, a to su: oglasi, analitika, certifikacija, izgradnja oblaka, igre sa više igrača, itd. Osim toga, *Unity* ima trgovinu imovinom u koju zajednica programera može preuzeti i prenijeti komercijalne i besplatne resurse treće strane poput tekstura, modela, dodataka, proširenja uređivača, pa čak i cijele primjere igara.

Značajan je po svojoj sposobnosti pozicioniranja igara na više platformi. Trenutno podržane platforme su *Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS family, Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL, Wii U, Windows, Windows Phone, Windows Store i Xbox One.*

2.1. Osnovni dijelovi Unity-a

Prilikom pokretanja *Unity*-a otvara se prazna scena na kojoj nema ništa te veoma neobično za korisnike koji je prvi put pokreću. Ono što se prvo može vidjeti su dva najbitnija prozora scena i igra pomoću kojih se sva radnja odvija prilikom kodiranja, testiranja, modeliranja, igranja, itd.



Slika 1. Scena

2.1.1. Scena

Scene sadrže okruženja i izbornike igre. Svaku datoteku koja predstavlja scenu može se smatrati jedinstvenom razinom. U svaku scenu postavlja se vlastito okruženje, prepreke i ukrasi, u osnovi dizajnira i gradi vlastita igra u dijelovima.

2.1.2. Objekt igre

Objekt igre (eng. *Game object*) najvažniji je koncept u programskom alatu *Unity*. Svaki objekt u igri je objekt igre, od likova i kolekcionarskih predmeta do svjetla, fotoaparata i specijalnih efekata. Međutim, objekt igre sam po sebi nije ništa sve dok mu ne pridružimo odgovarajuća svojstva. Također, djeluju kao spremnici za komponente, koji implementiraju funkcionalnost.

2.1.3. Objekti za ponovnu upotrebu

Objekti za ponovnu upotrebu (eng. *Prefab*) predstavljaju objekt igre u kompletu sa svim komponentama, svojstvima i konfiguracijom koji se ne nalazi u sceni nego u željenoj datoteci za ponovnu potrebu. Koristi se kada se javi potreba za kreiranjem više istih objekata pa je time omogućeno višestruko kreiranje instanci.

2.1.4. Svijetla

Svjetla su bitan dio svake scene. Dok mrežice i teksture definiraju oblik i izgled scene, svjetla definiraju boju i raspoloženje 3D okruženja. Vjerojatno će svatko raditi s više od jednog svjetla u svakoj sceni. Njihov zajednički rad zahtijeva malo prakse, ali rezultati mogu biti prilično nevjerojatni.

2.1.5. Kamere

Kamere se koriste za prikaz igraćeg svijeta igraču. Uvijek će postojati barem jedna kamera u sceni, ali može ih biti i više. Višestruke kamere mogu podijeliti zaslon za dva igrača ili stvoriti napredne prilagođene efekte. Omogućeno je animiranje kamera ili kontroliranje istih pomoću fizike. Praktički sve što se može zamisliti moguće je s kamerama, a mogu se koristiti tipične ili jedinstvene kamere ovisno o tome koje više odgovaraju stilu igre pojedinog igrača.

2.1.6. Fizika

Simulacija fizike omogućena je u projektu kako bih se osiguralo da se objekti pravilno ubrzavaju i reagiraju na sudare, gravitaciju i razne druge sile. Unity nudi različite implementacije fizičkog pogona koje se mogu koristiti u skladu sa potrebama pojedinog projekta: 3D, 2D, objektno orijentirane ili orijentirane na podatke.

2.1.7. Čvrsto tijelo

Čvrsto tijelo (eng. *Rigidbody*) glavna je komponenta koja omogućuje fizičko ponašanje objekta igre. Objekti igre se počnu raspršivati po sceni međusobnim udarom dvaju objekata. Dodjelom čvrstog tijela, objekt igre poprima masu i gravitaciju te se time omogućuje kretnja po terenu.

2.1.8. Zid objekta

Zidovi objekta (eng. *Colliders*) definiraju oblik objekta igre za potrebe fizičkih sudara. Te se mogu dijeliti prema 3D i 2D obliku:

- 3D - *Box Collider, Sphere Collider* i *Capsule Collider*
- 2D - *Box Collider 2D* i *Circle Collider 2D*

2.1.9. Skripte

Većina aplikacija treba skripte kako bih definirale uloge igrača te kako bi se događaji u igranju odvili na potreban način. Osim toga, skripte se mogu koristiti za stvaranje grafičkih

efekata, kontrolu fizičkog ponašanja objekata ili čak implementaciju prilagođenog AI sustava za likove u igri.

2.1.10. Navigacija

Navigacijski sustav omogućuje stvaranje likova koji se mogu inteligentno kretati po svijetu igre, koristeći navigacijske mreže koje se automatski kreiraju iz vaše scene geometrija. Dinamičke prepreke omogućuju izmjenu navigacije likovima tijekom izvođenja, dok veze izvan mreže omogućuju izgradnju specifičnih radnji poput otvaranja vrata ili skakanja s izbočine.

3. Razvoj borbene arene

Kreiranjem novog projekta putem *Unity Hub*-a prosljeđuje se na programski alata *Unity* te se automatski kreira nova igrača scena. Igra je zamišljena na način da se prilikom pokretanja korisniku prikaže glavni izbornik za odabir heroja. Nakon što svi korisnici odaberu svoga heroja biti će proslijeđeni na drugu scenu i borba započinje. Mapa se dijeli na dvije strane inspirirane prema Dota igrici *Radiant* i *Dire*, te su korisnici raspodijeljeni na timove. Igra se kontrolira pomoću standardnih strateških kontrola u stvarnom vremenu i prikazana je u trodimenzionalnoj izometrijskoj perspektivi. Također, mapa je sastavljena od triju traka na kojima se nalaze tornjevi čiji je cilj obrana od protivnika, a glavni cilj igrača je uništiti protivničku bazu odnosno glavni toranj nazvan *Ancient*.

Tim koji prvi uništi glavni toranj je pobjednik čime igra završava proglašenjem pobjednika. Na korisničkom sučelju biti će prikazan mjerač vremena koji će početi odbrojavati nakon što su svi igrači spojeni na igraču scenu te će se u 30. sekundi kao i svakih sljedećih 30 sekundi oživjeti protivnici (eng. *Minions*), u daljnjem kontekstu minioni, bazirani na umjetnoj inteligenciji. Minioni će napadati protivničke minione ukoliko su u blizini, u suprotnom napadati te će protivnički toranj koji će pucati u minione ukoliko su u blizini istog.

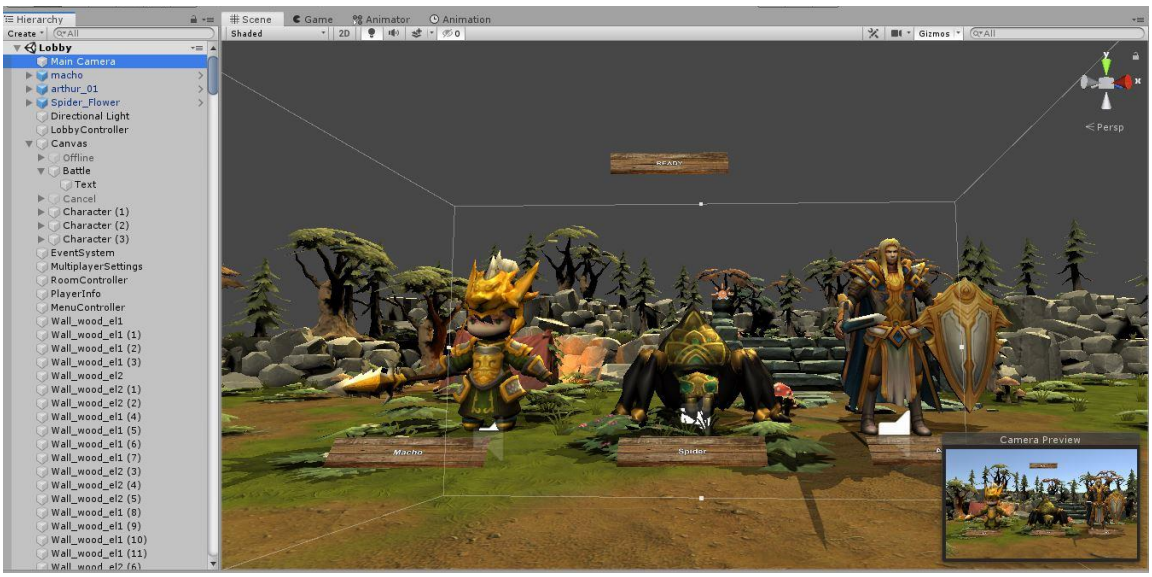
Svaki heroj ima svoje osobine (život, mana, bazna šteta (eng. *Base Damage*), šteta magije (eng. *Spell Damage*), iskustvo (eng. *Experience*), nivo (eng. *Level*)) te ispucavanjem magije ili običnog napadaja proizvode štetu protivniku. Iskustvo heroja se povećava kroz vrijeme i prilikom pogotkom miniona na način da igrač uništi istog pri čemu se dodjeljuje bonus iskustvo igraču što ga čini jačim ukoliko je iskustvo igrača veće od protivničkog igrača.

3.1. Scena glavnog izbornika

Kao što je prethodno navedeno, glavni izbornik sastoji se od tri heroja koje je moguće odabrati te je igrač zadužen potvrditi svoj odabir putem gumba spreman (eng. *Ready*). Svaki gumb je objekt igre (eng. *Game Object*) na koji je pridružena odgovarajuća skripta sa kontrolama koje se pozivaju pritiskom klika na isti.

Slika 2. Glavni izbornik

Prikaz izbornika omogućuje kamera koja je statično postavljena na željenu poziciju i rotaciju kako bih svaki korisnik imao isti prikaz. Većina izbornika u današnjim igrama koristi dvodimenzionalni prikaz, ali zbog jednostavnog korisničkog sučelja sa malo mogućnosti koristi se trodimenzionalni prikaz sa trodimenzionalnim modelima.



Slika 3. Scena glavnog izbornika

3.1.1. Objekti igre glavnog izbornika

Osim glavnih objekata (kamera, usmjereno svijetlo) koji se nalaze na sceni kreirani su objekti kontrola izbornika (eng. *Menu controller*), informacije igrača (eng. *Player info*), kontrola sobe (eng. *Room controller*), postavke više igrača (eng. *Multiplayer settings*) i kontrola predvorja (eng. *Lobby controller*). Najvažniji objekti su mrežni igrač (eng. *Photon network player*) i heroj igrača (eng. *Player avatar*) koji se ne nalaze na sceni nego su kreirani kao objekti za ponovnu upotrebu (eng. *Prefab*) jer će se koristiti u sceni igre.

