Simulacija minobacača

Seminarski rad iz kolegija "Interaktivni simulacijski sustavi"

Renato Majer, Dalen Grdić, David Kovačević, Mario Petek
20.01.2023.

Djelovođa: izv. Prof. dr. sc. Siniša Popović

***Sažetak* – Kroz interaktivnu simulaciju demonstrirana je okolina u kojoj vojnici sa stacionarne pozicije pomoću minobacača gađaju neprijateljske tenkove koji pristižu sa različitih strana i nalaze se na različitim udaljenostima od samih vojnika. Vojnici mogu okretati minobacač prema lijevo i prema desno te isto tako mijenjati kut pod kojim se ispaljuju mine. S druge strane, tenkovi ispaljuju projektile prema vojnicima, pa je vojnicima u cilju čim prije uništiti neprijateljske tenkove.**

1 Uvod

Vojne simulacije su jako bitne za uvježbavanje vojnih taktika ili provođenje vojnih vježbi bez stvarne opasnosti za njihove sudionike uz značajno smanjene troškove u odnosu na trošak stvarnog provođenja iste.

Okolina ove vojne simulacije je utvrđena pozicija vojnika s minobacačem s pogledom na otvoreno polje okruženo drvećem i raslinjem gdje se očekuje nailazak neprijateljskih tenkova. Tenkovi nailaze s obje strane i dobivaju položaj utvrđenih vojnika nakon čega se okreću prema poziciji minobacača i ulaze u konflikt.

Cilj simulacije je uvježbati korištenje minobacača u mogućoj stvarnoj situaciji gdje je njihova pozicija kompromitirana. Zadatak vojnika je uništiti neprijatelja korištenjem minobacača prije nego neprijatelj pogodi njihovu utvrđenu poziciju. Minobacač se može okretati strelicama lijevo i desno, kut gađanja strelicama gore i dolje, a mina se ispaljuje lijevim klikom miša.

Zanimljive značajke ove simulacije su modularnost same simulacije. Stvaranje i pucanje tenkova može se ostvariti ručno na stisak pripadajuće tipke ili kontinuirano čime se može kontrolirati opterećenje i opasnost za vojnike gdje je kontinuirano stvaranje i pucanje tenkova određeno nasumičnim vremenom. Dodatno, pucanje je ostvareno nasumičnom preciznošću kako bi simulirali stresnu situaciju bliskog pogotka u utvrđenu poziciju. Simulacija se dodatno može ponovno pokrenuti po potrebi, a sve mogućnosti prikazane su u lijevom uglu ekrana.

2 Uloge pojedinih članova tima

U sveukupnim poslovima na izradi seminarskog rada te pisanju ovog izvješća, članovi tima sudjelovali su na sljedeći način:

- Renato Majer – organizacija i podjela poslova, plan i način implementacije projekta, upravljanje repozitorijem, stvaranje scene i prizora, programiranje logike ciljanja i kretanja minobacača, pisanje dokumentacije (Sažetak, 1 – Uvod, 2 – Uloge pojedinih članova tima)
- Dalen Grdić – stvaranje prizora, pisanje dokumentacije (3 – Razvojno okruženje), modeliranje komponenti neprijatelja, programiranje logike kretanja i pucanja neprijatelja
- David Kovačević – programiranje logike stvaranja scene i neprijatelja, modeliranje komponenti minobacača, pisanje dokumentacije (4 – Programski kod, 5 – Pokretanje simulacije, 6 – Zaključak)
- Mario Petek – programiranje kretanja kamere u prizoru, modeliranje komponenti minobacača, pisanje dokumentacije (4 – Programski kod, 5 – Pokretanje simulacije, 6 – Zaključak)

3 Razvojno okruženje

3.1 Unity

Implementacija simulacije ostvarena je u programu Unity, popularnom razvojnom okruženju za razvoj video igara baziran na programskom jeziku C#. Unity omogućuje stvaranje fizike, 3D renderiranje, detekciju sudara i druge stvari koje našu simulaciju čine funkcionalnom. Alat je pogodan jer omogućuje njihovo direktno korištenje bez potrebe za programiranjem čime je fokus prebačen na samu implementaciju.

