SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SAMOSTALNA VJEŽBA – RAČUNALNA ANIMACIJA

Bubble Shooter

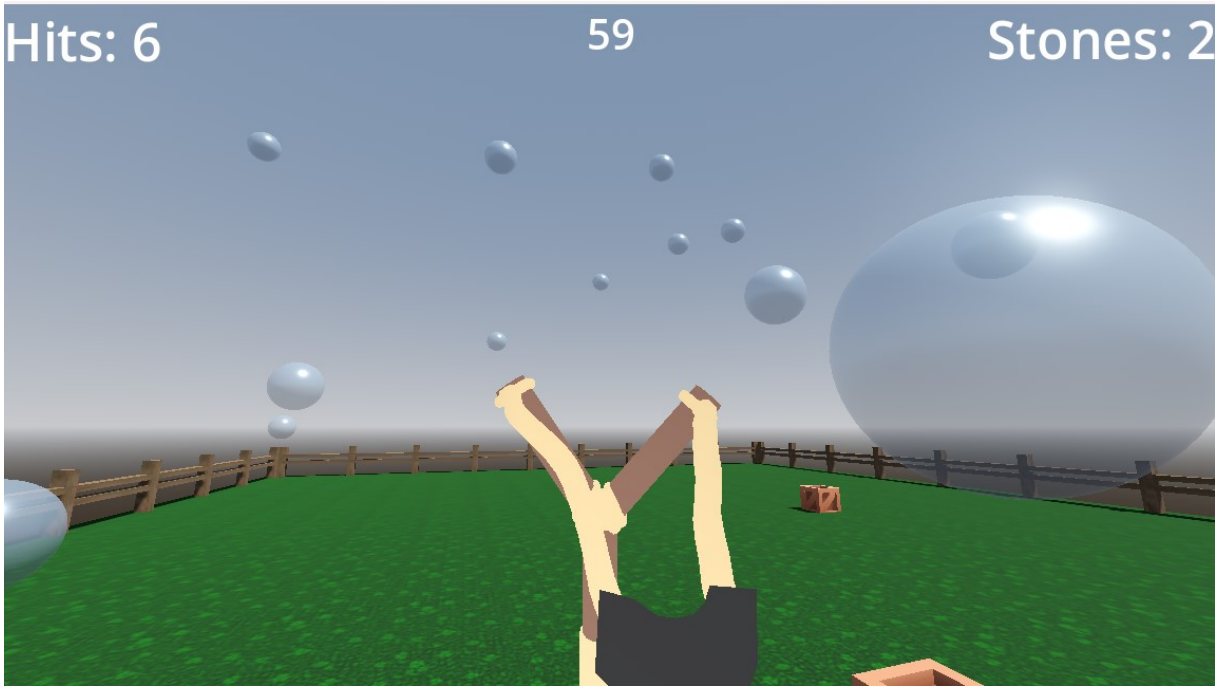
Luka Đud

Zagreb, siječanj 202

Opis igre

U sklopu samostalne vježbe iz kolegija Računalna Animacija implementirana je 3D igrica u prvom licu s elementima pucačine, odnosno FPS-a nazvana *Bubble Shooter*. Cilj igre je pogoditi što više mjehurića moguće u ograničenom vremenskom razdoblju od 100 sekundi pomoću praćke koja ispaljuje kamenčiće. Količina kamenčića je ograničena te je potrebno sakupljati pakete municije s terena kako se zaliha ne bi istrošila. Prostor igranja je površina veličine 400 metara kvadratnih te se mjehurići stvaraju u nasumičnim lokacijama uz nasumičan smjer i brzinu kretanja te predstavljaju čime simuliraju sustav čestica.

Igrač se pokreće pomoću tipki W, S, A i D, pri čemu smjer kretanja prati rotaciju kamere koja se kontrolira pokretom miša. Brzina kretanja ovisi o tome je li igrač u modu sprinta ili hodanja (Shift). Pucanje iz praćke aktivira se klikom na lijevog gumba miša, čime se ispaljuje kamenčić u smjeru u kojem je praćka usmjerena. Također igrač može pritisnuti tipku za skok (Space).



Implementacija

Igrica je razvijena korištenjem Godot Engine-a verzije 4.3, uz skriptiranje u GDScript-u, jeziku koji je vrlo sličan Python-u. Godot se temelji na hijerarhijskoj strukturi čvorova, što omogućava intuitivan i efikasan razvoj. Osim toga, Godot je besplatan, lako se instalira i pruža sve potrebne alate, što su ključni razlozi zbog kojih je izabran za implementaciju ove igrice.