3.1.1.1. Predvorje

Skripta se poziva na temelju objekta igre predvorja te se kreira statična instanca klase jer skripta nije namijenjena za promjenu varijabla tokom izvršavanja iste. Skripta proširuje klasu Photon proširenja te sve metode koje su nadjačane su zapravo metode iz Photon klase putem kojih su postavljene vlastite konfiguracije. Nadjačane metode koriste se spajanje korisnika na server te je ovdje najbitnija vlastita metoda *CreateRoom* koja se poziva prilikom nadjačanih metoda OnJoinRandomFailed ili OnCreateRoomFailed zbog toga što soba nije kreirana u trenutku poziva skripte što omogućuje kreiranje sobe bez obzira na pojavi grešaka prilikom izvršavanja skripte.

```
public class PhotonLobby : MonoBehaviourPunCallbacks
{
    public static PhotonLobby lobby;
    public GameObject battleButton;
    public GameObject cancelButton;

    private void Awake()
    {
        lobby = this;
    }

    private void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    public override void OnConnectedToMaster()
    {
        PhotonNetwork.AutomaticallySyncScene = true;
```

```
        battleButton.SetActive(true);
    }
    public void OnBattleButtonClicked()
    {
        PhotonNetwork.JoinRandomRoom();
    }
    public override void OnJoinRandomFailed(short returnCode, string
message)
    {
        CreateRoom();
    }
    void CreateRoom()
    {
        int randomRoomName = Random.Range(0, 10000);
        RoomOptions roomOps = new RoomOptions() { IsVisible = true, IsOpen
= true, MaxPlayers =
(byte)MultiPlayerSetting.multiPlayerSettings.maxPlayers };
        PhotonNetwork.CreateRoom("Room" + randomRoomName, roomOps);
    }
    public override void OnJoinedRoom()
    {
        Debug.Log("We are in a room");
    }

    public override void OnCreateRoomFailed(short returnCode, string
message)
    {
        CreateRoom();
    }

    public void OnCancelButtonClicked()
    {
        cancelButton.SetActive(false);
        battleButton.SetActive(true);
        PhotonNetwork.LeaveRoom();
    }
}
```

3.1.1.2. Soba

Skripta objekta sobe potrebna je za provjeru spojenih korisnika na server, kontrolu preusmjeravanja korisnika na igraču scenu i način pristupa prema željenoj sceni. Na način pristupa prema željenoj sceni podrazumijeva se olakšavanje prilikom testiranja same igre. Za inicijalizaciju klase koristi se *Singleton* uzorak dizajna koji omogućuje samo jednu instancu klase jer se neće više javljati potreba za novom instancom. Također, omogućena su dva pristupa: brzi start (eng. *Quick start*) i odgođeni start (eng. *Delay start*).

Ukoliko je omogućen brzi start tada nije potrebna kontrola broja spojenih igrača na server što omogućuje ulazak u igraču scenu bez obzira koliko je igrača trenutno spojeno, nadalje ukoliko je igra namijenjena za produkciju tada je potrebno omogućiti odgođeni start prilikom čega se provjerava postavljena varijabla koja predstavlja maksimalni broj igrača potreban za pokretanje igre i prijenos na igraču scenu. *Update* metoda potrebna je samo ukoliko je odabrani odgođeni start te se u beskonačnoj petlji provjerava broj spojenih igrača i u trenutku kada je količina spojenih igrača jednaka prema unaprijed postavljenoj konfiguraciji igra može započeti.

```
public class PhotonRoom : MonoBehaviourPunCallbacks, IInRoomCallbacks
{

    public static PhotonRoom room;
    private PhotonView PV;
    public bool isGameLoaded;
    public int currentScene;
    Player[] photonPlayers;
    public int playersInRoom;
    public int myNumberInRoom;
    public int PlayerInGame;
    private bool readyToCount;
    private bool readyToStart;
    public float startingTime;
    private float lessThenMaxPlayers;
    private float atMaxPlayers;
    private float timeToStart;

    private void Awake()
    {
        if (PhotonRoom.room == null)
```

```
    {
        PhotonRoom.room = this;
    }
    else
    {
        if (PhotonRoom.room != this)
        {
            Destroy(PhotonRoom.room.gameObject);
            PhotonRoom.room = this;
        }
    }
    DontDestroyOnLoad(this.gameObject);
}


public override void OnEnable()
{
    base.OnEnable();
    PhotonNetwork.AddCallbackTarget(this);
    SceneManager.sceneLoaded += OnSceneFinishedLoading;
}

public override void OnDisable()
{
    base.OnDisable();
    PhotonNetwork.RemoveCallbackTarget(this);
    SceneManager.sceneLoaded -= OnSceneFinishedLoading;
}

void Start()
{
    PV = GetComponent<PhotonView>();
    readyToCount = false;
    readyToStart = false;
    lessThenMaxPlayers = startingTime;
    atMaxPlayers = 6;
    timeToStart = startingTime;
}

void Update()
```

```
{
    if (MultiPlayerSetting.multiPlayerSettings.delayStart)
    {
        if (playersInRoom == 1)
        {
            RestartTimer();
        }
        if (!isGameLoaded)
        {
            if (readyToStart)
            {
                atMaxPlayers -= Time.deltaTime;
                lessThenMaxPlayers = atMaxPlayers;
                timeToStart = atMaxPlayers;
            }
            else if(readyToCount)
            {
                lessThenMaxPlayers -= Time.deltaTime;
                timeToStart = lessThenMaxPlayers;
            }
            if (timeToStart <= 0)
            {
                StartGame();
            }
        }
    }
}

public override void OnJoinedRoom()
{
    base.OnJoinedRoom();
    photonPlayers = PhotonNetwork.PlayerList;
    playersInRoom = photonPlayers.Length;
    myNumberInRoom = playersInRoom;
    PhotonNetwork.NickName = myNumberInRoom.ToString();
    if (MultiPlayerSetting.multiPlayerSettings.delayStart)
    {
        if (playersInRoom > 1)
        {
            readyToCount = true;
```

```
            }
            if (playersInRoom ==
MultiPlayerSetting.multiPlayerSettings.maxPlayers)
            {
                readyToStart = true;
                if( !PhotonNetwork.IsMasterClient)
                {
                    return;
                }
                PhotonNetwork.CurrentRoom.IsOpen = false;
            }
        }
        else
        {
            StartGame();
        }
    }

    public override void OnPlayerEnteredRoom(Player newPlayer)
    {
        base.OnPlayerEnteredRoom(newPlayer);
        photonPlayers = PhotonNetwork.PlayerList;
        playersInRoom++;
        if (MultiPlayerSetting.multiPlayerSettings.delayStart)
        {
            if (playersInRoom > 1)
            {
                readyToCount = true;
            }
            if (playersInRoom ==
MultiPlayerSetting.multiPlayerSettings.maxPlayers)
            {
                readyToStart = true;
                if (!PhotonNetwork.IsMasterClient)
                {
                    return;
                }
                PhotonNetwork.CurrentRoom.IsOpen = false;

            }
        }
```

```
    }

    void StartGame()
    {
        isGameLoaded = true;
        if (!PhotonNetwork.IsMasterClient)
        {
            return;
        }
        if (MultiPlayerSetting.multiPlayerSettings.delayStart)
        {
            PhotonNetwork.CurrentRoom.IsOpen = false;
        }

PhotonNetwork.LoadLevel(MultiPlayerSetting.multiPlayerSettings.multiplayerS
cene);
    }

    void RestartTimer()
    {
        lessThenMaxPlayers = startingTime;
        timeToStart = startingTime;
        atMaxPlayers = 6;
        readyToCount = false;
        readyToStart = false;
    }

    void OnSceneFinishedLoading(Scene scene, LoadSceneMode mode)
    {
        currentScene = scene.buildIndex;
        if (currentScene ==
MultiPlayerSetting.multiPlayerSettings.multiplayerScene)
        {
            isGameLoaded = true;
            if(MultiPlayerSetting.multiPlayerSettings.delayStart)
            {
                PV.RPC("RPC_LoadedGameScene", RpcTarget.MasterClient);
            } else
            {
                 RPC_CreatePlayer();
            }
```

```
        }
    }

    [PunRPC]
    private void RPC_LoadedGameScene()
    {
        PlayerInGame++;
        if (PlayerInGame == PhotonNetwork.PlayerList.Length)
        {
            PV.RPC("RPC_CreatePlayer", RpcTarget.All);
        }
    }

    [PunRPC]
    private void RPC_CreatePlayer()
    {
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"PhotonNetworkPlayer"), transform.position, Quaternion.identity, 0);
}
```

3.1.1.3. Kontrola izbornika, postavke više igrača i informacija igrača

Heroji, odnosno nazivi heroja smješteni su u polje koje se nalazi u skripti objekta igre informacija igrača te se na temelju klika na gumbove ispod određenog heroja poziva događaj (eng. *Event*) kreiran u skripti objekta igre kontrola izbornika koji postavlja vrijednost elementa polja, odnosno identifikaciju željenog heroja. Objekt igre kontrola izbornika služi za postavljanje postavke scena i način pristupa sceni igre.

```
public class MultiPlayerSetting : MonoBehaviour
{

    public static MultiPlayerSetting multiPlayerSettings;
    public bool delayStart;
    public int maxPlayers;
    public int menuScene;
    public int multiplayerScene;

    private void Awake()
    {
        if (MultiPlayerSetting.multiPlayerSettings == null)
        {
```

```
            MultiPlayerSetting.multiPlayerSettings = this;
        }
        else
        {
            if (MultiPlayerSetting.multiPlayerSettings != this)
            {
                Destroy(this.gameObject);
            }
        }
        DontDestroyOnLoad(this.gameObject);
    }
      }

public class MenuController : MonoBehaviour
{

    public void OnClickCharacterPick(int whichCharacter)
    {
        if (PlayerInfo.PI != null)
        {
            PlayerInfo.PI.mySelectedCharacter = whichCharacter;
            PlayerPrefs.SetInt("MyCharacter", whichCharacter);
        }
    }

      }

public class PlayerInfo : MonoBehaviour
{

    public static PlayerInfo PI;
    public int mySelectedCharacter;
    public string[] allCharacters;

    private void OnEnable()
    {
        if (PlayerInfo.PI == null)
        {
            PlayerInfo.PI = this;
        }
```

```
        else
        {
            if (PlayerInfo.PI != this)
            {
                Destroy(PlayerInfo.PI.gameObject);
                PlayerInfo.PI = this;
            }
        }
    }

    void Start()
    {
        if (PlayerPrefs.HasKey("MyCharacter"))
        {
            mySelectedCharacter = PlayerPrefs.GetInt("MyCharacter");
        }
        else
        {
            mySelectedCharacter = 0;
            PlayerPrefs.SetInt("MyCharacter", mySelectedCharacter);
        }
    }
}
```