Unity ima vizualni uređivač kojim u par klikova možemo stvoriti osnovne objekte, urediti, ali i upravljati njihovim svojstvima te promatrati ponašanje u prostoru. Programski jezik zaslužan za rukovanje kodom, logikom i drugim klasama jedinstvenih za Unity je C#. Sve navedeno omogućuje programerima raznih iskustva i vještina lakoću rukovanja ovim alatom. Unity također ima vlastiti forum i „***Assets store***“, mjesto gdje programeri prenose svoje kreacije i čine ih dostupnim široj zajednici zbog čega je izabran za izradu ove simulacije.

3.2 Glavni elementi

Unity svoje prizore naziva **scene**. To je prizor na kojem vidimo izgled naše trenutno ostvarene okoline. Scena sadrži **objekte** koje želimo da se u njoj nalaze i upravljaju tokom njenog izvođenja, a objekte je moguće razmještati. Svakom objektu i njegovim svojstvima može se pristupiti ugrađenim **inspektorom** koji nam daje ažurne informacije o objektu u prizoru. **Inspektor** je odličan alat kojim možemo i promijeniti svojstva i komponente objekta za vrijeme stvarnog izvođenja prizora.

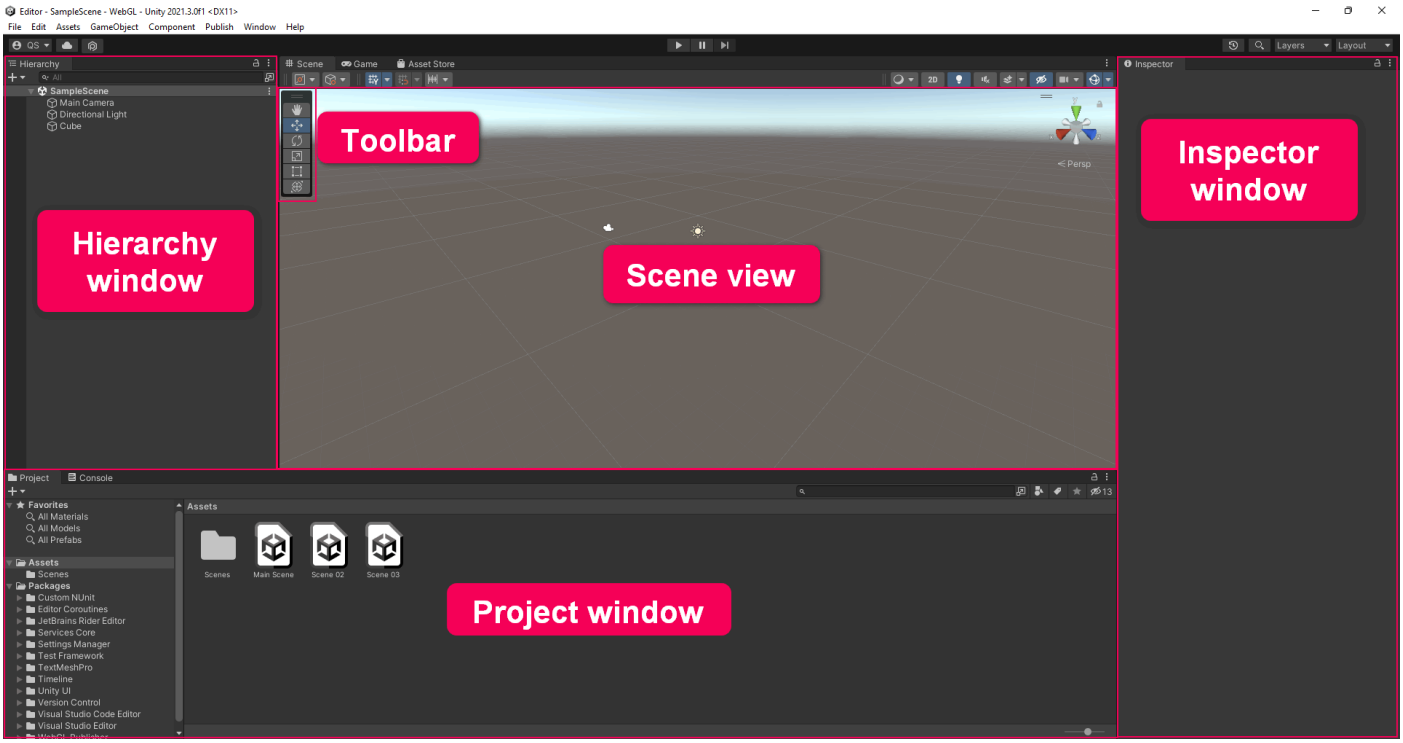
Scena također sadrži **kameru**, objekt koji koristimo da prikažemo scenu ili prizor korisniku iz određene pozicije ili kuta gledanja i daje korisniku oči u simulaciji. Također je u scenu važno uključiti izvor svjetlosti koji služi za osvjetljavanje scene i objekte svjetlošću iz tog smjera.