Mjehurići

Mjehurići implementirani su kao sustav čestica, gdje *Bubble* scena definira ponašanje pojedinačne čestice, dok je *Bubble_spawner* odgovoran za njihovo generiranje. U *Bubble.gd* skripti, unutar funkcije *_ready*, definira se brzina i smjer čestice koristeći slučajne vrijednosti. Funkcija *_process* ažurira poziciju čestice u svakom trenutku, čime se mjehurić kreće. S druge strane, u *Bubble_spawner.gd* skripti, koristi se funkcija *spawn_bubbles* za generiranje određenog broja čestica, a poziva se svake sekunde završetkom postavljenog brojača (*_on_timer_timeout*). Svaka čestica instancira se i pozicionira na slučajnu lokaciju unutar zadanog područja definiranog vektorom SPAWN_AREA_SIZE.

```gdscript
func _ready() -> void:
    speed = randf_range(0.1, 3)
    velocity = Vector3(
        randf_range(-1.0, 1.0),
        randf_range(-1.0, 1.0),
        randf_range(-1.0, 1.0)
    ).normalized() * speed

func _process(delta: float) -> void:
    position += velocity * delta
```

Bubble.gd (isječak)

```gdscript
extends Node3D

var bubble_scene = preload("res://Scenes/Bubble.tscn")
const SPAWN_AREA_SIZE = Vector3(10, 7, 10)
const BUBBLE_COUNT = 2

func _ready() -> void:
    spawn_bubbles(BUBBLE_COUNT)

func spawn_bubbles(count) -> void:
    for i in range(count):
        spawn_bubble()

func spawn_bubble() -> void:
    if bubble_scene:
        var bubble_instance = bubble_scene.instantiate()
        bubble_instance.position = Vector3(
            randf_range(-SPAWN_AREA_SIZE.x, SPAWN_AREA_SIZE.x),
            randf_range(0, SPAWN_AREA_SIZE.y),
            randf_range(-SPAWN_AREA_SIZE.z, SPAWN_AREA_SIZE.z)
        )
        add_child(bubble_instance)


func _on_timer_timeout() -> void:
    spawn_bubbles(BUBBLE_COUNT)
```

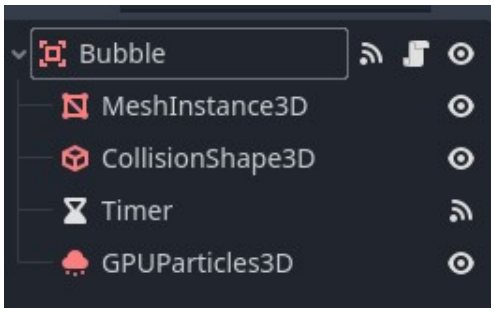
Bubble_spawner.gd

Kako bi mjehurić detektirao sudar s drugim objektima, u sceni mjehurića koristi se čvor tipa *Area3D*. Ovaj čvor omogućuje prepoznavanje kolizije s drugim tijelima unutar definiranog prostora, koji je oblikovan korištenjem čvora CollisionShape3D. Kada se dogodi sudar, aktivira se funkcija *_on_body_entered*, koja razlikuje sudare s metkom i ostalim objektima. Ako tijelo koje ulazi u područje pripada grupi *Bullet*, poziva se funkcija *_on_bubble_shot*. Ova funkcija bilježi pogodak povećavajući broj pogođenih mjehurića kod igrača. Osim toga, vizualni prikaz mjehurića se isključuje, a

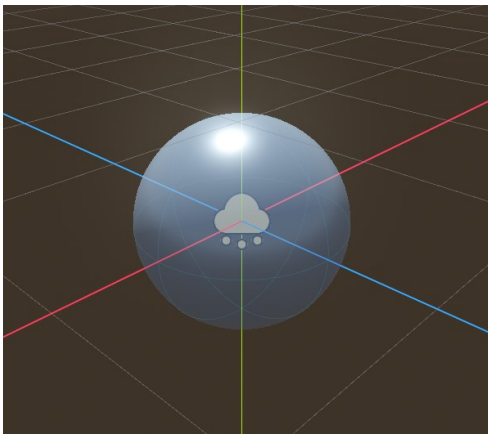
aktivira se emitiranje čestica kako bi se simuliralo pucanje mjehurića. Nakon kratkog vremenskog razmaka, mjehurić se uklanja iz scene. U slučaju da tijelo koje ulazi u područje nije metak ili ako mjehurić ne detektira koliziju unutar 15 sekundi, mjehurić također "puca" bez signalizacije pogotka. Vizualni prikaz se isključuje, čestice se aktiviraju, a mjehurić se uklanja iz scene nakon kratkog vremenskog razmaka. Na ovaj način, sustav osigurava pravilno upravljanje sudarima i ponašanjem mjehurića unutar igre.

```gdscript
func _on_body_entered(body: Node3D) -> void:
    if body.is_in_group("Bullet"):
        _on_bubble_shot()
    else:
        mesh.visible = false
        particles.emitting = true
        await get_tree().create_timer(1.0).timeout
        queue_free()

func _on_bubble_shot() -> void:
    player.hit_count += 1
    player.update_hit_counter()
    mesh.visible = false
    particles.emitting = true
    player._rectangle_timer()
    await get_tree().create_timer(1.0).timeout
    queue_free()
    return

func _on_timer_timeout() -> void:
    mesh.visible = false
    particles.emitting = true
    await get_tree().create_timer(1.0).timeout
    queue_free()
```

