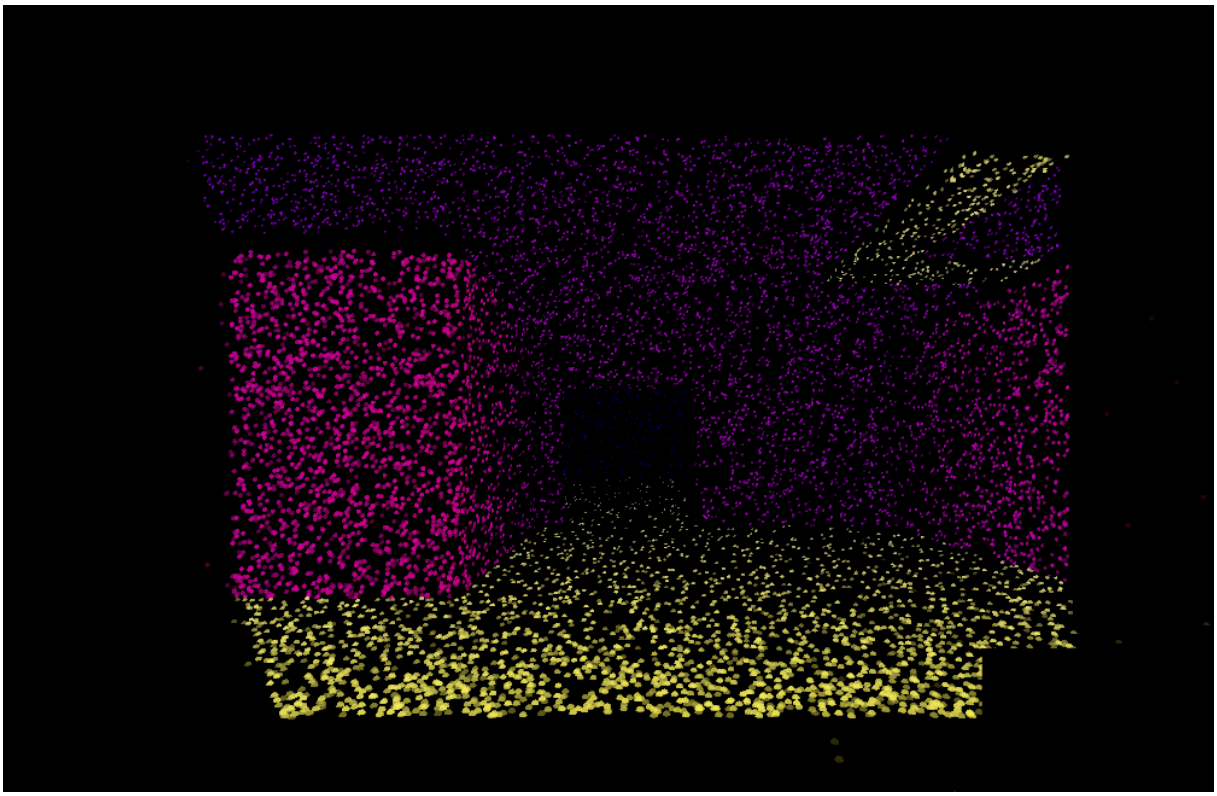
Računalna animacija

3. laboratorijska vježba

Patrick Kliček

Opis vježba:

U vježbi je napravljena kratka igra „Scanner“ u kojoj je zadatak igrača pomoću skenera orijentirati se u prostoru. Igra se sastoji od dvije sobe u kojima treba pronaći predmete za nastavljanje dalje.



Slika 1. Sken prostora

Implementacija i glavna prepreka:

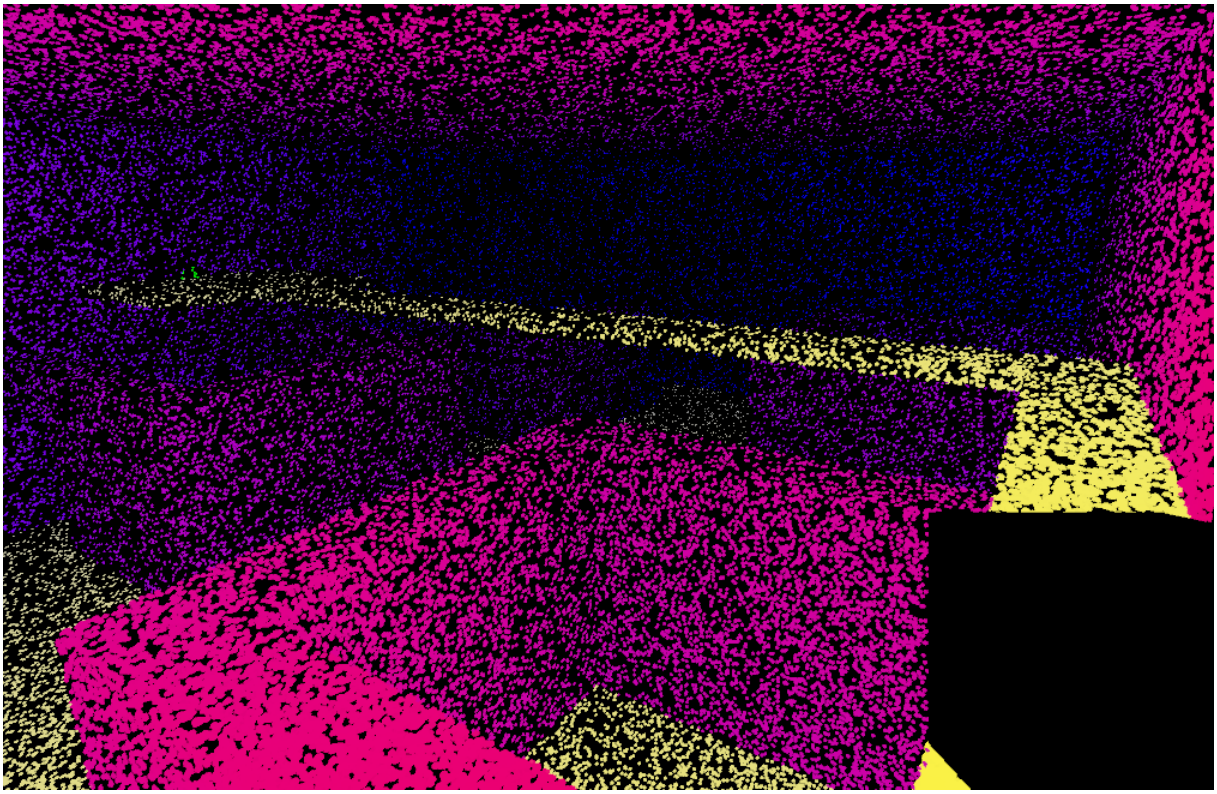
Igra je izrađena u Unityu, verzija 2022.3.28f1.

Prostor igre je razdvojen po „Tag-ovima“: Wall, Floor i Objective te ovisno o tome koji je tag objekta pri skeniranju prostora prikazati će se drugačija boja.

Funkcionalnost skeniranja prostora je izvedena pomoću metode bacanja zrake (engl. Raycast). Prije početka bacanja zraka, odabiru se točke na ekranu iz kojih će se bacati zraka.

```
float viewX = Random.Range(0.5f - squareViewportSize / 2f, 0.5f + squareViewportSize / 2f);
float viewY = Random.Range(0.5f - squareViewportSize / 2f, 0.5f + squareViewportSize / 2f);
```

Donji lijevi kut ekrana odgovara koordinatama (0,0), a gornji desni koordinatama (1,1). Odabir se centrira na (0.5, 0.5), te se zatim generira točka unutar kvadrata čija veličina ovisi o varijabli squareViewportSize. Ako je veličina kvadrata postavljena na 0, nasumična točka je zapravo fiksirana u sredini. Ako je veličina postavljena na 1, nasumična točka može biti bilo gdje unutar cijelog pogleda igrača, čime se skenira cijeli prostor pred njim.



