Foos-Game
Tehnička dokumentacija
Verzija 1.0

Studentski tim:

Davor Najev

Fran Androić

Antonio Hohnjec

Dominik Šarić

Marin Lovrinović

Nastavnik: Željka Mihajlović

Sadržaj

Tehnička dokumentacija

1. Opis razvijenog proizvoda

Foos-Game je računalna igra, simulacija stolnog nogometa, prilagođena za igru s kontrolerom. Namjena igre je vjerno rekreirati iskustvo igranja stvarnog stolnog nogometa, stoga je namijenjena za igru dva igrača.

Jedno od ključnih svojstava aplikacije je podrška za različite kontrolere, što igračima pruža fleksibilnost u izboru računalne periferije korištene pri igranju. Osim toga, igra je optimizirana brojnim platformama, omogućujući igračima da uživaju u iskustvu stolnog nogometa bez obzira na operacijski sustav njihovog računala. Grafičko sučelje je pristupačno i jednostavno, a igru je moguće pokrenuti gotovo instantno. Modeli su rađeni po uzoru na stvarne, s određenim stilističkim prilagodbama.

Za izradu igre korišten je pogon za računalne igre *Unity*, zbog raspona mogućnosti koje pruža, fleksibilnosti u pristupu razvoju aplikacija, te kompatibilnosti s više platformi. 3D modeli su napravljeni u programu *Blender*, zbog jednostavnosti učenja i korištenja njegovih alata i kompatibilnosti s različitim formatima.

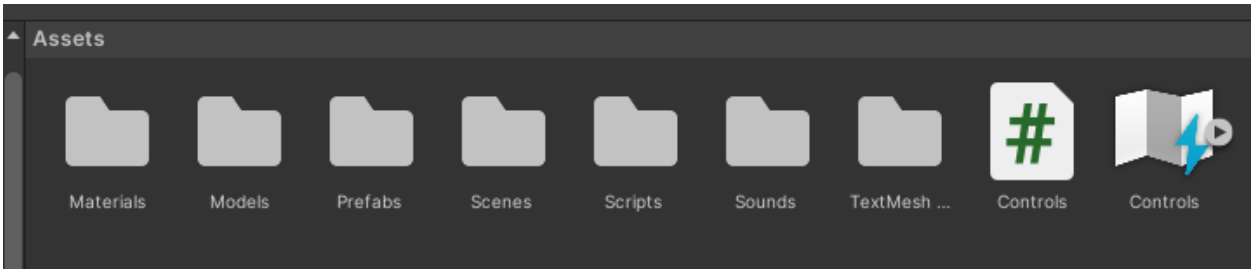
2. Tehničke značajke

2.1. Arhitektura programske potpore

Aplikacija je pisana u pogonu za računalne igre *Unity 2022.3.11f1* koristeći programski jezik *C#*. Modeli su napravljeni u programu za izradu 3D modela *Blender*. Za uređivanje koda korišteni su *Visual Studio 2022* i *Visual Studio Code*.

2.1.1. Osnove arhitekture

Programsko rješenje je strukturirano sukladno standardima razvoja u okruženju *Unity*, dakle datoteke su raspoređene u dodijeljene direktorije prema svojoj vrsti ili funkciji kako je pokazano na slici 1.



Slika 1. Struktura direktorija Unity projekta

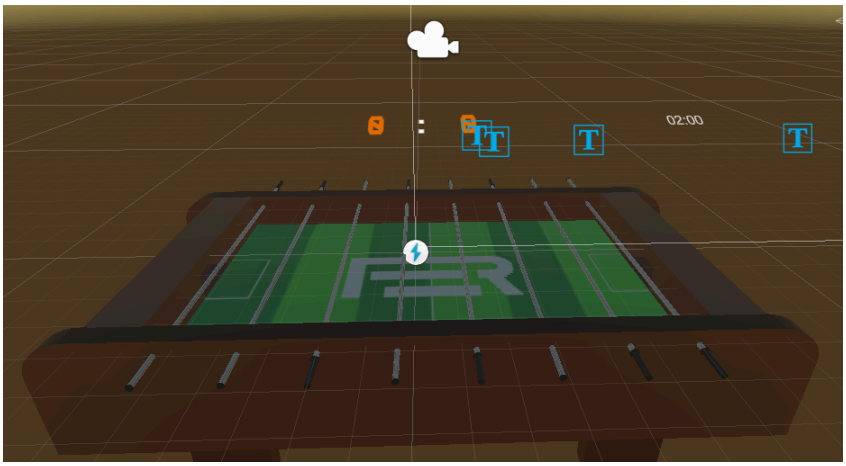
Svrha svakog direktorija je sljedeća:

- **Materials:** Sadrži materijale koje koriste *prefabs* i objekti.
- **Models:** Sadrži sirove modele uvezene iz Blendera.
- **Prefabs:** Objekti za stvari kao što su stol, šipka, loptica ili figura, koje uključuju modele, materijale, skripte i slično.
- **Scenes:** Sadrži scene kao što su *MainScene*, *SettingsScene*, *GameScene*, te *EndScene*.
- **Scripts:** *C#* skripte koje definiraju ponašanje komponenti.
- **Sounds:** Direktorij zvukova i glazbe koja se nalazi u igri.
- **TextMeshPro:** Generirani objekti ekstenzije *TextMeshPro* koji se koriste za izradu grafičkog sučelja.

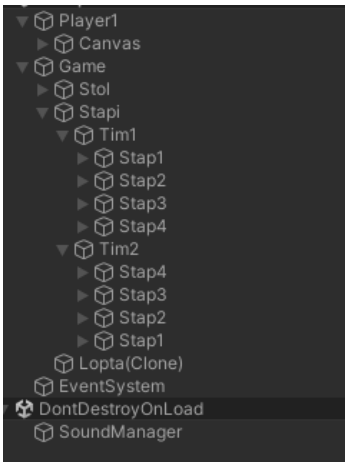
Skripta *Controls* i objekt *Controls* je generirao novi i moderni sustav programa *Unity*, a služe za upravljanje kontrolama.

2.1.2. Pregled glavne scene

U ovom potpoglavlju bit će objašnjena glavna scena projekt u kojoj se odvija igra. Sama scena prikazana je na slici 2, a hijerarhija objekata postavljenih u nju na slici 3. Glavnina ostalih scena je grafičko sučelje pa će one biti obrađene u svom potpoglavlju.



Slika 2. Glavna scena s objektima u sceni



Slika 3. Hijerarhija objekata

Objašnjenje objekata: *Player1* sadrži kameru i grafičko sučelje, *Game* sadrži sve druge objekte vidljive u sceni (stol, šipke, loptica…), kao i *GameController.cs* skriptu koja upravlja igrom tako što sadrži stanje igre i bitne funkcije koje se pozivaju u bitnim događajima. *EventSystem* se koristi za sustav događaja, a *SoundManager* za upravljanje zvukom.

