Colegiul Național ”Ion Creangă”

**Zwitch**

Proiect pentru examenul de atestare profesională la

INFORMATICĂ

Elev: Savin Ana-Bianca

Clasa: a XII-a A

Profesor coordonator: Danciu Alina

București, 2022

**Cuprins**

Introducere ..................................................................................................................3 pag

Descrierea aplicației ....................................................................................................4 pag

Implementare ..............................................................................................................6 pag

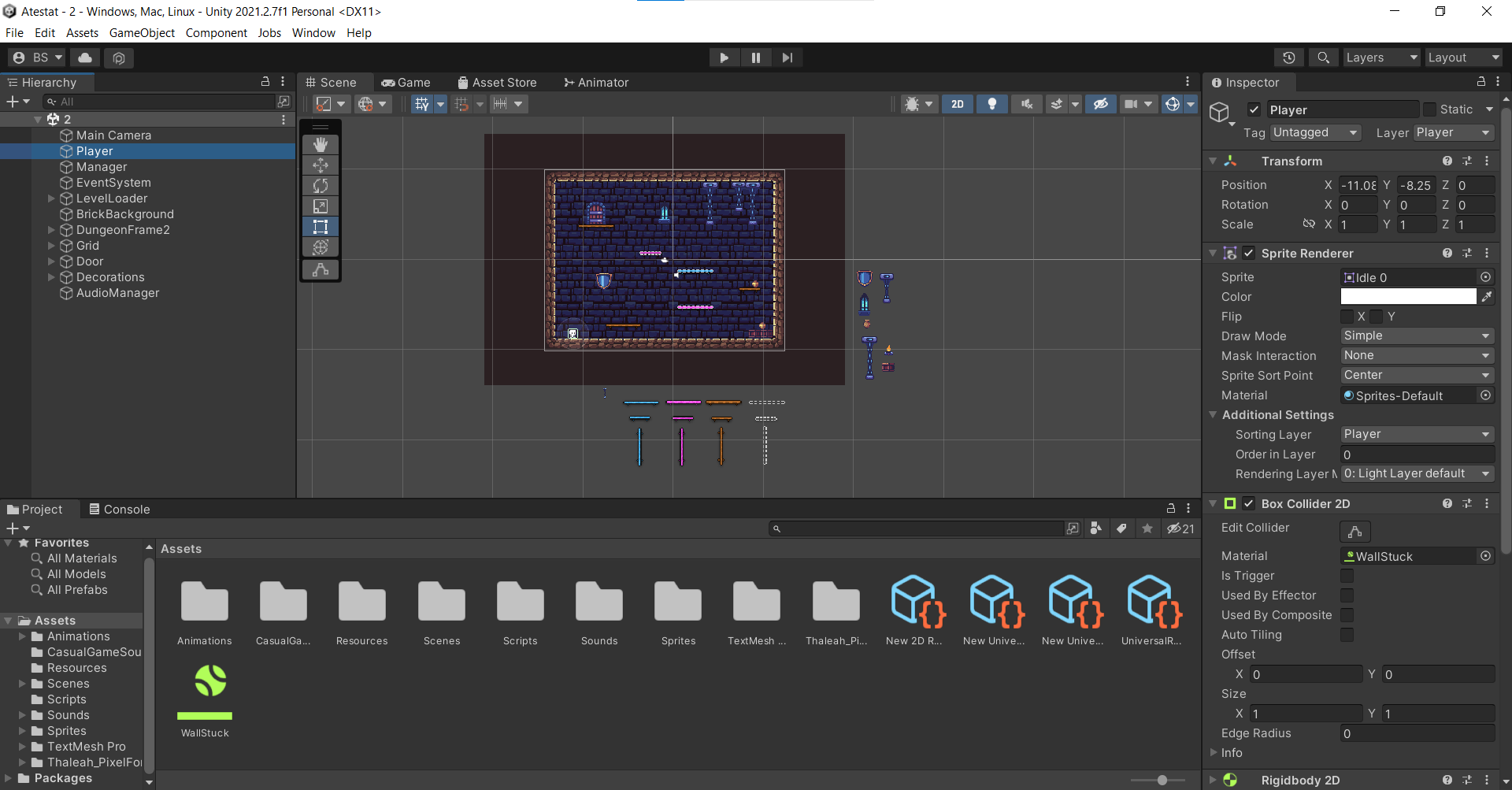
Încheiere ...................................................................................................................22 pag

Webografie ................................................................................................................23 pag

**Introducere**

La fel ca mulţi alţi oameni, pot spune ca îmi plac jocurile video, dar mă şi pasioneză modul in care sunt făcute. De mult timp am vrut sa realizez un joc până la capăt, iar acesta era momentul oportun. Pentru a duce la final proiectul meu, am folosit motorul grafic (engine) Unity, Visual Studio 2019 (codul fiind in C#) şi o aplicaţie Web numită Piskel.

**Unity** este un motor grafic ce permite crearea jocurilor video de orice fel pentru orice platforma, dar mă voi concentra pe dotările pentru cele 2D, din moment ce Zwitch este un astfel de joc.



Aici puteţi vedea interfaţa aplicaţiei. În partea de jos se află toate folderele ce separă şi organizează sprite-urile (denumirea pentru un desen folosit), script-urile (denumirea pentru un cod), muzica şi sunetele folosite. În stânga este o lista a tuturor elementelor ce se află în acest moment în scena, adică, în spaţiul din mijloc, unde se construieşte o anumita parte din joc (meniul şi fiecare nivel au o scena proprie spre exemplu). În dreapta este inspector-ul care arata proprietaţiile obiectului selectat.

De asemenea, Unity are un asset store (magazin de resurse) din care am descărcat mai multe pachete pentru majoritatea sprite-urilor, pentru muzică şi pentru sunete.

**Visual Studio** este aplicaţia folosită pentru editarea script-urilor, care sunt scrise în C#.

**Piskel** este o aplicaţie Web folosită pentru a crea desene pixel art, aşa cum puteţi să vedeţi mai jos.



Am folosit această aplicaţie pentru a desene câteva sprite-uri simple, de care aveam nevoie şi pentru care nu am găsit unele echivalente satisfăcătoare pe asset store.

**Descrierea aplicației**

Zwitch este un joc 2D de tip platformer ce conţine 15 nivele. Jucătorul controlează personajul cu săgeţile şi poate apăsa pe tasta z pentru a schimba între ce platforme sunt vizibile: cele albastre sau cele roz. De asemenea, el poate sa treacă printr-o platformă sărind din jos în sus pe ea şi poate să cadă prin aceasta, dacă stă pe platformă şi apasă săgeata de jos. Scopul fiecărui nivel este sa ajungi la uşa. În primul si al cincilea nivel se află tutorialul (fiinţa din poza de mai sus), care îi explică jucătorului ce trebuie sa facă.

Aici este o imagine cu primul nivel:



Dacă apăsăm pe z, va apărea platforma roz, unde se vede conturul ei, iar cea albastră dispare lăsând un contur in locul unde era:



Jocul are de asemenea un meniu, de unde utilizatorul poate selecta PLAY pentru a începe de la primul nivel, LEVELS pentru a selecta manual nivelul dorit, OPTIONS pentru a deschide un alt meniu ce permite ajustarea volumului si vizualizarea instrucţiunilor, QUIT pentru a închide aplicaţia.

Meniul principal arată astfel:



Meniul pentru selectarea nivelelor:



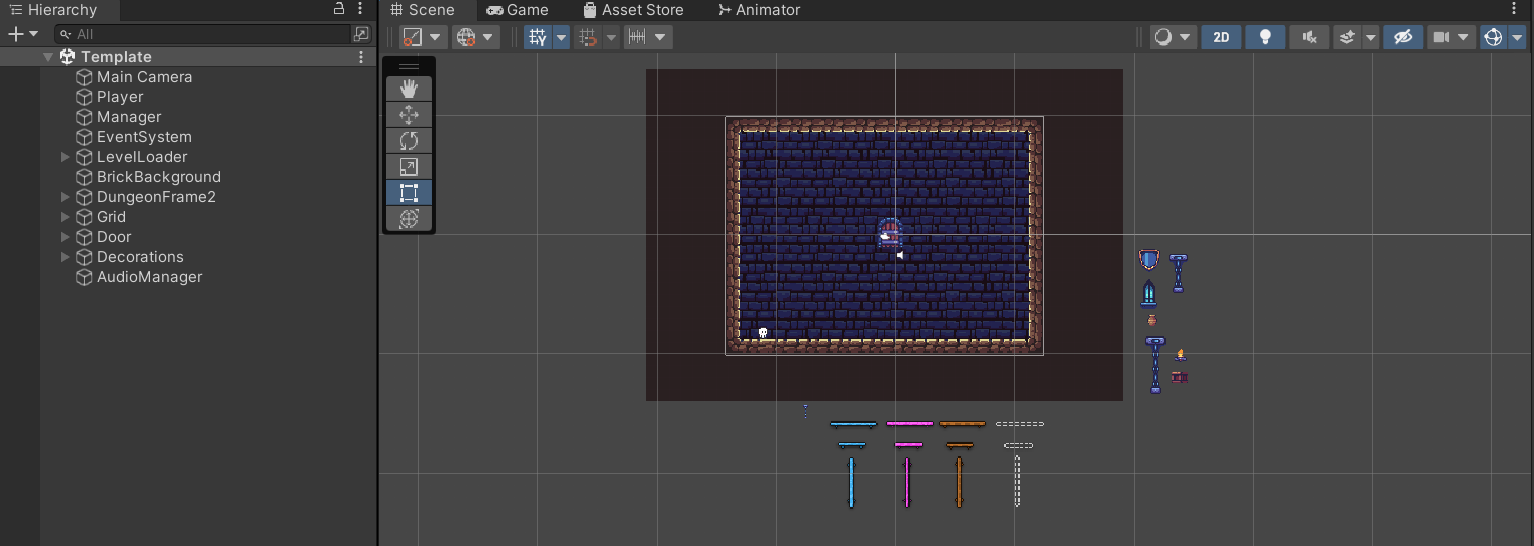
Exista si un meniu de pauză care poate fi accesat din orice nivel, prin apăsarea butonului din stânga sus. Acesta are opţiunea de a te întoarce la meniul principal sau de a relua acţiunea de dinaintea deschiderii lui.



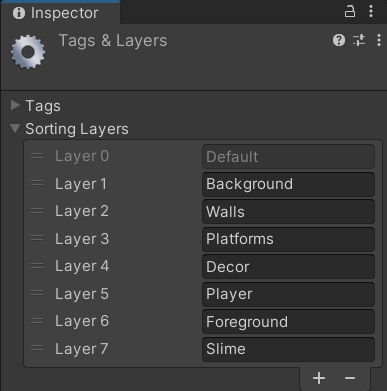
**Implementare**

***Scena***

După cum am menţionat şi mai înainte, fiecare nivel are propria scenă. Deoarece ele au nevoie de acelaşi setup am făcut o scenă template după care am creat nivelele.

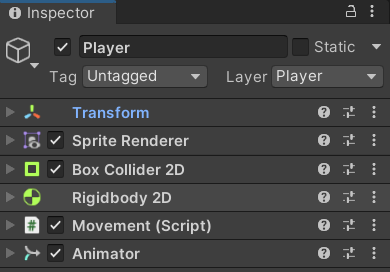


Câteva din elementele fundamentale ale fiecărei scene sunt: camera ce este fixată mereu la poziţia (0, 0), iar tot ce cuprinde ea este ceea ce va fi arătat pe ecran jucătorului; cadranul maro care are 4 collidere, pentru ca personajul sa nu poată ieşi din zona nivelului; Decorations, un element ce are rolul de a fi părintele tuturor sprite-urilor care au un rol decorativ. Toate elementele vizuale sunt ordonate în layere pentru a le stabili ordinea vizibilităţii lor (spre exemplu layer-ul player-ului are o prioritate mai mare decât cel al platformelor, pentru că, atunci când sare prin ele, el este vizibil).



***Player***

Personajul principal este alcătuit din o componenta Transform (ce memorează poziţia si dimensiunea player-ului), Sprite, Collider (ce îl face sa nu treacă prin alte obiecte), Rigidbody (ce face caracterul sa fie afectat de legile clasice ale fizicii, spre exemplu gravitaţia), script-ul ***Movement*** şi Animator (ce se ocupă de animaţiile player-ului).



Script-ul ***Movement*** este cel care se ocupă de mişcarea personajului, interpretând input-ul primit de la tastatură. Pentru realizarea lui am folosit clase şi funcţii predefinite din bibliotecile Unity.

Mai întâi am făcut declarările de variabile (variabilele publice pot fi editate din inspector):

[Header("Move")]

public Rigidbody2D rb;

public Animator anim;

public GameObject currentPlatform;

public bool GameIsPaused;

public float horizontal;

public float vertical;

public float speed;

public bool facingRight;

public bool leftArr;

public bool rightArr;

[Header("Jump")]

public LayerMask ground;

public bool upArrJump;

public bool grounded = false;

public float groundedSkin = 0.05f;

public float jumpForce;

public bool holdUpArr;

public bool jumpNow;

public Vector2 playerSize;

public Vector2 boxSize;

public Vector2 boxOffset;

public float colliderOffset;

[Header("Better Jump")]

public float smallJump;

public float fallJump;

public bool slowJump;

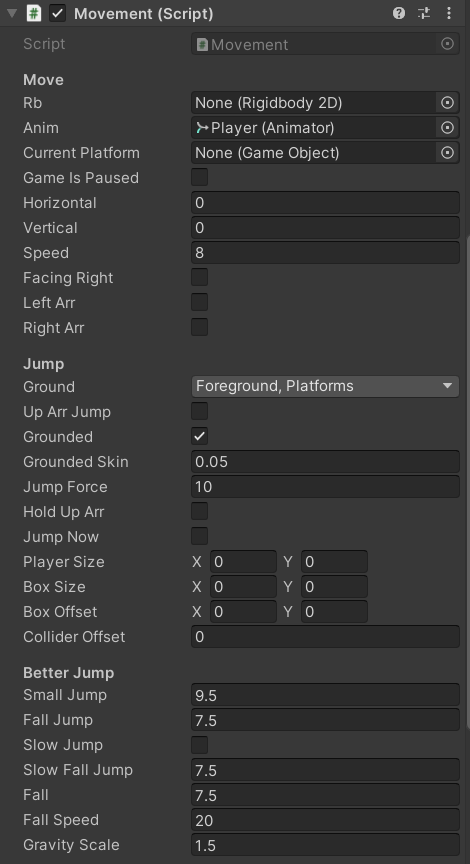
public float slowFallJump;

public float fall;

public float fallSpeed;

public float gravityScale = 1f;

Header-ul ajută la organizarea variabilelor, când sunt văzute în scenă, din inspector.



În continuare, am folosit funcţia **Awake**, predefinită de Unity, care este apelata o singură dată, când o scena noua este încărcată. Am utilizat-o pentru a memora informaţii despre player, precum dimensiunea lui şi pentru a seta o valoare de început pentru anumite variabile.

void Awake()

{

rb = GetComponent<Rigidbody2D>();

colliderOffset = 0.02f;

playerSize = GetComponent<BoxCollider2D>().size - Vector2.one \* colliderOffset;

boxOffset = GetComponent<BoxCollider2D>().offset;

boxSize = new Vector2(playerSize.x, groundedSkin);

facingRight = true;

rb.constraints = RigidbodyConstraints2D.None | RigidbodyConstraints2D.FreezeRotation;

}

Funcţia **Update** este cea în care citesc input-ul, memorându-l în variabile. Ea este apelată în continuu, o dată pe cadru.

void Update()

{

horizontal = Input.GetAxis("Horizontal");

vertical = Input.GetAxis("Vertical");

SmootherMovement();

if (Input.GetKeyDown(KeyCode.UpArrow) && grounded && !GameIsPaused)

{

upArrJump = true;

AudioManager.PlaySound("jump");

}

if (Input.GetKeyDown(KeyCode.DownArrow))

if (currentPlatform != null && !GameIsPaused)

{

StartCoroutine(DisableCollision());

AudioManager.PlaySound("FallThrough");

}

holdUpArr = Input.GetKey(KeyCode.UpArrow);

leftArr = Input.GetKey(KeyCode.LeftArrow);

rightArr = Input.GetKey(KeyCode.RightArrow);

}

Funcţia **FixedUpdate** cea care se ocupă de modificările explicite asupra personajului. Ea este apelată in continuu, de mai multe ori pe cadru (astfel dacă am citi input-ul aici, o parte din el s-ar pierde). De asemenea, aici sunt apelate funcţiile scrise de mine.

void FixedUpdate()

{

if(!GameIsPaused)

{

Move();

Jump();

Vector2 boxCenter = (Vector2)transform.position + GetComponent<BoxCollider2D>().offset + Vector2.down \* (playerSize.y + boxSize.y) \* 0.5f;

grounded = Physics2D.OverlapBox(boxCenter, boxSize, 0f, ground) && Mathf.Abs(rb.velocity.y) <= 0.2f;

if (facingRight && horizontal < 0)

Flip();

else if (!facingRight && horizontal > 0)

Flip();

}

Animation();

}

Funcţia **Move** mişcă player-ul prin schimbarea velocităţii rigidbody-ului, iar funcţia **SmootherMovement** manipulează input-ul din Update, pentru o mişcare mai potrivită.

void Move()

{

if (rightArr && leftArr)

rb.velocity = new Vector2(0f, rb.velocity.y);

else if (!grounded)

rb.velocity = new Vector2(horizontal \* speed \* 0.95f, rb.velocity.y);

else

rb.velocity = new Vector2(horizontal \* speed, rb.velocity.y);

}

void SmootherMovement()

{

if (leftArr && horizontal > 0f)

horizontal = 0;

else if (rightArr && horizontal < 0f)

horizontal = 0f;

}

Funcţia **Flip** roteşte sprite-ul, astfel încât player-ul sa fie orientat spre direcţia în care merge jucătorul.

void Flip()

{

facingRight = !facingRight;

if (!GetComponent<SpriteRenderer>().flipX)

GetComponent<SpriteRenderer>().flipX = true;

else if (GetComponent<SpriteRenderer>().flipX)

GetComponent<SpriteRenderer>().flipX = false;

}

Funcţia **Jump** face personajul sa sară, schimbând velocitatea Rigidbody-ului când este necesar. Pentru a crea un salt mai bun, gravitaţia scenei se schimbă în funcţie de cât de mult ţine jucătorul săgeata de sus apăsată. Se creează mai multe cazuri, astfel încât personajul să cadă cu o viteză adecvată oricând el ia mâna de pe buton.

void Jump()

{

if (grounded && !upArrJump)

jumpNow = false;

if (upArrJump)

{

rb.velocity = Vector2.up \* jumpForce;

jumpNow = true;

upArrJump = false;

}

if (rb.velocity.y < 0f && !jumpNow)

rb.gravityScale = fall;

else if (!slowJump)

{

if (!holdUpArr && jumpNow)

{

rb.gravityScale = smallJump;

}

}

else if (holdUpArr && rb.velocity.y > 0f && jumpNow)

{

rb.gravityScale = gravityScale;

}

else if (rb.velocity.y < 0f && jumpNow && holdUpArr)

{

rb.gravityScale = slowFallJump;

}

else if (!holdUpArr && jumpNow)

{

rb.gravityScale = fallJump;

}

else if (slowJump)

{

if (!holdUpArr && jumpNow)

{

rb.gravityScale = fallJump;

}

}

if (vertical > 0.45f)

vertical = 1;

if (vertical == 1f)

slowJump = true;

if (grounded)

{

rb.gravityScale = gravityScale;

slowJump = false;

}

if (rb.velocity.y < -fallSpeed)

rb.velocity = new Vector2(rb.velocity.x, Mathf.Clamp(rb.velocity.y, -fallSpeed, Mathf.Infinity));

}

Pentru a crea abilitatea player-ului de a cădea printr-o platformă, dacă acesta stă pe ea şi apasă săgeata de jos am folosit mai multe funcţii. Variabila currentPlatform memorează platforma pe care se afla player-ul, când intră în contact cu ea şi ia valoare null când collider-ele ies din contact. Funcţiile **OnCollisionEnter2D** şi **OnCollisionExit2D** se ocupă de acest lucru.

private void OnCollisionExit2D(Collision2D collision)

{

if (collision.gameObject.layer == 9)

currentPlatform = null;

}

private IEnumerator DisableCollision()

{

BoxCollider2D currentPlat = currentPlatform.GetComponent<BoxCollider2D>();

Physics2D.IgnoreCollision(currentPlat, GetComponent<BoxCollider2D>());

yield return new WaitForSeconds(0.5f);

Physics2D.IgnoreCollision(currentPlat, GetComponent<BoxCollider2D>(), false);

}

Folosim **IENUMERATOR**-ul pentru a dezactiva pentru 0.5 secunde a platformei prin care vrea jucătorul sa cadă. Funcţia StartCoroutine(DisableCollision()) este apelată în Update pentru a începe execuţia lui.

private IEnumerator DisableCollision()

{

BoxCollider2D currentPlat = currentPlatform.GetComponent<BoxCollider2D>();

Physics2D.IgnoreCollision(currentPlat, GetComponent<BoxCollider2D>());

yield return new WaitForSeconds(0.5f);

Physics2D.IgnoreCollision(currentPlat, GetComponent<BoxCollider2D>(), false);

}

Funcţia **Animation** are scopul de a modifica parametrii necesari schimbarii dintre animaţiile player-ului.

void Animation()

{

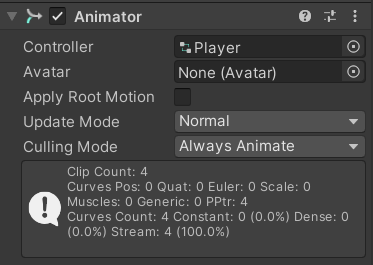
anim.SetFloat("Speed", Mathf.Abs(rb.velocity.x));

anim.SetFloat("Vertical", rb.velocity.y);

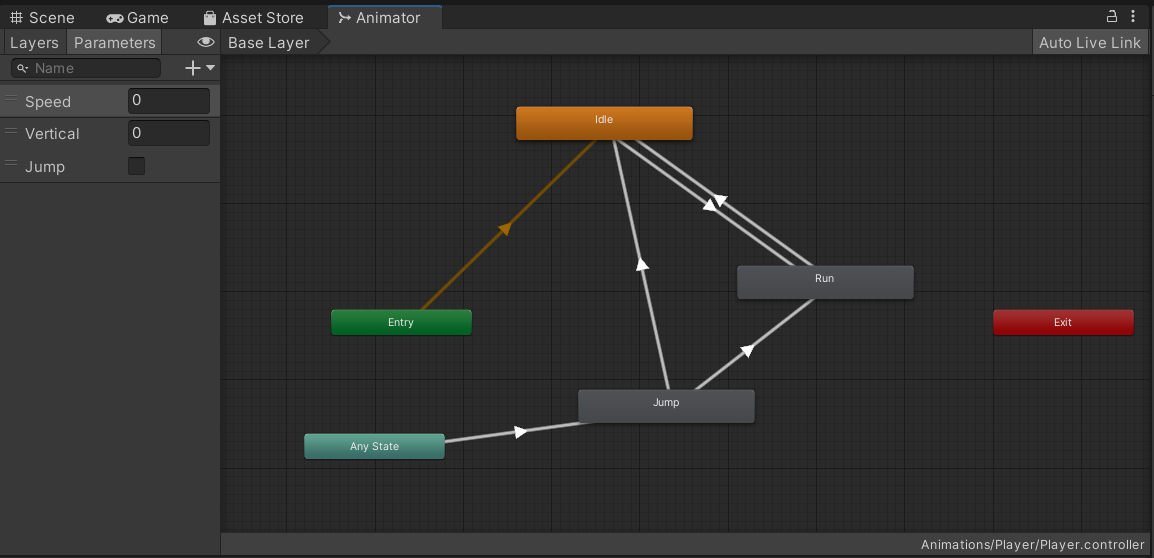
anim.SetBool("Jump", !grounded);

}

***Animator*** este componenta player-ului care se ocupa de animaţia ce trebuie redată.



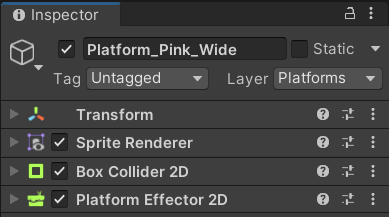
Aceasta primeşte un **Animator Controller** care în cazul de faţă se numeşte Player. El conţine toate animaţiile player-ului şi le organizează într-un Animation Tree care arată astfel:



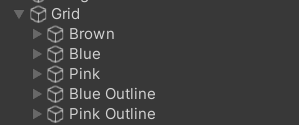
Săgeţile conţin condiţiile de schimbare a animaţiei, în funcţie de valoarea parametrilor aflaţi în lista din stânga şi a căror valoare este modificată prin cod, în funcţia Animation.

***Platforme***

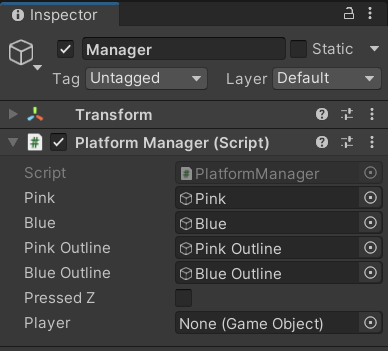
Fiecare platformă este alcătuită din: Transform, Sprite, Collider şi Platform Effector, care permite obiectelor ce conţin un Rigitbody (player-ul în cazul de faţă) să treacă prin ea prin partea de jos.



Pentru fiecare nivel, am creat un grid ce conţine 5 Tilemaps. Ele permit punerea platformelor la o distanţă măsurată, predefinită pe grid. Acestea separă platformele maro, platformele albastre, platformele roz, conturul platformelor albastre şi conturul platformelor roz.



Pentru schimbarea dintre platformele vizibile, am creat script-ul **PlatformManager**, pe care l-am ataşat unui element gol din scenă numit Manager. La apăsarea tastei z, dacă spre exemplu platformele albastre şi, implicit, conturul platformelor roz sunt vizibile, atunci ele sunt dezactivate, iar platformele roz şi conturul platformelor albastre sunt activate, şi vice-versa.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlatformManager : MonoBehaviour

{

public GameObject pink;

public GameObject blue;

public GameObject pinkOutline;

public GameObject blueOutline;

public bool pressedZ;

public GameObject player;

private void Awake()

{

pink.SetActive(false);

blueOutline.SetActive(false);

player = GameObject.Find("Player");

}

void Update()

{

if (Input.GetKeyDown(KeyCode.Z) && !player.GetComponent<Movement>().GameIsPaused)

pressedZ = true;

}

void FixedUpdate()

{

if (pressedZ)

{

AudioManager.PlaySound("ZPressed");

pink.SetActive(!pink.activeSelf);

blue.SetActive(!blue.activeSelf);

pinkOutline.SetActive(!pink.activeSelf);

blueOutline.SetActive(!blue.activeSelf);

pressedZ = false;

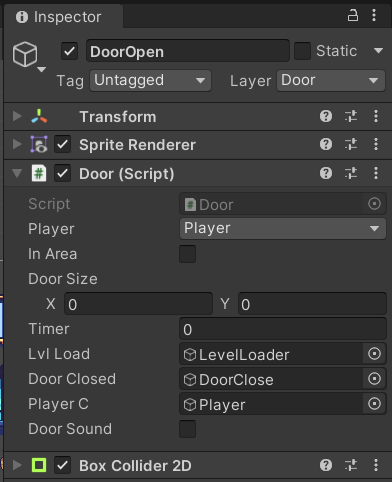
}

}

}

***Uşa***

Când jucătorul ajunge la uşă, începe un timer de 2 secunde. El trebuie sa stea 1 secunda în faţa uşii pentru ca aceasta să se deschidă. Dacă player-ul iese din aria de detectare din faţa uşii, timer-ul se restartează. Pentru a face uşa să se deschidă nu am folosit o animaţie propriu-zisă, deoarece ea ar avea doar 2 cadre statice. Astfel, sprite-ul cu uşa deschisă este părintele sprite-ului cu uşa închisă, el având ataşat script-ul **Door**, care mută uşa închisă în faţa celei deschise când trece 1 secundă.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Door : MonoBehaviour

{

public LayerMask player;

public bool inArea;

public Vector2 doorSize;

public float timer;

public GameObject lvlLoad;

public GameObject doorClosed;

public GameObject playerC;

public bool doorSound;

private void Awake()

{

doorSound = true;

doorSize = GetComponent<BoxCollider2D>().size;

}

void FixedUpdate()

{

Vector2 pointA = (Vector2)transform.position + GetComponent<BoxCollider2D>().offset + Vector2.right \* doorSize.x + Vector2.up \* doorSize.y;

Vector2 pointB = (Vector2)transform.position + GetComponent<BoxCollider2D>().offset + Vector2.left \* doorSize.x + Vector2.down \* doorSize.y;

inArea = Physics2D.OverlapArea(pointA, pointB, player);

if(!playerC.GetComponent<Movement>().GameIsPaused)

{

if (inArea)

timer -= Time.fixedDeltaTime;

else

timer = 2f;

if (timer <= 1f)

{

if(doorSound)

{

AudioManager.PlaySound("Door");

doorSound = false;

}

doorClosed.transform.position = new Vector3(doorClosed.transform.position.x, doorClosed.transform.position.y, 0f);

transform.position = new Vector3(transform.position.x, transform.position.y, 4f);

playerC.GetComponent<Movement>().rb.constraints = RigidbodyConstraints2D.FreezePositionY | RigidbodyConstraints2D.FreezeRotation | RigidbodyConstraints2D.FreezePositionX;

}

if (timer <= 0f)

lvlLoad.GetComponent<LevelLoader>().LoadNextLevel();

}

}

}

Când timer-ul ajunge la 0, apare o animaţie simplă, în care ecranul se face negru şi, apoi, se încarcă următoarea scenă. Pentru a realiza aceste lucru, am scris script-ul **LevelLoader**, pe care l-am ataşat unui component gol din scenă.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class LevelLoader : MonoBehaviour

{

public Animator transition;

public float transitionTime = 1f;

public void LoadNextLevel()

{

StartCoroutine(LoadLevel(SceneManager.GetActiveScene().buildIndex + 1));

}

IEnumerator LoadLevel(int levelIndex)

{

transition.SetTrigger("Start");

yield return new WaitForSeconds(transitionTime);

SceneManager.LoadScene(levelIndex);

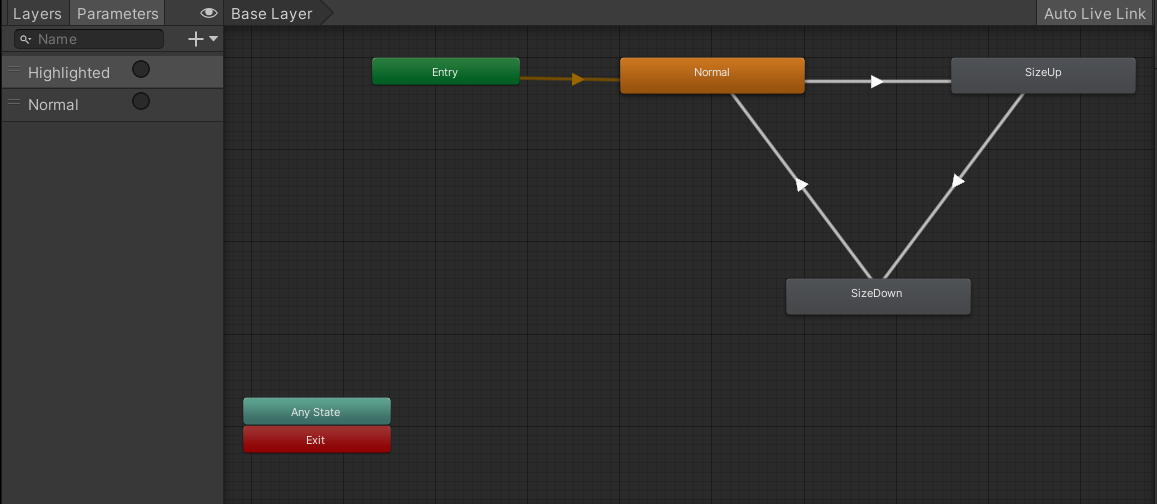
}

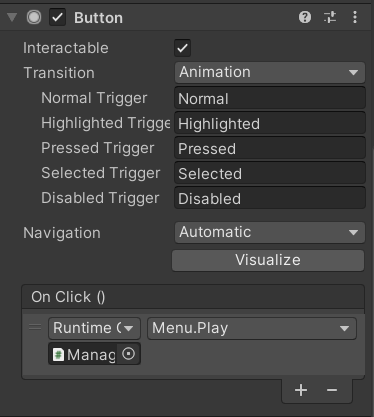
}

***Meniuri si butoane***

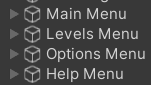
Fiecare meniu, buton şi text este situate pe un Canvas. Acesta este un element suprapus peste tot ceea ce se află în scenă. Spre exemplu, butonul de pauză din fiecare nivel se află pe Canvas, iar animaţia de trecere de la un nivel la altul face culoarea de fundal a Canvasului sa fie negru, iar, apoi, transparent.

Fiecare buton este părintele unei componente text, care reprezintă mesajul afişat pe el. Acesta are o animaţie care măreşte fontul textului, in cazul butoanelor reprezentate prin text alb sau care schimbă imaginea folosită şi mută textul butonului mai jos în cazul butoanelor de selectare a nivelului. De asemenea, butoanele au , implicit, un eveniment OnClick care reprezintă acţiunea efectuată dacă jucătorul apasă pe ele şi căruia îi poate fii atribuita o funcţie printr-un script. Pentru asta, am creat script-ul **Menu** ce conţine o multitudine de funcţii ataşate butoanelor corespunzătoare. Script-ul este ataşat unui element gol din scena numit Manager.





**Meniul principal** are propria scenă şi se desparte în 3 alte meniuri. Fiecare din ele este, de fapt, o componenta ce conţine mai multe butoane-copii, care fac legătura intre acestea.



Meniul principal are 4 butoane.



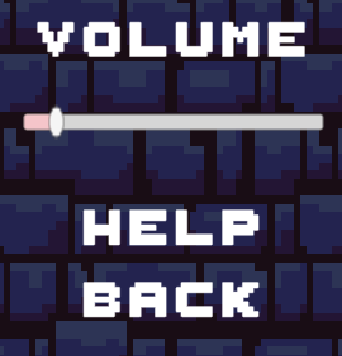


Levels şi Options fac legătura cu meniurile respective. Play începe jocul de la primul nivel, iar Quit închide aplicaţia.

**Meniul Levels** conţine toate butoanele pentru selectarea nivelului dorit şi un buton Back care duce înapoi la meniul principal.



**Meniul Options** conţine bara de ajustare a volumului, un buton spre meniul Help şi un buton Back care duce înapoi la meniul principal.

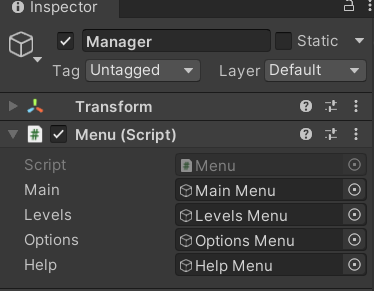
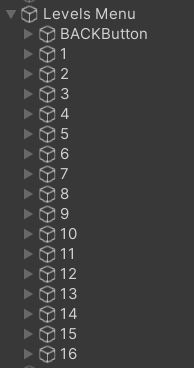


**Meniul Help** conţine un text cu instrucţiuniile jocului si un buton Back care duce înapoi la meniul Options.



După cum am spus şi mai sus, script-ul **Menu** este cel care ce conţine funcţiile pentru butoane. Meniurile care nu sunt afişate sunt la poziţia (1000, 1000) pe Canvas, adică înafara camerei. Dacă le-am dezactiva în loc să le mutăm, ar apărea o problema în care butoanele care fac legătura între meniuri rămân blocate în al doilea cadru al animaţiei lor. Aşa că, la început ne asigurăm că toate meniurile sunt active şi ca meniul principal se află la poziţia (0, 0), iar restul la (1000, 1000). Un buton care face trecerea de la un meniu la altul, le interschimbă pozţiile celor două.

Butoanele din meniul Levels au numele indexului scenei care reprezintă nivelul cu care fac legătura şi folosesc o funcţie care ia numele lor, îl transformă din string în int şi încarcă scena cu acel index.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.EventSystems;

public class Menu : MonoBehaviour

{

public GameObject main;

public GameObject levels;

public GameObject options;

public GameObject help;

public void Start()

{

main.SetActive(true);

levels.SetActive(true);

options.SetActive(true);

help.SetActive(true);

main.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

levels.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

options.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

help.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

}

public void Play()

{

AudioManager.PlaySound("ButtonPressed");

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

public void Quit()

{

AudioManager.PlaySound("ButtonPressed");

Application.Quit();

}

public void LoadLevel()

{

AudioManager.PlaySound("ButtonPressed");

SceneManager.LoadScene(int.Parse(EventSystem.current.currentSelectedGameObject.name));

}

public void MainToLevel()

{

AudioManager.PlaySound("ButtonPressed");

main.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

levels.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

}

public void LevelToMain()

{

AudioManager.PlaySound("ButtonPressed");

main.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

levels.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

}

public void MainToOptions()

{

AudioManager.PlaySound("ButtonPressed");

main.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

options.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

}

public void OptionsToMain()

{

AudioManager.PlaySound("ButtonPressed");

main.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

options.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

}

public void OptionsToHelp()

{

AudioManager.PlaySound("ButtonPressed");

help.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

options.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

}

public void HelpToOptions()

{

AudioManager.PlaySound("ButtonPressed");

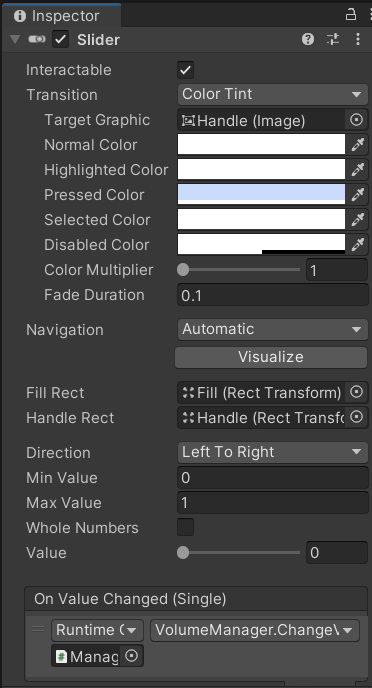
help.GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

options.GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

}

}

Pentru a schimba volumul şi a salva valoarea lui când aplicaţia este închisă, am scris script-ul **VolumeManager**, care este ataşat tot elementului Manager. De data aceasta Slider-ul (glisorul) are un eveniment OnValueChange care este legat de cod.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class VolumeManager : MonoBehaviour

{

public Slider slider;

void Start()

{

if (!PlayerPrefs.HasKey("musicVolume"))

{

PlayerPrefs.SetFloat("musicVolume", 0.5f);

Load();

}

else Load();

}

public void ChangeVolume()

{

AudioListener.volume = slider.value;

Save();

}

private void Load()

{

slider.value = PlayerPrefs.GetFloat("musicVolume");

}

private void Save()

{

PlayerPrefs.SetFloat("musicVolume", slider.value);

}

}

**Meniul de pauza** poate fi accesat din toate nivelele prin apăsarea butonului din stânga sus. El este, similar, aflat la poziţia (1000, 1000), până când jucătorul apasă butonul de pauză. Fundalul Canvas-ului se face mai închis la culoare, când meniul este deschis. Acesta conţine 2 butoane: Main Menu care duce jucătorul la meniul principal si Resume ce reia acţiunea jocului. Script-ul care se ocupă de aceste butoane este **PauseMenu**, ataşat componentei Manager.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour

{

public GameObject player;

void Start()

{

GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

GetComponent<Image>().color = new Color(0f, 0f, 0f, 0f);

}

public void OpenPauseMenu()

{

AudioManager.PlaySound("ButtonPressed");

GetComponent<RectTransform>().transform.localPosition = new Vector3(0, 0, 0);

GetComponent<Image>().color = new Color(0f, 0f, 0f, 0.5f);

player.GetComponent<Movement>().GameIsPaused = true;

}

public void ClosePauseMenu()

{

AudioManager.PlaySound("ButtonPressed");

GetComponent<RectTransform>().transform.localPosition = new Vector3(1000, 1000, 0);

GetComponent<Image>().color = new Color(0f, 0f, 0f, 0f);

player.GetComponent<Movement>().GameIsPaused = false;

}

public void GoToMainMenu()

{

AudioManager.PlaySound("ButtonPressed");

SceneManager.LoadScene(0);

}

}

***Sunetele şi muzica***

Camera are, implicit, o componenta Audio Listener care redă sunetele recepţionate de la elemente ce au o componenta Audio Source. Pentru ca jocul este 2D, si nu avem nevoie de un sistem audio prea complicat, Audio Source este plasat pe o componenta AudioManager din scenă. Muzica este selectată cu ajutorul ei şi pusa în continuu. Scriptul **AudioManager** este ataşat aceluiaşi element şi el încarcă sunetele din folderul Resources, conţinând şi o funcţie statică PlaySound, (ca să poată fi apelată din orice script) care pune un anumit sunet în funcţie de care este argumentul ei.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

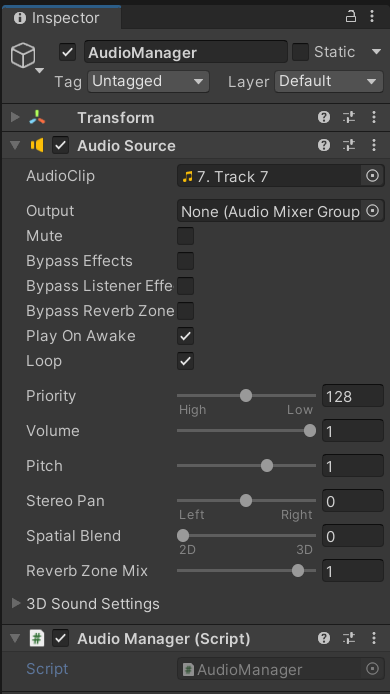
public class AudioManager : MonoBehaviour

{

public static AudioClip ZPressed, jump, buttonPressed, door, fallThrough;

static AudioSource audioS;

void Start()

 {

ZPressed = Resources.Load<AudioClip>("ZPressed");

jump = Resources.Load<AudioClip>("jump");

buttonPressed = Resources.Load<AudioClip>("ButtonPressed");

door = Resources.Load<AudioClip>("Door");

fallThrough = Resources.Load<AudioClip>("FallThrough");

audioS = GetComponent<AudioSource>();

}

public static void PlaySound(string clip)

{

switch (clip)

{

case "ZPressed":

audioS.PlayOneShot(ZPressed);

break;

case "jump":

audioS.PlayOneShot(jump);

break;

case "ButtonPressed":

audioS.PlayOneShot(buttonPressed);

break;

case "Door":

audioS.PlayOneShot(door);

break;

case "FallThrough":

audioS.PlayOneShot(fallThrough);

break;

}

}

}

**Încheiere**

***Ce am învăţat lucrând la Zwitch?***

Deşi ştiam sa lucrez în Unity de ceva timp, realizarea acestui joc m-a învăţat câteva lucruri noi. Ceva ce nu făcusem până acum este să lucrez cu sunete şi să pun muzică de fundal. Modul de animare al butoanelor a fost diferit faţă de ceea ce făceam până acum. De asemenea, a fost prima dată când am programat personajul astfel încât să poată cădea de pe platforme când vrea şi să sară prin ele când vine de jos.

Cu toate acestea, cel mai util lucru pe care l-am învăţat făcând Zwitch este legat de modul diferit în care am folosit Tilemaps. De obicei, este creată o paletă cu sprite-urile care vor fi folosite, dar asta nu merge în cazul meu, pentru că, nu le pot adăuga componente individual (cum ar fii collider sau platform effector). Aşa că am invăţat sa folosesc ceva numit GameobjectBrush, care permite folosirea unui element cu mai multe componente.

***Cum poate fii dezvoltat Zwitch?***

Fiind un joc, cred ca există nenumărate aspecte pe care le-aş putea adăuga sau îmbunătăţi.

Aş putea să adaug nivele şi să introduc mecanici noi (o mecanica este un element central al unui joc, spre exemplu aici există abilitatea de a schimba între platforme). Un exemplu de astfel de element nou pe care mă gândeam să-l adaug este un nou tip de platformă ce poate să arunce player-ul mai sus decât înălţimea la care poate să sară.

Există câteva aspecte pe care vreau să la îmbunătăţesc, principalul fiind uşa. Deoarece player-ul şi uşa au collidere, collision-ul între ele este dezactivat (ca player-ul să poată merge prin faţa uşii), şi, astfel, modul în care am scris codul pentru detectarea player-ului în faţa uşii şi pentru timer nu este atât de eficient, dar funcţionează. În plus, există câteva bug-uri minore despre care ştiu, ce pot fi rezolvate.

**Webografie**

* Documentaţia oficială Unity:
* <https://docs.unity3d.com/Manual/index.html>
* Câteva tutoriale care m-au ajutat:
* <https://www.youtube.com/watch?v=Lyeb7c0-R8c&ab_channel=PitiIT>
* <https://www.youtube.com/watch?v=jh3zD-wGBnw&ab_channel=MousawiDev>
* <https://www.youtube.com/watch?v=8pFlnyfRfRc&ab_channel=AlexanderZotov>
* Asset-urile pe care le-am folosit:
* <https://assetstore.unity.com/packages/2d/free-2d-mega-pack-177430>
* <https://assetstore.unity.com/packages/2d/characters/simple-2d-platformer-assets-pack-188518>
* <https://assetstore.unity.com/packages/2d/environments/medieval-pixel-art-asset-free-130131>
* <https://assetstore.unity.com/packages/2d/fonts/free-pixel-font-thaleah-140059>
* <https://assetstore.unity.com/packages/audio/music/8bit-music-album-051321-196147>
* <https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116>