Документација по предметот Тимски Проект

Создавање и имплементација на паркур систем базиран на издржливост (stamina) во 2Д платформер игра во Godot.

Членови:

Мартин Јаневски – 185016
Симон Мирчевски – 181207
Филип Стефановски – 186102

Линк до проектот на Github: https://github.com/MartinJanevski185016/TimskiProekt

1. Информации за софтверот

Пред да се започне со документација на самата игра, во прилог има неколку информации за самиот софвер Godot и кои се сите бенефити од негово користење:

- Godot е целосно бесплатен за користење, без лиценцни такси или провизии што го прави достапен за инди девелопери во мали тимови како нашиот.
- Самиот софтвер е лесен, со мала големина, и може да работи на различни платформи. Оваа ефикасност овозможува развој на игри без потреба од моќен хардвер. Дури и на некои помали проекти може да се променува кодот симултано со пуштена инстанца од игра во позадина.
- Godot има доста корисни додатни класи и алатки за создавање на 2Д игри што би биле споредливи со системот развиен во Unity. Исто така, поддржува развој на 3Д игри иако не е толку напреден во споредба со 2Д.
- Во Godot секој елемент во играта е сведен на структура наречена јазол (node), и овие јазли можат лесно да се комбинираат и организираат за да се создадат сцени и нивоа. Ова ја намалува комплексноста на самите игри – доколку се промени кодот во еден јазол нема да влијае на другите компоненти од играта.

- Godot користи свој јазик за креирање на скрипти наречен GDScript - сличен на Python. Самиот јазик е лесен за учење и нема многу правила кои треба да се запазат. Исто така ги поддржува и C#, C++ и VisualScript за тие што преферираат други програмски јазици.

- На игри креирани со Godot може да им се направи експорт во повеќе платформи вклучувајќи ги Windows, macOS, Linux, Android, iOS, и веб прелистувачи (доколку сакате да создадете веб-базирана игра). Самата интеграција со овие платформи лесно се имплементира во игрите, дури има и детални туторијали за целиот процес.

- Godot има активна заедница која придонесува за неговиот развој, обемна документација и голем број на онлајн туторијали, што го прави уште полесен за учење и за добивање на поддршка доколку се заглави во создавањето на сегмент од игрите.

- За разлика од софтвери како Unreal, Godot не наплаќа провизии на игрите што ги креирате, овозможувајќи повеќе финансиска слобода за девелоперите.

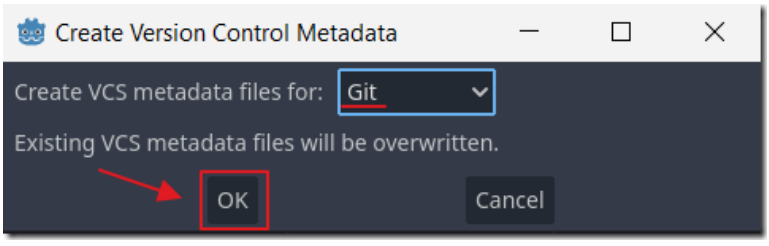
- Godot добива чести ажурирања и подобрувања од заедницата, при што со секоја нова верзија има некоја нова корисна алатка што го упростува процесот на правење на игри.

Сумаризирано во кратки црти, во споредба со други софтвери за создавање на игри како Unity и Unreal, Godot се истакнува во развојот на 2D игри и нуди поголема флексибилност со помали трошоци - што го прави посебно привлечен за мали тимови и инди девелопери.

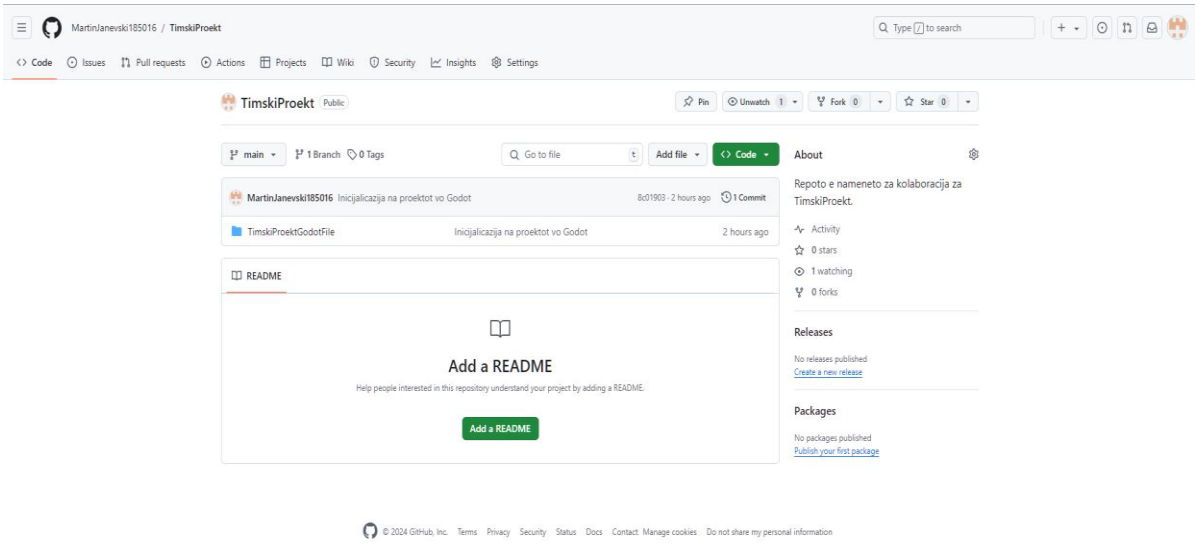
2. Верзионирање и изработка во тим

Годот има вграден систем за верзионирање и зачувување на развојот на самите игри/кодот во скриптите преку Git.

При креирање на самиот проект може да се имплементира верзионирање преку самото IDE на Годот. Истиот проект преку оваа функционалност беше прикачен на GitHub за колаборација со другите членови тимот.



На следниов линк има корисен туторијал со кој што може да се додаде алатка за пуштање на верзии на кодот кон репозиториумот на Github, иако некои членови преферираа директно користење на Git Bash наспроти интерфејс.