3.1.1.4. Mrežni igrač

Nakon što su svi korisnici spojeni na server, koristeći skriptu soba (eng. *Room*) kreira se instanca objekta mrežnog igrača (eng. *Network player*), zatim pokreće se skripta istog i ovisno o raspodijeljenom timu kreira se instanca heroja igrača na unaprijed postavljenoj lokaciji u sceni igre ovisno o kojoj se strani radi: Radiant ili Dire. Pozivom *Photon Pun* metode započinje sinkronizacija igrača i pokreću se sve skripte dodijeljene objektu heroja igrača (eng. *Player avatar*) koje će kasnije biti detaljnije opisane.

```
public class PhotonPlayer : MonoBehaviour
{
    public PhotonView PV;
    public GameObject myAvatar;
    public int myTeam;
```

```
    void Start()
    {
        PV = GetComponent<PhotonView>();
        if (PV.IsMine)
        {
            PV.RPC("RPC_GetTeam", RpcTarget.MasterClient);
        }
    }

    private void Update()
    {
        if(myAvatar == null && myTeam != 0)
        {
            if (myTeam == 1)
            {
                if (PV.IsMine)
                {
                    myAvatar =
PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
                        GameSetup.GS.spawnPoints[0].position,
GameSetup.GS.spawnPoints[0].rotation, 0);
                }
            }
            else
            {
                if (PV.IsMine)
                {
                    myAvatar =
PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
                        GameSetup.GS.spawnPoints[1].position,
GameSetup.GS.spawnPoints[1].rotation, 0);
                }
            }
        }

    }

    [PunRPC]
    void RPC_GetTeam()
    {
        myTeam = GameSetup.GS.nextPlayerTeam;
        GameSetup.GS.UpdateTeam();
```

```
        PV.RPC("RPC_SentTeam", RpcTarget.OthersBuffered, myTeam);
    }

    [PunRPC]
    void RPC_SentTeam(int whichTeam)
    {
        myTeam = whichTeam;
    }
}
```

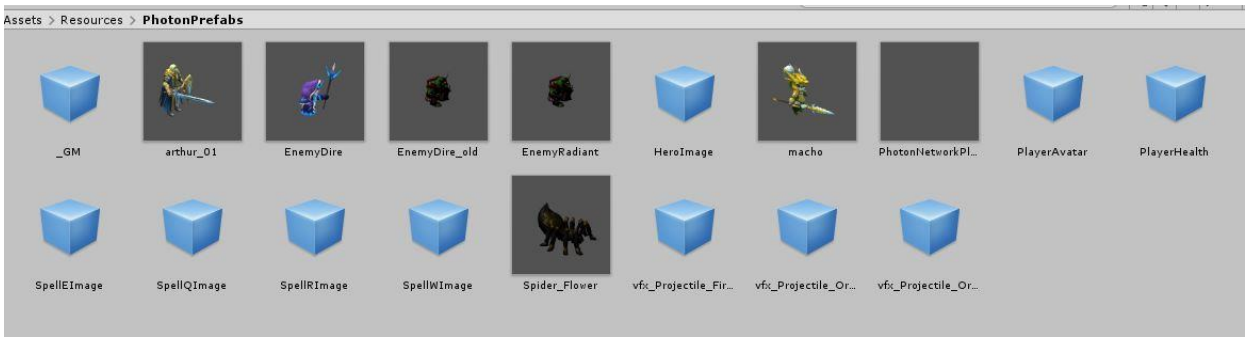
3.2. Scena igre

Scena igre glavna je scena na kojoj se odvija borba. Glavna scena sastoji se od nekoliko objekata igre:

- Teren (eng. *Level environment*)
- Oživljavanje protivnika (eng. *Wave spawner*)
- Postavke igre (eng. *Game setup*)
- Glavna kamera (eng. Main camera)
- Kamera za život objekata (eng. Health camera)
- Dire toranj (eng. Dire tower)
- Radiant toranj (eng. Radiant tower)
- Dire glavni toranj (eng. Dire ancient)
- Radiant glavni toranj (eng. Radiant ancient)

Osim navedenih objekata kreirani su i objekti koji se ne nalaze na sceni:

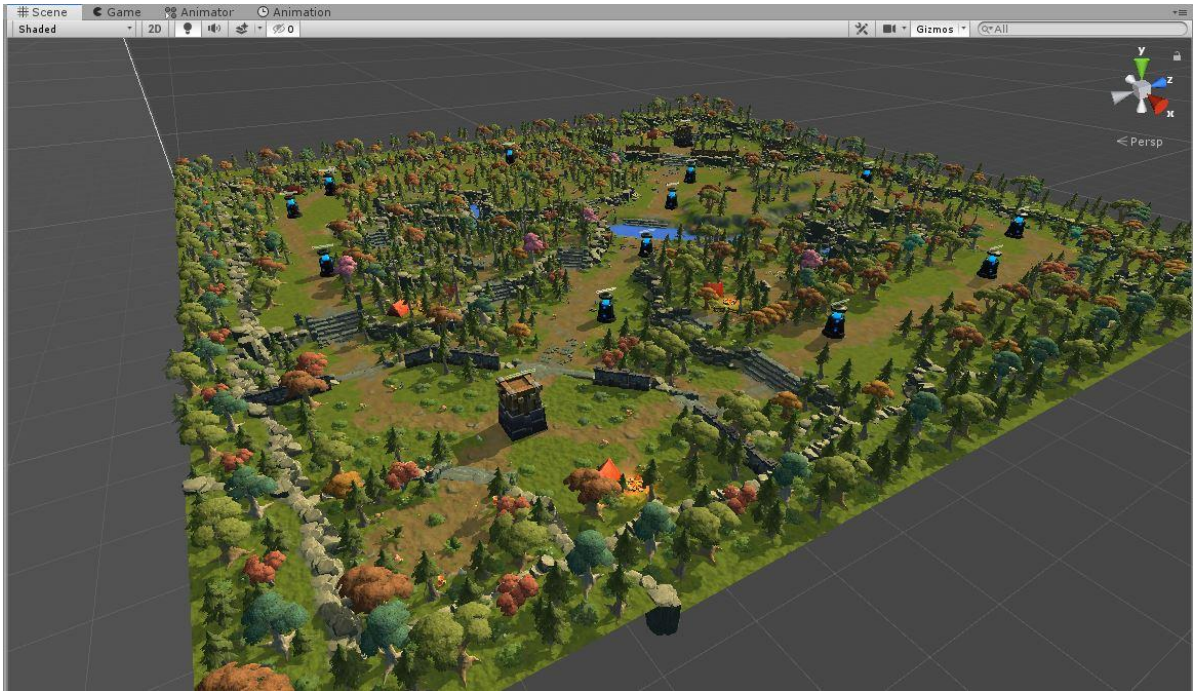
- Heroj igrača (eng. Player avatar)
- Magije (eng. Spells)
- Modeli heroja (eng. Avatar model)



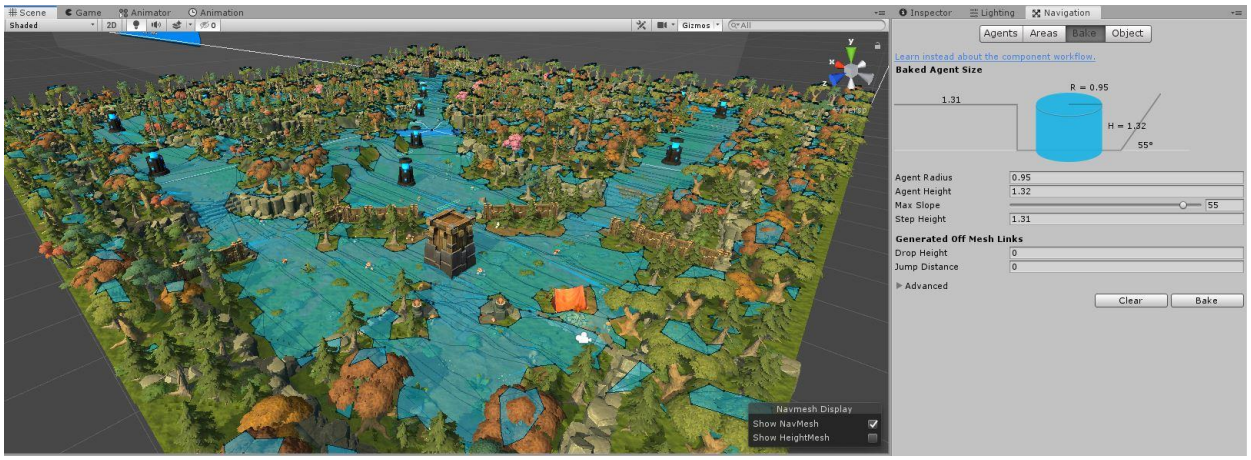
Slika 4. Modeli i objekti igre

3.2.1. Teren

Teren je podijeljen na dva dijela od kojih je jedna strana Ancient, a druga Dire. Strane su dijagonalno razdvojene rijekom te su na svakoj polovici postavljeni tornjevi i glavni tornjevi. Također svaki tim ima svoju bazu na kojoj su postavljeni glavni tornjevi kao i unaprijed definirane lokacije u kojem se oživljavaju minioni kao i heroji igrača. Ono što je najvažnije nakon izrade terena je kreiranje navigacije koja će omogućiti kretnju svih živih objekata na istom. Za kreiranje navigacije potrebno je otvoriti prozor za navigaciju i postaviti postavke kreiranja navigacije (eng. *Bake*). Nakon što su postavke postavljene potrebno je kliknuti na gumb kreiraj navigaciju (eng. *Bake*), time je navigacija kreirana i prikazana je plavom bojom unutar scene.