Bubble.gd (isječak)



Bubble scena

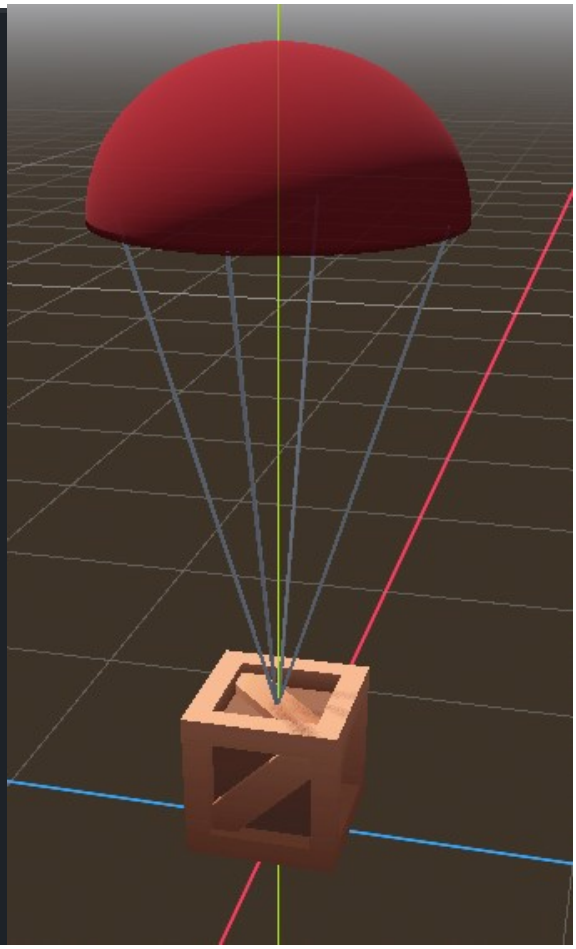


Municija (kamenčići)

Sustav za stvaranje municije u igri funkcionira na način da generira kutije (ispunjene kamenčićima) koje igrač može pokupiti. Kada se kutija stvori na nasumičnoj poziciji unutar definirane površine igranja, instanciran je objekt 10 metara zraku. Objekt se polako spušta prema tlu konstantnom brzinom, simulirajući pad. Tijekom spuštanja, kutija s padobranom nastavlja se kretati prema dolje pomoću *_process* funkcije sve dok te dotakne razinu na kojoj se nalazi teren. Kada dosegne tlo, aktivira se funkcija *_on_parachute_land*, koja uklanja objekt iz scene, i na njegovo mjesto postavlja novi objekt kutije tipa *Area3D* koji je moguće pokupiti kolizijom igrača i *CollisionShape3D* čvora navedenog objekta. Kada igrač pokupi kutiju, količina municije u igri povećava

se za 15 kamenčića. Proces stvaranja novih kutija aktivira se pomoću *Timer* čvora koji svakih 16 sekundi poziva funkciju za generiranje kutije.