Почетниот изглед на репото при креирање на празен проект во Годот

3. План за создавање на игра во Годот

Планот за содавање на играта е поделен во неколку главни сегменти карактеристични на платформер игри:

1. Создавање на функционален главен карактер со мапирани движења од типот:

    a. Движење во сите насоки (со додадена физика на успорување што би го направило природно)
    b. Скок
    c. Двоен скок
    d. Скок во воздух при паѓање од платформа (coyote jump)
    e. Скок од ѕид и одбивање од ѕидови
    f. Лизгање при држење на ѕид
    g. Спринт (нагло движење на земја за одбегнување на стапици)
    h. Клекнување под платформи и движење во клекната позиција

2. Создавање на нивоа со кои ќе се прикажат истите движења на карактерот:

    a. Физички тела кои треба да се избегнат со скок/двоен скок или скок од ѕид
    b. Тела кои треба да се поминат со клекнување или користење на спринт
    c. Проектили кои треба да се одбегнат со временско планирање на скок/лизгање од ѕид или користење на спринт
    d. Создавање на препреки кои ќе го натераат карактерот да изгуби во играта

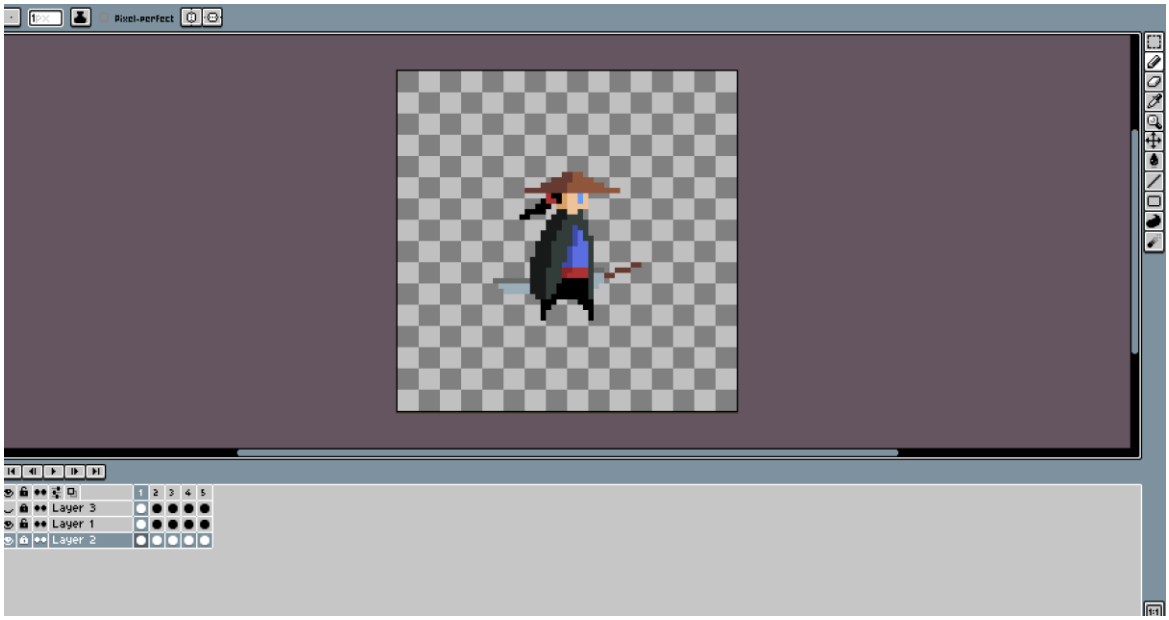
3. Создавање на систем за стамина на карактерот

По создавањето на самите движења на карактерот и нивоата во играта би дошол поволниот момент за создавање на таканаречен 'stamina' систем или со други зборови систем на издржливост на движење на карактерот. Овој систем би го лимитирал бројот на движења што може да ги направи играчот во одреден временски период со цел паметно да ги планира движењата и потешки да се препреките на играчот.

4. Создавање на карактерот и позадините

Сите позадини, препреки и анимации поврзани со карактерот се нацртани од страна на Мартин Јаневски во програма наречена Asperite.

Asperite е програма наменета специфично за цртање и анимација на карактери и позадини во видеоигри со 8-бит/пикселиран изглед.



Пример од интерфејсот на апликацијата Asperite

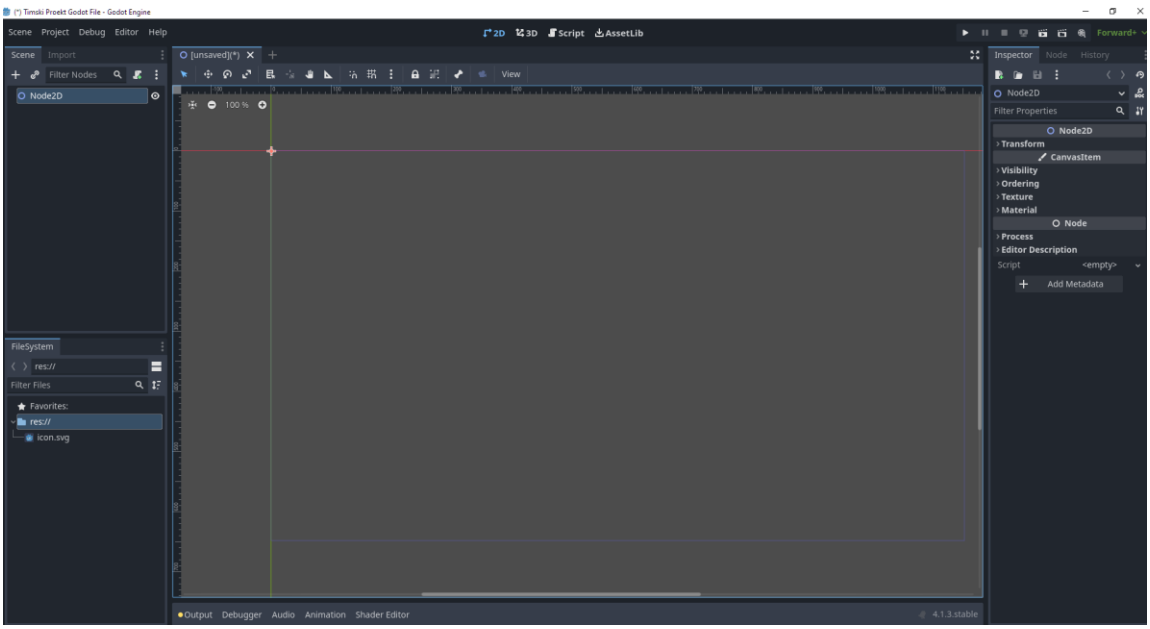
5. Документација на процесот на креирање на играта

1. Создавање на сцени во Годот

Сцените во Годот се клучен дел од развојот на самите проекти. Тие се збир од јазли организирани во структура на дрво, што претставува еден елемент од самата игра. Со помош на оваа функционалност, сцените за главниот лик, околина или кориснички интерфејс можат повторно да се користат и да се вметнуваат во други сцени – како различни нивоа.

