CAPÍTULO III: DESARROLLO

3.1 Capturas de la Aplicación (Documentación completa del desarrollo, Scripts, Sprites, Prefabs e imágenes)

Script Camera Movement

```
using UnityEngine;
using System.Collections;

public class CameraMovement : MonoBehaviour
{
    /// <summary>

    /// </summary>
    [SerializeField]
    private float cameraSpeed = 0;

    /// <summary>

    /// </summary>
    private float xMax;

    /// <summary>

    /// </summary>
    private float yMin;


    private void Update()
    {
        GetInput();
    }

    private void GetInput()
    {

        if (Input.GetKey(KeyCode.W))
        {
            transform.Translate(Vector3.up * cameraSpeed * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.A))
        {
            transform.Translate(Vector3.left * cameraSpeed * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.S))
        {
            transform.Translate(Vector3.down * cameraSpeed * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.D))
        {
            transform.Translate(Vector3.right * cameraSpeed * Time.deltaTime);
        }

        transform.position = new Vector3(Mathf.Clamp(transform.position.x, 0, xMax), Mathf.Clamp(transform.position.y, yMin, 0), -10);
```

```
    transform.position = new Vector3(Mathf.Clamp(transform.position.x, 0, xMax), Mathf.Clamp(transform.position.y,yMin,0),-10);
}

/// <summary>

/// </summary>
/// <param name="maxtile"></param>
public void SetLimits(Vector3 maxtile)
{

    Vector3 p = Camera.main.ViewportToWorldPoint(new Vector3(1, 0));


    xMax = maxtile.x - p.x;
    yMin = maxtile.y - p.y;

}
```

Script #2

```
using UnityEngine;

public class Hover : Singleton<Hover>
{
    /// <summary>

    /// </summary>
    private SpriteRenderer spriteRenderer;

    /// <summary>

    /// </summary>
    private SpriteRenderer rangeSpriteRenderer;


    void Start ()
    {

        this.spriteRenderer = GetComponent<SpriteRenderer>();

        this.rangeSpriteRenderer = transform.GetChild(0).GetComponent<SpriteRenderer>();
    }


    void Update ()
    {

        FollowMouse();
    }

    /// <summary>

    /// </summary>
    private void FollowMouse()
    {
        if (spriteRenderer.enabled)
        {

            transform.position = Camera.main.ScreenToWorldPoint(Input.mousePosition);


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

    }

```

```
46
47      /// <summary>
48
49      /// </summary>
50      /// <param name="sprite"></param>
51      public void Activate(Sprite sprite)
52      {
53
54          this.spriteRenderer.sprite = sprite;
55
56
57          spriteRenderer.enabled = true;
58
59          rangeSpriteRenderer.enabled = true;
60      }
61
62      /// <summary>
63
64      /// </summary>
65      public void Deactivate()
66      {
67
68          spriteRenderer.enabled = false;
69
70          rangeSpriteRenderer.enabled = false;
71
72          GameManager.Instance.ClickedBtn = null;
73
74
75      }
76  }
77
```

Script Level Manager

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

/// <summary>

/// </summary>
public class LevelManager : Singleton<LevelManager>
{
    /// <summary>
    /// An array of tilePrefabs, these are used for creating the tiles in the game
    /// </summary>
    [SerializeField]
    private GameObject[] tilePrefabs;

    /// <summary>

    /// </summary>
    [SerializeField]
    private CameraMovement cameraMovement;

    /// <summary>

    /// </summary>
    [SerializeField]
    private Transform map;

    /// <summary>

    /// </summary>
    private Point blueSpawn, redSpawn;

    /// <summary>

    /// </summary>
    [SerializeField]
    private GameObject bluePortalPrefab;