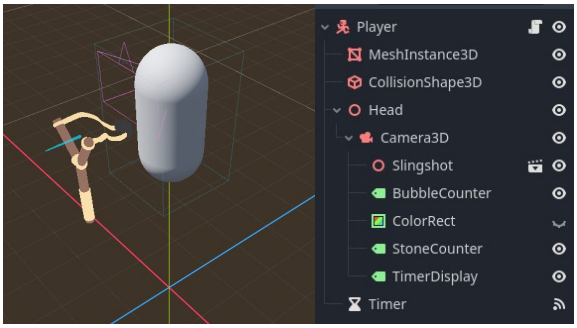
```gdscript
var parachutes = []

func spawn_stones(count) -> void:
    for i in range(count):
        spawn_stone()

func spawn_stone() -> void:
    if parachute_scene:
        var parachute_instance = parachute_scene.instantiate()

        parachute_instance.position = Vector3(
            randf_range(-SPAWN_AREA_SIZE.x, SPAWN_AREA_SIZE.x),
            10.0,
            randf_range(-SPAWN_AREA_SIZE.z, SPAWN_AREA_SIZE.z)
        )
        add_child(parachute_instance)

        parachutes.append(parachute_instance)

func _process(delta) -> void:
    for parachute_instance in parachutes:
        parachute_instance.position.y -= FALL_SPEED * delta

        if parachute_instance.position.y <= 0:
            _on_parachute_land(parachute_instance)

func _on_parachute_land(parachute_instance) -> void:
    var new_stone_instance = stone_pickup_scene.instantiate()
    new_stone_instance.position = parachute_instance.position
    new_stone_instance.rotation = parachute_instance.rotation

    parachute_instance.queue_free()
    add_child(new_stone_instance)

    parachutes.erase(parachute_instance)

func _on_timer_timeout() -> void:
    spawn_stones(PICKUP_COUNT)
```



Stone_spawner.gd

Igrač

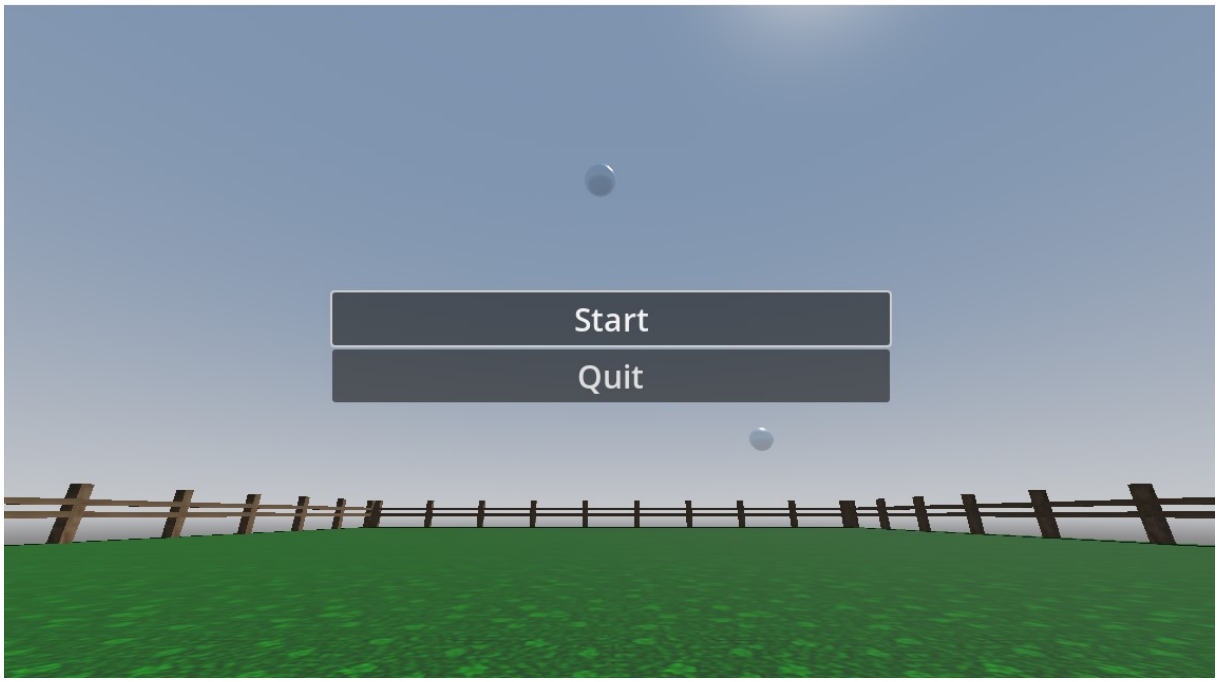
Igrač u igri ima mogućnost slobodnog kretanja unutar 3D prostora koristeći kombinaciju tipkovnice i miša. Korijen scene igrača je čvor tipa *CharacterBody3D* koji omogućava glatko i prirodno kretanje koristeći *move_and_slide* funkciju. Smjer kretanja izračunava se na temelju unosa s tipkovnice, pri čemu funkcija *Input.get("left", "right", "up", "down")* generira vektor smjera koji se transformira u lokalni koordinatni sustav igrača pomoću orijentacije kamere, omogućujući da kretanje uvijek prati smjer pogleda. Brzina i smjer kretanja kombiniraju se u varijablu *velocity,* koja predstavlja trenutnu brzinu u svim smjerovima.



