



Politechnika Wrocławska

Projektowanie i programowanie gier

Laboratorium

Raport z części podstawowej

Autor: Katarzyna Idasz
ITE W4N

Prowadzący: dr inż. arch. Tomasz Zamojski

Termin oddania: 29.04.2025r.

Wrocław, 2025

Spis treści

1	Wprowadzenie	2
2	Poradnik 2D	2
2.1	Tworzenie nowego projektu	2
2.2	Dodanie sceny gracza	2
2.3	Kod gracza	2
2.4	Tworzenie przeciwnika	5
2.5	Główna scena gry	6
2.6	HUD	7
2.7	Efekt końcowy	9
3	Poradnik 3D	10
3.1	Podstawy obiektów 3D	10
3.2	Dodanie gracza	11
3.3	Importowanie z Blender'a i rotacja kuli	13
3.4	Kamera, światło i środowisko	14
3.5	Tekstury i ściany	15
3.6	Menu	17
3.7	Dodanie przeciwników	18
3.8	Dodanie monet i HUDu	19
3.9	Dodatki	20

1 Wprowadzenie

Część podstawowa laboratorium zawiera wykonanie dwóch poradników z silnika i środowiska do tworzenia gier Godot. Pierwszy poradnik dotyczy gier 2D, a drugi 3D. W celu wykonania zadań pobrano Godot w wersji 4.4.1 na Windowsie 11.

2 Poradnik 2D

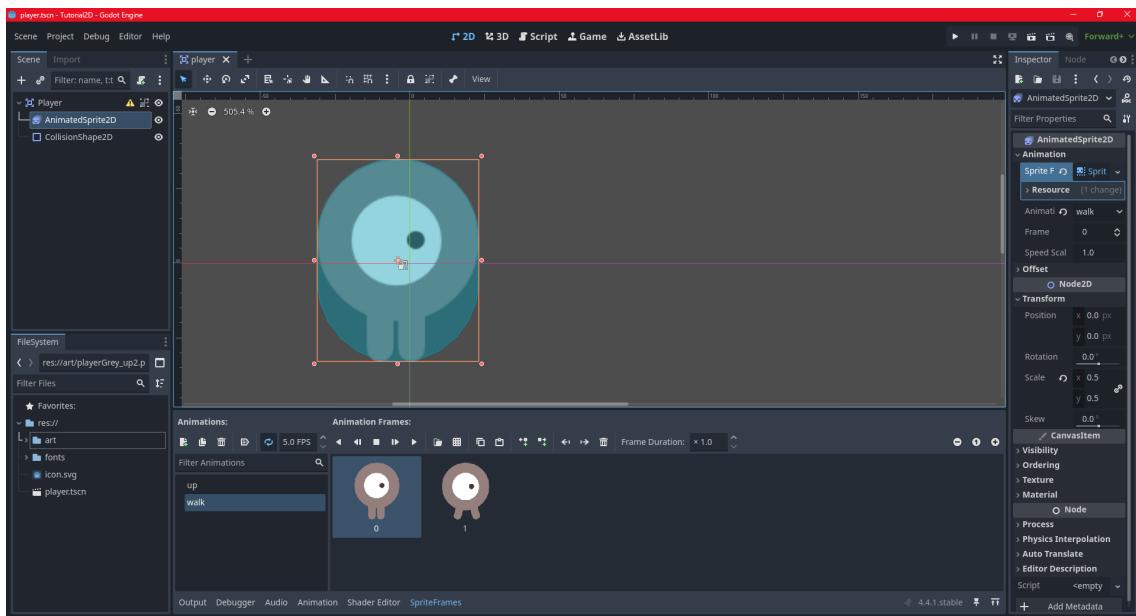
Pierwsze zadanie to wykonanie poradnika ze strony [dokumentacji silnika](#). W tej grze 2D gracza porusza postacią, która ma jak najdłużej nie dotknąć żadnego pojawiającego się przeciwnika.

2.1 Tworzenie nowego projektu

Pierwszym punktem w poradniku było utworzenie nowego projektu oraz zimportowanie gotowych assetów. Następnie należało ustawić rozmiar okna na 480 x 720 px oraz zmienić ustawienia skalowania.

2.2 Dodanie sceny gracza

Projekt w Godocie ma strukturę węzłów (ang. Node). Należało stworzyć węzeł klasy *Area2D* o nazwie *Player*. Następnie dodano dwóch potomków - animację klasy *AnimatedSprite2D* oraz model kolizji klasy *CollisionShape2D*. Dodano dwie animacje - walk oraz up, które składają się z gotowych assetów. Skalowanie *AnimatedSprite2D* ustawiono na 0.5. Utworzono hitbox postaci z klasy *CapsuleShape2D*.

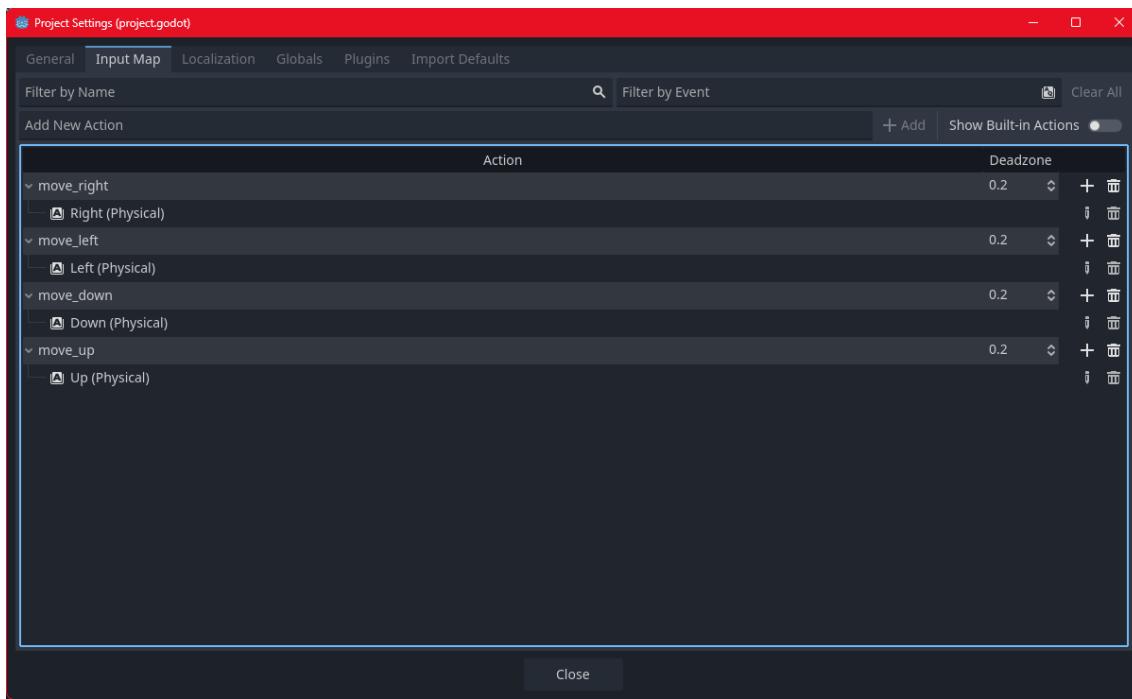


Rysunek 1: Utworzenie animacji i hitboxu postaci.

2.3 Kod gracza

Należało dodać skrypt do węzła Player w celu zaimplementowania ruchu gracza oraz jego animacji. Skrypt został napisany w języku GDScript, który składnią przypomina język Python.

Dodano cztery ruchy gracza w *Input Map*. Ruch gracza odbywa się poprzez naciśnięcie odpowiedniego klawisza - strzałki. Podpięcie odpowiedniego przycisku/klawisza w Godocie jest bardzo łatwe - konfiguracja odbywa się poprzez dodanie akcji w menadżerze akcji oraz naciśnięcie np. strzałki, która zostanie wykryta i przypisana do tej akcji.



Rysunek 2: Przypisanie klawiszy do odpowiednich akcji.

Skrypt składa się z:

- Zadeklarowania zmiennej globalnej *speed* odpowiadającej za prędkość poruszającego się gracza w pikselach na sekundę oraz zmiennej przechowującej rozmiar okna.
- Zdefiniowania funkcji *_ready()*, która zostaje wywołana kiedy pojawi się na scenie węzeł.
- Zdefiniowania funkcji *_process(delta)*, która wywoywana jest w każdej klatce i odpowiada za:
 - sprawdzenie, czy został wykryty input,
 - poruszenie postaci w podanym kierunku,
 - wybranie odpowiedniej animacji.
- Zdefiniowania funkcji *_on_body_entered(_body)*, która odpowiada za wykonane akcje po wykryciu kolizji.
- Zdefiniowania funkcji *start(pos)*, która pozwala na reset gracza przy rozpoczęciu nowej gry.

```

1 extends Area2D
2 signal hit
3
4 @export var speed = 400 # How fast the player will move (pixels/sec).
5 var screen_size # Size of the game window.
6
7 func _ready():
8     screen_size = get_viewport_rect().size
9     hide()
10
11 func _process(delta):
12     var velocity = Vector2.ZERO # The player's movement vector.
13     if Input.is_action_pressed("move_right"):
14         velocity.x += 1
15     if Input.is_action_pressed("move_left"):
16         velocity.x -= 1
17     if Input.is_action_pressed("move_down"):
18         velocity.y += 1
19     if Input.is_action_pressed("move_up"):
20         velocity.y -= 1
21
22     if velocity.length() > 0:
23         velocity = velocity.normalized() * speed
24         $AnimatedSprite2D.play()
25     else:

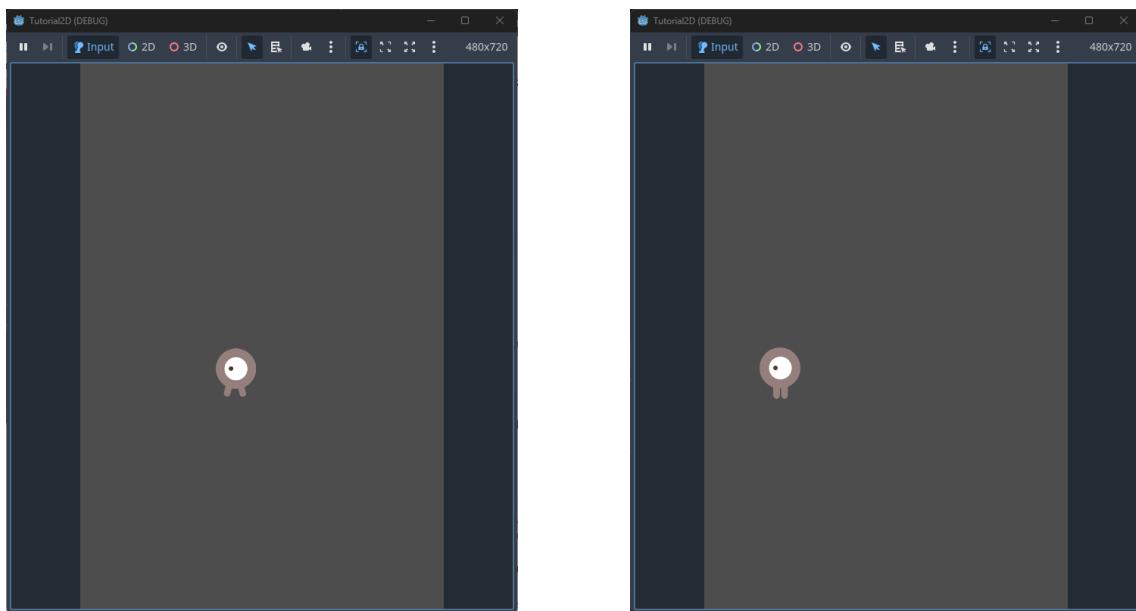
```

```

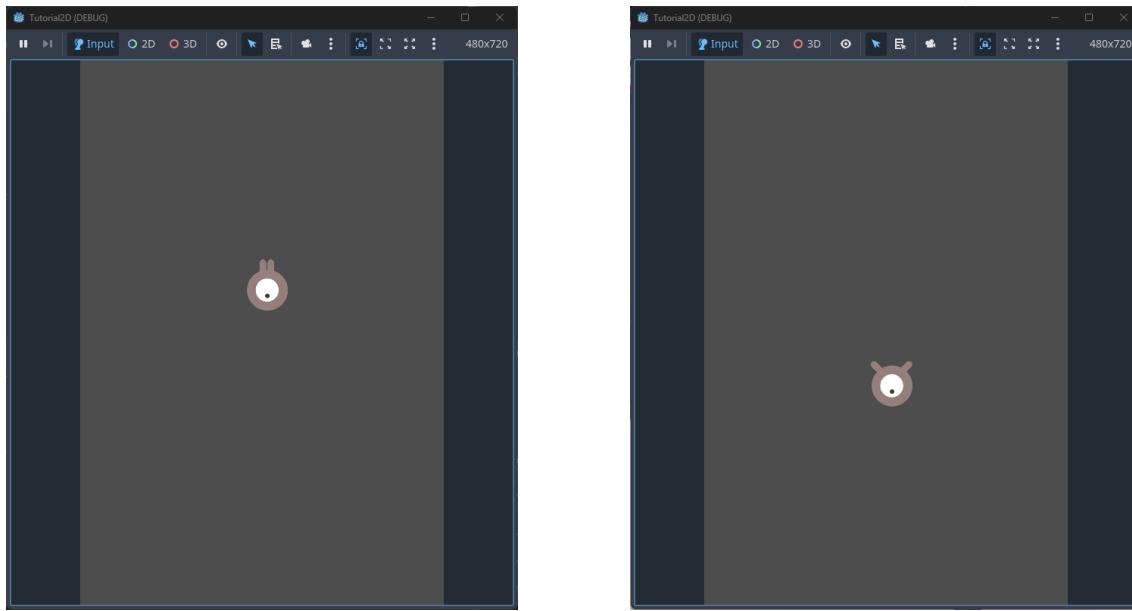
26     $AnimatedSprite2D.stop()
27
28     position += velocity * delta
29     position = position.clamp(Vector2.ZERO, screen_size)
30
31     if velocity.x != 0:
32         $AnimatedSprite2D.animation = "walk"
33         $AnimatedSprite2D.flip_v = false
34         # See the note below about the following boolean assignment.
35         $AnimatedSprite2D.flip_h = velocity.x < 0
36     elif velocity.y != 0:
37         $AnimatedSprite2D.animation = "up"
38         $AnimatedSprite2D.flip_v = velocity.y > 0
39
40
41 func _on_body_entered(body: Node2D) -> void:
42     hide() # Player disappears after being hit.
43     hit.emit()
44     # Must be deferred as we can't change physics properties on a physics callback.
45     $CollisionShape2D.set_deferred("disabled", true)
46
47
48 func start(pos):
49     position = pos
50     show()
51     $CollisionShape2D.disabled = false

```

Listing 1: Kod gracza



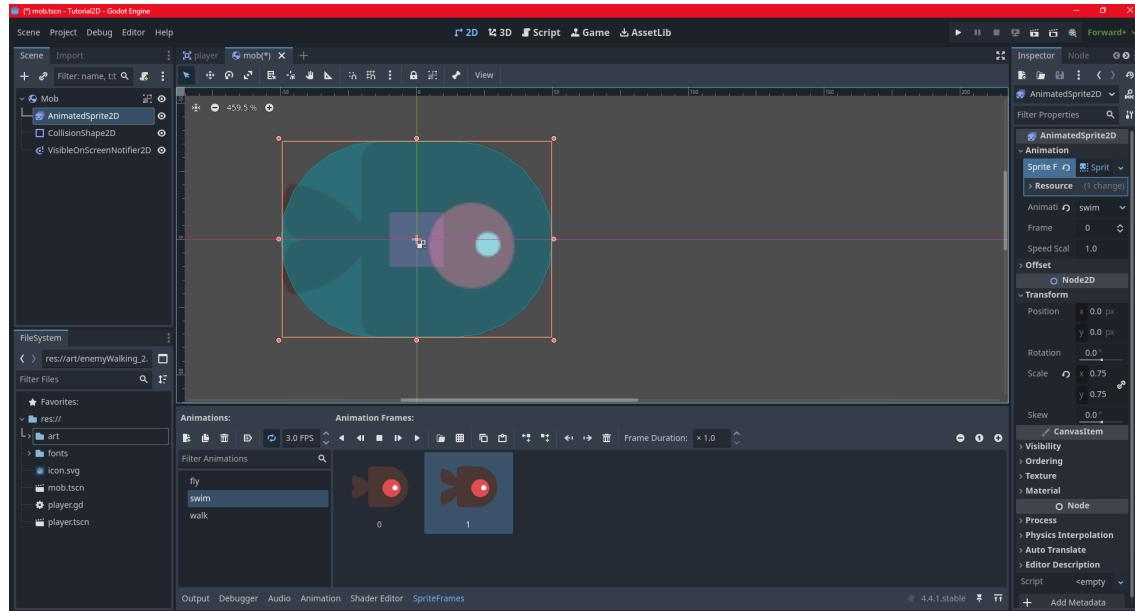
Rysunek 3: Animacja ruchu w lewo.



Rysunek 4: Animacja ruchu w dół.

2.4 Tworzenie przeciwnika

Utworzenie przeciwnika wygląda analogicznie do utworzenia gracza. Dodatkowo dodano do węzła *Mob* potomka *VisibleOnScreenNotifier2D*. Ustawiono *Gravity Scale* w węźle *Mob* na 0, w celu wyłączenia grawitacji. W sekcji *CollisionObject2D* odnaczeno 1 w *Mask* w celu usunięcia kolizji między innymi przeciwnikami. Dodano trzy animacje - *fly*, *swim* i *walk* - oraz zmieniono ich prędkość na 3 FPS'y. Przeskalowano przeciwnika w *AnimatedSprite2D* na wartość 0.75. Utworzono hitbox przeciwnika również z klasy *CapsuleShape2D*.



Rysunek 5: Utworzenie animacji i hitboxu przeciwnika.