Slika 2. Sken kada je squareViewportSize = 1

Nakon što je točka stvorena za svaku stvorenu točku baca se zraka u scenu te se provjerava jeli došlo do sudara sa scenom.

```
//Bacanje zrake
if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)){
    //Sudar sa zidom
    if (hit.transform.CompareTag("Wall")){
        //Dodaj tocku sudara u listu
        Matrix4x4 matrix = Matrix4x4.TRS(hit.point, Quaternion.identity, Vector3.one * hitSize);
        wallHits.Add(matrix);
        wallHitTimers.Add(hitDuration);
        float udaljenostOdIgraca = Mathf.InverseLerp(0f, maxDistance, hit.distance);
        wallColors.Add(Color.Lerp(Color.red, Color.blue, udaljenostOdIgraca));
    }

    //SUdar sa podom
    else if (hit.transform.CompareTag("Floor")){
        //Dodaj tocku sudara u listu
        Matrix4x4 matrix = Matrix4x4.TRS(hit.point, Quaternion.identity, Vector3.one * hitSize);
        floorHits.Add(matrix);
        floorHitTimers.Add(hitDuration);
        float udaljenostOdIgracat = Mathf.InverseLerp(0f, maxDistance, hit.distance);
        //Color purple = new Color(0.5f, 0f, 1f);
        floorColors.Add(Color.Lerp(Color.yellow, Color.gray, udaljenostOdIgracat));
    }
```

Ako je došlo do sudara sa scenom, provjerava se „Tag“ objekta sa kojim se zraka sudarila i u listu se dodaje: točka sudara, vrijeme trajanja života točke i boja točke. Boja točke ovisi o udaljenosti mjesta sudara od igrača pa tako točke koje su bliže sudaru imaju topliju boju, a one koje su dalje imaju hladniju boju.

Glavna prepreka u izradi je bila: „Kako instancirati tako puno objekata bez pada performansi?“. Rješenje problema je GPU Instanciranje. Prije instanciranja novih točaka potrebno je prvo provjeriti sve prijašnje te skratiti im vrijeme života.

```
//Projdi kroz listu i smanji vrijeme prikaza za zid
for (int i = wallHitTimers.Count - 1; i >= 0; i--) {
    wallHitTimers[i] -= Time.deltaTime;
    float alfa = wallHitTimers[i] / hitDuration; //Izracun transparentnosti
    wallColors[i] = new Color(wallColors[i].r, wallColors[i].g, wallColors[i].b, alfa);

    //Uklanjanje onih tocaka cije je vrijeme isteklo
    if (wallHitTimers[i] <= 0f){
        wallHitTimers.RemoveAt(i);
        wallHits.RemoveAt(i);
        wallColors.RemoveAt(i);
    }
}
```

Vrijeme života se koristi za računanje alfe točke, što je točka bliže kraju života to ona postaje više „transparentna“.

Nakon provjere života postojećih točaka potrebno je instancirati nove točke. Unity dopušta maksimalno 1024 instanciranja odjednom, a kako će tijekom igre biti više od 1024 točaka potrebno ih je instancirati u „baches“ veličine do 1024.

```
//Iscrtavaj sve dok ima tocaka za iscrtati
while (offsetWall < totalWall)
{
    //Moguce je iscrtati maksimalno 1023 objekata u jednom pozivu
    int count = Mathf.Min(BATCH_SIZE, totalWall - offsetWall);

    //Za svaku tocku kopiraj njenu TRS matricu i boju
    for (int i = 0; i < count; i++)
    {
        matrices[i] = wallHits[offsetWall + i];
        instanceColors[i] = wallColors[offsetWall + i];
    }

    //Posalji u propBlock boju za materijal
    propBlock.Clear();
    propBlock.SetVectorArray("_InstanceColor", instanceColors);

    //Poziv iscrtavvanjaa na GPU
    Graphics.DrawMeshInstanced(
        hitMesh,
        0,
        wallMaterial,
        matrices,
        count,
        propBlock
    );

    offsetWall += count;
}
```

U petlji se prvo određuje koliko točaka se mora instancirati, odnosno veličina „batcha“. Za svaku točku se uzima njeno mjesto sudara sa scenom te njena boja. Na kraju se prosljeđuje informacija za instanciranje: šalje se mesh objekta koji želimo instancirati, njegov materijal, transformacijska matrica, broj instanci i informacija o boji.