    /// <summary>

    /// </summary>
    [SerializeField]
    private GameObject redPortalPrefab;

    public Portal BluePortal { get; set; }

    /// <summary>
```

```
46
47      /// <summary>
48
49      /// </summary>
50      private Stack<Node> fullPath;
51
52      private Point mapSize;
53
54      /// <summary>
55
56      /// </summary>
57      public Dictionary<Point, TileScript> Tiles { get; set; }
58
59      /// <summary>
60
61      /// </summary>
62      public float TileSize
63      {
64          get { return tilePrefabs[0].GetComponent<SpriteRenderer>().sprite.bounds.size.x; }
65      }
66
67      /// <summary>
68
69      /// </summary>
70      public Stack<Node> Path
71      {
72          get
73          {
74              if (fullPath == null)
75              {
76                  GeneratePath();
77              }
78
79              return new Stack<Node>(new Stack<Node>(fullPath));
80          }
81      }
82
83      public Point BlueSpawn
84      {
85          get
86          {
87              return blueSpawn;
88          }
89      }
90
```

```
91      void Start()
92      {
93
94          CreateLevel();
95      }
96
97
98      void Update()
99      {
100
101     }
102
103     /// <summary>
104
105     /// </summary>
106     private void CreateLevel()
107     {
108         Tiles = new Dictionary<Point, TileScript>();
109
110
111         string[] mapData = ReadLevelText();
112
113         mapSize = new Point(mapData[0].ToCharArray().Length, mapData.Length);
114
115
116         int mapX = mapData[0].ToCharArray().Length;
117
118
119         int mapY = mapData.Length;
120
121         Vector3 maxTile = Vector3.zero;
122
123
124         Vector3 worldStart = Camera.main.ScreenToWorldPoint(new Vector3(0, Screen.height));
125
126         for (int y = 0; y < mapY; y++)
127         {
128             char[] newTiles = mapData[y].ToCharArray();
129
130             for (int x = 0; x < mapX; x++)
131             {
132
133                 PlaceTile(newTiles[x].ToString(), x, y, worldStart);
134             }
135         }
136
137         maxTile = Tiles[new Point(mapX - 1, mapY - 1)].transform.position;
```

```
        cameraMovement.SetLimits(new Vector3(maxTile.x + TileSize, maxTile.y - TileSize));

        SpawnPortals();

    }

    /// <summary>

    /// </summary>
    /// <param name="tileType">The type of tile to palce for example 0</param>
    /// <param name="x">x position of the tile</param>
    /// <param name="y">y position of the tile</param>
    /// <param name="worldStart">The world start position</param>
    private void PlaceTile(string tileType, int x, int y, Vector3 worldStart)
    {

        int tileIndex = int.Parse(tileType);


        TileScript newTile = Instantiate(tilePrefabs[tileIndex]).GetComponent<TileScript>();


        newTile.Setup(new Point(x, y), new Vector3(worldStart.x + (TileSize * x), worldStart.y - (TileSize * y), 0),map);


    }

    /// <summary>

    /// </summary>
    /// <returns>A string array with indicators of the tiles to place</returns>
    private string[] ReadLevelText()
    {

        TextAsset bindata = Resources.Load("Level") as TextAsset;


        string data = bindata.text.Replace(Environment.NewLine, string.Empty);


        return data.Split('-');
    }

    /// <summary>
```

```
    /// </summary>
    private void SpawnPortals()
    {

        blueSpawn = new Point(0, 0);
        GameObject tmp = (GameObject)Instantiate(bluePortalPrefab, Tiles[BlueSpawn].GetComponent<TileScript>().WorldPosition, Quaternion.identity);
        BluePortal = tmp.GetComponent<Portal>();
        BluePortal.name = "BluePortal";


        redSpawn = new Point(11, 6);
        Instantiate(redPortalPrefab, Tiles[redSpawn].GetComponent<TileScript>().WorldPosition, Quaternion.identity);
    }

    public bool InBounds(Point position)
    {
        return position.X >= 0 && position.Y >= 0 && position.X < mapSize.X && position.Y < mapSize.Y;
    }

    /// <summary>

    /// </summary>
    public void GeneratePath()
    {

        fullPath = AStar.GetPath(BlueSpawn, redSpawn);
    }
}

```

Script Monster

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

public class Monster : MonoBehaviour
{

    /// <summary>

    /// </summary>
    [SerializeField]
    private float speed;

    /// <summary>

    /// </summary>
    private Stack<Node> path;

    /// <summary>

    /// </summary>
    protected Animator myAnimator;

    /// <summary>

    /// </summary>
    public Point GridPosition { get; set; }

    /// <summary>

    /// </summary>
    public bool IsActive { get; set; }

    /// <summary>

    /// </summary>
    private Vector3 destination;

    private void Awake()
    {

        myAnimator = GetComponent<Animator>();
    }

    private void Update()
    {
        Move();
    }
```

```
46      {
47          Move();
48      }
49
50      /// <summary>
51
52      /// </summary>
53      public void Spawn()
54      {
55          transform.position = LevelManager.Instance.BluePortal.transform.position;
56
57
58
59          StartCoroutine(Scale(new Vector3(0.1f, 0.1f), new Vector3(1, 1),false));
60
61          SetPath(LevelManager.Instance.Path);
62      }
63
64      /// <summary>
65
66      /// </summary>
67      /// <param name="from">start scale</param>
68      /// <param name="to">end scale</param>
69      /// <returns></returns>
70      public IEnumerator Scale(Vector3 from, Vector3 to, bool remove)
71      {
72
73          float progress = 0;
74
75
76          while (progress <= 1)
77          {
78
79              transform.localScale = Vector3.Lerp(from, to, progress);
80              progress += Time.deltaTime;
81              yield return null;
82          }
83
84
85          transform.localScale = to;
86
87          IsActive = true;
88
89          if (remove)
90          {
91              Release();
92          }
```

```
94
95      /// <summary>
96
97      /// </summary>
98      public void Move()
99      {
100         if (IsActive)
101         {
102
103             transform.position = Vector2.MoveTowards(transform.position, destination, speed * Time.deltaTime);
104
105             if (transform.position == destination)
106             {
107
108                 if (path != null && path.Count > 0)
109                 {
110
111                     Animate(GridPosition, path.Peek().GridPosition);
112
113
114                     GridPosition = path.Peek().GridPosition;
115
116                     destination = path.Pop().WorldPosition;
117
118                 }
119             }
120         }
121
122     }
123
124     /// <summary>
125
126     /// </summary>
127     /// <param name="newPath">The unit's new path</param>
128     /// <param name="active">Indicates if the unit is active</param>
129     public void SetPath(Stack<Node> newPath)
130     {
131         if (newPath != null)
132         {
133
134             this.path = newPath;
135
136             Animate(GridPosition, path.Peek().GridPosition);
137
138
139             GridPosition = path.Peek().GridPosition;
```

```
141
142             destination = path.Pop().WorldPosition;
143         }
144     }
145
146     /// <summary>
147
148     /// </summary>
149     /// <param name="currentPos"></param>
150     /// <param name="newPos"></param>
151     public void Animate(Point currentPos, Point newPos)
152     {
153
154
155
156         if (currentPos.Y > newPos.Y)
157         {
158             myAnimator.SetInteger("Horizontal", 0);
159
160             myAnimator.SetInteger("Vertical", 1);
161         }
162
163         else if (currentPos.Y < newPos.Y)
164         {
165             myAnimator.SetInteger("Horizontal", 0);
166             myAnimator.SetInteger("Vertical", -1);
167         }
168
169         if (currentPos.Y == newPos.Y)
170         {
171
172             if (currentPos.X > newPos.X)
173             {
174                 myAnimator.SetInteger("Vertical", 0);
175                 myAnimator.SetInteger("Horizontal", -1);
176             }
177
178             else if (currentPos.X < newPos.X)
179             {
180                 myAnimator.SetInteger("Vertical", 0);
181                 myAnimator.SetInteger("Horizontal", 1);
182             }
183         }
184     }
185
186     /// <summary>
187
```

```
185
186     /// <summary>
187
188     /// </summary>
189     /// <param name="other"></param>
190     private void OnTriggerEnter2D(Collider2D other)
191     {
192         if (other.tag == "RedPortal")
193         {
194
195             StartCoroutine(Scale(new Vector3(1, 1), new Vector3(0.1f, 0.1f),true));
196
197
198             other.GetComponent<Portal>().Open();
199
200             GameManager.Instance.Lives--;
201         }
202     }
203
204     /// <summary>
205
206     /// </summary>
207     private void Release()
208     {
209
210         IsActive = false;
211
212
213         GridPosition = LevelManager.Instance.BlueSpawn;
214
215
216         GameManager.Instance.RemoveMonster(this);
217
218
219         GameManager.Instance.Pool.ReleaseObject(gameObject);
220     }
221
222
223 }
224
```

Script Portal

C: > Users > brayr > Downloads > TowerDefense_9_5 > TDTut > Assets > Scripts > Portal.cs

```
using UnityEngine;
using System.Collections;

public class Portal : MonoBehaviour {

    /// <summary>

    /// </summary>
    private Animator myAnimator;


    void Start () {


        myAnimator = GetComponent<Animator>();
    }

    /// <summary>

    /// </summary>
    public void Open()
    {

        myAnimator.SetTrigger("Open");
    }
}

```

Script Proyectiles

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

public class Projectile : MonoBehaviour {

    private Monster target;

    private Tower parent;


    void Start ()
    {

    }


    void Update ()
    {
        MoveToTarget();
    }

    public void Initialize(Tower parent)
    {
        this.target = parent.Target;
        this.parent = parent;
    }

    private void MoveToTarget()
    {
        if (target != null && target.IsActive)
        {

            transform.position = Vector3.MoveTowards(transform.position, target.transform.position, Time.deltaTime * parent.ProjectileSpeed);


            Vector2 dir = target.transform.position - transform.position;


            float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;

            transform.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
        }
        else if (!target.IsActive)
        {
            GameManager.Instance.Pool.ReleaseObject(gameObject);
        }
    }
}
```

Script Cuadros de fondo

```
using UnityEngine;
using UnityEngine.EventSystems;

/// <summary>

/// </summary>
public class TileScript : MonoBehaviour
{
    /// <summary>

    /// </summary>
    public Point GridPosition { get; private set; }

    public bool IsEmpty { get; private set; }

    private Tower myTower;

    /// <summary>

    /// </summary>
    private Color32 fullColor = new Color32(255, 118, 118, 255);

    /// <summary>

    /// </summary>
    private Color32 emptyColor = new Color32(96, 255, 90, 255);

    private SpriteRenderer spriteRenderer;

    public bool Walkable { get; set; }

    public bool Debugging { get; set; }

    /// <summary>

    /// </summary>
    public Vector2 WorldPosition
    {
        get
        {
            return new Vector2(transform.position.x + (GetComponent<SpriteRenderer>().bounds.size.x / 2), transform.position.y - (GetComponent<SpriteRenderer>().bounds.size.y / 2));
        }
    }

    void Start()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
    }

```

```
48      |
49
50
51      void Update()
52      {
53
54      }
55
56      /// <summary>
57
58      /// </summary>
59      /// <param name="gridPos">The tiles grid position</param>
60      /// <param name="worldPos">The tiles world postion</param>
61      public void Setup(Point gridPos, Vector3 worldPos, Transform parent)
62      {
63          Walkable = true;
64          IsEmpty = true;
65          this.GridPosition = gridPos;
66          transform.position = worldPos;
67          transform.SetParent(parent);
68          LevelManager.Instance.Tiles.Add(gridPos, this);
69      }
70
71      /// <summary>
72
73      /// </summary>
74      private void OnMouseOver()
75      {
76          if (!EventSystem.current.IsPointerOverGameObject() && GameManager.Instance.ClickedBtn != null)
77          {
78              if (IsEmpty && !Debugging)
79              {
80                  ColorTile(emptyColor);
81              }
82              if (!IsEmpty && !Debugging)
83              {
84                  ColorTile(fullColor);
85              }
86              else if (Input.GetMouseButtonDown(0))
87              {
88                  PlaceTower();
89              }
90          }
91          else if (!EventSystem.current.IsPointerOverGameObject() && GameManager.Instance.ClickedBtn == null && Input.GetMouseButtonDown(0))
92          {
93              if (myTower != null)
94              {
95                  GameManager.Instance.SelectTower(myTower);
96              }
```

```
96              }
97              else
98              {
99                  GameManager.Instance.DeselectTower();
100             }
101         }
102
103     }
104
105     private void OnMouseExit()
106     {
107         if (!Debugging)
108         {
109             ColorTile(Color.white);
110         }
111
112     }
113
114     /// <summary>
115     /// </summary>
116     private void PlaceTower()
117     {
118
119         GameObject tower = (GameObject)Instantiate(GameManager.Instance.ClickedBtn.TowerPrefab, transform.position, Quaternion.identity);
120
121
122         tower.GetComponent<SpriteRenderer>().sortingOrder = GridPosition.Y;
123
124
125         tower.transform.SetParent(transform);
126
127         this.myTower = tower.transform.GetChild(0).GetComponent<Tower>();
128
129
130         IsEmpty = false;
131
132
133         ColorTile(Color.white);
134
135
136         GameManager.Instance.BuyTower();
137
138         Walkable = false;
139     }
140
141     /// <summary>
142
143     /// </summary>
144     /// <param name="newColor"></param>
```

```
        /// </summary>
        /// <param name="newColor"></param>
        private void ColorTile(Color newColor)
        {

            spriteRenderer.color = newColor;
        }
    }
```

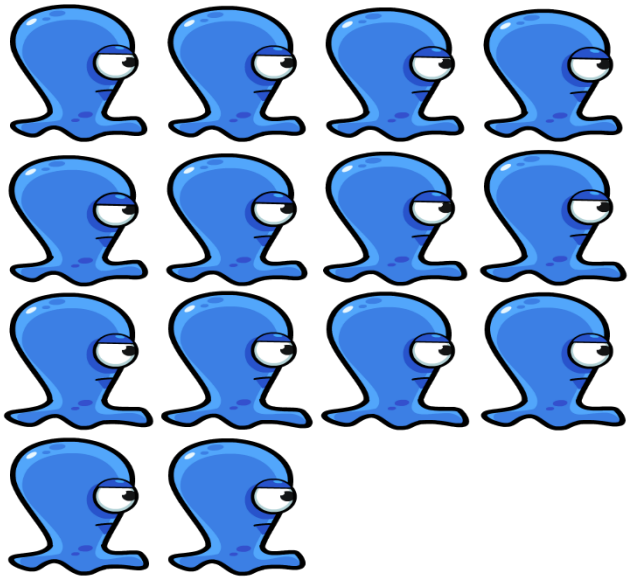
Script Tower

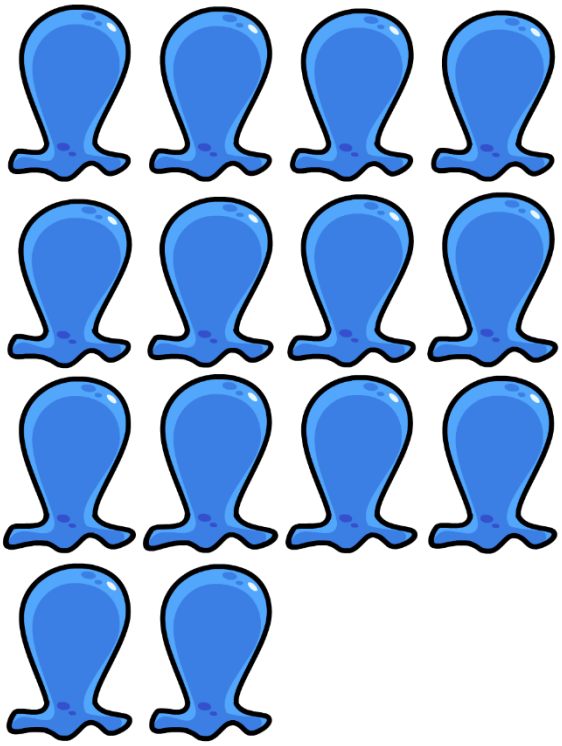
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Tower : MonoBehaviour {

    /// <summary>

    /// </summary>
    [SerializeField]
    private string projectileType;

    [SerializeField]
    private float projectileSpeed;

    private Animator myAnimator;

    public float ProjectileSpeed
    {
        get { return projectileSpeed; }
    }

    /// <summary>

    /// </summary>
    private SpriteRenderer mySpriteRenderer;

    /// <summary>

    /// </summary>
    private Monster target;

    public Monster Target
    {
        get { return target; }
    }

    /// <summary>

    /// </summary>
    private Queue<Monster> monsters = new Queue<Monster>();

    private bool canAttack = true;

    private float attackTimer;

    [SerializeField]
    private float attackCooldown;

```

```
49
50
51      void Awake()
52      {
53          myAnimator = transform.parent.GetComponent<Animator>();
54          mySpriteRenderer = GetComponent<SpriteRenderer>();
55      }
56
57      void Update ()
58      {
59
60          Attack();
61      }
62
63      /// <summary>
64
65      /// </summary>
66      public void Select()
67      {
68          mySpriteRenderer.enabled = !mySpriteRenderer.enabled;
69      }
70
71      /// <summary>
72
73      /// </summary>
74      private void Attack()
75      {
76          if (!canAttack)
77          {
78
79              attackTimer += Time.deltaTime;
80
81              if (attackTimer >= attackCooldown)
82              {
83                  canAttack = true;
84                  attackTimer = 0;
85              }
86          }
87
88          if (target == null && monsters.Count > 0)
89          {
90              target = monsters.Dequeue();
91
92          }
93          if (target != null && target.IsActive)
94          {
95              if (canAttack)
96              {
```

```
91
92          }
93          if (target != null && target.IsActive)
94          {
95              if (canAttack)
96              {
97                  Shoot();
98
99                  myAnimator.SetTrigger("Attack");
100
101                 canAttack = false;
102             }
103
104         }
105     }
106
107     /// <summary>
108
109     /// </summary>
110     private void Shoot()
111     {
112
113         Projectile projectile = GameManager.Instance.Pool.GetObject(projectileType).GetComponent<Projectile>();
114
115
116         projectile.transform.position = transform.position;
117
118         projectile.Initialize(this);
119     }
120
121     public void OnTriggerEnter2D(Collider2D other)
122     {
123         if (other.tag == "Monster")
124         {
125             monsters.Enqueue(other.GetComponent<Monster>());
126         }
127     }
128
129     public void OnTriggerExit2D(Collider2D other)
130     {
131         if (other.tag == "Monster")
132         {
133             target = null;
134         }
135     }
136 }
137
```

Sprite Monster

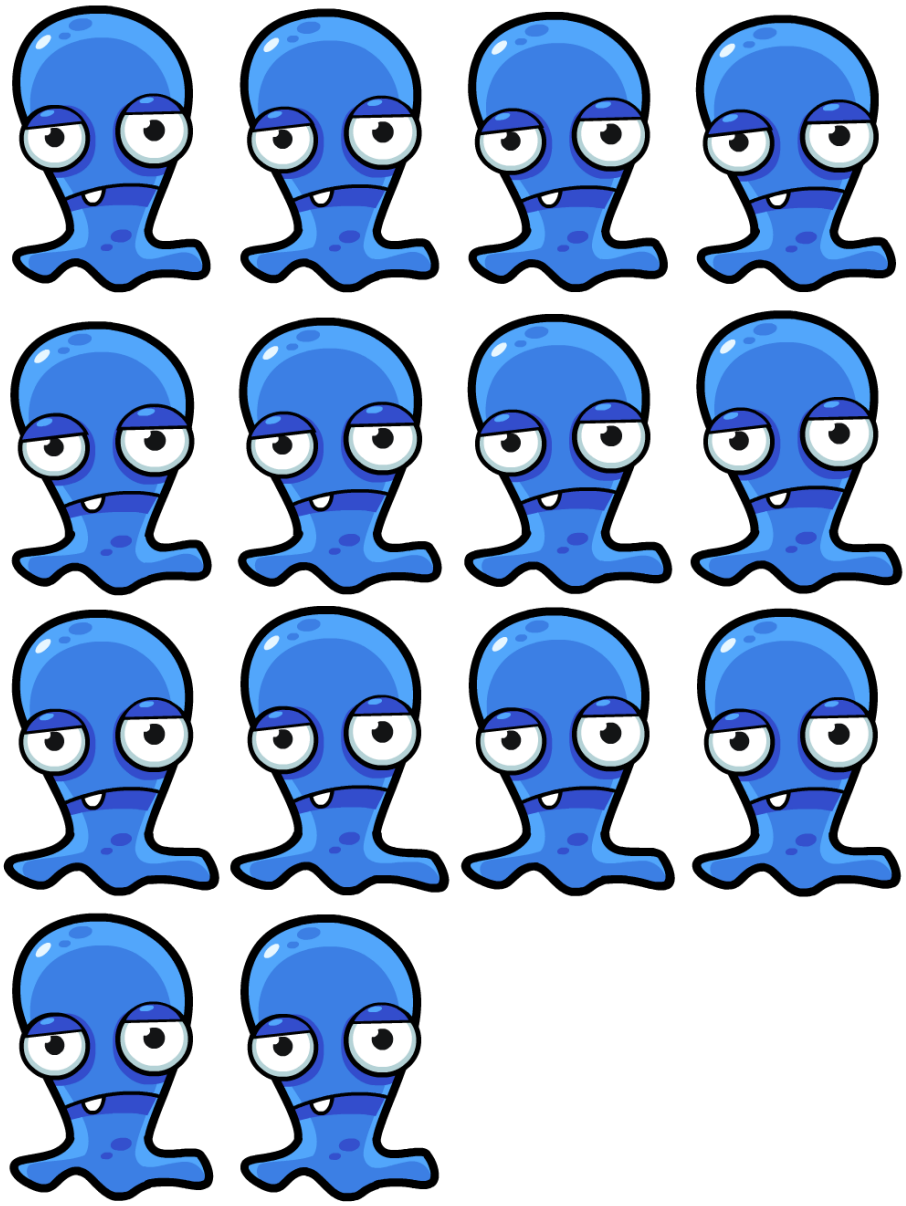




Sprite Portal

Sprite Torres

Sprite Proyectiles



Prefab Tiles

3.2 Prototipo

https://github.com/Brayron/Proyecto-Final-Videojuegos/tree/Prototipo

3.3 Perfiles de Usuarios

Este juego está diseñado para jugadores casuales, ya que no tiene puntuación ni forma de competir contra otros jugadores

3.4 Usabilidad

Este prototipo no ofrece dificultad alguna para ser usado por un público amplio de personas, ya que solo ofrece un método diferente de jugabilidad ya que solo es colocar torres de manera estratégica junto al dinero para comprarlas.

3.5 Test

En la versión de prueba se pudo notar que los aliens a la hora de cruzar por las torres, cuando eran atacados por las torres no reciben daño alguno. Aún no se ha realizado la pantalla del menú inicial y del game over. No hay música ni sonidos y el dinero no se gestiona de manera dinámica, sino que hay mucho dinero para hacer las pruebas. Las torres no suben de nivel y la animación de los ataques de las torres no se desaparecen al colisionar con los aliens.

3.6 Versiones de la Aplicación

Alfa

Nuestro juego está pensado en tener una versión Alfa que, en esencia, tendrá el propósito de mostrar el juego y descubrir los bugs críticos que podrían impedir el buen funcionamiento del juego.

En esta versión hemos reconocido errores después de varias pruebas de jugabilidad, entre ellos pudimos apreciar:

- Los enemigos no reciben los ataques, sin embargo, los ataques se mantienen persiguiéndolos
- Los enemigos no siguen la ruta establecida
- Las torres se pueden colocar en el camino de los enemigos
- No se reproduce la música del juego

Beta

Luego, una vez que el equipo considere que el juego está completo para un lanzamiento inicial, sacaremos una versión Beta, esta versión será capaz de funcionar correctamente y se utilizará para encontrar bugs menores y recibirá actualizaciones para expansión dependiendo de la retroalimentación del usuario

Final

Finalmente saldrá la versión completa, esta ya no recibirá actualizaciones de contenido, existiendo la posibilidad de añadirle expansiones a parte que se podrá adquirir con un pago adicional.