Тоа овозможува лесно креирање сложени проекти со комбинирање на помали, управливи делови. Овој модуларен пристап ја подобрува организацијата и го забрзува развојот на игрите.

Доколку создадеме нов празен проект во Годот непроменетиот интерфејс со почетен 2Д јазол ни изгледа како на наведената слика:

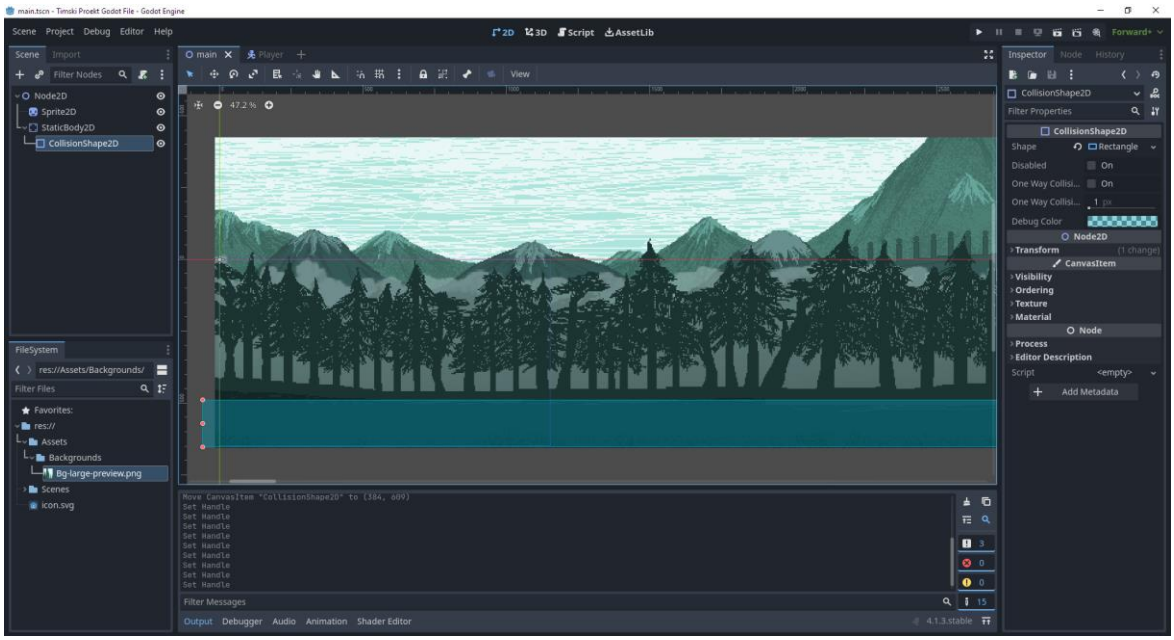


Додаваме две различни сцени, едната за самото ниво во играта (main scene) во кое ќе се придвижува карактерот и друга за создавање на самиот карактер (Player scene).

Во иднина ќе го инстанцираме карактерот во нивото за играта и така ќе проверуваме секоја функционалност на карактерот дали работи како испланирано.

Во сцената за нивото додаваме два јазли од тип Sprite2D (јазол со слика за додавање на позадина) и StaticBody2D (јазол за додавање на под со интегрирана физика) кој понатаму ќе има интеракција со карактерот. Потоа на тој StaticBody2D јазол ќе му додадеме CollisionShape2D јазол во форма на правоаголник кој ќе го покрива долниот дел на сликата и ќе претставува под на играта.

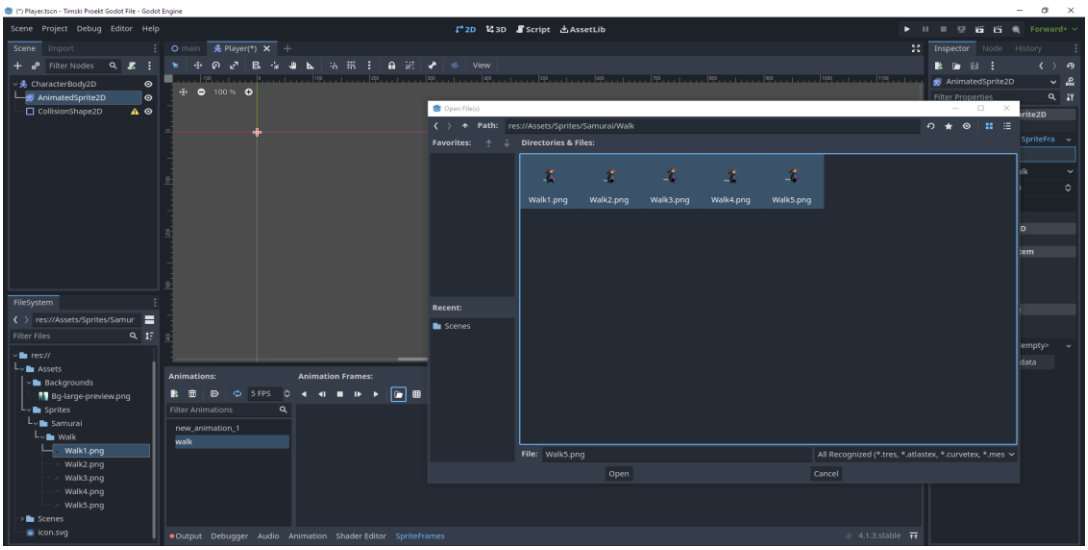
Откако ќе ги додадеме формата на нашиот под и позадинската слика во фолдер за позадини, сцената ќе ни изгледа како наведено:



За сцената со играчот додаваме неколку типови на јазли со различни функционалности:

- CharacterBody2D – има своја почетна физика и гравитација и е наменет за карактери кои ќе ги контролира самиот играч
- AnimatedSprite2D – се користи за интеграција на анимации со движења предизвикани од тастатура
- CollisionShape2D – физичка форма која ја користи карактерот за интеракција со други површини