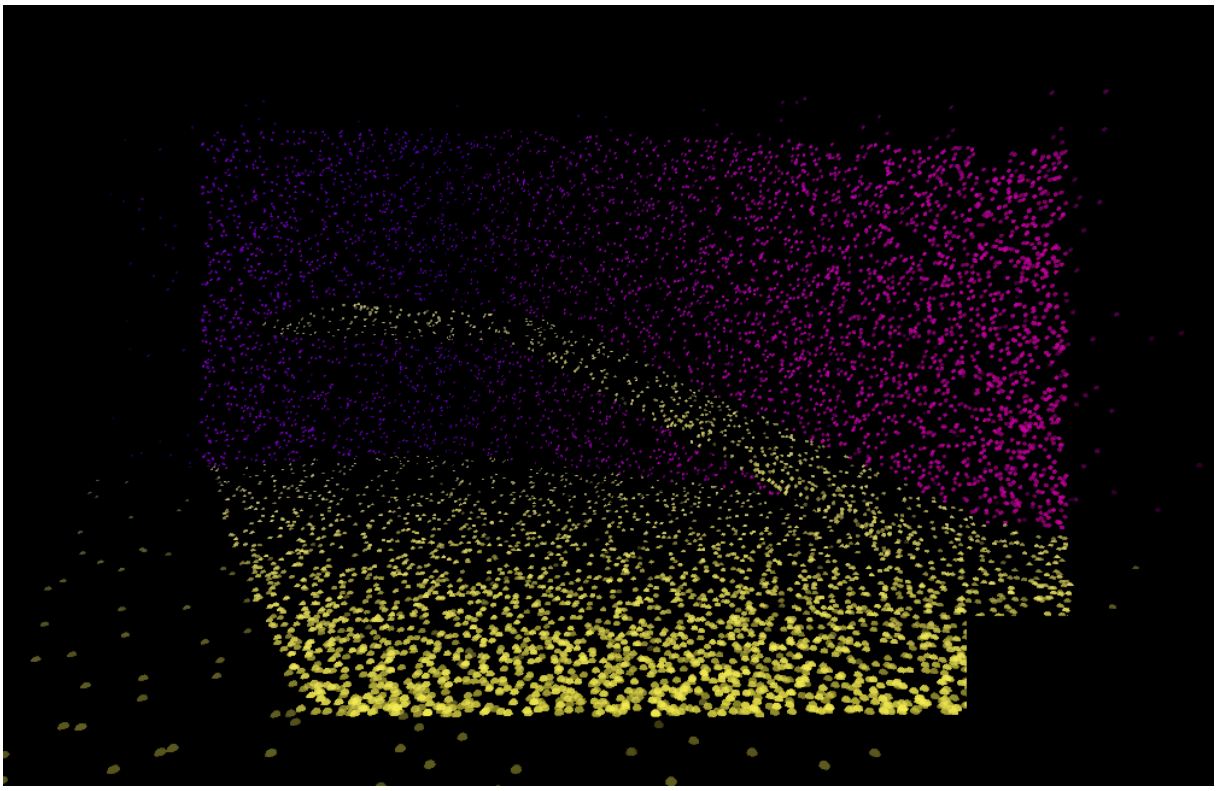
Upute za korištenje

Igrač se kreće u prostoru sa tipkama A/S/W/D, okreće kameru pomicanjem miša, skenira prostor držanjem lijevog klika miša te pokupi predmet sa tipkom E kada se onda prikaže.