Prozor **Game** prikazuje simulaciju kroz objekt **kamere** koji smo postavili u sceni. Korisnik ovaj prozor koristi za interakciju i gledanje u simulaciji za vrijeme njenog izvođenja. Prozor **Scene** prikazuje prizor iz perspektive prostora simulacije i omogućuje nam da se slobodno krećemo po prizoru i stvaramo promjene koje postaju vidljive iz perspektive korisnika za vrijeme njenog izvođenja.

3.3 Objekti i modeli

Prizor scene bez objekta je ekran bez prikaza što znači da i sama kamera ima ulogu objekta u sceni što čini objekte nezaobilaznim dijelom u procesu implementacije. Kada objekt ili grupa objekata ima nama semantičko značenje ono nazivamo model. Model nam omogućuje jednostavniju implementaciju ponašanja i manipulaciju objekata u sceni grupiranjem objekata u logičku skupinu.

Da bi mogli koristiti objekte moramo imati način da njima manipuliramo. Zato svaki objekt sadrži komponente. Svaki objekt mora imati komponentu transformacije[1]. Komponenta transformacije nam je važna jer sadrži podatke o poziciji, rotaciji i skaliranju objekta i omogućuje manipuliranje položaja i rotacije objekta, a opcionalno može imati i komponente važne za ovu simulaciju poput skripti[2] čije instance povezujemo s objektima i koristimo za pisanje programskog koda za određeni objekt, krutih tijela[3] koje nam služe da naše tijelo zauzima prostor i koristi fiziku tijela, sudarala[4] koje prate sudare između dva kruta tijela u simulaciji, zvučne izvore[5] za zvučne podražaje u simulaciji, animatore[6] za vizualne podražaje u simulaciji, ali i mnoge druge.



Slika 1. Grafičko sučelje alata Unity. Preuzeto iz [7]

4 Programski kod

Da bi upravljali simulacijom, određeni objekti moraju biti programirani. Objekti znaju svoj položaj i oblik u prizoru, ali ne znaju se kretati ili pozicionirati za vrijeme izvođenja bez postojanja programskog koda koji upravlja njihovim svojstvima. Zbog toga su objekti kojima želimo manipulirati i mijenjati njihova svojstva tijekom izvođenja upravljani skriptama pisanim u programskom jeziku C#.