Slika 5. Teren



Slika 6. Navigacija terena

3.2.1.1. Tornjevi

Skripte glavnih tornjeva nemaju nikakvu implementaciju osim što klase proširuju klasu koja se naziva *LivingEntity*. Navedenu klasu proširuju sve klase koje bi trebale predstavljati život odgovarajućem objektu igre. Uništenjem glavnog tornja uništava se objekt igre, zatim se provjerava njegova oznaka i ukoliko oznaka sadržava u nazivu *Ancient,* tada se provjerava glavni naziv tornja na temelju kojeg se kreira objekt završetka igre (eng. Eng game) i prosljeđuju se parametri koji predstavljaju kraj igre i proglašenje pobjednika. Što se tiče skripata tornjeva za obranu, sastoje se od dva elementa: glavna skripta kontrola tornja i kontrola projektila tornja. Nakon što se inicijaliziraju varijable potrebno je pozvati posebnu metodu *InvokeRepeating* koja omogućuje pozivanje metode slične kao *Update,* ali razlika je da navedena metoda prima tri parametra: naziv metode za izvođenje, početno vrijeme poziva metode za izvođenje i ono što je najbitnije brzina osvježavanja metode. Pozivom metode za izvođenje pretražuju se svi živi neprijateljski objekti na sceni za odgovarajućom oznakom te se provjerava udaljenost između neprijatelja i tornja, ukoliko su neprijatelji unutar distance pucnja tornja tada će se kreirati novi objekt, odnosno instanca klase projektila tornja kojem će se proslijediti objekt neprijatelja i objekt projektila će pratiti neprijatelja sve dok istog ne pogodi pri čemu će proizvesti određenu štetu. U suprotnom, ako toranj nije locirao miniona tada će provjeravati da li je igrač unutar distance pucnja te će isto učiniti za igrača.

Slika 7. Tornjevi i glavni tornjevi

```
public class DireTowerController : LivingEntity
{
    private Transform target;
    LivingEntity live;
    Transform tower;
```

```
    [Header("Attributes")]
    public float range = 15f;
    public float fireRate = 10f;
    private float fireCountdown = 0f;

    [Header("Unity Setup Fields")]
    public string enemyTag = "EnemyRadiant";
    public GameObject bulletPrefab;
    public Transform firePoint;

    public override void Start()
    {
        base.Start();
        tower = GameObject.FindGameObjectWithTag("DireTower").transform;
        live = tower.GetComponent<LivingEntity>();
        InvokeRepeating("UpdateTarget", 0f, 0.5f);
    }

    void UpdateTarget()
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        float shortestDistance = Mathf.Infinity;
        GameObject nearestEnemy = null;
        foreach (GameObject enemy in enemies)
        {
            float distanceToEnemy = Vector3.Distance(transform.position,
enemy.transform.position);
            if (distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemy;
            }
        }

        GameObject player = GameObject.FindGameObjectWithTag("Player_0");
        if (player)
        {
            float distanceToPlayer = Vector3.Distance(transform.position,
player.transform.position);
            if (distanceToPlayer < shortestDistance)
            {
```

```
                shortestDistance = distanceToPlayer;
                nearestEnemy = player;
            }
        }

        if (nearestEnemy != null && shortestDistance <= range)
        {
            target = nearestEnemy.transform;
        } else
        {
            target = null;
        }
    }

    void Update()
    {
        if (target == null)
        {
            return;
        }

        if (fireCountdown <= 0)
        {
            Shoot();
            fireCountdown = 1f / fireRate;
        }

        fireCountdown -= Time.deltaTime;

    }

    void Shoot()
    {
        GameObject bulletGO = (GameObject)Instantiate(bulletPrefab,
firePoint.position, firePoint.rotation);
        DireTowerProjectileController bullet =
bulletGO.GetComponent<DireTowerProjectileController>();

        if (bullet != null)
        {
            bullet.Seek(target);
```

```
        }
    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, range);
    }
}

public class DireTowerProjectileController : MonoBehaviour
{

    Transform target;
    public float speed = 70f;
    private float damage = 50;
    LivingEntity targetEntity;

    public void Seek(Transform _target)
    {
        target = _target;
    }

    void Update()
    {
        if (target == null)
        {
            Destroy(gameObject);
            return;
        }

        Vector3 dir = target.position - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;

        if (dir.magnitude <= distanceThisFrame)
       {
           HitTarget();
           return;
       }

       transform.Translate(dir.normalized * distanceThisFrame, Space.World);
    }
```

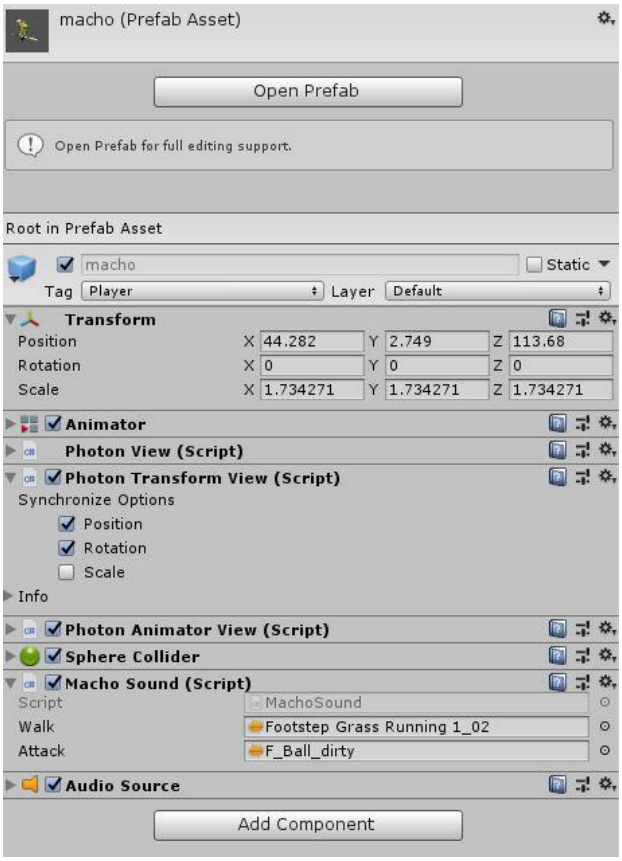
```
    void HitTarget()
    {
        targetEntity = target.GetComponent<LivingEntity>();
        if (targetEntity)
        {
            targetEntity.TakeDamage(damage, target.tag);
        }

        Destroy(gameObject);
    }

}
```

3.2.2. Glavni igrač

Svakome igraču dodijeljen je objekt heroja koji predstavlja svakoga korisnika spojenog na server. Model heroja sastoji se od nekoliko skripata potrebnih za realizaciju kontrola (kretnja, pucanje, karakteristike, …). Heroji se razlikuju po izgledu, magijama i karakteristikama, te svaki prethodno odabrani heroj iz glavnom izbornika nadovezuje se na objekt heroja i moguć je odabir između 3 heroja: Macho, Spider i Arthur. Kako bih animacije i zid igrača (eng. *Collider*) bile sinkronizirane kroz server potrebno je svakome modelu heroja dodijeliti posebne skripte kreirane putem *Photon* proširenja *Photon Transform View* i *Photon Animator View* koji omogućuju odabir postavki za sinkronizaciju.



Slika 8. Photon sinkronizacija pozicije i rotacije modela

Ono što čini heroja živim su komponente dodijeljene objektu heroja: zid objekta (eng. *Collider*), tijelo objekta (eng. *Rigidbody*), navigacija heroja (eng. *Nav Mesh Agent*) te skripte postavke heroja (eng. Avatar setup), magije (eng. Abilities), kontrola igrača (eng. *Player controller*), iskustvo igrača (eng. *Player experience*). Prethodno je objašnjeno na koji način se kreira instanca klase postavke heroja pa time započinje izvršavanje iste. Postavljaju se vrijednost varijabla putem inspektora na objektu igre te se putem Photon metode postavlja vrijednost indeksa odabranog heroja. Samim time postavljaju se ostale karakteristike: model heroja, pozicija i rotacija heroja, animacija, lokacija za ispaljivanje magije i običnog napada, korisničko sučelje na temelju odabranog heroja i tim igrača.

Glavna skripta potrebna za kretanje, pokretanje animacije i pucnjavu je skripta kontrole igrača (eng. Player controller). Poprilično je velika te se najprije inicijaliziraju osnove varijable poput početne vrijednost štete običnog napada, magije, bonus štete na magiju, nivo, odgoda pucnja nakon korištenja magije, oznake heroja, oznake neprijatelja, komponente korisničkog sučelja, …

Osim postavljanja osnovnih vrijednost pokreću se dvije metode putem Unity funkcije *StartCoroutine* koje služe za pokretanje *IEnumerator* tipa objekta. Ovakav tip objekta koristi se kada se javi potreba za pozivanje određene funkcije kroz vrijeme, pa se tako prilikom inicijalizacije varijabli pokreće metoda za uvećavanje vrijednosti štete napada i magije prilikom postignuća novog nivoa heroja i metoda za uvećavanje iskustva tokom vremena (20 dodatnog iskustva svakih 20 sekundi). Nakon što su vrijednost postavljanje putem *Start* metode pokreće se *Update* metoda te se provjerava igračev život, mana i sve ostale vrijednost koje će se mijenjati prilikom korištenja magija, iskustva i gubitka života kako bih se mogle ažurirate vrijednost prikazane na korisničkom sučelju. Osim karakteristika, također se provjeravaju pritisci tipka na tipkovnici ili klikom na određeni gumb na mišu. Ukoliko se desnim klikom miša klikne na bilo koju lokaciju na terenu heroj će putem funkcije za kretanje po navigaciji (eng. *SetDirection*) krenuti prema odabranoj lokaciji, u suprotnom ukoliko se klikne na protivnika, provjerava se distanca između pozicije heroja i odabranog objekta kao i oznaka. Većom distancom od distance dometa osnovnog napada, heroj će krenuti prema protivniku također koristeći *StartCoroutine* funkciju i ukoliko će protivnik biti u dometu osnovnog napada tada će se heroj prestati kretati i ispaliti će se metak prema protivniku. Nužno je bilo kreirati u istoj funkciji provjeru odgađanja pucnja tako da heroj ne može ispaljivati metke prilikom svakog tipka desnim klikom miša.