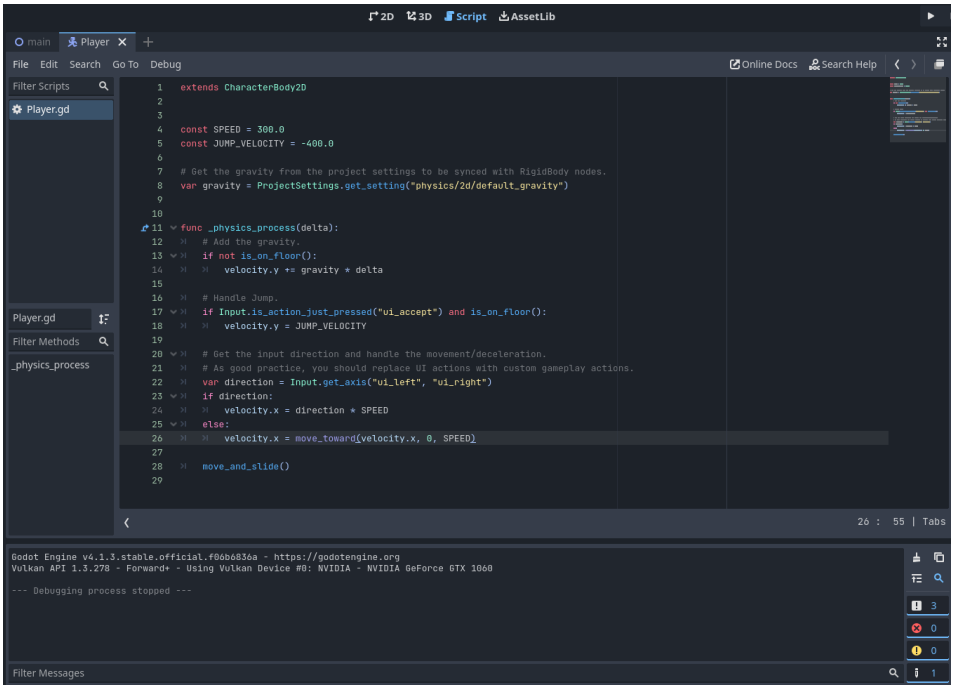
Ја додаваме формата на карактерот (во нашиов случај капсула) и анимациите за придвижување на карактерот (ставени во посебен фолдер) во самата игра. Треба да ја импортираме секоја анимација со посебно име за да може понатаму да се искористат во скриптата. Пример за како да го направиме тоа ќе покажеме со импортирање на walk анимацијата. За почеток треба во AnimatedSprite2D јазолот да одбереме SpriteFrames како тип на анимација и да додадеме нова анимација со име walk. Во таа анимација ќе ги импортираме сликите од нашиот карактер и ќе ги наместиме по редослед.



На сликата е прикажан пример со прозорецот за импортирање на анимации

2. Создавање на скриптата на карактерот преку код

За да започнеме со кодирање на движењата на самиот карактер мора да креираме нова скрипта и да ја поврземе со неговата сцена. При креирање на оваа скрипта Годот има почетен код за скок и движење на тој CharacterBody2D јазол.



Почетна скрипта генерирана за CharacterBody2D јазол во Годот

Структурата на кодот која ќе се запазува при создавање на движења на карактерот преку скрипта е следна:

- Сите функции за придвижување на карактерот ќе бидат одделени и ќе се повикуваат во главната функција за процесирање (што се обновува на секоја секунда) и се нарекува **_physics_process(delta).**

- Доколку треба да се направи интеракција помеѓу дете и родител јазол (инстанца на карактер во сцената на позадина како пример) ќе се користи 'call down, signal up' принципот. Тоа подразбира повикување на јазол кој е подолу во хиерархијата преку функција за повик, но повикување на родител-јазол преку сигнали (посебни структури во Годот користени за интеракција при некое движење, како клик на копче).
- Анимациите ќе се процесираат во посебна функција со цел полесно да се анализира транзицијата помеѓу движења, наспроти барање на секоја анимација во секоја функција за движење одделно.

Запазувајќи ја оваа структура, ги одделуваме движењата на карактерот во посебни функции кои ќе ги крстиме:

- apply_gravity() – Доколку карактерот не е на под или рамна површина под неговото тело, се нанесува на него силата на гравитација.

- handle_jump() – Доколку играчот го кликне копчето за скок карактерот се придвижува по Y оска со одредена брзина која може да се конфигурира.

- handle_wall_jump() – Доколку играчот е допрен до површина која се смета за ѕид и го кликне копчето за скок заедно со насоката на движење карактерот скока во таа насока и се одбива од ѕидот

- handle_acceleration() –доколку играчот ги кликне копчињата за движење лево или десно карактерот се придвижува по X оска во насоката со одредена брзина, при што се користи вградена функција од Годот наречена move_toward() за движењето да се зголемува до одредена брзина. Со помош на таа додатна функција, движењето на карактерот изгледа поприродно.

- handle_air_acceleration() – слично на handle_acceleration() функцијата, оваа функција се користи за забрзувањето на карактерот во воздух при скок да изгледа поприродно.

- apply_friction() – Ако карактерот е во движење лево или десно оваа функција го успорува после одредено време до 0 брзина.

- apply_air_resistance() – Функцијата служи за успорување на движењето слично на apply_friction() функцијата, но нанесено во воздух.

- handle_dash() – нагло и кратко хоризонтално движење на карактерот во насока лево или десно

- handle_wall_hold() – Лизгањето надолу по ѕидови се успорува доколку играчот ја држи насоката на движење прицврстен за ѕид. Целта на таа функционалност е да му обезбеди доволно време на играчот да скокне/се придвижи до друга површина.

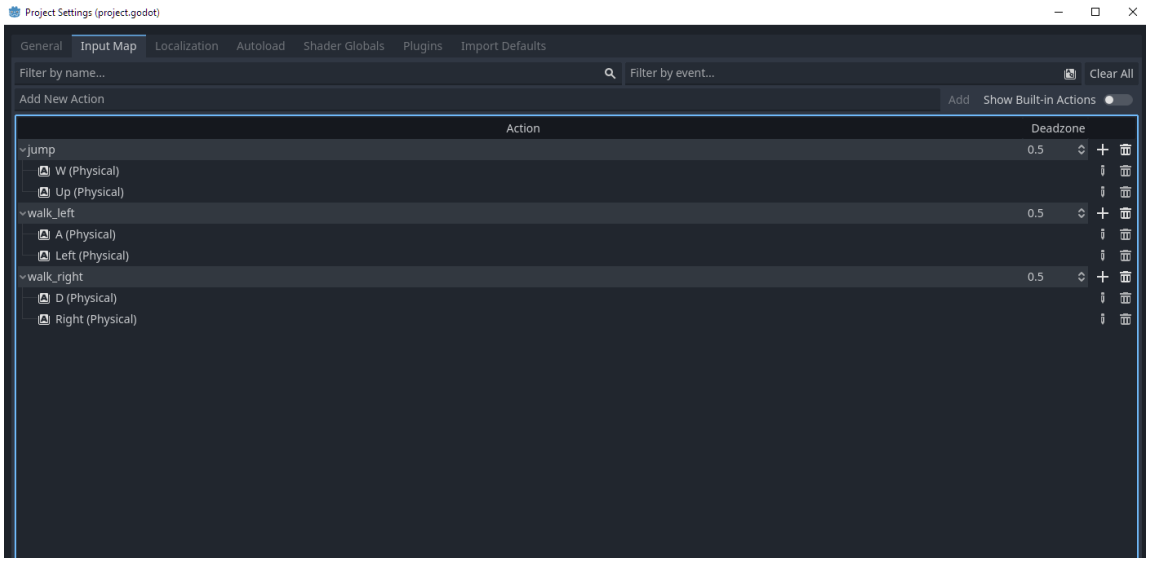
- handle_crouch() – Со помош на функцијата карактерот клекнува доколку се кликне и држи копчето за клечење. Функцијата исто така ја намалува формата за колизии на карактерот додека е во клекната положба