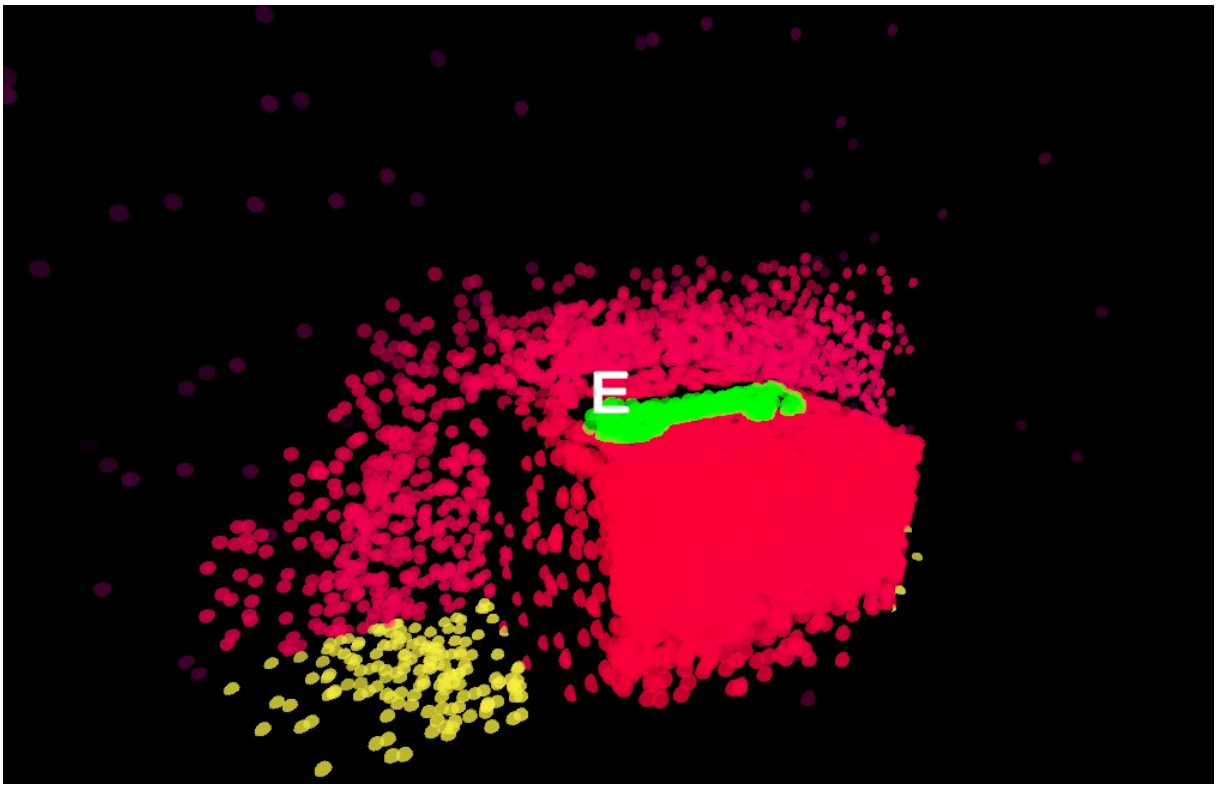
Boje koje nastaju skeniranjem imaju različita značenja: plava označava zid, žuta označava pod/krov, a zelena označava „objective“.

Upute za pokretanje

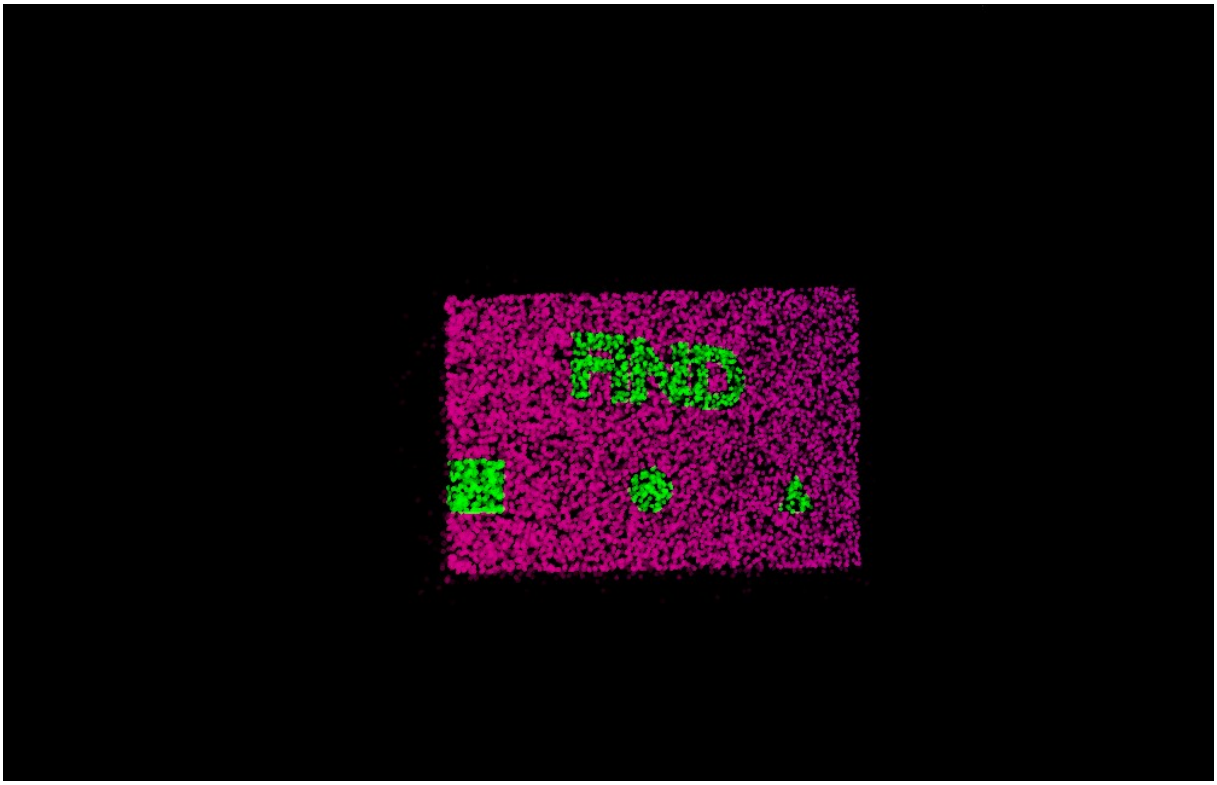
Potrebno je sa Githuba (https://github.com/PatrickKlicek/Racunalna-Animacija) preuzeti projekt za editor ili .exe build. Ako igrač želi eksperimentirati sa veličinom skener kvadrata, meshom koji se instancira, bojama ili slično potrebno je preuzeti projekt za editor u suprotnom dovoljno je pokrenuti .exe.