Osim osnovnog napada, heroj ima dva dvije magije koje se pokreću tipkom na tipkovnicu „W“ ili „Q“ po standardnom obliku za ispaljivanje magije u MOBA igrama. Također u *Update* metodi provjerava se pritiska tipka tipkovnice i pritiskom na iste dozvoljava se pritisak na lijevu tipku miša. Odabirom magije „Q“ ili osnovnog napada pokreću se metoda u kojima se

provjerava određeni indeks heroja. Na temelju *Switch* uvjeta provjerava se indeks heroja i kreira se instanca klase skripte moći (eng. *Abilities*) koja vraća objekt igre, odnosno metak ovisno o indeksu heroja i vrsti magije (obični napad, magija „Q“, magija „W“). Uspješnim kreiranjem instance poziva se skripta sa kreiranog objekta za ispaljivanje magije ili običnog napada. Magija „Q“ i obični napadaj radi na sličan princip kao i ispaljivanje kod tornja, odnosno kreirani objekt će pratiti protivnika sve dok mu ne napravi štetu. Za razliku od navedenih načina, magija „W“ ispaljuje se od pozicije heroja prema odabranoj poziciji pritiskom lijeve tipke miša te će putem trake (eng. *Raycast*) provjeriti da li je pogodio objekt sa određenim slojem i pogotkom sloja sa nazivom *Shootable* objekt će primiti štetu.

```
public class AvatarSetup : MonoBehaviour
{

    private PhotonView PV;
    public int characterValue;
    public GameObject myCharacter;
    public Animator animator;
    public Transform spellCastPoint;
    public GameObject spellQ;
    public int myTeam;
    public float playerHealth;
    public int playerDamage;

    void Start()
    {
        PV = GetComponent<PhotonView>();
        animator = GetComponent<Animator>();
        if (animator == null)
        {
            animator = GetComponent<Animator>();

        }
        if (PV.IsMine)
        {
            PV.RPC("RPC_AddCharacter", RpcTarget.AllBuffered,
PlayerInfo.PI.mySelectedCharacter);
        }
    }
```

```
    private void Update()
    {
        if (PV.IsMine)
        {
            if (playerHealth <= 0)
            {
                PhotonNetwork.Destroy(gameObject);
                return;
            }
        }
    }

    [PunRPC]
    void RPC_AddCharacter(int whichCharacter)
    {
        characterValue = whichCharacter;
        myCharacter =
PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
PlayerInfo.PI.allCharacters[whichCharacter]), transform.position,
transform.rotation);
        myCharacter.transform.parent = transform;
        animator = myCharacter.GetComponent<Animator>();
        spellCastPoint = myCharacter.transform.Find("SpellCastPoint");
        GameSetup.GS.setupUI(characterValue);
        myTeam = GameSetup.GS.nextPlayerTeam;

    }
      }
public class PlayerController : LivingEntity
      {
          NavMeshAgent agent;
          public float rotateSpeedMovement = 0.1f;
          float rotateVelocity;
          private PhotonView PV;
          private AvatarSetup avatarSetup;
          Transform player;
          Abilities ability;
          bool clicked;
          public LayerMask shootableMask;

          enum PlayerState { WALK, IDLE, SPELL_Q, AUTOATTACK, SPELL_W }
```

```
        PlayerState playerState;

        enum spellType { SPELL_Q, SPELL_W, SPELL_E, SPELL_R, NO_SPELL,
AUTO_ATTACK };
        spellType currentSpell;

        private Transform oldTarget;

        private string enemyTag;
        private string playerTag;
        private string enemyPlayerTag;
        private string enemyTowerTag;
        private string enemyAncientTag;

        //machoAttackSpeed
        public float attackDelay = 1.0f;
        private float nextDamageEvent;

        //spiderPlayerAttack
        private bool dealDmg = false;
        private float respawnTimer = 0;

        //PlayerStats
        public float playerDamage = 25;
        public float spellBonusDamage = 25;
        public float playerMana = 340;
        public float playerStartingMana = 340;
        PlayerExperience playerExperience;
        Text playerLevelText;
        Text playerDmgText;
        Text playerSpellDmgText;
        Text playerHealthText;
        Text playerManaText;
        Text playerCooldownText;
        Slider playerHealthSlider;
        Slider playerManaSlider;
        private float tempLevel;
        private float playerSpellDmg;

        private float cooldownSpell = 30;
        private float spellTimer = 0;
```

```
        public override void Start()
        {
            base.Start();
            PV = GetComponent<PhotonView>();
            agent = gameObject.GetComponent<NavMeshAgent>();
            avatarSetup = GetComponent<AvatarSetup>();
            player = GameObject.FindGameObjectWithTag("Player").transform;
            ability = GetComponent<Abilities>();
            clicked = false;
            currentSpell = spellType.NO_SPELL;

            playerExperience = GetComponent<PlayerExperience>();
            playerLevelText                                          =
GameObject.Find("PlayerLevel").GetComponent<Text>();
            playerDmgText                                            =
GameObject.Find("PlayerDamage").GetComponent<Text>();
            playerSpellDmgText                                       =
GameObject.Find("PlayerBonusSpellDamage").GetComponent<Text>();
            playerHealthText                                         =
GameObject.Find("PlayerHealthText").GetComponent<Text>();
            playerManaText                                           =
GameObject.Find("PlayerManaText").GetComponent<Text>();
            playerHealthSlider                                       =
GameObject.Find("PlayerHealthSlider").GetComponent<Slider>();
            playerManaSlider                                         =
GameObject.Find("PlayerManaSlider").GetComponent<Slider>();
            playerCooldownText                                       =
GameObject.Find("PlayerCooldownText").GetComponent<Text>();

            playerHealthSlider.maxValue = health;
            playerHealthSlider.value = health;
            playerLevelText.text                                     =
playerExperience.playerLevel.ToString();
            playerDmgText.text = playerDamage.ToString();
            playerSpellDmg = playerDamage + spellBonusDamage;
            playerSpellDmgText.text = playerSpellDmg.ToString();
            playerMana = playerStartingMana;
            playerManaSlider.maxValue = playerMana;
            playerManaSlider.value = playerMana;
            playerHealthText.text  =  health.ToString()  +  "  /  "  +
startingHealth.ToString();
            playerManaText.text  =  playerMana.ToString()  +  "  /  "  +
playerStartingMana.ToString();
```

```
    playerCooldownText.text = spellTimer.ToString();

    tempLevel = playerExperience.playerLevel;

    if (avatarSetup.myTeam == 2)
    {
        enemyTag = "EnemyDire";
        playerTag = "Player_0";
        enemyTowerTag = "DireTower";
        enemyAncientTag = "DireAncient";
    }
    if (avatarSetup.myTeam == 1)
    {
        enemyTag = "EnemyRadiant";
        playerTag = "Player_1";
        enemyTowerTag = "RadiantTower";
        enemyAncientTag = "RadiantAncient";
    }
    gameObject.tag = playerTag;
    if (playerTag.Equals("Player_0"))
    {
        enemyPlayerTag = "Player_1";
    }
    if (playerTag.Equals("Player_1"))
    {
        enemyPlayerTag = "Player_0";
    }
    StartCoroutine(PlayerStatsIncreaseOverTime());
    StartCoroutine(PlayerExperienceIncreaseOverTime());
}

void Update()
{
    if (PV.IsMine)
    {
        if (playerMana <= 0)
        {
            playerMana = 0;
        }
        if (playerExperience.playerLevel > tempLevel)
```

```
                {
                    startingHealth += 40;
                    playerStartingMana += 40;
                    playerDamage += 20;
                    playerSpellDmg += playerDamage;
                    playerHealthSlider.maxValue = startingHealth;
                    playerManaSlider.maxValue = playerStartingMana;
                    tempLevel = playerExperience.playerLevel;
                }
                if (spellTimer <= 0)
                {
                    spellTimer = 0;
                }
                else
                {
                    spellTimer -= Time.deltaTime;
                }
                playerCooldownText.text = ((int) spellTimer).ToString();
                playerHealthSlider.value = health;
                playerManaSlider.value = playerMana;
                playerLevelText.text                                    =
playerExperience.playerLevel.ToString();
                playerDmgText.text = playerDamage.ToString();
                playerSpellDmgText.text = playerSpellDmg.ToString();
                playerHealthText.text  =  health.ToString()  +  "  /  "  +
startingHealth.ToString();
                playerManaText.text  =  playerMana.ToString()  +  "  /  "  +
playerStartingMana.ToString();
                RaycastHit hit;

                if (Input.GetMouseButtonDown(1))
                {
                    if
(Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit,
Mathf.Infinity))
                    {
                        agent.isStopped = false;
                        if     (hit.transform.tag     ==     enemyTag     ||
hit.transform.tag == enemyPlayerTag || hit.transform.tag == enemyTowerTag ||
hit.transform.tag == enemyAncientTag)
                        {
```

```
StartCoroutine(AutoAttackCoroutine(hit.transform, playerExperience));
                        }
                        else
                        {
                            agent.SetDestination(hit.point);
                            playerState = PlayerState.WALK;
                        }

                        Quaternion           rotationToLookAt           =
Quaternion.LookRotation(hit.point - transform.position);
                        float                 rotationY                 =
Mathf.SmoothDampAngle(transform.eulerAngles.y,
                            rotationToLookAt.eulerAngles.y,
                            ref rotateVelocity,
                            rotateSpeedMovement * (Time.deltaTime * 5));
                        transform.eulerAngles = new Vector3(0, rotationY,
0);
                    }

                }

                if (Input.GetKeyDown(KeyCode.Q))
                {
                    clicked = true;
                    currentSpell = spellType.SPELL_Q;
                }
                if (Input.GetKeyDown(KeyCode.W))
                {
                    clicked = true;
                    currentSpell = spellType.SPELL_W;

                }
                if (clicked)
                {
                    if (Input.GetMouseButtonDown(0))
                    {


                        if    (currentSpell.Equals(spellType.SPELL_Q)    &&
playerMana >= 60)
                        {
```

```
                                if (spellTimer <= 0)
                                {
                                    if
(Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit,
Mathf.Infinity))
                                    {

                                        if (hit.transform.tag == enemyTag ||
hit.transform.tag == enemyPlayerTag)
                                        {

                                            if
(Vector3.Distance(player.position, hit.transform.position) < 20)
                                            {
                                                spellTimer = cooldownSpell;
                                                agent.isStopped = true;
                                                currentSpell                =
spellType.AUTO_ATTACK;
                                                SpellQ(hit.transform,
hit.transform.tag);
                                            }
                                        }
                                    }

                            }
                            if (currentSpell.Equals(spellType.SPELL_W))
                            {
                                if
(Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit,
Mathf.Infinity))
                                {
                                    if      (Vector3.Distance(player.position,
hit.point) < 30)
                                    {
                                        SpellW(hit);
                                    }
                                }
                            }
                        }

                        if (Input.GetMouseButtonDown(1))
```

```
                    {
                        clicked = false;
                    }
                }

                float speed = agent.velocity.magnitude / agent.speed;
                string animationAction;
                if (avatarSetup.characterValue == 0)
                {
                    animationAction = "speedv";
                    avatarSetup.animator.SetFloat(animationAction,
speed);
                }
                if (avatarSetup.characterValue == 1)
                {
                    animationAction = "Speed";
                    avatarSetup.animator.SetFloat(animationAction,
speed);
                }

                if (avatarSetup.characterValue == 2)
                {
                    if (speed > 0)
                    {
                        avatarSetup.animator.Play("arthur_walk_01", 0);
                    }

                }

            }

        }

        IEnumerator PlayerStatsIncreaseOverTime()
        {
            while (true)
            {
                yield return new WaitForSeconds(5);
                if (health < startingHealth)
                {
                    health += 20;
```

```
                }
                if (playerMana < playerStartingMana)
                {
                    playerMana += 10;
                }
            }
        }

        IEnumerator PlayerExperienceIncreaseOverTime()
        {
            while (true)
            {
                yield return new WaitForSeconds(20);
                playerExperience.playerExperience += 20;
            }
        }

        IEnumerator AutoAttackCoroutine(Transform target, PlayerExperience
_playerExperience)
        {
            while (target)
            {
                Vector3 dirToTarget = (target.transform.position -
transform.position).normalized;
                Vector3 targetPosition = target.transform.position -
dirToTarget;
                agent.SetDestination(targetPosition);
                if (Vector3.Distance(player.position, targetPosition) <
10)
                {
                    agent.isStopped = true;
                    if (Time.time >= nextDamageEvent)
                    {
                        nextDamageEvent = Time.time + attackDelay;
                        currentSpell = spellType.AUTO_ATTACK;
                        AutoAttack(target, target.tag, _playerExperience);
                    }
                }
                else
                {
                    nextDamageEvent = Time.time + attackDelay;
                }
```

```
                yield return new WaitForSeconds(0.0001f);
                if (Input.GetMouseButtonDown(1))
                {
                    yield break;
                }
            }
        }

        private void AutoAttack(Transform target, string tag,
PlayerExperience _playerExperience)
        {
            switch (avatarSetup.characterValue)
            {
                case 0:
                    avatarSetup.animator.SetTrigger("Active");
                    GameObject autoAttack =
(GameObject)Instantiate(ability.GetSpell("AutoAttack",
avatarSetup.characterValue), avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                    MachoAutoAttack projectile =
autoAttack.GetComponent<MachoAutoAttack>();
                    if (projectile != null)
                    {
                        projectile.Seek(_playerExperience, target, tag,
playerDamage);
                    }
                    break;
                case 1:
                    avatarSetup.animator.SetFloat("multiplier", 10);
                    avatarSetup.animator.SetTrigger("Attack");
                    GameObject autoAttackSpider =
(GameObject)Instantiate(ability.GetSpell("AutoAttack",
avatarSetup.characterValue), avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                    MachoAutoAttack projectileSpider =
autoAttackSpider.GetComponent<MachoAutoAttack>();
                    if (projectileSpider != null)
                    {
                        projectileSpider.Seek(_playerExperience, target,
tag, playerDamage);
                    }
                    break;
                case 2:
                    avatarSetup.animator.Play("arthur_active_01", 0);
```

```
                    GameObject              autoAttackArthur              =
(GameObject)Instantiate(ability.GetSpell("AutoAttack",
avatarSetup.characterValue),           avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                    MachoAutoAttack           projectileArthur           =
autoAttackArthur.GetComponent<MachoAutoAttack>();
                    if (projectileArthur != null)
                    {
                        projectileArthur.Seek(_playerExperience,   target,
tag, playerDamage);
                    }
                    break;
                }
            }

            private void SpellQ(Transform target, string tag)
            {
                switch (avatarSetup.characterValue)
                {
                    case 0:
                        GameObject                  spellQ                  =
(GameObject)Instantiate(ability.GetSpell("SpellQ",
avatarSetup.characterValue),           avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                        MachoAutoAttack             projectile              =
spellQ.GetComponent<MachoAutoAttack>();
                        if (projectile != null)
                        {
                            playerMana -= 60;
                            avatarSetup.animator.SetTrigger("Active");
                            projectile.Seek(playerExperience,   target,   tag,
playerSpellDmg);
                        }
                        break;
                    case 1:
                        GameObject                spellQSpider                =
(GameObject)Instantiate(ability.GetSpell("SpellQ",
avatarSetup.characterValue),           avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                        MachoAutoAttack           projectileSpider           =
spellQSpider.GetComponent<MachoAutoAttack>();

                        if (projectileSpider != null)
                        {
                            playerMana -= 60;
```

```
                        avatarSetup.animator.SetFloat("multiplier", 10);
                        avatarSetup.animator.SetTrigger("Attack");
                        projectileSpider.Seek(playerExperience,    target,
tag, playerSpellDmg);
                    }
                    break;
                case 2:
                    GameObject                spellQArthur                =
(GameObject)Instantiate(ability.GetSpell("SpellQ",
avatarSetup.characterValue),           avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
                    MachoAutoAttack           projectileArthur            =
spellQArthur.GetComponent<MachoAutoAttack>();

                    if (projectileArthur != null)
                    {
                        playerMana -= 60;
                        avatarSetup.animator.Play("arthur_active_01", 0);
                        projectileArthur.Seek(playerExperience,    target,
tag, playerSpellDmg);
                    }
                    break;

            }

        }

        private void SpellW(RaycastHit hit)
        {
            GameObject                        spellW                        =
(GameObject)Instantiate(ability.GetSpell("SpellW",
avatarSetup.characterValue),           avatarSetup.spellCastPoint.position,
avatarSetup.spellCastPoint.rotation);
            MachoWSpell projectile = spellW.GetComponent<MachoWSpell>();
            if (projectile != null)
            {
                projectile.Seek(hit);
            }
        }

    }
```

```
public class PlayerExperience : MonoBehaviour
{

    public int playerNextLevel = 1;

    public int playerLevel = 0;

    public int playerExperience = 0;

    public int nextLevelExperience = 100;

    Slider playerExperienceSlider;

    Text playerExperienceText;

    private void Start()
    {
        playerExperienceSlider =
GameObject.Find("PlayerExperienceSlider").GetComponent<Slider>();
        playerExperienceText =
GameObject.Find("PlayerExperienceText").GetComponent<Text>();
        playerExperienceText.text = playerExperience.ToString() + " / " +
nextLevelExperience.ToString();
        playerExperienceSlider.maxValue = nextLevelExperience;
        playerExperienceSlider.value = playerExperience;
    }

    private void Update()
    {
        playerExperienceSlider.value = playerExperience;
        if (playerExperience > nextLevelExperience)
        {
            playerLevel++;
            playerNextLevel++;
            playerExperienceSlider.minValue = playerExperience;
            nextLevelExperience += nextLevelExperience;
            playerExperienceSlider.maxValue = nextLevelExperience;
        }
        playerExperienceText.text = playerExperience.ToString() + " / " +
nextLevelExperience.ToString();
    }
```

```
    }
public class Abilities : MonoBehaviour
{
    [Header("Macho hero")]
    public GameObject machoAutoAttack;
    public GameObject machoSpellQ;
    public GameObject machoSpellW;

    [Header("Spider hero")]
    public GameObject spiderAutoAttack;
    public GameObject spiderSpellQ;
    public GameObject spiderSpellW;

    [Header("Arthur hero")]
    public GameObject arthurAutoAttack;
    public GameObject arthurSpellQ;
    public GameObject arthurSpellW;


    public GameObject GetSpell(string spellType, int heroId)
    {
        switch(spellType)
        {
            case "SpellQ":
                return spellQ(heroId);
            case "SpellW":
                return spellW(heroId);
            case "AutoAttack":
                return autoAttack(heroId);
            default:
                return null;
        }
    }

    private GameObject spellQ(int heroId)
    {
        if (heroId == 0)
        {
            return machoSpellQ;
        }
```

```
    if (heroId == 1)
    {
        return spiderSpellQ;
    }
    if (heroId == 2)
    {
        return arthurSpellQ;
    }
    return machoSpellQ;
}

private GameObject spellW(int heroId)
{
    if (heroId == 0)
    {
        return machoSpellW;
    }

    if (heroId == 1)
    {
        return spiderSpellW;
    }
    if (heroId == 2)
    {
        return arthurSpellW;
    }
    return machoSpellW;
}

private GameObject autoAttack(int heroId)
{
    if (heroId == 0)
    {
        return machoAutoAttack;
    }
    if (heroId == 1)
    {
        return spiderAutoAttack;
    }
    if (heroId == 2)
```

```
            {
                return arthurAutoAttack;
            }
            return machoAutoAttack;
        }
    }
```