- attempt_uncrouch() – доколку карактерот се наоѓа во клекната позиција иако играчот го ослободи копчето за клекнување, фукнцијата го задржува карактерот клекнат се додека не се ослободи простор над неговата глава за исправување

Потоа сите ги повикуваме во главниот _physics_process(delta), со тоа што тие се извршуваат при секој нов frame. Тоа овозможува анимацијата и физичкото движење на карактерот да се промени точно во моментот коа ке има промена на состојбата.

Вклучувањето на анимациите во самите движења е направено во посебна нова функција наречена update_animations(), при што се овозможува полесно планирање на транзициите помеѓу секое движење.

За да функционира кодот со новите копчиња, наместо стандарните мапирања на копчиња ќе додадеме нови преку кликање на Project > Input Map > Add new Action:



На сликата е прикажан интерфејс за мапирање на нови движења преку копчиња од тастатура

Сите функционалности во кои е потребен некој тип на временски интервал, како на пример истекување на одредено време за да заврши одредено движење, се имплементирани преку Тајмери.

Тајмер е класа во Годот која има атрибути за подесување на временски интервали. Содржи голем број на променливи како начин на повторување, фрекфенција на извршување (во зависност во кој процес би се вметнал тајмерот).
Во класата за играчот има два тајмери – CoyoteJumpTimer и DashTimer

CoyoteJumpTimer е временски интервал искористен во функцијата за скок. Доколку играчот падне од било која пратформа, во скриптата се активира тајмерот при што на играчот му е дозволено да скокне иако не е на земја во наредните 0.2 секунди.

DashTimer се користи за одредување на времето колку долго играчот ќе се движи забрзано при кликање на компчето за забрзано движење.

Системот на издржливост (стамина) е имплементиран во посебна класа и инстанциран во главното ниво, како и карактерот.

Во самата класа на карактерот има референца до скриптата на системот за издржвост, при што при секое движење на карактерот (доколку има доволно стамина да го изврши) се одзема одредена стамина вредност

low_stamina_player() функцијата се користи за прикажување на визуелен ииндикатор дека карактерот нема доволно стамина за извршување на наредното движење.