4.1 MonoBehaviour[8]

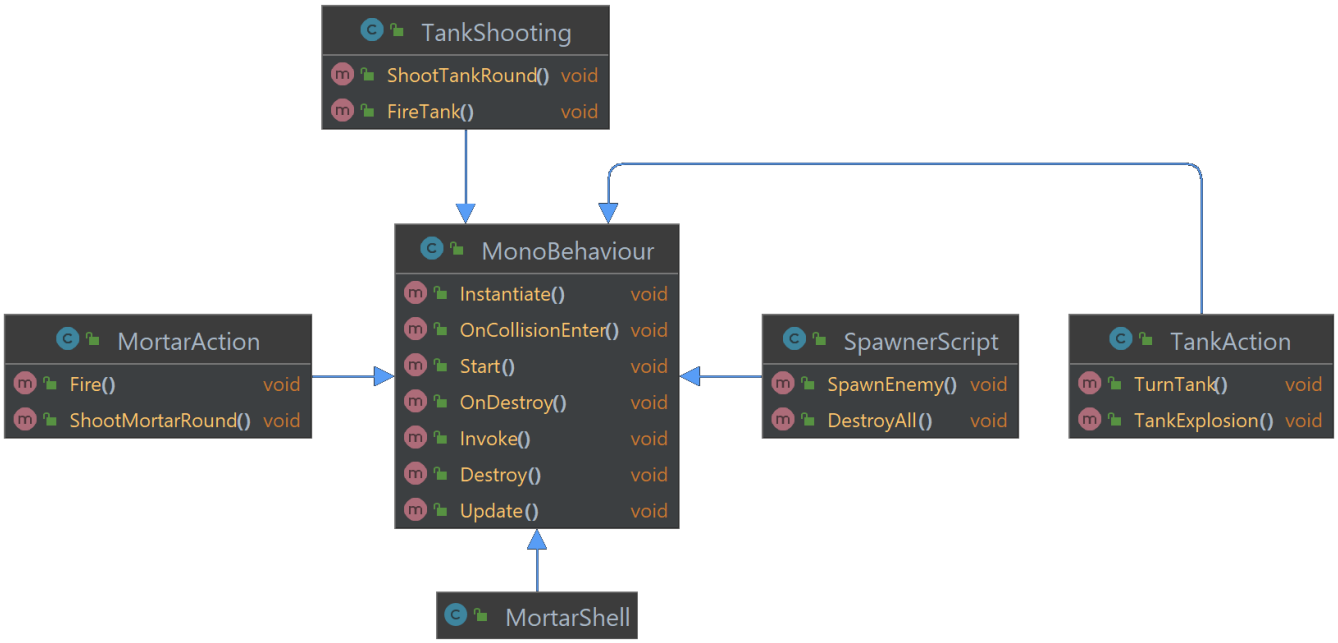
Sve klase ove simulacije nasljeđuju baznu klasu **MonoBehaviour** za implementaciju posebnih funkcija.

Funkcija **Start** se poziva na početku pokretanja skripte i koristi se za inicijalizaciju varijabli potrebne za ispravno izvođenje ostalih funkcija.

Update je korisna funkcija koja se poziva nakon svake prikazane sličice i najčešće se koristi za implementaciju kretanja objekta u prizoru.

Funkcija **Instantiate** i **Destroy** koriste se za stvaranje i uništavanje objekata u prizoru i najčešće su implementirane kroz korištenje funkcije **OnCollisionEnter** kojom pratimo sudare različitih objekata u simulaciji.

Osim ovih funkcija koristi se i **Invoke** koja omogućuje pozivanje funkcija nakon određenog vremenskog perioda što nam daje opciju za tempiranje određenih događaja u prizoru.



Slika 2. UML dijagram klasa opisane u ovoj dokumentaciji, kreirano pomoću alata IntelliJ IDEA 2022.2.3 [9]

4.2 MortarShell

Ovom klasom modeliramo let mine iz minobacača. Mina ima komponentu **krutog tijela** kojom možemo upravljati masom, ali i površinskim i kutnim **otporom** što omogućuje realističan **kosi hitac** minobacačem. Također sadrži komponentu **sudarača** koji na sudar s terenom stvara eksploziju s kraterom na mjestu udara ili običnu eksploziju u slučaju drugih objekata. Ukoliko se nađe ispod razine terena, objekt mine će se uništiti funkcijom **Destroy**.

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

public class MortarShell : MonoBehaviour {

    private Rigidbody mortarRigidbody;
    public GameObject explosionPrefab;
    public GameObject explosionNoCrater;

    void Start () {
        mortarRigidbody = GetComponent<Rigidbody>();
    }

    void Update () {
        this.transform.forward = Vector3.Slerp(this.transform.forward, mortarRigidbody.velocity.normalized, Time.deltaTime);
        if(transform.position.y < -1f) {
            Destroy(gameObject);
        }
    }

    private void OnCollisionEnter(Collision collision) {
        Instantiate(explosionPrefab, new Vector3(this.transform.position.x, 0.20f, this.transform.position.z), Quaternion.identity);
        GameObject player = GameObject.FindWithTag("Player");

        if(player != null)
            if(player.GetComponent<CrewCollider>().canDie)
                if(Vector3.Distance(gameObject.transform.position, player.transform.position) <= 5f)
                    Destroy(player);