3.2.3. Ponovno oživljavanje AI miniona

Početkom igre započinje odbrojavanje vremena te se kao što je prethodno opisano u tridesetoj sekundi oživljavaju protivnici kao i svakih sljedećih 30 sekundi sve dok nije proglašen pobjednik. U skripti za ponovno oživljavanje (eng. *Wave spawner*) pokreće se brojač vremena te se ažurira vrijeme prikazano na korisničkom sučelju putem *Update* metode. Osim brojača vremena kreirana je varijabla tipa *Enum* koja omogućuje dva stanja prilikom oživljavanja miniona: oživljavanje (eng. *Spawning*) i čekanje (eng. *Waiting*). Koristeći već poznatu funkciju *StarCoroutine* provjeravaju se stanja te se nakon poziva funkcije unutar navedene oživljavaju minioni putem Photon funkcije za kreiranje instance kako bih isti bili sinkronizirani između svih igrača u igri. Nadalje, kreiranjem instance objekata igre miniona pokreću se skripte za Dire i Radiant AI protivnike. Slično kao i za tornjeve pretražuju se živi objekti prema prioritetu redom protivnički toranj, protivnički minioni, protivnički glavni toranj. Kroz *Update* funkciju provjerava se da li su minioni blizu jedni drugome te se prema tome promjeni smjer napadaja ukoliko je trenutni smjer postavljen prema smjeru tornja ili glavnog tornja. Samim time provjerava se udaljenost između pozicije miniona i protivnika te ukoliko je protivnik u dometu tada će preko *StartCoroutine* funkcije početi stanje napada i suprotno.