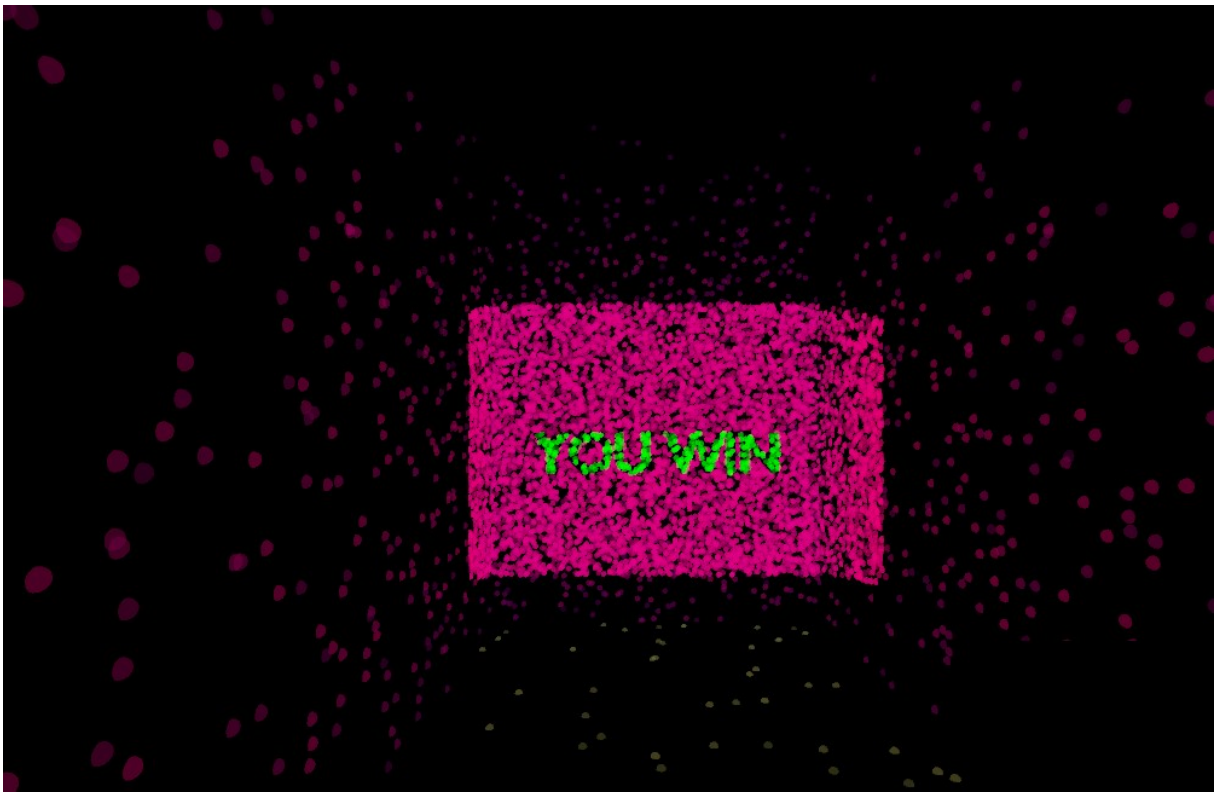
Slika 3.



Slika 4.



Slika 5.



Slika 6.