        Destroy(gameObject);
    }
}

```

Slika 3. Programski kod klase MortarShell

4.3 MortarAction

Da bi mogli izbaciti minu iz minobacača, moramo implementirati funkcije koje će stvoriti minu i izbaciti je u smjeru gađanja. Ova klasa koristi metodu **Fire** koja se poziva iz vanjske skripte na stisak gumba za pucanje, a koja stvara vizualni efekt izbacivanja mine, a zatim funkciju **FireMortarRound** koja stvara objekt mine te se na objekt mine primjenjuje sila izbačaja.

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

public class MortarAction : MonoBehaviour {

    private AudioSource myAudio;
    public ParticleSystem[] gunFx;
    public AudioClip fireSound;
    public Rigidbody mortarPrefab;
    public Transform mortarBarrelEnd;

    // shell speed
    private float shellVelocity = 22000f;

    // Use this for initialization
    void Start() {
        myAudio = GetComponent<AudioSource>();
    }

    public void Fire() {
        myAudio.clip = fireSound;
        myAudio.loop = false;
        myAudio.Play();
        foreach (ParticleSystem fire in gunFx) {
            fire.Play();
        }

        ShootMortarRound();
    }

    public void ShootMortarRound() {
        mortarBarrelEnd.rotation = mortarBarrelEnd.rotation;
        Rigidbody emptyShellInstance = Instantiate(mortarPrefab, mortarBarrelEnd.position, (mortarBarrelEnd.rotation)) as Rigidbody;
        emptyShellInstance.AddForce(mortarBarrelEnd.forward * shellVelocity);
    }
}
```

Slika 4. Programski kod klase MortarAction

4.4 TankAction

Slično klasi **MortarAction** ovom klasom definirana je jednostavna funkcionalnost kretanja neprijatelja. Neprijatelji se kreće prema sredini, a zatim prema utvrđenoj poziciji. Neprijatelj staje na nasumičnoj poziciji gdje predstavlja opasnost vojnicima.

Kod klase je u dodatku.

4.5 TankShooting

Ova klasa implementira logiku ispaljivanja projektila u nasumičnom smjeru prema vojnicima kako bi simulirali promašaje. Dodatno, projektili se mogu i ručno ispaljivati na stisak gumba, ali i podesiti da pogodak projektila u vojnike završava simulaciju.

Kod klase je u dodatku.

4.6 SpawnerScript

Da bi upravljali tokom simulacije potrebno je implementirati stvaranje neprijatelja, utvrđene pozicije i slično. Zato ova skripta omogućuje ručno ili periodičko stvaranje neprijatelja kako bi kontrolirali opterećenje i stres nad vojnicima i resetiranje prizora na stisak tipke.

Kod klase je u dodatku.

5 Pokretanje simulacije

Simulacija se pokreće dvostrukim klikom miša na izvršnu datoteku **„MortarSimulation.exe“** koja se nalazi u projektnoj mapi **Build** čime započinje simulacija.

6 Zaključak

Cilj simulacije je postaviti vojnike u stresnu situaciju koja bi ih potencijalno mogli zateći. Osim toga, za vojnike je važno da razviju motoriku upravljanja minobacačem kako bi bili što spremniji u slučaju stvarnog konflikta. Iako današnje 3D simulacije mogu prikazati izrazito realistične situacije, da bi postigli puno bolje efekte, ova simulacija trebala bi se implementirati za virtualnu realnost, gdje će simulacija biti stvarnija i uzrokovati realniji stres. Virtualna realnost bi korištenjem raznih senzora pokreta tijela, ponajviše ruku te uz stvarni model minobacača s nekim oblikom trzaja realnije stimulirala stres nad vojnicima čime bi postigli bolji efekt.

7 Literatura

[1] Unity Technologies. (23. prosinca 2022.) *Transform* [https://docs.unity3d.com/ScriptReference/Transform.html] Datum pristupa: 7. prosinca 2023.

[2] Unity Technologies. (19. ožujka 2018.) *Creating and Using Scripts* [https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html] Datum pristupa: 7. prosinca 2023.