Skrypt przeciwnika składa się z:

- Zdefiniowania funkcji *_ready()*, która zostaje wywołana kiedy pojawi się na scenie węzeł - wybierana jest losowa animacja.
- Zdefiniowania funkcji *_on_visible_on_screen_notifier_2d_screen_exited()*, która odpowiada za usunięcie przeciwnika, kiedy opuści okno.

```

1 extends RigidBody2D
2
3 func _ready():
4     var mob_types = Array($AnimatedSprite2D.sprite_frames.get_animation_names())
5     $AnimatedSprite2D.animation = mob_types.pick_random()
6     $AnimatedSprite2D.play()
7
8
9 func _on_visible_on_screen_notifier_2d_screen_exited():
10    queue_free()

```

Listing 2: Kod przeciwnika

2.5 Główna scena gry

Główna scena gry to obiekt klasy *Node* o nazwie *Main*. Jako dzieci tego węzła dodano:

- Instancję węzła *Player*.
- *MobTimer* klasy *Timer* - odpowiada za to, jak często generują się przeciwnicy - 0.5s.
- *ScoreTimer* klasy *Timer* - do inkrementacji wyniku - co 1s.
- *StartTimer* klasy *Timer* - opóźnienie przed startem - 2s.
- *StartPosition* klasy *Marker2D* - określa pozycję startową gracza - (240, 450).
- *MobPath* klasy *Path2D* - prostokąt na krańcach okna - określa możliwe miejsca pojawienia się nowego przeciwnika.
 - *MobSpawnLocation* klasy *PathFollow2D* - do podążania za ścieżką.

Skrypt sceny głównej składa się z:

- Zdefiniowania funkcji *game_over()* i *new_game()*, które definiują zachowanie na początku i końcu gry.
- Zdefiniowania funkcji trzech timerów - *_on_mob_timer_timeout()* odpowiada za tworzenie przeciwników, wybieranie dla nich losowej pozycji startowej i wprowadzanie ich w ruch.

```

1 extends Node
2
3 @export var mob_scene: PackedScene
4 var score
5
6 func game_over():
7     $ScoreTimer.stop()
8     $MobTimer.stop()
9
10
11 func new_game():
12     score = 0
13     $Player.start($StartPosition.position)
14     $StartTimer.start()
15
16
17 func _on_mob_timer_timeout():
18     # Create a new instance of the Mob scene.
19     var mob = mob_scene.instantiate()
20
21     # Choose a random location on Path2D.
22     var mob_spawn_location = $MobPath/MobSpawnLocation
23     mob_spawn_location.progress_ratio = randf()
24
25     # Set the mob's position to the random location.
26     mob.position = mob_spawn_location.position
27
28     # Set the mob's direction perpendicular to the path direction.
29     var direction = mob_spawn_location.rotation + PI / 2
30
31     # Add some randomness to the direction.
32     direction += randf_range(-PI / 4, PI / 4)
33     mob.rotation = direction

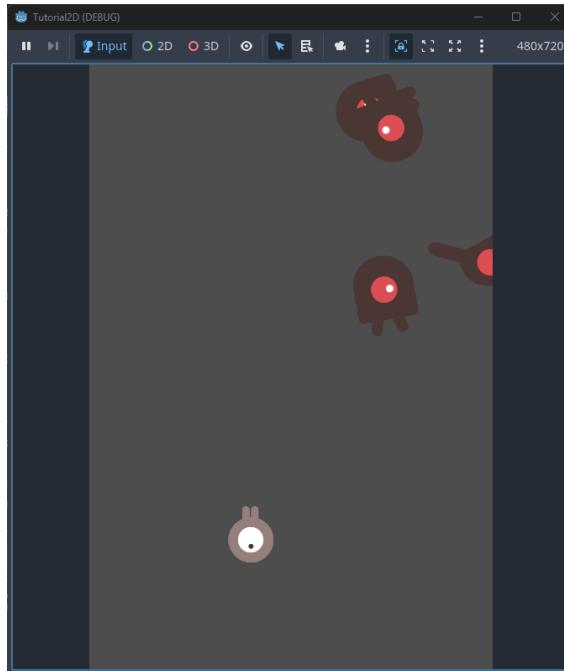
```

```

34
35 # Choose the velocity for the mob.
36 var velocity = Vector2(randf_range(150.0, 250.0), 0.0)
37 mob.linear_velocity = velocity.rotated(direction)
38
39 # Spawn the mob by adding it to the Main scene.
40 add_child(mob)
41
42
43 func _on_score_timer_timeout():
44     score += 1
45
46
47 func _on_start_timer_timeout():
48     $MobTimer.start()
49     $ScoreTimer.start()
50
51
52 func _ready():
53     new_game()

```

Listing 3: Główna scena - prototyp



Rysunek 6: Prototyp gry.

2.6 HUD

Kolejnym elementem jest dodanie interfejsu użytkownika, który będzie wyświetlał wynik, komunikaty oraz przycisk rozpoczęcia gry. Dodano węzeł *HUD* klasy *CanvasLayer*. Dziećmi tego węzła są:

- *ScoreLabel* klasy *Label* - wyświetla wynik.
- *Message* klasy *Timer* - wyświetla komunikat.
- *StartButton* klasy *Button* - przycisk do rozpoczęcia gry.
- *MessageTimer* klasy *Timer* - opóźnienie po wyświetleniu komunikatu.

Załadowano nową czcionkę i zmieniono jej rozmiar na 64 piksele. Ustwaiiono wszystkie elementy na scenie.



Rysunek 7: Ustawianie elementów na scenie.

W skrypcie zdefiniowano działania poszczególnych elementów HUD'u.

```

1 extends CanvasLayer
2
3 # Notifies 'Main' node that the button has been pressed
4 signal start_game
5
6
7 func show_message(text):
8     $Message.text = text
9     $Message.show()
10    $MessageTimer.start()
11
12
13 func show_game_over():
14     show_message("Game Over")
15     # Wait until the MessageTimer has counted down.
16     await $MessageTimer.timeout
17
18     $Message.text = "Dodge the Creeps!"
19     $Message.show()
20     # Make a one-shot timer and wait for it to finish.
21     await get_tree().create_timer(1.0).timeout
22     $StartButton.show()
23
24
25 func update_score(score):
26     $ScoreLabel.text = str(score)
27
28
29 func _on_start_button_pressed():
30     $StartButton.hide()
31     start_game.emit()
32
33 func _on_message_timer_timeout():
34     $Message.hide()
```

Listing 4: HUD

Dodano instancję *HUD* do węzła *Main*. W skrypcie main.gd dodano aktualizację HUD'u (np. wyniku). Dodano grupę przeciwników, aby usunąć ich na raz przed rozpoczęciem nowej gry. Dodatkowo dodano tło poprzez klasę *ColorRect* oraz muzykę (*AudioStreamPlayer*) - zapętloną w tle oraz efekt dźwiękowy śmierci. Utworzono nowy skrót klawiszowy *ENTER* rozpoczynający grę.

```

1 extends Node
2
3 @export var mob_scene: PackedScene
4 var score
5
6 func game_over():
7     $ScoreTimer.stop()
8     $MobTimer.stop()
9     $HUD.show_game_over()
```

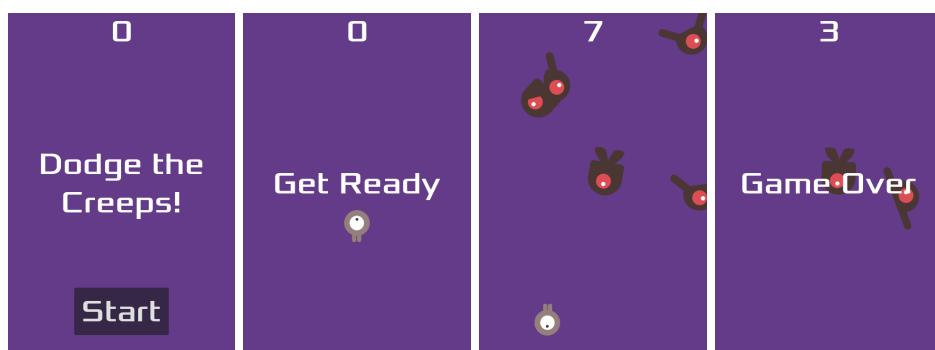
```

10   $Music.stop()
11   $DeathSound.play()
12
13
14 func new_game():
15     score = 0
16     $Player.start($StartPosition.position)
17     $StartTimer.start()
18     $HUD.update_score(score)
19     $HUD.show_message("Get Ready")
20     get_tree().call_group("mobs", "queue_free")
21     $Music.play()
22
23
24 func _on_mob_timer_timeout():
25     # Create a new instance of the Mob scene.
26     var mob = mob_scene.instantiate()
27
28     # Choose a random location on Path2D.
29     var mob_spawn_location = $MobPath/MobSpawnLocation
30     mob_spawn_location.progress_ratio = randf()
31
32     # Set the mob's position to the random location.
33     mob.position = mob_spawn_location.position
34
35     # Set the mob's direction perpendicular to the path direction.
36     var direction = mob_spawn_location.rotation + PI / 2
37
38     # Add some randomness to the direction.
39     direction += randf_range(-PI / 4, PI / 4)
40     mob.rotation = direction
41
42     # Choose the velocity for the mob.
43     var velocity = Vector2(randf_range(150.0, 250.0), 0.0)
44     mob.linear_velocity = velocity.rotated(direction)
45
46     # Spawn the mob by adding it to the Main scene.
47     add_child(mob)
48
49
50 func _on_score_timer_timeout():
51     score += 1
52     $HUD.update_score(score)
53
54
55 func _on_start_timer_timeout():
56     $MobTimer.start()
57     $ScoreTimer.start()
58
59
60 func _ready():
61     pass

```

Listing 5: Główna scena - finalna wersja

2.7 Efekt końcowy



Rysunek 8: Skończona gra.