2.1.3. Skripta za upravljanje igrom

Skripta *GameController.cs* samo s definicijama metoda, bez implementacija:

```
public class GameController : MonoBehaviour
{
      [SerializeField] TextMeshProUGUI scoreText1;
      [SerializeField] TextMeshProUGUI scoreText2;

      public int score1 = 0;
      public int score2 = 0;
      public GameObject lopta;
      private GameObject loptaInstanca;
      private SoundManager soundManager;

      private List<Transform> cameras;
      private List<Vector3> initialCameraPositions;

      void Start();
      void Update();
      private void OnDestroy();
      public void ResetRound();
      private void ShakeTable();
      IEnumerator ShakeCameras();
      public void scoreTeam1();
      public void scoreTeam2();
}
```

Možemo vidjeti da ova skripta ima mogućnost pristupa bitnim akcijama i upravlja bitnim aspektima životnog ciklusa igre, ima mogućnost dodavanja bodova timovima, može ponovno pokretati runde, protresti stol, upravlja zvukom i slično.

Pri inicijalizaciji svake *Unity C#* skripte, prvo se izvrši metoda *Start*, njena definicija je sljedeća:

```
void Start()
{
      soundManager = FindObjectOfType<SoundManager>();
      soundManager.PlayIngameMusic();
      cameras = FindObjectsOfType<Camera>().Select(x =>
x.transform).ToList();
      initialCameraPositions = cameras.Select(x =>
x.position).ToList();
      ResetRound();
}
```

Prvo se dohvati *SoundManager* objekt te ga se pita da počne izvoditi *in-game* glazbu. *SoundManager* objekt se instancira na početku igre u glavnom izborniku, gdje se postavljaju njegove početke postavke (uključujući i *in-game* glazbu). Nakon toga se u listu postavljaju reference na kamere, kao i na njihove početne pozicije (te reference se koriste u ostalim metodama). Nakon toga se još poziva metoda *ResetRound* koja označava početak prve runde.

Definicija metode *ResetRound*:

```
public void ResetRound() {
       if (loptaInstanca) {
              Destroy(loptaInstanca);
       }
       loptaInstanca = Instantiate(lopta, new Vector3(0.438f, 0, 0),
Quaternion.identity, transform);
      int strength = 100;
      int x = Random.Range(-strength, strength);
      int z = Random.Range(-strength, strength);
      loptaInstanca.GetComponent<Rigidbody>()
            .AddForce(new Vector3(x, 0, z));
}
```

Kao što se može vidjeti, ova metoda uništava postojeću lopticu (ako postoji), stvara novu i izbacuje ju u nasumičnom smjeru. Loptica poziva vlastitu skriptu *BallHandler.cs* u kojoj se detektira kolizija s golovima, te se poziva funkcija *scoreTeam1* ili *scoreTeam2*, ovisno o tome u koji je gol ušla loptica. Definicije spomenutih funkcija:

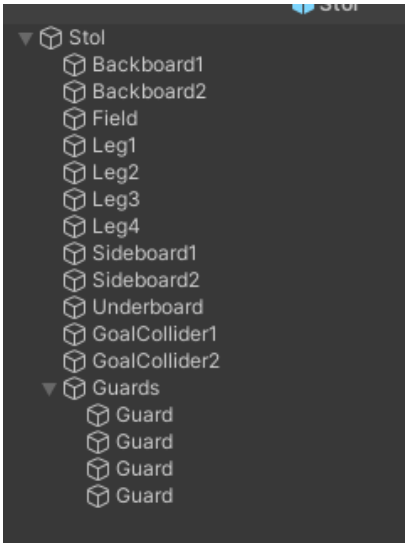
```
public void scoreTeam1()
{
      score1++;
      scoreText1.text = score1.ToString();
}
public void scoreTeam2()
{
      score2++;
      scoreText2.text = score2.ToString();
}
```

2.1.4. Predlošci Prefab

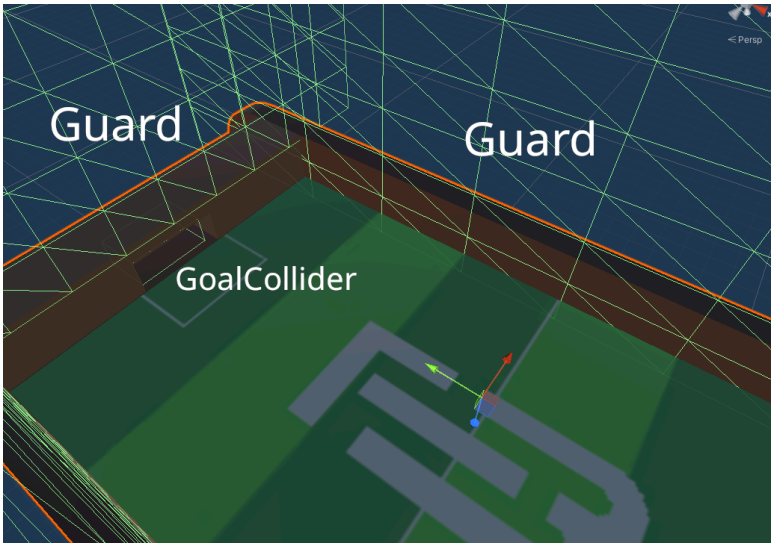
U razvojnom okruženju *Unity*, *prefab* je vrsta predloška koji se koristi za stvaranje jednog ili više sličnih objekata u igri. Ovo je korisno jer u njega možemo pohraniti sve što nam treba za jedan cjeloviti objekt kojeg možemo ponovno iskoristiti na više mjesta.

Prefab Stol modelira stol za stolni nogomet. Na slici 4. možemo vidjeti navedene njegove komponente: *Backboard1*, *Backboard2*, *Field*, *Leg1*, *Leg2*, *Leg3*, *Leg4*, *Sideboard1*, *Sideboard2* i *Underboard* koji su dijelovi fizičkog stola i svi imaju svoj *collider*. Komponente *GoalCollider1* i *GoalCollider2* služe za detekciju golova. Kada loptica prođe kroz jednog od njih, pokrenut će metodu *onTrigger* u skripti *BallHandler.cs* koja će očitati tag komponente *GoalCollider* te prema tome dodijeliti bodove timu koji je zabio gol. *Guard* objekti služe za zaštitu od prolaska loptice preko ruba stola. Vizualni prikaz navedenih komponenti vidi se na slici 5.