[3] Unity Technologies. (23. prosinca 2022.) *Rigidbody* [https://docs.unity3d.com/ScriptReference/Rigidbody.html] Datum pristupa: 7. prosinca 2023.

[4] Unity Technologies. (23. prosinca 2022.) *BoxCollider* [https://docs.unity3d.com/ScriptReference/BoxCollider.html] Datum pristupa: 7. prosinca 2023.

[5] Unity Technologies. (23. prosinca 2022.) *AudioSource* [https://docs.unity3d.com/ScriptReference/AudioSource.html] Datum pristupa: 7. prosinca 2023.

[6] Unity Technologies. (23. prosinca 2022.) *Animator* [https://docs.unity3d.com/ScriptReference/Animator.html] Datum pristupa: 7. prosinca 2023.

[7] Unity Technologies. (11. kolovoza 2022.) *Explore the Unity Editor* [https://learn.unity.com/tutorial/explore-the-unity-editor-1#6273f00fedbc2a7f158cc1ee] Datum pristupa: 7. prosinca 2023.

[8] Unity Technologies. (23. prosinca 2022.) *MonoBehaviour* [https://docs.unity3d.com/ScriptReference/MonoBehaviour.html] Datum pristupa: 7. prosinca 2023.

[9] JetBrains, s.r.o. (5. listopada 2022.) IntelliJ 2022.2.3 [https://www.jetbrains.com/idea/download/other.html] Datum pristupa: 8. siječnja 2023.

8 Dodatak

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

public class TankAction : MonoBehaviour {

    public Rigidbody tankBody;
    public GameObject explosionPrefab;
    private TankShooting tankShootingScript;
    private float moveAfterRotation = 20f;
    private float moveBeforeRotation = 120.0f;
    private float rotationSpeed = 3.0f;
    private float speed = 3.0f;
    private GameObject crew;

    void Start () {
        tankBody = GetComponent<Rigidbody>();
        moveBeforeRotation += Random.Range(-30f, 0f);
        moveAfterRotation += Random.Range(0f, 30f);
        tankShootingScript = this.GetComponentInChildren<TankShooting>();
        crew = GameObject.FindWithTag("Player");
    }

    void Update() {
        if(moveBeforeRotation > 0f) {
            tankBody.velocity = transform.forward * speed;
            moveBeforeRotation -= tankBody.velocity.magnitude * Time.deltaTime;
            return;
        }

        TurnTank();

        if(moveAfterRotation > 0f) {
            tankBody.velocity = transform.forward * speed;
            moveAfterRotation -= tankBody.velocity.magnitude * Time.deltaTime;
            return;
        }

        if(moveAfterRotation <= 0f) {
            GameObject crew = GameObject.FindWithTag("Player");
            if(crew == null) {
                tankShootingScript.canShoot = false;
            } else {
                tankShootingScript.tankBarrelEnd.transform.LookAt(crew.transform);
                tankShootingScript.canShoot = true;
            }
        }
    }

    void TurnTank() {
        if(crew == null) {
            return;
        } else {
            Vector3 lookDirection = (crew.transform.position - transform.position).normalized;

            Quaternion lookRotation = Quaternion.LookRotation(lookDirection);
            transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * rotationSpeed);
        }
    }

    void TankExplosion() {
        Instantiate(explosionPrefab, this.transform.position, Quaternion.identity);
        Destroy(gameObject);
    }