Slika 9. Radiant minion

Slika 10. Dire minion

```
public class WaveSpawner : MonoBehaviour
{

    public enum SpawnState { SPAWNING, WAITING };

    [System.Serializable]
    public class WaveDire
    {
        public string name;
        public Transform enemyDire;
    }

    [System.Serializable]
    public class WaveRadiant
    {
        public string name;
        public Transform enemyRadiant;
    }

    public WaveDire waveDire;
    public WaveRadiant waveRadiant;
    private int nextWave = 0;

    public float timeBetweenWaves = 0;
    private float waveCountdown;

    private SpawnState state = SpawnState.WAITING;

    public Transform[] spawnPoints;

    bool startSpawn = true;
    Text gameTimerText;
    float currentTime;


    void Start()
    {
        waveCountdown = timeBetweenWaves;
        gameTimerText =
GameObject.Find("GameTimerText").GetComponent<Text>();
    }
```

```
    void Update()
    {
        if (PhotonNetwork.IsMasterClient)
        {
            currentTime += Time.deltaTime;
            float minutes = Mathf.FloorToInt(currentTime / 60);
            float seconds = Mathf.FloorToInt(currentTime % 60);
            gameTimerText.text = string.Format("{0:00}:{1:00}", minutes,
seconds);
            waveCountdown -= Time.deltaTime;

            if (waveCountdown <= 0)
            {
                state = SpawnState.SPAWNING;
                waveCountdown = timeBetweenWaves;
            }
            else
            {
                if (state == SpawnState.SPAWNING && startSpawn)
                {
                    StartCoroutine(SpawnWave(waveDire, waveRadiant));
                }

            }
        }

    }

    IEnumerator SpawnWave(WaveDire _waveDire, WaveRadiant _waveRadiant)
    {
        state = SpawnState.SPAWNING;
        SpawnEnemy(_waveDire.enemyDire, _waveRadiant.enemyRadiant);
        state = SpawnState.WAITING;
        yield break;
    }

    void SpawnEnemy(Transform _enemyDire, Transform _enemyRadiant)
    {
```

```
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[0].position, spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[1].position, spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[2].position, spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[3].position, spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[4].position, spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[5].position, spawnPoints[1].rotation);


        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[0].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[1].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[2].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[3].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[4].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[5].position + new Vector3(0.0f, 0.0f, 1.0f),
spawnPoints[1].rotation);


        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[0].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[1].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[2].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[3].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[4].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[5].position + new Vector3(0.0f, 0.0f, 2.0f),
spawnPoints[1].rotation);
```

```
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[0].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[1].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[2].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[3].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[4].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[5].position + new Vector3(0.0f, 0.0f, 3.0f),
spawnPoints[1].rotation);


        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[0].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[1].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[2].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[3].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[1].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyDire"), spawnPoints[4].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[0].rotation);

        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",
"EnemyRadiant"), spawnPoints[5].position + new Vector3(0.0f, 0.0f, 4.0f),
spawnPoints[1].rotation);


    }


}


public class EnemyRadiant : LivingEntity
{
    public static event System.Action OnDeathStatic;

    Animator anim;

    public enum State { Idle, Chasing, Attacking };

    public enum AttackPriority { ATTACK_PLAYER, ATTACK_CREEPS, ATTACK_TOWER
};
```

```
    State currentState;
    NavMeshAgent pathfinder;
    Transform target;
    LivingEntity targetEntity;
    AudioSource hurtPlayer;
    bool hasTarget;
    bool isSinking;
    float attackDistanceThreshold = 1f;
    float timeBetweenAttacks = 2;
    float damage = 15;
    float nextAttackTime;
    float myCollisionRadius;
    float targetCollisionRadius;

    bool changePathToCreeps = false;
    void Awake()
    {
        pathfinder = GetComponent<NavMeshAgent>();
        hurtPlayer = GetComponent<AudioSource>();
        GameObject[] enemies =
GameObject.FindGameObjectsWithTag("DireTower");
        GameObject[] enemyCreeps =
GameObject.FindGameObjectsWithTag("EnemyDire");
        GameObject enemyAncient =
GameObject.FindGameObjectWithTag("DireAncient");


        float shortestDistance = Mathf.Infinity;
        GameObject nearestEnemy = null;
        foreach (GameObject enemy in enemies)
        {
            float distanceToEnemy = Vector3.Distance(transform.position,
enemy.transform.position);
            if (distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemy;
            }
        }

        foreach (GameObject enemyCreep in enemyCreeps)
        {
```

```
            float distanceToEnemy = Vector3.Distance(transform.position,
enemyCreep.transform.position);
            if (distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemyCreep;
            }
        }

        if (nearestEnemy == null)
        {
            nearestEnemy = enemyAncient;
        }

        if (nearestEnemy != null)
        {
            target = nearestEnemy.transform;
            hasTarget = true;
            targetEntity = target.GetComponent<LivingEntity>();
            myCollisionRadius = GetComponent<CapsuleCollider>().radius;
            targetCollisionRadius = GetComponent<CapsuleCollider>().radius;
        }


    }
    public override void Start()
    {
        base.Start();
        anim = GetComponent<Animator>();
        hasTarget = false;
        if (hasTarget && target != null)
        {
            currentState = State.Chasing;
            targetEntity.OnDeath += OnTargetDeath;
            StartCoroutine(UpdatePath());
        }
    }

    public override void TakeHit(float damage, string tag)
    {
        if (damage >= health)
```

```
        {
            if (OnDeathStatic != null)
            {
                OnDeathStatic();
            }
        }
        base.TakeHit(damage, tag);
    }
    void OnTargetDeath()
    {
        hasTarget = false;
        currentState = State.Idle;
    }

    void FixedUpdate()
    {

        bool creepsAreNearest = CheckCloseToTag("EnemyDire", 10);
        if (creepsAreNearest && changePathToCreeps == false)
        {
            hasTarget = false;
        }
        if (hasTarget && target != null)
        {
            if (Time.time > nextAttackTime)
            {
                float sqrDstToTarget = (target.position -
transform.position).sqrMagnitude;
                if (sqrDstToTarget < Mathf.Pow(attackDistanceThreshold +
myCollisionRadius + targetCollisionRadius + 1, 2))
                {
                    GetComponent<Rigidbody>().velocity = Vector3.zero;
                    nextAttackTime = Time.time + timeBetweenAttacks;
                    StartCoroutine(Attack());
                }
            }
        }
        else
        {

            pathfinder = GetComponent<NavMeshAgent>();
```

```
            hurtPlayer = GetComponent<AudioSource>();
            GameObject[] enemies =
GameObject.FindGameObjectsWithTag("DireTower");
            GameObject[] enemyCreeps =
GameObject.FindGameObjectsWithTag("EnemyDire");
            GameObject enemyAncient =
GameObject.FindGameObjectWithTag("DireAncient");
            float shortestDistance = Mathf.Infinity;
            GameObject nearestEnemy = null;
            foreach (GameObject enemy in enemies)
            {
                float distanceToEnemy =
Vector3.Distance(transform.position, enemy.transform.position);
                if (distanceToEnemy < shortestDistance)
                {
                    shortestDistance = distanceToEnemy;
                    nearestEnemy = enemy;
                }
            }

            foreach (GameObject enemyCreep in enemyCreeps)
            {
                float distanceToEnemy =
Vector3.Distance(transform.position, enemyCreep.transform.position);
                if (distanceToEnemy < shortestDistance)
                {
                    shortestDistance = distanceToEnemy;
                    nearestEnemy = enemyCreep;
                    changePathToCreeps = true;
                }
            }

            if (nearestEnemy == null)
            {
                nearestEnemy = enemyAncient;
            }

            if (nearestEnemy != null)
            {
                target = nearestEnemy.transform;
                hasTarget = true;
                targetEntity = target.GetComponent<LivingEntity>();
```

```
                myCollisionRadius = GetComponent<CapsuleCollider>().radius;
                targetCollisionRadius =
GetComponent<CapsuleCollider>().radius;
                if (hasTarget && target != null && gameObject &&
targetEntity.doUpdate)
                {
                    currentState = State.Chasing;
                    targetEntity.OnDeath += OnTargetDeath;
                    StartCoroutine(UpdatePath());
                }
            }
        }
    }

    IEnumerator Attack()
    {
        anim.Play("Attack01");
        currentState = State.Attacking;
        pathfinder.enabled = false;
        Vector3 originalPosition = transform.position;
        Vector3 dirToTarget = (target.position -
transform.position).normalized;
        Vector3 attackPosition = target.position - dirToTarget *
(myCollisionRadius);

        float attackSpeed = 3;
        float percent = 0;
        bool hasAppliedDamage = false;
        while (percent <= 1)
        {
            if (percent >= .5f && !hasAppliedDamage)
            {
                hasAppliedDamage = true;
                targetEntity.TakeDamage(damage, target.tag);
            }
            percent += Time.deltaTime * attackSpeed;
            yield return null;
        }
        currentState = State.Chasing;
        pathfinder.enabled = true;
    }
```

```
    IEnumerator UpdatePath()
    {

        float refreshRate = .25f;
        while (hasTarget && target != null)
        {
            pathfinder.enabled = true;
            if (currentState == State.Chasing)
            {
                anim.Play("Run");
                Vector3 dirToTarget = (target.position -
transform.position).normalized;
                Vector3 targetPosition = target.position - dirToTarget *
(myCollisionRadius + targetCollisionRadius);
                if (!dead)
                {

                    pathfinder.SetDestination(targetPosition);
                }
            }
            yield return new WaitForSeconds(refreshRate);
        }
    }

    bool CheckCloseToTag(string tag, float minimumDistance)
    {
        GameObject[] goWithTag = GameObject.FindGameObjectsWithTag(tag);

        for (int i = 0; i < goWithTag.Length; ++i)
        {
            if (Vector3.Distance(transform.position,
goWithTag[i].transform.position) <= minimumDistance)
                return true;
        }

        return false;
    }

    private void OnCollisionEnter(Collision collision)
    {
```

```
            GetComponent<Rigidbody>().isKinematic = true;
        }
}
```

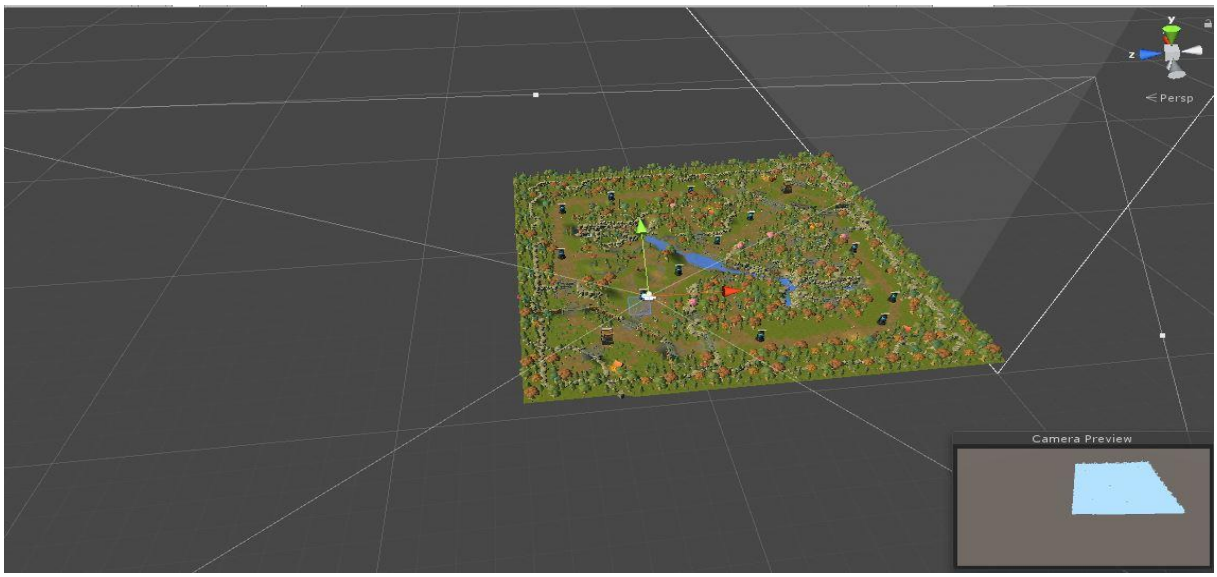
3.2.4. Glavna kamera

Kamera u sceni igre preuzeta je putem *GitHub*-a u svrhu lakše konfiguracije. Postavljene su razne postavke koje omogućuju različite poglede prema scene. Konfiguracija je postavljena prema konfiguraciji MOBA igre te omogućuje kretnju kamere kada se mišem prijeđe na granicu zaslona x i y koordinate dvodimenzionalnog modela.

Slika 11. Glavna kamera

3.2.5. Kamera života živih objekata

Kamera života predstavlja kameru čija je rotacija i pozicija usmjerena prema klizaču koji predstavlja život živih objekata. Skripta kamere nalazi se na svakom objektu sa životom osim igrača te se rotira u smjeru živih objekata tako da korisnik uvijek ima uvid na trenutno stanje života.



Slika 12. Kamera života živih objekata

```
public class HealthCanvas : MonoBehaviour
{

    private void LateUpdate()
    {
        Camera cam =
GameObject.Find("HealthCamera").GetComponent<Camera>();
        transform.LookAt(cam.transform);
        transform.Rotate(0, 180, 0);
    }
}
```

3.2.6. Korisničko sučelje heroja

Korisničko sučelje heroja sastoji se od klizača za života, mane i iskustva. Klizači se smanjuju odnosno povećavaju u odnosu na parametre koji su prethodno detaljno opisani. Nadalje, dvije slike pored slike heroja predstavljaju magiju te lijevo u kutu imaju brojač koji predstavlja da li je magija spremna za korištenje ili nije. Tekstualni dokument „dmg“ predstavlja trenutnu snagu običnog napada dok „spell“ predstavlja snagu magije.

Slika 13. Korisničko sučelje heroja

3.2.7. Kraj igre

Objekt kraj igre poziva se na početku igre te se onemogućuju sve komponente djeteta na temelju parametra koji predstavlja da li igra traje ili ne. Onemogućavanjem djeteta još uvijek je omogućen roditelj što omogućuje poziv objekta prilikom uništenja glavnom tornja te se vrijednost parametra postavi na istinu te će se igra zaustaviti, proglasiti pobjednika i omogućiti će se gumb putem kojeg se može izaći iz igre.

Slika 14. Korisničko sučelje kraja igre

```
public class ExitGame : MonoBehaviour
{

    public bool isEnded = false;

    public string winner = "";

    Text victoryText;

    private void Update()
    {
        if (isEnded)
        {
            for (int i = 0; i < gameObject.transform.childCount; i++)
            {
                GameObject child =
gameObject.transform.GetChild(i).gameObject;
                if (child != null)
                    child.SetActive(true);
            }
            victoryText =
GameObject.Find("VictoryText").GetComponent<Text>();
            victoryText.text = winner;
        } else
        {
            for (int i = 0; i < gameObject.transform.childCount; i++)
            {
                GameObject child =
gameObject.transform.GetChild(i).gameObject;
                if (child != null)
                    child.SetActive(false);
            }
        }
    }

    public void CloseGame()
    {
        Application.Quit();
    }
}
```

4. Zaključak

Rad je osmišljen i prikazan kroz teorijski i praktičan dio. Na detaljan način opisani su alati koje se koriste za izradu objekata, izgled i sučelje. Cilj ovog diplomskog rada je na temelju teorijskog i praktičnog znanja objediniti korištene programe i spojiti ih u cjelinu koja će predstavljati gotovu funkcionalni aplikaciju u ovom slučaju igru borbene arene sa više igrača. Modeli koji se koriste unutar igre preuzeti su putem *Unity Assets Store*-a. Za razvoj komercijalnih igara potrebno je kreirati vlastite terene, modele, heroje, mape, itd. Većina industrija koja se bavi razvojem igara imaju vlastite resurse u području developmenta i 3D modeliranja, te se na taj način ostvaruje brži i efikasniji rad. Ukoliko se pojavi želja za 3D postoje alati poput Maya-e i Autodesk 3D-s koji uz malo volje i truda mogu biti jako jednostavni za naučiti. Programski alat Unity je vrlo lagan za naučiti i veoma je „user friendly“. Svaka osoba koja bih htjela isprobati Unity trebala bi proći najprije kroz dokumentaciju jer se iz prve čini veoma zbunjujuće.

Igra borbene arene u ovom diplomskome radu zamišljena je kao prototip u smjeru Dota 2 igre. Usporedbom mapa, korisničkim sučeljem i tehnikama igre vrlo je nalik Dota-e te se ovime radom dokazuje da se uz malo truda, volje i znanja može implementirati, odnosno razviti gotova igra sa skoro svom funkcionalnošću kao *Dota* ili *League of Legends* kroz veoma kratko vrijeme i uz minimalni broj resursa.

Danas postoji velika količina dostupnih video materijala i opširno opisanih dokumentacija koja veoma olakšaju pristup bilo da se radi o programskom jeziku, alatu ili nešto treće. Već uz malu količinu znanja i volje moguće je samostalno izraditi komercijalnu igru. Zaključno najbitnija je upornost i ideja.

Popis literature

[1] Infogamerhub, Unity Multiplayer Tutorial Setup, https://www.infogamerhub.com/unity-multiplayer-tutorial-setup/ , Dostupno: 13. rujna 2020.

[2] Infogamerhub, Multiplayer Quick Start, https://www.infogamerhub.com/multiplayer-quick-start/ , Dostupno: 20. srpnja 2020.

[3] Infogamerhub, Multiplayer Delay Start, https://www.infogamerhub.com/unity-multiplayer-delay-start/ , Dostupno: 13. rujna 2020..

[4] Infogamerhub, Unity Multiplayer Custom Matchmaking, https://www.infogamerhub.com/unity-multiplayer-custom-matchmaking/ , Dostupno: 13. rujna 2020..

[5] Infogamerhub, Quick Start Matchmaking Photojn Add On, https://staging.infogamerhub.com/product/quick-start-matchmaking-photon-add-on/ , Dostupno: 13. rujna 2020.

[6] Infogamerhub, Unity Multiplayer Code Matchmaking, https://www.infogamerhub.com/unity-multiplayer-code-matchmaking/ , Dostupno: 13. rujna 2020.

[7] Infogamerhub, Set Up Pun 2, https://www.infogamerhub.com/set-up-pun-2-in-unity/ , Dostupno: 13. rujna 2020.

[8] Infogamerhub, Create multiplayer Rooms With Photon 2 In Unity, https://www.infogamerhub.com/create-multiplayer-rooms-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[9] Infogamerhub, Starting Multiplayer Games With Photon 2 In Unity, https://www.infogamerhub.com/starting-multiplayer-games-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[10] Infogamerhub, Instantiating Player Avatars With Photon 2 In Unity, https://www.infogamerhub.com/instantiating-player-avatars-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[11] Infogamerhub, Photon Transfomr View With Photon 2 In Unity, https://www.infogamerhub.com/photon-transform-view-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[12] Infogamerhub, RPC Functions With Photon 2 In Unity, https://www.infogamerhub.com/rpc-functions-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[13] Infogamerhub, Teams And Spawning With Photon 2 In Unity, https://www.infogamerhub.com/teams-and-spawning-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[14] Infogamerhub, Custom Matchmaking With Photon 2 In Unity, https://www.infogamerhub.com/custom-matchmaking-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[15] Infogamerhub, Syncing Animations With Photon 2 In Unity, https://www.infogamerhub.com/syncing-animations-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[16] Infogamerhub, Instantiating Player Avatars With Photon In Unity, https://www.infogamerhub.com/instantiating-player-avatars-with-photon-2-in-unity/ , Dostupno: 13. rujna 2020.

[17] Freecodecamp, Unity Game Engine Guide How To Get Started With The Most Popular Game Engine Out There, https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/ , Dostupno: 13. rujna 2020.

[18] J. Huizinga, The Power Of The Game Engine, https://prespective-software.com/the-power-of-the-game-engine/ , Objavljeno: 28. studenog 2019.

[19] Brackey, How To Make A Tower Defense Game E02-E0-5, https://www.youtube.com/watch?v=aFxucZQ_5E4&ab_channel=Brackeys , Dostupno: 13. rujna 2020.

[20] Unity, Manual, https://docs.unity3d.com/Manual/index.html , Dostupno: 13. rujna 2020.

Popis slika

Slika 1. Scena 3
Slika 2. Glavni izbornik 7
Slika 3. Scena glavnog izbornika 7
Slika 4. Modeli i objekti igre 19
Slika 5. Teren 20
Slika 6. Navigacija terena 20
Slika 7. Tornjevi i glavni tornjevi 21
Slika 8. Photon sinkronizacija pozicije i rotacije modela 25
Slika 9. Radiant minion Slika 10. Dire minion 43
Slika 11. Glavna kamera 54
Slika 12. Kamera života živih objekata 55
Slika 13. Korisničko sučelje heroja 56
Slika 14. Korisničko sučelje kraja igre 56