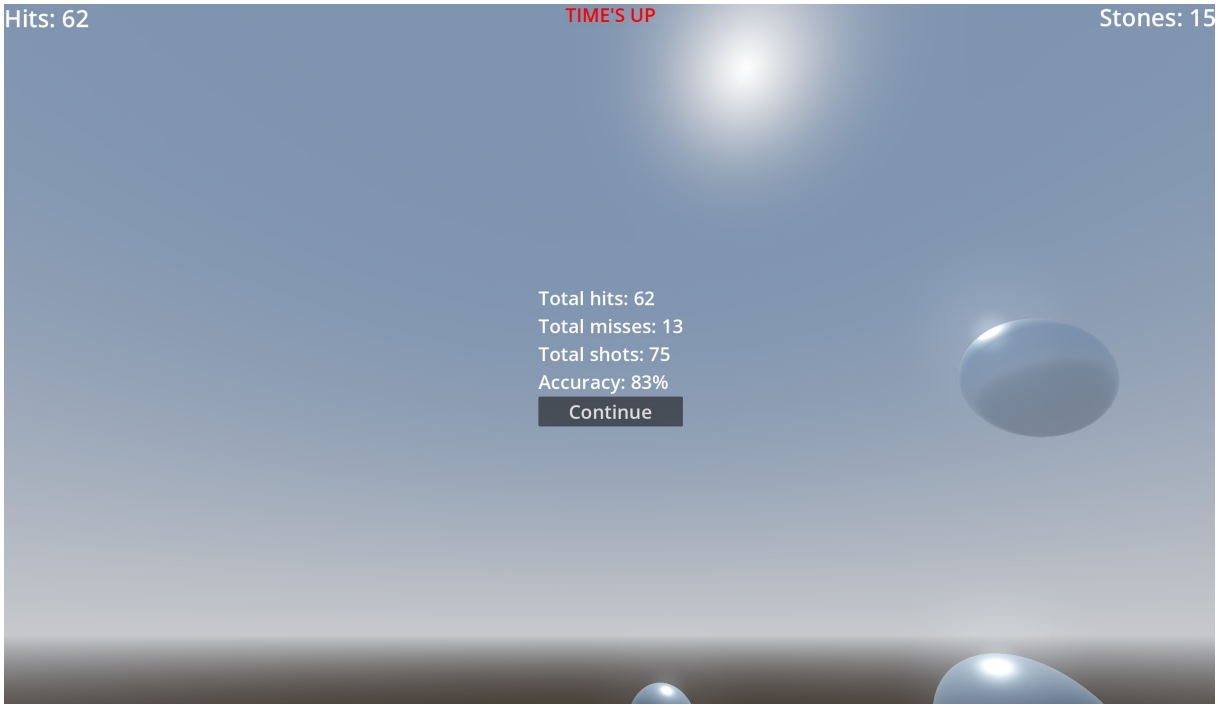
```gdscript
func _physics_process(delta: float) -> void:
	if not is_on_floor():
		velocity += get_gravity() * delta

	if Input.is_action_just_pressed("jump") and is_on_floor():
		velocity.y = JUMP_VELOCITY

	if Input.is_action_pressed("sprint"):
		speed = SPRINT_SPEED
	else:
		speed = WALK_SPEED
		
	var input_dir = Input.get_vector("left", "right", "up", "down")
	var direction = (head.transform.basis * Vector3(input_dir.x, 0, input_dir.y)).normalized()

	if direction:
		velocity.x = direction.x * speed
		velocity.z = direction.z * speed
	else:
		velocity.x = 0.0
		velocity.z = 0.0

	if Input.is_action_pressed("shoot"):
		if !sling_animation.is_playing() and stone_count > 0:
			sling_animation.play("Shoot")
			await sling_animation.animation_finished
			instance = bullet.instantiate()
			instance.position = sling_barrel.global_position
			instance.transform.basis = sling_barrel.global_transform.basis
			get_parent().add_child(instance)
			stone_count -= 1
			update_stone_counter()
			total_shots += 1
		if stone_count == 0:
			no_ammo_label.visible = true
			await get_tree().create_timer(1).timeout
			no_ammo_label.visible = false

	move_and_slide()
```

Player.gd

Glavni izbornik



Završni prikaz



Upute za pokretanje

1. Otvoriti terminal i klonirati projekt
2. Pokrenuti Godot
3. Učitati projekt u Godot s direktorijem bubblegame
4. Pokrenuti pomoću sučelja (play ikona)