    void OnCollisionEnter(Collision collision) {
        if(collision.gameObject.CompareTag("Shell")) { // invoke tank explosion
            Debug.Log("Hit");
            Invoke("TankExplosion", 1.0f);
        } else if(collision.gameObject.CompareTag("Enviroment")) { // destroy enviroment objects
            Destroy(collision.gameObject);
        }
    }
}
```

Slika 5. Programski kod klase TankAction

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

public class TankShooting : MonoBehaviour {

    private AudioSource myTankAudio;
    public ParticleSystem[] gunFx;
    public AudioClip fireSound;
    public Rigidbody tankRoundPrefab;
    public Transform tankBarrelEnd;
    private Quaternion originalBarrelEnd;
    private SpawnerScript spawnerScript;
    private GameObject crew;
    public bool canShoot = false;
    private float cooldown;
    private float azmuthSlop = 3f;
    private float cooldownRate = 5f;

    // shell speed
    private float shellVelocity = 1000000f;

    // get audio component at start
    void Start() {
        spawnerScript = GameObject.FindWithTag("GameController").GetComponent<SpawnerScript>();
        myTankAudio = GetComponent<AudioSource>();
        crew = GameObject.FindWithTag("Player");
        cooldown = cooldownRate + Random.Range(-2f, 2f);
    }

    void Update() {

        originalBarrelEnd = tankBarrelEnd.rotation;

        cooldown -= Time.deltaTime;

        if((Input.GetKeyDown(KeyCode.T) || spawnerScript.autoShooting) && cooldown < 0f && canShoot) {
            FireTank();
            cooldown = cooldownRate + Random.Range(-2f, 2f);
        }
    }

    public void FireTank() {
        myTankAudio.clip = fireSound;
        myTankAudio.loop = false;
        myTankAudio.Play();
        foreach (ParticleSystem fire in gunFx) {
            fire.Play();
        }

        ShootTankRound();
    }

    public void ShootTankRound() {
        Quaternion slop = Quaternion.Euler(Random.Range(-azmuthSlop, azmuthSlop), Random.Range(-azmuthSlop, azmuthSlop), 0);

        Transform tmp = tankBarrelEnd;
        if(crew != null) {
            tmp.LookAt(crew.transform);
        } else {
            tmp = tankBarrelEnd;
        }

        tmp.rotation = tankBarrelEnd.rotation * slop;
        Rigidbody emptyShellInstance = Instantiate(tankRoundPrefab, tmp.position, (tmp.rotation)) as Rigidbody;
        emptyShellInstance.AddForce(tankBarrelEnd.forward * shellVelocity);
    }
}
```

Slika 6. Programski kod klase TankShooting

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

public class SpawnerScript : MonoBehaviour {

    public GameObject tankPrefab;
    public GameObject mortarCrewPrefab;
    private GameObject left;
    private GameObject right;
    private GameObject mortarCrew;
    private Vector3 spawnPosition = new Vector3(0, 0, -100);

    private float cooldownTime = 10f;
    private float spawnRate;
    private bool spawnContinuously = false;
    private float spawnDistance = 120f;
    public bool autoShooting = false;

    void Start() {
        spawnRate = -cooldownTime;
        mortarCrew = Instantiate(mortarCrewPrefab, spawnPosition, Quaternion.identity);
    }

    void Update() {
        spawnRate -= Time.deltaTime;

        if(Input.GetKeyDown(KeyCode.C)) spawnContinuously = true;
        else if(Input.GetKeyDown(KeyCode.S)) spawnContinuously = false;
        else if(Input.GetKeyDown(KeyCode.A)) autoShooting = true;
        else if(Input.GetKeyDown(KeyCode.M)) autoShooting = false;
        else if((Input.GetKey("space") || spawnContinuously) && spawnRate < 0f) SpawnEnemy();
        else if(Input.GetKeyDown(KeyCode.R)) DestroyAll();
    }

    public void DestroyAll() {
        Destroy(left);
        Destroy(right);
        Destroy(mortarCrew);
        mortarCrew = Instantiate(mortarCrewPrefab, spawnPosition, Quaternion.identity);
    }

    public void SpawnEnemy() {
        float random = Random.Range(-1f, 1f);

        if(random > 0f && right == null) right = Instantiate(tankPrefab, new Vector3(spawnDistance, 0, -10), Quaternion.Euler(0, -90, 0));
        else if(random <= 0f && left == null) left = Instantiate(tankPrefab, new Vector3(-spawnDistance, 0, -10), Quaternion.Euler(0, 90, 0));
        else if(right == null) right = Instantiate(tankPrefab, new Vector3(spawnDistance, 0, -10), Quaternion.Euler(0, -90, 0));
        else if(left == null) left = Instantiate(tankPrefab, new Vector3(-spawnDistance, 0, -10), Quaternion.Euler(0, 90, 0));

        spawnRate = cooldownTime;
    }
}
```

Slika 7. Programski kod klase SpawnerScript