Во прилог е прикажан целиот код за карактерот после сите промени:

```
extends CharacterBody2D



const SPEED = 500.0
const JUMP_VELOCITY = -600.0
const ACCELERATION = 1200.0
const AIR_ACCELERATION = 600.0
const FRICTION = 2000.0
const AIR_RESISTANCE = 1500.0
var WALL_SLIDE = 150.0
var air_jump = false
var just_wall_jumped = false
var is_crouching = false
var dash_orientation
var color = self.modulate
var original_scale = scale

var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

@onready var animated_sprite_2d = $AnimatedSprite2D
@onready var coyote_jump_timer = $"CoyoteJump Timer"
@onready var dash_timer = $"Dash Timer"
@onready var stamina_bar = $"../CanvasLayer/StaminaBar"
@onready var player_shape = $PlayerShapeUncrouched
@onready var player_shape_crouched = $PlayerShapeCrouched
@onready var shape_cast_2d = $ShapeCast2D

func _physics_process(delta):
	apply_gravity(delta)
	handle_wall_jump()
```

```
		handle_jump()
		var input_axis = Input.get_axis("walk_left", "walk_right")
		handle_crouch()
		handle_dash(input_axis)
		handle_acceleration(input_axis, delta)
		handle_air_acceleration(input_axis, delta)
		handle_wall_hold(delta)
		apply_friction(input_axis, delta)
		apply_air_resistance(input_axis, delta)
		update_animations(input_axis)
		var was_on_floor = is_on_floor()
		move_and_slide()
		var just_left_ledge = was_on_floor and not is_on_floor() and velocity.y >= 0
		if just_left_ledge:
			coyote_jump_timer.start()
		just_wall_jumped = false


func apply_gravity(delta):
		if not is_on_floor():
			velocity.y += gravity * delta


func handle_dash(input_axis):
		if dash_timer.is_stopped() and input_axis != 0 and (Input.is_action_just_pressed("dash_left") or
Input.is_action_just_pressed("dash_right")) and not stamina_bar.is_exhausted(stamina_bar.dash_drain):
			stamina_bar.calculate_stamina(stamina_bar.dash_drain)
			velocity.x = 1000 * input_axis
			dash_timer.start()
		elif((Input.is_action_just_pressed("dash_left") or Input.is_action_just_pressed("dash_right")) and
stamina_bar.is_exhausted(stamina_bar.dash_drain)):
			stamina_bar.low_stamina_effect()
			low_stamina_player()
```

```
func handle_wall_jump():
	if not is_on_wall_only(): return
	var wall_normal = get_wall_normal()
	if Input.is_action_just_pressed("jump") and is_on_wall_only() and not stamina_bar.is_exhausted(stamina_bar.wall_jump_drain):
		stamina_bar.calculate_stamina(stamina_bar.wall_jump_drain)
		velocity.x = wall_normal.x * SPEED
		velocity.y = JUMP_VELOCITY
		just_wall_jumped = true
	elif(Input.is_action_just_pressed("jump") and stamina_bar.is_exhausted(stamina_bar.wall_jump_drain)):
		stamina_bar.low_stamina_effect()
		low_stamina_player()


func handle_jump():
	if is_on_floor() or coyote_jump_timer.time_left > 0.0:
		air_jump = true
		if Input.is_action_just_pressed("jump") and not stamina_bar.is_exhausted(stamina_bar.jump_drain):
			stamina_bar.calculate_stamina(stamina_bar.jump_drain)
			velocity.y = JUMP_VELOCITY
		elif(Input.is_action_just_pressed("jump") and stamina_bar.is_exhausted(stamina_bar.jump_drain)):
			stamina_bar.low_stamina_effect()
			low_stamina_player()
	elif not is_on_floor():
		if Input.is_action_just_released("jump") and velocity.y < JUMP_VELOCITY / 2:
			velocity.y = JUMP_VELOCITY / 2


		if Input.is_action_just_pressed("jump") and air_jump and not just_wall_jumped and not stamina_bar.is_exhausted(stamina_bar.jump_drain):
			stamina_bar.calculate_stamina(stamina_bar.jump_drain)
			velocity.y = JUMP_VELOCITY * 0.8
			air_jump = false
		elif(Input.is_action_just_pressed("jump") and stamina_bar.is_exhausted(stamina_bar.jump_drain)):
```

```
                    stamina_bar.low_stamina_effect()
                    low_stamina_player()

func handle_acceleration(input_axis, delta):
    if not is_on_floor(): return
    if input_axis != 0:
        velocity.x = move_toward(velocity.x, SPEED * input_axis, ACCELERATION * delta)

func handle_air_acceleration(input_axis, delta):
    if is_on_floor(): return
    if input_axis != 0:
        velocity.x = move_toward(velocity.x ,SPEED * input_axis, AIR_ACCELERATION * delta)

func handle_wall_hold(delta):
    if is_on_floor(): return
    if is_on_wall_only() and (Input.is_action_pressed("walk_left") or Input.is_action_pressed("walk_right")):
        velocity.y = move_toward(velocity.y, WALL_SLIDE, ACCELERATION * delta)

func apply_friction(input_axis, delta):
    if input_axis == 0 and is_on_floor():
        velocity.x = move_toward(velocity.x, 0, FRICTION * delta)

func apply_air_resistance(input_axis, delta):
    if input_axis == 0 and not is_on_floor():
        velocity.x = move_toward(velocity.x, 0, AIR_RESISTANCE * delta)

func handle_crouch():
    if is_on_floor() and Input.is_action_pressed("crouch") and not is_crouching:
        player_shape.disabled = true
        player_shape_crouched.disabled = false
        is_crouching = true
```

```
	elif is_crouching:
		if Input.is_action_just_released("crouch"):
			attempt_uncrouch()
		elif not Input.is_action_pressed("crouch"):
			attempt_uncrouch()


func attempt_uncrouch():
	if not shape_cast_2d.is_colliding():
		player_shape.disabled  = false
		player_shape_crouched.disabled  = true
		is_crouching = false



func update_animations(input_axis):


	if(dash_timer.is_stopped()):
		if is_on_floor() and input_axis !=0:
			if(not is_crouching):
				animated_sprite_2d.flip_h  = (input_axis < 0)
				animated_sprite_2d.play("walk")
			else:
				animated_sprite_2d.flip_h  = (input_axis < 0)
				animated_sprite_2d.play("crouch")
		elif not is_on_floor():
			#flip direction if action pressed
			if(Input.is_action_pressed("walk_left")  or Input.is_action_pressed("walk_right")):
				animated_sprite_2d.flip_h  = (input_axis < 0)
		#jumping and walking
			if is_on_wall_only() and (Input.is_action_pressed("walk_left")  or
Input.is_action_pressed("walk_right")):
				animated_sprite_2d.play("wall_fall")
```

```
                #jump up
                elif velocity.y < 0.0:
                        #handle  double jump if too soon pressed
                        if animated_sprite_2d.is_playing()  and Input.is_action_just_pressed("jump"):
                                animated_sprite_2d.stop()
                                animated_sprite_2d.play("jump")
                        animated_sprite_2d.play("jump")
                #falling
                elif velocity.y > 0.0:
                        animated_sprite_2d.play("fall")
        else:
                if (not is_crouching):
                        animated_sprite_2d.play("idle")
                else:
                        animated_sprite_2d.play("crouch_idle")
    else:
        if(Input.is_action_just_pressed("dash_left")  or Input.is_action_just_pressed("dash_right")):
                dash_orientation =              (input_axis < 0)
        animated_sprite_2d.flip_h  = dash_orientation
        animated_sprite_2d.play("dash")


func low_stamina_player():
    create_tween().tween_property(self, 'modulate', Color.DIM_GRAY, 0.1)
    create_tween().tween_property(self, "scale", scale * 1.2, 0.3)
    await get_tree().create_timer(0.1).timeout
    create_tween().tween_property(self, 'modulate', color, 0.1)
    create_tween().tween_property(self, "scale", original_scale, 0.3)


func _on_dash_timer_timeout():
    dash_timer.stop()
```

3. Создавање на Стамина систем (скрипта и сцена)

Главната функционалност на стамина системот се процесира преку вградената _process() функција, која чека промени на секоја секунда. Доколку во скирптата на играчот пристигне информација дека тој потрошил стамина, тогаш се појавува визуелниот прика з на стамината во форма на вградена Годот класа наречена ProgressBar.

ProgressBar е Годот класа која се користи за прикажување на линеарна промена на некоја вредност. Во нашиот случај се користи за прикување на моменталната вредност на стамина која ја има играчот, како и моментот кога таа природно се регенерира.

Индикаторот за стамина се прикажува на екранот доколку потрошиме одреден дел од неа и исчезнува при нејзино целосно пополнување. Истото во код е изведено преку два тајмери.

Едниот тајмер се користи за давање на доволно време на функцијата за да ја пополни стамината, а другиот за визуелен приказ на минимално ниво на стамина со цел корисникот да е информиран дека мора да почека за да се регенерира.

Функцијата за намалување на стамината е директно поврзана со скриптата на играчот, при што секое движење на играчот е поврзано со различна вредност од стамината. Количествата на одземена стамина од различните типови на движење се зачувани соодветно во посебни константи.