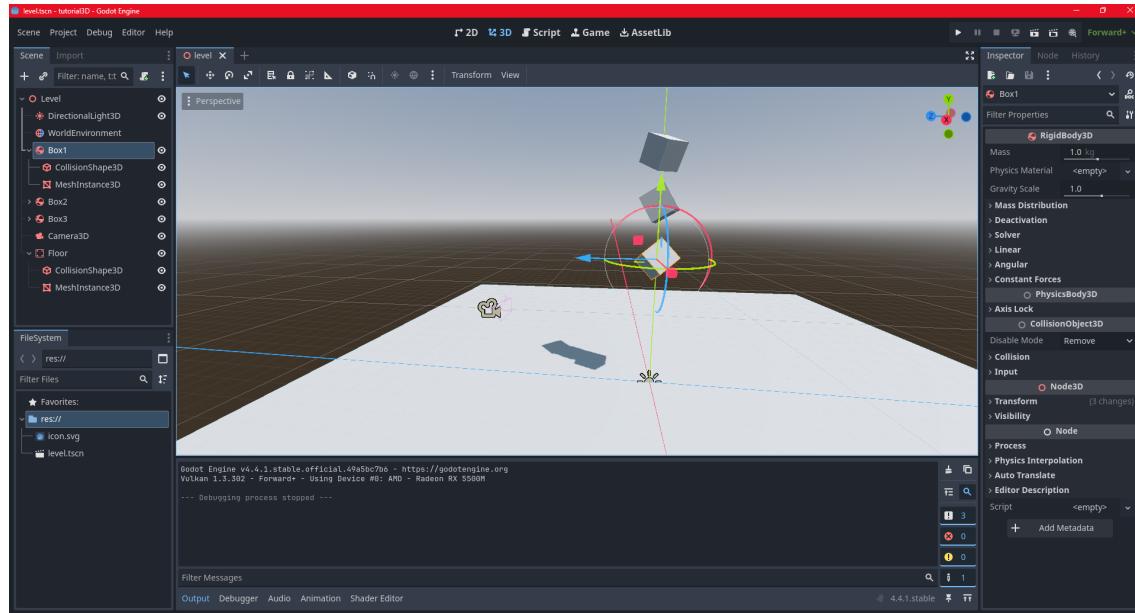
3 Poradnik 3D

Drugim zadaniem było wykonanie poradnika Godot 3 Tutorial Series autorstwa BornCG. Jest to prosta gra 3D, w której gracz porusza piłką. Celem jest zbieranie monet oraz unikanie przeciwników.

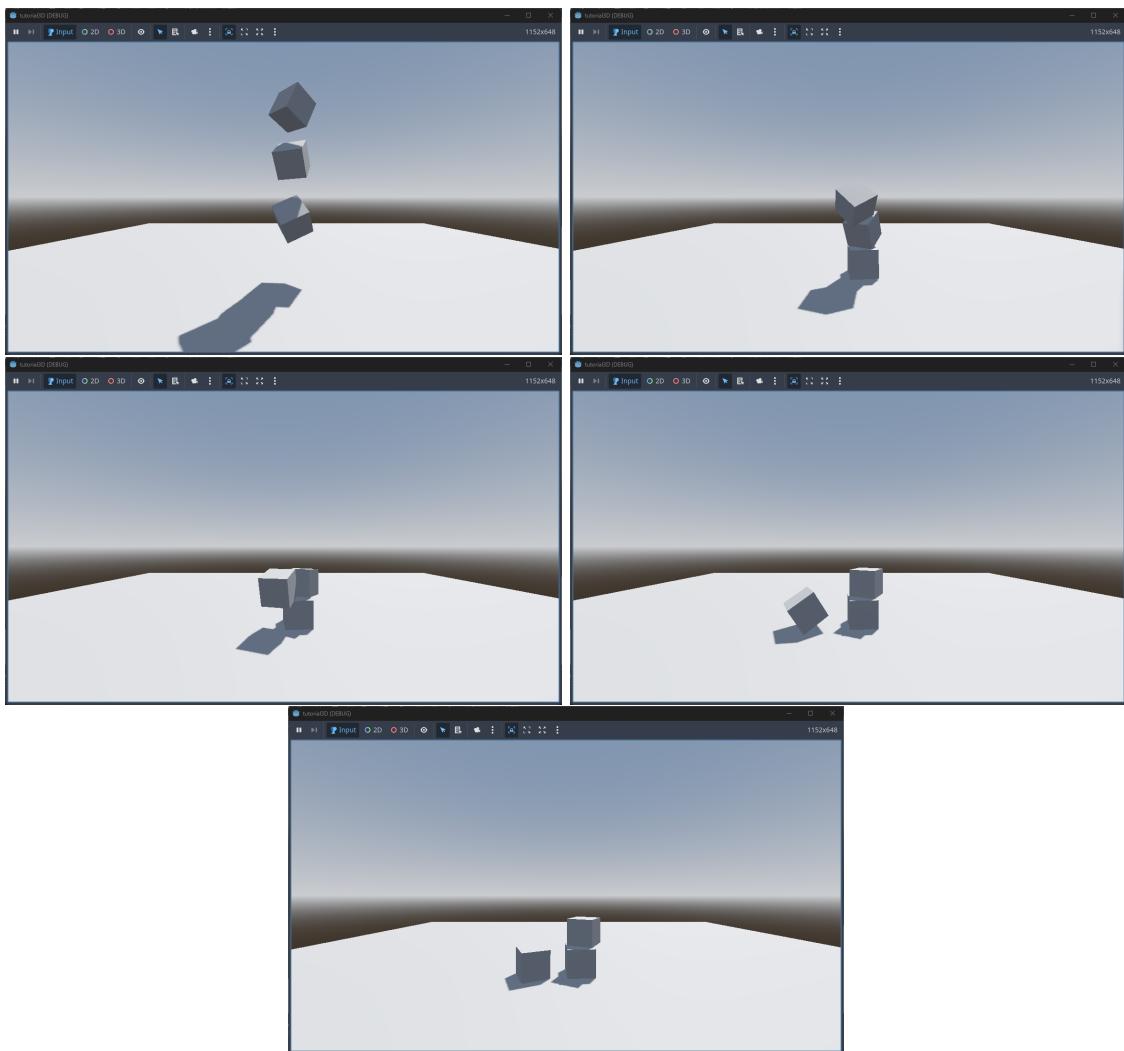
3.1 Podstawy obiektów 3D

Utworzono podłogę (*StaticBody3D*) oraz trzy sześcianny (*RigidBody3D*) z kolizją i meshem (*CollisionShape3D - BoxShape* oraz *MeshInstance3D - BoxMesh*). Aby scena była możliwa do zaobserwowania dodano kamerę *Camera3D*. Ze względu na zmiany w Godocie 4.0, należało manualnie dodać światło poprzez *Add Environment*, a następnie *Add Sun*.

Sześcianny zostały tak ustawione, aby sprawdzić fizykę i kolizję między obiektami.



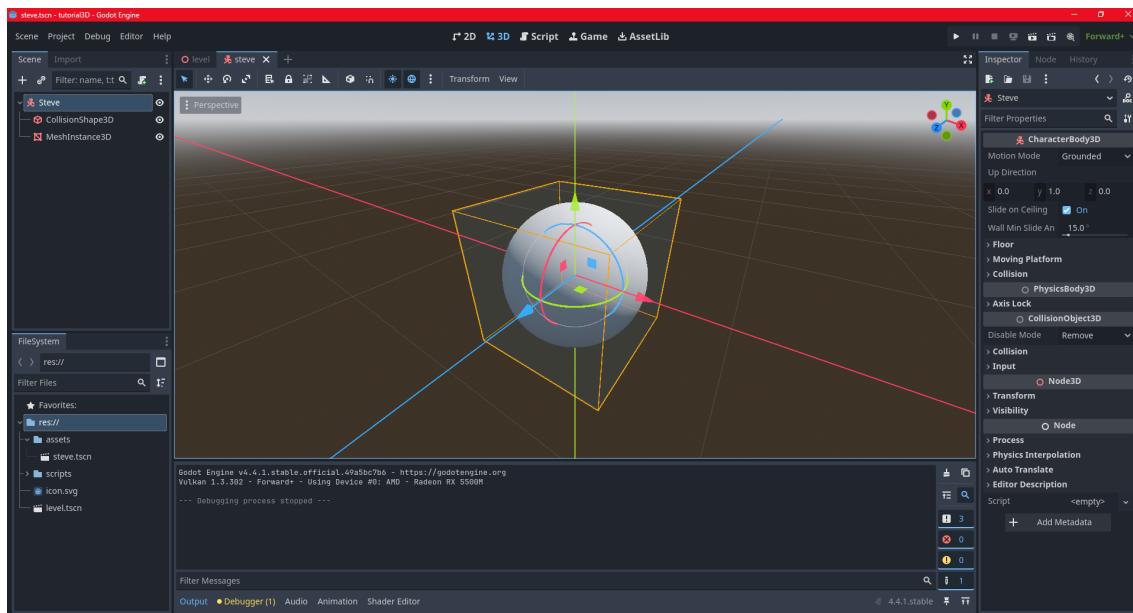
Rysunek 9: Scena.



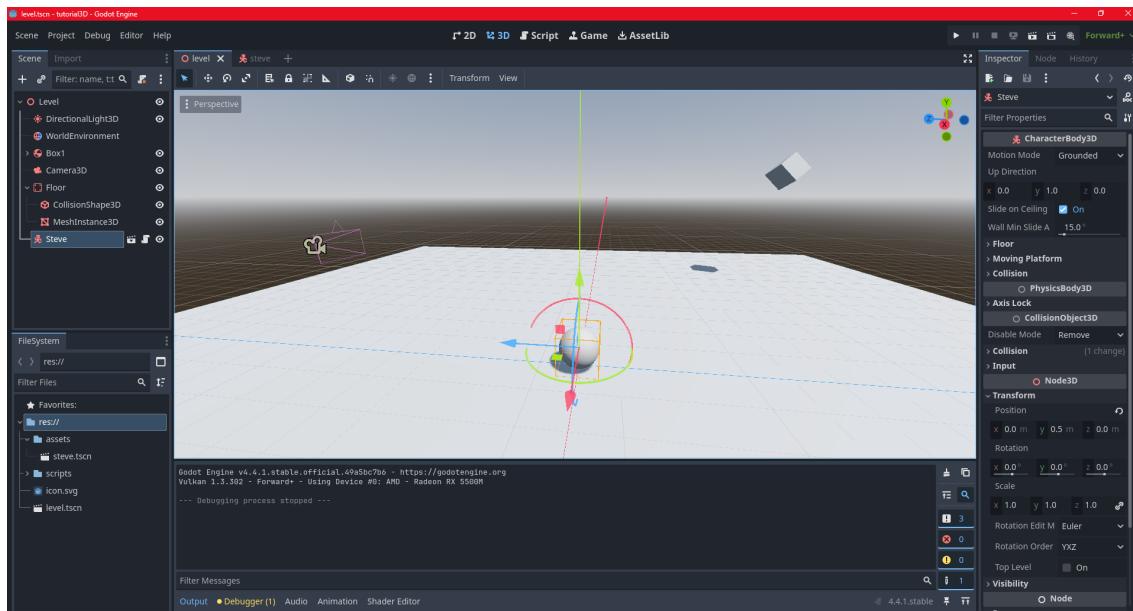
Rysunek 10: Fizyka kostek.

3.2 Dodanie gracza

Gracz steruje kulą klasy *CharacterBody3D* o nazwie *Steve*. Tak jak w przypadku kostek, dodano *CollisionShape3D* oraz *MeshInstance3D* z taką różnicą, że mesh jest o kształcie *SphereMesh*.



Rysunek 11: Scena gracza



Rysunek 12: Scena.

W celu implementacji poruszania się gracza, dodano skrypt *steve.gd*. Względem poradnika zmieniono kilka rzeczy:

- Pozbyto się *velocity = Vector3(0,0,0)* - *velocity* jest zmienną zdefiniowaną dla *CharacterBody3D*, czyli nie można definiować nowej zmiennej o tej samej nazwie.
- Zmieniono *move_and_slide(velocity)* na *move_and_slide()* - w Godocie 4 ta funkcja nie przyjmuje argumentów, pobiera wartości ze zdefiniowanego *velocity*.

```

1 extends CharacterBody3D
2
3 const SPEED = 5
4
5 func _ready():
6     pass
7
8 func _physics_process(delta: float) -> void:
9     if Input.is_action_pressed("ui_right") and Input.is_action_pressed("ui_left"):

```

```

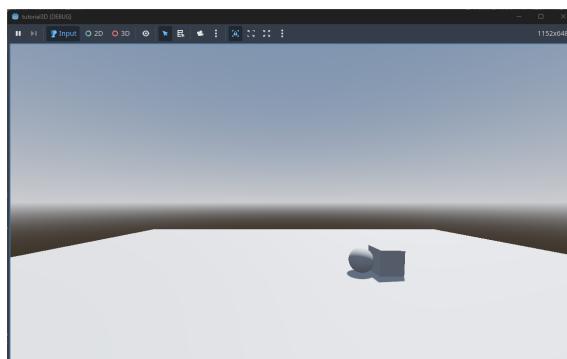
10     velocity.x = 0
11 elif Input.is_action_pressed("ui_right"):
12     velocity.x = SPEED
13 elif Input.is_action_pressed("ui_left"):
14     velocity.x = -SPEED
15 else:
16     velocity.x = lerp(velocity.x, 0.0, 0.1)
17
18 if Input.is_action_pressed("ui_up") and Input.is_action_pressed("ui_down"):
19     velocity.z = 0
20 elif Input.is_action_pressed("ui_up"):
21     velocity.z = -SPEED
22 elif Input.is_action_pressed("ui_down"):
23     velocity.z = SPEED
24 else:
25     velocity.z = lerp(velocity.z, 0.0, 0.1)
26
27 move_and_slide()

```

Listing 6: Kod gracza

Po uruchomieniu sceny okazało się, że kolizja kuli z kostką nie działa w zamierzony sposób – kula nie przesuwała kostki. Aby to naprawić, dzięki wiedzy z poradnika 2D, ustawiono:

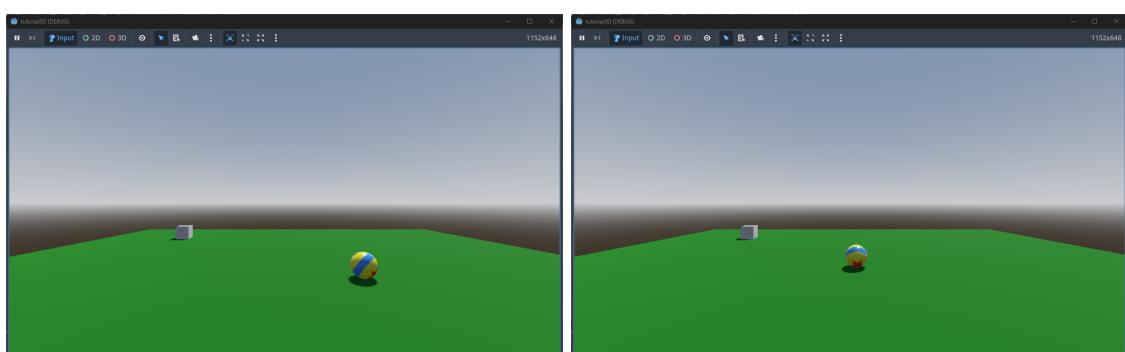
- Floor: w *Inspector*>*CollisoonObject3D*>*Colision* pozostawiono *Layer* na 1.
- Steve: w *Inspector*>*CollisoonObject3D*>*Colision* ustawiono *Layer* na 2, a *Mask* pozostawiono na 1 - kula jest w warstwie drugiej, koliduje z podłogą.
- Box: w *Inspector*>*CollisoonObject3D*>*Colision* ustawiono *Layer* na 4, a *Mask* ustawiono na 1, 2 i 4 - kostka jest w warstwie czwartej, koliduje z podłogą, kulą i innymi kostkami.



Rysunek 13: Przesuwanie kostki kulą.

3.3 Importowanie z Blender'a i rotacja kuli

Pobrano gotowy mesh z poradnika i zainportowano go do projektu. Ustawiono go jako mesh kuli. Dodatkowo zmieniono kolor podłogi. Zaimplementowano rotację piłki poprzez rotację samego mesh'a. Dodatkowo tak jak w poradniku 2D ustawiono akcje *move_left/right/up/down* i przypisano do nich strzałki i WSAD.



Rysunek 14: Rotacja kuli.

```

1 extends CharacterBody3D
2
3 const SPEED = 5
4 const ROTSPED = 7
5
6 func _ready():
7     pass
8
9 func _physics_process(delta: float) -> void:
10    if Input.is_action_pressed("move_right") and Input.is_action_pressed("move_left"):
11        :
12        velocity.x = 0
13    elif Input.is_action_pressed("move_right"):
14        velocity.x = SPEED
15        $MeshInstance3D.rotate_z(deg_to_rad(-ROTSPEED))
16    elif Input.is_action_pressed("move_left"):
17        velocity.x = -SPEED
18        $MeshInstance3D.rotate_z(deg_to_rad(ROTSPEED))
19    else:
20        velocity.x = lerp(velocity.x, 0.0, 0.1)
21
22    if Input.is_action_pressed("move_up") and Input.is_action_pressed("move_down"):
23        velocity.z = 0
24    elif Input.is_action_pressed("move_up"):
25        velocity.z = -SPEED
26        $MeshInstance3D.rotate_x(deg_to_rad(-ROTSPEED))
27    elif Input.is_action_pressed("move_down"):
28        velocity.z = SPEED
29        $MeshInstance3D.rotate_x(deg_to_rad(ROTSPEED))
30    else:
31        velocity.z = lerp(velocity.z, 0.0, 0.1)
32
32 move_and_slide()

```

Listing 7: Kod gracza

3.4 Kamera, światło i środowisko

Stworzono kamerę, która podąża za kulą. Ze względu na usunięcie *InterpolatedCamera* w Go docie 4, stworzono stworzono zwykłą kamerę *Camera3D* i skrypt imitujący *InterpolatedCamera*.

```

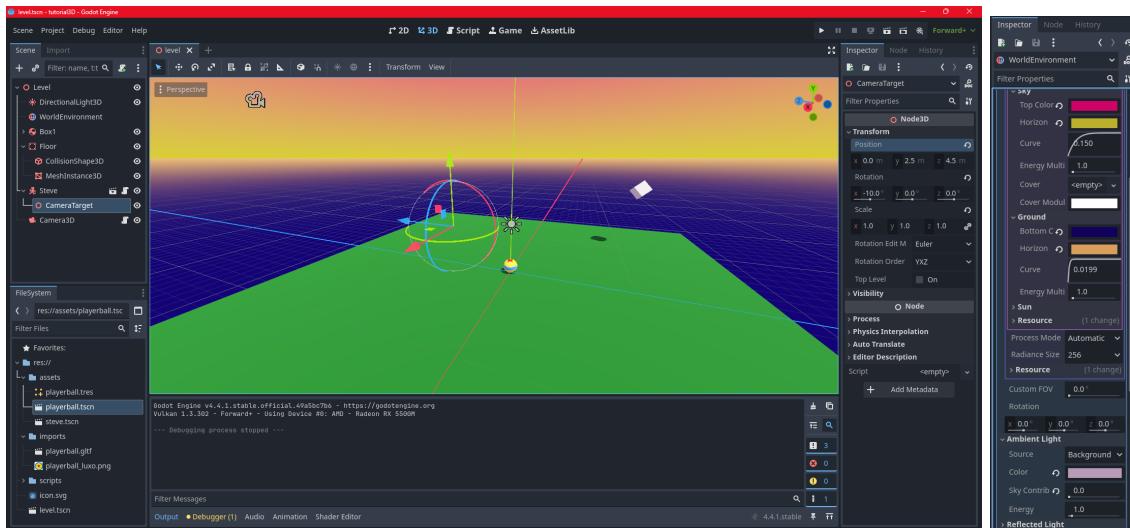
1 extends Camera3D
2 @export var target: NodePath
3 @export var speed := 1.0
4 @export var enabled: bool
5
6 func _process(delta: float) -> void:
7    var target_node := get_node(target) as Node3D
8    if not enabled or not target_node:
9        return
10
11    global_transform = global_transform.interpolate_with(
12        target_node.global_transform, speed * delta)

```

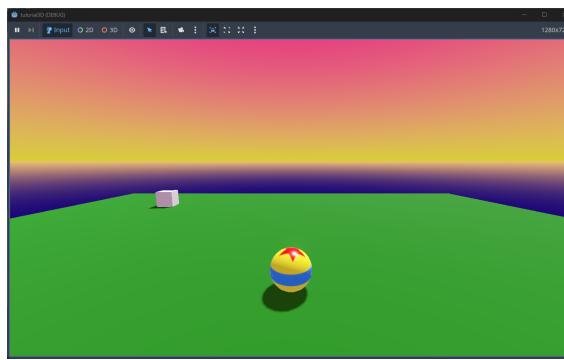
Listing 8: Kod kamery

Dzięki temu w Inspektorze kamery można wybrać cel oraz prędkość. Celem kamery jest *CameraTarget* klasy *Node3D*, który ustawiony jest przed kulą. Kamera znajduje się w oddali, aby podczas uruchomienia gry była widoczna cała scena.

Zmodyfikowano *WorldEnvironment* i zmieniono kolor nieba oraz światła.



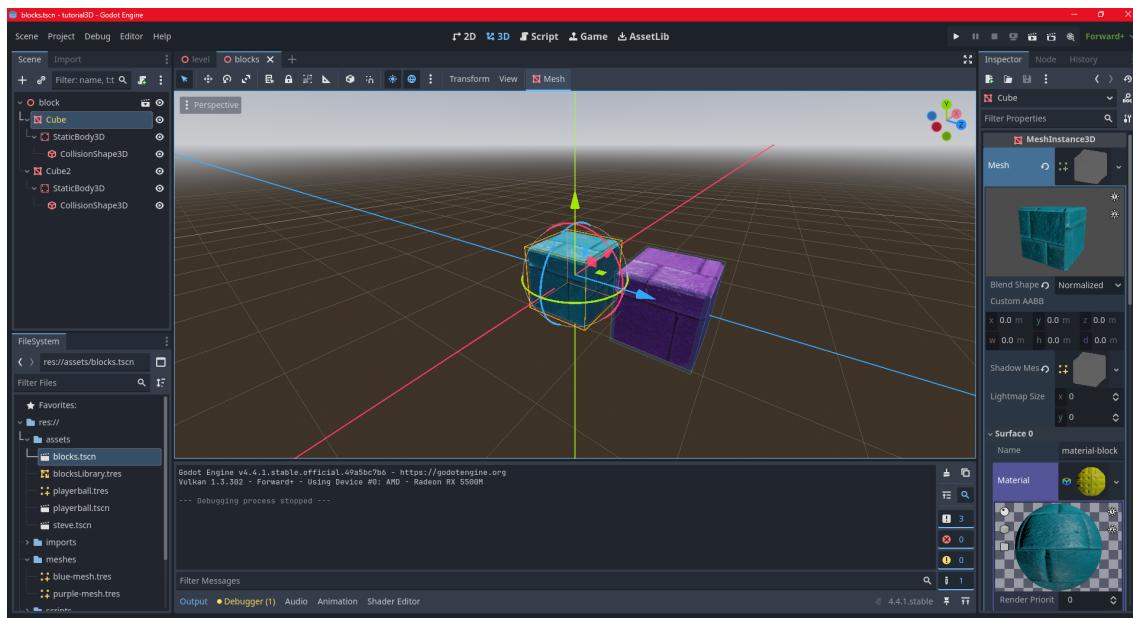
Rysunek 15: Scena.



Rysunek 16: Niebo.

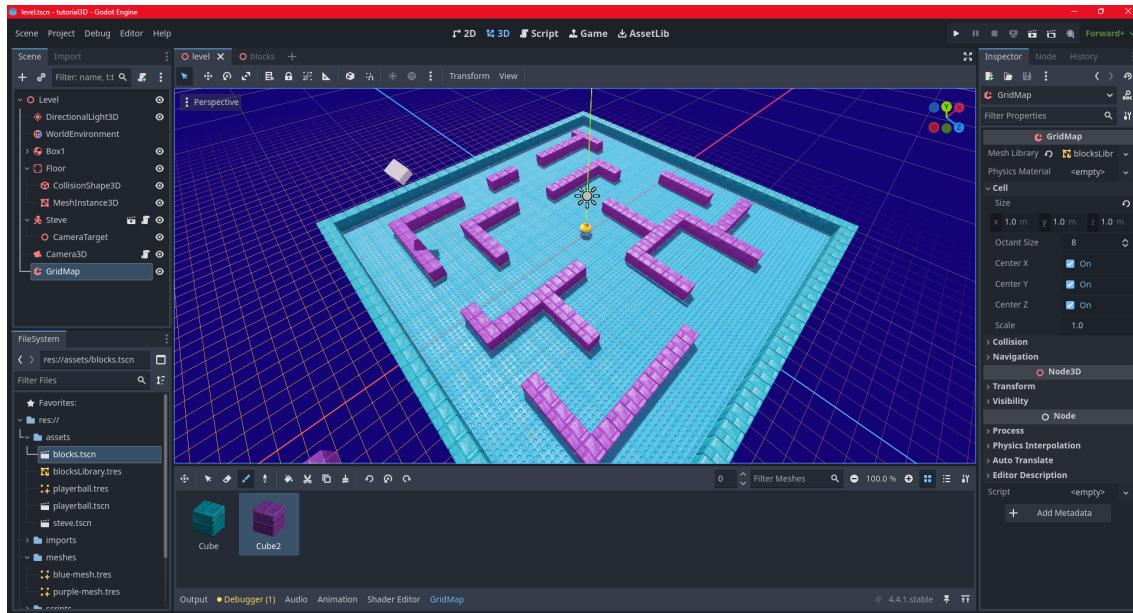
3.5 Tekstury i ściany

Na podłodze ustalone zainportowaną teksturę klocków Lego. Następnie zainportowano klocki, z którego będą składać się ściany. Zapisano jego mesh oraz zmieniono jego kolor na niebieski. Skopiowano ten mesh i zmieniono kolor na fioletowy. Dodano dwa klocki o tych kolorach do sceny i wyeksportowano ją jako *MeshLibrary*.



Rysunek 17: Mesh Library.

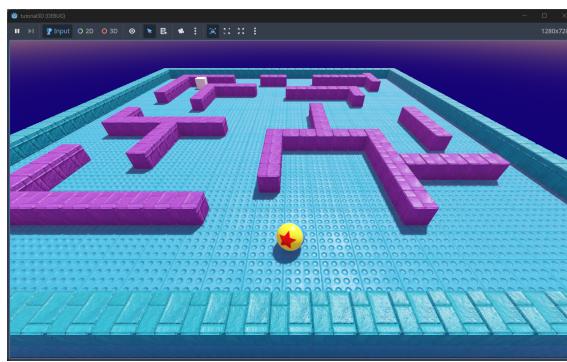
Do sceny głównej dodano *GridMap*, której ustawiono *blocksLibrary.tres* jako *Mesh Library*. Ustawiono te klocki na mapie. Dodano im cienie w *WorldEnvironment>SSAO>Enabled* oraz *Light Affect*.



Rysunek 18: Dodawanie ścian na *GridMap*.



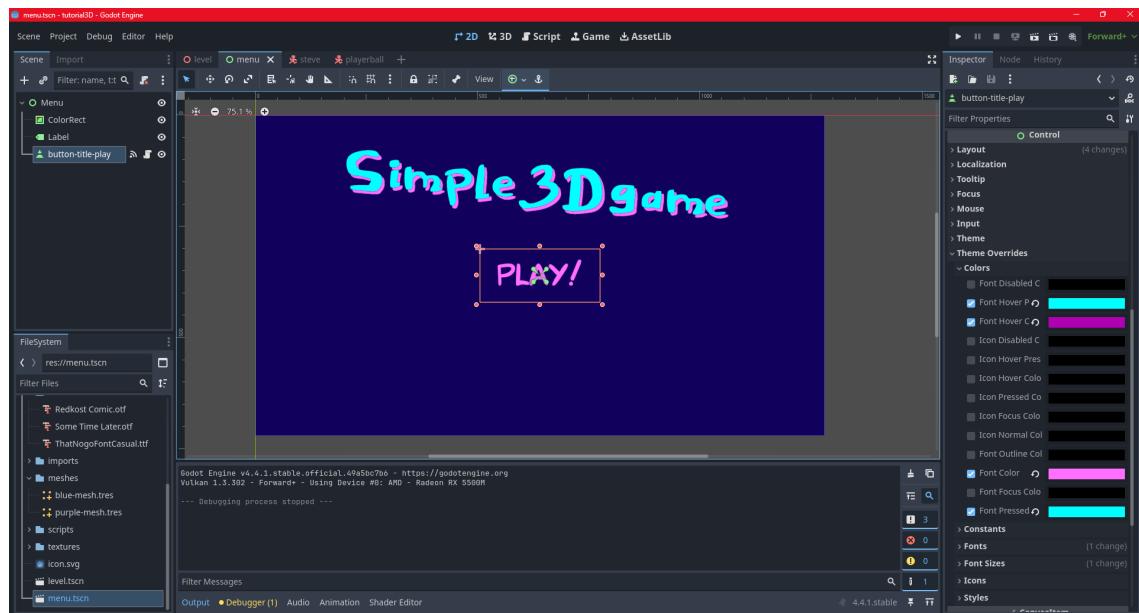
Rysunek 19: Układ poziomu.



Rysunek 20: Scena.

3.6 Menu

Dodano proste menu z jednym przyciskiem rozpoczynającym grę. Proces tworzenia jest taki sam jak w przypadku menu z poradnika 2D.



Rysunek 21: Menu.

```

1 extends Button
2
3 func _ready() -> void:
4     pass
5
6 func _on_pressed() -> void:
7     get_tree().change_scene_to_file("res://level.tscn")

```

Listing 9: Kod przycisku

3.7 Dodanie przeciwników

Zimportowano plik *enemy.gltf*. Zmieniono główny węzeł na *Area2D* o nazwie *Enemy*. Dodano *CollisionShape3D* o kształcie *SphereShape3D*. Dodano do sceny jednego przeciwnika. Podpięto sygnał *_on_enemy_body_entered()* w celu przekazania informacji o kolizji oraz zakończeniu gry. Funkcja została zdefiniowana w *steve.gd*, czyli skrypcie kuli.

Stworzono ekran *game_over.tscn*. Zmiana na tę scenę wywoływana jest w *_on_enemy_body_entered()*.



Rysunek 22: Ekran Game Over.

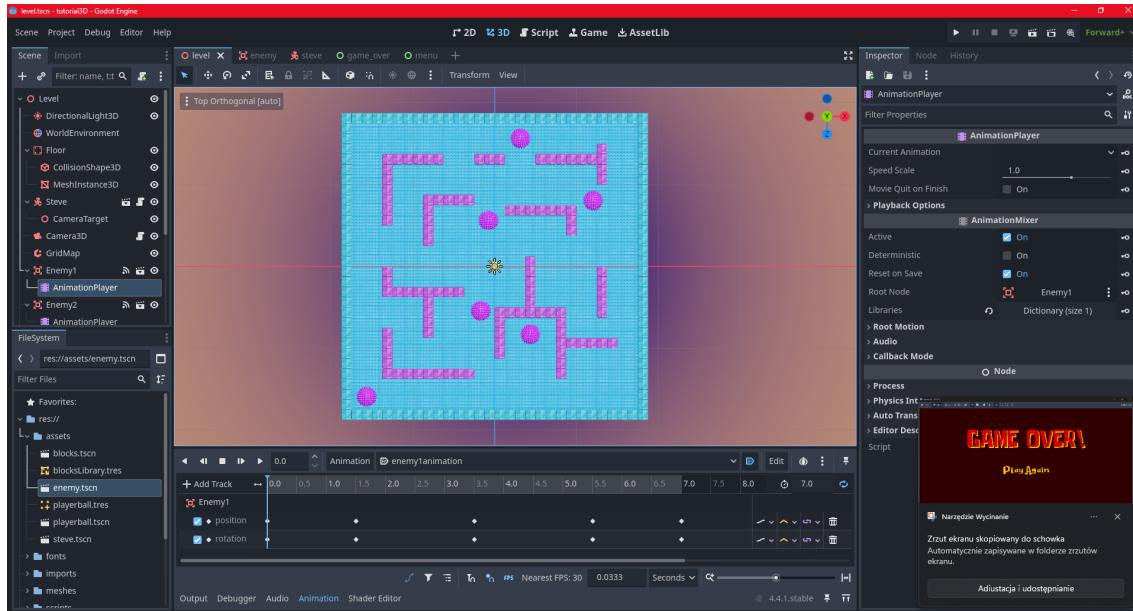
```

1 func _on_enemy_body_entered(body: Node3D) -> void:
2     if body == self:
3         get_tree().change_scene_to_file("res://game_over.tscn")

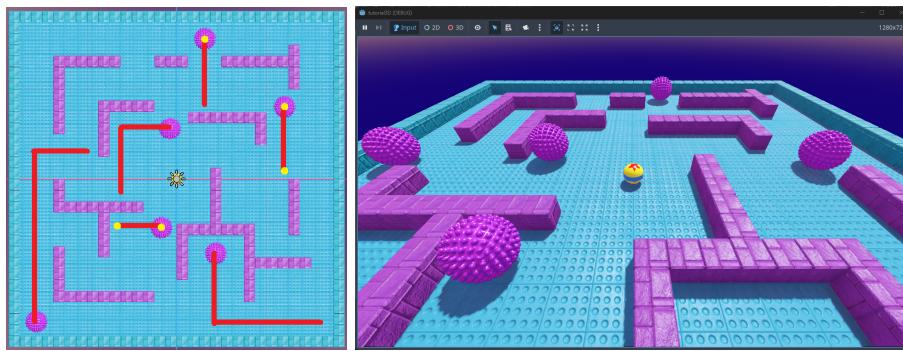
```

Listing 10: Kod zakończenia gry

Do sceny poziomu dodano sześciu przeciwników - każdemu dodano *AnimationPlayer* i przypisano unikatowe animacje. Na rysunku 24 kolorem czerwonym zaznaczono tor przeciwników. Żółta kropka oznacza, że przeciwnik zatrzymuje się na krótki czas w tym miejscu.



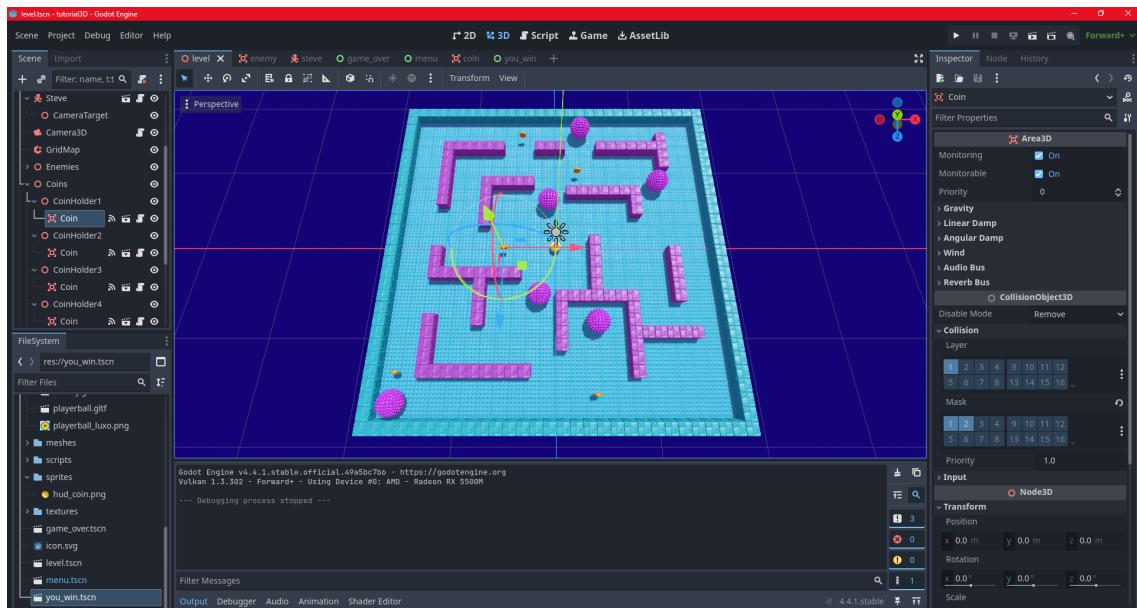
Rysunek 23: Scena.



Rysunek 24: Animacje przeciwników.

3.8 Dodanie monet i HUDu

Na początek zainportowano *coin.gltf* oraz zrobiono nową scenę *coin.tscn*. Dodano *CollisionShape3D* o kształcie *CylinderShape3D* oraz Timer o czasie 0.4s - po upływie czasu moneta zostanie usunięta. Dodatkowo dodano *AnimationPlayer* z animacją podnoszenia monety. Następnie do sceny poziomu dodano *Node3D* - *Coins*, a jako jego dziecko dodano *Node3D* - *CoinHolder1* i do niego podpięto scenę monety. Zduplikowano *CoinHolder* 4 razy. Ustawiono monety na mapie. Dodano skrypt w scenie monety *coin.gd*, gdzie dodano funkcję zbierania monet oraz emisję sygnału o zebranej monecie po upływie czasu.



Rysunek 25: Scena.

```

1  extends Area3D
2
3  signal coinCollected
4
5  func _ready():
6      pass
7
8
9  func _physics_process(_delta: float) -> void:
10     rotate_y(deg_to_rad(3))
11
12
13 func _on_body_entered(body: Node3D) -> void:
14     if body.name == "Steve":
15         $AnimationPlayer.play("bounce")
16         $Timer.start()
17
18 func _on_timer_timeout() -> void:

```

```

20     emit_signal("coinCollected")
21     queue_free()

```

Listing 11: Kod monety

Dodano HUD do głównej sceny, który wyświetla ilość zdobytych monet. Składa się z:

- *Sprite2D* - zdjęcie monety,
- *Label* - "X",
- *Counter* - wyświetla ilość zdobytych monet,
 - *Timer* - po zdobyciu 5 monet *Timer* jest włączany. Po upłynięciu 1s wyświetla się ekran "You Win!".

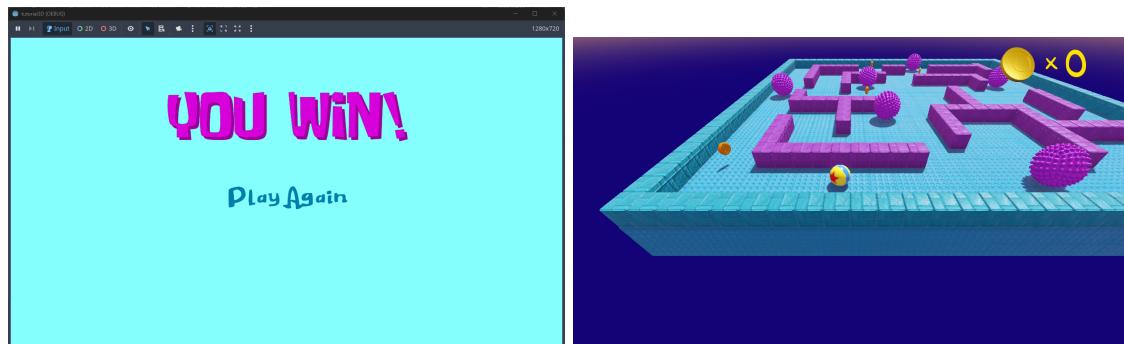
```

1 extends Label
2
3 var coins = 0
4
5 func _ready():
6     text = str(coins)
7
8
9 func _on_coin_collected() -> void:
10    coins = coins + 1
11    _ready()
12    if coins == 5:
13        $Timer.start()
14
15
16 func _on_timer_timeout() -> void:
17     get_tree().change_scene_to_file.call_deferred("res://you_win.tscn")

```

Listing 12: Kod licznika

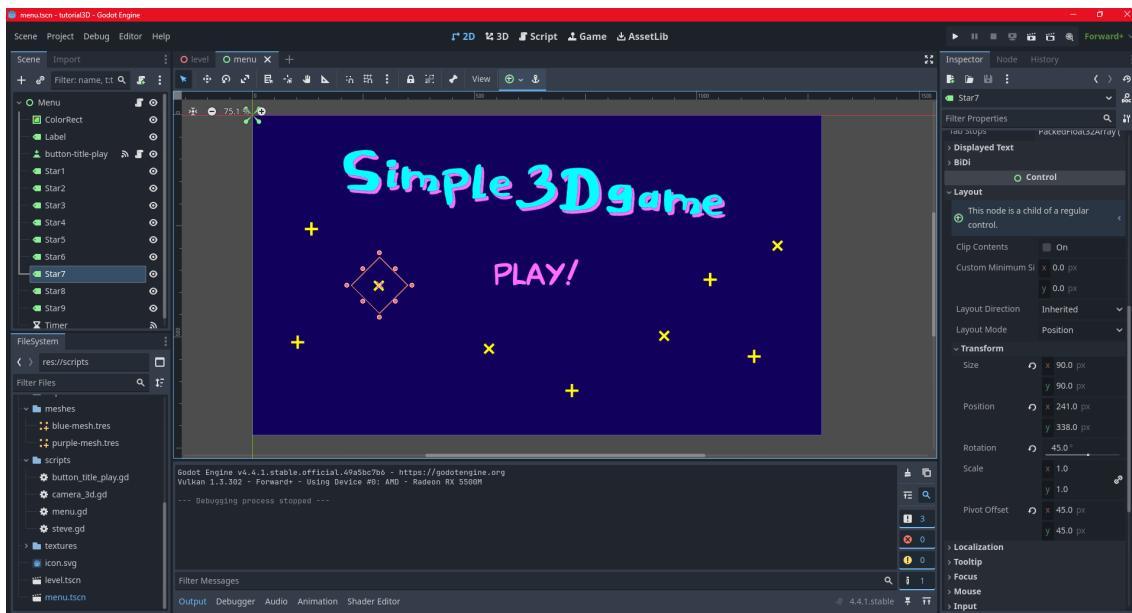
Utworzono scenę *you_win.tscn*.



Rysunek 26: Skończona gra.

3.9 Dodatki

Dodano do menu gwiazdy i ich animację poprzez rotację o 45 stopni. Każda gwiazda to *Label*, którego tekst to "+". Niektóre gwiazdy na początku już są obrócone o 45 stopni. Dodano *Timer*, który wywołuje *timeout()* co 0.5s. Dodano skrypt do węzła *Menu*, *YouWin* i *GameOver*.



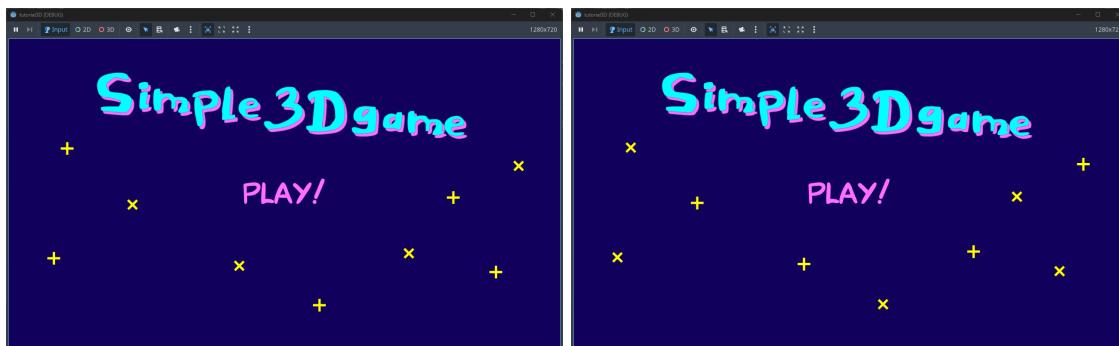
Rysunek 27: Menu z gwiazdami.

```

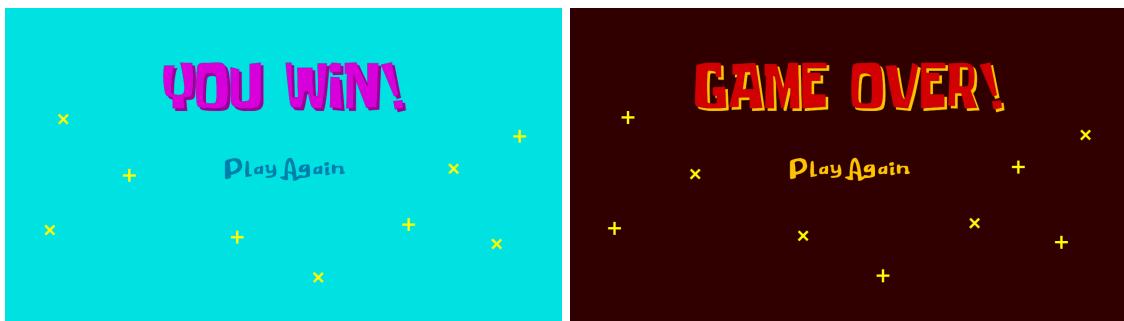
1 extends Control
2
3 func _ready():
4     $Timer.start()
5
6 func _on_timer_timeout():
7     for i in range(1, 10):
8         var label_name = "Star" + str(i)
9         var label_node = get_node(label_name)
10        label_node.rotation_degrees += 45
11
12        if label_node.rotation_degrees >= 360:
13            label_node.rotation_degrees = 0

```

Listing 13: Kod Menu



Rysunek 28: Menu z animacją gwiazd.



Rysunek 29: Pozostałe menu z animacją gwiazd.

Literatura

- Poradnik 2D [https://docs.godotengine.org/en/stable/getting_started/first_2d_game/index.html]
- Poradnik 3D [https://www.youtube.com/playlist?list=PLda3VoSoc_TSBB0BYwcmlamF1UrjVtccZ]