TEHNIČKA ŠKOLA RUĐERA BOŠKOVIĆA

Završni rad

Videoigra „ZR“

Učenik:

Adam Kolar

Mentor:

Zlatko Nadarević

Zagreb, travanj 2018.

Sadržaj

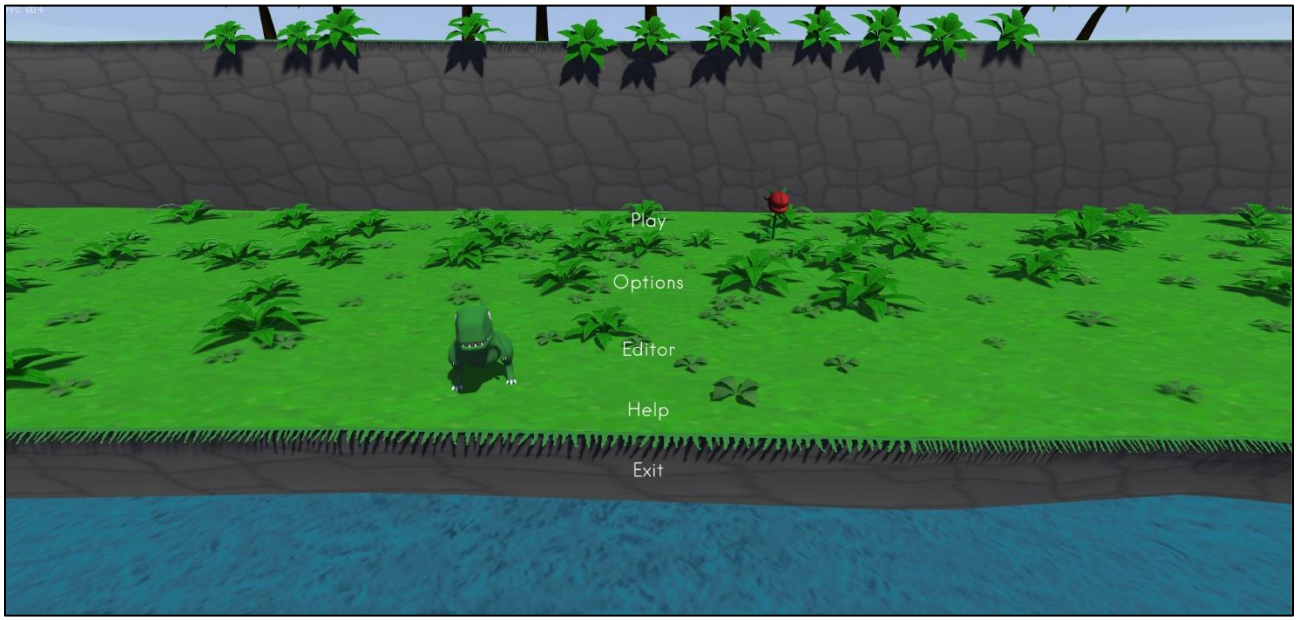
1. Uvod

U ovom ću završnom radu opisati način funkcioniranja videoigre „ZR“, to je igra tipa 3D platformer što znači da jedna od glavnih stvari u igri skakanje po platformama.

Igra ima 5 razina tzv. „level-a“ uz jednu dodatnu uz mogućnost izrade level-a pomoću level editor-a, u osnovi se level rješava na način da se iz početne pozicije dođe u krajnju prolazeći kroz prepreke i uništavajuću neprijatelje.

Igra je bazirana na Unity Game engine-u te je napisana u programskom jeziku C# u programu Visual Studio te koristi XML za pohranu podataka o objektima, level-ima te postavkama igre.

Za izradu grafike za igru, što su 3D modeli i teksture korišten je program Blender (za izradu 3D modela, teksturiranje, animiranje te renderiranje određenih ikona).



Slika 1 Izgled glavnog menija igre

2. Programiranje

Kao što je i prije spomenuto igra je pisana u potpunosti u programskom jeziku C#,

Kod je pisan u datoteke čije ime je ujedno i ime klase koju datoteka sadrži, gotovo sve klase osim GameData i LevelLoader su naslijeđene od klase MonoBehaviour.

Za rad su Unity Game Engine-om korišteni su imenski prostori UnityEngine, UnityEngine.UI, UnityEngine.Events i UnityEngine.EventSystems, a za rad sa datotekama System.IO i System.Xml te za liste i kolekcije System.Collections.Generic i System.Collections.

1. Klasa Scene

Klasa Scene u osnovi upravlja igrom, u sceni je dodijeljena objektu zvanom SceneController.

Klasa scene u sebi sadrži deklaraciju tipa GameState, varijablom tog tipa i svojstvom za tu varijablu koje prilikom postavljanja vrijednosti te varijable pomoću naredbe switch mijenja i trenutnu glazbu u igri.

```
public class Scene : MonoBehaviour {

    public enum GameState   //state-ovi igre
    {
        menu,
        playing,
        editing,
        paused,
        won
    }

    static GameState gs = 0;
           public static GameState currentGameState    //svojstvo koje prilikom promjene
                                                         stanja igre mijenja glazbu u igri
    {
        get { return gs; }
        set
        {
            gs = value;
            switch (gs)
            {
                case GameState.menu:
                    if (PlayerCamera.main.a7.clip != PlayerCamera.main.menuMusic)
                    {
                        PlayerCamera.main.a7.clip = PlayerCamera.main.menuMusic;
                        PlayerCamera.main.a7.Play();
                    }
                    break;
```

```
                case GameState.playing:
                    if (PlayerCamera.main.a7.clip != PlayerCamera.main.gameMusic)
                    {
                        PlayerCamera.main.a7.clip = PlayerCamera.main.gameMusic;
                        PlayerCamera.main.a7.Play();
                    }
                    break;
                case GameState.editing:
                    PlayerCamera.main.a7.Stop();
                    PlayerCamera.main.a7.clip = null;
                    break;
                case GameState.paused:
                    break;
                case GameState.won:
                    break;
                default:
                    break;
            }
        }

}
```

Nakon toga slijede statičke varijable za baratanje s objektima u kodu i osnovni parametri za level-e.

```
    public static Scene main;   //za lakše baratanje

    public static GameObject rootObject;    //root objekt koji sadrži sve objekte level-a
                                              u sceni

    public static Light mainLight;          //glavna svjetlost u sceni

    public static Camera mainCamera;        //glavna kamera u sceni

    public static Player player;            //referenca na igrača

    public static float minHeight = -10f;   //min dozvoljena visina igrača

    public static string nextLevel = "";    //path do datoteke sljedećeg level-a
```

Zatim slijedi metoda Awake koje se poziva inicijalizacijom skripte tj. početkom igre te služi za inicijalizaciju igre i postavljanje stanja kamera, igrača, inicijalizacija root objekta, čitanje postavki igre iz XML-a te njihova primjena.

```
void Awake()    //inicijalizacija
    {
        if (main == null)
            main = this;
        if (main != this)
            main = this;
        if (!mainCamera)
            mainCamera = Camera.main;
```

```
        if(!rootObject)
        {
            GameObject obj = GameObject.FindGameObjectWithTag("Root");
            if (obj)
                rootObject = obj;
            else
            {
                rootObject = new GameObject("Root");
                rootObject.tag = "Root";
            }
        }
        if (!mainLight)
            mainLight = GameObject.FindObjectOfType<Light>();
        player = GameObject.FindObjectOfType<Player>();
        XmlDocument settings = new XmlDocument();
        settings.Load("GameSettings.xml");
        XmlNode options = settings.GetElementsByTagName("Options")[0];
        PlayerCamera.main.a7.volume = float.Parse(options.ChildNodes[0].InnerText);
        GameData.main.musicVolumeSlider.value = PlayerCamera.main.a7.volume;
        if (bool.Parse(options.ChildNodes[1].InnerText))
        {
            //ToggleFPS();
            GameData.main.fpsToggle.isOn = true;
        }
        LoadMenu(); //učitaj meni..
    }
```

Na kraju te metode poziva se metoda LoadMenu, tokom igre često se poziva te je njena uloga baš kao što i njeno ime govori, učitavanje menija, prilikom toga je potrebno za to je potrebno isključiti nepotrebne UI grupe objekata, uključiti sve UI objekte menija, učitati level menija i podesiti određene postavke.

```
public void LoadMenu()  //učitavanje meni-ja
    {
        Cursor.visible = true;
        if (currentGameState == GameState.editing)  //ako je igrač prije bio u level
                                                      editor-u, počisti sve što je od
                                                      toga ostalo
            GameEditor.Clear();
        if (LevelLoader.currentLevel != "Menu.xml") //učitaj Level menija (ako već nije
                                                      učitan)
            LevelLoader.Load("Menu.xml");
        //isključi sve ostale UI grupe objekata
        for (int i = 0; i < GameData.main.youDiedStuff.Length; i++)
            GameData.main.youDiedStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.youWonStuff.Length; i++)
            GameData.main.youWonStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.inGameStuff.Length; i++)
            GameData.main.inGameStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.pauseStuff.Length; i++)
            GameData.main.pauseStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.editorStuff.Length; i++)
            GameData.main.editorStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.levelSelectionStuff.Length; i++)
            GameData.main.levelSelectionStuff[i].gameObject.SetActive(false);
```

```
        for (int i = 0; i < GameData.main.optionsStuff.Length; i++)
            GameData.main.optionsStuff[i].SetActive(false);
        GameData.main.helpUIStuff.SetActive(false);
        Time.timeScale = 1f;    //za uklanjanje pauze (ako je bila)
        //općenite postavke scene i igre...
        PlayerCamera.main.mode = PlayerCamera.PlayerCameraMode.Stand;
        Player.playing = false;
        currentGameState = GameState.menu;
        //uključi UI menija
        for(int i = 0; i < GameData.main.mainButtons.Length; i++)
            GameData.main.mainButtons[i].gameObject.SetActive(true);
        if (Player.main)
            Player.main.anim.Play("Idles", 0); //dino treba biti u idle animaciji
    }
```

Nakon toga, veći dio metoda je samo za procesiranje događaja pozvanih od UI objekata. Metoda SaveOptions sprema opcije klikom na gumb „Back“ iz grupe (polja) optionsStuff.

```
public void SaveOptions()   //spremanje promjenjenih opcija igre
    {
        XmlDocument settings = new XmlDocument();
        settings.Load("GameSettings.xml");
        XmlNode options = settings.GetElementsByTagName("Options")[0];
        options.ChildNodes[0].InnerText = PlayerCamera.main.a7.volume.ToString();
        options.ChildNodes[1].InnerText = GameData.main.fpsToggle.isOn.ToString();
        settings.Save("GameSettings.xml");
    }
```

Metoda toggleFPS se poziva promjenom vrijednosti checkbox-a za FPS (Frames per second – framerate igre) te samo invertira aktivno stanje objekta.

```
    public void ToggleFPS ()    //za uključivanje prikaza FPS-a
    {
        GameData.FPSTextR.SetActive(!GameData.FPSTextR.activeSelf);
    }
```

Metoda OpenOptions, baš kao što i njeno ime nalaže se poziva klikom na „Options“ gumb, ona samo uključuje sve UI objekte opcija i isključuje one nepotrebne.

```
    public void OpenOptions()   //Otvaranje opcija igre
    {
        for (int i = 0; i < GameData.main.optionsStuff.Length; i++) //uključi UI objekte
                                                                      opcija
            GameData.main.optionsStuff[i].SetActive(true);
        //isključi sve nepotrebne UI objekte
        for (int i = 0; i < GameData.main.mainButtons.Length; i++)
            GameData.main.mainButtons[i].gameObject.SetActive(false);
        for (int i = 0; i < GameData.main.levelSelectionStuff.Length; i++)
            GameData.main.levelSelectionStuff[i].gameObject.SetActive(false);

    }
```

Isto tako i metoda LoadHelp, učitava „pomoć“ od igre tj. objekt koji sadrži UI objekte u pomoći od igre.

```
public void LoadHelp()
{
    //isključi nepotrebne UI objekte...
    for (int i = 0; i < GameData.main.mainButtons.Length; i++)
        GameData.main.mainButtons[i].gameObject.SetActive(false);
    //uključi help UI
    GameData.main.helpUIStuff.SetActive(true);
}
```

Metoda restartLevel se poziva prilikom pauze kada igrač želi početi level ispočetka, uglavnom „resetira“ sva potrebna svojstva i pokreće ponovno učitavanje level-a.

```
public void restartLevel()  //ponovno pokretanje level-a
{
    currentGameState = GameState.playing;
    //isključi nepotreban UI
    for (int i = 0; i < GameData.main.youDiedStuff.Length; i++)
        GameData.main.youDiedStuff[i].SetActive(false);
    for (int i = 0; i < GameData.main.youWonStuff.Length; i++)
        GameData.main.youWonStuff[i].SetActive(false);
    LevelLoader.Load(LevelLoader.currentLevel); //reload-aj level
    player.motion = Vector3.zero;   //resertiraj player-ove postavke
    player.currentJump = 0;
          Player.playing = false;
    Player.main.jump = false;
    Player.main.hp_max = GameData.main.playerMaxHp;
    Player.main.anim.SetBool("Dead", false);
    Player.main.anim.SetBool("Idle", true);
    Player.main.anim.Play("Idles", 0);
    unPauseGame();                  //ukloni pauzu
}
```

Metoda LoadEditor učitava game editor, čisti scenu menija postavlja stanje igre u „editing“ te inicira level editoru da se inicijalizira.

```
public void LoadEditor()    //djelomična inicijializacija game editora
{
    LevelLoader.Load("Empty.xml");  //učitaj prazni level
    //postavke scene i igre...
    PlayerCamera.main.mode = PlayerCamera.PlayerCameraMode.Editor;
    Player.playing = false;
    //isključi nepotrebne UI objekte...
    for (int i = 0; i < GameData.main.mainButtons.Length; i++)
        GameData.main.mainButtons[i].gameObject.SetActive(false);
    for (int i = 0; i < GameData.main.inGameStuff.Length; i++)
        GameData.main.inGameStuff[i].SetActive(false);
    for (int i = 0; i < GameData.main.optionsStuff.Length; i++)
        GameData.main.optionsStuff[i].SetActive(false);
    //uključi UI editora
```

```
        for (int i = 0; i < GameData.main.editorStuff.Length; i++)
            GameData.main.editorStuff[i].SetActive(true);
        currentGameState = GameState.editing;
        GameEditor.Setup();     //editor se treba inicijalizirati
        Player.main.anim.SetBool("Idle", false);    //dino ne smije biti u ikakvoj
                                                      animaciji
        Player.main.anim.Play("Nothing", 0);
    }
```

Metoda Quit je isto vezana za događaje UI objekata, samo isključuje igru.

```
    public void Quit()  //izlaz iz igre...
    {
        Application.Quit();
    }
```

I zadnja i najkompleksnija metoda u cijeloj klasi zvana od UI objekata je metoda LoadLevelSelection, ukratko ona učitava korisniku dostupne level-e za igru, isključuje nepotrebne UI objekte te uključuje one za prikaz selekcije level-a, stvara UI objekte iz „predloška“ za pristup level-ima, ti objekti su tipke i svakoj se pridružuje događaj za učitavanje level-a, ime level-a postaje tekst tipke, metoda i za svaki level provjerava da li je zaključan i da li ga je igrač otključao te ako ga igrač nije otključao, igrač ga neće moći igrati. Predložak iz kojega se ti gumbi rade je objekt kreiran u editoru te spremljen, svaki novi gumb je nova i uređena kopija tog predloška, ta kopija je postavljena kao dijete određenog UI objekta koji je „container“ za level-e te ih pravilno poredava po zaslonu neovisno o količini. Način na koji igra otkriva level-e je taj da igra očekuje da u direktoriju u kojem je .exe datoteka igre postoje direktoriji Levels i CustomLevels, u tim direktorijima nalaze se mape u kojima je datoteka sa informacijama o level-u (Level.xml).

```
public void LoadLevelSelection()    //učitavanje level selekcije
    {
          InitUnlockedLevels ();          //update-aj otključane level-e
        if (LevelLoader.currentLevel != "Menu.xml")
            LevelLoader.Load("Menu.xml");
        //update-aj scenu (ako već nije..)
        PlayerCamera.main.mode = PlayerCamera.PlayerCameraMode.Stand;
        currentGameState = GameState.menu;
        //isključi nepotrebne UI objekte
        for (int i = 0; i < GameData.main.mainButtons.Length; i++)
            GameData.main.mainButtons[i].gameObject.SetActive(false);
        for (int i = 0; i < GameData.main.inGameStuff.Length; i++)
            GameData.main.inGameStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.optionsStuff.Length; i++)
            GameData.main.optionsStuff[i].SetActive(false);
        //uključu UI objekte level selekcije
        for (int i = 0; i < GameData.main.levelSelectionStuff.Length; i++)
            GameData.main.levelSelectionStuff[i].gameObject.SetActive(true);
```

```
 //počisti već učitane tipke za level-e (ako postoje)
 for (int i = 0; i < GameData.main.levelsContainer.transform.childCount; i++)
 {
     Destroy(GameData.main.levelsContainer.transform.GetChild(i).gameObject);
 }
      for (int i = 0; i < GameData.main.customLevelsContainer.transform.childCount;
                                                                                 i++)
 {

Destroy(GameData.main.customLevelsContainer.transform.GetChild(i).gameObject);
 }
 List<string> levels = new
       List<string>(System.IO.Directory.GetDirectories("Levels"));
//pretpostavka je da svaka
    mapa u direktoriju Levels sadrži Level.xml datoteku koja sadrži info o levelu
 for (int i = 0; i < levels.Count; i++)
 {
     levels[i] = levels[i].Replace('\\', '/');   //za podršku na Mac OS X-u
                     string arg = System.IO.Directory.GetCurrentDirectory() + "/" +
                               levels[i] + "/Level.xml";    //argument za delegaciju
              if (System.IO.File.Exists (arg)) {
//ako datoteka postoji, kreiraj gumb za učitavanje level-a po template-u za to
                     GameObject obj = Instantiate
                     (GameData.main.levelButtonConstructor);
                     obj.name = levels [i].Remove (0, 7);
                     obj.transform.SetParent
                     (GameData.main.levelsContainer.transform);
                     obj.transform.GetChild (0).GetComponent<Text> ().text =
                     obj.name;
                     XmlDocument doc = new XmlDocument ();
                     doc.Load (arg);
                     XmlNodeList l = doc.GetElementsByTagName ("Locked");    //za
                     provjeru da li je level zaključan
                     bool locked = false;
                     if (l.Count > 0)
                            locked = bool.Parse (l [0].InnerText);
                     if (!locked ||
                            GameData.UnlockedLevels.Contains("/"+levels[i]+"/Level.
                     xml"))   //ako nije zaključan,
                                                          igrač može igrati level
                            obj.GetComponent<Button> ().onClick.AddListener
                                   (delegate {
                                   LevelLoader.Load (arg);
                            }); //dok korisnik stisne gumb, level se počinje
                                  učitavati
                     else {
                            obj.transform.GetChild (1).gameObject.SetActive (true);
                     //u suprotnom, zaključaj level (uključuje lokot)
                     }
              }
 }
 levels.Clear(); //stvaranje cutom levela po istom principu kao i za obične Levele
 levels = new List<string>(System.IO.Directory.GetDirectories("CustomLevels"));
 for (int i = 0; i < levels.Count; i++)
 {
     levels[i] = levels[i].Replace('\\', '/');
              string arg = System.IO.Directory.GetCurrentDirectory() + "/" +
       levels[i] + "/Level.xml";
```

```
                    if (System.IO.File.Exists (arg)) {
                        GameObject obj = Instantiate
                        (GameData.main.levelButtonConstructor);
                        obj.name = levels [i].Remove (0, 13);
                        obj.transform.SetParent
                        (GameData.main.customLevelsContainer.transform);
                        obj.transform.GetChild (0).GetComponent<Text> ().text =
                        obj.name;
                        XmlDocument doc = new XmlDocument ();
                        doc.Load (arg);
                        XmlNodeList l = doc.GetElementsByTagName ("Locked");
                        bool locked = false;
                        if (l.Count > 0)
                            locked = bool.Parse (l [0].InnerText);
                        if (!locked ||
                        GameData.UnlockedLevels.Contains("/"+levels[i]+"/Level.xml"))
                            obj.GetComponent<Button> ().onClick.AddListener
                            (delegate {
                                LevelLoader.Load (arg);
                            });
                        else {
                            obj.transform.GetChild (1).gameObject.SetActive (true);
                        }
                    }
            }
            Player.main.anim.Play("Idles", 0);  //dino je u početku u idle animaciji
        }
```

Unutar te metode nalazi se metoda InitUnlockedLevels koje ažurira listu string-ova koja sadrži putove do level-a koje je igrač završio, ti podaci se čitaju iz GameSettings.xml datoteke.

```
    public void InitUnlockedLevels() {
        XmlDocument doc = new XmlDocument ();
        doc.Load (System.IO.Directory.GetCurrentDirectory () +
                "/GameSettings.xml");
        XmlNodeList l = doc.GetElementsByTagName ("LevelPath");
        for (int i = 0; i < l.Count; i++) {
            if (!GameData.UnlockedLevels.Contains (l [i].InnerText))
                GameData.UnlockedLevels.Add (l [i].InnerText);
        }
    }
```

Nakon toga slijedi metoda clearUIAndStartLevel koje radi ono što i njeno ime nalaže, isključuje UI objekte i postavlja stanje igre u „playing“.

```
public void clearUIAndStartLevel()  //prije početka levela
    {
        currentGameState = GameState.playing;   //postavljanje ispravong game state-a
        Player.playing = true;
        //isključi sve nepotrebne UI objekte
        for (int i = 0; i < GameData.main.mainButtons.Length; i++)
            GameData.main.mainButtons[i].gameObject.SetActive(false);
        for (int i = 0; i < GameData.main.levelSelectionStuff.Length; i++)
```

```
            GameData.main.levelSelectionStuff[i].gameObject.SetActive(false);
        for (int i = 0; i < GameData.main.youWonStuff.Length; i++)
            GameData.main.youWonStuff[i].SetActive(false);
        for (int i = 0; i < GameData.main.optionsStuff.Length; i++)
            GameData.main.optionsStuff[i].SetActive(false);
        //uključi "in game" UI objekte
        for (int i = 0; i < GameData.main.inGameStuff.Length; i++)
            GameData.main.inGameStuff[i].SetActive(true);
    }
```

Metode PauseGame i unPauseGame rade suprotno jedna od druge, pauza zaustavlja vrijeme i mijenja njeno stanje u igri dok ga druga metoda vraća, pauza uključuje UI objekte pauze dok un-pause isključuje.

```
public void PauseGame()         //pauziranje igre
    {
        if (Player.playing)
        {
            currentGameState = GameState.paused;
            Time.timeScale = 0f;                    //zamrzni vrijeme
            for (int i = 0; i < GameData.main.pauseStuff.Length; i++)
                GameData.main.pauseStuff[i].SetActive(true);            //uključi UI od
                                                                          pauze
            Player.playing = false;
        }
    }

    public void unPauseGame()   //uklanjanje pauze
    {
        if (!Player.playing && currentGameState != GameState.won)
        {
            currentGameState = GameState.playing;
            Time.timeScale = 1f;
            for (int i = 0; i < GameData.main.pauseStuff.Length; i++)   //isključi UI
                                                                          pauze
                GameData.main.pauseStuff[i].SetActive(false);
            Player.playing = true;
        }
    }
```

Metoda playerDied se poziva kada igrač umre, uglavnom uključuje UI objekte za tu situaciju te postavlja određena svojstva na potrebne vrijednosti.

```
    public void playerDied()    //kada igrač umre
    {
        PlayerCamera.main.mode = PlayerCamera.PlayerCameraMode.Stand; //resertiraj
                                                                        postavke igrača
        Player.main.hp_max = GameData.main.playerMaxHp;
        for (int i = 0; i < GameData.main.youDiedStuff.Length; i++)
            GameData.main.youDiedStuff[i].SetActive(true);              //uključi
                                                                          potreban UI
        Cursor.visible = true;
        Player.playing = false;
        Player.main.outMotion = Vector3.zero;
```

```
        Player.main.motion = Vector3.zero;
    }
```

Metoda playerWon se poziva kada igrač riješi level, uključuje UI objekte za tu situaciju te podešava određena svojstva na određene vrijednosti, također pregčedava dali je sljedeći level (ako postoji) zaključan, ako je otklučava ga uređujući GameSettings.xml datoteku dodajući novi element s putanjom do to level-a.

```
public void playerWon()     //kada igrač pobijedi
    {
        currentGameState = GameState.won;   //postavi postavke igre...
        Cursor.visible = true;
        Player.main.hp_max = GameData.main.playerMaxHp;
        PlayerCamera.main.mode = PlayerCamera.PlayerCameraMode.Stand;
        for (int i = 0; i < GameData.main.youWonStuff.Length; i++)  //uključi potrebne UI
                                                                     objekte
            GameData.main.youWonStuff[i].SetActive(true);
        Player.playing = false;
             if (!string.IsNullOrEmpty (nextLevel) && File.Exists(nextLevel)) {
//ako postoji sljedeći level nakon ovoga i zaključan je, otključaj ga, također uključi
  next level gumb
                    XmlDocument doc = new XmlDocument ();
                    doc.Load (nextLevel);
                    XmlNode locked = doc.ChildNodes [1].ChildNodes [5];
                    if (bool.Parse (locked.InnerText)) {
                           string l = nextLevel.Replace (Directory.GetCurrentDirectory
                           (), "");
                           if (!GameData.UnlockedLevels.Contains (l)) {
                                  XmlDocument gs = new XmlDocument ();
                                  gs.Load (Directory.GetCurrentDirectory () +
                                  "/GameSettings.xml");
                                  XmlNode lp = gs.CreateElement (string.Empty,
                                  "LevelPath", string.Empty);
                                  lp.AppendChild (gs.CreateTextNode (l));
                                  gs.ChildNodes [1].ChildNodes [0].AppendChild (lp);
                                  gs.Save (Directory.GetCurrentDirectory () +
                                  "/GameSettings.xml");
                                  GameData.UnlockedLevels.Add (l);
                           }
                    }
             }
    }
```

Statička metoda clearChildren briše djecu danog objekta, imena djece se prije toga mijenjaju u „deleted“ jer se samo brisanje obavlja pri kraju frame-a igre a prije toga druge skripte mogu pristupiti tim objektima.

```
    static void clearChildren(GameObject o) //uklanjanje child objekata..
    {
        for (int i = 0; i < o.transform.childCount; i++)
        {
            if (o.transform.GetChild(i).childCount > 0)
                clearChildren(o.transform.GetChild(i).gameObject);
```

```
            o.transform.GetChild(i).name = "deleted";
            o.transform.GetChild(i).gameObject.SetActive(false);
            Destroy(o.transform.GetChild(i).gameObject);
        }
    }
```

Metoda clearScene „čisti“ scenu, briše sve objekte level-a unutar root objekta, te resetira određene postavke i stanje igre.

```
    public static void ClearScene() //čiščenje scene
    {
        if (Player.main)
        {
            Player.main.hp_max = GameData.main.playerMaxHp;  //resertiraj postavke igrača
            Player.main.outMotion = Vector3.zero;
            Player.main.motion = Vector3.zero;
            Player.main.anim.SetBool("Dead", false);
            Player.main.anim.SetBool("Walking", false);
            Player.main.anim.SetBool("Idle", true);
            if (currentGameState == GameState.playing)
                Player.main.anim.Play("Idles");
            else
                Player.main.anim.Play("Nothing");
        }
        clearChildren(rootObject);                      //počisti sve objekte level-a
    }
```

I na kraju cijele klase nalazi se metoda Update, koja se poziva svakog frame-a u igri, ona provjerava ako igrač igra igru, da li je pritisnuo Escape tipku za pauzu, i ako je igra i ako nije uključuje/isključuje vidljivost kursora i uređuje hp bar (health points) koji to i pokazuje.

```
    void Update()   //svakog frame-a u igri
    {
              //print (Player.playing+" "+currentGameState);
        if (Input.GetKeyDown(KeyCode.Escape))        //uključuj/isključuj
                                                       pauzu ako je pritisnut space
            if (currentGameState == GameState.playing)
                PauseGame();
            else if (currentGameState == GameState.paused)
                unPauseGame();
        if(currentGameState == GameState.playing)              //update-aj hp bar i text
                                                                hp-a igrača
        {
            if (Player.main.isAlive)
                Cursor.visible = false;
            GameData.main.hpBarImage.fillAmount = Player.main.hp / Player.main.hp_max;
            GameData.main.hpText.text = Player.main.hp.ToString("0");
        }
        if (currentGameState == GameState.paused)
            Cursor.visible = true;
    }
```

```
}
```

2. Klasa GameData

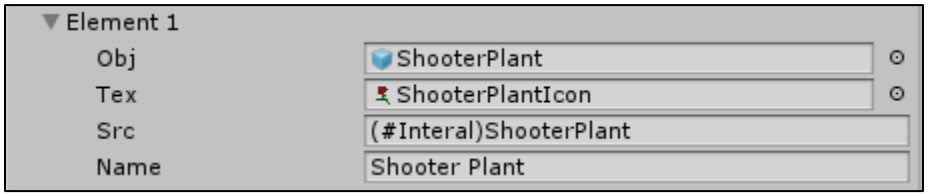
Klasa GameData ne definira ponašanje igre već samo sadrži podatke i određene reference za lakše dohvaćanje u kodu, stvari koje sadrži su materijali za određene svrhe, shader-i koji se koriste u igri, grupe objekata za UI, reference na specifične objekte i sl.

Klasa je naslijeđena od klase MonoBehaviour te je dodana kao komponenta objektu SceneController radi postavljanja referenci u editoru kao komponenti.

Klasa u sebi sadrži deklaraciju strukture ObjNIcon koja sadrži referencu na objekt, njegovu ikonu, referencu u zapisu level-a (src) te njegovo ime.

```
[System.Serializable]   //dodaje mogučnost uređivanja klase u editoru
public struct ObjNIcon  //klasa za konfiguriranje editor object item-a za level
                          editor za unutarnje (#Interal) objekte
{
    public GameObject obj;  //referenca na objet item-a
    public Texture2D tex;   //slika objekta
    public string src;      //izvor objekta
    public string name;     //ime objekta
}
```

Dodavanje te strukture je bilo potrebno radi lakšeg dodavanja „unutarnjih“ objekata igre, oni su u dodani u igru tijekom građenja projekta u binarnim datotekama, a „normalni“ objekti tj. njihove informacije se nalaze u xml datotekama igre i reference na njihove materijale, teksture i 3D modele.



Slika 2 Izgled ObjNIcon strukture u editoru

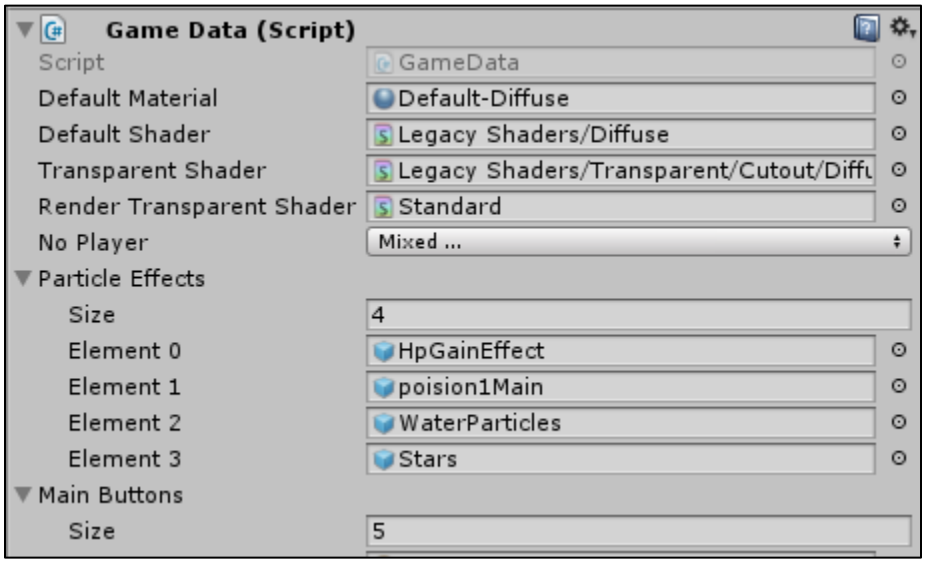
Nakon toga slijede deklaracije javnih varijabli kojima se u inspektoru dodjeljuje ili modificira vrijednost.

```
public static GameData main;        //za lakše dohvačanje nekih varijabli

    public Material DefaultMaterial;    //standardni materijal u igri

    public Shader defaultShader, transparentShader, renderTransparentShader; //korišteni
                                                                               shader-i u igri
```

```
    public LayerMask noPlayer;  //layer mask bez player-a

    public GameObject[] particleEffects;        //grupa particle effect-a

    public Button[] mainButtons;                //tipke meni-a

    public GameObject levelsContainer;          //objekt koji sadrži tipke koje pokreću
                                                  level-e

    public GameObject customLevelsContainer;    //objekt koji sadrži tipke koje pokreću
                                                  custom level-e

    public GameObject[] optionsStuff;           //UI opcije igre

    public GameObject[] levelSelectionStuff;    //polje objekata koje sadrži UI elemente
                                                  level selekcije

    public GameObject[] pauseStuff;             //polje koje sadrži UI elmente kada je
                                                  igra pauzirana

    public GameObject[] youDiedStuff;           //polje koje sadrži UI elemente kada
                                                  igrač umre

    public GameObject[] youWonStuff;            //polje koje sadrži UI elemente kada
                                                  igrač pobijedi

    public GameObject[] editorStuff;            //polje koje sadrži UI elemente level
                                                  editor-a

    public GameObject[] inGameStuff;            //polje koje sadrži UI elemente kada
                                                  igrač igra

    public GameObject helpUIStuff;            //sadrži pomoć (kontrole) igre

    public GameObject levelButtonConstructor;   //template tipka za učitavanje level-a

    public GameObject FPSText;                  //tekst FPS-a igre

    public Mesh[] Meshes;                       //default mesh-evi igre
    /// <summary>
    /// 0 - Cube
    /// 1 - Sphere
    /// 2 - Capsule
    /// 3 - Quad
    /// </summary>

    public Color editorDeselectedColor, editorSelectedColor, editorDisabledColor;
//boje UI elemenata u editoru

    public Material editorSelectedMaterial,     //materijal koji objet ima dok je
                                                  selektiran u editoru
                    TriggerMaterial,            //materijal ta trigger objekte
                    InvincibleColliderMaterial, //materijal za objekte u editoru koji
                                                  su igraču nevidljivi prilikom igranja
                    WinTriggerMaterial;         //materijal koji označuje trigger-e u
                                                  editoru koji donose pobijedu igraču

       public Button nextLevelButton;           //referenca na tipku koja učitava
```

```
                                                    level nakon trenutog (ako postoji)

    public Image hpBarImage;                        //hp bar igrača

    public Text hpText;                             //prikaz hp-a igrača u obliku teksta

    public ObjNIcon[] interalObjects;               //objekti rađeni u unity editoru

       public List<string> unlockedLevels;             //lista path-ova do level-a koje
                                                         je igrač otključao

    public Toggle fpsToggle;                        //checkbox za prikaz FPS-a

    public Slider musicVolumeSlider;                //Slider za jačinu zvuka

    public float playerMaxHp = 4;
```

U editoru dio te komponente izgleda ovako:



Slika 3 izgled klase GameData kao komponente

Nakon toga za neke od tih varijabli dodana su statička svojstva za lakše dohvaćanje tih podataka u daljnjem kodu.

```
public static List<string> UnlockedLevels { get { return main.unlockedLevels; } set {
                    main.unlockedLevels = value; } }

    public static Mesh[] GameMeshes { get { return main.Meshes; } }

    public static Material DefaultGameMaterial { get { return main.DefaultMaterial; } }

    public static Shader DefaultShader { get { return main.defaultShader; } }

    public static Shader TransparentShader { get { return main.transparentShader; } }

    public static Shader RenderTransparentShader { get { return
```

```
                main.renderTransparentShader; } }

    public static GameObject[] ParticleEffects { get { return main.particleEffects; } }

       public static Button NextLevelButton { get { return main.nextLevelButton; } set {
                                                main.nextLevelButton = value; } }

    public static ObjNIcon[] InteralObjects { get { return main.interalObjects; } }

    public static GameObject FPSTextR { get { return main.FPSText; } }
```

I na kraju skripte nalazi se metoda Awake koja se poziva kada se skripta inicijalizira te jedina stvar koja se inicijalizira je statička varijabla main tipa klase GameData.

```
void Awake()    //inicijalizacija..
{
    if(!main)
        main = this;
}
```

3. Klasa LevelLoader

Statička klasa LevelLoader služi za učitavanje level-a, objekata, materijala, tekstura „izvan“ igre. Način na koji je igra osmišljena da radi je takav da za razliku od običnih igara baziranih na Unity-u koje sve te podatke imaju spremljene u binarnim datotekama, igra ima podatke o svojim objektima i level-ima u xml datotekama, slike (teksture) u png formatu, modele u obj formatu i materijale u material datotekama. Time se potreba za ponovnim građenjem igre u slučaju potrebe za promjenom nekog objekta smanjuje jer svaki put kada se igra pokrene ti objekti se ponovo učitaju iz tih datoteka i time čak i napredniji korisnici mogu dodavati svoje objekte u igru i igra je zbog toga na neki način „moddable“ tj. daje korisniku opciju da radi svoje modifikacije i to rijetko koja druga igra na Unity game engine-u ima. No s time ima i nedostataka, jedan od njih je taj da ta „modabilnost“ ne sadrži sve moguće opcije za objekte i scenu koje Unity game engine nudi, već samo one osnovnije, realizacija s mogućnošću uređivanja svega bi bila previše kompleksna, i druga stvar koja je nedostatak je to da korisnik s time može lako urediti određene datoteke te si olakšati igru ili je uništiti u smislu da više ne može funkcionirati ako su određene datoteke obrisane, no opet to se može napraviti svakoj igri ili programu, ali u ovom slučaju je to nešto lakše izvesti.

Klasa započinje deklaracijom varijable currentLevel tipa string, ona sadrži putanju do datoteke trenutnog level-a igre, nakon toga slijedi metoda Load koja učitava level iz

datoteke čija je putanja dan argument, u toj metodi se dan argument sprema u varijablu currentLevel te stvara objekt tipa XmlDocumet kojemu kaže da učita danu datoteku te poziva Garbage Collector da počisti memoriju igre ako je potrebno i inicira sceni da igra može započeti.

```
public static class LevelLoader
{

    public static string currentLevel;

    public static void Load(string file)
    {
        currentLevel = file;
        XmlDocument doc = new XmlDocument();
        doc.Load(file);
        LoadScene(doc);
        System.GC.Collect();
        Scene.main.clearUIAndStartLevel();
    }
```

Nakon toga slijedi metoda za parsanje određenih tipova podataka te zauzvrat vraća taj tip, tip metode se određuje u kodu jer je metoda generička, metoda ime danog tipa prosljeđuje naredbi switch za određen tip iz danog tekstnog argumenta očita vrijednost i sprema ga u taj tip, podržani tipovi su Vector3, RigidbodyConstraints i Color.

```
public static T ParseType<T>(string value)
    {
        T v = default(T);
        switch (typeof(T).Name)
        {
            case "Vector3":
                List<string> parts = new List<string>(value.Split('(', ',', ')', ' ',
                                                                    '\n'));
                parts.RemoveAll(string.IsNullOrEmpty);
                v = (T)System.Convert.ChangeType(new Vector3(float.Parse(parts[0]),
                     float.Parse(parts[1]), float.Parse(parts[2])), typeof(T));
                break;
            case "RigidbodyConstraints":
                //tttfff
                if (value.Length >= 6)
                {
                    RigidbodyConstraints rc = new RigidbodyConstraints();
                    rc |= value[0] == 't' ? RigidbodyConstraints.FreezePositionX : 0;
                    rc |= value[1] == 't' ? RigidbodyConstraints.FreezePositionY : 0;
                    rc |= value[2] == 't' ? RigidbodyConstraints.FreezePositionZ : 0;
                    rc |= value[3] == 't' ? RigidbodyConstraints.FreezeRotationX : 0;
                    rc |= value[4] == 't' ? RigidbodyConstraints.FreezeRotationY : 0;
                    rc |= value[5] == 't' ? RigidbodyConstraints.FreezeRotationZ : 0;
                    v = (T)System.Convert.ChangeType(rc, typeof(T));
                }
                break;
            case "Color":
```

```
                //FFFFFFFF
                if (value.Length >= 8)
                {
                    v = (T)System.Convert.ChangeType(new Color(
                        byte.Parse(value.Substring(0, 2),
                            System.Globalization.NumberStyles.HexNumber) / 255f,
                        byte.Parse(value.Substring(2, 2),
                            System.Globalization.NumberStyles.HexNumber) / 255f,
                        byte.Parse(value.Substring(4, 2),
                            System.Globalization.NumberStyles.HexNumber) / 255f,
                        byte.Parse(value.Substring(6, 2),
                            System.Globalization.NumberStyles.HexNumber) / 255f),
                        typeof(T));
                    //Debug.Log(v);
                }
                break;
            default:

                break;
        }
        return v;
    }
```

Nakon toga slijedi deklaracija triju varijabli, prva služi za učitavanje 3D modela iz obj datoteka u igru, druga je rječnik koji sadrži učitane teksture a treća je isto rječnik koji sadrži učitane materijale.

```
    static ObjImporter objImporter = new ObjImporter();

    static Dictionary<string, Texture2D> loadedTextures = new Dictionary<string,
                                                          Texture2D>();

    static Dictionary<string, Material> loadedMaterials = new Dictionary<string,

                                                          Material>();
```

Zatim slijedi metoda za učitavanje 3D modela (mesh-eva), način na koji radi je taj da iz danog string-a metoda provjeri da li ima naznaku da je u podacima igre ili je u vanjskoj datoteci, ako je u vanjskoj datoteci onda od objImporter varijable zahtjeva da ga učita, u suprotnom vraća model iz klase GameData ili ako je string prazan ova metoda vraća vrijednost null.

```
static Mesh LoadMesh(string value)
    {
        if (string.IsNullOrEmpty(value))
            return null;
        Mesh mesh;
        if (value.Substring(0, 10) == "(#Interal)")
        {
            switch (value.Substring(10, value.Length - 10))
            {
                case "Cube":
```

```
                mesh = GameData.GameMeshes[0];
                break;
            case "Sphere":
                mesh = GameData.GameMeshes[1];
                break;
            case "Capsule":
                mesh = GameData.GameMeshes[2];
                break;
            case "Quad":
                mesh = GameData.GameMeshes[3];
                break;
            default:
                mesh = GameData.GameMeshes[0];  //Set cube as default mesh
                break;
        }
    }
    else
    {
        mesh = objImporter.ImportFile(System.IO.Directory.GetCurrentDirectory() +
          value);
    }
    return mesh;
}
```

Zatim slijedi metoda za učitavanje materijala, koja po istom principu prima argument tipa string koji je put do datoteke tog materijala, datoteka nastavka material ne sadrži ništa drugo nego xml kod o informacijama tog materijala, neke druge postavke koje u xml kodu nisu navedene, a mnoge se ni ne mogu navesti se postavljaju u određene vrijednosti, u ovoj metodi uspješno učitan materijal se sprema rječnik za učitane materijale tako da se isti materijal neće morati učitavati više puta, ova metoda također učitava i teksturu dodanu tom materijalu i stavlja je u rječnik učitanih tekstura i dodaje mu prikladni shader ovisno o tome kako je navedeno u material datoteci.

```
static Material LoadMaterial(string value)
{
    if (string.IsNullOrEmpty(value))
        return null;
    if (loadedMaterials.ContainsKey(value.Trim()))
    {
        return loadedMaterials[value.Trim()];
    }
    else
    {
        if (value.Replace(" ", "") == "(#Interal)Default")
        {
            return GameData.DefaultGameMaterial;
        }
        XmlDocument doc = new XmlDocument();
        doc.Load(System.IO.Directory.GetCurrentDirectory() + value);
        string keym = value.Trim();
        loadedMaterials[keym] = new Material(GameData.DefaultShader);
```

```
            loadedMaterials[keym].SetFloat("_Glossiness", 0f);
            loadedMaterials[keym].SetFloat("_Metallic", 0.5f);
            if (!string.IsNullOrEmpty(doc.ChildNodes[0].ChildNodes[0].InnerText))
            {
                string key = doc.ChildNodes[0].ChildNodes[0].InnerText.Trim();
                if (!loadedTextures.ContainsKey(key))
                {
                    loadedTextures[key] = new Texture2D(2, 2, TextureFormat.RGBA32, true,
                                                        true);

loadedTextures[key].LoadImage(System.IO.File.ReadAllBytes(System.IO.Directory.GetCurrentD
                                irectory() + key), false);
                    loadedTextures[key].wrapMode = TextureWrapMode.Repeat;
                    loadedTextures[key].filterMode =
                     (FilterMode)System.Enum.Parse(typeof(FilterMode),
              doc.ChildNodes[0].ChildNodes[0].Attributes[0].Value);
                    loadedTextures[key].anisoLevel =
                     int.Parse(doc.ChildNodes[0].ChildNodes[0].Attributes[1].Value);
                    if (doc.ChildNodes[0].ChildNodes[0].Attributes["Transparent"] !=
                            null)
                    {
                        //loadedTextures[key].alphaIsTransparency =
bool.Parse(doc.ChildNodes[0].ChildNodes[0].Attributes["Transparent"].InnerText);
                        if
(bool.Parse(doc.ChildNodes[0].ChildNodes[0].Attributes["Transparent"].InnerText))
                        {
                            loadedMaterials[keym].shader = GameData.TransparentShader;
                            loadedMaterials[keym].SetFloat("_Mode", 1f);
                            loadedMaterials[keym].SetFloat("_Metallic", 1f);
                            loadedMaterials[keym].SetFloat("_Glossiness", 0f);
                            loadedMaterials[keym].SetFloat("_Cutoff", 0.65f);
                            loadedMaterials[keym].EnableKeyword("_ALPHATEST_ON");
                        }
                    }
                    loadedMaterials[keym].mainTexture = loadedTextures[key];
                }
                else
                {
                    loadedMaterials[keym].mainTexture = loadedTextures[key];
                }
            }
            loadedMaterials[keym].color =
                            ParseType<Color>(doc.ChildNodes[0].ChildNodes[1].InnerText);
            if (!string.IsNullOrEmpty(doc.ChildNodes[0].ChildNodes[2].InnerText))
            {
                string key = doc.ChildNodes[0].ChildNodes[2].InnerText.Trim();
                if (!loadedTextures.ContainsKey(key))
                {
                    loadedTextures[key] = new Texture2D(2, 2, TextureFormat.RGBA32,
                     false);
                    loadedTextures[key].LoadImage(System.IO.File.ReadAllBytes(
                     System.IO.Directory.GetCurrentDirectory() + key), false);
                    loadedMaterials[keym].SetTexture("_BumbMap", loadedTextures[key]);
                }
                else
                {
                    loadedMaterials[keym].SetTexture("_BumbMap", loadedTextures[key]);
                }
```

```
            }
            return loadedMaterials[keym];
        }
    }
```

Onda slijedi metoda za učitavanje Collider-a objekata, Collider je uglavnom nevidljivi dio objekta koji je fizički model za sudaranje s drugim objektima. Metoda za argumente uzima ime Collider-a, objekt kojemu se taj argument dodaje, te ima opcionalne argumente da li je collider konveksan (konveksni poligon) i da li je collider trigger (ako je collider trigger, on se ne sudara, već samo šalje poruke ako se sudari s nekim drugim collider-om koji nije trigger).

```
    static void LoadCollider(string name, GameObject obj, bool convex = false, bool
trigger = false)
    {
        if (string.IsNullOrEmpty(name))
            return;
        switch (name)
        {
            case "BoxCollider":
                obj.AddComponent<BoxCollider>().isTrigger = trigger;
                break;
            case "SphereCollider":
                obj.AddComponent<SphereCollider>().isTrigger = trigger;
                break;
            case "CapsuleCollider":
                obj.AddComponent<CapsuleCollider>().isTrigger = trigger;
                break;
            case "MeshCollider":
                MeshCollider mc = obj.AddComponent<MeshCollider>();
                mc.convex = convex;
                mc.isTrigger = trigger;
                break;
            default:
                obj.AddComponent<BoxCollider>().isTrigger = trigger;
                break;
        }
    }
```

Zatim slijedi ključna metoda LoadObj koja učitava objekte, kao argumente uzima string koji sadrži putanju do xml datoteke tog objekta i referencu praznog objekta kojeg će metoda nadograditi. Metoda prvo provjerava da li je string ispravan, ako je, nastavlja s izvođenjem. Zatim provjerava da li je dan string oznaka unutrašnjeg objekta u igri konfiguriranog u editoru i postavljenog u GameData klasi, za svaki takav objekt stvara se novi, a stari se briše, pozicija, rotacija i veličina tog objekta u odnosu na stari ostaju isti, za Shooter i Sniper Plant objekte potrebno je podesiti duljinu klipa animacije ispaljivanja tako da se preklapa sa postavkom rateOfFire. Ako objekt nije iz engine-a, onda ga je

potrebno iščitati iz njegove xml datoteke te ga kreirati po tome, tipovi objekata koji u igri postoje su Prepreke (Obstacle) – zidovi, pod i klifovi, Dekoracije - dekorativni objekti (Decoration), Pomične Prepreke - platforme (MovingObstacle), Trigger - nevidljivi objekt s kojim se igrač ne sudara ali prelaskom kroz njega daje određene posljedice, CollectableItem – stavi koje igrač skuplja, AI – neprijatelji igrača, Ammo – objekt koji ispaljuje AI tipa shooter. Za svaki objekt učitava se njegov model (ako ga ima), materijal i tekstura (ako ih ima), collider (ako ga ima) i objekti tipa AI, MovingObstacle, Ammo, Trigger i CollectableItem dobivaju svoje komponente konfigurirane xml-om.

```
public static void LoadObj(string path, GameObject obj)
 {
     if (string.IsNullOrEmpty(path))
         return;
     //Debug.Log(path);
     if(path.Contains("(#Interal)"))
     {
         path = path.Replace("(#Interal)", "");
         switch(path)
         {
             case "Water":
                 Vector3 scalew = obj.transform.localScale;
                 obj = Object.Instantiate(GameData.InteralObjects[0].obj,
                  obj.transform.position, obj.transform.rotation,
                  obj.transform.parent);
                 obj.transform.localScale = scalew;
                 obj.isStatic = true;
                 //Debug.Log("Water");
                 break;
             case "ShooterPlant":
                 Vector3 scalesp = obj.transform.localScale;
                 var obja = Object.Instantiate(GameData.InteralObjects[1].obj,
                  obj.transform.position, obj.transform.rotation,
                  obj.transform.parent);
                 obja.transform.localScale = scalesp;
                 obja.name = obj.name;
                 Object.Destroy(obj);
                 obj = obja;
                 AI aisp = obj.GetComponent<AI>();
                 AnimationClip ac = null;
                 for (int i = 0; i <
                  aisp.anim.runtimeAnimatorController.animationClips.Length; i++)
                     if (aisp.anim.runtimeAnimatorController.animationClips[i].name ==
                         "shoot") {
                         ac = aisp.anim.runtimeAnimatorController.animationClips[i];
                         break;
                     }
                 if (ac != null)
                     aisp.anim.SetFloat("ShootAnimSpeed", ac.length / (1f /
                  aisp.rateOfFire + aisp.relaxTime));
                 break;
             case "AnkPatrol":
```

```
                    Vector3 scalespap = obj.transform.localScale;
                    var objak = Object.Instantiate(GameData.InteralObjects[2].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    objak.transform.localScale = scalespap;
                    objak.name = obj.name;
                    Object.Destroy(obj);
                    obj = objak;
                    break;

                case "AnkAttack":
                    Vector3 scalespapa = obj.transform.localScale;
                    var objaka = Object.Instantiate(GameData.InteralObjects[3].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    objaka.transform.localScale = scalespapa;
                    objaka.name = obj.name;
                    Object.Destroy(obj);
                    obj = objaka;
                    break;
                case "LVL1Base":
                    Vector3 scale11 = obj.transform.localScale;
                    obj = Object.Instantiate(GameData.InteralObjects[4].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    obj.transform.localScale = scale11;
                    obj.isStatic = true;
                    break;
                case "IntroMesh":
                    Vector3 scale12 = obj.transform.localScale;
                    obj = Object.Instantiate(GameData.InteralObjects[5].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    obj.transform.localScale = scale12;
                    obj.isStatic = true;
                    break;
                case "LargeCliff":
                    Vector3 scale13 = obj.transform.localScale;
                    obj = Object.Instantiate(GameData.InteralObjects[6].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    obj.transform.localScale = scale13;
                    obj.isStatic = true;
                    break;
                case "SniperPlant":
                    Vector3 scalessp = obj.transform.localScale;
                    var objas = Object.Instantiate(GameData.InteralObjects[7].obj,
                     obj.transform.position, obj.transform.rotation,
                     obj.transform.parent);
                    objas.transform.localScale = scalessp;
                    objas.name = obj.name;
                    Object.Destroy(obj);
                    obj = objas;
                    AI aisps = obj.GetComponent<AI>();
                    //aisp.anim["idle"].speed = 2.25f;
                    //aisp.anim.speed = aisp.anim["shoot"].length / (1f / aisp.rateOfFire
+ aisp.relaxTime);
                    AnimationClip acs = null;
```

```
            for (int i = 0; i < aisps. anim. runtimeAnimatorController.
                    animationClips.Length; i++)
                if (aisps.anim.runtimeAnimatorController.animationClips[i].name
                    == "shoot")
                {
                    acs = aisps.anim.runtimeAnimatorController.animationClips[i];
                    break;
                }
            if (acs != null)
                aisps.anim.SetFloat("ShootAnimSpeed", acs.length / (1f /
                aisps.rateOfFire + aisps.relaxTime));
            break;
        case "LVL4Base":
            Vector3 scale14 = obj.transform.localScale;
            obj = Object.Instantiate(GameData.InteralObjects[8].obj,
             obj.transform.position, obj.transform.rotation,
             obj.transform.parent);
            obj.transform.localScale = scale14;
            obj.isStatic = true;
            break;
        case "LVL5Base":
            Vector3 scale15 = obj.transform.localScale;
            obj = Object.Instantiate(GameData.InteralObjects[9].obj,
             obj.transform.position, obj.transform.rotation,
             obj.transform.parent);
            obj.transform.localScale = scale15;
            obj.isStatic = true;
            break;
    }
    return;
}
if (!File.Exists(path))
    path = System.IO.Directory.GetCurrentDirectory() + path;
XmlDocument doc = new XmlDocument();
doc.Load(path);
XmlNode objData = doc.ChildNodes[0];
switch (objData.Attributes["Type"].Value)
{
    case "Obstacle":
        if (objData.Attributes["CustomTag"] != null)
            obj.tag = objData.Attributes["CustomTag"].Value;
        else
            obj.tag = "Obstacle";
        obj.isStatic = true;
        obj.AddComponent<MeshFilter>().mesh =
      LoadMesh(objData.ChildNodes[0].ChildNodes[0].InnerText);
        MeshRenderer renderer = obj.AddComponent<MeshRenderer>();
        renderer.sharedMaterial =
      LoadMaterial(objData.ChildNodes[0].ChildNodes[1].InnerText);
        renderer.lightProbeUsage = UnityEngine.Rendering.LightProbeUsage.Off;
        renderer.reflectionProbeUsage =
      UnityEngine.Rendering.ReflectionProbeUsage.Off;
        LoadCollider(objData.ChildNodes[0].ChildNodes[2].InnerText, obj,
        objData.ChildNodes[0].ChildNodes[2].Attributes.Count != 0 ?
        bool.Parse(objData.ChildNodes[0].ChildNodes[2].Attributes[0].Value) :
             false);
        break;
    case "MovingObstacle":
```

```
            if (objData.Attributes["CustomTag"] != null)
                obj.tag = objData.Attributes["CustomTag"].Value;
            else
                obj.tag = "MovingObstacle";
            obj.AddComponent<MeshFilter>().mesh =
          LoadMesh(objData.ChildNodes[1].ChildNodes[0].InnerText);
            MeshRenderer rend = obj.AddComponent<MeshRenderer>();
            rend.sharedMaterial =
          LoadMaterial(objData.ChildNodes[1].ChildNodes[1].InnerText);
            rend.lightProbeUsage = UnityEngine.Rendering.LightProbeUsage.Off;
            rend.reflectionProbeUsage =
          UnityEngine.Rendering.ReflectionProbeUsage.Off;
            LoadCollider(objData.ChildNodes[1].ChildNodes[2].InnerText, obj,
          objData.ChildNodes[1].ChildNodes[2].Attributes.Count != 0 ?
          bool.Parse(objData.ChildNodes[1].ChildNodes[2].Attributes[0].Value) :
false);
            Rigidbody r = obj.AddComponent<Rigidbody>();
            r.mass = float.Parse(objData.ChildNodes[0].ChildNodes[0].InnerText);
            r.useGravity = bool.Parse(objData.ChildNodes[0].ChildNodes[1].InnerText);
            r.interpolation =
          (RigidbodyInterpolation)System.Enum.Parse(typeof(RigidbodyInterpolation),
objData.ChildNodes[0].ChildNodes[2].InnerText);
            r.collisionDetectionMode =
          (CollisionDetectionMode)System.Enum.Parse(typeof(CollisionDetectionMode),
objData.ChildNodes[0].ChildNodes[3].InnerText);
            r.constraints =
          ParseType<RigidbodyConstraints>(objData.ChildNodes[0].ChildNodes[4].InnerTe
xt + objData.ChildNodes[0].ChildNodes[5].InnerText);
            MovingObstacle mo = obj.AddComponent<MovingObstacle>();
            mo.type =
          (MovingObstacle.ObstacleType)System.Enum.Parse(typeof(MovingObstacle.Obstac
leType), objData.ChildNodes[1].Attributes[0].Value);
            XmlNodeList l = objData.ChildNodes[1].ChildNodes[3].ChildNodes;
            mo.points = new MovingObstacle.Point[l.Count];
            for (int j = 0; j < l.Count; j++)
            {
                mo.points[j] = new MovingObstacle.Point
                 (float.Parse(l[j].ChildNodes[0].InnerText) < Time.fixedDeltaTime ?
                 Time.fixedDeltaTime : float.Parse(l[j].ChildNodes[0].InnerText),
                 ParseType<Vector3>(l[j].ChildNodes[1].InnerText));
            }
            break;

        case "Decoration":
            if (objData.Attributes["CustomTag"] != null)
                obj.tag = objData.Attributes["CustomTag"].Value;
            else
                obj.tag = "Decoration";
            obj.isStatic = true;
            obj.AddComponent<MeshFilter>().mesh =
            LoadMesh(objData.ChildNodes[0].ChildNodes[0].InnerText);
            MeshRenderer rend_ = obj.AddComponent<MeshRenderer>();
            rend_.lightProbeUsage = UnityEngine.Rendering.LightProbeUsage.Off;
            rend_.reflectionProbeUsage =
            UnityEngine.Rendering.ReflectionProbeUsage.Off;
            rend_.material =
            LoadMaterial(objData.ChildNodes[0].ChildNodes[1].InnerText);
            break;
```

```
            case "StartPos":
                Scene.player.transform.position = obj.transform.position;
                break;
            case "Trigger":
                if (objData.ChildNodes[0].ChildNodes[1].InnerText.Trim() ==
                    "MeshCollider")
                {
                    obj.AddComponent<MeshFilter>().mesh =
                    LoadMesh(objData.ChildNodes[0].ChildNodes[0].InnerText);
                    LoadCollider(objData.ChildNodes[0].ChildNodes[1].InnerText, obj,
                    true, true);
                }
                else
                {
                    LoadCollider(objData.ChildNodes[0].ChildNodes[1].InnerText, obj,
                    false, true);
                }
                Rigidbody tr = obj.AddComponent<Rigidbody>();
                tr.constraints = RigidbodyConstraints.FreezeAll;
                tr.useGravity = false;
                Trigger trig = obj.AddComponent<Trigger>();
                List<string> ls = new
                    List<string>(objData.ChildNodes[0].ChildNodes[2].InnerText.Split('
                    ', '(', ')', '\n'));
                ls.RemoveAll(string.IsNullOrEmpty);
                if (ls.Count > 0)
                {
                    switch (ls[0])
                    {
                        case "PlayerLose":
                            trig.OnPlayerEnter = Trigger.EventAction.PlayerLose;
                            trig.arg1 = null;
                            break;
                        case "PlayerGainHp":
                            trig.OnPlayerEnter = Trigger.EventAction.PlayerGainHp;
                            trig.arg1 = float.Parse(ls[1]);
                            break;
                        case "PlayerWin":
                            trig.OnPlayerEnter = Trigger.EventAction.PlayerWin;
                            trig.arg1 = null;
                            break;
                        default:
                            ///NOTHING
                            break;
                    }
                }
                ls = new
List<string>(objData.ChildNodes[0].ChildNodes[3].InnerText.Split(' ', '(', ')', '\n'));
                ls.RemoveAll(string.IsNullOrEmpty);
                if (ls.Count > 0)
                {
                    switch (ls[0])
                    {
                        case "PlayerLose":
                            trig.OnPlayerStay = Trigger.EventAction.PlayerLose;
                            trig.arg2 = null;
                            break;
                        case "PlayerGainHp":
```

```
                            trig.OnPlayerStay = Trigger.EventAction.PlayerGainHp;
                            trig.arg2 = float.Parse(ls[1]);
                            break;
                        case "PlayerWin":
                            trig.OnPlayerStay = Trigger.EventAction.PlayerWin;
                            trig.arg2 = null;
                            break;
                        default:
                            ///NOTHING
                            break;
                    }
                }
                ls = new
List<string>(objData.ChildNodes[0].ChildNodes[4].InnerText.Split(' ', '(', ')', '\n'));
                ls.RemoveAll(string.IsNullOrEmpty);
                if (ls.Count > 0)
                {
                    switch (ls[0])
                    {
                        case "PlayerLose":
                            trig.OnPlayerExit = Trigger.EventAction.PlayerLose;
                            trig.arg3 = null;
                            break;
                        case "PlayerGainHp":
                            trig.OnPlayerExit = Trigger.EventAction.PlayerGainHp;
                            trig.arg3 = float.Parse(ls[1]);
                            break;
                        case "PlayerWin":
                            trig.OnPlayerExit = Trigger.EventAction.PlayerWin;
                            trig.arg3 = null;
                            break;
                    }
                }
                break;

            case "CollectableItem":
                obj.layer = 8;
                obj.AddComponent<MeshFilter>().mesh =
        LoadMesh(objData.ChildNodes[0].ChildNodes[0].InnerText);
                MeshRenderer rend__ = obj.AddComponent<MeshRenderer>();
                rend__.material =
        LoadMaterial(objData.ChildNodes[0].ChildNodes[1].InnerText);
                rend__.lightProbeUsage = UnityEngine.Rendering.LightProbeUsage.Off;
                rend__.reflectionProbeUsage =
        UnityEngine.Rendering.ReflectionProbeUsage.Off;
                LoadCollider(objData.ChildNodes[0].ChildNodes[2].InnerText, obj,
        objData.ChildNodes[0].ChildNodes[2].Attributes.Count != 0 ?
        bool.Parse(objData.ChildNodes[0].ChildNodes[2].Attributes[0].Value) : false,
        true);
                Rigidbody rig__ = obj.AddComponent<Rigidbody>();
                rig__.interpolation = RigidbodyInterpolation.Interpolate;
                rig__.useGravity = false;
                CollectableItem ci = obj.AddComponent<CollectableItem>();
                ci.action =
              (CollectableItem.CollectableItemGain)
        System.Enum.Parse(typeof(CollectableItem. CollectableItemGain),
        objData.ChildNodes[0].ChildNodes[3].InnerText);
                ci.dissapearEffect =
```

```
UnityEngine.MonoBehaviour.Instantiate(GameData.ParticleEffects[int.Parse(objData.
ChildNodes[0].ChildNodes[6].InnerText)], Scene.rootObject.transform);
          ci.dissapearEffect.transform.position = ci.transform.position;
          ci.dissapearEffect.SetActive(false);
          switch (ci.action)
          {
              case CollectableItem.CollectableItemGain.MaxHpIncrease:
                  ci.arg1 =
        float.Parse(objData.ChildNodes[0].ChildNodes[4].InnerText);
                  ci.arg2 = null;
                  break;
              case CollectableItem.CollectableItemGain.HealPlayer:
                  ci.arg1 =
        float.Parse(objData.ChildNodes[0].ChildNodes[4].InnerText);
                  ci.arg2 = null;
                  break;
              case CollectableItem.CollectableItemGain.BoostSpeed:
                  ci.arg1 =
        float.Parse(objData.ChildNodes[0].ChildNodes[4].InnerText);
                  ci.arg2 =
        float.Parse(objData.ChildNodes[0].ChildNodes[5].InnerText);
                  break;
              default:
                  break;
          }
          break;
      case "AI":
          obj.layer = 9;
          obj.tag = "AI";
          Rigidbody airig = obj.AddComponent<Rigidbody>();
          airig.mass = float.Parse(objData.ChildNodes[0].ChildNodes[0].InnerText);
          airig.useGravity =
bool.Parse(objData.ChildNodes[0].ChildNodes[1].InnerText);
          airig.interpolation =
(RigidbodyInterpolation)System.Enum.Parse(typeof(RigidbodyInterpolation),
objData.ChildNodes[0].ChildNodes[2].InnerText);
          airig.collisionDetectionMode =
(CollisionDetectionMode)System.Enum.Parse(typeof(CollisionDetectionMode),
objData.ChildNodes[0].ChildNodes[3].InnerText);
          airig.constraints =
ParseType<RigidbodyConstraints>(objData.ChildNodes[0].ChildNodes[4].InnerText +
objData.ChildNodes[0].ChildNodes[5].InnerText);
          obj.AddComponent<MeshFilter>().mesh =
LoadMesh(objData.ChildNodes[1].ChildNodes[0].InnerText);
          MeshRenderer rend___ = obj.AddComponent<MeshRenderer>();
          rend___.material =
LoadMaterial(objData.ChildNodes[1].ChildNodes[1].InnerText);
          rend___.lightProbeUsage = UnityEngine.Rendering.LightProbeUsage.Off;
          rend___.reflectionProbeUsage =
UnityEngine.Rendering.ReflectionProbeUsage.Off;
          LoadCollider(objData.ChildNodes[1].ChildNodes[2].InnerText, obj, false,
              false);
          BoxCollider boxcol = obj.AddComponent<BoxCollider>();
          boxcol.isTrigger = true;
          boxcol.size =
              ParseType<Vector3>(objData.ChildNodes[1].ChildNodes[3].InnerText);
          boxcol.center =
              ParseType<Vector3>(objData.ChildNodes[1].ChildNodes[4].InnerText);
```

```
            AI ai = obj.AddComponent<AI>();
            ai.hp = float.Parse(objData.ChildNodes[1].ChildNodes[5].InnerText);
            ai.hp_max = float.Parse(objData.ChildNodes[1].ChildNodes[6].InnerText);
            ai.type = (AI.AIType)System.Enum.Parse(typeof(AI.AIType),
                 objData.ChildNodes[1].ChildNodes[7].InnerText);
            ai.speed = float.Parse(objData.ChildNodes[1].ChildNodes[8].InnerText);
            ai.rotateStop =
                 bool.Parse(objData.ChildNodes[1].ChildNodes[9].InnerText);
            ai.rotSpeed =
                 float.Parse(objData.ChildNodes[1].ChildNodes[10].InnerText);
            ai.patrolPoints = new
                 Vector3[objData.ChildNodes[1].ChildNodes[11].ChildNodes.Count];
            for (int i = 0; i < ai.patrolPoints.Length; i++)
                ai.patrolPoints[i] =
 ParseType<Vector3>(objData.ChildNodes[1].ChildNodes[11].ChildNodes[i].InnerText);
            ai.toleratedDistance =
float.Parse(objData.ChildNodes[1].ChildNodes[12].InnerText);
            ai.activeDistance =
float.Parse(objData.ChildNodes[1].ChildNodes[13].InnerText);
            ai.onGroundRaycastLenght =
float.Parse(objData.ChildNodes[1].ChildNodes[14].InnerText);
            ai.checksBeforeItGoes =
bool.Parse(objData.ChildNodes[1].ChildNodes[15].InnerText);
            ai.forwardRacastCheckLength =
float.Parse(objData.ChildNodes[1].ChildNodes[16].InnerText);
            ai.downRaycastCheckLength =
float.Parse(objData.ChildNodes[1].ChildNodes[17].InnerText);
            GameObject ammo = new GameObject();
            ammo.transform.parent = Scene.rootObject.transform;
            LoadObj(System.IO.Directory.GetCurrentDirectory() +
objData.ChildNodes[1].ChildNodes[18].InnerText, ammo);
            ai.ammo = ammo;
            ai.ammoLifeTime =
float.Parse(objData.ChildNodes[1].ChildNodes[19].InnerText);
            ai.rateOfFire =
float.Parse(objData.ChildNodes[1].ChildNodes[20].InnerText);
            ai.relaxTime =
float.Parse(objData.ChildNodes[1].ChildNodes[21].InnerText);
            ai.shootPointOffset =
ParseType<Vector3>(objData.ChildNodes[1].ChildNodes[22].InnerText);
            ai.shootHeight =
float.Parse(objData.ChildNodes[1].ChildNodes[23].InnerText);
            ai.damageOnCollision =
float.Parse(objData.ChildNodes[1].ChildNodes[24].InnerText);
            break;
        case "Ammo":
            obj.layer = 11;
            Rigidbody ammorig = obj.AddComponent<Rigidbody>();
            ammorig.mass =
float.Parse(objData.ChildNodes[0].ChildNodes[0].InnerText);
            ammorig.useGravity =
bool.Parse(objData.ChildNodes[0].ChildNodes[1].InnerText);
            ammorig.interpolation =
(RigidbodyInterpolation)System.Enum.Parse(typeof(RigidbodyInterpolation),
           objData.ChildNodes[0].ChildNodes[2].InnerText);
            ammorig.collisionDetectionMode =
(CollisionDetectionMode)System.Enum.Parse(typeof(CollisionDetectionMode),
          objData.ChildNodes[0].ChildNodes[3].InnerText);
```

```
                ammorig.constraints = ParseType<RigidbodyConstraints>
                (objData.ChildNodes[0].ChildNodes[4].InnerText +
                    objData.ChildNodes[0].ChildNodes[5].InnerText);
                obj.AddComponent<Ammo>();
                GameObject effect = (MonoBehaviour.Instantiate (GameData.ParticleEffects
            [int.Parse(objData.ChildNodes[1].ChildNodes[0].InnerText)]));
                effect.transform.position = obj.transform.position;
                effect.transform.parent = obj.transform;
                LoadCollider(objData.ChildNodes[1].ChildNodes[1].InnerText, obj, false,
                    true);
                obj.transform.localScale =
            ParseType<Vector3>(objData.ChildNodes[1].ChildNodes[2].InnerText);
                obj.SetActive(false);
                break;
        }
    }
```

Slijedeća i zadnja ključna metoda je LoadScene, koja kao argument uzima xml dokument koji sadrži informacije o level-u. Metoda učitava objekte, stavlja ih na dane pozicije, rotacije i veličine, postavlja parent-e, postavlja kameru i igrača na mjesto, učitava postavke kamere i scene, pregledava postoji li level nakon danog te na temelju toga ažurira gumb za prijelaz na sljedeći level, u suprotnom ga isključuje.

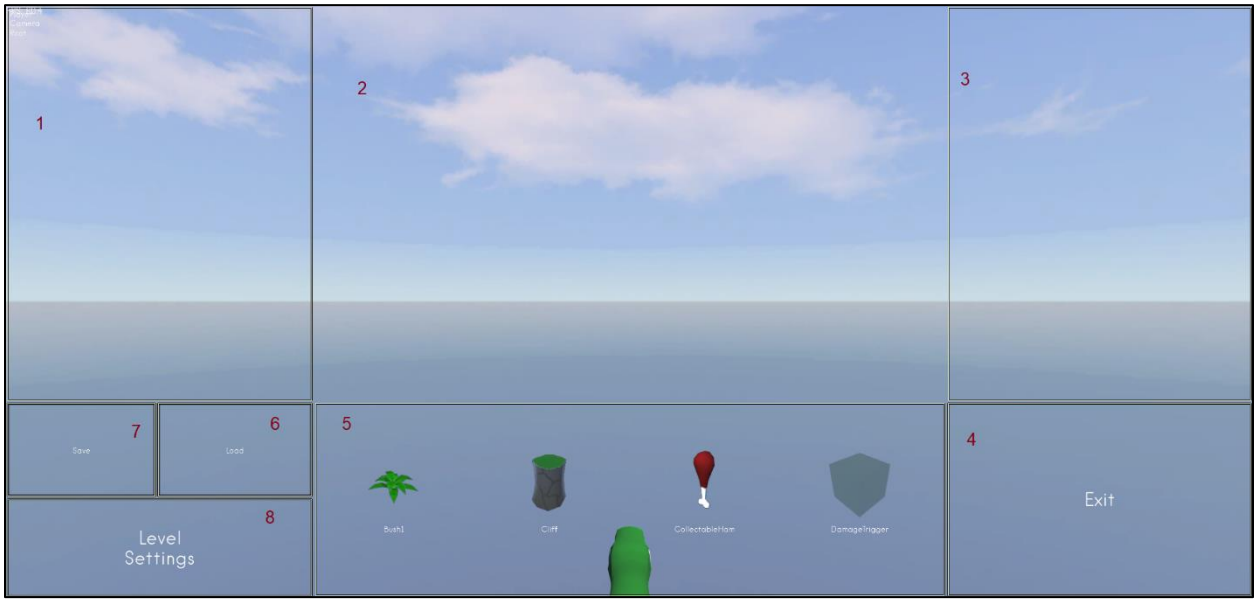
```
static void LoadScene(XmlDocument document)
    {
        Scene.ClearScene();
        XmlNodeList objs = document.GetElementsByTagName("Object");
        for (int i = 0; i < objs.Count; i++)
        {
            if (objs[i].Attributes.Count != 0)
            {
                GameObject obj = new GameObject(objs[i].Attributes["Name"].Value);
                //Debug.Log("Adding: " + obj.name);
                obj.transform.position =
                    ParseType<Vector3>(objs[i].ChildNodes[0].ChildNodes[0].InnerText);
                obj.transform.eulerAngles =
                    ParseType<Vector3>(objs[i].ChildNodes[0].ChildNodes[1].InnerText);
                obj.transform.localScale =
                    ParseType<Vector3>(objs[i].ChildNodes[0].ChildNodes[2].InnerText);
                obj.transform.parent = Scene.rootObject.transform;
                if (!string.IsNullOrEmpty(objs[i].ChildNodes[0].ChildNodes[3].InnerText))
                {
                    GameObject p =
                     GameObject.Find(objs[i].ChildNodes[0].ChildNodes[3].InnerText);
                    if (p != null)
                    {
                        obj.transform.parent = p.transform;
                        obj.isStatic = false;
                        //Debug.Log("Parent found: " + p.name);
                    }
                }
                if ( objs[i].ChildNodes[1].InnerText != null ||
                    objs[i].ChildNodes[1].InnerText != "")
                {
```

```
                    LoadObj(objs[i].ChildNodes[1].InnerText, obj);
                }
            }
        }
        XmlNode scene = document.ChildNodes[1].ChildNodes[1];
        Scene.minHeight = float.Parse(scene.ChildNodes[0].InnerText);
        XmlNode player = document.ChildNodes[1].ChildNodes[2];
        Scene.player.transform.position =
      ParseType<Vector3>(player.ChildNodes[0].ChildNodes[0].InnerText);
        Scene.player.transform.eulerAngles =
      ParseType<Vector3>(player.ChildNodes[0].ChildNodes[1].InnerText);
        Scene.player.transform.localScale =
      ParseType<Vector3>(player.ChildNodes[0].ChildNodes[2].InnerText);
        Scene.player.hp = Scene.player.hp_max;
        Scene.player.isAlive = true;
        Player.playing = true;
        XmlNode camOptions = document.ChildNodes[1].ChildNodes[3];
        PlayerCamera.main.transform.position =
              ParseType<Vector3>(camOptions.ChildNodes[0].InnerText);
        PlayerCamera.main.transform.eulerAngles =
              ParseType<Vector3>(camOptions.ChildNodes[1].InnerText);
        PlayerCamera.main.mx = PlayerCamera.main.my = 0f;
        PlayerCamera.main.mode =
      (PlayerCamera.PlayerCameraMode)System.Enum.Parse(typeof
      (PlayerCamera.PlayerCameraMode), camOptions.ChildNodes[2].InnerText);
        if (PlayerCamera.main.mode == PlayerCamera.PlayerCameraMode.PointLookAt)
        {
            PlayerCamera.main.points = new
              Vector3[camOptions.ChildNodes[3].ChildNodes.Count];
            for (int i = 0; i < PlayerCamera.main.points.Length; i++)
            {
                PlayerCamera.main.points[i] =
              ParseType<Vector3>(camOptions.ChildNodes[3].ChildNodes[i].InnerText);
            }
            PlayerCamera.main.pointOffset =
              ParseType<Vector3>(camOptions.ChildNodes[4].ChildNodes[0].InnerText);
        }
        XmlNode nextLevel = document.ChildNodes[1].ChildNodes[4];
        Scene.nextLevel = Directory.GetCurrentDirectory() + nextLevel.InnerText;
        if (!File.Exists(Scene.nextLevel))
        {
            Scene.nextLevel = "";
            GameData.main.nextLevelButton.transform.GetChild(0)
      .GetComponent<UnityEngine.UI.Text>().color = GameData.main.editorDisabledColor;
            GameData.main.nextLevelButton.onClick.RemoveAllListeners();
        }
        else
        {
            string n = Scene.nextLevel;
            GameData.main.nextLevelButton.onClick.AddListener((delegate { Load(n); }));
            GameData.main.nextLevelButton.transform.GetChild(0)
              .GetComponent<UnityEngine.UI.Text>().
              color = GameData.main.editorDeselectedColor;
        }

    }

}
```

4. Klasa GameEditor

Ovo je najveća klasa u igri po količini koda, u osnovi upravlja cijelim game editor-om koji iz prvog pogleda ne izgleda toliko kompleksno, no procesi koji se u njemu odvijaju znaju biti takvi, sadrži metode koje se često izvode i metode koje se izvode događajima UI objekata od editora i mnogo varijabli od kojih se većina može urediti u Unity editoru, kao komponenta nalazi se u SceneController objektu.

Sučelje editora izgleda ovako:



Slika 4 Izgled game editor-a

Podijeljeno je na 8 dijelova:

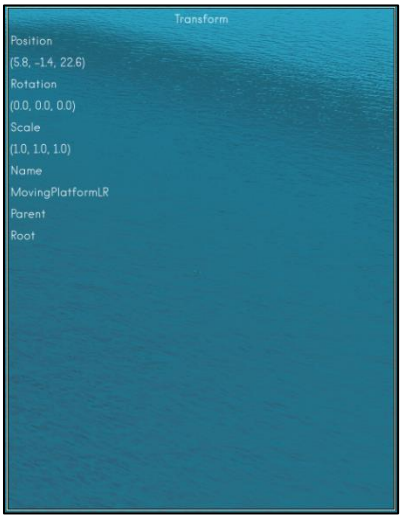
1. Hijerarhija objekata – prikaz svih objekata u level-u, u svakom level-u su tri bitna objekta, igrač (dinosaur), kamera i root objekt koji sadrži sve druge objekte u level-u ako svoju djecu, pomoću hijerarhije moguće je selektirati objekte klikom na njihov naziv koji tada promjeni boju zadanu iz GameData klase, i deselektirati objekte drugim klikom na njihova imena, moguće je postaviti i roditelje objekta pomičući buduće djecu na njihov budući parent, dodani objekt ne može biti izvan root objekta u hijerarhiji i ne može biti dijete kamere ili igrača.
2. Pregled Level-a – kako izgleda level, u njemu je moguće selektirati objekte klikom na njih, ako se drži tipka shift onda se selektira više objekata od

jednom, klikom u prazninu deselektiraju se svi objekti, objekte je moguće pomicati, rotirati i skalirati određenim alatima, alati za te promjene imaju mijenjaju se tipkama W (pomak pozicije), E (promjena rotacije) i R (promjena veličine). Ako je kursor u tome dijelu editora, kameru je moguće

Slika 5 Alati za pomicanje objekata

slobodno pomicati koristeći WASD tipke ili strelice, okretati desnom tipkom miša te pomakom miša, zumirati rotacijom kotačića miša i hover-ati pritiskom kotačića miša te njegovim pomakom.

3. Postavke objekta – prikazuje poziciju, rotaciju, veličinu, ime i ime roditelja objekta te nudi njihovu promjenu, bitno je naglasiti kao će se te postavke pokazivati samo ako je selektiran jedan objekt, uređivanje više objekata u ovome nije moguće, isto tako igraču je moguće samo mijenjati poziciju, kameri poziciju i rotaciju te root objekt se uopće ne može mijenjati.



Slika 6 Izgled postavki objekta

4. Gumb za izlaz iz editora

5. Lista objekata za dodavanje u scenu – sadrži sve objekte koji mogu biti u level-u, listom se može obilaziti rotacijom kotačića miša, objekt se dodaje u level drag n drop metodom, svaki vanjski objekt ima svoju ikonu koju editor sam stvori prilikom njegovog učitavanja, način na koji to funkcionira je taj da svaki objekt u Unity game engine-u ima svoj sloj (layer), i u sceni postoje skriveno svijetlo i skrivena kamera koja prikazuje objekte samo u tom sloju, tako da dok editor prilikom svoga učitavanja i dohvaćanja svih objekata shvati da ikona za jedan objekt fali, on stvori jedan primjerak tog objekta (zapravo stvara primjerak za svaki objekt koji bivaju isključeni), prebaci ga u sloj za renderiranje, ime tog sloja je RenderOnly, uključi ga te kaže toj specijalnoj kameri za renderiranje koji pritom i uključuje da renderira taj objekt, slika na toj kameri korisniku nije vidljiva, ali editor to sliku dobiva i sprema ju u png formatu u direktoriju datoteke objekta (Objects direktorij od igre), te modificira xml datoteku objekta u kojoj dodaje put do slike te datoteke, renderirana slika je u 256x256 rezoluciji, I tako prilikom učitavanja objekata editor za svaki uzme ili napravi ikonu, te kreira UI objekt po određenom predlošku objekta za koji ima danu referencu u editoru. Početkom drag n drop-a stava se kopija povučenog objekta i ako je povučena u pregled level-a, objekt se dodaje u taj level, u suprotnom on biva isključen u sceni, ponovnim drag n dropom tog objekta on se uključuje i ponovo može biti stavljenu level, razlog isključivanja objekta umjesto brisanja je izbjegavanje alokacija i oslobođenja memorije što zapravo i ne stvara tolike probleme na modrenom hardware-u, no ipak je optimalnije rješenje.

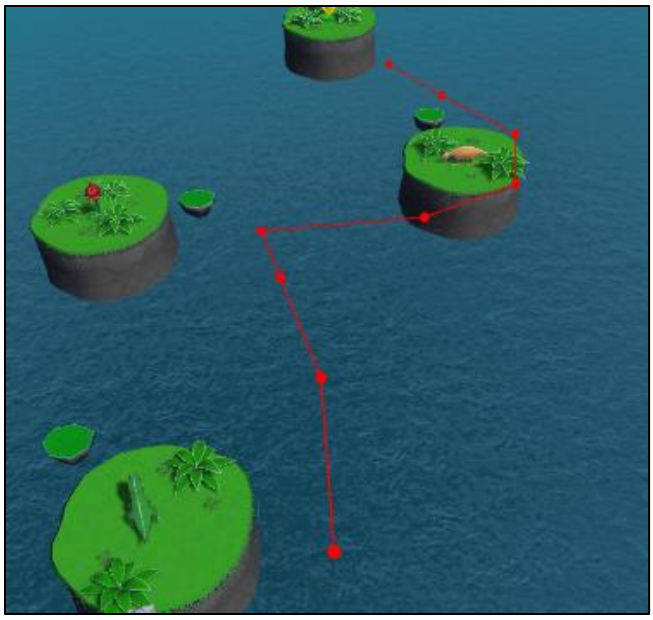
Slika 7 Izgled odabira objekata

6. Load gumb za učitavanje već izrađenih level-a, klikom na njega u pregledu level-a pojavljuju se gumbi za učitavanje pojedinog level-a, ponovnim klikom na ovaj gumb ako ni jedan od level-a nije odabran samo zatvara te gumbe, valja napomenuti da se to level-i mogu obrisati klikom na koš za smeće.

Slika 8 Izgled odabira level-a za učitavanje

7. Save gumb koji sprema trenutni level u editoru
8. Level Settings gumb – otvara postavke level-a, UI se pokazuje u mjestu pogledu level-a, ponovnim klikom na taj gumb taj UI se zatvara, podaci o level-u koji se ti mogu odrediti su:
    a. Ime level-a – ujedno i ime direktorija u kojemu će se nalaziti xml datoteka level-a.
    b. Minimalna dozvoljena koordinata na Y osi od igrača i AI objekata, ako su ispod te razine, automatski umiru.
    c. Tip kamere – kako će se kamera u level-u ponašati, može biti ThirdPerson, PointLookAt i Stand, realizirano je kao drop down lista, ako je odabrana opcija PointLookAt onda se u level-u dodaje specijalni objekt crvena kuglica koja određuje točke po kojima će

kamera ići, te kuglice su spojene crvenim linijama i dodaju se duplicate naredbom (Ctrl + D)



Slika 9 Izgled točaka za kameru u editor-u

d. Odabir slijedećeg Level-a – level koji će biti ponuđen kao slijedeći nakon završavanja trenutnog, izbor level-a je identičan kao i u opciji za učitavanje level-a, samo što se level-i ne mogu obrisati i postoji izbor da tog level-a nema.

Slika 10 Izgled odabira slijedećeg level-a

e. Offset za točke kamere, točka, odnosno vektor koji se pribraja točkama kamere prilikom prijelaza između točaka, u osnovi vektori i

točke po metodi za njihovo parsanje trebaju izgledati u formatu (X, Y, Z) gdje su X, Y i Z realni brojevi pisani sa točkom ako imaju decimalni dio, no metoda za parsanje je tolerantna pa prihvaća i zapis u obliku X Y Z samo sa razmacima.

Slika 11 Izgled postavki level-a

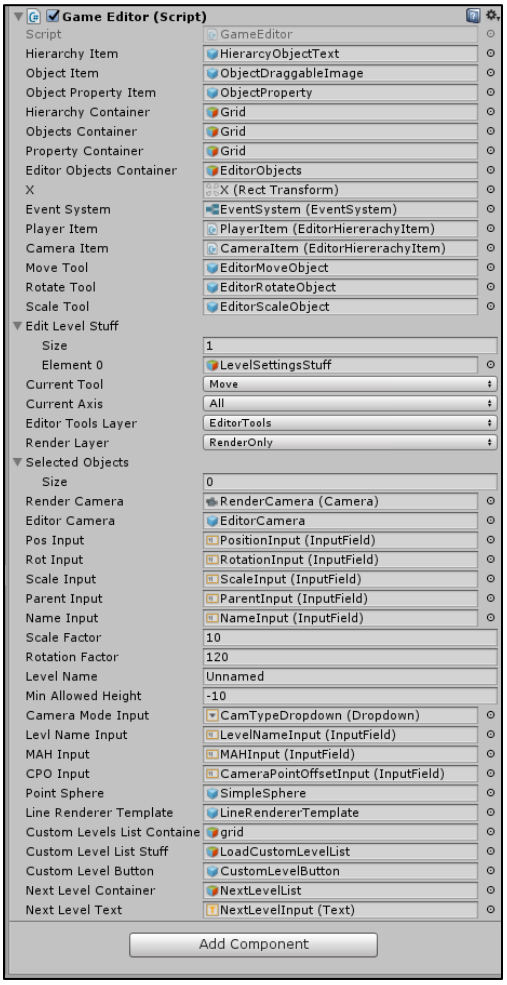
Klasa počinje deklaracijom svakakvih varijabli, svaka ima svoju svrhu u igri, u početku je statička varijabla main koja sadrži statičku referencu na editor, za lakše dohvaćanje u kodu, nakon toga slijede varijable stanja rada editora (da li uređuje postavke ili učitava level-e ili samo radi level), nakon toga slijede template objekti za UI objekte za hijerarhiju, listu objekata i svojstva objekata, njihovi container-i (sadrže ih kao djecu i prikazuju ih), EditorObjectsContainer sadrži primjerke svih učitanih objekata, i točke kamere (crvene kuglice i njihov LineRenderer), objekt X je samo tekst koji se postavlja na poziciju kursora ako igrač želi ubaciti objekt izvan pogleda na scenu, zatim slijedi referenca na EventSystem igre, ta varijabla se ne koristi, zatim slijede reference na UI objekte hijerarhije od igrača i kamere, nakon toga idu reference za alate za mijenjanje objekata u scene pogledu, nakon toga slijedi grupa (polje) UI objekata od postavki level-a, varijabla za pamćenje početne udaljenosti kamere od alata za uređivanje objekata, nakon toga slijede deklaracije 2 enumeratora, jedan za trenutni alat za uređivanje objekta, i jedan za trenutnu os uređivanja

te njihove varijable s početnim vrijednostima, nakon toga slijede maske slojeva za alate i sloj za renderiranje ikoni objekata, onda slijedi lista selektiranih objekata koja je u početku prazna jer ništa nije selektirano, referenca na kameru za renderiranje ikona, referenca na objekt kamere u igri u editoru koji određuje početnu rotaciju i poziciju kamere u level-u i u pogledu na scenu u editoru izgleda kao kamera. Zatim slijede reference za UI polja za upis svojstava objekata, varijabla koja govori da li korisnik vuče objekt u scenu, referenca na objekt koji se vuče, liste za razmak pozicija prilikom pomicanja, rotacije prilikom rotiranja i veličine objekata prilikom skaliranja, nakon toga slijede faktori koji određuju jačinu rotacije/skaliranja objekata pomoću alata, ime level-a, minimalna visina prije smrti, reference na UI objekte postavki level-a, reference za točke kamere (crvene kuglice) i prikazivač linija između njih (LineRenderer), Set sa imenima svih objekata koji služi tome da imena novih objekta ne budu ista kao i kod starih, rječnik sa učitanim objektima te kao ključeve koristi put do datoteke objekta, container-i za skupine tipki level-a i referenca za tekst imena slijedećeg level-a.

```
public class GameEditor : MonoBehaviour
{

    public static GameEditor main;

    public static bool editingSettings { get { return loadLevels || editSettings; } }

    public static bool loadLevels = false, editSettings = false;

    public GameObject hierarchyItem, objectItem, objectPropertyItem;

    public GameObject hierarchyContainer, objectsContainer, PropertyContainer;

    public GameObject EditorObjectsContainer;

    public RectTransform X;

    public UnityEngine.EventSystems.EventSystem eventSystem;

    public EditorHiererachyItem playerItem, cameraItem;

    public GameObject moveTool, rotateTool, scaleTool;

    public GameObject[] editLevelStuff;

    float startCameraDistanceFromTools;

    public enum EditTool { move, rotate, scale }
    public enum EditAxis { all, x, y, z }

    public EditTool currentTool = EditTool.move;
```

```
public EditAxis currentAxis = EditAxis.all;

public LayerMask editorToolsLayer;

public LayerMask renderLayer;

public List<GameObject> selectedObjects = new List<GameObject>();

public Camera renderCamera;

public GameObject editorCamera;

public UnityEngine.UI.InputField posInput, rotInput, scaleInput, parentInput,
          nameInput;

bool dragging = false;
GameObject dragObj;
List<Vector3> dragOffsetPos = new List<Vector3>();
List<Vector3> scales = new List<Vector3>(), rots = new List<Vector3>();
Vector3 dragScreenPoint;
Vector2 startDragPoint;

public float scaleFactor, rotationFactor;

public string levelName = "Unnamed";

public float minAllowedHeight = -10f;

public UnityEngine.UI.Dropdown cameraModeInput;

public UnityEngine.UI.InputField levlNameInput, MAHInput, CPOInput;

public GameObject pointSphere;

public GameObject lineRendererTemplate;

public HashSet<string> names = new HashSet<string>();

public Dictionary<string, EditorObjectItem> LoadedEditorObjects = new
         Dictionary<string, EditorObjectItem>();

public GameObject CustomLevelsListContainer, CustomLevelListStuff, CustomLevelButton,
         NextLevelContainer;

public UnityEngine.UI.Text nextLevelText
```

U Unity editoru klasa GameEditor kao koponenta izgleda ovako:



Slika 12 Izgled klase Game Editor kao komponente

Nakon toga slijede metode za provjeru dali je dani objekt dijete ili unuk root objekta i metoda start koja inicijalizira statičku main varijablu.

```
public static bool isChildOfRoot(GameObject obj)
{
    Transform t = obj.transform;
    if (t == Scene.rootObject.transform)
        return true;
    while (t.parent != null)
    {
        t = t.parent;
        if (t == Scene.rootObject.transform)
            return true;
    }

    return false;
}
```

```
void Start()
{
    main = this;
}
```

Zatim slijedi metoda Setup koja svaki put inicijalizira editor za rad, kao što je i prije spomenuto potrebno je učitati sve dostupne objekte, level editor i uredi da se s njima može baratati i isključuje u smislu da ne rade ono što bi trebali nego samo stoje. Za objekte koji u svom xml-u nemaju teksturu on stvara novu renderirajući ih u rezoluciji 256x256 u png formatu te put do te slike upisuje u xml datoteku objekta, srodno s time on učitava i unutarnje objekte na temelju klase ObjsNIcons koji su konfigurirani u editoru.

```
public static void Setup()
{
    main.startCameraDistanceFromTools = 10;
    PlayerCamera.main.UpdateEditorRotation();
    var lg = GameEditor.main.objectsContainer.
          GetComponent<UnityEngine.UI.GridLayoutGroup>();
    float s = ((Screen.width * 0.5f) - lg.padding.left - lg.spacing.x * 4f) / 4f;
    lg.cellSize = new Vector2(s, s);
    main.names.Add("Player");
    main.names.Add("Root");
    main.names.Add("Camera");
    main.editorCamera.SetActive(true);
    main.UpdateHierarchy();
    if (editingSettings)
    {
        for (int i = 0; i < main.editLevelStuff.Length; i++)
        {
            main.editLevelStuff[i].SetActive(true);
        }
    }
    else
    {
        for (int i = 0; i < main.editLevelStuff.Length; i++)
        {
            main.editLevelStuff[i].SetActive(false);
        }
    }
    //Load objects from Objects folder
    string dir = Directory.GetCurrentDirectory() + "/Objects/";
    string[] files = Directory.GetFiles(dir);
    for (int i = 0; i < files.Length; i++)
    {
        string fileName = Path.GetFileName(files[i]);
        if (Path.GetExtension(fileName) == ".xml")
        {
            XmlDocument doc = new XmlDocument();
            doc.Load(files[i]);
            string imgpath = doc.ChildNodes[0].ChildNodes
                 [doc.ChildNodes[0].ChildNodes.Count - 1].InnerText;
            GameObject objItem = Instantiate(GameEditor.main.objectItem);
            objItem.transform.SetParent(GameEditor.main.objectsContainer.transform);
```

```
EditorObjectItem eoi = objItem.GetComponent<EditorObjectItem>();
//print(files[i]);
main.LoadedEditorObjects.Add(files[i], eoi);
eoi.obj = new GameObject();
eoi.obj.name = fileName.Remove(fileName.Length - 4, 4);
LevelLoader.LoadObj(files[i], eoi.obj);
EditorObject eo = eoi.obj.AddComponent<EditorObject>();
eo.src = files[i].Replace(Directory.GetCurrentDirectory(), "");
eoi.obj.SetActive(false);
var co = eoi.obj.GetComponent<Collider>();
if (co)
{
    if (co.GetType() == typeof(MeshCollider))
        ((MeshCollider)co).convex = true;
    co.isTrigger = true;
}
MeshFilter mf = eoi.obj.GetComponent<MeshFilter>();
if (mf && mf.mesh)
{
    if (mf.mesh.vertices.Length == 0)
    {
        if (co)
        {
            if (co.GetType() == typeof(BoxCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[0];
            }
            else
        if (co.GetType() == typeof(SphereCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[1];
            }
            else
        if (co.GetType() == typeof(CapsuleCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[2];
            }
            else if (co.GetType() == typeof(MeshCollider))
            {
                mf.sharedMesh =
               eoi.obj.GetComponent<MeshCollider>().sharedMesh;
            }
        }
        else
        {
            mf.mesh = null;
        }
    }
}
if (!mf)
{
    if (co)
    {
        if (!mf)
            mf = eoi.obj.AddComponent<MeshFilter>();
        if (co.GetType() == typeof(BoxCollider))
        {
            mf.sharedMesh = GameData.GameMeshes[0];
```

```
        }
        else
        if (co.GetType() == typeof(SphereCollider))
        {
            mf.sharedMesh = GameData.GameMeshes[1];
        }
        else
        if (co.GetType() == typeof(CapsuleCollider))
        {
            mf.sharedMesh = GameData.GameMeshes[2];
        }
        else if (co.GetType() == typeof(MeshCollider))
        {
            mf.sharedMesh = eoi.obj.GetComponent<MeshCollider>()

            .sharedMesh;
        }
    }
}
else
{
    if (!co)
    {
        MeshCollider mc = eoi.obj.AddComponent<MeshCollider>();
        co = mc;
        mc.convex = true;
        mc.isTrigger = true;
        mc.sharedMesh = mf.sharedMesh;
    }
}
MeshRenderer mr = eoi.obj.GetComponent<MeshRenderer>();
if (!mr)
{
    mr = eoi.obj.AddComponent<MeshRenderer>();
    Trigger t = eoi.obj.GetComponent<Trigger>();
    if (t && (t.OnPlayerEnter == Trigger.EventAction.PlayerWin ||
            t.OnPlayerStay == Trigger.EventAction.PlayerWin ||
     t.OnPlayerExit == Trigger.EventAction.PlayerWin))
    {
        mr.sharedMaterial = GameData.main.WinTriggerMaterial;
    }
    else
    {
        mr.sharedMaterial = GameData.main.TriggerMaterial;
    }
}
if (mr && !mr.sharedMaterial)
{
    if (eoi.obj)
    {
        mr.sharedMaterial = GameData.main.InvincibleColliderMaterial;
    }
    else
    {
        Trigger t = eoi.obj.GetComponent<Trigger>();
        if (t && (t.OnPlayerEnter == Trigger.EventAction.PlayerWin ||
                  t.OnPlayerStay == Trigger.EventAction.PlayerWin ||
            t.OnPlayerExit == Trigger.EventAction.PlayerWin))
```

```
            {
                mr.sharedMaterial = GameData.main.WinTriggerMaterial;
            }
            else
            {
                mr.sharedMaterial = GameData.main.TriggerMaterial;
            }
        }
    }
    eo.objMat = mr.sharedMaterial;
    Rigidbody ri = eoi.obj.GetComponent<Rigidbody>();
    if (ri)
    {
        ri.useGravity = false;
    }
    eoi.obj.transform.parent =
         GameEditor.main.EditorObjectsContainer.transform;
    if (File.Exists(Directory.GetCurrentDirectory() + imgpath))
    {
        Texture2D tex = new Texture2D(2, 2);
        tex.LoadImage(File.ReadAllBytes(Directory.GetCurrentDirectory() +
         imgpath));
        objItem.GetComponent<UnityEngine.UI.Image>().sprite =
         Sprite.Create(tex, new Rect(0, 0, 256, 256), new Vector2());
    }
    else      //ako ne postoji ikona objekta, renderiraj jednu za taj objekt
    {
        GameEditor.main.renderCamera.gameObject.SetActive(true);
        eoi.obj.transform.position = Vector3.zero;
        main.renderCamera.transform.position =
         eoi.obj.transform.TransformPoint(-1, 1, -1) *
         mr.bounds.size.magnitude;
        eoi.obj.SetActive(true);
        int oldlayer = eoi.obj.layer;
        eoi.obj.layer = 13;
        main.renderCamera.transform.LookAt(mr.bounds.center);
        Renderer rend = eoi.obj.GetComponent<Renderer>();
        Shader oldsh = null;
        bool changed = false;
        if(rend.material)
        {
            if(rend.material.shader == GameData.TransparentShader)
            {
                oldsh = rend.material.shader;
                rend.material.shader = GameData.RenderTransparentShader;
                rend.material.SetFloat("_Mode", 1f);
                rend.material.SetFloat("_Metallic", 1f);
                rend.material.SetFloat("_Glossiness", 0f);
                rend.material.SetFloat("_Cutoff", 0.65f);
                rend.material.EnableKeyword("_ALPHATEST_ON");
                changed = true;
            }
        } else if(rend.sharedMaterial)
        {
            if(rend.sharedMaterial.shader == GameData.TransparentShader)
            {
                oldsh = rend.material.shader;
                rend.sharedMaterial.shader =
```

```
                    GameData.RenderTransparentShader;
                    rend.sharedMaterial.SetFloat("_Mode", 1f);
                    rend.sharedMaterial.SetFloat("_Metallic", 1f);
                    rend.sharedMaterial.SetFloat("_Glossiness", 0f);
                    rend.sharedMaterial.SetFloat("_Cutoff", 0.65f);
                    rend.sharedMaterial.EnableKeyword("_ALPHATEST_ON");
                    changed = true;
                }
            }
            main.renderCamera.Render();
            Texture2D tex = new Texture2D(256, 256, TextureFormat.ARGB32, false);
            RenderTexture.active = main.renderCamera.targetTexture;
            tex.ReadPixels(new Rect(0, 0, 256, 256), 0, 0);
            //tex.alphaIsTransparency = true;
            tex.Apply();
            objItem.GetComponent<UnityEngine.UI.Image>().sprite =
             Sprite.Create(tex, new Rect(0, 0, 256, 256), new Vector2());
            var data = tex.EncodeToPNG();
            File.WriteAllBytes(dir + fileName.Replace(".xml", ".png"), data);
            doc.ChildNodes[0].ChildNodes[doc.ChildNodes[0].ChildNodes.Count -
             1].InnerText = "/Objects/" + fileName.Replace(".xml", ".png");
            if(changed)
            {
                if (rend.material)
                    rend.material.shader = oldsh;
                if (rend.sharedMaterial)
                    rend.sharedMaterial.shader = oldsh;
            }
            eoi.obj.layer = oldlayer;
            eoi.obj.SetActive(false);
            RenderTexture.active = null;
            main.renderCamera.targetTexture.Release();
            GameEditor.main.renderCamera.gameObject.SetActive(false);
            //print("rendered");
        }
        objItem.transform.GetChild(0).GetComponent<UnityEngine.UI.Text>().text =
             fileName.Remove(fileName.Length - 4, 4);
        doc.Save(files[i]);
    }
}
//Load interal objects
for (int i = 0; i < GameData.InteralObjects.Length; i++)
{
    GameObject objItem = Instantiate(GameEditor.main.objectItem);
    objItem.transform.SetParent(GameEditor.main.objectsContainer.transform);
    EditorObjectItem eoi = objItem.GetComponent<EditorObjectItem>();
    //print(files[i]);
    main.LoadedEditorObjects.Add(GameData.InteralObjects[i].src, eoi);
    eoi.obj = Instantiate(GameData.InteralObjects[i].obj);
    eoi.obj.name = GameData.InteralObjects[i].name;
    EditorObject eo = eoi.obj.AddComponent<EditorObject>();
    eo.src = GameData.InteralObjects[i].src;
    eoi.obj.SetActive(false);
    var co = eoi.obj.GetComponent<Collider>();
    if (co)
    {
        if (co.GetType() == typeof(MeshCollider))
            ((MeshCollider)co).convex = true;
```

```
    co.isTrigger = true;
}
if (eoi.obj.layer != 9)
{
    MeshFilter mf = eoi.obj.GetComponent<MeshFilter>();
    if (mf && mf.mesh)
    {
        if (mf.mesh.vertices.Length == 0)
        {
            if (co)
            {
                if (co.GetType() == typeof(BoxCollider))
                {
                    mf.sharedMesh = GameData.GameMeshes[0];
                }
                else
            if (co.GetType() == typeof(SphereCollider))
                {
                    mf.sharedMesh = GameData.GameMeshes[1];
                }
                else
            if (co.GetType() == typeof(CapsuleCollider))
                {
                    mf.sharedMesh = GameData.GameMeshes[2];
                }
                else if (co.GetType() == typeof(MeshCollider))
                {
                    mf.sharedMesh =
                      eoi.obj.GetComponent<MeshCollider>().sharedMesh;
                }
            }
            else
            {
                mf.mesh = null;
            }
        }
    }
    if (!mf)
    {
        if (co)
        {
            if (!mf)
                mf = eoi.obj.AddComponent<MeshFilter>();
            if (co.GetType() == typeof(BoxCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[0];
            }
            else
            if (co.GetType() == typeof(SphereCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[1];
            }
            else
            if (co.GetType() == typeof(CapsuleCollider))
            {
                mf.sharedMesh = GameData.GameMeshes[2];
            }
            else if (co.GetType() == typeof(MeshCollider))
```

```
                {
                    mf.sharedMesh =
                    eoi.obj.GetComponent<MeshCollider>().sharedMesh;
                }
            }
        }
        else
        {
            if (!co)
            {
                MeshCollider mc = eoi.obj.AddComponent<MeshCollider>();
                co = mc;
                mc.convex = true;
                mc.isTrigger = true;
                mc.sharedMesh = mf.sharedMesh;
            }
        }
        MeshRenderer mr = eoi.obj.GetComponent<MeshRenderer>();
        if (!mr)
        {
            mr = eoi.obj.AddComponent<MeshRenderer>();
            Trigger t = eoi.obj.GetComponent<Trigger>();
            if (t && (t.OnPlayerEnter == Trigger.EventAction.PlayerWin ||
             t.OnPlayerStay == Trigger.EventAction.PlayerWin || t.OnPlayerExit ==
                    Trigger.EventAction.PlayerWin))
            {
                mr.sharedMaterial = GameData.main.WinTriggerMaterial;
            }
            else
            {
                mr.sharedMaterial = GameData.main.TriggerMaterial;
            }
        }
        if (mr && !mr.sharedMaterial)
        {
            if (eoi.obj)
            {
                mr.sharedMaterial = GameData.main.InvincibleColliderMaterial;
            }
            else
            {
                Trigger t = eoi.obj.GetComponent<Trigger>();
                if (t && (t.OnPlayerEnter == Trigger.EventAction.PlayerWin ||
                    t.OnPlayerStay == Trigger.EventAction.PlayerWin ||
                    t.OnPlayerExit == Trigger.EventAction.PlayerWin))
                {
                    mr.sharedMaterial = GameData.main.WinTriggerMaterial;
                }
                else
                {
                    mr.sharedMaterial = GameData.main.TriggerMaterial;
                }
            }
        }
        eo.objMat = mr.sharedMaterial;
    } else      //objekt je AI
    {
        //eo.objMat =
```

```
        ((eoi.obj.transform.GetChild(0).GetComponent<SkinnedMeshRenderer>()) !=
    null ??
    eoi.obj.transform.GetChild(0).GetComponent<SkinnedMeshRenderer>().sharedMaterial :
    eoi.obj.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
            SkinnedMeshRenderer skmr =
          eoi.obj.transform.GetChild(0).GetComponent<SkinnedMeshRenderer>();
            if (skmr)
            {
                eo.objMat = skmr.sharedMaterial;
            } else
            {
                MeshRenderer mr =
                 eoi.obj.transform.GetChild(0).GetComponent<MeshRenderer>();
                eo.objMat = mr.sharedMaterial;
            }
        }
        Rigidbody ri = eoi.obj.GetComponent<Rigidbody>();
        if (ri)
        {
            ri.useGravity = false;
        }
        eoi.obj.transform.parent = GameEditor.main.EditorObjectsContainer.transform;
        objItem.GetComponent<UnityEngine.UI.Image>().sprite =
          Sprite.Create(GameData.InteralObjects[i].tex, new Rect(0, 0, 256, 256), new
          Vector2());
        objItem.transform.GetChild(0).GetComponent<UnityEngine.UI.Text>().text =
          GameData.InteralObjects[i].src;
    }
}
```

Nakon toga slijedi deklaracija dviju listi za UI objekte hijerarhije i objekte točaka kamere te referenca na LineRenderer komponentu te indeks trenutnog UI objekta hijerarhije.

```
List<GameObject> hierarchyItems = new List<GameObject>();

List<GameObject> camGizmoObjs = new List<GameObject>();

LineRenderer lr;

int ci = 0;
```

Zatim slijede metode za zbrajanje sve aktivne djece jednog objekta, preimenovanje objekata koji iniciraju točke kamere (crvene kuglice), i metoda za imenovanje UI objekata hijerarhije ovisno o tome koje su dijete, unuk…

```
int countAllActiveChildren(Transform t)
{
    int count = 0;
    for (int i = 0; i < t.childCount; i++)
    {
        if (t.GetChild(i).gameObject.activeSelf)
        {
            count++;
            if (t.GetChild(i).childCount > 0)
```

```
                count += countAllActiveChildren(t.GetChild(i));
            }
        }
        return count;
    }

    void renameGizmoObjs()
    {
        for (int i = 0; i < camGizmoObjs.Count; i++)
        {
            camGizmoObjs[i].name = "CameraPoint" + (i + 1);
        }
    }

    string lvlToStr(int l)
    {
        if (l == 0)
        {
            return "";
        }
        else
        {
            System.Text.StringBuilder sb = new System.Text.StringBuilder();
            for (int i = 0; i < l; i++)
                sb.Append("\t");
            return sb.ToString();
        }
    }
```

Zatim slijedi metoda za izgradnju hijerarhije, uglavnom ako u grupi za UI objekte hijerarhije već posoji jedan takav objekt, on samo biva ažuriran, ako je prazan (null) ili ipak treba dodati novi, dodaje novi, ako objekt ima svoju djecu, rekurzivno se poziva za taj objekt i trenutna razina „dijece“ se poveća za jedan.

```
    void buildHierarchy(Transform t, int level)
    {
        for (int i = 0; i < t.childCount; i++)
        {
            while (i < t.childCount && (!t.GetChild(i).gameObject.activeSelf ||
              !t.GetChild(i).GetComponent<EditorObject>())) i++;
            if (i >= t.childCount)
                break;
            if (ci >= hierarchyItems.Count)
            {
                hierarchyItems.Add(Instantiate(hierarchyItem));
                hierarchyItems[ci].transform.SetParent(hierarchyContainer.transform);
                hierarchyItems[ci].GetComponent<UnityEngine.UI.Text>().text =
                     lvlToStr(level) + t.GetChild(i).name;
                EditorHiererachyItem ehi =
                     hierarchyItems[ci].GetComponent<EditorHiererachyItem>();
                ehi.refObj = t.GetChild(i).gameObject;
                ehi.selected = selectedObjects.Contains(ehi.refObj);
            }
            if (hierarchyItems[ci] != null)
            {
```

```
                hierarchyItems[ci].SetActive(true);
                hierarchyItems[ci].GetComponent<UnityEngine.UI.Text>().text =
                    lvlToStr(level) + t.GetChild(i).name;
                EditorHiererachyItem ehi =
              hierarchyItems[ci].GetComponent<EditorHiererachyItem>();
                ehi.refObj = t.GetChild(i).gameObject;
                ehi.selected = selectedObjects.Contains(ehi.refObj);
            }
            else
            {
                hierarchyItems.Add(Instantiate(hierarchyItem));
                hierarchyItems[ci].transform.SetParent(hierarchyContainer.transform);
                hierarchyItems[ci].GetComponent<UnityEngine.UI.Text>().text =
                    lvlToStr(level) + t.GetChild(i).name;
                EditorHiererachyItem ehi =
                    hierarchyItems[ci].GetComponent<EditorHiererachyItem>();
                ehi.refObj = t.GetChild(i).gameObject;
                ehi.selected = selectedObjects.Contains(ehi.refObj);
            }
            ci++;
            if (t.GetChild(i).childCount > 0)
                buildHierarchy(t.GetChild(i), level + 1);
        }
    }
```

Nakon toga slijedi metoda za ažuriranje hijerarhije koja ponovo gradi hijerarhiju počevši od root objekta.

```
    public void UpdateHierarchy()
    {
        Transform root = Scene.rootObject.transform;
        int objCount = countAllActiveChildren(root);
        for (int i = 0; i < hierarchyItems.Count; i++)
        {
            hierarchyItems[i].SetActive(false);
        }
        if (objCount > 0)
        {
            ci = 0;
            buildHierarchy(root, 1);
        }
    }
```

Onda slijede metode za selekciju i deselekciju objekata, uglavnom kada je objekt selektiran pokazan je u žutoj boji, tj. promijenjen mu je materijal, objekt se nalazi na listi selektiranih objekata, deselektiranje objekata radi obrnuti proces, vraća njegov pravi materijal te ga uklanja s list selektiranih objekata.

```
    public void selectObject(GameObject obj)
    {
        if (obj.GetComponent<Player>())
        {
            playerItem.selected = true;
```

```
            Player.main.mr.sharedMaterial = GameData.main.editorSelectedMaterial;
            selectedObjects.Add(obj);
        }
        else if (obj.tag == "EditorCam")
        {
            cameraItem.selected = true;
            obj.GetComponent<MeshRenderer>().sharedMaterial =
              GameData.main.editorSelectedMaterial;
            selectedObjects.Add(obj);
        }
        else
        {
            GameEditor.main.selectedObjects.Add(obj);
            if (obj.layer != 9)
            {
                var mr = obj.GetComponent<MeshRenderer>();
                if (mr)
                {
                    mr.material = GameData.main.editorSelectedMaterial;
                }
            } else
            {
                if (obj.transform.childCount > 0)
                {
                    var mr = obj.transform.GetChild(0).GetComponent<MeshRenderer>();
                    if (mr)
                    {
                        mr.material = GameData.main.editorSelectedMaterial;
                    }

                    else
                    {
                        var skmr =
                     obj.transform.GetChild(0).GetComponent<SkinnedMeshRenderer>();
                        if (skmr)
                        {
                            skmr.sharedMaterial = GameData.main.editorSelectedMaterial;
                        }
                    }
                } else
                {
                    var mr = obj.GetComponent<MeshRenderer>();
                    if (mr)
                    {
                        mr.material = GameData.main.editorSelectedMaterial;
                    }
                }
            }
            UpdateHierarchy();
        }
        dragging = false;
        scales.Clear();
        rots.Clear();
        dragOffsetPos.Clear();
    }

    public void deselectObject(GameObject obj)
    {
```

```
    if (obj.GetComponent<Player>())
    {
        playerItem.selected = false;
        Player.main.mr.material = obj.GetComponent<EditorObject>().objMat;
        selectedObjects.Remove(obj);
    }
    else if (obj.tag == "EditorCam")
    {
        cameraItem.selected = false;
        obj.GetComponent<MeshRenderer>().sharedMaterial =
          obj.GetComponent<EditorObject>().objMat;
        selectedObjects.Remove(obj);
    }
    else
    {
        selectedObjects.Remove(obj);
        if (obj.layer != 9)
        {
            var mr = obj.GetComponent<MeshRenderer>();
            if (mr)
                mr.sharedMaterial = obj.GetComponent<EditorObject>().objMat;
        } else
        {
            if (obj.transform.childCount > 0)
            {
                var mr = obj.transform.GetChild(0).GetComponent<MeshRenderer>();
                if (mr)
                    mr.sharedMaterial = obj.GetComponent<EditorObject>().objMat;
                else
                {
                    var skmr =
                 obj.transform.GetChild(0).GetComponent<SkinnedMeshRenderer>();
                    if (skmr)
                        skmr.sharedMaterial =
                 obj.GetComponent<EditorObject>().objMat;
                }
            } else
            {
                var mr = obj.GetComponent<MeshRenderer>();
                if (mr)
                    mr.sharedMaterial = obj.GetComponent<EditorObject>().objMat;
            }
        }
        UpdateHierarchy();
    }
}

public void deselectAll()
{
    if (selectedObjects.Count > 0)
    {
        playerItem.selected = false;
        cameraItem.selected = false;
        for (int i = 0; i < selectedObjects.Count; i++)
        {
            if (selectedObjects[i])
            {
                var mr = selectedObjects[i].GetComponent<MeshRenderer>();
```

```
                    if (mr)
                        mr.sharedMaterial =
                     selectedObjects[i].GetComponent<EditorObject>().objMat;
                    else if (selectedObjects[i].GetComponent<Player>())
                        Player.main.mr.sharedMaterial =
                     selectedObjects[i].GetComponent<EditorObject>().objMat;
                    if(selectedObjects[i].layer == 9)
                    {
                        if (selectedObjects[i].transform.childCount > 0)
                        {
                            var amr =
              selectedObjects[i].transform.GetChild(0).GetComponent<MeshRenderer>();
                            if (amr)
                                amr.sharedMaterial =
              selectedObjects[i].GetComponent<EditorObject>().objMat;
                            else
                            {
                                var skmr =
              selectedObjects[i].transform.GetChild(0)
              .GetComponent<SkinnedMeshRenderer>();
                                if (skmr)
                                    skmr.sharedMaterial =
              selectedObjects[i].GetComponent<EditorObject>().objMat;
                            }
                        } else
                        {
                            mr = selectedObjects[i].GetComponent<MeshRenderer>();
                            if (mr)
                                mr.sharedMaterial =
              selectedObjects[i].GetComponent<EditorObject>().objMat;
                        }
                    }
                }
            }
            selectedObjects.Clear();
        }
        UpdateHierarchy();
    }
```

Nakon toga slijede metode pozivane od UI objekata (InputField) koje služe za postavljanje rotacije objekata, pozicije, imena, veličine i roditelja trenutno selektiranog objekta.

```
    public void applyPos(string v)
    {
        if (selectedObjects.Count > 0)
        {
            try
            {
                Vector3 pos = LevelLoader.ParseType<Vector3>(v);
                selectedObjects[0].transform.position = pos;
            }
#pragma warning disable CS0168 // Variable is declared but never used
            catch (System.Exception e)
#pragma warning restore CS0168 // Variable is declared but never used
            {
                posInput.text = selectedObjects[0].transform.position.ToString();
```

```
            }
        }
    }

    public void applyRot(string v)
    {
        if (selectedObjects.Count > 0 && !selectedObjects[0].GetComponent<Player>() &&
              selectedObjects[0].tag != "EditorCamPoint")
        {
            try
            {
                Vector3 rot = LevelLoader.ParseType<Vector3>(v);
                selectedObjects[0].transform.eulerAngles = rot;
                rots.Clear();
            }
#pragma warning disable CS0168 // Variable is declared but never used
            catch (System.Exception e)
#pragma warning restore CS0168 // Variable is declared but never used
            {
                rotInput.text = selectedObjects[0].transform.eulerAngles.ToString();
            }
        }
    }

    public void applyScale(string v)
    {
        if (selectedObjects.Count > 0 && !selectedObjects[0].GetComponent<Player>() &&
              selectedObjects[0].tag != "EditorCam" && selectedObjects[0].tag !=
              "EditorCamPoint")
        {
            try
            {
                Vector3 s = LevelLoader.ParseType<Vector3>(v);
                selectedObjects[0].transform.localScale = s;
                scales.Clear();
            }
#pragma warning disable CS0168 // Variable is declared but never used
            catch (System.Exception e)
#pragma warning restore CS0168 // Variable is declared but never used
            {
                scaleInput.text = selectedObjects[0].transform.localScale.ToString();
            }
        }
    }

    public void applyParent(string v)
    {
        if (!string.IsNullOrEmpty(v))
        {
            if (v == "Root")
                selectedObjects[0].transform.parent = Scene.rootObject.transform;
            if (selectedObjects.Count > 0 && !selectedObjects[0].GetComponent<Player>()
                    && selectedObjects[0].tag != "EditorCam" && selectedObjects[0].tag
                    != "EditorCamPoint")
            {
                GameObject p = GameObject.Find(v);
                if (p && p.GetComponent<EditorObject>() && !p.GetComponent<Player>() &&
              selectedObjects[0].tag != "EditorCam" && selectedObjects[0].tag !=
```

```
                        "EditorCamPoint" && isChildOfRoot(p))
                {
                    selectedObjects[0].transform.parent = p.transform;
                    UpdateHierarchy();
                }
                else
                {
                    parentInput.text = selectedObjects[0].transform.parent.name;
                }
            }
            else
            {
                parentInput.text = "";
            }
        }
    }

    public void applyName(string v)
    {
        if (selectedObjects.Count > 0 && !string.IsNullOrEmpty(v))
        {
            if (!selectedObjects[0].GetComponent<Player>() && selectedObjects[0].tag !=
              "EditorCam" && selectedObjects[0].tag != "EditorCamPoint")
                if (!names.Contains(v))
                {
                    if (names.Contains(selectedObjects[0].name))
                        names.Remove(selectedObjects[0].name);
                    names.Add(v);
                    selectedObjects[0].name = v;
                    UpdateHierarchy();
                }
                else
                {
                    nameInput.text = selectedObjects[0].name;
                }
        }
    }
```

Onda slijede metode za imenovanje objekata, prva dodaje broj do imena objekta, druga na postojeće ime povećava broj dok se ne nađe korišteno ime za metodu.

```
    string addNumberIn(string s)
    {
        bool hasNumberIn = false;
        int startIndex = s.Length;
        while (startIndex - 1 >= 0 && char.IsNumber(s[startIndex - 1]))
            startIndex--;
        hasNumberIn = (startIndex != s.Length);
        if (hasNumberIn)
        {
            int n = int.Parse(s.Substring(startIndex));
            s = s.Substring(0, startIndex) + (n + 1).ToString();
            return s;
        }
        else
        {
```

```
            return s + 1;
        }
    }

    public string findAnotherName(string n)
    {
        string s = n;
        while (names.Contains(s))
        {
            s = addNumberIn(s);
        }
        return s;
    }
```

Nakon toga slijede metode za preimenovanje naziva level-a i određivanje minimalne dozvoljene visine koje su zvane pomoću događaja pripadnih UI objekata u postavkama level-a.

```
    public void applyLevelName(string v)
    {
        if (!string.IsNullOrEmpty(v))
        {
            levelName = v;
        }
        else
        {
            levlNameInput.text = levelName;
        }

    }

    public void applyMinHeight(string v)
    {
        float f;
        if (float.TryParse(v, out f))
        {
            minAllowedHeight = f;
            MAHInput.text = f.ToString("F7");
        }
        else
        {
            MAHInput.text = minAllowedHeight.ToString("F7");
        }
    }
```

Zatim slijedi metoda Update koja se kod svakog UI objekta zove jednom u svakom frame-u igre. Ukratko ako je trenutno stanje igre editing kod metode se izvodi, prva stvar koja se radi provjera da li je LMB bio otpušten, ako je smatra se da igrač više ne vuče objekte u editoru, lista za razliku pozicija se čisti, a liste za poziciju i rotaciju se ažuriraju na vrijednosti od pripadnih objekata, također se ispituje da li su pritisnute određene tipke za mijenjanje alata. Onda se u scenu dodaju točke od kamere (crvene kuglice) ako već

nisu i njihov line renderer se ažurira. Ako je selektiran samo jedan objekt, pokazuju se njegove postavke u određenom prozoru, ako nije, UI tog prozora se isključuje. Zatim se pomoću Raycasting-a (slanje zrake od jedne točke u određenom smjeru, provjeravajući da li je zraka što pogodila i ako je onda daje info o tom objektu i pogotku), u ovom slučaju zraka se šalje iz kamere prema smjeru određenim pritiskom miša, uglavnom to je procedura za provjeravanje da li je korisnik kliknuo na neki objekt. U ovom slučaju taj objekt trenutni alat u uporabi, pozicija miša se sprema te se određuje os promjene rotacije, pozicije ili skale, pomicanjem miša ako je korisnik pritisnuo na jedan od alata mijenja se i ono što i navode, za računanje promjene koristi se udaljenost kursora od početne lokacije pritiska koja se množi s određenim faktorom (kod veličine i rotacije) početne vrijednosti prije promjene odrađene alatom se spremaju na pripade liste. Nakon toga se provjerava dali su pritisnute tipke Ctrl i D, ako jesu, selektirani objekti se dupliciraju i selektiraju (ako ih ima), naravno, kameru i igrača nije moguće duplicirati. Zatim se provjerava da li je igrač pritisnuo Delete, ako je selektirani objekti se brišu, kameru i igrača nije moguće obrisati. Nakon toga slijedi niz naredbi koje ispituju da li igrač kliknuo na neki objekt te ako je, ispituje da li ga treba selektirati ili deselektirati, ako igrač nije kliknuo na ništa, svi objekti se deselektiraju.

```
void Update()
{
    if (Scene.currentGameState == Scene.GameState.editing)
    {
        if (Input.GetKeyUp(KeyCode.Mouse0))
        {
            dragging = false;
            dragOffsetPos.Clear();
            for (int i = 0; i < scales.Count && i < selectedObjects.Count; i++)
            {
                scales[i] = selectedObjects[i].transform.localScale;
            }
            for (int i = 0; i < rots.Count && i < selectedObjects.Count; i++)
                rots[i] = selectedObjects[i].transform.eulerAngles;
        }
        if (Input.GetKeyDown(KeyCode.W))
            currentTool = EditTool.move;
        if (Input.GetKeyDown(KeyCode.E))
            currentTool = EditTool.rotate;
        if (Input.GetKeyDown(KeyCode.R))
            currentTool = EditTool.scale;
        bool mouseIsInGameWindow = (Input.mousePosition.x / Screen.width >= 0.25f &&
          Input.mousePosition.x / Screen.width <= 0.75f) && (Input.mousePosition.y /
          Screen.height >= 0.33f) && !editingSettings;
        if (cameraModeInput.value == (int)PlayerCamera.PlayerCameraMode.PointLookAt)
        {
```

```
    if (!lr)
    {
        lr =
  Instantiate(lineRendererTemplate.gameObject).GetComponent<LineRenderer>();
    }
    lr.gameObject.SetActive(true);
    if (camGizmoObjs.Count == 0)
    {
        var o = Instantiate(pointSphere, editorCamera.transform.position +
  Vector3.forward, Quaternion.identity, EditorObjectsContainer.transform);
        o.name = "CameraPoint1";
        camGizmoObjs.Add(o);
    }
    lr.positionCount = camGizmoObjs.Count;
    for (int i = 0; i < camGizmoObjs.Count; i++)
    {
        lr.SetPosition(i, camGizmoObjs[i].transform.position);
    }
}
else
{
    for (int i = 0; i < camGizmoObjs.Count; i++)
    {
        camGizmoObjs[i].SetActive(false);
    }
    if (lr)
        lr.gameObject.SetActive(false);
}
if (selectedObjects.Count == 1 && mouseIsInGameWindow)
{
    PropertyContainer.SetActive(true);
    posInput.text = selectedObjects[0].transform.position.ToString();
    rotInput.text = selectedObjects[0].transform.eulerAngles.ToString();
    scaleInput.text = selectedObjects[0].transform.localScale.ToString();
    nameInput.text = selectedObjects[0].name;
    if (selectedObjects[0].transform.parent)
        parentInput.text = selectedObjects[0].transform.parent.name;
    else
        parentInput.text = "";
}
else if (selectedObjects.Count != 1)
{
    PropertyContainer.SetActive(false);
}
if (selectedObjects.Count > 0)
{
    Vector3 pos = Vector3.zero;
    for (int i = 0; i < selectedObjects.Count; i++)
        pos += selectedObjects[i].transform.position;
    pos /= selectedObjects.Count;
    switch (currentTool)
    {
        case EditTool.move:
            moveTool.SetActive(true);
            rotateTool.SetActive(false);
            scaleTool.SetActive(false);
            moveTool.transform.position = pos;
            moveTool.transform.localScale = Vector3.one *
```

```
            (Vector3.Distance(moveTool.transform.position,
            Camera.main.transform.position) / startCameraDistanceFromTools);
                break;
            case EditTool.rotate:
                moveTool.SetActive(false);
                rotateTool.SetActive(true);
                scaleTool.SetActive(false);
                rotateTool.transform.position = pos;
                rotateTool.transform.localScale = Vector3.one *
                    (Vector3.Distance(moveTool.transform.position,
                    Camera.main.transform.position) /
            startCameraDistanceFromTools);
                break;
            case EditTool.scale:
                moveTool.SetActive(false);
                rotateTool.SetActive(false);
                scaleTool.SetActive(true);
                scaleTool.transform.position = pos;
                scaleTool.transform.localScale = Vector3.one *
                    (Vector3.Distance(moveTool.transform.position,
                    Camera.main.transform.position) /
                    startCameraDistanceFromTools);
                break;
            default:
                break;
        }
        if (Input.GetKey(KeyCode.Mouse0) && mouseIsInGameWindow)
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 10000f, editorToolsLayer,
            QueryTriggerInteraction.Collide))
            {
                if (!dragging)
                {
                    dragging = true;
                    if (hit.collider.gameObject.name == "MoveAll" ||
                    hit.collider.gameObject.name == "ScaleAll" ||
            hit.collider.gameObject.name == "EditorRotateObject")
                        currentAxis = EditAxis.all;
                    if (hit.collider.gameObject.name == "XAxis" ||
                          hit.collider.name == "RotateX")
                        currentAxis = EditAxis.x;
                    if (hit.collider.gameObject.name == "YAxis" ||
                          hit.collider.name == "RotateY")
                        currentAxis = EditAxis.y;
                    if (hit.collider.gameObject.name == "ZAxis" ||
                          hit.collider.name == "RotateZ")
                        currentAxis = EditAxis.z;
                    dragScreenPoint = Camera.main.WorldToScreenPoint(pos);
                    startDragPoint = Input.mousePosition;
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        dragOffsetPos.Add(selectedObjects[i].transform.position -
                          Camera.main.ScreenToWorldPoint(new
                          Vector3(Input.mousePosition.x, Input.mousePosition.y,
                          dragScreenPoint.z)));
                        scales.Add(selectedObjects[i].transform.localScale);
```

```
        }
    }
}
switch (currentTool)

{
    case EditTool.move:
        if (dragging)
        {
            Vector3 curScreenPoint = new
              Vector3(Input.mousePosition.x, Input.mousePosition.y,
              dragScreenPoint.z);
            switch (currentAxis)
            {
                case EditAxis.all:
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        if (selectedObjects.Count >
                                dragOffsetPos.Count)
                        {
                            for (int j = dragOffsetPos.Count; j <
                                selectedObjects.Count; j++)

        dragOffsetPos.Add(selectedObjects[j].transform.position –
        Camera.main.ScreenToWorldPoint(new
        Vector3(Input.mousePosition.x, Input.mousePosition.y,
        dragScreenPoint.z)));
                        }
                        Vector3 curPosition =
 Camera.main.ScreenToWorldPoint(curScreenPoint) + dragOffsetPos[i];
                        selectedObjects[i].transform.position =
                            curPosition;
                    }
                    break;
                case EditAxis.x:
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        if (selectedObjects.Count >
                     dragOffsetPos.Count)
                        {
                            for (int j = dragOffsetPos.Count; j <
                            selectedObjects.Count; j++)

                            dragOffsetPos.Add(selectedObjects[j].tran
                            sform.position -
                            Camera.main.ScreenToWorldPoint(new
                            Vector3(Input.mousePosition.x,
                            Input.mousePosition.y,
                            dragScreenPoint.z)));
                        }
                        Vector3 curPosition =
                     Camera.main.ScreenToWorldPoint(curScreenPoint) +
                     dragOffsetPos[i];
                        selectedObjects[i].transform.position = new
                            Vector3(curPosition.x,
                     selectedObjects[i].transform.position.y,
                     selectedObjects[i].transform.position.z);
                    }
```

```
                    break;
                case EditAxis.y:
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        if (selectedObjects.Count >
                            dragOffsetPos.Count)
                        {
                            for (int j = dragOffsetPos.Count; j <
                            selectedObjects.Count; j++)

                dragOffsetPos.Add(selectedObjects[j].transform.position
                - Camera.main.ScreenToWorldPoint(new
                Vector3(Input.mousePosition.x, Input.mousePosition.y,
                dragScreenPoint.z)));
                        }
                        Vector3 curPosition =
                       Camera.main.ScreenToWorldPoint(curScreenPoint) +
                dragOffsetPos[i];
                        selectedObjects[i].transform.position = new
                Vector3(selectedObjects[i].transform.position.x,
                curPosition.y,
                selectedObjects[i].transform.position.z);
                    }
                    break;
                case EditAxis.z:
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        if (selectedObjects.Count >
                            dragOffsetPos.Count)
                        {
                            for (int j = dragOffsetPos.Count; j <
                selectedObjects.Count; j++)

dragOffsetPos.Add(selectedObjects[j].transform.position –
Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x,
Input.mousePosition.y, dragScreenPoint.z)));
                        }
                        Vector3 curPosition =
Camera.main.ScreenToWorldPoint(curScreenPoint) + dragOffsetPos[i];
                        selectedObjects[i].transform.position = new
       Vector3(selectedObjects[i].transform.position.x,
       selectedObjects[i].transform.position.y, curPosition.z);
                    }
                    break;
            }
        }
        break;
    case EditTool.rotate:
        if (dragging)
        {
            switch (currentAxis)
            {
                case EditAxis.x:
                    float d = (Input.mousePosition.y –
       startDragPoint.y) / (Screen.height * 0.5f);
                    for (int i = 0; i < selectedObjects.Count; i++)
                    {
                        if (rots.Count < selectedObjects.Count)
```

```
                                {
                                    for (int j = rots.Count; j <
                            selectedObjects.Count; j++)

                        rots.Add(selectedObjects[j].transform.eulerAngles);
                                }
                                if (!selectedObjects[i].GetComponent
                        <Player>() && selectedObjects[i].tag != "EditorCamPoint")
                                {
                                    selectedObjects[i].transform.eulerAngles
                = new Vector3(rots[i].x + d * rotationFactor, rots[i].y, rots[i].z);
                                }
                            }
                            break;
                        case EditAxis.y:
                            float e = -(Input.mousePosition.x –
                        startDragPoint.x) / (Screen.width * 0.5f);
                            for (int i = 0; i < selectedObjects.Count; i++)
                            {
                                if (rots.Count < selectedObjects.Count)
                                {
                                    for (int j = rots.Count; j <
                                 selectedObjects.Count; j++)

                                 rots.Add(selectedObjects[j].transform.eulerAngle
                                 s);
                                }
                                if (!selectedObjects[i]
        .GetComponent<Player>() && selectedObjects[i].tag != "EditorCamPoint")
                                {
                                    selectedObjects[i].transform.eulerAngles
                = new Vector3(rots[i].x, rots[i].y + e * rotationFactor, rots[i].z);
                                }
                            }
                            break;
                        case EditAxis.z:
                            float f = (Input.mousePosition.x –
                        startDragPoint.x) / (Screen.width * 0.5f);
                            for (int i = 0; i < selectedObjects.Count; i++)
                            {
                                if (rots.Count < selectedObjects.Count)
                                {
                                    for (int j = rots.Count; j <
                                 selectedObjects.Count; j++)

                        rots.Add(selectedObjects[j].transform.eulerAngles);
                                }
                                        if (!selectedObjects[i]
        .GetComponent<Player>() && selectedObjects[i].tag != "EditorCamPoint")
                                {
                                    selectedObjects[i].transform.eulerAngles
                = new Vector3(rots[i].x, rots[i].y, rots[i].z + f * rotationFactor);
                                }
                            }
                            break;
                        default:
                            break;
                    }
```

```
            }
            break;

        case EditTool.scale:
            if (dragging)
            {
                switch (currentAxis)
                {
                    case EditAxis.all:
                        float d = (Input.mousePosition.x –
                   startDragPoint.x) / (Screen.width * 0.5f);
                        for (int i = 0; i < selectedObjects.Count; i++)
                        {
                            if (scales.Count < selectedObjects.Count)
                            {
                                for (int j = scales.Count; j <
                             selectedObjects.Count; j++)

                scales.Add(selectedObjects[j].transform.localScale);
                            }
                            if (!selectedObjects[i].
                                GetComponent<Player>() &&
                                selectedObjects[i].tag != "EditorCam" &&
                                selectedObjects[i].tag !=
                                "EditorCamPoint")
                                selectedObjects[i].transform.localScale =
                                scales[i] * Mathf.Pow(scaleFactor, d);
                        }
                        break;
                    case EditAxis.x:
                        float e = (Input.mousePosition.x –
                          startDragPoint.x) / (Screen.width * 0.5f);
                        for (int i = 0; i < selectedObjects.Count; i++)
                        {
                            if (scales.Count < selectedObjects.Count)
                            {
                                for (int j = scales.Count; j <
                                selectedObjects.Count; j++)

                       scales.Add(selectedObjects[j].transform.localScale);
                            }
                            if (!selectedObjects[i].
                                GetComponent<Player>() &&
                                selectedObjects[i].tag != "EditorCam" &&
                                selectedObjects[i].tag !=
                                "EditorCamPoint")
                            {
                                selectedObjects[i].transform.localScale =
new Vector3(scales[i].x * Mathf.Pow(scaleFactor, e), scales[i].y,
                                scales[i].z);
                            }
                        }
                        break;
                    case EditAxis.y:
                        float f = (Input.mousePosition.y –
                          startDragPoint.y) / (Screen.height * 0.5f);
                        for (int i = 0; i < selectedObjects.Count; i++)
                        {
```

```
                            if (scales.Count < selectedObjects.Count)
                            {
                                for (int j = scales.Count; j <
                                       selectedObjects.Count; j++)

                        scales.Add(selectedObjects[j].transform.localScale);
                            }
                            if (!selectedObjects[i].
                                GetComponent<Player>() &&
                                selectedObjects[i].tag != "EditorCam" &&
                                selectedObjects[i].tag !=
                                "EditorCamPoint")
                            {
                                selectedObjects[i].transform.localScale =
                        new Vector3(scales[i].x, scales[i].y *
                        Mathf.Pow(scaleFactor, f), scales[i].z);
                            }
                        }
                        break;
                      case EditAxis.z:
                        float g = (Input.mousePosition.x -
                          startDragPoint.x) / (Screen.width * 0.5f);
                        for (int i = 0; i < selectedObjects.Count; i++)
                        {
                            if (scales.Count < selectedObjects.Count)
                            {
                                for (int j = scales.Count; j <
                                selectedObjects.Count; j++)

                        scales.Add(selectedObjects[j].transform.localScale);
                            }
                            if (!selectedObjects[i].
                                GetComponent<Player>() &&
                                selectedObjects[i].tag != "EditorCam" &&
                                selectedObjects[i].tag !=
                                "EditorCamPoint")
                            {
                                selectedObjects[i].transform.localScale =
                        new Vector3(scales[i].x, scales[i].y, scales[i].z *
                        Mathf.Pow(scaleFactor, g));
                            }
                        }
                        break;
                    }
                }
                break;
            }
        }
                        if (Input.GetKey(KeyCode.LeftControl) &&
         Input.GetKeyDown(KeyCode.D))   ///Duplicate Objects
        {
            for (int i = 0; i < selectedObjects.Count; i++)
            {
                if (!selectedObjects[i].GetComponent<Player>() &&
                    selectedObjects[i].tag != "EditorCam" &&
                 selectedObjects[i].tag != "EditorCamPoint")
                {
                    var o = Instantiate(selectedObjects[i]);
```

```
                o.transform.parent = selectedObjects[i].transform.parent;
                o.name = selectedObjects[i].name;
                if (names.Contains(o.name))
                {
                    o.name = findAnotherName(o.name);
                    names.Add(o.name);
                }
                else
                {
                    names.Add(o.name);
                }
                if (o.layer != 9)
                {
                    MeshRenderer mr =
                selectedObjects[i].GetComponent<MeshRenderer>();
                    if (mr) mr.sharedMaterial =
                selectedObjects[i].GetComponent<EditorObject>().objMat;
                } else
                {
                    MeshRenderer mr =
                       selectedObjects[i].transform.GetChild(0).
                       GetComponent<MeshRenderer>();
                    if (mr) mr.sharedMaterial =
                selectedObjects[i].GetComponent<EditorObject>().objMat;
                    else
                    {
                        SkinnedMeshRenderer skmr =
                       selectedObjects[i].transform.GetChild(0).
                       GetComponent<SkinnedMeshRenderer>();
                         if(skmr)
                             skmr.sharedMaterial =
                selectedObjects[i].GetComponent<EditorObject>().objMat;
                    }
                }

                selectedObjects[i] = o;
            }
            else if (selectedObjects[i].tag == "EditorCamPoint")
            {
                var o = Instantiate(selectedObjects[i]);
                o.name = "CameraPoint" + (camGizmoObjs.Count + 1);
                o.transform.parent = EditorObjectsContainer.transform;
                deselectObject(selectedObjects[i]);
                selectObject(o);
                camGizmoObjs.Add(o);
            }
        }
        UpdateHierarchy();
    }
    if (Input.GetKey(KeyCode.Delete))   //Delete Object
    {
        for (int i = 0; i < selectedObjects.Count; i++)
        {
            if (!selectedObjects[i].GetComponent<Player>() &&
                selectedObjects[i].tag != "EditorCam" &&
                selectedObjects[i].tag != "EditorCamPoint")
            {
                selectedObjects[i].SetActive(false);
```

```
                names.Remove(selectedObjects[i].name);
                Destroy(selectedObjects[i]);
            }
            else if (selectedObjects[i].tag == "EditorCamPoint" &&
                      camGizmoObjs.Count > 1)
            {
                selectedObjects[i].SetActive(false);
                Destroy(selectedObjects[i]);
                camGizmoObjs.Remove(selectedObjects[i]);
                renameGizmoObjs();
            }
        }

        dragOffsetPos.Clear();
        scales.Clear();
        rots.Clear();
        dragging = false;
        deselectAll();
    }
}
else
{
    moveTool.SetActive(false);
    rotateTool.SetActive(false);
    scaleTool.SetActive(false);
    dragging = false;
}
if (Input.GetKeyDown(KeyCode.Mouse0) && mouseIsInGameWindow && !dragging)
{
    dragging = false;
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, 10000f))
    {
        if (hit.collider.GetComponent<EditorObject>())
        {
            GameObject obj = hit.collider.gameObject;
            if (Input.GetKey(KeyCode.LeftShift))
            {
                if (!selectedObjects.Contains(obj))
                {
                    selectObject(obj);
                }
                else
                {
                    deselectObject(obj);
                }
            }
            else
            {
                if (selectedObjects.Count > 0)
                {
                    deselectAll();
                    selectObject(obj);
                }
                else
                {
                    selectObject(obj);
```

```
                        }
                    }
                }
            }
            else
            {
                if (!Input.GetKey(KeyCode.LeftShift))
                    deselectAll();
            }
        }
    }
}
```

Nakon toga slijede dvije identične metode za brisanje svih child objekata, pošto se objekti zapravo brišu pri kraju frame-a u igri, druga metoda svakom objektu mijenja ime po zadanom argumentu te time se drugim skriptama blokira pristup tim objektima preko imena i isključuje ih.

```
static void eraseChildern(GameObject o)
{
    for (int i = 0; i < o.transform.childCount; i++)
        Destroy(o.transform.GetChild(i).gameObject);
}

static void eraseChildern(GameObject o, string n)
{
    for (int i = 0; i < o.transform.childCount; i++)
    {
        o.transform.GetChild(i).name = n;
        o.transform.GetChild(i).gameObject.SetActive(false);
        Destroy(o.transform.GetChild(i).gameObject);
    }
}
```

Zatim slijedi metoda za čišćenje editora koja briše sve privremene vrijednosti, sve dodane objekte, sve stvorene UI objekte i čisti scenu.

```
public static void Clear()
{
    main.deselectAll();
    main.NextLevelContainer.SetActive(false);
    main.CPOInput.text = Vector3.zero.ToString();
    for (int i = 0; i < main.camGizmoObjs.Count; i++)
        Destroy(main.camGizmoObjs[i]);
    main.camGizmoObjs.Clear();
    editSettings = false;
    for (int i = 0; i < main.editLevelStuff.Length; i++)
    {
        main.editLevelStuff[i].SetActive(false);
    }
    loadLevels = false;
    main.CustomLevelListStuff.SetActive(false);
    eraseChildern(main.CustomLevelsListContainer);
```

```
        eraseChildern(main.EditorObjectsContainer);
        for (int i = 0; i < main.objectsContainer.transform.childCount; i++)
        {
            EditorObjectItem eoi =
main.objectsContainer.transform.GetChild(i).GetComponent<EditorObjectItem>();
            if (eoi.copy)
                Destroy(eoi.copy);
            eoi.gameObject.name = "del";
            Destroy(eoi.gameObject);
        }
        main.editorCamera.SetActive(false);
        if (main.lr)
            main.lr.gameObject.SetActive(false);
        main.LoadedEditorObjects.Clear();
        main.names.Clear();
        main.moveTool.SetActive(false);
        main.rotateTool.SetActive(false);
        main.scaleTool.SetActive(false);
        main.scales.Clear();
        main.rots.Clear();
        main.dragOffsetPos.Clear();
    }
```

Onda slijedi metoda koja se poziva od UI objekta koji otvare postavke Level-a.

```
    public void OpenLevelSettings()
    {
        editSettings = !editSettings;

        NextLevelContainer.SetActive(false);

        if (loadLevels)
        {
            CustomLevelListStuff.SetActive(false);
            eraseChildern(CustomLevelsListContainer);
            loadLevels = false;
        }

        if (editSettings)
        {
            for (int i = 0; i < editLevelStuff.Length; i++)
            {
                editLevelStuff[i].SetActive(true);
            }
        }
        else
        {
            for (int i = 0; i < editLevelStuff.Length; i++)
            {
                editLevelStuff[i].SetActive(false);
            }
        }
    }
```

Zatim slijedi metoda koja sprema sve objekte scene u xml datoteku level-a u zadani element kao djecu, objektu koje sprema su djeca danog root objekta.

```
void SaveObjs(XmlElement container, XmlDocument doc, GameObject root)
    {
        Transform rt = root.transform;
        for (int i = 0; i < rt.childCount; i++)
        {
            while (i < rt.childCount && !rt.GetChild(i).gameObject.activeSelf) i++;
            if (i < rt.childCount && rt.GetChild(i).GetComponent<EditorObject>())
            {
                XmlElement el = doc.CreateElement(string.Empty, "Object", string.Empty);
                el.SetAttribute("Name", rt.GetChild(i).name);
                XmlElement t = doc.CreateElement(string.Empty, "Transform",
                     string.Empty);
                XmlElement pos = doc.CreateElement(string.Empty, "Position",
                     string.Empty);
                XmlElement rot = doc.CreateElement(string.Empty, "Rotation",
                     string.Empty);
                XmlElement scale = doc.CreateElement(string.Empty, "Scale",
                     string.Empty);
                XmlElement parent = doc.CreateElement(string.Empty, "Parent",
                     string.Empty);
                XmlText v = doc.CreateTextNode
              (rt.GetChild(i).transform.position.ToString("F7"));
                pos.AppendChild(v);
                v = doc.CreateTextNode
              (rt.GetChild(i).transform.eulerAngles.ToString("F7"));
                rot.AppendChild(v);
                v = doc.CreateTextNode
              (rt.GetChild(i).transform.localScale.ToString("F7"));
                scale.AppendChild(v);
                if (rt.GetChild(i).parent)
                    v = doc.CreateTextNode(rt.GetChild(i).parent.name);
                else
                    v = doc.CreateTextNode("");
                parent.AppendChild(v);
                t.AppendChild(pos);
                t.AppendChild(rot);
                t.AppendChild(scale);
                t.AppendChild(parent);
                el.AppendChild(t);
                XmlElement src = doc.CreateElement(string.Empty, "ObjRef", string.Empty);
                v = doc.CreateTextNode(rt.GetChild(i).GetComponent<EditorObject>().src);
                src.AppendChild(v);
                el.AppendChild(t);
                el.AppendChild(src);
                container.AppendChild(el);
                if (rt.GetChild(i).childCount > 0)
                    SaveObjs(container, doc, rt.GetChild(i).gameObject);
            }
        }
    }
```

Nakon toga slijedi niz metoda pozivanih od UI objekata preko događaja, ono što metode rade govore njihova imena.

```
public void acitivateNextLevelList()
{
    NextLevelContainer.SetActive(true);
    for (int i = 0; i < editLevelStuff.Length; i++)
        editLevelStuff[i].SetActive(false);
    Transform nlg = NextLevelContainer.transform.GetChild(1);
    string[] lvls = Directory.GetDirectories("CustomLevels");
    eraseChildern(nlg.gameObject);
    GameObject noneButton = Instantiate(CustomLevelButton, nlg);
    Transform _tn = noneButton.transform.GetChild(2);
    if (_tn)
        _tn.gameObject.SetActive(false);
    UnityEngine.UI.Text t =
          noneButton.transform.GetChild(0).GetComponent<UnityEngine.UI.Text>();
    t.text = "None";
    t.color = Color.red;
    UnityEngine.UI.Button b = noneButton.GetComponent<UnityEngine.UI.Button>();
    string s = "None";
    b.onClick.AddListener(delegate { GameEditor.main.applyNextLevel(s, "None"); });
    for (int i = 0; i < lvls.Length; i++)
    {
        GameObject lb = Instantiate(CustomLevelButton, nlg);
        _tn = lb.transform.GetChild(2);
        if (_tn)
            _tn.gameObject.SetActive(false);
        UnityEngine.UI.Text lt =
          lb.transform.GetChild(0).GetComponent<UnityEngine.UI.Text>();
        lt.text = ((lvls[i].Replace("CustomLevels", "")).Replace("\\",
          "")).Replace("/", "");
        UnityEngine.UI.Button lbc = lb.GetComponent<UnityEngine.UI.Button>();
        string l = "/" + lvls[i].Replace("\\", "/") + "/Level.xml";
        lbc.onClick.AddListener(delegate { GameEditor.main.applyNextLevel(l,
          lt.text); });
    }
}

public void applyNextLevel(string path, string n)
{
    nextLevelText.text = n;
    nextLevelText.name = path;
    NextLevelContainer.SetActive(false);
    for (int i = 0; i < editLevelStuff.Length; i++)
        editLevelStuff[i].SetActive(true);
}

public void deleteLevel(string path)
{
    if (Directory.Exists(path))
        Directory.Delete(path, true);
    if (loadLevels)
    {
        loadLevels = false;
        activateCustomLevelsList();
```

```
        }
    }

    public void activateCustomLevelsList()
    {
        loadLevels = !loadLevels;
        NextLevelContainer.SetActive(false);
        if (loadLevels)
        {
            if (editSettings)
            {
                for (int i = 0; i < editLevelStuff.Length; i++)
                {
                    editLevelStuff[i].SetActive(false);
                }
                editSettings = false;
            }
            CustomLevelListStuff.SetActive(true);
            if (CustomLevelsListContainer.transform.childCount > 0)
                eraseChildern(CustomLevelsListContainer);
            List<string> levels = new List<string>
              (System.IO.Directory.GetDirectories("CustomLevels"));
            for (int i = 0; i < levels.Count; i++)
            {
                GameObject obj = Instantiate(CustomLevelButton);
                obj.name = levels[i].Remove(0, 13);
                obj.transform.SetParent(CustomLevelsListContainer.transform);
                obj.transform.GetChild(0).GetComponent<UnityEngine.UI.Text>().text =
                     obj.name;
                string arg = System.IO.Directory.GetCurrentDirectory() + "/" + levels[i]
                     + "/Level.xml";
                obj.GetComponent<UnityEngine.UI.Button>().onClick.AddListener(delegate {
                     GameEditor.main.LoadLevel(arg); });
                Transform delb = obj.transform.GetChild(2);
                if (delb)    //delete level button init
                {
                    string arg2 = System.IO.Directory.GetCurrentDirectory() + "/" +
                     levels[i];

              delb.GetComponent<UnityEngine.UI.Button>().onClick.AddListener(delegate {
              GameEditor.main.deleteLevel(arg2); });
                }
            }
        }
        else
        {
            CustomLevelListStuff.SetActive(false);
            eraseChildern(CustomLevelsListContainer);
        }
    }

    public void applyCameraPointOffset(string v)
    {
        try
        {
            LevelLoader.ParseType<Vector3>(v);
        }
#pragma warning disable CS0168 // Variable is declared but never used
```

```
        catch (System.Exception e)
#pragma warning restore CS0168 // Variable is declared but never used
        {
            CPOInput.text = "(0,0,0)";
        }
    }
```

Zatim slijedi metoda za učitavanje postojećeg level-a u editor, iščitava podatke iz dane xml datoteke i na način sličan kao i kod LevelLoader klase učitava objekte u scenu, ovime se editor resetira.

```
   public void LoadLevel(string path)
    {
        eraseChildern(Scene.rootObject, "deleted");
        Clear();
        Setup();
        UpdateHierarchy();
        levelName = Path.GetFileName(Path.GetDirectoryName(path));
        levlNameInput.text = levelName;
        XmlDocument doc = new XmlDocument();
        doc.Load(path);
        XmlNodeList objs = doc.ChildNodes[1].ChildNodes[0].ChildNodes;
        if (objs != null)
        {
            for (int i = 0; i < objs.Count; i++)
            {
                if (objs[i].Attributes.Count != 0)
                {
                                //print(Directory.GetCurrentDirectory() +
                                 objs[i].ChildNodes[1].InnerText);
                    GameObject obj;
                    //print(objs[i].ChildNodes[1].InnerText);
                    if (objs[i].ChildNodes[1].InnerText.Contains("(#Interal)"))
                        obj = Instantiate(LoadedEditorObjects
                     [objs[i].ChildNodes[1].InnerText].obj);
                    else
                        obj = Instantiate(LoadedEditorObjects
                     [Directory.GetCurrentDirectory() +
                     objs[i].ChildNodes[1].InnerText].obj);
                    obj.SetActive(true);
                    //print(obj.activeSelf);
                    obj.name = objs[i].Attributes["Name"].Value;
                    if (!names.Contains(obj.name))
                        names.Add(obj.name);
                    else
                    {
                        obj.name = findAnotherName(obj.name);
                        names.Add(obj.name);
                    }
                    obj.transform.position =
        LevelLoader.ParseType<Vector3> (objs[i].ChildNodes[0].ChildNodes[0].InnerText);
                    obj.transform.eulerAngles = LevelLoader.ParseType<Vector3>
                            (objs[i].ChildNodes[0].ChildNodes[1].InnerText);
                    obj.transform.localScale = LevelLoader.ParseType<Vector3>
                            (objs[i].ChildNodes[0].ChildNodes[2].InnerText);
                    obj.transform.SetParent(Scene.rootObject.transform);
```

```
            if (!string.IsNullOrEmpty
             (objs[i].ChildNodes[0].ChildNodes[3].InnerText))
            {
                GameObject p =
             GameObject.Find(objs[i].ChildNodes[0].ChildNodes[3].InnerText);
                if (p != null)
                {
                    obj.transform.SetParent(p.transform);
                    //print(obj.name + "->" + obj.transform.parent.name);
                    obj.isStatic = false;
                }
            }

        }
    }
}
XmlNode sceneInfo = doc.ChildNodes[1].ChildNodes[1];
MAHInput.text = sceneInfo.ChildNodes[0].InnerText;
XmlNode playerInfo = doc.ChildNodes[1].ChildNodes[2];
XmlNode playerTransform = playerInfo.ChildNodes[0];
Scene.player.transform.position =
      LevelLoader.ParseType<Vector3>(playerTransform.ChildNodes[0].InnerText);
Scene.player.transform.eulerAngles =
      LevelLoader.ParseType<Vector3>(playerTransform.ChildNodes[1].InnerText);
Scene.player.transform.localScale =
      LevelLoader.ParseType<Vector3>(playerTransform.ChildNodes[2].InnerText);
XmlNode cam = doc.ChildNodes[1].ChildNodes[3];
editorCamera.transform.position =
      LevelLoader.ParseType<Vector3>(cam.ChildNodes[0].InnerText);
editorCamera.transform.eulerAngles =
      LevelLoader.ParseType<Vector3>(cam.ChildNodes[1].InnerText);
cameraModeInput.value = (int)(PlayerCamera.PlayerCameraMode)
      System.Enum.Parse(typeof(PlayerCamera.PlayerCameraMode),
      cam.ChildNodes[2].InnerText);
if (cameraModeInput.value == (int)PlayerCamera.PlayerCameraMode.PointLookAt)
{
    XmlNode points = cam.ChildNodes[3];
    //add points
    if (!lr)
    {
        lr = Instantiate(lineRendererTemplate.gameObject).
             GetComponent<LineRenderer>();
    }
    lr.gameObject.SetActive(true);
    if (points.ChildNodes.Count == 0)
    {
        var o = Instantiate(pointSphere, editorCamera.transform.position +
      Vector3.forward, Quaternion.identity, EditorObjectsContainer.transform);
        o.name = "CameraPoint1";
        camGizmoObjs.Add(o);
    }
    for (int i = 0; i < points.ChildNodes.Count; i++)
    {
        var o = Instantiate(pointSphere,
                    LevelLoader.ParseType<Vector3>(points.ChildNodes[i].InnerText)
                    , Quaternion.identity, EditorObjectsContainer.transform);
        o.name = "CameraPoint" + (camGizmoObjs.Count + 1);
        camGizmoObjs.Add(o);
```

```
            }
            lr.positionCount = camGizmoObjs.Count;
            for (int i = 0; i < camGizmoObjs.Count; i++)
            {
                lr.SetPosition(i, camGizmoObjs[i].transform.position);
            }
        }
        XmlNode cpo = cam.ChildNodes[4];
        CPOInput.text = cpo.InnerText;
        XmlNode nextLevel = doc.ChildNodes[1].ChildNodes[4];
        if (string.IsNullOrEmpty(nextLevel.InnerText))
        {
            nextLevelText.name = "None";
            nextLevelText.text = "None";
        }
        else
        {
            nextLevelText.name = nextLevel.InnerText;
            nextLevelText.text = (((nextLevel.InnerText.Replace("CustomLevels",
              "")).Replace("\\", "")).Replace("/", "")).Replace(".xml", "");
        }
        UpdateHierarchy();
    }
```

I na kraju ove najkompleksnije klase u igri nalazi se metoda za spremanje level-a koja se pokreće klikom na gumb za spremanje level-a, sve što trenutno stanje editora sadrži se sprema u datoteku Level.xml u direktoriju imena tog level-a (ako ne postoji, metoda ga stvara), a taj direktorij će biti u direktoriju CustomLevels.

```
    public void SaveLevel()
    {
        XmlDocument doc = new XmlDocument();
        XmlDeclaration xmlDeclaration = doc.CreateXmlDeclaration("1.0", "UTF-16", null);
        XmlElement root = doc.DocumentElement;
        doc.InsertBefore(xmlDeclaration, root);
        XmlElement scene = doc.CreateElement(string.Empty, "Scene", string.Empty);
              //main node
        doc.AppendChild(scene);
        XmlElement objs = doc.CreateElement(string.Empty, "Objects", string.Empty);
              //objects node
        SaveObjs(objs, doc, Scene.rootObject);  //save data about objects
        scene.AppendChild(objs);
        XmlElement sceneInfo = doc.CreateElement(string.Empty, "SceneInfo",
              string.Empty);  //scene info node
        scene.AppendChild(sceneInfo);
        XmlElement minHeight = doc.CreateElement(string.Empty, "MinHeight",
              string.Empty);
        XmlText heightLevel = doc.CreateTextNode(minAllowedHeight.ToString("F7"));
        minHeight.AppendChild(heightLevel);
        sceneInfo.AppendChild(minHeight);
        XmlElement playerInfo = doc.CreateElement(string.Empty, "PlayerInfo",
              string.Empty);    //player info
        scene.AppendChild(playerInfo);
        XmlElement transform = doc.CreateElement(string.Empty, "Transform",
              string.Empty);
```

```
 playerInfo.AppendChild(transform);
 XmlElement pos = doc.CreateElement(string.Empty, "Position", string.Empty);
 XmlElement rot = doc.CreateElement(string.Empty, "Rotation", string.Empty);
 XmlElement scale = doc.CreateElement(string.Empty, "Scale", string.Empty);
 XmlText val = doc.CreateTextNode(Scene.player.transform.position.ToString("F7"));
 pos.AppendChild(val);
 transform.AppendChild(pos);
 val = doc.CreateTextNode(Scene.player.transform.eulerAngles.ToString("F7"));
 rot.AppendChild(val);
 transform.AppendChild(rot);
 val = doc.CreateTextNode(Scene.player.transform.localScale.ToString("F7"));
 scale.AppendChild(val);
 transform.AppendChild(scale);
 XmlElement cameraInfo = doc.CreateElement(string.Empty, "CameraInfo",
       string.Empty);    //camera info
 pos = doc.CreateElement(string.Empty, "Position", string.Empty);
 val = doc.CreateTextNode(cameraItem.refObj.transform.position.ToString("F7"));
 pos.AppendChild(val);
 cameraInfo.AppendChild(pos);
 rot = doc.CreateElement(string.Empty, "Rotation", string.Empty);
 val = doc.CreateTextNode(cameraItem.refObj.transform.eulerAngles.ToString("F7"));
 rot.AppendChild(val);
 cameraInfo.AppendChild(rot);
 XmlElement cameraMode = doc.CreateElement(string.Empty, "CameraMode",
       string.Empty);
 val =
 doc.CreateTextNode(((PlayerCamera.PlayerCameraMode)cameraModeInput.value).ToStrin
 g());
 cameraMode.AppendChild(val);
 cameraInfo.AppendChild(cameraMode);
 XmlElement points = doc.CreateElement(string.Empty, "Points", string.Empty);
 for (int i = 0; i < camGizmoObjs.Count; i++)
 {
     XmlElement point = doc.CreateElement(string.Empty, "Point", string.Empty);

point.AppendChild(doc.CreateTextNode(camGizmoObjs[i].transform.position.ToString("
F7")));
     points.AppendChild(point);
 }
 cameraInfo.AppendChild(points);
 XmlElement po = doc.CreateElement(string.Empty, "PointOffset", string.Empty);
 po.AppendChild(doc.CreateTextNode(CPOInput.text));
 cameraInfo.AppendChild(po);
 scene.AppendChild(cameraInfo);
 XmlElement nextLevel = doc.CreateElement(string.Empty, "NextLevel",
       string.Empty);
 nextLevel.AppendChild(doc.CreateTextNode(nextLevelText.name == "None" ? "" :
       nextLevelText.name));
 scene.AppendChild(nextLevel);
 XmlElement locked = doc.CreateElement(string.Empty, "Locked", string.Empty);
 locked.AppendChild(doc.CreateTextNode("False"));
 scene.AppendChild(locked);
 if (!Directory.Exists(Directory.GetCurrentDirectory() + "/CustomLevels/" +
       levelName))
     Directory.CreateDirectory(Directory.GetCurrentDirectory() + "/CustomLevels/"
       + levelName);
 doc.Save(Directory.GetCurrentDirectory() + "/CustomLevels/" + levelName +
       "/Level.xml");
```

```
#if UNITY_EDITOR
        print("Saved");
#endif
    }

}
```

5. Klasa EditorObject

Klasa EditorObject je komponenta koju svaki objekt s kojim se može baratati u editoru more imati, sadrži samo dvije varijable, originalni materijal koji objekt koristi i izvor iz koje datoteke on dolazi, unutarnji objekti u izvoru imaju posebnu naznaku (#Interal), taj varijabla izvor (src) se kasnije koristi u zapisivanju objekta u level kao referenca u xml kodu.

```
public class EditorObject : MonoBehaviour {

    public Material objMat; //oroginalni materijal objekta

    public string src;  //izvor objekta (relativni path do njegove xml datoteke)

}
```

6. Klasa EditorHierarchyItem

Ova klasa osim što je naslijeđena od klase MonoBehaviour, sadrži i tri sučelja kako bi se objekti koji je koriste mogli vući i koristiti kao drag n drop objekti, svako od sučelja zahtjeva deklaraciju jedne metode i klasa taj zahtjev ispunjava. Objekti koji sadrže ovu klasu su UI objekti u hijerarhiji, svaki takav objekt prezentira jedan objekt u sceni, ti objekti imaju labelu s imenom pripadnog objekta.

Klasa počinje varijablama koje su pripadni objekt ovog item-a, varijable stanja i svojstvo koje mijenja boju labele ako je pripadni objekt selektiran te kako se ovim item-om može upravljati te tekstna komponenta koju ovaj item sadrži.

```
public class EditorHiererachyItem : MonoBehaviour, IDropHandler, IPointerClickHandler,
                                    IBeginDragHandler {


    public GameObject refObj;
    /// <summary>
    /// referenca na pripadni objekt
    /// </summary>


    //MeshRenderer mr;
```

```
    bool sel = false;   //dali je objekt selektiran?
          public bool selected { get { return sel; } set { sel = value; txt.color = sel ?
             GameData.main.editorSelectedColor : GameData.main.editorDeselectedColor; } }
                                                    //promjena boje prilikom postavljanja

    public bool selecable = true, draggable = true;     //kako se ovim item-om može
                                                         upravljati

       Text txt;   //tekst komponenta item-a
```

Nakon toga slijede četiri metode, prva je Awake koja inicijalizira varijablu txt.

```
    void Awake()
    {
        txt = GetComponent<Text>(); //dobivanje reference teksta ovog objekta
     }
```

Zatim slijede metode koje zahtijevaju sušelja, pozivaju se kada korisnik odbaci neki objekt ovog tipa na ovaj nakon povlačenja (postavlja sve selektirane objekte kao djecu pripadnog objekta ovog item-a), kada klikne na objekt (selektira ga) i kada igrač počinje vući ovaj objekt (selektira ga).

```
    public void OnDrop(PointerEventData eventData)  //metoda se poziva kada korisnik
                                                     dropa neki drugi item na ovaj
    {
        if (draggable || refObj == Scene.rootObject)    //ako se item može draggati ili
//je item za root object, postavi sve selektirane objekte umjesto player-a i kamere kao
//djecu ref Objekta trenutnog item-a
        {
            if (GameEditor.main.selectedObjects.Count > 0)
            {
                for (int i = 0; i < GameEditor.main.selectedObjects.Count; i++)
                {
                    if (!GameEditor.main.selectedObjects[i].GetComponent<Player>() &&
                            GameEditor.main.selectedObjects[i].tag != "EditorCam")
                        GameEditor.main.selectedObjects[i].transform.parent =
                            refObj.transform;
                }
                GameEditor.main.UpdateHierarchy();  //update hijerarhije nakon promjena
            }
        }
    }

    public void OnPointerClick(PointerEventData eventData)  //metoda se poziva kada
//korisnik klikne na trenutni item, ako je item selektibilan onda selektiraj/deselektiraj
//item i njegov objekt
    {
        if (selecable)
        {
            selected = !selected;
            if (selected)
            {
                GameEditor.main.selectObject(refObj);
```

```
            }
            else
            {
                GameEditor.main.deselectObject(refObj);
            }
        }
    }

    public void OnBeginDrag(PointerEventData eventData) //metoda se poziva kada korisnik
                                                          počinje povlačiti ovaj item, ako
                                                        je item "draggabilan" onda ga
                                                        selektiraj
    {
        if (draggable)
        {
            if (!selected)
            {
                selected = true;
                GameEditor.main.selectObject(refObj);
            }
        }
    }
}
```

7. Klasa EditorObjectItem

Objekti koji sadrže ovu klasu su objekti iz izbora objekata editor-a, svaki ima svoju sliku (ikonu) i labelu, klasa također koristi i dva sučelja koja omogućavaju ovim objektima da se povlače i otpuste (drag n drop), dvi obi objekti su generirani od strane editor-a, svi objekti koji sadrže ovu klasu su generirani prilikom učitavanja editora.

Klasa ima samo dvije varijable, jedna koja sadrži objekt koji ona prezentira i jedan je kopija objekta prve varijable koji se stavlja u scenu.

```
public class EditorObjectItem : MonoBehaviour, IDragHandler, IEndDragHandler
{

    public GameObject obj;  //objekt na koji se ovaj item referira

    public GameObject copy = null;  //kopija tog objekta (koji se dodaje u scenu pri
                                      početku drag-a, na drop-u se uključuje)
```

Nakon toga slijede dvije metode, prva se poziva kada se objekt povlači, druga kada se objekt otpusti.

```
    public void OnDrag(PointerEventData eventData)  //kada se item povlači
    {
        bool mouseIsInGameWindow = (Input.mousePosition.x / Screen.width >= 0.25f &&
Input.mousePosition.x / Screen.width <= 0.75f) && (Input.mousePosition.y / Screen.height
>= 0.33f) && !GameEditor.editingSettings;//provjera dali je kursor unutar scene preview-a
        GameEditor.main.deselectAll();  //deselektiraj sve objekte u editoru
```

```
    Collider[] co;                  //svi sudarači kopije referiranog objekta
    if (copy == null) //ako kopija ne postoji, stvori novu, pri tome se
                        pobrini da dobije ime koje se već koristi
    {
        copy = Instantiate(obj);
        if (!GameEditor.main.names.Contains(obj.name))
        {
            copy.name = obj.name;
            GameEditor.main.names.Add(obj.name);
        } else
        {
            copy.name = GameEditor.main.findAnotherName(obj.name);
            GameEditor.main.names.Add(copy.name);
        }
        copy.SetActive(false);
    }
    co = copy.GetComponents<Collider>();    //dodjeli refrece collider-a u co
    if (mouseIsInGameWindow)                //ako je kursor u scene preview-u
    {
        GameEditor.main.X.gameObject.SetActive(false);      //isključi objet X
        copy.SetActive(true);                               //aktiviraj kopiju
        RaycastHit hit;  //zraka koja se šalje od kamere u scenu te ako pogodi nešto,
                           pozicija kopije je malo pomaknuta od te površine
        if (co.Length > 0)                          //isključi sve collider-e kopije
            for (int i = 0; i < co.Length; i++)
                co[i].enabled = false;
        if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out
                 hit, 250f))
        {
            copy.transform.position = hit.point + hit.normal / 2f;
        }
        else
        {
            copy.transform.position = Camera.main.ScreenToWorldPoint(new
          Vector3(Input.mousePosition.x, Input.mousePosition.y, 15f));
        }
        if (co.Length > 0)                                    //ponovo uključi sve
                                                            collider-e kopije
            for (int i = 0; i < co.Length; i++)
                co[i].enabled = true;
    }
    else                                    //ako kursor nije u scene preview-u
    {
        GameEditor.main.X.gameObject.SetActive(true);   //uključi objekt X i postavi
                    ga na poziciju kursora, kopiju isključi (ako već nije isključena)
        GameEditor.main.X.anchoredPosition = Input.mousePosition - (new
          Vector3(Screen.width / 2f, Screen.height / 2f));
        copy.SetActive(false);
    }
}

public void OnEndDrag(PointerEventData eventData)   //kada se item prestane dragg-ati
{
    bool mouseIsInGameWindow = (Input.mousePosition.x / Screen.width >= 0.25f &&
          Input.mousePosition.x / Screen.width <= 0.75f) && (Input.mousePosition.y /
          Screen.height >= 0.33f); //provjera dali je kursor unutar scene preview-a
    GameEditor.main.X.gameObject.SetActive(false);  //isključi X (ako već nije bio
                                                    uključen)
```

```
        if (mouseIsInGameWindow)    //ako je kursor u scene preview-u
        {
            copy.transform.parent = Scene.rootObject.transform; //postavi root kao parent
                                                                novog objekta u sceni kopije)
            var co = copy.GetComponent<Collider>();             //uključi collider kopije
            if (co)
                co.enabled = true;
            copy = null;                                        //poništi refrencu na
                                                                    kopiju
            GameEditor.main.UpdateHierarchy();                  //dodaj objekt u
                                                                    hijerarhiju
        }
        else        //ako kursore nije u scene preview-u, samo isključi kopiju
        {
            copy.SetActive(false);
        }
    }
}
```

8. Klasa Player

Ovom klasom korisniku se omogućuje da kontrolira igrača, pridružena je objektu igrača kao komponenta. U dijelovima, klasa počinje sa deklaracijom varijabli i strukturom za info o skokovima.

```
public class Player : MonoBehaviour
{

    [System.Serializable]
    public struct Jump  //info o skokovima
    {
        public float height;
        public float time;
        public float gravity;
        public float velocity;

        public Jump(float h, float t)   //v=sqrt(2*a*s), g = 2*s/(t/2)^2
        {
            height = h;
            time = t;
            gravity = 2 * h / ((t / 2) * (t / 2));
            velocity = Mathf.Sqrt(2 * gravity * h);
        }

        public void Recalculate()
        {
```

```
            gravity = 2 * height / ((time / 2) * (time / 2));
            velocity = Mathf.Sqrt(2 * gravity * height);
        }

    }

    CapsuleCollider cs; //collider objekta

    public float hp = 100f; //trenutni health point-i

    public float hp_max = 100f; //max mogući hp-ovi

    public float speed = 2.5f;  //brzina pomicanja

    public int currentJump = 0; //index trenutnog skoka u polju skokova
    public Jump[] jumps;    //polje skokova

    public float gravity = -9.81f;  //gravitacija (ako igrač nije napravio skok)

    public Animator anim;   //animator komponenta za izvođenje animacija

    public bool grounded;   //da li je igrač na tlu?

    public Rigidbody rig;  //Rigidbody componenta objekta

    public Vector3 motion = Vector3.zero;   //vector pomaka u svakom
                                              fizičkom(FixedUpdate) frameu
    Vector3 platformVelocity = Vector3.zero, jumpVelocity = Vector3.zero;   //pribrajaju
                                                 se svakog fizičkog frame-a na motion

    public Vector2 inputVector = Vector2.zero;  //input za naprijed/nazad, lijevo/desno
    public bool jump = false;   //input za skok
    public bool isAlive = true; //da li je igrač živ?

    public float outWhenHit = 2.0f; //faktor za out motion
    public Vector3 outMotion;   //pomak izazvan vanjskim utjecajima (npr. sudar s AI-em)

    public float rayLen = 1.01f; //duljina linije koja se cast-a u provjeri da li je
                                  igrač na tlu

    public SkinnedMeshRenderer mr;  //animirana mesh renderer komponenta igrača (dino)

    public LayerMask groundLayers;  //layeri objekta koji se smatraju tlom prilikom
                                      provjere da li je igrač na tlu

    public static Player main;  //za lakše dohvačanje u drugim skriptama (uvjek postoji
                                 točno jedan igrač u sceni!)

    public static bool playing = false; //da li igrač igra?

       GameObject waterSplashParticle; //objekt koji se pokaže dok igrač padne u vodu
```

Onda slijede metode za inicijaliziranje Awake (poziva se inicijalizacijom skripta) i Start (poziva se prvog frame-a igre).

```
    void Awake() //inicijalizacija prije prvog framea
```

```
{
    main = this;
}

void Start()    //inicijalizacija pri prvom frame-u
{
    rig = GetComponent<Rigidbody>();
    cs = GetComponent<CapsuleCollider>();
    for (int i = 0; i < jumps.Length; i++)  //za svaki slučaj potrebno je
   rekalkulirati svaki skok (promjenom visine i trajanja skoka u editoru gravitacija
    i početna brzina se ne mijenjaju!)
        jumps[i].Recalculate();
    waterSplashParticle = Instantiate(GameData.ParticleEffects[2]); //kreiraj efekt
                              "pljuska" vode, isključi ga, dodaj mu koponentu za
                              automatsko isključivanje i vrijeme toga postavi u 1 s
    waterSplashParticle.SetActive(false);
    waterSplashParticle.AddComponent<AutoDisable>().time = 1.0f;
   }
```

Zatim slijede dvije korutine, metode koje se mogu prekidati u izvođenju i izvoditi kasnije i za svaku varijabla koja inicira da li se korutina izvađa ili ne.

```
bool si = false;    //dali se izvodi korutina SwitchIdle?
bool ch = false;    //dali se izvodi korutina smoothChangeIdleState?

IEnumerator smoothChangeIdleState()     //u jednoj sekundi prebacuje IdleRand iz
                                      animatora u suprotnu vrijednost (1->0, 0->1)
{
    ch = true;
    float aim = anim.GetFloat("IdleRand") == 0f ? 1f : 0f, vel = 0f, current =
          anim.GetFloat("IdleRand");
    while (current != aim)
    {
        vel += Time.deltaTime;
        current = Mathf.Lerp(aim == 0f ? 1f : 0f, aim, vel);
        anim.SetFloat("IdleRand", current);
        //print("anim: " + aim + " current: " + current);
        yield return new WaitForSeconds(0.04f);
    }
    ch = false;
}

IEnumerator SwitchIdle()    //sve dok je igrač živ, svakih 3-5 sekundi generiraj
                              random 0 ili 1 te ako su različiti od IdleRand-a,
                              promjeni IdleRand u smoothChangeIdleState korutini
{
    si = true;
    while (isAlive)
    {
        if (Mathf.Round(Random.Range(0f, 1f)) != anim.GetFloat("IdleRand") && !ch)
            StartCoroutine(smoothChangeIdleState());
        yield return new WaitForSeconds(Random.Range(3f, 5f));
    }
    si = false;
   }
```

Nakon toga slijede metode Update i FixedUpdate.

```
    void Update()   //poziva se svakog frame-a u igri
    {
        if (isAlive && !si) //ako SwitchIdle nije pokrenut, pokreni ga
            StartCoroutine(SwitchIdle());
        if (playing && Scene.currentGameState != Scene.GameState.editing)
        {
            if (transform.position.y < Scene.minHeight) //ako je igrač ispod min.
                                        dozvoljene y razine level-a, on umire
                Die();
            if (isAlive)
            {
#if UNITY_STANDALONE    //dio koda koji se prevodi samo za PC platforme (Win, Mac i
                        Linux)
                if (!jump)
                    jump = Input.GetKeyDown(KeyCode.Space); //jump je vrijednost
#endif
                float angle = Mathf.Atan2(inputVector.x, inputVector.y) * Mathf.Rad2Deg;
                      //kut koji input vector čini sa y+ osi
                if (anim.GetBool("Walking"))    //ako se igrač pomiče, pomiči i njegovu
                    rotaciju ovisno o načinu rada kamere pomoću Rigidbody komponente
                    switch (PlayerCamera.main.mode)
                    {
                        case PlayerCamera.PlayerCameraMode.ThirdPerson: //Slerp daje
                                                                klaki pomak rotacije
                            rig.MoveRotation(Quaternion.Slerp(transform.rotation,
                            Quaternion.Euler(transform.eulerAngles.x,
                            Mathf.MoveTowardsAngle(transform.eulerAngles.y,
                            Scene.mainCamera.transform.eulerAngles.y + angle, 50),
                            transform.eulerAngles.z), 15 * Time.fixedDeltaTime));
                            break;
                        case PlayerCamera.PlayerCameraMode.PointLookAt:
                            rig.MoveRotation(Quaternion.Slerp(transform.rotation,
                            Quaternion.Euler(transform.eulerAngles.x,
                            Mathf.LerpAngle(transform.eulerAngles.y,
                            Scene.mainCamera.transform.eulerAngles.y + angle, 50 *
                            Time.fixedDeltaTime), transform.eulerAngles.z), 15 *
                            Time.fixedDeltaTime));
                            break;
                        default:
                            rig.MoveRotation(Quaternion.Slerp(transform.rotation,
                            Quaternion.Euler(transform.eulerAngles.x,
                            Mathf.LerpAngle(transform.eulerAngles.y, angle, 50 *
                            Time.deltaTime), transform.eulerAngles.z), 15 *
                            Time.fixedDeltaTime));
                            break;
                    }
            }
        }
    }

    public bool killingAI = false; //da li je igrač unutar trigger-a koji uništava AI
objekte
    public bool jumpPending = false;//jeli skok u izvođenju

    void FixedUpdate()  //fizički frame igre (poziva se svakih 20 ms(ovisno o Fixed
```

```
                                Timestep-u u postavkama vremena))
    {
        if (playing && Scene.currentGameState != Scene.GameState.editing)
        {
            rig.useGravity = false;
            if (rig.IsSleeping())
                rig.WakeUp();   //ako igrač igra, uključi Rigidbody komponentu koja
                                  simulira fizičke događaje
#if UNITY_STANDALONE
            inputVector = (new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical"))).normalized;   //input vektor ovisi o vrijednostima za
              horizontalne i vetikalne osi i treba biti normaliziran (magnituda = 1)
#endif
            float mag = inputVector.magnitude;  //ili je 1 (strelice ili jedna od wsad
                            tipki je pritisnuta) ili 0 (ništa nije pritisnuto)
            //float angle = Mathf.Atan2(inputVector.x, inputVector.y) * Mathf.Rad2Deg;
              //kut koji input vector čini sa y+ osi
            Physics.queriesHitTriggers = false; //privremeno isključi pogađanje trigger
                     collidera sa zrakama
            grounded = Physics.SphereCast(new Ray(transform.position, Vector3.down),
              0.4f, rayLen, groundLayers); //provjeri dali je igrač na tlu "šaljući
                                             kuglu" prema dolje
            Physics.queriesHitTriggers = true;
            anim.SetBool("Grounded", grounded); //informiraj animatora da je igrač na tlu
            motion = transform.TransformDirection(Vector3.forward * mag * speed) +
              Vector3.up * rig.velocity.y + platformVelocity + (outMotion * outWhenHit);
        //pomak je relatvan s obzirm na roataviju igrača, ovisan o brzini na y osi (za
        skokove i gravitaciju) i vanjske pomake (od platformi i sudara sa AI objektima)
            platformVelocity = Vector3.zero;    //resertiraj platform velocity na nul
                                                  vektor
            anim.SetBool("Walking", mag > 0.01f ? true : false);    //ako je magnituda
                            input vekora veća od nule, animator smatra se da igrać hoda
            if (grounded && !killingAI && !jumpPending) //ako je igrač na tlu i nije u
                                                  sudaru sa triggerom AI objekta
                                                  resertiraj njegove skokove
                                                  (omogućava novu seriju skokova)
                currentJump = 0;
            if (jump && !jumpPending)   //ako je zatražen skok i trenutni se ne obavlja
            {
                if (currentJump + 1 <= jumps.Length)
                {   //skoči ako je moguće
                    anim.SetTrigger("Jump"); //informiraj animator o skoku, event u
                                                animaciji će pokrenuti DoJump() Metodu
                    jumpPending = true; //priprema za skok (JumpPrepare klip) počinje

                }
                jump = false;   //resertiraj jump
                //print("jumpy " + currentJump);    //korištenu u debug-u
            }
            if (jump && jumpPending && currentJump == jumps.Length) //svaki novi skok
                            može se izvesti nakon kratkog vremena, ako trenutni skok nije
                            zadnji, novi se može odraditi u kratkom vremenu
                jump = false;   //resertiraj jump
            if (jumpVelocity.y != 0f)   //ako je skok zadan
            {
                motion.y = jumpVelocity.y;  //postavi brzinu skoka na zatraženu
                jumpVelocity = Vector3.zero;    //resertiraj jumpVelocity na nul vektor
            }
```

```
            rig.velocity = motion;  //postavi trenutnu brzinu gibanja na sve što je prije
                                        bilo uračunato

            if (currentJump == 0)   //gravitacija
            {
                rig.AddForce(Vector3.down * -gravity);  //"normalna" tj. zadana
                                                            gravitacija
            }
            else
            {
                rig.AddForce(Vector3.down * +jumps[currentJump - 1].gravity);
//gravitacija skoka (potrebno kako bi skok trajao onoliko koliko je zadan)
            }
        }
        else    //ako igrač ne igra, isključi Rigidbody komponentu
        {
            if (!rig.IsSleeping())
                rig.Sleep();
        }
        if (Scene.currentGameState == Scene.GameState.playing && !isAlive)  //ako je
                                                igrač umro, vuci ga prema dolje
            rig.AddForce(Vector3.up * -10);
        //print(grounded + " v " + rig.velocity + " m " + motion);  //korišteno za
              debug...
        //print(currentJump);
      }
```

Zatim slijedi metoda koja se poziva u određenom trenuku animacije pripreme za skok, te korutina koja nakon nekog vremena igraču omogučuje drugi skok.

```
    public void DoJump()    //metoda za izvođenje skokova
    {
        if (currentJump + 1 <= jumps.Length)    //ako je skok moguć
        {
            jumpVelocity.y = jumps[currentJump].velocity;   //postavi jump velocity na
                                                              brzinu trenutnog skoka
            currentJump++;  //postavi index trenutnog skoka na index sljedećeg
            grounded = false;   //smatra se da igrač nemože biti na tlu ako je upravo
                                  skočio
            StartCoroutine(AllowAnotherJump()); //za mogučnost drugog skoka treba prijeći
                                                  određeno vrijeme
            //anim.SetTrigger("Jump");
            //print("jump: " + currentJump);
        }

    }

    bool aaj = false;   //dali je korutina AllowAnotherJump u izvođenju

    IEnumerator AllowAnotherJump()  //nakon određenog vremena (0.21 s) omogućuje drugi
                                        skok
    {
        if (!aaj)
        {
            aaj = true;
            yield return new WaitForSeconds(0.21f);
```

```
            jumpPending = false;
            aaj = false;
            yield break;
        }
    }
```

Zatim slijede metode koje se pozivaju pri sudarima igrača s drugim objektima

```
void OnTriggerEnter(Collider col)   //poziva se kada se igrač sudari sa collider-om
                                      koji je trigger
{
    if (playing)
    {
        if (Scene.currentGameState != Scene.GameState.editing)
        {
            if (col.tag == "AI")    //ako je objekt s kojim se igrač sudario tipa AI
            {
                AI ai = col.GetComponent<AI>(); //referenca za lakše i brže baratanje
                                                  komponentom
                if (transform.position.y - cs.height / 2f > col.transform.position.y
                 + col.bounds.size.y / 2f && ai.alive)
                        //ako je igrač iznad AI-jevog trigger collider-a i
                                    AI je je živ
                {
                    ai.gainHp(-1);  //ai dobiva štetu (umire jer mu je hp 1)
                    //if (currentJump + 1 > jumps.Length)    //resertiraj skokove ako
                        je potrebno
                    //    currentJump = 0;
                    anim.CrossFade("JumpPrepare", 0.18f, 0);   //napravi jedan skok
                                                                 od AI objekta
                    killingAI = true;   //uloga ove varijable je uglavnom ta da se u
                                        FixedUpdate-u currentJump opet ne resetira na
                                        nulu jer kad igrač skoči ili padne na AI raycast
                                        gotovo uvijek detektira da je on došao na tlo
                    jumpPending = false;    //poništi trenutni skok (ako uopče ima
                                              skoka)
                                            //print("ai hit " + currentJump);
                                            //korišteno prilikom debug-a
                }
            }
            if(col.gameObject.layer == 4)   //ako je objekt voda
            {
                if(rig.velocity.y < -0.5)
                {
                    waterSplashParticle.transform.position = new Vector3
                        (transform.position.x, col.transform.position.y + 0.1f,
                        transform.position.z);
                          //postavi splash objekt na xz poziciju igrača i malo više od
                               vode
                    waterSplashParticle.SetActive(true);
                }
            }
            if(col.gameObject.layer == 11)  //ako je objekt municija AI-a
            {
                GainHp(-1);
                Ammo amm = col.gameObject.GetComponent<Ammo>(); //isključi taj ammo
                amm.StopAllCoroutines();
```

```
                ParticleSystem ps = amm.GetComponent<ParticleSystem>();
                if (ps)
                    ps.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
                for (int i = 0; i < amm.transform.childCount; i++)
                {
                    ps = amm.transform.GetChild(i).GetComponent<ParticleSystem>();
                    if (ps)
                        ps.Stop(true,
                 ParticleSystemStopBehavior.StopEmittingAndClear);
                }
                if (amm.onCollisionParticles)
                {
                    amm.onCollisionParticles.SetActive(true);
                                amm.onCollisionParticles.transform.position =
                       (amm.transform.position + transform.position) / 2f;
                    amm.onCollisionParticles.GetComponent<ParticleSystem>().Play();
                }
                amm.gameObject.SetActive(false);
            }
        }
    }
}

void OnTriggerExit(Collider col)    //poziva se kada je igrač izašao iz sudara sa
                                     trigger collider-om
{
    if (Scene.currentGameState != Scene.GameState.editing && playing)
    {
        if (col.tag == "AI")
        {
            //print("ai out " + currentJump);
            killingAI = false;  //dopusti da grounded ponovo može resertirati serije
                                    skokova
        }
    }
}

void OnCollisionStay(Collision col) //poziva se sve dok je igrač u kontaktu sa nekim
                                     collider-om
{
    if (Scene.currentGameState != Scene.GameState.editing)
    {
        if (playing)
        {
            if (col.collider.tag == "MovingPlatform")   //ako igrač igra i ako je na
                 pomičnoj platformi postavi njeguvu brzinu na brzinu platforme tako
                              da se pomiče s platformom
            {
                platformVelocity = col.collider.GetComponent<Rigidbody>().velocity;
            }
        }
    }
  }
```

I nakon toga Metode koje se pozivaju pobjedom, smrti ili mijenjanjem hp-a (health points) igrača te korutina koja povećava/smanjuje brzinu pomicanja igrača na određeno vrijeme, u igri se ne koristi.

```
    public void Die()   //poziva se kada igrač umire
     {
         isAlive = false;
         rig.useGravity = true;
         Scene.main.playerDied();    //informiraj scenu o smrti igrača
         anim.SetBool("Dead", true);     //informiraj animator-a o smrti
#if UNITY_EDITOR
         print("death"); //ispisuje se samo u editoru
#endif
     }

     public void Win()   //poziva se kada igrač pobijedi (sudari se sa WinTriggerom)
     {
#if UNITY_EDITOR
         print("won");   //ispisuje se samo u editoru
#endif
         playing = false;
         Scene.main.playerWon(); //informiraj scenu o tome da je level gotov
         anim.SetBool("Jump", false); //na kraju se izvodi animacija idle
         anim.SetBool("Walking", false);
         anim.SetBool("Idle", true);
         anim.CrossFade("Idles", 0.4f);
     }

     public void GainHp(float gain)  //promjena hp-a igrača ( < 0 daje damage (štetu), > 0
                                          daje liječenje (healing))
     {
         hp = Mathf.Clamp(hp + gain, 0f, hp_max);    //postavi hp u interval od 0 do
                                                        hp_max
         if (hp <= 0f)
             Die();  //ako je hp <= 0 igrač je izgubio...
     }

     public IEnumerator GainSpeed(float boost, float seconds)    //povećava brzinu za
        određen boost na određeno vrijeme (za speed powerup-ove), u igri se ne koristi
     {
         speed += boost;
         yield return new WaitForSeconds(seconds);
         speed -= boost;
         yield break;
     }

    }
```

9. Klasa PlayerCamera

Ova klasa je pridružena objektu kamere u sceni, kamera je ujedno i jednini izvor zvuka u igri, pa manjim dijelom i ovaj skript time upravlja. Kamera ima 4 načina rada, Third Person (kamera se može okretati oko igrača), PointLookAt (kamera se miče

po danim točkama te gleda u igrača), Editor (ponašanje kamera u editoru) i Stand (samo stoji). Kod je slijedeći:

```
public class PlayerCamera : MonoBehaviour {

    //U kojem načinu kamera radi
    public enum PlayerCameraMode
    {
        ThirdPerson,
        PointLookAt,
        Stand,
        Editor
    }

    public static PlayerCamera main;    //za lakše dobivanje komponente u drugim
                                          skriptama

    public PlayerCameraMode mode = PlayerCameraMode.ThirdPerson;    //trenutni način rada

    public float minDistance = 5f;  //min i max udaljenosti od igrača
    public float maxDistance = 15f;

    public float smoothFactorPosition = 5f; //koliko glatko će se pozicija kamere
                                             mijenjati u ThirdPerson načinu rada

    public float distance = 10f;    //trenutna udaljenost kamere od igrača

    public float yAngleMinTP = 0f, yAngleMaxTP = 80f, yAngleMinFP = 0f, yAngleMaxFP =
                    175f;
                     //min i max kutevi na y rotacijskoj osi za Third Person i druge
                           načine rada

    public float xRotationFactor = 2f;  //brzina mjenjanja rotacije na x rotacijskoj osi

    //public float zoomFactor = 2.333333f;

    public Vector3 lookAtPointPositionOffset;   //dodatni offset koji se pribraja
                                  igračevoj poziciji kada kamera gleda u njega

    public Vector3[] points;    //point-i PointLookAt mode-a

       public Vector3 pointOffset; //dodatni offset koji se pribraja poziciji point-a
                                  prilikom traženja najbližeg

    public int currentPoint = 0;    //trenutni point na kojemu je kamera

    Transform player;   //Transform komponenta player-a

    public float mx, my;    //mouse x i mouse y, određuju pomak u ThirdPerson načinu rada

    public float switchSpeed = 0.85f;   //brzina mjenjanja point-a u PointLookAt mode-u

    //float w = 0f;

    public float editorSpeed = 3f;  //brzina pomicanja kamere u editoru
```

```
    public float editorRotationSpeed = 2f;  //brzina rotiranja u editoru

    public float editorScrollSpeed = 10f;   //brzina zumiranja u editoru

    public AudioClip menuMusic, gameMusic;  //zvučni klipovi za muziku u igri

    float rotationX = 0, rotationY = 0; //varijable koje pamte rotaciju

    public AudioSource a7;     //za mijenjanje zvuka

    public UnityEngine.UI.Slider volumeSlider;  //Slider za mijenjanje jačine zvuka

    float d = 0;    //udaljenost kamere i igrača

    public void SetVolume()  //postaljanje jačine zvuka (zvano od UI slider-a)
    {
        a7.volume = volumeSlider.value;
    }

    void Awake()    //inicijalizacija...
    {
        main = this;
        a7 = GetComponent<AudioSource>();
        a7.clip = menuMusic;
        a7.loop = true;
        a7.Play();
        if (Scene.player != null)
            player = Scene.player.GetComponent<Transform>();
        d = -main.distance;
    }

    void Update()   //svakog frame-a dobavljaj vrijednosti za vrijable koje služe kao
input (mx i my)
    {
        mx += Input.GetAxis("Mouse X");
        /*if(!Scene.player.grounded)
            w = 0f;
        w += Input.GetAxis("Vertical");*/
        if (mode == PlayerCameraMode.ThirdPerson)
            my = Mathf.Clamp((my - Input.GetAxis("Mouse Y")), yAngleMinTP, yAngleMaxTP);
        else
            my = Mathf.Clamp((my - Input.GetAxis("Mouse Y")), yAngleMinFP, yAngleMaxFP);
    }

       void LateUpdate () {    //ova metoda poziva se nakon Update metode svakog frame-a,
        if (player == null)
            player = Scene.player.transform;
        switch (mode)   //svaki mode radi na zaseban način
        {
            case PlayerCameraMode.ThirdPerson:  //igrač može kameru pomicati mišem oko
                                                    njega
                d = -distance;
                RaycastHit h;
                Physics.queriesHitTriggers = !Physics.queriesHitTriggers;
                if (Physics.Linecast(player.position, transform.position, out h))
                {
                    d = -Vector3.Distance(player.position, h.point);
                    if (d > -1f) d = -1f;
```

```
                }
                Physics.queriesHitTriggers = !Physics.queriesHitTriggers;

                Vector3 dir = new Vector3(0, 0, d);

                Quaternion rot = Quaternion.Euler(my, mx * xRotationFactor, 0f);

                transform.position = Vector3.Lerp(transform.position, (player.position +
              lookAtPointPositionOffset) + rot * dir, smoothFactorPosition *
              Time.deltaTime);

                transform.LookAt(player.position + lookAtPointPositionOffset);

                break;
            case PlayerCameraMode.PointLookAt:  //kamera se pommiče po pointima najbližim
                                                    igraču i gleda u njega
                if (points.Length < currentPoint)
                    currentPoint = 0;
                    float cd = Vector3.Distance(points[currentPoint] + pointOffset,
                     player.position);

                    for (int i = 0; i < points.Length; i++)
                    {
                        if (i != currentPoint && Vector3.Distance(points[i] +
                            pointOffset, player.position) < cd)
                            currentPoint = i;
                    }

                    /*if (currentPoint + 1 < points.Length && cd >
Vector3.Distance(points[currentPoint + 1], player.position))
                        currentPoint++;

                    if (currentPoint - 1 >= 0 && cd >
Vector3.Distance(points[currentPoint - 1], player.position))
                        currentPoint--;*/

                    transform.position = Vector3.Lerp(transform.position,
                     points[currentPoint], switchSpeed * Time.deltaTime);

                    transform.LookAt(player);

                break;
            case PlayerCameraMode.Stand:    //stoji...
                //nothing
                break;
            case PlayerCameraMode.Editor:   //detaljna kontrola nad kamerom, ako je
kursor unutar preview prozora kamera se može pomicati vertikalno i horizontalno, hover-
ati, rotirati dok je pritisnuta desna tipka miša i pomicati naprijed/nazad kotačićem
                //print(Input.mousePosition);
                bool mouseIsInGameWindow = (Input.mousePosition.x / Screen.width >= 0.25f
                     && Input.mousePosition.x / Screen.width <= 0.75f) &&
                     (Input.mousePosition.y / Screen.height >= 0.33f) &&
                     !GameEditor.editingSettings;
                if (mouseIsInGameWindow)
                {
                    if (Input.GetKey(KeyCode.Mouse2))   //hover
                    {
                        transform.position += transform.TransformDirection(new Vector3( -
```

```
                    Input.GetAxis("Mouse X"), -Input.GetAxis("Mouse Y"), 0)) *
                    editorSpeed * Time.deltaTime;
                }
                else    //obični pomak strelicama
                {
                    Quaternion r = transform.rotation;
                    transform.eulerAngles = new Vector3(0f, transform.eulerAngles.y,
                        transform.eulerAngles.z);
                    transform.Translate((new Vector3(Input.GetAxis("Horizontal"), 0,
                        Input.GetAxis("Vertical"))) * editorSpeed * Time.deltaTime);
                    transform.rotation = r;
                }
                            if (Input.GetKey (KeyCode.Mouse1)) {    //rotiraj ako
                                                    je pritisnuta desna tipka miša
                    //Vector3 r = transform.eulerAngles + new Vector3 (Input.GetAxis
                        ("Mouse Y"), -Input.GetAxis ("Mouse X")) * editorRotationSpeed
                        * Time.deltaTime;
                    //transform.rotation = Quaternion.Euler(r);

                    rotationX += Input.GetAxis("Mouse X") * editorRotationSpeed *
                        Time.deltaTime;
                    rotationY += Input.GetAxis("Mouse Y") * editorRotationSpeed *
                        Time.deltaTime;
                    Quaternion xQuaternion = Quaternion.AngleAxis(rotationX,
                        Vector3.up);
                    Quaternion yQuaternion = Quaternion.AngleAxis(rotationY, -
                        Vector3.right);
                    transform.localRotation = Quaternion.identity * xQuaternion *
                        yQuaternion;
                    //transform.RotateAround(transform.TransformPoint(0,0,5), new
                        Vector3(Input.GetAxis("Mouse Y") *
                        Time.deltaTime,Input.GetAxis("Mouse X") * Time.deltaTime,0),
                        editorRotationSpeed);
                }
                    transform.Translate(0, 0, Input.mouseScrollDelta.y *
                    editorScrollSpeed);    //scoll (pomak naprijed nazad)
            }
            break;
        default:
            break;
    }
}

public void UpdateEditorRotation() {    //za vraćanje rotacije u editoru
    Quaternion xQuaternion = Quaternion.AngleAxis(rotationX, Vector3.up);
    Quaternion yQuaternion = Quaternion.AngleAxis(rotationY, -Vector3.right);
    transform.localRotation = Quaternion.identity * xQuaternion * yQuaternion;
}

//za prikaz point-ova u editoru
void OnDrawGizmos()
{
#if UNITY_EDITOR
    Gizmos.color = Color.red;
    if(mode == PlayerCameraMode.PointLookAt)
    {
        for(int i = 0; i < points.Length-1; i++)
        {
```

```
                Gizmos.DrawLine(points[i], points[i + 1]);
            }
        }
#endif
    }
}
```

10. Klasa AI

AI (Artificial Intelligence) objekti su objekti koji igraču otežavaju rješavanje level-a jer ga napadaju i rade štetu, no igrač može uništiti AI objekt bez većih problema samim skokom na AI objekt, AI objekti u igri su AnkPatroller, AnkAttacker, ShooterPlant i SniperPlant. Po tipu AI može patrolirati, napadati igrača kada se previše približi i pucati municiju na igrača ako je previše blizu. Kod je slijedeći:

```
[RequireComponent(typeof(Rigidbody))]   //koristi Rigidbody komponentu za pokrete
public class AI : MonoBehaviour {

    public float hp = 100f,         //trenutni hp
                 hp_max = 100f;     //max mogući hp

    public enum AIType { Patrol, Attack, Shooter }  //tipovi

    public AIType type;     //tip AI-a

    public float speed;                 //brzina pomicanja
    public bool rotateStop = true;      //hoće li se Ai rotitati u smjeru sljedećeg
                                         patrol pointa (ako je tipa Patrol) ili se može
                                         rotirati (ako je tipa Attack ili Shooter)
    public float rotSpeed = 1.5f;       //faktor rotiranja

    public int currentPoint = 0;        //trenutni patrol point
    public Vector3[] patrolPoints;      //patrol point-i relativni s obzirom na početnu
                                         poziciju objekta u početku levela
    public float toleratedDistance;     //udaljenost od trenutnog point-a za koju se
                                         smatra da je AI došao do point-a

    public float activeDistance;        //ako je igrač bliži AI-u od te udaljenosti i ako
                                  je AI tipa Attack ili Shooter, napad na igrača počinje

    Rigidbody rig;  //referenca na Rigidbody komponentu objekta

    Vector3 relativePos;    //pozicija objekta pri početku levela

    public bool alive = true;   //da li je AI živ?

    public float onGroundRaycastLenght = 0.501f;    //duljina zrake koja provjerava da li
                                                       je AI na tlu

    public bool checksBeforeItGoes = true;          //dali je AI "pametan", tj provjerava
                                                       što ga čeka na putu
```

```
public float forwardRacastCheckLength, downRaycastCheckLength;  //zrake koje
                                                "pametan" AI koristi da provjeri put

public GameObject ammo; //za Shooter AI-ove, objekt (municija) koji AI baca prema
                                igraču

public float ammoLifeTime = 3f; //koliko dugo je objekt ammo uključen

GameObject[] ammos;                 //svi ammo-i ovog AI-a
int ca = 0;                         //index trenutnog ammo objekta u polju ammos
bool shooting = false;              //dali AI puca?

public float rateOfFire;            //koliko ammo objekata ovaj AI može ispaliti u
                                      sekundi

public float relaxTime = 1f;        //dodatno vrijeme čekanja nakon što AI ispali
                                      metak

public Vector3 shootPointOffset;    //pozicija s obzirom na poziciju i rotaciju AI-a
                                      iz koje se ispaljuje ammo

public float shootHeight = 7f;      //max visina koju ammo dostiže pri putu do
                                      player-a

public float damageOnCollision;     //damage koji player dobije kada se sudari s ovim
                                      objektom

public Animator anim;              //za animaicije..
                                    /// <summary>
                                    /// idle
                                    /// walk
                                    /// shoot
                                    /// death
                                    /// turn        //180°
                                    /// nothing
                                    /// </summary>


public Vector3 deathScaleFactor = Vector3.one;    //vektor s kojim se scale collider-
                                                    a množi dok AI umre

//public float firingStartDelay = 0f;               //vrijeme u sec prije nego što
                                                          Shooter AI krene pucati

void playAnim(string clipName)  //"robusna" metoda za pokretanje animacija
{
    if(anim)
    {
        bool contains = false;
        for (int i = 0; i < anim.parameterCount; i++)   //provjerava postoji li
          parametar u animatoru, ako postoji i ako je tipa bool, postavlja ga u true
            if (anim.parameters[i].name == clipName)
            {
                contains = true;
                break;
            }
        if(contains)
        {
```

```
                for (int i = 0; i < anim.parameterCount; i++)
                    if (anim.parameters[i].name == clipName)
                        anim.SetBool(clipName, true);
                    else if (anim.parameters[i].type ==
                            AnimatorControllerParameterType.Bool)
                        anim.SetBool(anim.parameters[i].name, false);
            }
        }
    }

    void Start() {                          //inicijalizacija
        rig = GetComponent<Rigidbody>();
        relativePos = rig.position;
        if(ammo)
        {
            ammos = new GameObject[(((int)rateOfFire) * ((int)ammoLifeTime)) + 1];
//ukupni broj ammo-a = KolikoIhMožeIspalitiUSekundi * KolikoAmmoTraje + 1,

//više ammo-a za Shooter AI-a od te brojke nije potrebno, ammo-i se ne brišu i ponovo
       stvaraju,

//nego se samo isključiju i ponovo uključuju radi boljih performansi (object pooling)
            for (int i = 0; i < ammos.Length; i++)  //svakoj referenci u polju ammos-a
 dodaj novi novi ammo i inicijaliziraj ga
            {
                ammos[i] = Instantiate(ammo);
                ammos[i].transform.parent = Scene.rootObject.transform;
                if (!ammos[ca].GetComponent<Ammo>())
                    ammos[ca].AddComponent<Ammo>();
                ammos[i].GetComponent<Ammo>().lifeTime = ammoLifeTime;
                ammos[i].SetActive(false);
            }
        }
        playAnim("nothing");
    }

    void FixedUpdate()      //fizički frame
    {
        if (alive && Scene.player.isAlive && Scene.currentGameState ==
              Scene.GameState.playing) //preduvjeti za rad
        {
            if (transform.position.y < Scene.minHeight) //baš kao i igrač, AI umire ako
                                                        je ispod min. dozvoljene razine
                Die();
            switch (type)
            {
                case AIType.Patrol:         //ako je AI tipa Patrol
                    Vector3 direction = (-(rig.position - (relativePos +
                      patrolPoints[currentPoint]))).normalized;      //smjer prema
                                                                  sljedećem point-u
                    direction.y = 0;
//smjer treba biti po XZ plohi
                    direction.Normalize();
                    //Debug.DrawLine(transform.position, transform.position +
direction*5, Color.green);
                    rig.velocity = (direction * speed * Time.fixedDeltaTime);
//postavi brzinu da pomiče AI preme sljedećem point-u
                    playAnim("walk");
```

```
            if (Vector3.Distance(rig.position, patrolPoints[currentPoint] +
             relativePos) <= toleratedDistance)  //ako je AI stigao na point
            {
                if (rotateStop)     //ako je rotateStop true, rotiraj ga prema
                                        sljedećm point-u
                {
                    Quaternion dir = Quaternion.LookRotation(-direction);
                    Quaternion rot = Quaternion.RotateTowards(rig.rotation, dir,
                     rotSpeed * Time.fixedDeltaTime);   //rotiraj trenutnu
                                                       rotaciju prema potrebnoj
                    rig.MoveRotation(rot);
                    playAnim("turn");
                    if (Quaternion.Dot(rig.rotation, dir) > 0.99998f)   //ako su
                                                       rotacije gotovo jednake
                    {
                        currentPoint = (currentPoint + 1) % patrolPoints.Length;
                                                    //idi na sljedeći point
                    }
                }
                else
                {
                    currentPoint = (currentPoint + 1) % patrolPoints.Length;
                                                    //idi na sljedeći point
                }
                rig.velocity = Vector3.zero;    //zaustavi objekt na neko vrijeme
            }
            break;
        case AIType.Attack:         //ako je AI tipa Attack
            bool onGround = Physics.Raycast(rig.position, Vector3.down,
                          onGroundRaycastLenght);         //provjeravaj dali je
                                                            AI na tlu
            playAnim("idle");
            //Debug.DrawRay(transform.position, -(transform.position –
                   Scene.player.transform.position), Color.red);
            if(Vector3.Distance(rig.position, Scene.player.transform.position) <=
                   activeDistance &&  onGround)  //ako je AI na tlu i igrač mu je
                                                       pre blizu
            {
                Vector3 directionToPlayer = (Scene.player.transform.position –
                   transform.position);  //smjer prema igraču
                directionToPlayer.y = 0f;
                directionToPlayer.Normalize();  //normaliziraj (postavi duljunu
                                                na 1) po XZ plohi
                //Debug.DrawLine(transform.position, transform.position +
                   directionToPlayer*5, Color.green);
                playAnim("walk");
                if (rotateStop)     //ako se koristi rotacija, rotiraj AI prema
                                        igraču
                {
                    rig.MoveRotation(Quaternion.RotateTowards(rig.rotation,
                    Quaternion.LookRotation(directionToPlayer), rotSpeed *
                    Time.fixedDeltaTime));
                }

                  if (checksBeforeItGoes) //ako je AI "pametan"

                {
```

```
                        //Debug.DrawLine(rig.position, rig.position +
directionToPlayer * forwardRacastCheckLength, Color.magenta);
                        //Debug.DrawLine(rig.position + directionToPlayer *
forwardRacastCheckLength, (rig.position + directionToPlayer * forwardRacastCheckLength) +
Vector3.down * downRaycastCheckLength, Color.red);
                        //pošalji jednu zraku naprijed, drugu zraku od kraja te zrake
prema dolje, ako nijedna detektira ništa, AI se slobodno može kretati, u suprotnom AI
samo stoji
                        if (!Physics.Raycast(rig.position, Vector3.forward,
      forwardRacastCheckLength, GameData.main.noPlayer) && Physics.Raycast(rig.position
      + directionToPlayer * forwardRacastCheckLength, Vector3.down,
      downRaycastCheckLength, GameData.main.noPlayer))
                        {
                            rig.velocity = directionToPlayer * speed *
                              Time.fixedDeltaTime;
                        } else
                        {
                            rig.velocity = Vector3.zero;
                        }
                    } else
                    {
                        rig.velocity = directionToPlayer * speed *
                              Time.fixedDeltaTime;
                    }
                } else
                {
                    playAnim("idle");
                }
                break;
            case AIType.Shooter:    //ako je AI tipa Shooter
                if (!shooting)
                    playAnim("idle");
                if (Vector3.Distance(rig.position, Scene.player.transform.position)
                        <= activeDistance) {    //ako je igrač preblizu
                    Vector3 directionToPlayer = (Scene.player.transform.position -
                        rig.position);
                    directionToPlayer.y = 0f;
                    directionToPlayer.Normalize();
                    //print(directionToPlayer + " " + rig.rotation);
                    if (rotateStop)     //ako se AI može rotirati, rotiraj ga prema
                                            player-u
                    {
                        //print("h");
                        transform.rotation = (Quaternion.RotateTowards(rig.rotation,
                        Quaternion.LookRotation(directionToPlayer), rotSpeed *
                        Time.fixedDeltaTime));
                    }
                    if(ammo && Quaternion.Dot(rig.rotation,
                        Quaternion.LookRotation(directionToPlayer)) > 0.99998f)
                         //ako ima municije i ako je AI
                        usmjeren prema igraču, počni pucati (ako već nije započeto)
                    {

                        if(!shooting)
                            StartCoroutine(shoot());
                    }
                }
                break;
```

```
                default:
                    break;
            }
        } else  //ako preduvijeti za rad nisu ispunjeni, AI ne radi ništa, jedino u
                    meniju izvodi animaciju idle
        {
            rig.velocity = Vector3.zero;
            rig.angularVelocity = Vector3.zero;
            if (Scene.currentGameState == Scene.GameState.editing)
                playAnim("nothing");
            if (Scene.currentGameState == Scene.GameState.menu)
                playAnim("idle");
        }
    }

    IEnumerator shoot()     //za Shooter AI-ove, korutina ispaljuje metke na igrača
    {
        shooting = true;
        playAnim("shoot");
        //print("shooting");
        //if (firingStartDelay > 0.00001f)
        //    yield return new WaitForSeconds(firingStartDelay);
        while(alive && Vector3.Distance(rig.position, Scene.player.transform.position) <=
              activeDistance && Player.main.isAlive)   //sve do je AI živ i igrač je u
                                                        kritičnoj blizini
        {
            if (!anim)  //animirani AI puca po eventu iz animacije
            {
                shootAmmo();
                yield return new WaitForSeconds(1f / rateOfFire);
                ca = (ca + 1) % ammos.Length;   //pripremi sljedeći ammo za paljbu
                yield return new WaitForSeconds(relaxTime);
            } else
            {
                playAnim("shoot");
                yield return new WaitForFixedUpdate();
            }
        }
        shooting = false;
    }

    public void shootAmmo()
    {

        ammos[ca].SetActive(true);
//aktiviraj trenutni ammo
        ammos[ca].transform.position = transform.position +
              transform.TransformDirection(shootPointOffset);         //postavi ammo na
       početno mjesto uz offset

//Ispali ammo, brzina se računa formulom:

//Vxz   = Dx / (sqrt(-2*h/g) + sqrt(2*(Dy-h)/g)) za XZ plohu

//Vy    = sqrt(-2*g*h) za Y os

//Dx    - udaljenost igrača i ovog AI-a po XZ plohi
```

```
//h     - shootHeight

//g     - gravitacija po y osi

//Dy    - udaljenost igrača i ovog AI-a po y osi

        ammos[ca].GetComponent<Rigidbody>().velocity =
            transform.TransformDirection(Vector3.forward * (Vector3.Distance(new
       Vector3(transform.position.x, 0f, transform.position.z), new
       Vector3(Scene.player.transform.position.x + Player.main.rig.velocity.x, 0f,
       Scene.player.transform.position.z + Player.main.rig.velocity.z)) / (Mathf.Sqrt((-
       2f * shootHeight) / Physics.gravity.y) + Mathf.Sqrt((2f * ((-
       Mathf.Abs(Scene.player.transform.position.y - transform.position.y)) -
       shootHeight)) / Physics.gravity.y)))
            + Vector3.up * Mathf.Sqrt(-2f * Physics.gravity.y * shootHeight));
        ammos[ca].GetComponent<Ammo>().StartCoroutine("end");   //ammo se isključuje
                                                                    nakon nekog vremena
        if (anim)
            ca = (ca + 1) % ammos.Length;
    }

    public void OnCollisionEnter(Collision col) //ako se igrač sudai s ovim AI-om, daj mu
                                                 damage i odbaci ga u suprotom smjeru
    {
        if (alive)
        {
            if (col.collider.tag == "Player")
            {
                Scene.player.GainHp(damageOnCollision);
                Scene.player.outMotion = -(transform.position -
                    Scene.player.transform.position).normalized *
                    Scene.player.outWhenHit;
                //print("jump out");
            }
        }
    }

    public void OnCollisionExit()   //kada se igrač prestane sudarati s ovim objektom,
                                        poništi sve vanjske pokrete
    {
        //if (alive)
            Player.main.outMotion = Vector3.zero;
    }

    public void gainHp(float gain)  //za dodavanje/uklanjanje hp-as
    {
        hp = Mathf.Clamp(hp + gain, 0f, hp_max);
        if (hp <= 0f)
            Die();
    }

    public void Die()   //poziva se kada ai umre (hp <= 0)
    {
        alive = false;
        rig.constraints = RigidbodyConstraints.FreezeAll;
        playAnim("death");
        BoxCollider[] bcs = GetComponents<BoxCollider>();
```

```
        if (bcs.Length > 1) //deathScaleFactor radi samo na box collider-ima
        {
            bcs[0].size = new Vector3(bcs[0].size.x * deathScaleFactor.x, bcs[0].size.y *
              deathScaleFactor.y, bcs[0].size.z * deathScaleFactor.z);
            bcs[0].center = bcs[0].center - absVec3(Vector3.one - deathScaleFactor);
        }
        if(bcs.Length > 2)
        {
            bcs[2].enabled = false; //isključi gornji trigger
        }
        rig.isKinematic = true;     //za poništavanje fizike objekta
    }

    Vector3 absVec3(Vector3 v)  //abs svakog dijela vektora
    {
        return new Vector3(Mathf.Abs(v.x), Mathf.Abs(v.y), Mathf.Abs(v.z));
    }

  }
```

11. Klasa Ammo

Klasa koji imaju svi objekti koje ispaljuje AI tipa Shooter, ukratko, kada se metak ispali, nakon nekog vremena on se sam isključi, ili ako se sudari s određenim preprekama, on se odmah isključi. Kod je slijedeći:

```
[RequireComponent(typeof(Rigidbody))]   //za rad tahtijeva komponentu rigidbody
public class Ammo : MonoBehaviour {

    public float lifeTime;  //vrijeme života metka, nakon tog vremena ovaj objekt se
                                isključuje

    public GameObject onCollisionParticles; //objekt koji se stvara tokom sudara ovog
                                            metka

    void Start()
    {
        onCollisionParticles = Instantiate(onCollisionParticles);   //stvori kopiji
                                                                    čestica za sudaranje
        onCollisionParticles.transform.parent = Scene.rootObject.transform; //postavi
                                                root kao parent tog efekta sudaranja
        //print("e");
    }

       IEnumerator end()   //korutina koja se izvodi tokom izvođenja ovog objekta
    {
        if (onCollisionParticles)   //ako postoji efekt sudara, isključi ga
        {
            onCollisionParticles.SetActive(false);

        }
        yield return new WaitForSeconds(lifeTime);  //čekaj da vrijeme života metka
                                                      prijeđe
        ParticleSystem ps = GetComponent<ParticleSystem>(); //ako ovaj objekt ima na sebi
              ParticleSystem komponentu, isključi je, isto vrijedi i za djecu ovog objekta
        if (ps)
```

```
            ps.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
        for (int i = 0; i < transform.childCount; i++)
        {
            ps = transform.GetChild(i).GetComponent<ParticleSystem>();
            if (ps)
                ps.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
        }
        gameObject.SetActive(false);
    }

    void OnTriggerEnter(Collider col)   //ako se objekt sudari s nečime (metoda se tada
                                          poziva)
    {
        if ((col.gameObject.layer & Scene.player.groundLayers.value) == 1 &&
              !col.gameObject.GetComponent<AI>())    //ako se sudari s nekim preprekama
        {
            StopAllCoroutines();    //ammo prestaje postojati
            ParticleSystem ps = GetComponent<ParticleSystem>();     //zaustavi sve efekte
                                                                    ovog objekta i njegove djece
            if (ps)
                ps.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
            for(int i = 0;i < transform.childCount; i++)
            {
                ps = transform.GetChild(i).GetComponent<ParticleSystem>();
                if (ps)
                    ps.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
            }
            gameObject.SetActive(false);    //isključi objekt
        }
    }

}
```

12. Klasa AutoDisable

Ukratko, ova klasa nakon nekog vremena isključuje objekt kojemu je dodijeljena, čim se uključi, pokreće korutinu koja će isključiti njen objekt.

```
public class AutoDisable : MonoBehaviour {

    public float time;  //vrijeme u sekundama prije isključenja

       void OnEnable() {   //kad se skripta uključi, pokreni korutinu koja ga isključuje
        StartCoroutine(DisableAfterTime());
        ParticleSystem ps = GetComponent<ParticleSystem>(); //ako je objekt particle
                                                              system, pokreni ga
        if(ps)
        {
            ps.Play(true);
        }
       }

    IEnumerator DisableAfterTime()
    {
        yield return new WaitForSeconds(time);  //pričekaj određeno vrijeme
```

```
        gameObject.SetActive(false);            //isključi objekt
    }

}
```

13. Klasa CollectableItem

Ovu komponentu sadrže objekti koje igrač može pokupiti, kada ih pokupi, oni se isključe, poboljšaju određeno svojstvo igrača i „izbace“ određeni efekt. Kod je slijedeći:

```
public class CollectableItem : MonoBehaviour {

       public enum CollectableItemGain
    {
        MaxHpIncrease,  //povećava hp_max player-a
        HealPlayer,     //povećava hp player-a, uporabom ovg tima moguće je i napraviti
                           objet koji smanjuje hp player-a, not to nije moralno
        BoostSpeed,     //povećava brzinu player-a na neko vrijeme
        None            //ništa
    }

    public CollectableItemGain action;  //koji tip poboljšanja daje ovaj item?

    public object arg1, arg2;   //argumenti poboljšanja

    public GameObject dissapearEffect;  //objekt koji se pojavi nakon što se pokubi ovaj
                            item (neki particle effect)

    Rigidbody rig;  //rigidbody komponenta ovog objekta

    public static Vector3 angVel = new Vector3(0, 2.00712864f, 0);  //brzina rotacije
                           svakog collectable item-a

    void Start()    //inicijalizacija
    {
        rig = GetComponent<Rigidbody>();
    }

    void FixedUpdate()  //fizički frame
    {
        if (Scene.currentGameState == Scene.GameState.playing)  //postavi brzinu item-a
            rig.angularVelocity = angVel;
    }

    void OnTriggerStay(Collider col)    //poziva se kada se nekim objet sudari sa ovim
                                          item-om
    {
        if(col.tag == "Player" && Scene.currentGameState == Scene.GameState.playing)
                           //ako se sudario igrač prilikom igranja
        {
            switch (action)     //ovisno o tipu, poboljšaj/pogoršaj player-a
            {
                case CollectableItemGain.MaxHpIncrease:
                    Scene.player.hp_max += (float)arg1;
                    break;
                case CollectableItemGain.HealPlayer:
```

```
                    Scene.player.GainHp((float)arg1);
                    break;
                case CollectableItemGain.BoostSpeed:

              Scene.player.StartCoroutine(Scene.player.GainSpeed((float)arg1,(float)arg2)
              );
                    break;
                default:
                    break;
            }
            dissapearEffect.SetActive(true);    //ukluči efekt nestanka
            gameObject.SetActive(false);    //isključi ovaj objekt
        }
    }

}
```

14. Klasa FPSStats

Ova klasa je dodijeljena UI objektu koji prikazuje framerate igre, izmjenjuje vrijednost na svojoj tekstnoj komponenti dvaput u sekundi pomoću korutine. Kod:

```
public class FPSStats : MonoBehaviour {

    float delta = 0f;   //promjena vremena od slijedećeg frame-a

    UnityEngine.UI.Text txt;    //tekst na koji se ispisuje

       //inicijalizacija
       void Start () {
        txt = GetComponent<UnityEngine.UI.Text>();
        txt.text = "";
        StartCoroutine(UpdateTxt());
       }

       //svakog frame-a računaj promjenu vremena
       void Update () {
        delta += (Time.deltaTime - delta) * 0.1f;
       }

    IEnumerator UpdateTxt() //Update-aj text dvaput u sekundi
    {
        while(true)
        {
            if (delta != 0f)
            {
                txt.text = "FPS: " + (1f / delta).ToString("00.0");
            }
            yield return new WaitForSeconds(0.5f);
        }
    }
  }
```

15. Klasa MovingObstacle

Objekti koji su platforme te se miču iz točke u točku imaju ovu klasu. Kod:

```
[RequireComponent(typeof(Rigidbody))]   //za rad skripta treba rigidbody komponentu na
                                          objektu
public class MovingObstacle : MonoBehaviour {

       [System.Serializable]   //točka i vrijeme koje treba da se dođe do nje (za
                                  linearne prepreke)
    public class Point
    {
        public float time;  //vrijeme prijelaza do točke (za uglađeni tip to je samo
                                  brzina prijelaza)
        public Vector3 point;   //točka

        public Point(float t, Vector3 pos)
        {
            time = t;
            point = pos;
        }

        public static implicit operator Vector3(Point p)    //za baratenje točkom kao
                                                    vektorom u kodu radi jednotavnosti
        {
            return p.point;
        }

    }

    public enum ObstacleType { Linear, Smoothed };  //mogući tipovi prepreke, linearni
                                                       pomaci i "uglađeni"

    public ObstacleType type = ObstacleType.Linear; //tip prepreke

    public Point[] points;  //točke po kojima se prepreka miče

    public int currentPoint = 0;    //index trenutne točke na koju ide prepreka

    Vector3 relativePos,    //pozicija objekta u početku levela
            basePos;        //pozicija točke s koje objekt ide na sljedeću

    Rigidbody rig;          //Rigidbody komponenta objekta

    void Start()    //Inicijalizacija...
    {
        rig = GetComponent<Rigidbody>();
        relativePos = rig.position;
        basePos = rig.position;
    }

    void FixedUpdate()  //Fizički frame igre
    {
        if (Scene.currentGameState == Scene.GameState.playing)  //ako igrač igra igru
        {
            if (points.Length != 0) //ako objekt ima point-ove
            {
```

```
            //radi pomak s obzirom na tip prepreke
            if (type == ObstacleType.Linear)    //za linearnu prepreku
            {
                Vector3 fromPos = (-(rig.position - (relativePos +
        points[currentPoint].point)));   //smjer od kuda objekt kreće
                rig.velocity = (fromPos.normalized * (Vector3.Distance(basePos,
        points[currentPoint] + relativePos)) / points[currentPoint].time);
    //postavi brzinu tako da objekt stigne na sljedeći point na određeno vrijeme
                if (Vector3.Distance(rig.position, points[currentPoint] +
    relativePos) <= (rig.velocity.magnitude * Time.fixedDeltaTime) ||
    Vector3.Distance(rig.position, points[currentPoint] + relativePos) <= 0.001f)
    //ako je objekt stigao na točku
                {
                    basePos = points[currentPoint] + relativePos;       //točka s
                                                koje se kreće postaje trenutna točka
                    currentPoint = (currentPoint + 1) % points.Length;  //povećaj
                                                                        currentPoint
                    rig.velocity = Vector3.zero;                        //zaustavi
                                                                               objekt
                }
            }
            else    //za smoothed tip prepreke
            {
                rig.velocity = (-(rig.position - (points[currentPoint] +
    relativePos)) / points[currentPoint].time);    //brzina je proporcinalna putu do
            sljedećeg point-a, a ornuto proporcinalna cremenu da se stigne do point-a
                if (Vector3.Distance(rig.position, points[currentPoint] +
                 relativePos) < 0.001f)    //ako je objekt stigao na točku
                {
                    basePos = points[currentPoint] + relativePos;       //točka s
                                                koje se kreće postaje trenutna točka
                    currentPoint = (currentPoint + 1) % points.Length;  //povećaj
                                                                        currentPoint
                    rig.velocity = Vector3.zero;                        //zaustavi
                                                                        objekt
                }
            }
        }
    }
  }
```

16. Klasa Trigger

Ovu klasu imaju objekti tipa Trigger, nevidljivi objekti s kojima s igrač fizički ne sudara, određenim sudarom pozivaju određenu radnji što može loše ili dobro utjecati na igrača, te radnje su umiranje, pobjeda i dobivanje/gubitak hp-a. Kod:

```
public class Trigger : MonoBehaviour
{

    public enum EventAction     //što sudar s igračem donosi
    {
        None,
        PlayerWin,
```

```
    PlayerLose,
    PlayerGainHp
}

public EventAction OnPlayerEnter;   //što se dogodi dok se igrač sudari s trigger-om

public EventAction OnPlayerStay;    //što se događa dok je igrač u sudaru s trigger-
                                            om

public EventAction OnPlayerExit;    //što se dogodi dok igrač izađe iz trigger-a

public object arg1, arg2, arg3;     //argumenti za događaje

void ProcessActions(EventAction ea, object arg) //procesiranje događaja za njegov tip
{
    switch (ea)
    {
        case EventAction.None:
            break;
        case EventAction.PlayerGainHp:
            Scene.player.GainHp((float)arg);
            break;
        case EventAction.PlayerWin:
            Scene.player.Win();
            break;
        case EventAction.PlayerLose:
            Scene.player.Die();
            break;
        default:
            break;
    }
}

void OnTriggerEnter(Collider col)   //poziva se dok se nešto počinje sudarati sa
                                      trigger-om
{
    if (Scene.currentGameState == Scene.GameState.playing)
    {
        if (col.tag == "Player")    //ako je objekt koji se sudario igrač, procesiraj
                                      događaje za ovaj trigger
            ProcessActions(OnPlayerEnter, arg1);
    }
}

void OnTriggerStay(Collider col)    //poziva se dok je nešto u sudaru sa trigger-om
{
    if (Scene.currentGameState == Scene.GameState.playing)
    {
        if (col.tag == "Player")    //ako je objekt koji se sudara igrač, procesiraj
                                      događaje za ovaj trigger
            ProcessActions(OnPlayerStay, arg2);
    }
}

void OnTriggerExit(Collider col)    //poziva se dok nešto izađe iz sudara sa trigger-
                                      om
{
    if (Scene.currentGameState == Scene.GameState.playing)
```

```
        {
            if (col.tag == "Player")    //ako je objekt koji je izašao iz sudara igrač,
                                          procesiraj događaje za ovaj trigger
                ProcessActions(OnPlayerExit, arg3);
        }
    }

}
```

17. Dodatne skripte

Osim svih tih klasa koje upravljaju igrom, u projektu su korištene dvije dodatne skripte, ObjImporter za učitavanje 3D modela tokom izvođenja igre i WaterBasic, koji kontrolira vodu u igri.

ObjImporter učitava samo statičke 3D modele u obj formatu, igra ne podržava učitavanje animiranih 3D modela tokom izvođenja jer je implementacija dosta kompleksna, izvor ove skripte naveden je u literaturi.

WaterBasic je standardni resurs koji dolazi sa instalacijom engine-a, dostupan je bilo kome za bilo kakvu uporabu kao i ObjImporter, taj skript omogućava vodi da se pomiče, bez njega bi voda bila kao nepomična tekstura.

18. XML Stabla

Kao što je i prije navedeno, igra koristi XML (Extensible Markup Language) za zapis vanjske imovine, uključujući postavke igre, level-e, objekte, materijale, meni (koristi isti zapis kao i level), prazan level u editor-u (koristi isti zapis kao i level, samo služi kao predložak), postoji i stablo za opis Ammo objekta, ali se u igri ne koristi, već je zamijenjen unutrašnjim objektom. Sve te informacije su zapisane u xml datotekama, mada su za zapis materijala korištene datoteke s ekstenzijom material, u njima se ne krije ništa drugo nego xml kod, koji sadrži informacije o tom materijalu.

Skripte koje rade sa XML kodom u igri su Scene, LevelLoader i GameEditor, najbitnija klasa za rad s XML datotekama je XmlDocumet koja nudi metode i svojstva za čitanje i pisanje po XML dokumentima.

1. Stablo Level-a

Koristi se za zapis Level-a igre. Sadrži podatke o poziciji, rotaciji, veličini, imenu i roditelju svakog objekta, svaki objekt je zasebni element u grani Objects, sadrži minimalnu dozvoljenu visinu level-a, poziciju, rotaciju i veličinu igrača, poziciju i rotaciju

kamere, tip kamere te njene točke, Offset za te točke, put do slijedećeg level-a te info da li je level zaključan, ovakve datoteke generira game editor.

Jednostavniji primjer:

```
<?xml version="1.0" encoding="UTF-16"?>
<Scene>
  <Objects>
    <Object Name="LVL1Base">
      <Transform>
        <Position>(-2.5902790, -0.8000000, 7.5932830)</Position>
        <Rotation>(0.0000000, 0.0000000, 0.0000000)</Rotation>
        <Scale>(8.0971710, 8.0971710, 8.0971710)</Scale>
        <Parent>Root</Parent>
      </Transform>
      <ObjRef>(#Interal)LVL1Base</ObjRef>
    </Object>
    <Object Name="Cliff">
      <Transform>
        <Position>(0.8887471, 1.5259390, 5.7780480)</Position>
        <Rotation>(0.0000000, 0.0000000, 0.0000000)</Rotation>
        <Scale>(1.4024100, 1.4024100, 1.4024100)</Scale>
        <Parent>Root</Parent>
      </Transform>
      <ObjRef>/Objects/Cliff.xml</ObjRef>
    </Object>
</Objects>
  <SceneInfo>
    <MinHeight>-10.0000000</MinHeight>
  </SceneInfo>
  <PlayerInfo>
    <Transform>
      <Position>(2.1103320, 0.0000000, -2.8804550)</Position>
      <Rotation>(0.0000000, 0.0000000, 0.0000000)</Rotation>
      <Scale>(1.0000000, 1.0000000, 1.0000000)</Scale>
    </Transform>
  </PlayerInfo>
  <CameraInfo>
    <Position>(7.5722820, 5.0000000, -2.6886030)</Position>
    <Rotation>(20.0000100, 265.6250000, 0.0000000)</Rotation>
    <CameraMode>PointLookAt</CameraMode>
    <Points>
      <Point>(7.6536000, 5.0000000, -2.5747590)</Point>
      <Point>(7.6536000, 5.0000000, 2.4327780)</Point>
      <Point>(7.1408150, 5.3663570, 7.5632650)</Point>
      <Point>(4.5815980, 6.6176240, 19.2738200)</Point>
      <Point>(-1.3243040, 7.1771580, 20.2985000)</Point>
      <Point>(-11.0455500, 9.0342640, 19.2299800)</Point>
      <Point>(-15.5507600, 10.5133800, 15.0684900)</Point>
      <Point>(-15.5814300, 11.5935700, 7.4374880)</Point>
    </Points>
    <PointOffset>(0,0,0)</PointOffset>
  </CameraInfo>
  <NextLevel>/Levels/Level2/Level.xml</NextLevel>
  <Locked>False</Locked>
 </Scene>
```

2. Stabla objekata

Vanjski objekti u igri mogu biti tipa dekoracije, prepreke, AI, pomična prepreka, stvari za sakupljanje (CollectableItem) i Trigger-i, svaki objekt ima svoju ikonu (za editor) i ime.

Dekoracije su objekti s kojima se igrač ne sudara, oni su samo u sceni kako bi uljepšali level, zato xml od dekoracije sadrži samo podatke o 3D modelu i materijalu.

```
<Object Type="Decoration">
  <Decoration>
    <Mesh>/Models/Bush1.obj</Mesh>
    <Material>/Materials/Bush1.material</Material>
  </Decoration>
  <Icon>/Objects/Bush1.png</Icon>
   </Object>
```

Prepreke su poput dekoracija, samo što se igrač i AI objekti sudaraju s njima.

```
<Object Type="Obstacle">
  <Obstacle>
    <Mesh>/Models/Cliff.obj</Mesh>
    <Material>/Materials/Cliff.material</Material>
    <Collider Convex="True">MeshCollider</Collider>
  </Obstacle>
  <Icon>/Objects/Cliff.png</Icon>
   </Object>
```

Pomične prepreke za razliku od običnih koje samo stoje se gibaju po danim točkama.

```
<Object Type="MovingObstacle" CustomTag="MovingPlatform">
  <Rigidbody>
    <Mass>1000000000</Mass>
    <UseGravity>false</UseGravity>
    <Interpolate>Interpolate</Interpolate>
    <CollisionDetection>Discrete</CollisionDetection>
    <FreezePosition>fff</FreezePosition>
    <FreezeRotation>ttt</FreezeRotation>
  </Rigidbody>
  <MovingObstacle Type="Linear">
    <Mesh>/Models/MP1.obj</Mesh>
    <Material>/Materials/MP1.material</Material>
    <Collider Convex="True">MeshCollider</Collider>
    <Points>
      <Point>
        <Time>6</Time>
        <Position>(0,0,0)</Position>
```

```
      </Point>
      <Point>
        <Time>6</Time>
        <Position>(2.666,0,10.5)</Position>
      </Point>
    </Points>
  </MovingObstacle>
  <Icon>/Objects/MovingPlatformLR.png</Icon>
   </Object>
```

U igri nisu korišteni vanjski AI objekti, ali su i dalje podržani, nedostatak je to što vanjski objekti ne mogu biti animirani pa vanjski AI objekt nije toliko impresivan bez animacija.

```
<Object Type="AI">
  <Rigidbody>
    <Mass>1</Mass>
    <UseGravity>true</UseGravity>
    <Interpolate>Interpolate</Interpolate>
    <CollisionDetection>Discrete</CollisionDetection>
    <FreezePosition>fff</FreezePosition>
    <FreezeRotation>ttt</FreezeRotation>
  </Rigidbody>
  <AI>
    <Mesh>(#Interal)Cube</Mesh>
    <Material>/Materials/Mat1.material</Material>
    <Collider1>BoxCollider</Collider1>
    <Collider2Size>(0.95,0.2,0.95)</Collider2Size>
    <Collider2Center>(0,0.5,0)</Collider2Center>
    <Hp>1</Hp>
    <HpMax>1</HpMax>
    <AIType>Attack</AIType>
    <Speed>100</Speed>
    <Rotates>true</Rotates>
    <RotateSpeed>120</RotateSpeed>
    <PatrolPoints>
      <point>(-3,0,0)</point>
      <point>(3,0,0)</point>
    </PatrolPoints>
    <ToleratedDistance>0.15</ToleratedDistance>
    <ActiveDistance>8</ActiveDistance>
    <OnGroundRaycastLength>0.501</OnGroundRaycastLength>
    <ChecksBeforeItGoes>true</ChecksBeforeItGoes>
    <ForwardRayscastCheckLength>1.0</ForwardRayscastCheckLength>
    <DownRaycastcheckLength>1.0</DownRaycastcheckLength>
    <AmmoObj>/Ammos/poison1.xml</AmmoObj>
    <AmmoLifeTime>3</AmmoLifeTime>
    <RateOfFire>2</RateOfFire>
    <RelaxTime>0</RelaxTime>
    <ShootPointOffset>(0,0.5,0.5)</ShootPointOffset>
    <shootHeight>2</shootHeight>
    <damageOnCollision>-1</damageOnCollision>
  </AI>
  <Icon>/Objects/AITest.png</Icon>
   </Object>
```

CollectableItem objekti se samo rotiraju i igrač ih može pokupiti.

```
<Object Type="CollectableItem">
  <CollectableItem>
    <Mesh>/Models/Arrow.obj</Mesh>
    <Material>/Materials/LeafDecor1.material</Material>
    <Collider>SphereCollider</Collider>
    <OnCollect>None</OnCollect>
    <arg1>
    </arg1>
    <arg2>
    </arg2>
    <DissapearEffect>3</DissapearEffect>
  </CollectableItem>
  <Icon>/Objects/PointerArrow.png</Icon>
   </Object>
```

Trigger objekti su u igri nevidljivi, ali opet odlučuju o svemu.

```
<Object Type="Trigger">
  <Trigger>
    <Mesh>(#Interal)Cube</Mesh>
    <Collider>BoxCollider</Collider>
    <OnPlayerEnter>PlayerWin()</OnPlayerEnter>
    <OnPlayerStay>
    </OnPlayerStay>
    <OnPlayerExit>
    </OnPlayerExit>
  </Trigger>
  <Icon>/Objects/WinTrigger.png</Icon>
   </Object>
```

3. Stablo postavki

Sadrži puteve do otključanih level-a, postavke jačine zvuka i prikaza FPS-a, jedina datoteka s ovim stablom je GameSettings.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<Settings>
  <UnlockedLevels>
    <LevelPath>/Levels/Level2/Level.xml</LevelPath>
    <LevelPath>/Levels/Level3/Level.xml</LevelPath>
    <LevelPath>/Levels/Level4/Level.xml</LevelPath>
    <LevelPath>/Levels/Level5/Level.xml</LevelPath>
  </UnlockedLevels>
  <Options>
    <MusicVlume>0.1684409</MusicVlume>
    <ShowFps>True</ShowFps>
  </Options>
   </Settings>
```

4. Stablo materijala

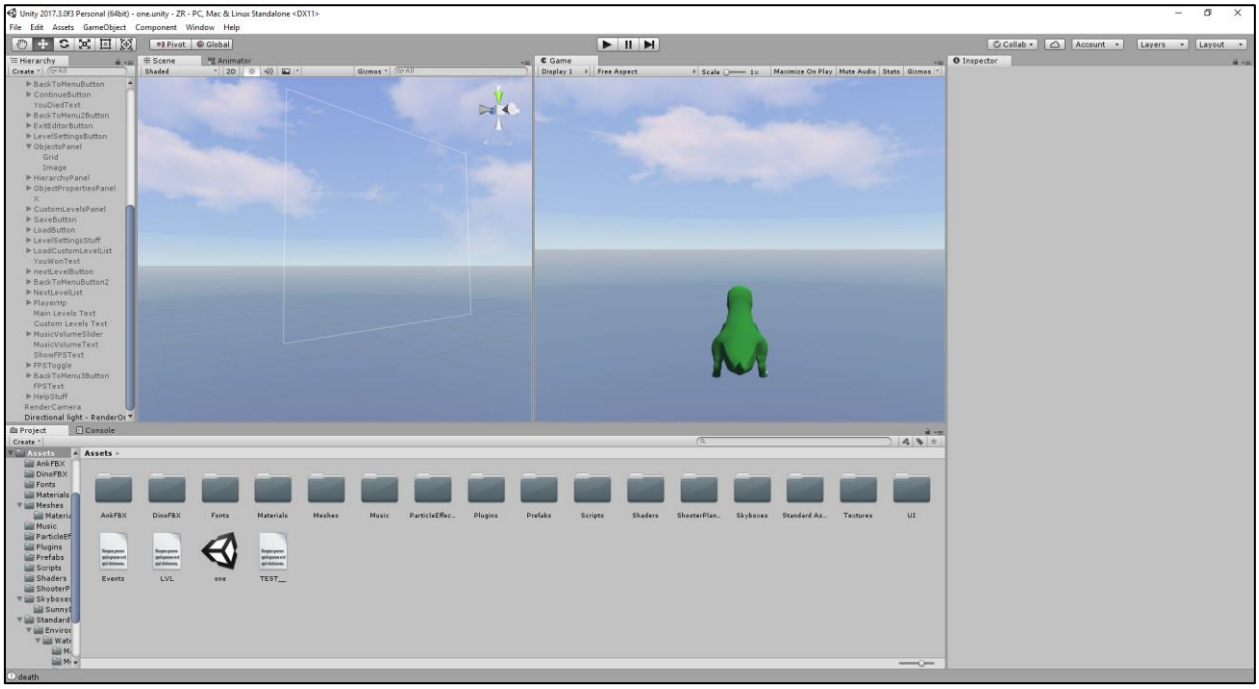
Sadrži put do teksture materijala, boju materijala, da li je materijal transparentan, filtriranje teksture, anizotropni level, boju materijala u hex zapisu RGBA, materijali u igri imaju element normal map koji se zapravo i ne koristi radi jednostavnosti, tako da što piše u tom elementu, bit će ignorirano.

```
<Material>
	<Texture Filtering="Bilinear" AnisoLevel="16">/Textures/BushTex1.png</Texture>
	<Color>FFFFFF00</Color>
	<NormalMap></NormalMap>
</Material>
```

Potrebno je napomenuti to da je poredak elemenata u xml datotekama igre iznimno važan te „pobrkane“ datoteke neće biti dobro učitane jer tako rade metode za čitanje u skriptama, po redoslijedu elemenata.

3. Rad na Unity Game Engine-u

Rad na samom engine-u je manje-više rad sa korisničkim sučeljem, prozor Unity-a izgleda ovako (barem postava prozora korištena pri kreiranju igre, UI engine-a je moguće prilagoditi vlastitim potrebama):



Slika 13 Izgled projekta u Unity Game engine-u

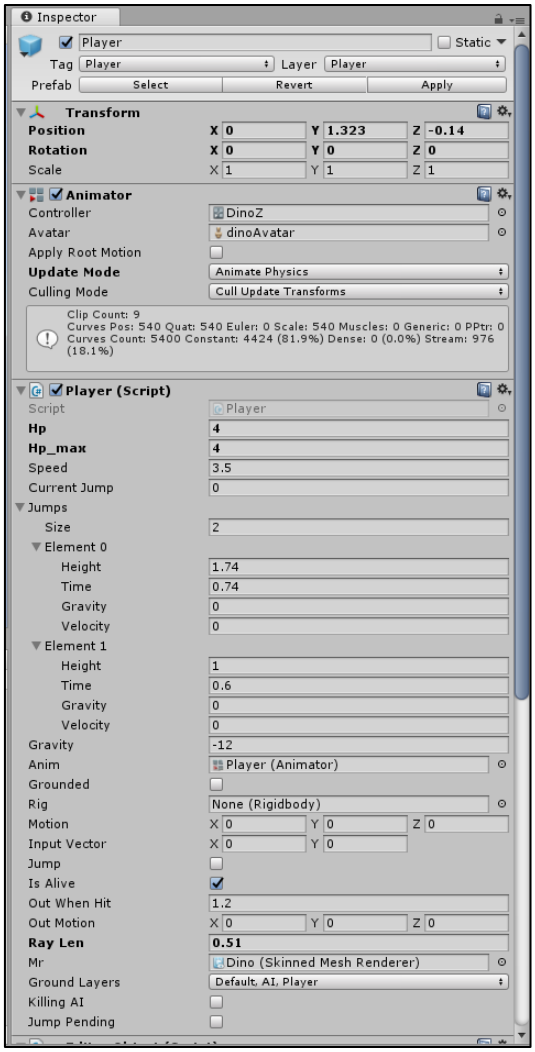
Otvoreno je 5 prozora, pogled na projekt, inspektor, pogled na igru, pogled na scenu i hijerarhija objekata u sceni.

U inspektoru se uređuju svojstva objekata, pogled na igru kada se pokrene igra unutar editor-a (Ctrl + P), u pogledu na scenu, scena se uređuje, hijerarhija je popis svih objekata u sceni, a pogled na projekt pokazuje svu imovinu igre (Assets) .

Engine kao i mnogi drugi programi ima traku izbornika u kojoj se određuju njegove postavke, editiraju objekti, ubacivanje ili kreiranje stvari za igru, dodavanje objekata, dodavanje komponenti objektima, novih prozora u programu i pomoć.

1. Inspector

Prozor engine-a koji služi za uređivanje svojstava selektiranih objekata u Scene prozoru ili Project prozoru (ime, layer, tag, pozicija, rotacija, veličina i statičnost), dodavanje i uklanjanje komponenti te uređivanje njihovih svojstava.

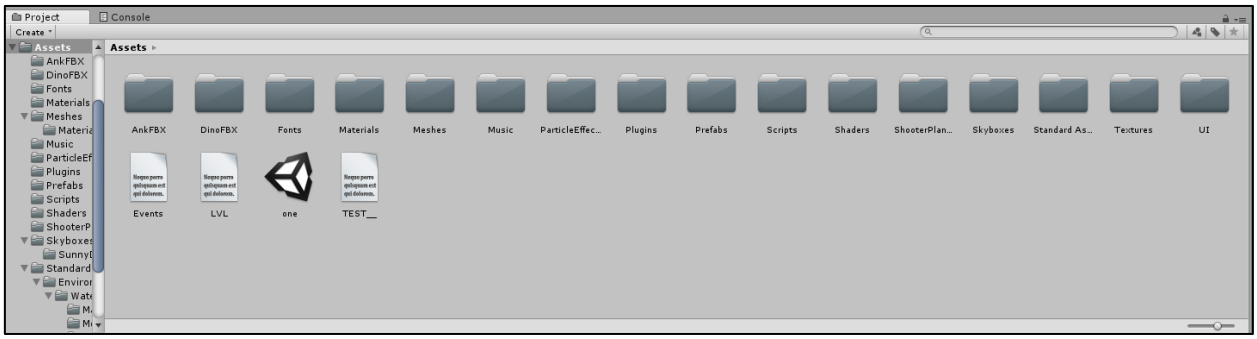


Slika 14 Izgled inspektora

Bitna stvar je da se klasama koje su naslijeđene od MonoBehaviour-a mogu određivati vrijednosti javnih ne-statičnih varijabli i vidjeti njihove vrijednosti tokom izvođenja igre u engine-u.

2. Project

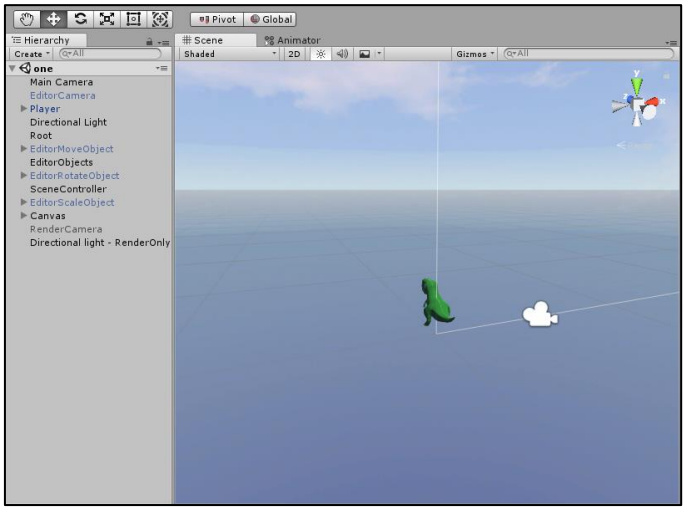
Ovaj prozor nudi pregled imovine igre, to uključuje skripte, 3D modele, materijale, teksture, objekte u igri, scene u igri (Igra koristi samo jednu scenu), zvukovi (U igri su jedini zvukovi dvije melodije), fontovi (Igra koristi font zvan Hanken Light), animatori, datoteke i to su svi tipovi imovine (Asset-ova) korišteni u projektu, radi organizacije, imovina je uređena po mapama. Ukratko prozor Project je kao mali File Browser projekta.



Slika 15 Svi unutarnji resursi igre u Project View-u

3. Scene i Hierarchy

Usko vezani prozori, jer scene prikazuje izgled scene, a Hierarchy popis svih objekata, u oba prozora objekti se mogu selektirati, u scene prozoru objektima se može mijenjati pozicija (W), rotacija (E) i Veličina (R) te 2D pozicija i veličina (tipka T, za UI objekte).



Slika 16 Scene i Hierarchy prozori u igri

Igra sadrži samo jednu scenu u kojoj je sve, od editora, postavki do level-a, ova izvedba baš i nije uobičajena za igre u Unity-u jer većina njih ima više scena koje imaju različite

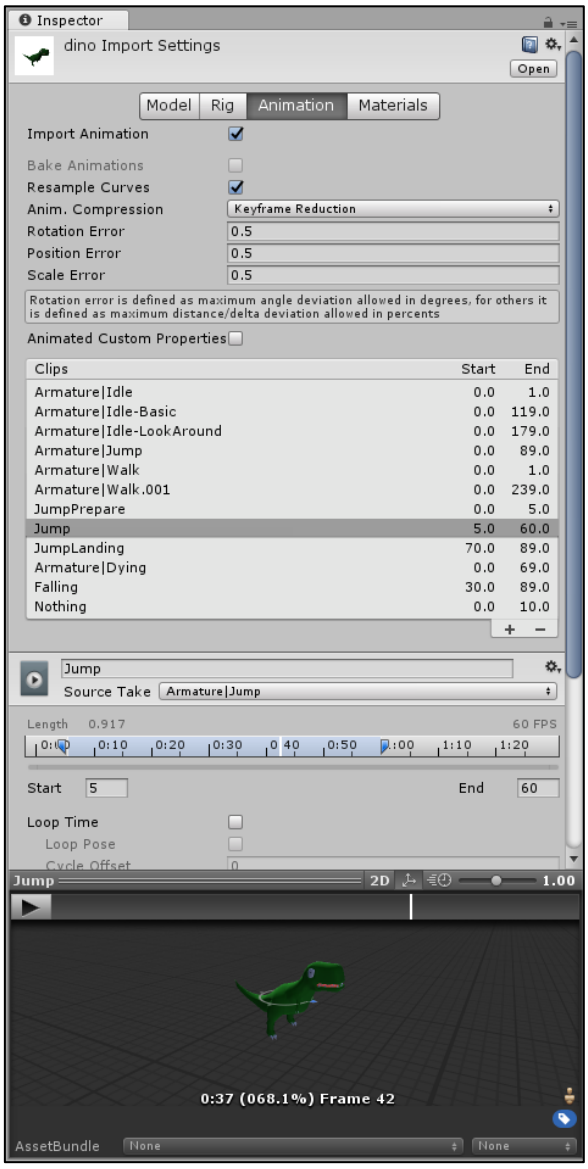
uloge npr. prva scena je glavni meni, druga izbor level-a, treća prvi level itd. S takvim načinom, izrada vlastitih level-a (level editor) nije moguća za ostvarivanje, ali pošto ova igra ima samo jednu scenu koju kontrolira, to je moguće i tako je izvedeno.

Početni objekti u jednoj sceni u igri su:

- Main Camera – kamera koja nudi pogled u igru
- Editor Camera – prikazuje početnu lokaciju kamere i igri
- Player – dinosaur tj. igrač
- Directional Light – Izvor svjetlosti u sceni, kada ga ne bi bilo, igra bi bila mračna
- Root – sadrži sve objekte level-a
- Editor(Move/Rotate/Scale)Object – alati u game editoru za uređivanje objekata
- EditorObjects – prazan objekt koji sadrži objekte kojima game editor raspolaže
- SceneController – prazan objekt koji sadrži komponente za upravljanje scenom
- Canvas – sadrži sve UI objekte
- RenderCamera – kamera koja služi game editor-u za renderiranje ikoni objekata
- Directional Light – RenderOnly – svijetlo koje osvjetljava renderirane objekte

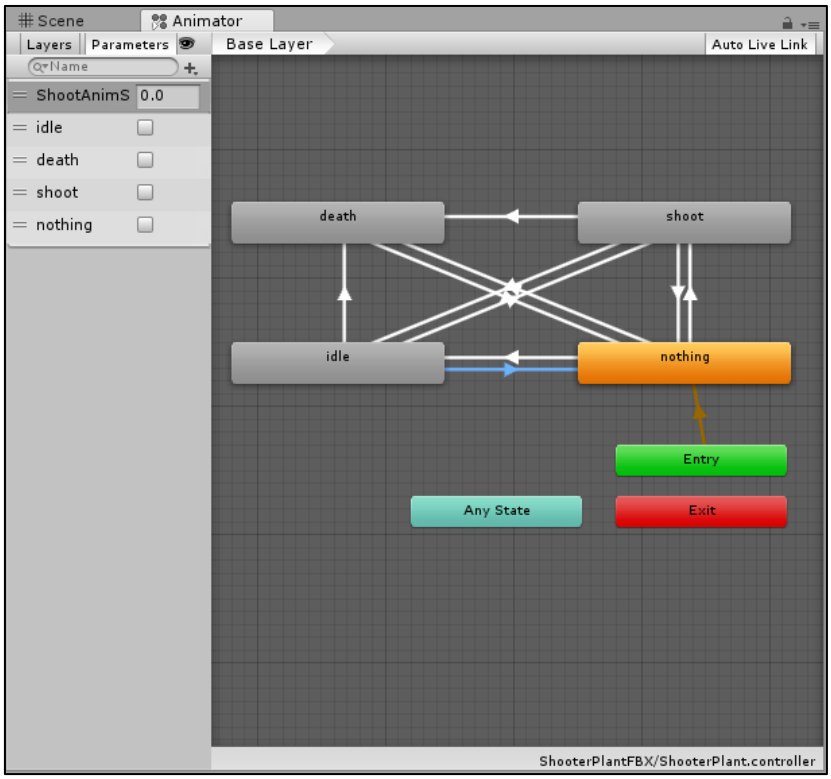
4. Rad animacija i efekata u igri

Jedini animirani objekti u igri su Dino, Ank (drugi dinosaur) i ShooterPlant. Animirani su u programu Blender, u igru ubačeni u FBX formatu za 2D animacije, pošto odmah ubačeni klipovi u igru često nisu dobro postavljeni u smislu da im klipovi nemaju dobre duljine, neki nisu postavljeni u način ponavljanja tokom izvođenja (Loop) i za dinosaur-a i ShooterPlant objekte u klipovima JumpPrepare i shoot bilo je potrebno dodati događaje za skok i ispaljivanje metaka, zato ih je u inspektoru bilo potrebno urediti pomoću jednostavnog korisničkog sučelja.



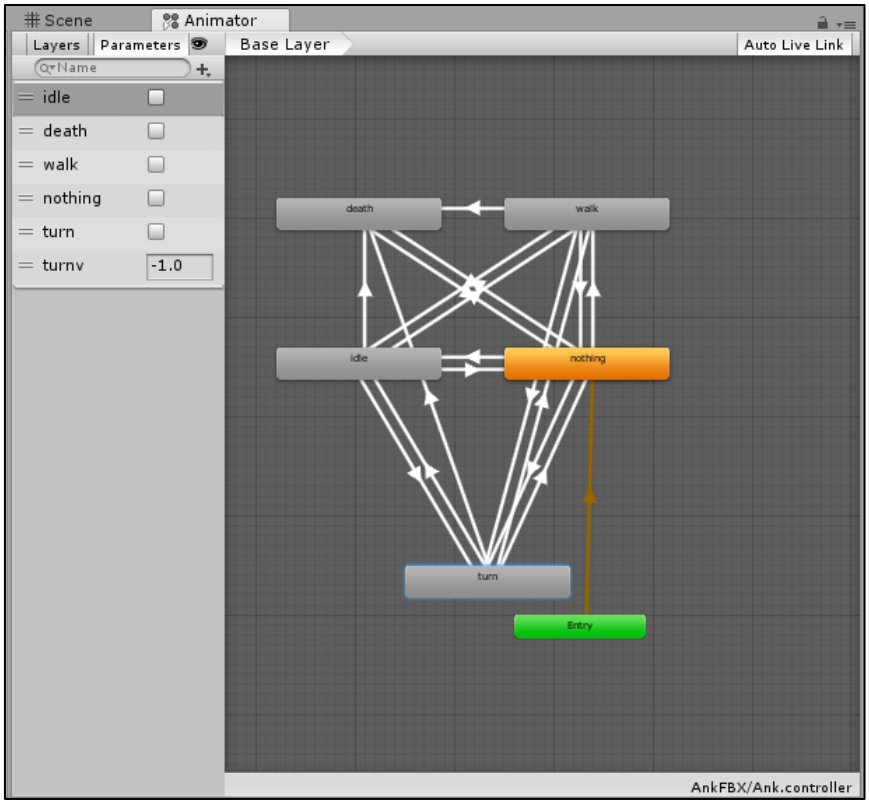
Slika 17 Opcije animacija za dinosaura

Druga važna stvar u igri je sustav izvođenja animacija (Animator), u osnovi Animator je state machine, u kojem su stanja klipovi ili skupine klipova animacija (Blend Trees, „pomiješani“ klipovi, za svaki Blend Tree postoji brojčani (float) parametar animator-a koji određuje koja animacija se izvodi), ta stanja vezana su tranzicijama koje su uvjetovane parametrima Animatora, ako je parametar istinit tranzicija ide iz trenutnog klipa na klip na koji ga tranzicija navodi, animacije se mogu direktno pokrenuti metodama CrossFade (glađi prijelaz) i Play (direktan prijelaz) klase animator, u igri su tri animatora, animator dinosaura, animator ank-a i animator biljke.



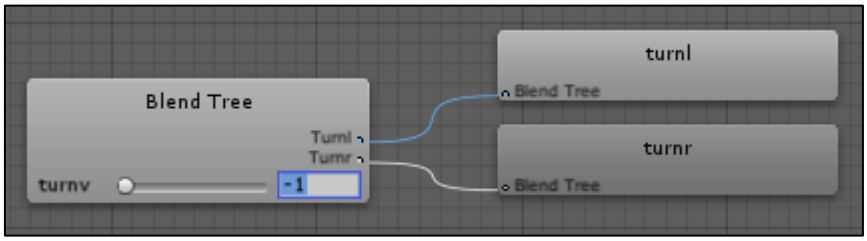
Slika 18 Animator ShooterPlant-a

U ovom animatoru su 4 parametra tipa Bool i jedan parametar tipa float koji određuje brzinu izvođenja animacije shoot kako bi biljaka imala animaciju sinkroniziranu s ispaljivanjem metaka, uvijete za tranzicije su imena parametra i klipa na koji tranzicija pokazuje.

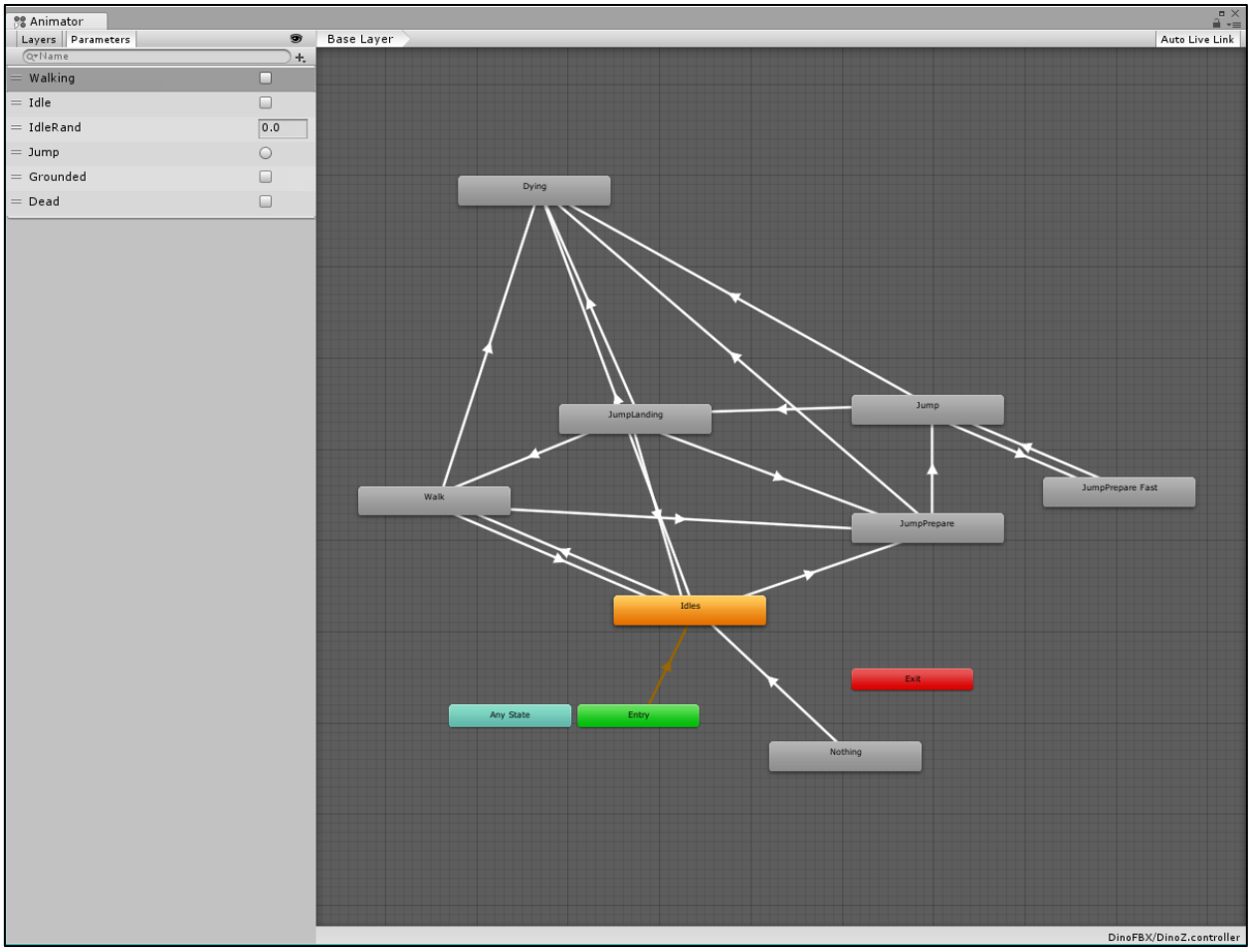


Slika 19 Animator Ank-a

Radi po istom principu ako i animator ShooterPlant-a, float parametar turnv određuje koja animacija okretanja se izvodi (prema lijevo ili prema desno), no vrijednost tog parametra je uvijek -1 tako da animacija koja se izvodi u Blend Tree-u za okret je prema lijevo, jer tokom testiranja, iskazalo se da smjer po kojemu se objekt rotira dobro uklapa nevažno o smjeru okreta animacije.



Slika 20 Grupa animacija za smjer (turn) Ank-a



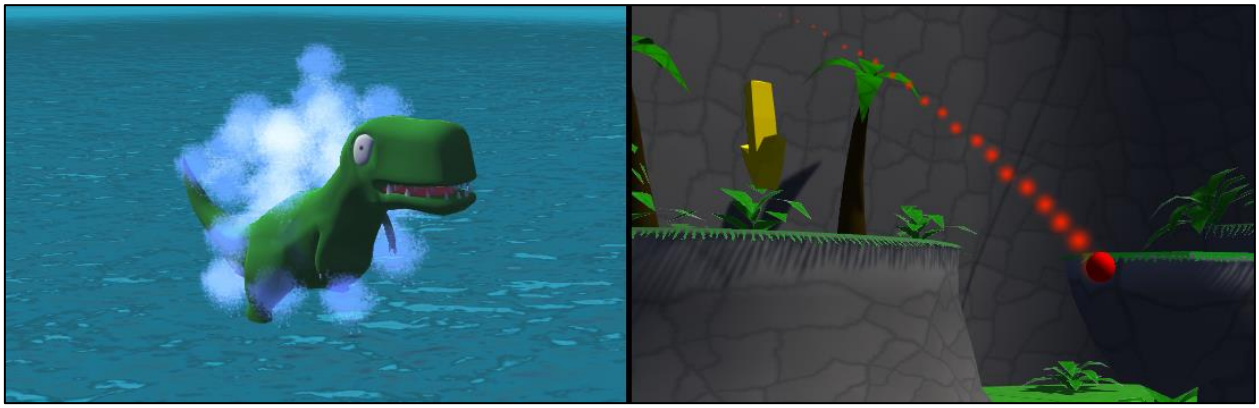
Slika 21 Animator Dinosaura

Uobičajena animacija u izvođenju kod igrača je idle, klip walk izvodi se kada je igrač u pokretu, za skok se koristi parametar jump koji izvodi metodu jump u klasi Player, nakon JumpPrepare klipa, izvođenje se prebacuje na klip Jump, u slučaju double jump-a, izvođenje se brzo prebacuje na JumpPrepare Fast klip koji poziva metodu Jump u klasi Player, nakon toga klip Jump se ponovo kontinuirano izvodi. Ako igrač nakon skoka padne na tlo, brzo se izvede animacija JumpLanding te nakon nje Walk ili Idle klipovi ovisno o tome dali je igrač u pokretu. Parametar IdleRand određuje koja od dvije Idle animacije se izvodi, vrijednost tog parametra određuju korutine u klasi Player. Kada igrač umre izvede se animacija Dying neovisno o trenutnoj animaciji izvođenja.

Metode za postavljanje vrijednosti parametara su SetFloat(string ime, float vrijednost), SetBool(string ime, float vrijednost) i SetTrigger(string ime) u klasi Animator koja je ujedno i komponenta u Unity-u, podatak tipa Trigger u animatoru nije ništa drugo nego Bool koji

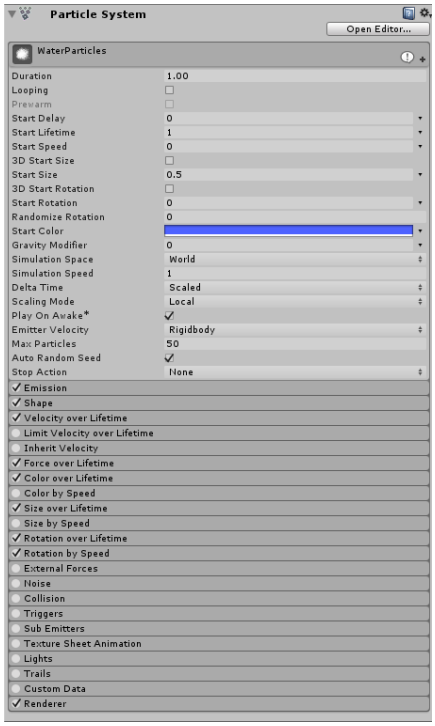
kada biva postavljen u vrijednost true se postavlja u vrijednost false kada se tranzicija s njime provede. Za dohvaćanje vrijednosti parametara koriste se metode GetBool (string ime) i GetFloat (string ime) u klasi Animator.

Efekti u igri su objekti koji sadrže komponentu particle system, pod pojmom efekti misli se na efekte s česticama, primjeri tih objekata su pljusak vode ili ostaci metka od ShooterPlant-a.



Slika 22 Primjeri particle efekata

Izrađeni su uređivanjem komponente Particle System u inspektoru.



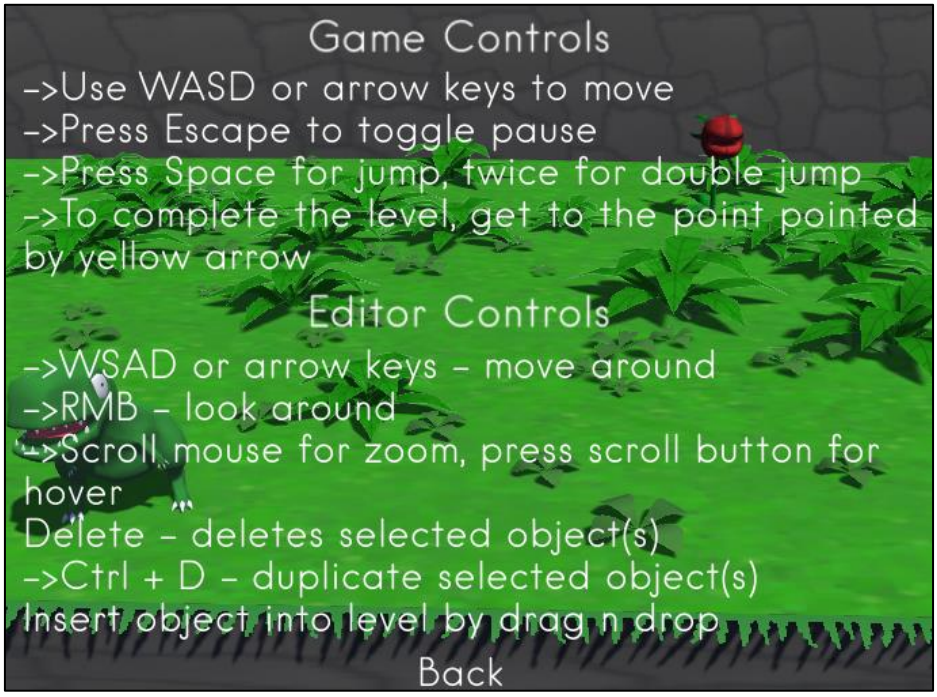
Slika 23 Komponenta Particle System

5. Korisničko sučelje igre

Korisničko sučelje igre se uglavnom sastoji od gumbova i labela, uz nekoliko polja za upis u editoru, sadrži i jedan dropdown meni u editoru, jedan checkbox (prikaz FPS-a) i slider (prilagođivanje jačine zvuka) i slike (ikone objekata).

Slika 24 Opcije Igre

Promjenom vrijednosti slider-a poziva se metoda SetVolume u klasi kamera jer je kamera jednini audio izvor u sceni. Promjenom checkbox-a poziva se metoda ToggleFPS u klasi Scene.

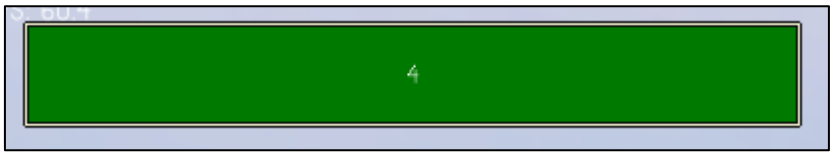


Slika 25 pomoć u igri (kontrole)

UI pomoći nije ništa nego skup labela koje objašnjavaju kontrole.

Slika 26 Izbor level-a

Izbor level-a je generiran u klasi Scene, ako level-a ima previše tj. ne stanu u zaslon, onda se njima može scroll-ati jer su djeca praznog objekta s komponentom (Vertical) Layout Group čiji roditelj ima Scroll Rect komponentu koja omogućuje scroll-anje.



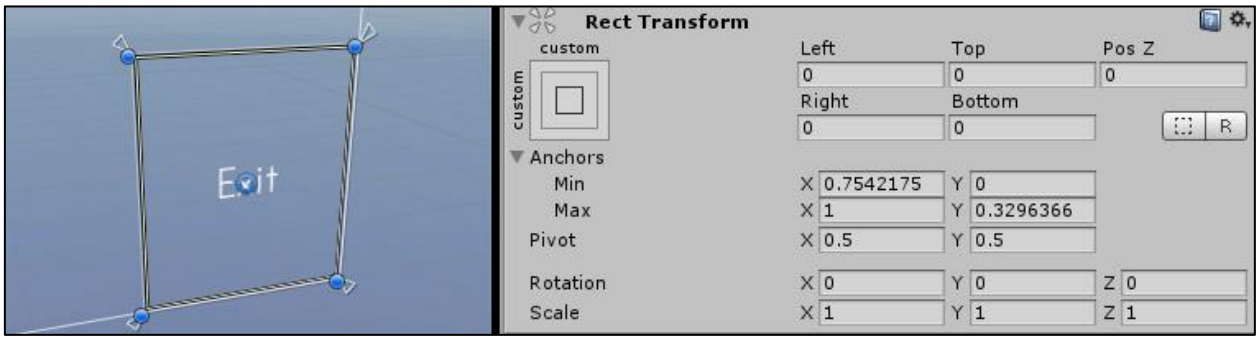
Slika 27 Indikator HP-a igrača

Realiziran je kao slika unutar okvira (druga slika) koja se izdužuje/skraćuje ovisno o HP-u igrača.

Slika 28 Opcije igre tokom pauze

Slika 29 UI kada igrač umre/pobijedi

UI se prilagođava svim rezolucijama jer svi elementi imaju konfiguriranu RectTransform komponentu, koja ima opciju anchors koja određuje poziciju i veličinu UI elementa s obzirom na dani postotak zaslona koji zauzima. Primjer se vidi na izlaznom gumb-u iz game editor-a.

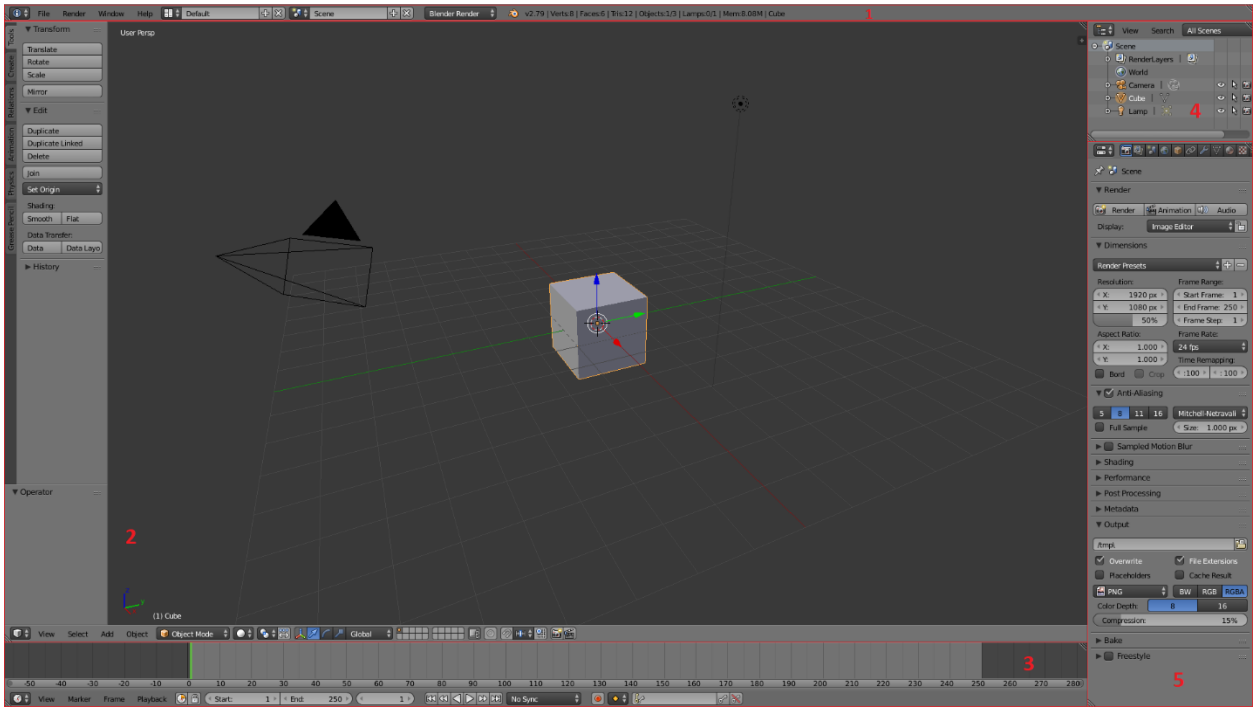


Slika 30 Konfiguracija Rect Transform-a gumba za izlaz iz editor-a

4. Grafičko dizajniranje

Grafika u igri je jednostavna, dovoljno jednostavna da bi ju većina računala mogla pokretati sa prihvatljivim framerate-om, svi 3D modeli i teksture su kreirani i programu za 3D modeliranje zvanom Blender.

Blender je open source program namijenjen za 3D modeliranje, rigging, animiranje, simulaciju, renderiranje, sadrži i mogućnosti za video editiranje.



Slika 31 Korisničko sučelje Blender-a

Sadrži 5 aktivnih editor-a:

1. Info – sadrži osnovne alate za spremanje/učitavanje datoteka, postavke programa, alat za promjenu ili dodavanje „screenova“ (rasporeda editora u prozoru), dodavanje scena, promjenu renderer-a, prikaz svih odrađenih radnji…
2. 3D View – pogled na scenu, na tom prozoru obavlja se većina radnji od modeliranja do animiranja, selektiranja objekata i sl.
3. Timeline – služi za animiranje
4. Outliner – zapis svih scena i objekata u hijerarhijskom obliku te neke od njihovih postavki

5. Properties – svojstva renderiranja, scene i objekata

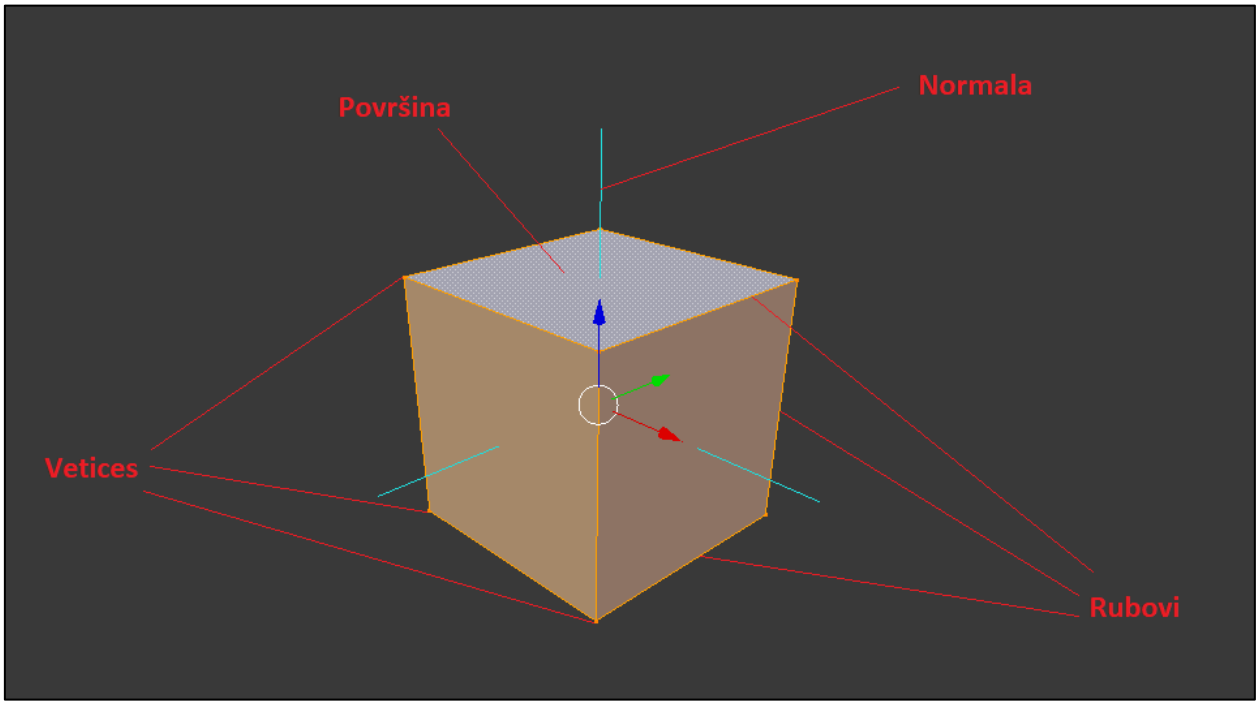
1. Modeliranje

3D model se u osnovi sastoji od 3D „točaka“ takozvanih Vertex-a (Vertices), to su u osnovi samo pozicije u 3D prostoru.

Vertex-i se mogu spajati te se time dobivaju rubovi (Edges).

I konačno dio 3D modela koju rubovi okružuju je površina (Face), valja spomenuti kako i uz to postoje i 3D vektori zvani normale (Normals) tj. okomice, uglavnom njihova najbitnija vrijednost je njihov smjer jer se time određuje u kojem smjeru je površina okrenuta, u Blender-u površine su dvostrane, no postoje aplikacije u kojima su površine jednostrane pa su zato normale bitne, jedan od tih aplikacija je Unity game engine.

Kombinacija svih tih tri, odnosno četiri elemenata se naziva Mesh (Mreža).



Slika 32 Osnovni dijelovi 3D modela

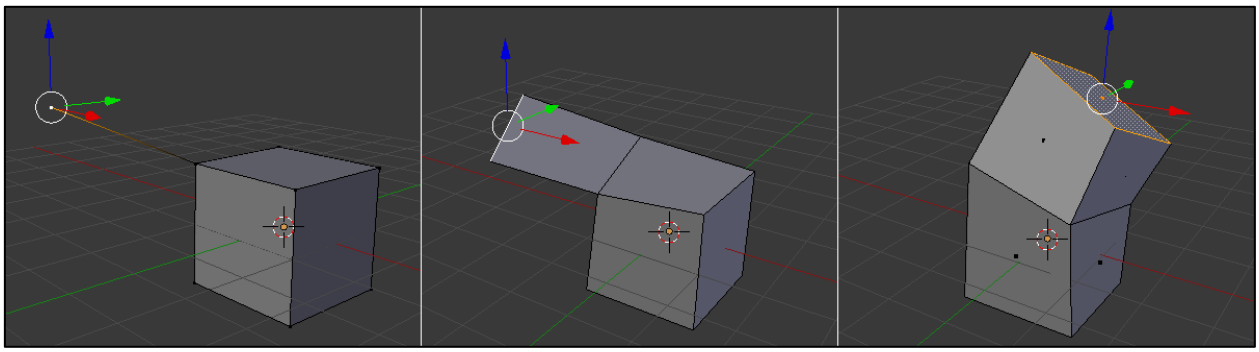
Modeliranje se obavlja u Edit Mode-u, na taj mode lakše je prijeći pomoću Tab tipke, u njemu se editira samo jedan objekt, ako je više objekata selektirano, editira se onaj koji je zadnji selektiran.

U Edit Mode-u postoji više načina selekcije dijelova Mesh-a, to su Vertex, Edge i Face:

Slika 33 načini selekcije dijelova modela u blender-u

Za sva ta tri elementa vrijede naredbe pomaka, rotacije i promjene veličine (G, R i S tipke), no valja napomenuti kako za Vertex-e rotacija i promjena veličine ne vrijede jer je to samo točka u prostoru.

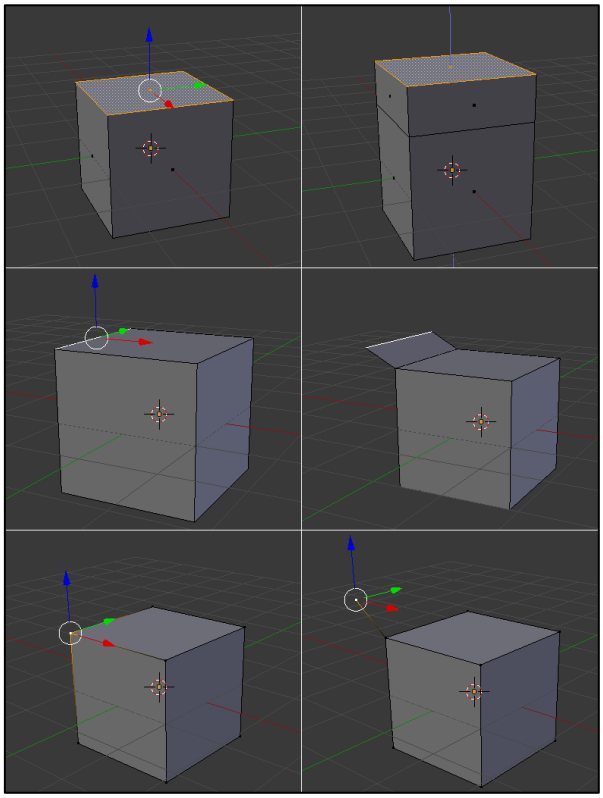
Za dodavanje novog vertex-a potrebno je držati Ctrl te lijevom tipkom miša kliknuti na prostor gdje će se dodati, ako je prije toga bila selektirana točka, nova točka će biti spojena s tom točkom, ako je prije toga bio selektiran rub, stvoriti će se novi takav rub i biti spojen sa selektiranim te će između njih biti nova površina, ako je prije bila selektirana površina, stvoriti će se nova površina na toj lokaciji koja će biti spojena sa rubovima stare površine, stara površina će biti izbrisana.



Slika 34 Dodavanje geometrije 3D modelu

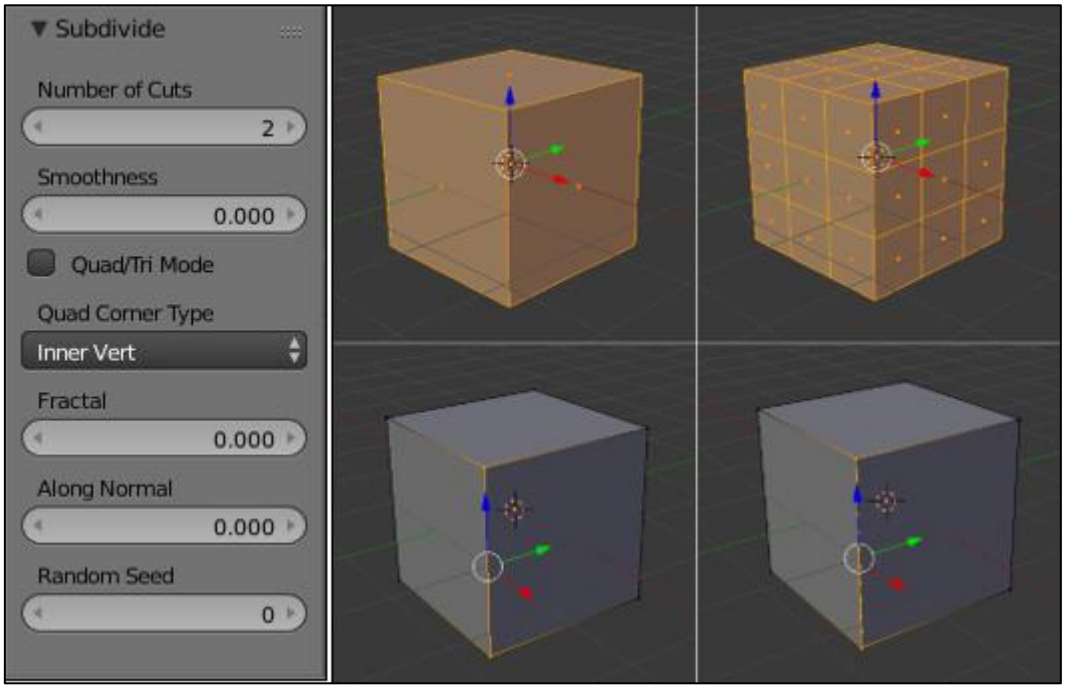
Još neki od korištenih alata za modeliranje:

- Extrude (E) – Izdužuje Vertex, Edge ili Površinu, površinu izdužuje po normali (opet je moguće odrediti specifičnu os (X,Y ili Z)), Vertex ili Edge po pomaku miša, na kraju je lijevom tipkom potrebno potvrditi promjenu.



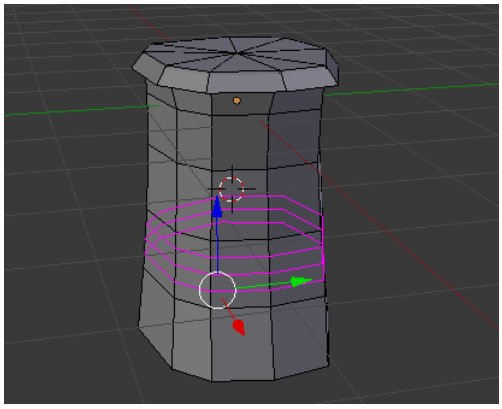
Slika 35 Rezultati Extrude naredbe

- Subdivide (W) – „Reže“ tj. razdvaja površinu u manje dijelove (ako je moguće, inače razdvaja rubove oko nje) ili rub na više manjih rubova, može se odrediti koliko presjeka je potrebno napraviti u donjem lijevom izborniku 3D View-a.



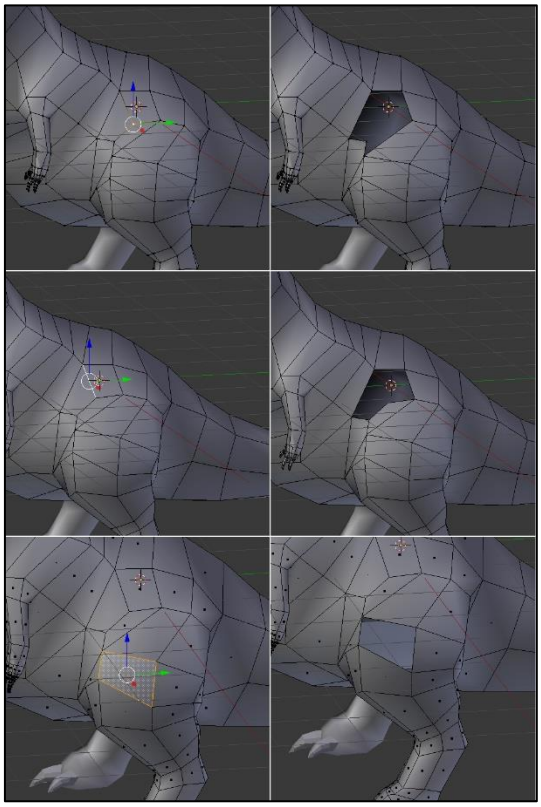
Slika 36 Rezultati Subdivide naredbe

- Loop Cutting (Ctrl + R) – Rezanje po nekoj „petlji“, moguće je odrediti koliko petlji će se odraditi okretanjem kotačića miša, pomicanjem miša nakon lijevog klika određuje se položaj rezanja, pokazivač miša je potrebno postaviti na objekt jer se time određuje smjer rezanja, na kraju je opet potrebno sve potvrditi drugim lijevim klikom.



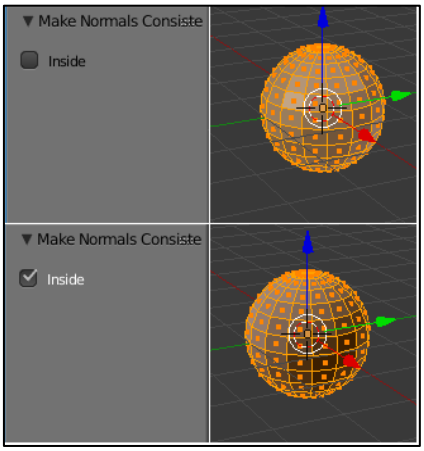
Slika 37 Loopcut naredba

- Brisanje u edit mode-u je nešto drugačije nego u edit mode-u, postoje tri osnovne opcije:
    - Brisanje vertex-a – briše vertex i sve rubove i površine vezane uz njega
    - Brisanje edge-ova – briše edge i face-ove vezane u njega
    - Brisanje face-ova – briše samo face



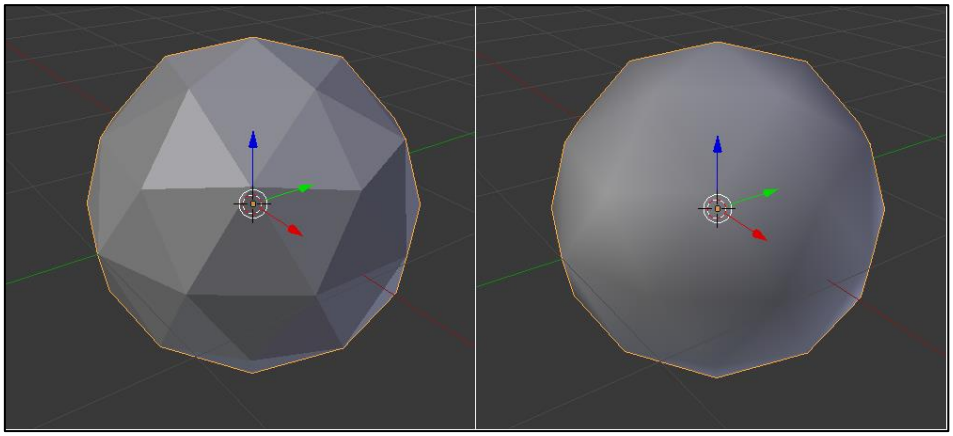
Slika 38 Brisanje dijelova modela

- Preračunavanje normala (Ctrl + N) – Tokom modeliranja moguće je da neke površine dobiju obrnute normale (površina je zatamnjena), ova naredba to popravlja, jedna opcija koju nudi je smjer normala (prema unutrašnjosti ili prema van).



Slika 39 Krivo okrenute normale su zatamnjene

- Postavljanje glatkoće/izoštrenosti površina – U izborniku Shading / UVs može pod tekstom Faces može se odrediti dali su površine glatke ili oštre.



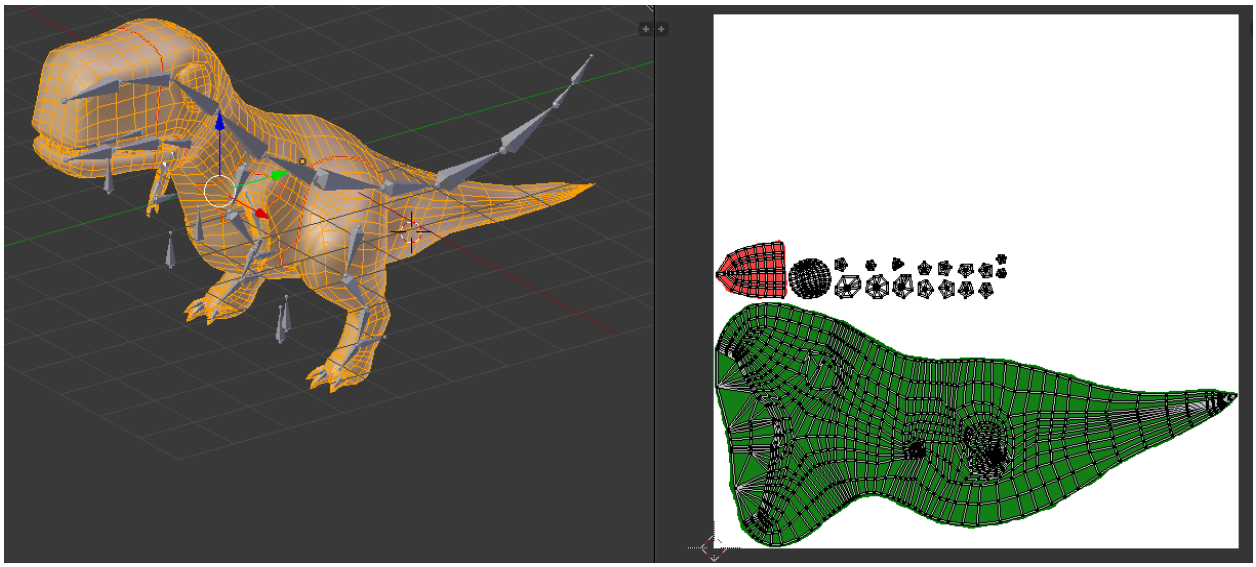
Slika 40 Oštra/Glatka površina

2. UV Mapping i Texturing

Za dodavanje teksture na neki model, potrebno je imati samu teksturu i model treba imati UV mapu.

UV mapa je „2D prikaz“ modela, određuje kako se tekstura postavlja na model, u blender-u je UV mape moguće generirati i nakon toga uređivati, za to se koristi prozor UV Image Editor.

Primjeri UV mape kod dinosaura:



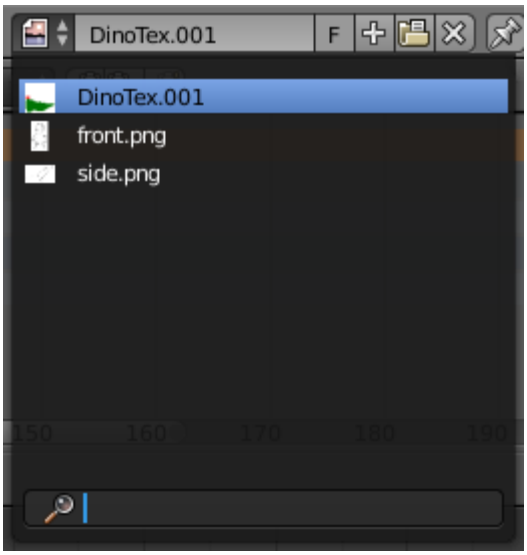
Slika 41 UV Mapa dino-a

Za generiranje UV mape u blender-u, potrebno je biti u edit mode-u, tipka kojom se pokreće naredba je U.

UV vertex-i i face-ovi se mogu pomicati (G), rotirati (R) i povećavati (S), važno je da su njihovi pripadni dijelovi modela selektirani u 3D View-u.

Za dodavanje teksture objektu za pregled, u donjem djelu UV Image Editora je potrebno stvoriti sliku ili dodati neku postojeću.

Za prikaz te slike, u 3D View-u je potrebno postaviti način prikaza na Texture.



Slika 42 Odabir teksture

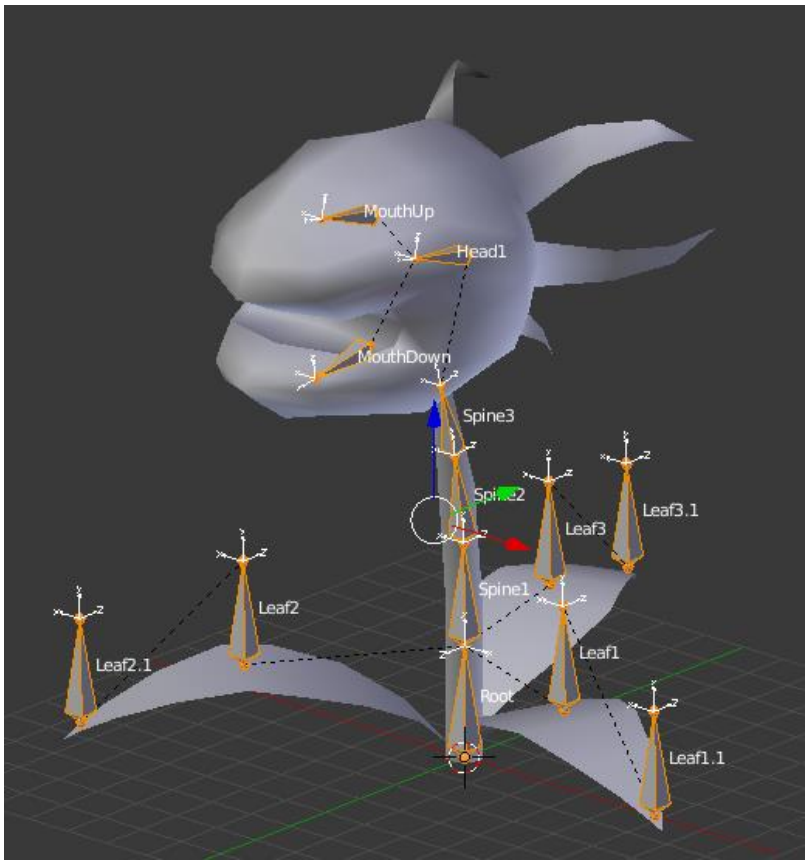
3. Armature

Prije nego što se neki model može detaljno animirati, on treba imati armaturu, to je objekt koji je načinjen od kostiju (bones). Svaka kost treba imati pridružen dio modela kojim ona upravlja tako npr. kost za prst bit trebala imati sve vertex-a prsta.

Kosti mogu biti spojene (time jedino prva kost određuje položaj cijele grupe, duge kost se onda ne mogu pomicati i promjena jedne kosti u njihovom nizu odnosno strukturi utječe na sve sljedeće kosti), jedna može biti „parent“ od druge odnosno mogu se hijerarhijski raspoređivati, time promjene na roditeljskoj kosti utječu i na djecu, i kosti još mogu biti neovisne jedna o drugoj.

Armaturu je lako dodati u Object Mode-u te urediti u Edit Mode-u, Kosti se u edit mode-u mogu dodavati produljivanjem od krajeva (Extrude – E, time su odmah spojene), gumbom Add, duplikacijom (Ctrl + D). Mogu se spajati naredbom Parent (Ctrl + P, daje opciju da ih spoji ili samo učini parent-om).

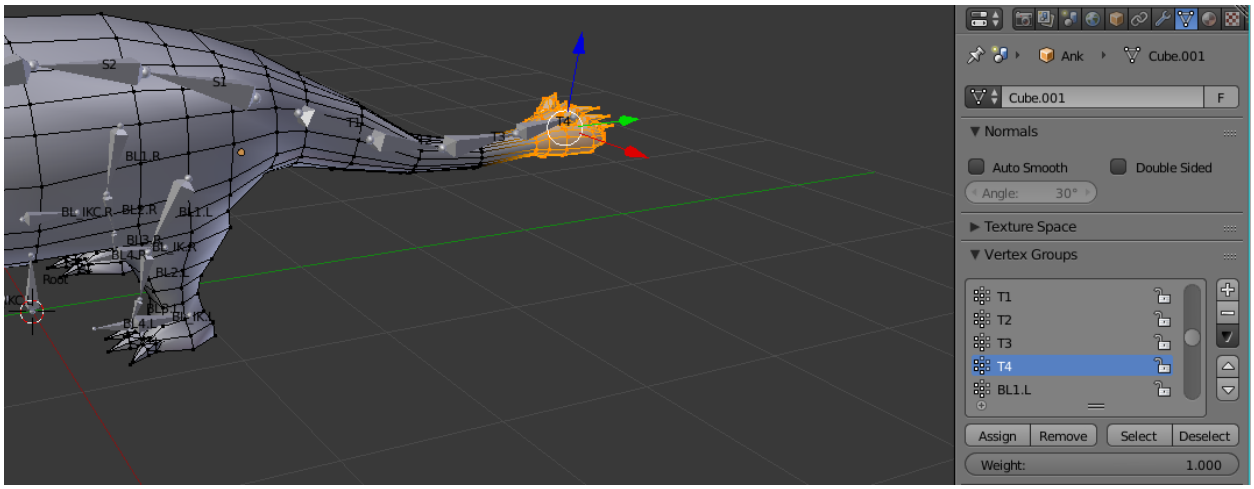
Postavke kostiju (npr. ime) mogu se mijenjati u prozoru Properties pod izbornikom za armaturu ili Bone.



Slika 43 Armatura ShooterPlant-a

Kosti sa parent-om su povezani crticama a spojene kosti se neće odvojiti kako god postavljene bile.

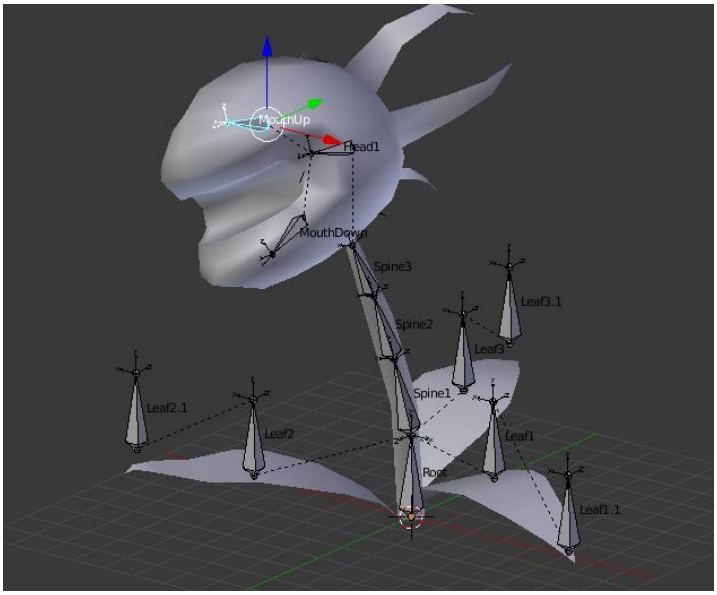
Za „spajanje“ armature s modelom potrebno je selektirati model pa armaturu i nakon toge pritisnuti Ctrl + P te odabrati opciju With Empty Groups, time se objektu modela dodaju grube vertex-a za svaku kost, ime grupe je ime kosti koja će „upravljati“ vertex-ima te grupe. Te grupe su trenutno prazne i mogu se vidjeti u izborniku Properties prozora, nakon toga je potrebno otići u Edit Mode za model te dodijeliti vertex-e pripadnim kostima, to je jednostavno, potrebno je selektirati pripadne vertex-e te njihovu grupu pa pritisnuti Assign, nakon toga nije loše kliknuti na lokot od grupe tako da se to više neće mjenjati (da ne bi došlo do neke slučajne dodjele koja nije potrebna).



Slika 44 Grupe vertex-a Ank-a

U postavkama za Armaturu u Properties prozoru je moguće uključiti opciju X-Ray koja će prikazati kosti ako su unutar modela i opciju koja prikazuje njihova imena za lakše snalaženje.

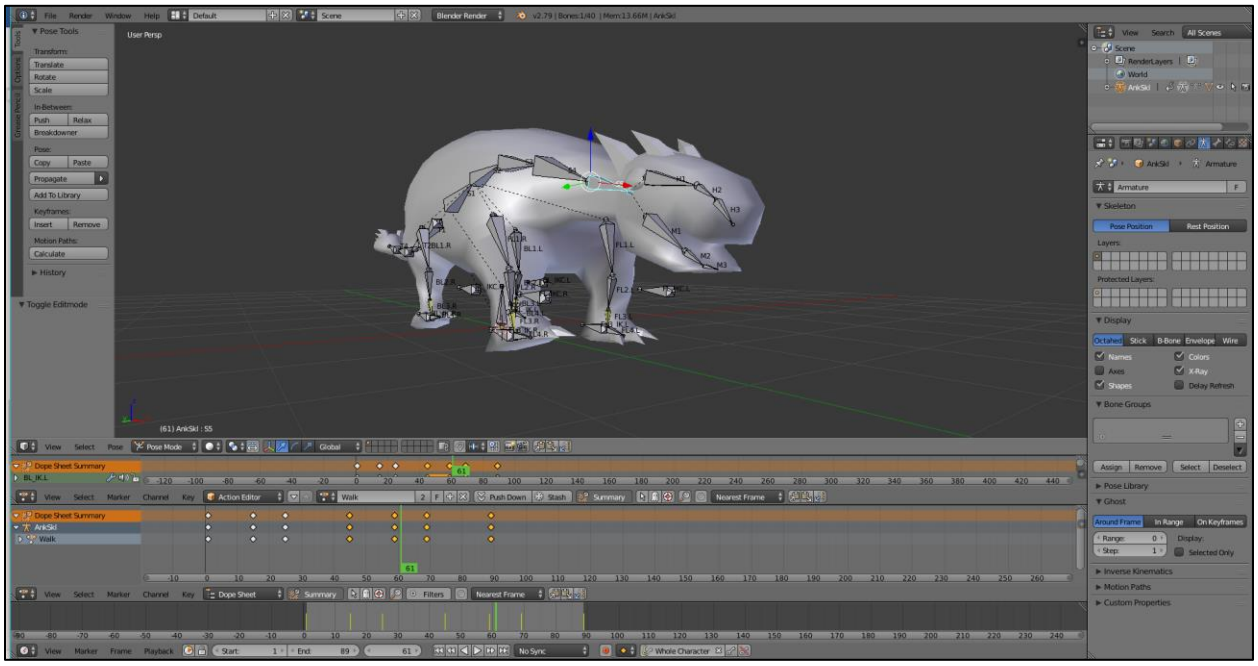
Nakon dodjele je moguće otići u Pose Mode za Armaturu te pomicanjem kostiju testirati ispravnost kostiju (dali je sve dobro bilo dodijeljeno), ako je sve u redu model je spreman za animiranje.



Slika 45 Model ShooterPlant-a upravljan kostima

4. Animiranje

Animiranje u blender-u se bazira na keyframe-ovima, svaki keyframe sadrži određene informacije o stanju objekta u jednom frame-u, tako npr. keyframe za bone sadrži informacije o poziciji, rotaciji i veličini bone-a, za animaciju je potrebno imati barem 2 keyframe-a između kojih se dešava tranzicija između njihovih vrijednosti evaluirana po krivulji koju blender automatski generira te ju je moguće uređivati. Keyframe-ovi se mogu dodavati u Object (Animiraju se objekti, ne dijelovi objekata tipa bone-ovi) i Pose mode-u (uglavnom za animiranje bone-ova), prečac s kojim se dodaju je tipka I (Insert keyframe menu) te nudi odabir za koju je vrijednost taj keyframe. Prozori uglavnom korišteni za animiranje su 3D View (za animiranje), Dope Sheet prozor (daje bolji uvid u keyframe-ove te njihovo uređivanje), Timeline (vrijeme animacije i playback) i Graph Editor (za uređivanje krivulja vrijednosti keyframe-a).



Slika 46 Izgled prozora za animiranje Ank Modela

5. Zaključak

Ukratko, po mom mišljenju igra je mogla biti bolja, ali nije loša, Projekt ima mnogo linija koda, ali ipak veoma malo objekata koji bi ga učinili zanimljivijim, ideja za razdvajanje vrsta objekata na vanjske te mogućnost izrade vlastitih level-a i korištenje XML-a se iz prve ruke činila kao odlična, ali je donijela više komplikacija nego koristi, no ipak igra je funkcionalna i za izradu svega uloženo najmanje 800 sati rada i tek na kraju sam shvatio da sam mogao odabrati nešto jednostavnije za završni rad.

6. Literatura

Referenca Unity-a - https://docs.unity3d.com/ScriptReference/

Izvor ObjImporter klase - http://wiki.unity3d.com/index.php?title=ObjImporter

Formula za AI tipa Shooter - https://www.youtube.com/watch?v=IvT8hjy6q4o

Referenca XmlDocument klase –

https://msdn.microsoft.com/en-us/library/system.xml.xmldocument(v=vs.110).aspx

Blender-ov manual - https://docs.blender.org/manual/en/dev/

Stranica s koje su preuzeti zvukovi - www.dl-sounds.com

Vlastiti seminar za predmet Multimedija s temom 3D modeliranje