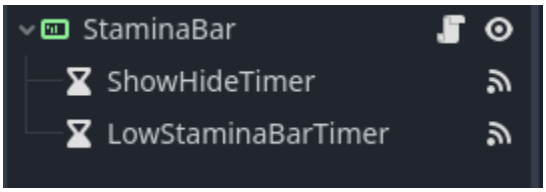
Финалната скрипта за системот на стамина:

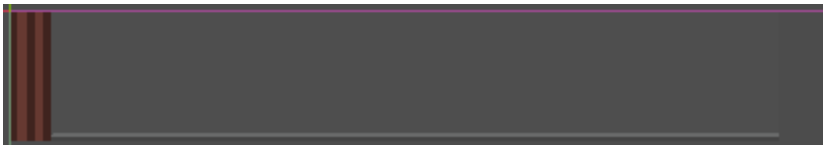
```
extends TextureProgressBar


@onready var show_hide_timer = $"ShowHideTimer"
@onready var low_stamina_timer = $LowStaminaBarTimer
@export var jump_drain = 20.0
@export var dash_drain = 40.0
@export var wall_jump_drain = 15.0
var color = self.modulate


func _ready():
```

```
		modulate.a = 0
		value = 100
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
		if(Input.is_action_just_pressed('jump') or Input.is_action_just_pressed('dash_left') or
Input.is_action_just_pressed('dash_right')):
			turn_visible()
			show_hide_timer.start()
		if(show_hide_timer.is_stopped()):
			value += 1
		if value == 100:
			turn_invisible()

func turn_invisible():
		create_tween().tween_property(self, 'modulate:a', 0, 0.3)

func turn_visible():
		create_tween().tween_property(self, 'modulate:a', 1, 0.3)

func low_stamina_effect():
		position.x += 10
		create_tween().tween_property(self, 'modulate', Color.RED, 0.1)
		await get_tree().create_timer(0.1).timeout
		position.x -= 10
		create_tween().tween_property(self, 'modulate', color, 0.1)

func calculate_stamina(type):
		value -= type

func is_exhausted(type):
		return type > value
```

```
func _on_show_hide_timer_timeout():
	show_hide_timer.stop()


func _on_low_stamina_bar_timer_timeout():
	low_stamina_timer.stop()
```

Конфигурација на сцената и слика од нацртаниот стамина индикатор:

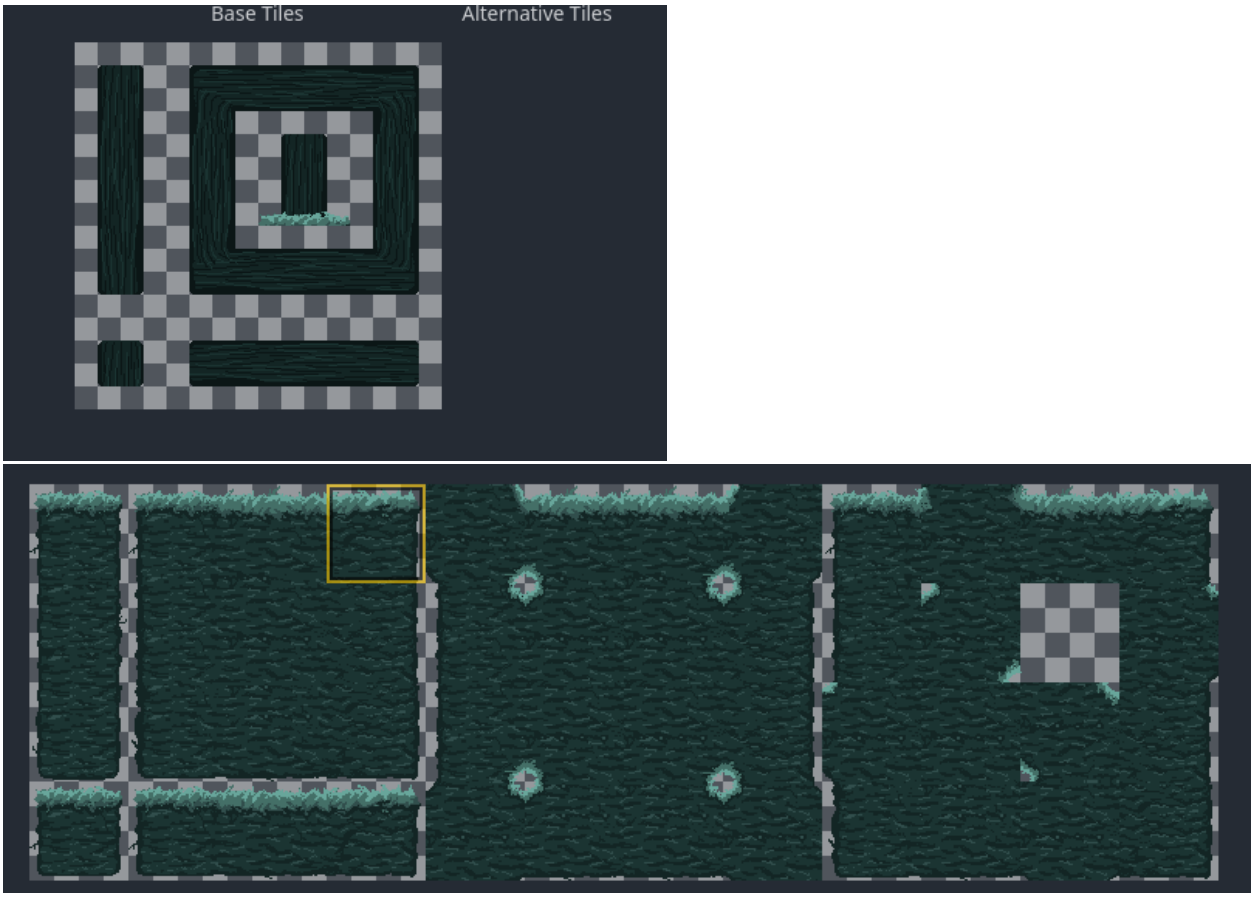




4. Создавање на тела за колизија и приказ на движењата од стамина системот

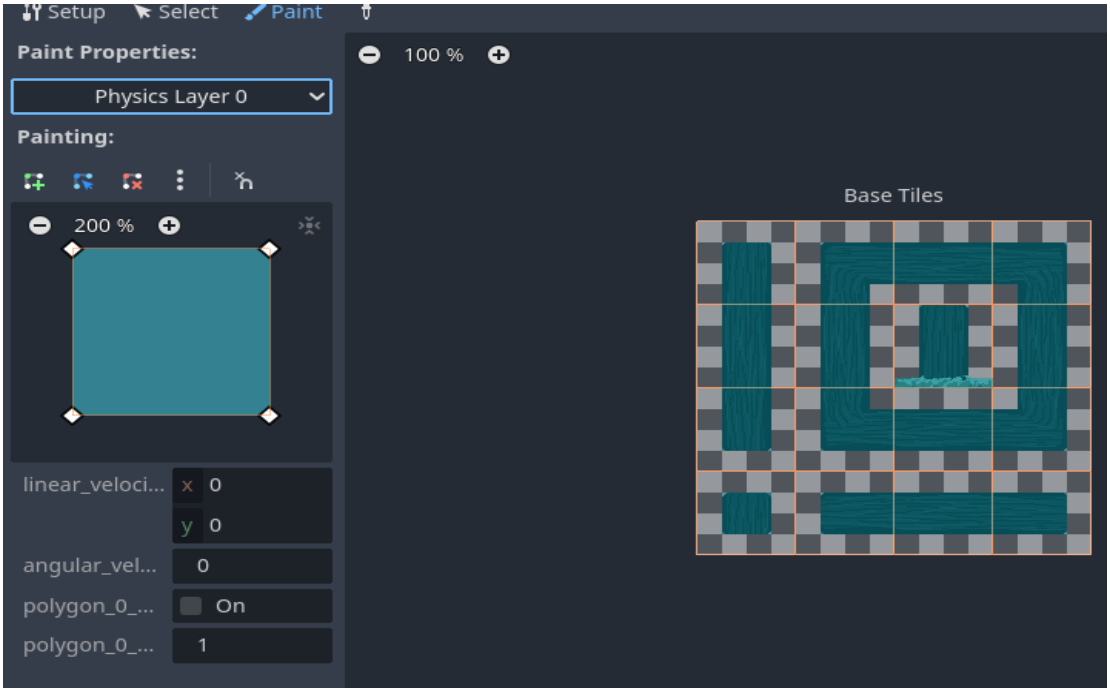
Во Годот постои систем наменет за генерирање на физички тела поделени во коцки (во зависност од потребниот облик) кои се користат за колизија со карактерот и создавање на препреки наречен Tilemap систем.

Во истиот систем постојат структури наречени Tilemaps за кои што се нацртани текстури за трева и дрвја во соодветната шема.

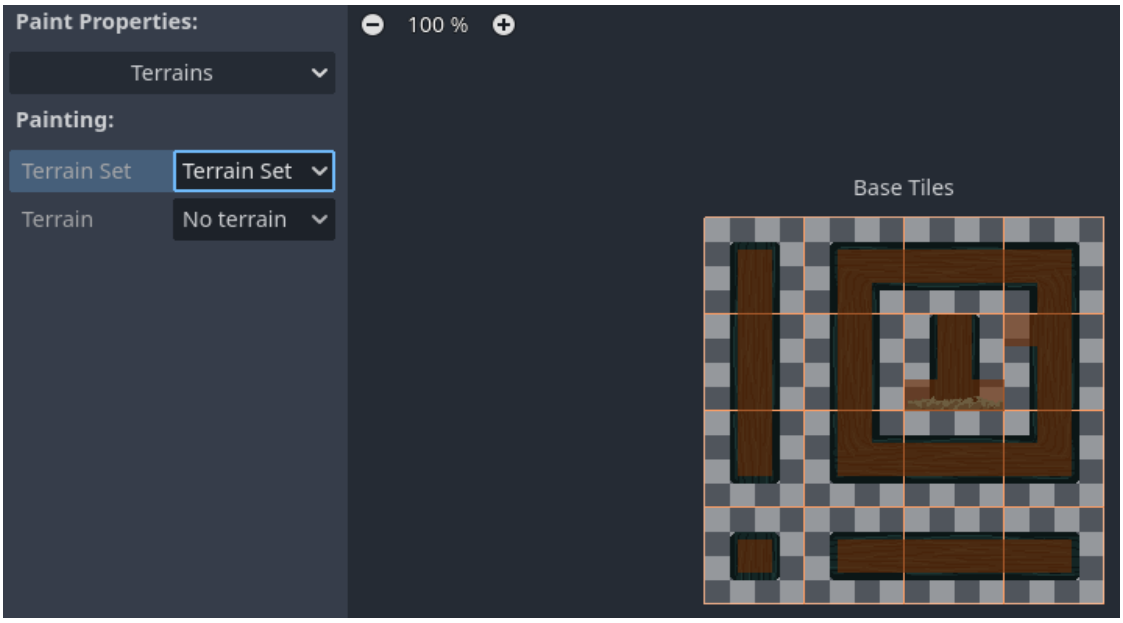


На сликата се прикажани тексурите за дрво и трева

Потоа, на истите се цртаат физички тела за колизија со карактерот и секвенци по кои треба да се генерираат со помош на клик и придвижување на глушецот во било која насока на движење.



Визуелен едитор за мапирање на физичко тело на текстурите



Визуелен едитор за мапирање на секвенци за генерирање на текстурите

По мапирање на секоја од текстурите, истите се поврзуваат во сцената на нивото заедно со играчот, стамина системот и камерата која го следи играчот во самата игра.

5. Поврзување на стамина системот, играчот и текстурите врз кои ќе се движи во главната сцена

Во главната сцена се инстанцира сцената на играчот, сцената на стамина системот и две посебни инстанци за текстурата на тревата и текстурата на дрвото.
Потоа, се создава нов тип на јазол наречен Camera2D кој служи за следење на карактерот при движење во самата игра. Истиот јазол се подесува да е лимитиран до позадината во која се движи и фокусира стрикно на движењата на играчот за промена на неговата позиција.

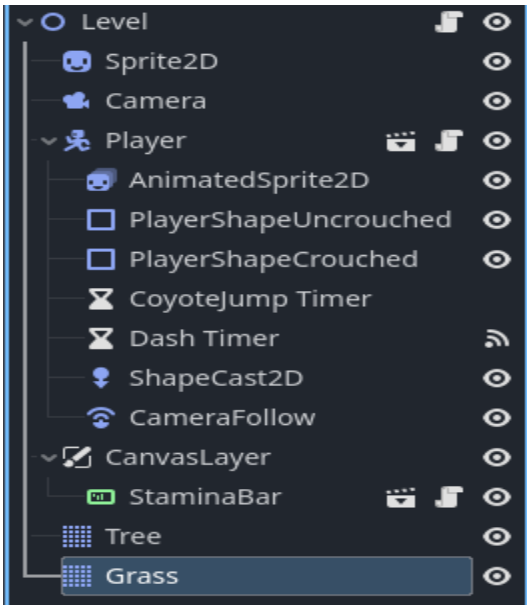
Стамина индикаторот се прицврстува во левиот горен агол на сцената со помош на скрипта создадена во главната сцена со фиксни координати во прозорот на камерата.

Подолу е прикажана скриптата за прицврстување:

```
extends Node2D


func _ready():


	$CanvasLayer/StaminaBar.position = Vector2(20, 20)
```

Со помош на генераторот на текстурите ја цртаме мапата на нивото и ги поставуваме местата каде карактерот треба да скокне, клекне, забрза движење и падне.

По завршување на целиот процес сцената на нивото ни изгледа вака:



Доколку се пушти финалниот проект приказот на целата игра е следен: