Giriş

Bu rapor “Çöldeki Şövalye” isimli 2D olarak Unity üzerinde C# script dili kullanılarak geliştirilen oyun projesini açıklamaktadır. Rapor oyunun geliştirme aşamasındaki durumunu anlattığından tüm detaylar henüz bu raporda bulunmayacak, detaylı oyun sonuç raporu final döneminde hazırlanacaktır. Oyunun ilk aşaması olan bu seviyede, bir şövalye görünümüne sahip oyuncu ve çöl totem isimli düşman bulunmaktadır. Bu aşamada kullanıcı ara yüzü tasarımı aşamasına henüz geçilmemiştir.

Oyuna Genel Bakış

Oyuncunun görevi çölde düşmanlarla karşılaşan şövalyenin düşmanları mızrak vuruşlarıyla öldürmesi ve bölümü tamamlamasıdır. Projenin ilerleyen kısımlarında düşmanlar tarafından hasar alınacak ve oyuncunun bu hasarlardan korunması ve kaçması gerekecektir. Aksi halde şövalye hayatını kaybedecek ve oyuncu hedefine ulaşamamış olacaktır.

Kullanılan Teknolojiler

Oyun Motoru: Unity 2020.3.19f1

Programlama Dili: C#

Kodlama Editörü: Visual Studio Code 1.62.1

Sınıflar ve Sorumlulukları

Numarası	Sınıf Adı	Sorumlulukları
1	DamageNumberSc.cs	Düşmana verilen hasarları oyuncuya bildirmek ve düşman öldüğünde Die() metodu ile düşmanı oyun sahnesinden kaldırmaktır.
2	EnemyController.cs	Düşmanın sahip olduğu can seviyesini takip eder ve düşmanın içerisinde bulunduğu yaşam, hasar alma ve ölüm durumlarını boolean tipindeki bayraklar ile belirler. Düşman animasyon durumları bu bayraklar aracılığıyla belirlenir.

3	PlayerCombatController.cs	Oyuncunun saldırı sırasındaki olaylarını izlemetedir.
4	PlayerController.cs	Karakterin zıplama, yürüme, yerde olması durumu gibi durumlarını izlemektedir. Oyuncudan alınan girdileri takip ederek animasyon ve hareketlerin oluşmasını sağlar.
5	PlayerData.cs	Karakterin sağlık durumunu kontrol eder. Mevcut, maksimum ve minimum can seviyeleri bu sınıfta tutulabilir ve gerektiğinde getter/setter metotlarıyla can seviyesinde değişiklik yapılabilmektedir.
6	PlayerGroundCheckScript.cs	Oyuncunun yerde olup olmaması durumunu kontrol ederek bunun bir bayrakta tutar ve istendiğinde getter metoduyla ulaşılabilir.
7	PlayerHealth.cs	Karakterin can seviyesinin tutulduğu bar ve üzerindeki can seviyesini rakamsal olarak sürekli kontrol ederek oyuncuya gösterir. Karakterin mevcut durumdaki can seviyesine getter ile ulaşılabilir.
8	RestingZone.cs	Karakter dinlenme bölgesine girdiğinde burada “sol alt” tuşu ile can seviyesini yükseltebilir. Bu işlemin script kodu RestingZone.cs dosyası içerisinde bulunur.

Ana Karakter

Şekil 1 Ana Karakter (Şövalye)

Oyundaki ana karakter Şekil 1'deki şövalyedir.

Düşman Karakteri

Şekil 2 Düşman Karakter (Totem)

Şekil 2'deki Totem, oyundaki düşman karakterdir.

Karakter Silahı

Şekil 3 Şövalye Saldırısı

Karakterin silahı kendi mızrağıdır. Şekil 3'te saldırı anındaki mızrak durumu görülebilir. Mızrak bölgesine giren düşman saldırı devam ettiği sürece hasar alacaktır.

Diğer Nesneler

Oyunda kullanılan diğer nesneler aşağıdaki görsellerde belirtilmiştir.

Şekil 4 Ağaç 1

Şekil 5 Kaktüs 1

Şekil 6 Kaktüs 2

Şekil 7 Kaktüs 3

Şekil 8 Kutu

Şekil 9Taş Blok

Şekil 10 İskelet

Şekil 11 Çimen

Şekil 12Ağaç 2

Zemin Stilleri (TileSets)

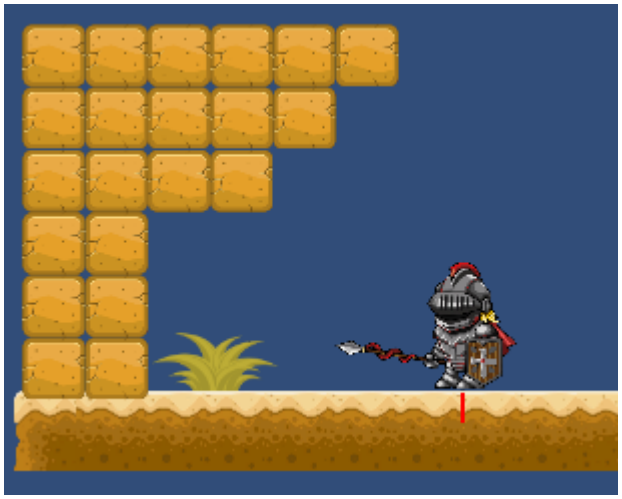
Şekil 13 Zemin Sol

Şekil 14 Zemin Orta

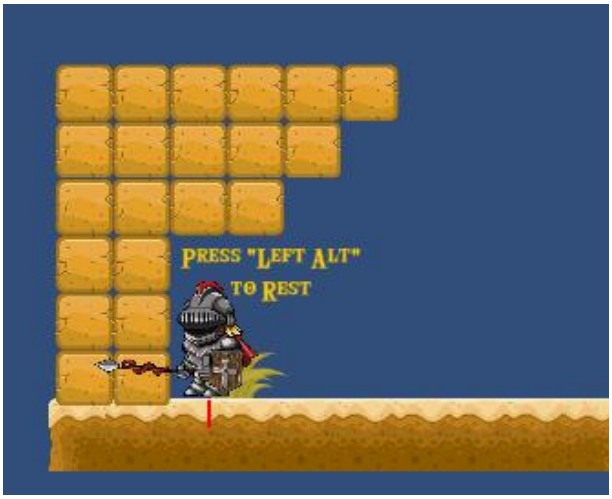
Şekil 15 Zemin Sağ

Zemini oluşturan parçaların Şekil 13'te başlangıç, Şekil 14'te orta kısım ve Şekil 15'te son kısımları görülebilmektedir.

Can Seviyesi Dolum Bölgesi



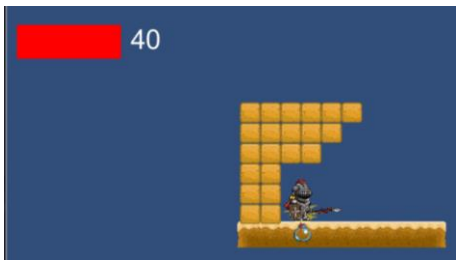
Şekil 16 Can Seviyesi Dolum Bölgesi Deaktif



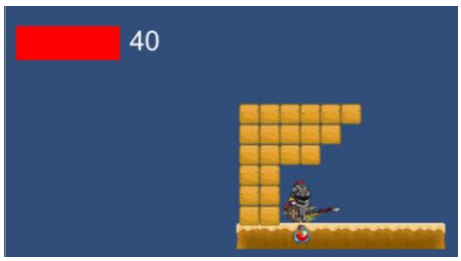
Şekil 17 Can Dolum Bölgesi Aktif

Şekil 16'da karakterin can dolum bölgesi görülmektedir. Oyuncu bu bölgenin dışındayken Şekil 16'daki gibi görünen bölge, karakter bölgeye girdiği anda ekrana gelen "Press 'left alt' to Rest" yazısının ekrana geldiği Şekil 17'deki görünümünü almaktadır.

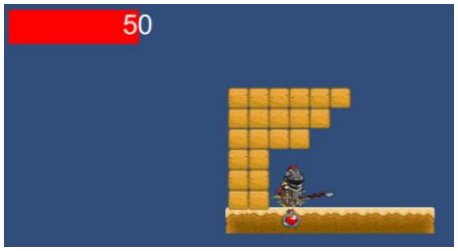
Karakter can dolum bölgesinde iken oyuncu “sol alt” tuşuna basılı tutarsa Şekil 18’deki gibi can dolumu başlayacaktır. Can seviyesi dolumu sırasında animasyon olarak heal pot’un dolduğu sırasıyla Şekil 18, Şekil 19 ve Şekil 20’deki gibi görülecektir.



Şekil 18 Can Dolumu



Şekil 19 Can Dolumu



Şekil 20 Can Dolumu

Karakter Saldırı Durumu

Şekil 21 Hasar Değeri

Karakter düşmana saldırdığında verdiği hasarı ekranda Şekil 21’deki gibi oyuncuya gösterilmektedir.

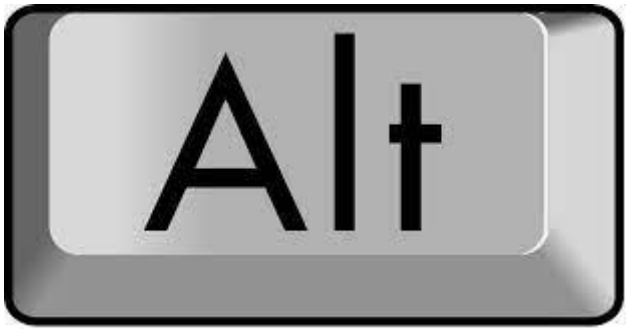
Oyuncu Kontrolleri

Saldırı Tuşu

Şekil 22 Saldırı Tuşu

Oyuncu Şekil 22'de görüldüğü gibi mouse sol tuşuna bastığında karakter saldırıya başlayacaktır.

Can Dolum Tuşu



Şekil 23 Can Dolum Tuşu (Sol Alt)

Oyuncu klavye sol bölümünde bulunan Alt tuşuna bastığında oyuncu dinlenme bölgesinde iken canı dolmaktadır.

Yön Tuşları

Şekil 24 Yön Tuşları

Karakteri sol ve sağ yöne hareket ettirmek için Şekil 24'teki yön tuşları kullanılabilir.

Zıplama Tuşu

Şekil 25 Zıplama Tuşu (Space)

Oyuncu karakterin zıplaması için Space tuşuna basmalıdır (Şekil 25).

Oyun Sahnesi

Şekil 26 Oyun Sahnesi

Oyun sahnesi genel görünümü Şekil 26'da görüldüğü gibidir.

Kameralar

Oyuncu takip kamerası olarak Unity eklentisi olan Cinemachine özelliğinden yararlanılmıştır. Buna göre oyuncu hareketleri kamera tarafından takip edilecek ve karakterin bulunduğu bölgeler oyuncuya gösterilebilecektir.

Kodlar

DamageNumberSc.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class DamageNumberSc : MonoBehaviour
{
    void Start()
    {

        Die();
    }
    void Update()
    {

    }

    void ShowDamage(float value)
    {
        this.GetComponent<TMP_Text>().text = value.ToString();
        if(value>0f)
        {
            this.GetComponent<TMP_Text>().color = Color.red;
        }
        else
        {
            this.GetComponent<TMP_Text>().color = Color.green;
        }
    }

    private void Die()
    {
        Destroy(gameObject,1f);
    }
}
```

EnemyController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyController : MonoBehaviour
{

    private float maxHealth = 100f;
    private float currentHealth;

    private Animator enemyAnimator;

    //States
    private bool isIdle = true;
    private bool isHurt = false;
    private bool isDead = false;


    private void Awake()
    {
        enemyAnimator = this.GetComponent<Animator>();
    }


    private void Start()
    {
        currentHealth = maxHealth;
        isIdle = true;
        isHurt = false;
        isDead = false;
    }

    private void Damage(float[] attackDetails)
    {
        float damageValue = attackDetails[0];

        if(currentHealth - damageValue < 0f)
        {
            if(isDead==false)
            {
            //enemy dies
            Debug.Log("Enemy Dies");
            isDead = true;
            enemyAnimator.SetBool("isDead", isDead);
            //Die();
            }
        }
```

```
        else
        {
            //Enemy lives
            currentHealth-=damageValue;
            Debug.Log("Enemy Current Health: " + currentHealth);

            isHurt = true;
            isIdle = false;
            enemyAnimator.SetBool("isHurt", isHurt);
            enemyAnimator.SetBool("isIdle", isIdle);
        }
    }

    private void FinishHurtAnimation()
    {
        isHurt = false;
        isIdle = true;
        enemyAnimator.SetBool("isHurt", isHurt);
        enemyAnimator.SetBool("isIdle", isIdle);
    }

    private void Die()
    {
        Destroy(gameObject);
    }
}
```

PlayerCombatController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerCombatController : MonoBehaviour
{
    [SerializeField] private GameObject DamageNumbersObject;

    [SerializeField] private bool canAttack = true;
    [SerializeField] private float attackTimer;
    [SerializeField] private float attackRadius, attackDamage;


    [SerializeField] private Transform attackPoint;

    [SerializeField] LayerMask isDamageable;

    private float lastInputTime;

    private Animator combatAnimator;
```

```
[SerializeField] private bool isAttacking, gotAttackInput, Attack;

private float[] attackDetails = new float[3];


void Awake()
{
}

// Start is called before the first frame update
void Start()
{
    combatAnimator = this.GetComponent<Animator>();

    canAttack = true;
    combatAnimator.SetBool("canAttack",canAttack);
}

// Update is called once per frame
void Update()
{
    CheckAttackInput();
    CheckAttack();
}

private void onDrawGizmos()
{
    Gizmos.DrawWireSphere(attackPoint.position, attackRadius);
}

private void CheckAttackInput()
{
    if(Input.GetButton("Fire1") && canAttack)
    {
        gotAttackInput = true;
        lastInputTime = Time.time;
    }
}

private void FinishAttack()
{
    isAttacking = false;
    Attack = false;
    combatAnimator.SetBool("isAttacking",isAttacking);
    combatAnimator.SetBool("Attack",false);
}


private void CheckAttackHitBox()
```

```
    {
        Collider2D[] hitObjects =
Physics2D.OverlapCircleAll(attackPoint.transform.position, attackRadius,
isDamageable);

        attackDetails[0] = attackDamage;
        attackDetails[1] = this.transform.position.x;
        attackDetails[2] = this.transform.position.y;


        foreach(Collider2D collider in hitObjects)
        {
            //Debug.Log("Collided");
            //Debug.Log(collider.gameObject);

            collider.gameObject.SendMessage("Damage", attackDetails);

            //Damage Numbers
            GameObject damageNumberClone = Instantiate(DamageNumbersObject,
this.transform.position, Quaternion.identity);
            damageNumberClone.SendMessage("ShowDamage",attackDamage);
        }
    }

    private void CheckAttack()
    {
        if(gotAttackInput == true && isAttacking == false)
        {
            isAttacking = true;
            Attack = true;

            combatAnimator.SetBool("Attack", true );
            combatAnimator.SetBool("isAttacking", isAttacking);

            gotAttackInput = false;
        }
    }
}
```

PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    private Rigidbody2D playerRB;
    private BoxCollider2D playerCollider;
```

```
    private SpriteRenderer spriteRenderer;
    private Animator playerAnimator;


    [SerializeField] private float walkingSpeed = 2.5f;
    [SerializeField] private float jumpAmount = 5f;
    [SerializeField] private float groundCheckPerimeter = 0.2f;

    //Ground Check
    public GameObject GroundChecker;
    private PlayerGroundCheckScript groundCheckScript;


    //player input values
    private float hInput;
    private bool jumpInput;

    //Player State
    [SerializeField] private bool isIdle = true;
    [SerializeField] private bool isWalking = false;
    [SerializeField] private bool isOnGround = false;
    [SerializeField] private bool isJumping = false;

    //Slope Properties
    private float slopeDownAngle;
    private float slopePreviousAngle=0f;
    private Vector2 slopeNormalPerp;

    //Direction Control
    private bool playerDir = true; //true: 1 right direction, false: 0 left
direction

    void Awake()
    {
        playerRB = this.GetComponent <Rigidbody2D>();
        playerCollider = this.GetComponent <BoxCollider2D>();
        spriteRenderer = this.GetComponent <SpriteRenderer>();
        playerAnimator = this.GetComponent <Animator>();

        groundCheckScript =
GroundChecker.GetComponent<PlayerGroundCheckScript>();
    }
    // Start is called before the first frame update
    void Start()
    {
    }
```

```
    // Update is called once per frame
    void Update()
    {
        CheckInput();
        CheckMovement();
        UpdateAnimations();
    }

    void FixedUpdate()
    {
        ApplyMovement();
        CheckGround();
        CheckSlope();
    }

    private void UpdateAnimations()
    {
        playerAnimator.SetBool("isWalking", isWalking);
        playerAnimator.SetBool("isIdle", isIdle);
        playerAnimator.SetBool("isJumping", isJumping);
        playerAnimator.SetBool("isOnGround", isOnGround);


    }


    void ApplyMovement()

    {
        this.transform.position += new Vector3(hInput * walkingSpeed *
Time.deltaTime, 0f, 0f);
        playerRB.velocity = new Vector2(hInput*walkingSpeed ,
playerRB.velocity.y);
        if (jumpInput == true) { jumpInput = false; Jump(); }
    }

    private void CheckSlope()
    {
        RaycastHit2D groundHit = Physics2D.Raycast(playerCollider.bounds.center -
new Vector3(0f, playerCollider.bounds.extents.y + 0.05f, 0f), Vector2.down,
0.5f);
        Debug.DrawRay(playerCollider.bounds.center - new
Vector3(0f,playerCollider.bounds.extents.y+0.05f,0f), Vector2.down*0.5f,
Color.red);

        if(groundHit.collider != null)
        {
            slopeDownAngle = Vector2.Angle(groundHit.normal, Vector2.up);
            slopeNormalPerp = Vector2.Perpendicular(groundHit.normal).normalized;
```

```
            Debug.DrawRay(groundHit.point, slopeNormalPerp, Color.blue);
            Debug.DrawRay(groundHit.point, groundHit.normal, Color.green);

        }
        else
        {
            transform.eulerAngles = new Vector3(transform.eulerAngles.x,
transform.eulerAngles.y, 0f);
        }


    }

   private void CheckMovement()
    {
        if(playerDir && hInput < -0.15f)
        {
            Flip();
        }
        else if (!playerDir && hInput > 0.15f)
        {
            Flip();
        }

        if(Mathf.Abs(playerRB.velocity.x) > 0.01f)
        {
            isIdle = false;
            if(isJumping==false && isOnGround==true )
            {
                isWalking = true;
            }
        }
        else
        {
            if(isOnGround==true)
            {
                isIdle = true;
                isJumping = false;
            }
            isWalking = false;

        }
    }

    private void Flip()
    {
        playerDir = !playerDir;

        this.transform.Rotate(0f,180f,0f);
    }
```

```
    void CheckInput()
    {
        hInput = Input.GetAxis("Horizontal");

        //Jump
        if (Input.GetButton("Jump"))
        {
            jumpInput = true;
        }


    }
    void Jump()
    {
        if (isOnGround == true && isJumping == false)
        {
            playerRB.velocity = new Vector2(playerRB.velocity.x, jumpAmount);
            isJumping = true;
            isWalking = false;
            isIdle = false;
        }

    }

    void CheckGround()
    {
        isOnGround = groundCheckScript.getIsOnGround();

        }

    }
```

PlayerData.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class PlayerData : MonoBehaviour
{
    private float maxHealth = 100f;
    private float minHealth = 0f;
    private float currHealth = 40f;
```

```
    public void setHealth(float setAmount)
    {
        this.currHealth = setAmount;
        if(this.currHealth > this.maxHealth)
        {
            this.currHealth = this.maxHealth;
        }
        if(this.currHealth < this.minHealth)
        {
            this.currHealth = this.minHealth;
        }
    }

    public float getMaxHealth()
    {
        return this.maxHealth;
    }

    public float getMinHealth()
    {
        return this.minHealth;
    }

    public float getCurrHealth()
    {
        return this.currHealth;
    }
}
```

PlayerGroundCheckScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerGroundCheckScript : MonoBehaviour
{
    private bool isOnGround = false;
    private int enteredContactCount = 0;

    private void OnCollisionEnter2D(Collision2D other)
    {
        isOnGround = true;
        enteredContactCount++;
    }
```

```
    Private void OnCollisionExit2D(Collision2D other)
    {
        enteredContactCount--;

        if (enteredContactCount == 0)
        {
            isOnGround = false;
        }

    }

    public bool getIsOnGround()
    {
        return this.isOnGround;
    }

}
```

PlayerHealth.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayerHealth : MonoBehaviour
{
    private PlayerData Player1data;

    //HealthBar Game Objects
    [SerializeField] private float HealthBarWidth = 400f;
    public GameObject HealthBarImageObj;
    public GameObject HealthBarTextObj;

    private RectTransform HealthBarTransform;
    private Text HealthBarText;

    void Awake()
    {
        Player1data = new PlayerData();

        HealthBarTransform = HealthBarImageObj.GetComponent<RectTransform>();
        HealthBarText = HealthBarTextObj.GetComponent<Text>();
    }
```

```
    void Start()
    {
        //Debug.Log("Player Health: " + Player1data.getCurrHealth());
        HealthBarText.text = Player1data.getCurrHealth().ToString();

        HealthBarTransform.sizeDelta = new Vector2(
(Player1data.getCurrHealth()/Player1data.getMaxHealth())*HealthBarWidth ,
HealthBarTransform.sizeDelta.y);
    }

    // Update is called once per frame
    void Update()
    {
        UpdateHealthBar();
    }

    void UpdateHealthBar()
    {
        if (Input.GetKeyDown(KeyCode.KeypadPlus))
        {
            Player1data.setHealth(Player1data.getCurrHealth() + 10f);
            HealthBarText.text = Player1data.getCurrHealth().ToString();
            HealthBarTransform.sizeDelta = new
Vector2((Player1data.getCurrHealth() / Player1data.getMaxHealth()) *
HealthBarWidth, HealthBarTransform.sizeDelta.y);
        }
        if (Input.GetKeyDown(KeyCode.KeypadMinus))
        {
            Player1data.setHealth(Player1data.getCurrHealth() - 10f);
            HealthBarText.text = Player1data.getCurrHealth().ToString();
            HealthBarTransform.sizeDelta = new
Vector2((Player1data.getCurrHealth() / Player1data.getMaxHealth()) *
HealthBarWidth, HealthBarTransform.sizeDelta.y);
        }
    }

    public void changeHealth(float Amount)
    {
        Player1data.setHealth(Player1data.getCurrHealth() + Amount);
        HealthBarText.text = Player1data.getCurrHealth().ToString();
        HealthBarTransform.sizeDelta = new Vector2(Player1data.getCurrHealth() /
Player1data.getMaxHealth() * HealthBarWidth, HealthBarTransform.sizeDelta.y);

    }

    public float getPlayerHealth()
    {
        return Player1data.getCurrHealth();
    }
}
```

RestingZone.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RestingZone : MonoBehaviour
{

    [SerializeField] private GameObject TextMeshProObj;
    [SerializeField] private GameObject GrassNormal;


    private GameObject PlayerObject;
    private PlayerHealth PlayerHealthSc;



    private bool isPlayerEntered = false;
    private bool healState = false;

    private float timerMax = 2f;
    private float timer;


    // Start is called before the first frame update
    void Start()
    {
        TextMeshProObj.SetActive(false);
        GrassNormal.SetActive(false);

        timer = timerMax;

    }

    private void OnTriggerEnter2D(Collider2D other)
    {

        if (other.gameObject.CompareTag("Player"))
        {
            PlayerObject = other.gameObject;

            PlayerHealthSc = PlayerObject.GetComponent<PlayerHealth>();
```

```
            isPlayerEntered = true;
            Debug.Log("You Entered Resting Zone");
            TextMeshProObj.SetActive(true);
        }
        GrassNormal.SetActive(false);

    }

    private void OnTriggerExit2D(Collider2D other)
    {

        healState = false;

        ///if (other.gameObject.CompareTag("Player"))
        if (other.gameObject == PlayerObject)
        {
            isPlayerEntered = false;
            Debug.Log("You Exited Resting Zone");
            TextMeshProObj.SetActive(false);
            GrassNormal.SetActive(false);

        }
    }
    // Update is called once per frame
    void Update()
    {
        if (isPlayerEntered && Input.GetButton("Fire2"))
        {
            if(healState==false) healState = true;
            TextMeshProObj.SetActive(false);

            if (healState == true)
            {
                timer = timer - Time.deltaTime;
                if(timer <= 0)
                {
                    HealPlayer();
                    timer = timerMax;
                }
                Debug.Log("Heal Getting...");
                GrassNormal.SetActive(true);


            }
        }
        else
                GrassNormal.SetActive(false);
    }

    private void HealPlayer()
```

```
    {

        if (PlayerHealthSc.getPlayerHealth() >= 100)
        {
            healState = false;
        }
        PlayerHealthSc.changeHealth(10f);
    }
}
```

Kaynakça

1) “2d Fantasy Knıght Free Character Sprite” isimli şövalye karakterine linki verilen site üzerinden ulaşılabilir:
https://craftpix.net/freebies/2d-fantasy-knight-free-sprite-sheets/

2) “Free Desert Platformer Tıleset”e linki verilen site üzerinden ulaşılabilir:
https://www.gameart2d.com/uploads/3/0/9/1/30917885/deserttileset.zip

3) “The Wild Breath of Zelda” yazı tipine linki verilen site üzerinden ulaşılabilir:
http://www.dafont.com/the-wild-breath-of-zelda.font

4) “buff_totem_sprite_sheet_v1.1.png” düşman karakterine linki verilen site üzerinden ulaşılabilir:
https://opengameart.org/content/pixel-art-totem-sprites