Sljedeći *prefab* je *Stap*. On modelira jednu od osam šipki kojima upravljaju igrači. Iako je dio prefaba samo šipka bez modela igrača, oni se automatski postavljaju na šipku pri početku igre. Skripta koja ih stvara - *StapHandler.cs* ima javni parameter koji kontrolira koliko će modela igrača biti na šipki. Postoje dva modela igrača, jedan za svaki tim.



Slika 4. Hijerarhija stola



Slika 5. *Prefab* Stol s vidljivim Guardom i GoallColliderom

2.2. Sustav upravljanja

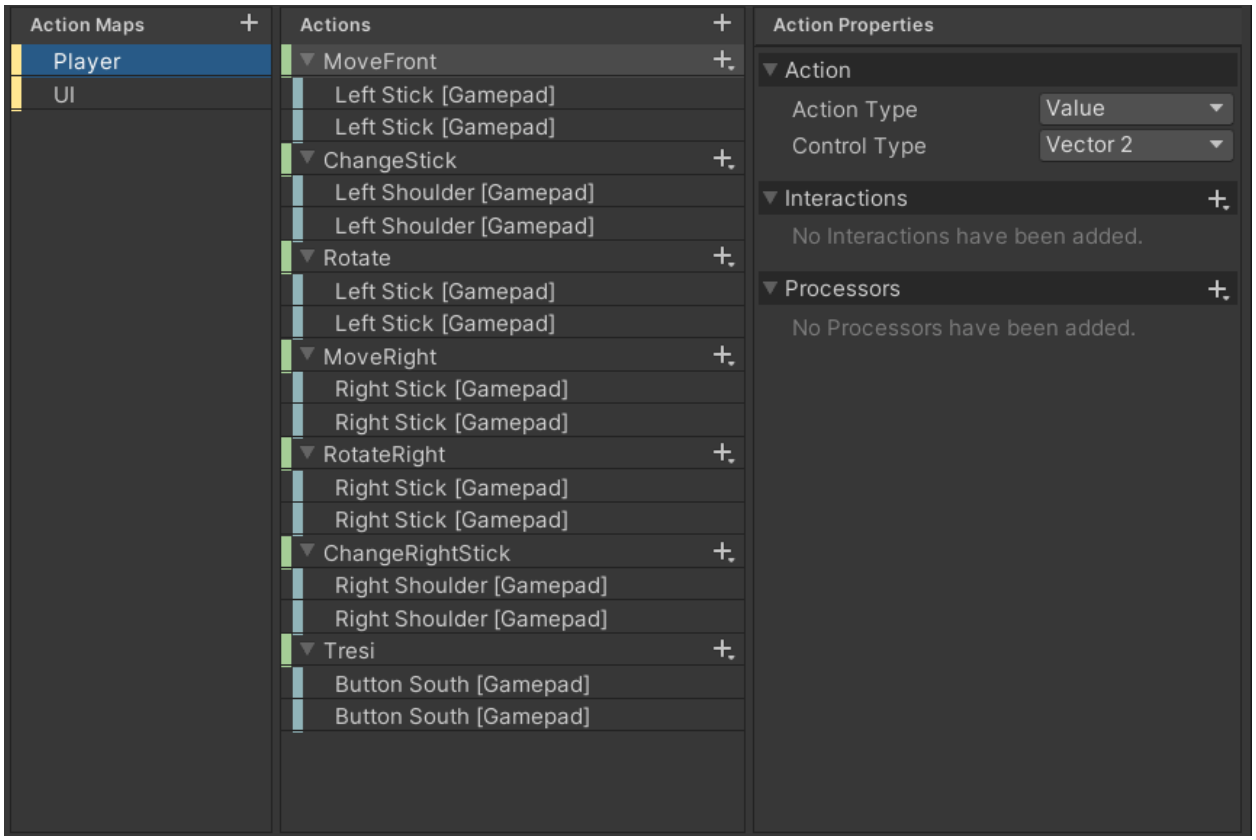
Za upravljanje igrom i njezinim sučeljem kao uređaj koristimo kontroler. Sustav upravljanja sastoji se od nekoliko povezanih dijelova iz novog *Unity Input System* sustava za upravljanje.

2.2.1. Akcijske mape

Prvo što moramo postaviti su akcije sučelja (*Input Actions*). One nam omogućuju izradu više akcijskih mapa, ovisno što je potrebno mapirati. To konkretno znači da možemo u njemu zadati više skupova kontrola ovisno o tome za što nam trebaju. Za kontroliranje igrača koristimo *Action Map Player*, dok za upravljanje *UI* elementima koristimo *Action Map UI*.

Nadalje, za svaku akcijsku mapu možemo zadati akciju (*Action*), koja će opisivati neku radnju, što ćemo moći iskoristiti dohvaćanjem u skripti za kretanje igrača.

Na slici 6. možemo vidjeti primjer akcijske mape *Player* s komponentama kretanja i rotiranja gdje imamo odvojene ulaze za dvije lijeve i dvije desne šipke. Vrsta upravljanja (*Control Type*), postavljena je na dvodimenzionalni vektor (*Vector2*) jer je to upravo ono što nam vraća ulazni signal s kontrolera.

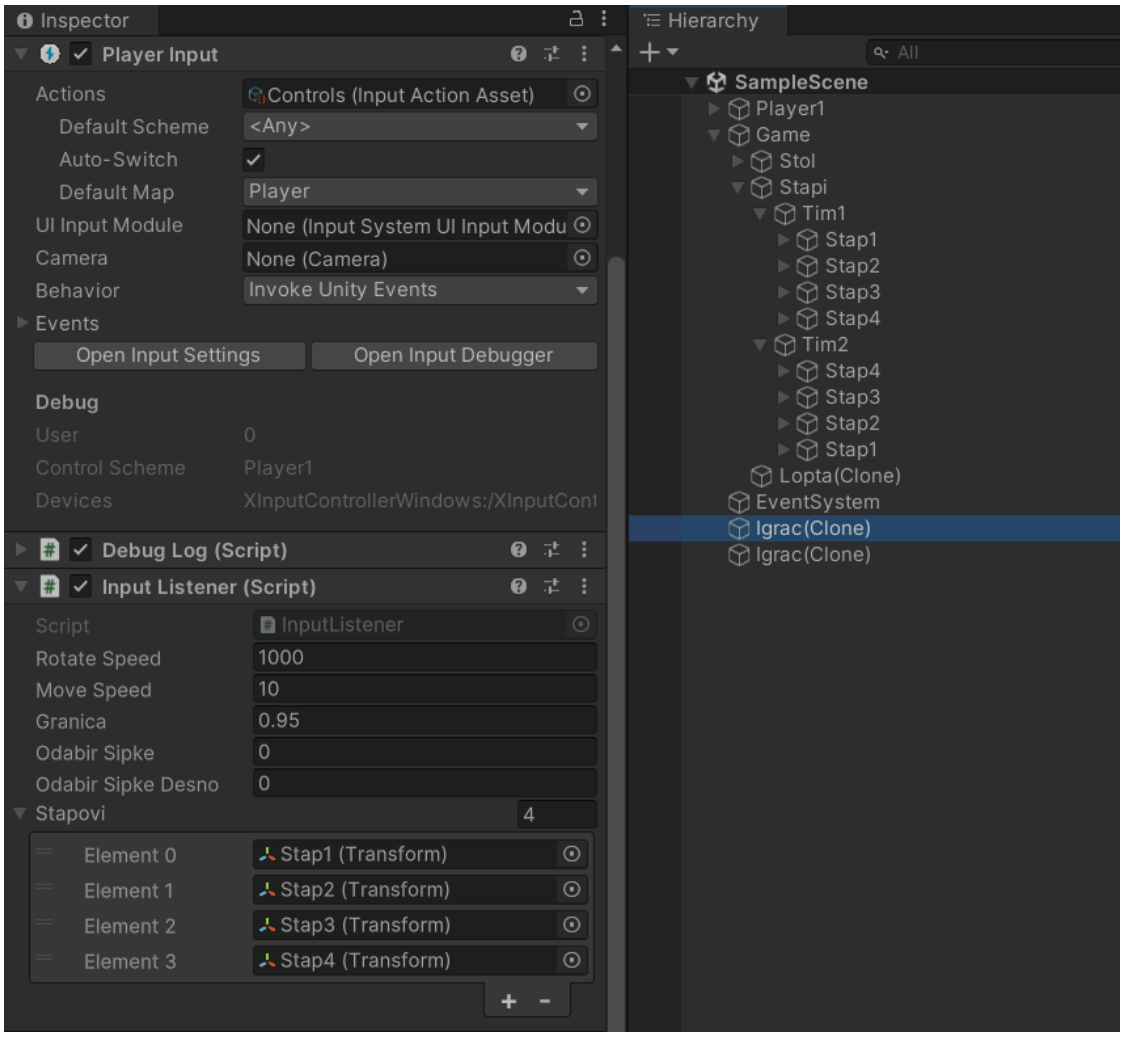


Slika 6. Akcijske mape

2.2.2. *Igračev unos*

Unity s novim sustavom za upravljanje unosa (*Input System*) pruža i dvije gotove skripte: *Player Input* i *Player Input Manager*. To radi na način da se izradi *prefab* figure kojom će svaki igrač upravljati, na njega se postavi oblik igračevog unosa (*Player Input*) i pomoćna skripta za upravljanje, te se gotovi *prefab* ubacuje u upravljač igračevog unosa (*Player Input Manager*), koji je postavljen na neku od roditeljskih komponenti (Slika 7.).

U našoj igri svaki igrač upravlja sa 4 šipke, pa nije bilo moguće u *Player Input Manager* staviti *prefab* šipke jer bi se tada za dva igrača stvorile samo dvije šipke, nego smo stvorili prazan game object kao *prefab* na kojeg smo stavili *Player Input* skriptu, te vlastite skripte u kojima smo pozivanjem *Player Input* skripte pozivali vlastite funkcije (*Unity Events*) na svim šipkama, gdje smo onda programski ispitivali upravlja li zadani kontroler *Tim1* ili *Tim2*.



Slika 7. Sustav unosa pri pokrenutoj igri i s inicijaliziranim kontrolama

2.2.3. *Skripta za upravljanje kontrolama*

InputListener (Slika 8.) je skripta koja služi za dohvaćanje šipki i *Player Input* komponente koja se nalazi na komponenti *Igrac*.

Skripta se sastoji od 3 dijela:

- inicijalizacije
- dohvaćanja *listenera*
- ažuriranje pozicije, tj. transformacija šipki

```
public class InputListener : MonoBehaviour
{
    private Controls controls;
    PlayerInput playerInput;

    Vector2 move;
    Vector2 moveRight;
    Vector2 rotate;
    Vector2 rotateRight;

    public float rotateSpeed = 1000f;
    public float moveSpeed = 10f;
    public float granica = 0.95f;

    //odabir lijeve ili desne sipke => 0 - lijeva, 1 - desna
    public int odabirSipke = 0;
    public int odabirSipkeDesno = 0;

    public Transform[] stapovi = null;
    GameController gameController;

    Unity Message | 0 references
    private void Start()
    {
        move = Vector2.zero;
        moveRight = Vector2.zero;
        rotate = Vector2.zero;
        rotateRight = Vector2.zero;

        gameController = FindObjectOfType<GameController>();

        playerInput = GetComponent<PlayerInput>();

        GameObject Tim = GameObject.Find(Gamepad.current.device.ToString() == gameController.kontroler1 ? "Tim1" : "Tim2");

        stapovi = new Transform[Tim.transform.childCount];

        for (int i = 0; i < Tim.transform.childCount; i++)
        {
            stapovi[i] = Tim.transform.GetChild(i);
        }
    }
```

Slika 8. Dio koda za inicijalizaciju *input listener* skripte

Zatim treba inicijalizirati varijable na početne vrijednosti, dohvatiti štapove i spremiti ih u listu (Slika 9.).

```
public void OnMoveFront(InputAction.CallbackContext context)
{
    move = context.ReadValue<Vector2>();
}

0 references
public void OnRotate(InputAction.CallbackContext context)
{
    rotate = context.ReadValue<Vector2>();
}

0 references
public void OnChangeStick(InputAction.CallbackContext context)
{
    OdabirIgraca();
}

0 references
public void OnMoveRight(InputAction.CallbackContext context)
{
    moveRight = context.ReadValue<Vector2>();
}
0 references
public void OnRotateRight(InputAction.CallbackContext context)
{
    rotateRight = context.ReadValue<Vector2>();
}
0 references
public void OnChangeRightStick(InputAction.CallbackContext context)
{
    OdabirIgracaDesno();
}

0 references
public void OnTresnja(InputAction.CallbackContext context)
{
    gameController.ShakeTable();
}
```

Slika 9. Dio koda za dohvaćanje *listenera*

Nakon što se dohvate *listeneri*, iz konteksta se uzimaju vrijednosti koje se prosljeđuju na zadane varijable iz kojih će se u sljedećem dijelu koda čitati vrijednosti i izvoditi transformacija nad šipkama (Slika 10.).

```
2 references
void MoveStap(Vector2 rotateStick, Vector2 moveStick, Transform transform)
{
    if (Mathf.Abs(transform.position.z) < granica)
    {
        Vector3 m = new Vector3(0, 0, moveStick.y) * moveSpeed * Time.deltaTime;
        transform.Translate(m, Space.World);
    }

    if (transform.position.z >= granica)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y, granica - 0.01f);
    }
    else if (transform.position.z <= -granica)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y, -granica + 0.01f);
    }

    Vector3 r = rotateSpeed * Time.deltaTime * new Vector3(0, 0, rotateStick.x);
    transform.Rotate(r, Space.World);
}

Unity Message | 0 references
private void Update()
{
    MoveStap(rotate, move, stapovi[odabirSipke]);

    MoveStap(rotateRight, moveRight, stapovi[odabirSipkeDesno + 2]);
}
```

Slika 10. Dio koda za transformaciju šipki

Sada koristimo funkciju *MoveStap* u koju prosljeđujemo učitane vrijednosti iz *listenera*, i jednu od šipki iz liste kojom želimo upravljati, te nad njom izvodimo transformaciju: *transform.Translate* ili *transform.Rotate*.

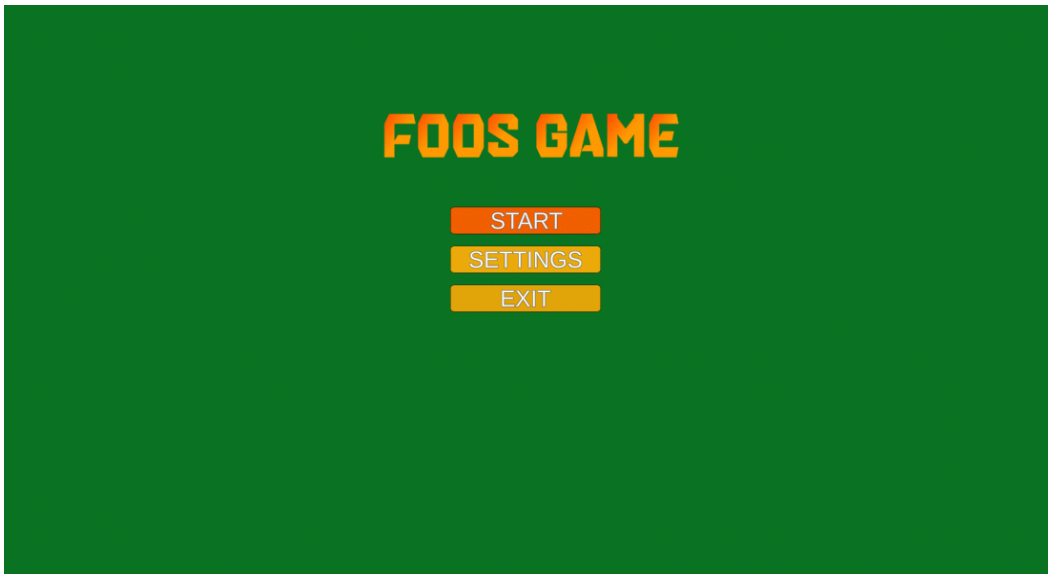
U funkciji *Update* pozivamo funkciju *MoveStap* s parametrima *rotate*, *move* i *stapovi[odabirSipke]*, čime upravljamo objektima *Stap1* i *Stap2*, a u funkciji *MoveStap* s parametrima *rotateRight*, *moveRight* i *stapovi[odabirSipkeDesno + 2]* upravljamo objektima *Stap3* i *Stap4*.

2.3. Grafičko sučelje

Grafičko sučelje igre provlači se kroz četiri već spomenute scene: naslovna scena (*MainScene*), scena s postavkama (*SettingsScene*), scena igre (*GameScene*) i završna scena (*EndScene*). U nastavku ovog poglavlja bit će objašnjena funkcionalnost sučelja vezana uz svaku scenu.

2.3.1. Naslovna scena

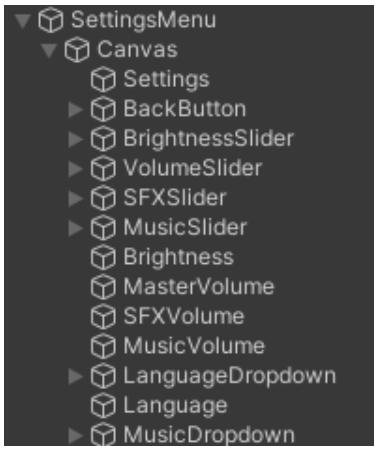
Grafičko sučelje prve scene sastoji se od naziva igre i jednostavnog skupa od tri gumba: *Start*, *Settings* i *Exit*. Pritiskom na gumb *Start* aktivna scena se mijenja u *GameScene*, te se pokreće igra. Pritiskom na gumb *Settings* otvara se *SettingsScene* u kojoj je moguće mijenjati neke od postavki igre o kojima će biti riječi u idućim potpoglavljima. Pritiskom na gumb *Exit* izlazi se iz igre.



Slika 11. Naslovna scena igre

2.3.2. *Scena s postavkama*

Grafičko sučelje scene s postavkama ima nešto složeniju strukturu, prikazanu sljedećim slikama.



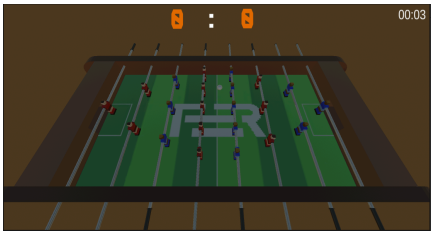
Slika 12. Hijerarhija scene

Slika 13. Grafičko korisničko sučelje scene s postavkama

Svaki od prikazanih klizača i padajućih izbornika povezan je s odgovarajućom jednostavnom metodom kojom se bilježi trenutna vrijednost kako bi se pomoću te vrijednosti moglo djelovati na navedene parametre postavki igre.

2.3.3. Scena igre

Grafičko sučelje ove scene sastoji se od prikaza trenutnog rezultata igre i mjerača vremena kojim se prikazuje vrijeme preostalo do kraja runde. Kako bi se prikazala vrijednost trenutnog rezultata potrebno je bilo povezati varijable *score1* i *score2* s odgovarajućim oznakama koristeći jednostavnu skriptu. Za mjerač vremena također je implementirana jednostavna skripta.



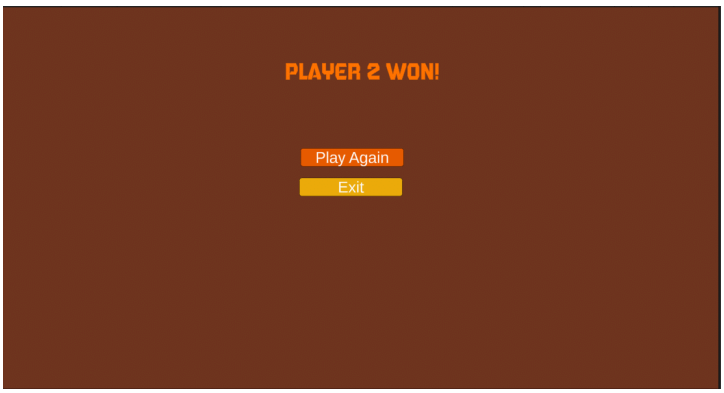
Slika 14. Grafičko korisničko sučelje scene igre u tijeku

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

Unity Script (1 asset reference) | 0 references
public class Timer : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI timerText;

    [SerializeField] float remainingTime;

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (remainingTime > 0)
        {
            remainingTime -= Time.deltaTime;
            int minutes = Mathf.FloorToInt(remainingTime / 60);
            int seconds = Mathf.FloorToInt(remainingTime % 60);
            timerText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
        }
        else
        {
            timerText.text = string.Format("{0:00}:{1:00}", 0, 0);
            SceneManager.LoadScene(3);

        }
```

Slika 15. Skripta za mjerač vremena

2.3.4. Završna scena

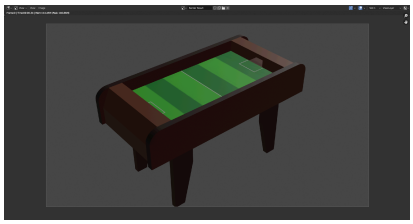
Najjednostavnija scena je *EndScene*, koja se pokreće pri isteku vremena igre, a sastoji se od oznake kojom se deklarira pobjednik (ako postoji), gumba za ponovno pokretanje igre, te gumba za izlazak iz igre.



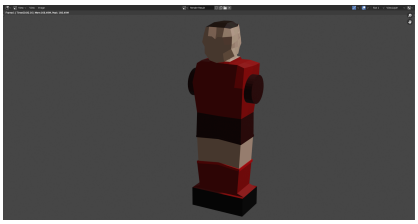
Slika 16. Prikaz scene završetka igre

2.4. Modeli

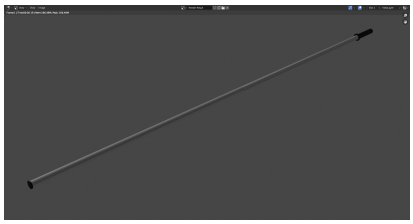
Postupak izrade modela potrebnih za igru, kao i samu skupinu 3D objekata može se podijeliti na tri dijela: stol, figura i šipka. Model stola zapravo nije ujedinjen u jedan objekt, nego se njegovim dijelovima može nezavisno manipulirati, čime se olakšava rješavanje problema logike i fizike same igre. Iz istog razloga, gradivni elementi figure su spojeni u jedinstveni objekt, kao i elementi šipke. Oblik i dimenzije svih modela rađeni su na temelju referentnih slika stvarnih predmeta.



Slika 17. Prikaz stola



Slika 18. Prikaz figure



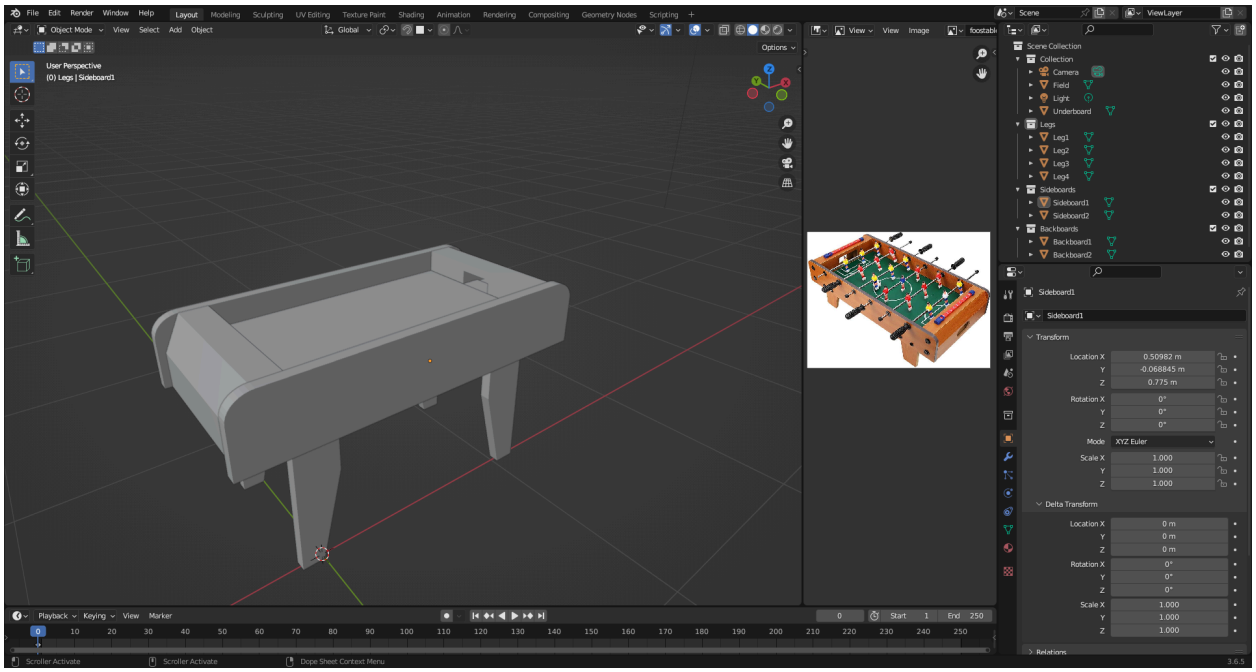
Slika 19. Prikaz šipke

2.4.1. Programska potpora

Glavni alat korišten za izradu 3D modela prisutnih unutar virtualnog okruženja jest Blender 3.6 - besplatan, ali moćan alat koji u sklopu svog sučelja pruža mogućnost stvaranja i oblikovanja modela, pridruživanja materijala i tekstura, te njegov prikaz u kontekstu osvjetljenja i kadriranja. Unatoč širokom opsegu usluga koje pruža ovaj program, nije pogodan za uređivanje dvodimenzionalnih slika pošto nije opremljen s osnovnim alatima za takvo što, pa je u tu svrhu korišten Adobe Photoshop CC 2019, popularni i iznimno svestran alat za stvaranje i uređivanje slika. Za potrebne instrukcije, rješenja i materijale za učenje korištenja Blenderu pristupilo se mrežnim stranicama YouTube, te StackExchange i službenoj Blender dokumentaciji.

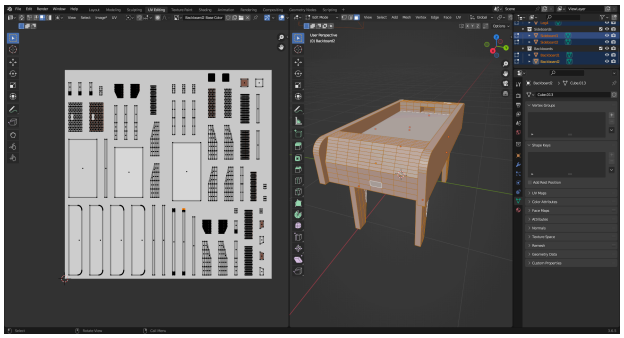
2.4.2. Izrada modela stola

Modelu svake komponente dan je osnovni oblik mijenjajući dimenzije kocke, koja je jedna od osnovnih objekata (“meshes”) koji se mogu dodati u radni prostor iz tome posvećenog menija unutar Blendera. Nakon toga su se uređivanjem točaka, bridova i ploha geometrijske mreže objekata, oblici modela doveli do razine relativno bliske referentnom stolu.

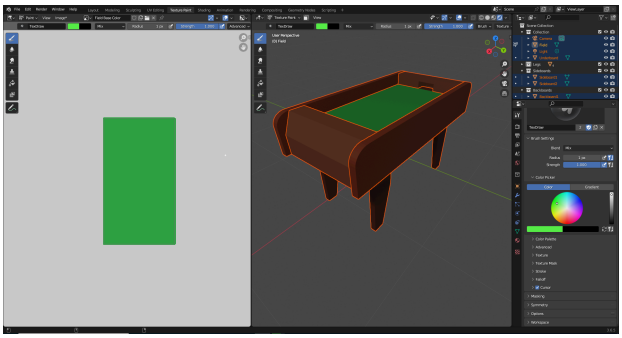


Slika 20. Raspored elemenata modela stola

Nakon toga je za objekt svakog elementa modela izvedena UV mapa i postavljena na sliku u .png formatu. Na taj način se unutar Blendera može ručnim bojanjem (kistom ili kanticom) stvoriti tekstura elementa čija UV mapa se boja. Elementi su obojani jednobojno, jednostavnim stilom čime se postiže nerealistični, nacrtani izgled prikladan za ovakvu aplikaciju. Što se tiče zelenog terena za igru, pruge u zelenim nijansama i bijele crte povučene su u Blenderu, a naknadno je njegova .png datoteka uvezena u Photoshop, zajedno s logom FER-a preuzetim s mreže koji je postavljen na sredinu terena.



Slika 21. UV mapa elemenata objekta

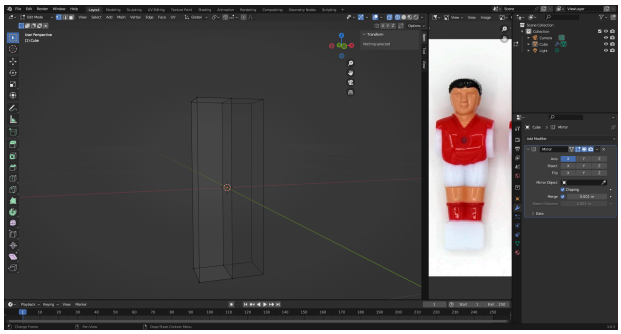


Slika 22. Bojanje teksture polja

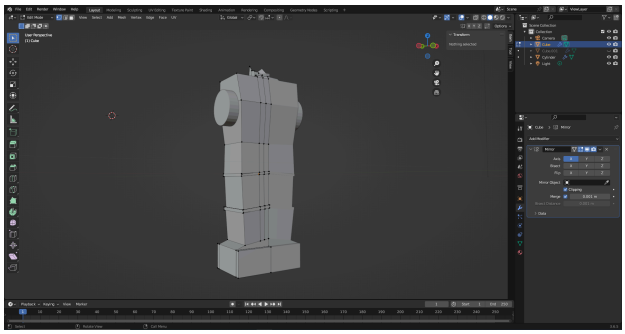
2.4.3. Izrada modela figure

Za razliku od postupka modeliranja stola, pri modeliranju figure, krajnji proizvod se sastoji od samo jednog objekta, odnosno svi elementi prisutni tijekom stvaranja modela na kraju su povezani u jednu nerazdvojnu cjelinu kojom je lako baratati u drugim programima.

Kao i kod izrade stola, krenulo se od osnovnog oblika - kvadra. Nakon postavljanja željene dimenzije primjenjuje se modifikator osnog zrcaljenja, kojim se olakšava posao jer se samo jedna strana figure mora ručno modelirati dok se druga automatski zrcali, čime je osna simetrija osigurana.

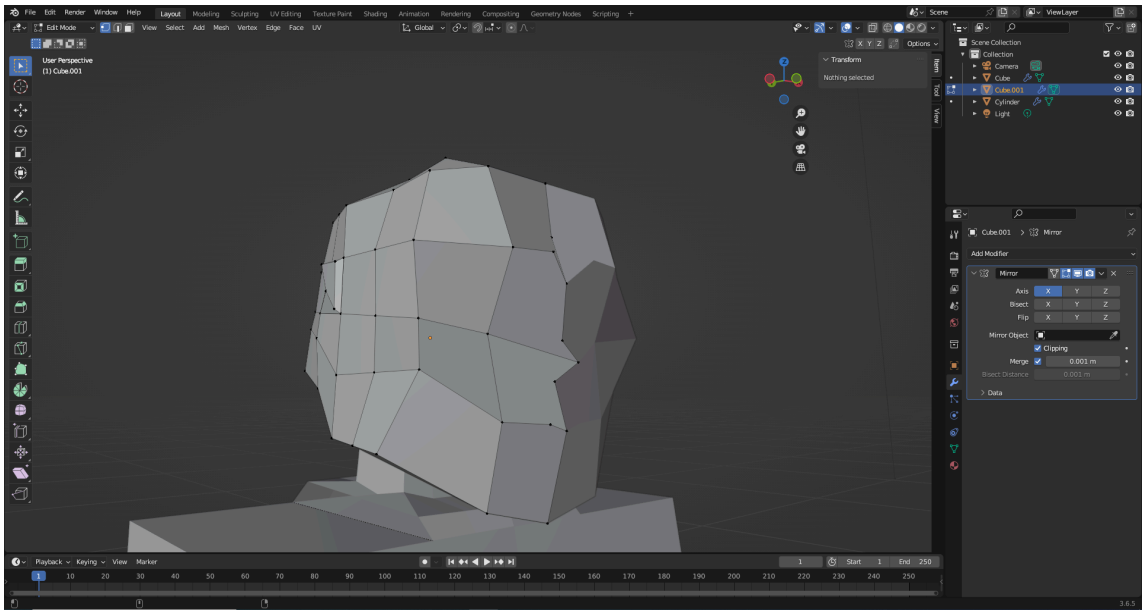


Slika 23. Zrcaljeni kvadar



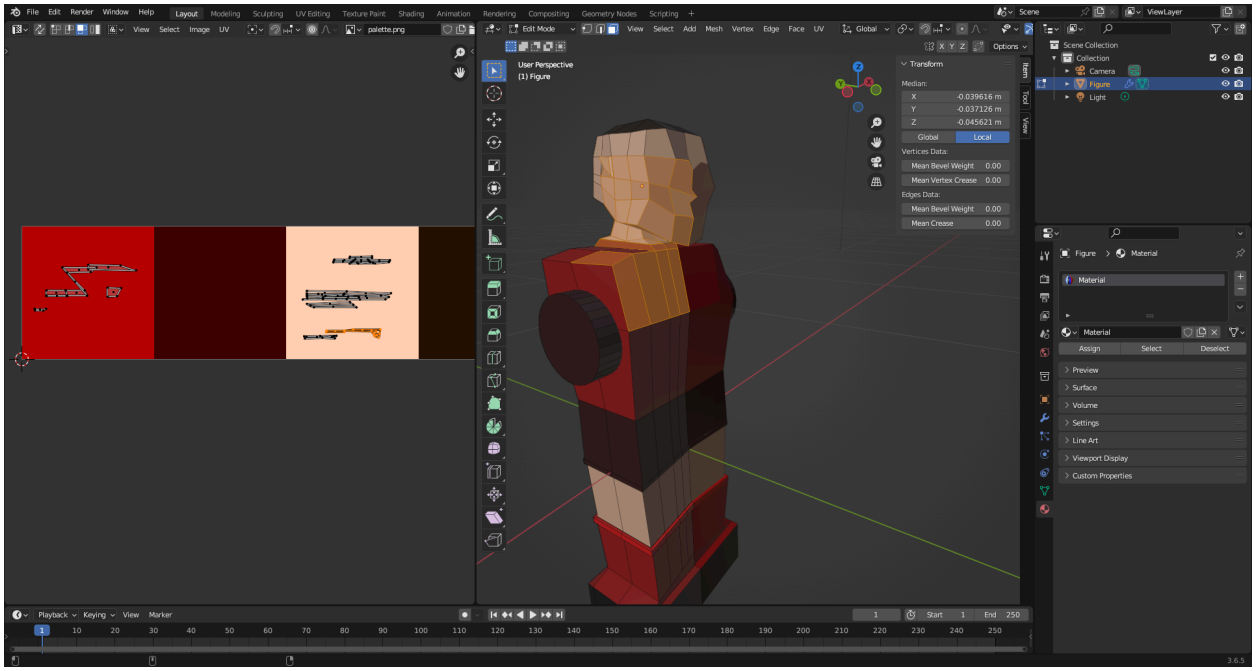
Slika 24. Oblik tijela figure

Nakon toga se posebno modelirala glava figurice za koju je bilo potrebno koristiti veći broj ploha u odnosu na tijelo kako bi se postigao željeni oblik.



Slika 25. Oblik modela glave

Glava i tijelo povezali su se u jednu cjelinu i zatim kreće bojanje. Primijenjen je temeljni materijal čiju teksturu je tada potrebno ručno ispuniti. Odabrana metoda bila je da se odabere paleta koja će se koristiti za objekt i stvori se slika gdje su pikseli odabranih boja poredani jedan za drugim. Zatim se UV mapa objekta odmota na tu sliku i 2D preslike željenih ploha se mišem pomaknu na željeni piksel na slici. Ova metoda je prikladna za simplističke objekte za koje se želi postići izgled igračke ili upravo figurice, jer postoje oštri prijelazi između boja. Treba pripaziti da se unutar Blendera i Unityja ugasi automatska interpolacija. Također, kada se 3D model želi izvesti iz Blendera u .fbx formatu, potrebno je vanjske datoteke kao što su slike koje služe kao teksture eksplicitno upakirati zajedno sa samim modelom.



Slika 26. Postupak bojanja teksture figure

2.4.4. *Izrada modela šipke*

Postupak izrade šipke jednak je onome figure, no radi se o jednostavnijem krajnjem obliku, odnosno iz početnog valjka su definirani ručka i čep.

3. Upute za korištenje

3.1. Instalacija i pokretanje

Dovoljno je pokrenuti *FoosGame.exe* pokretačku datoteku. Nije potrebna dodatna instalacija. Pokretačku datoteku moguće je preuzeti na Github repozitoriju projekta: https://github.com/spinzed/foos-game

3.2. Postavke

Kada se pokrene igra, moguće je otići u postavke igre pritiskom na gumb *Settings*. Tamo je moguće postaviti postavke zvuka, glazbe tijekom igre, osvjetljenja i slično.

3.3. Upravljanje

Foos-Game je namijenjen za igru pomoću kontrolera. Šipke se kontroliraju pomoću analognih palica. Lijevom analognom palicom kontroliraju se dvije lijeve šipke, a desnom dvije desne.

Svaka palica može upravljati samo jednom od te dvije šipke, stoga postoje odvojeni gumbi kojima se odabire koja od dviju šipki lijeve odnosno desne palice su aktivirane za kontrolu u danom trenutku. Pritiskom *LT (L1)* tipke na kontroleru lijeva palica mijenja koju šipku kontrolira, a pritiskom na *RT (R1)* desna.

Pritiskom na gumb *A*, stol će se zatresti, sa svrhom izbacivanja loptice iz pozicije u kojoj ju ne može dotaknuti nijedna figura.

3.4. Minimalna konfiguracija

Procesor: x86-64 procesor sa skupom instrukcija SSE2

Radna memorija: 1 GB

Grafička kartica: s podrškom za DX10, DX11, DX12, OpenGL 3.2, Vulkan ili Metal

Diskovni prostor: 300 MB

4. Literatura

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

[2] StackExchange: https://blender.stackexchange.com

[3] Blender docs: https://docs.blender.org

[4] "Blender Guru", "Blender 3.0 Beginner Donut Tutorial (OLD)", "03.12.2021.": https://www.youtube.com/playlist?list=PLjEaoINr3zgFX8ZsChQVQsuDSjEqdWMAD

[5] "PIXXO 3D", "Tutorial: Blender MODELLING For Absolute Beginners | Low Poly Girl", "May 27, 2022": https://www.youtube.com/watch?v=sbCW0Cs7aI8&t=2428s

[6] "Buvesa Game Development", "How To Texture UV Colors In Blender (like Imphenzia)", "10.01.2022.": https://www.youtube.com/watch?v=8NEmx0cHwoI&list=LL&index=7