2K48++

Proyek Akhir OOP kelompok 21

Anggota Kelompok

Muhamad Rey Kafaka Fadlan
Raddief Ezra Satrio Andaru
Muhammad Rafli
Izzan Nawa Syarif

Overview

2K48++ adalah sebuah game puzzle minimalis yang sederhana namun sangat adiktif. Game ini menantang pemain untuk menggabungkan angka-angka pada papan permainan hingga mencapai tile 2048. Dengan mekanisme yang mudah dipahami tetapi sulit dikuasai.

Pemain Menggeser Tile yang terdiri angka 2, 4, 8, 16, 32, 64, 128, 256, 512,dan 1024 untuk mencapai angka 2048. Pemain akan diberikan area terbatas untuk menggeser Tile yang dimulai dari angka 2. Tile dengan angka yang sama jika bertemu dengan Tile yang sama juga akan berubah menjadi 1 Tile dengan angka yang lebih besar selanjutnya. Selama permainan setiap gerakan pemain akan memunculkan 2 buah Tile angka 2 di area yang kosong. Pemain harus mencapai angka 2048 sebelum area yang disediakan penuh karena munculnya Tile baru dari setiap pergerakan pemain.

Design Pillars

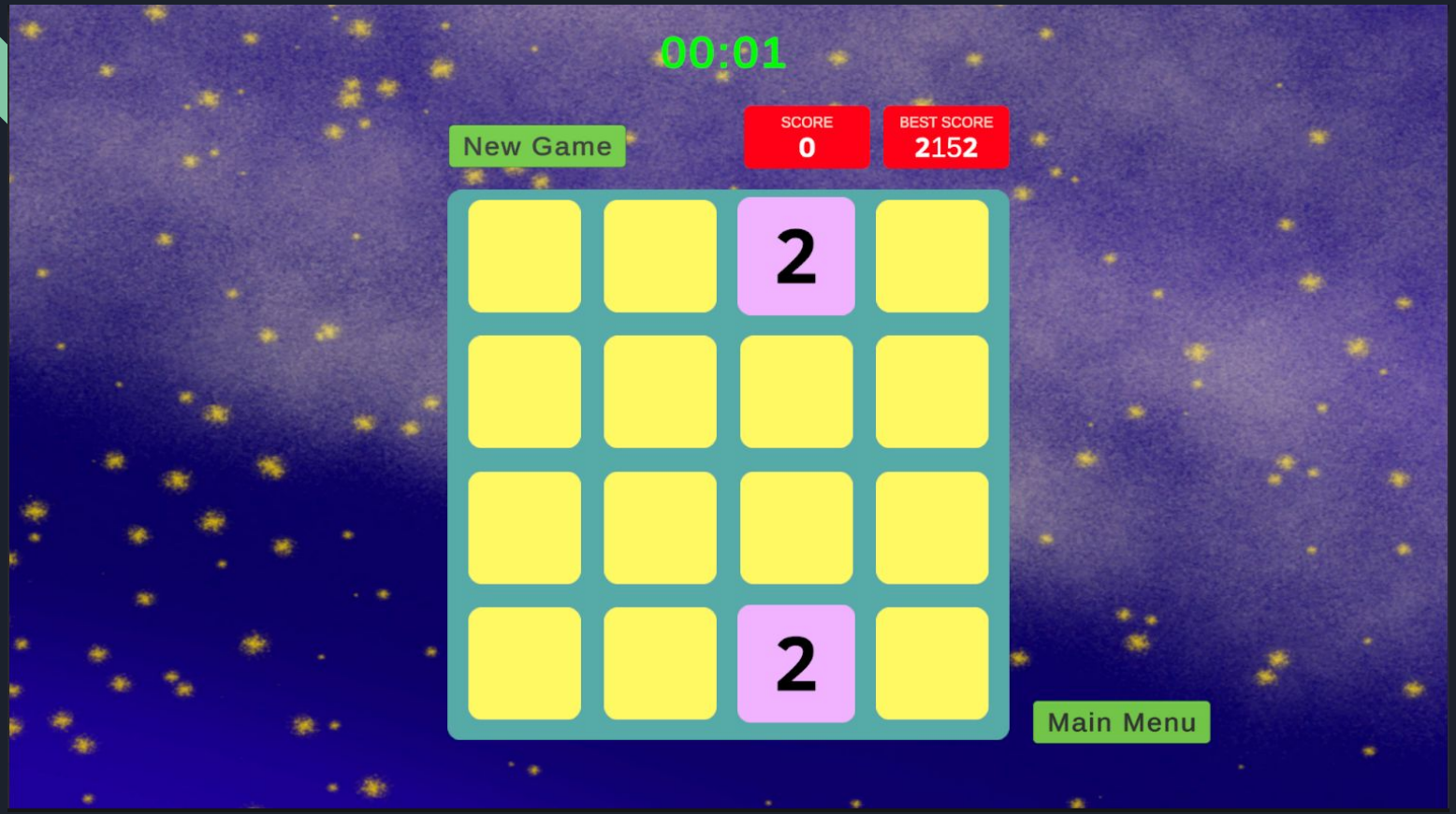
- Mudah dimainkan oleh siapa saja, bahkan bagi pemula.
- Mengharuskan pemain untuk berpikir dan merencanakan langkah-langkahnya dengan hati-hati.

Feature

- Kombinasi tile angka hingga mencapai angka yang tertinggi.
- Meraih skor tertinggi dan bersaing dengan pemain lain untuk memecahkan skor tertinggi
- Mentrack berapa lama untuk mencapai angka terakhir dengan timer
- Volume setting

Main interface

Gameplay Interface



Gameover Screen

Game Manager.cs

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

[DefaultExecutionOrder(-1)]
public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }

    public TileBoard board;
    public CanvasGroup gameOver;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI hiscoreText;


    private int score ;

    [SerializeField] private float fadeDuration = 0.5f; // Configurable
fade duration

    private void Awake()
    {
        if (Instance != null)
        {
            Destroy(gameObject); // Changed from DestroyImmediate
        }
        else
        {
            Instance = this;
        }
    }

    private void OnDestroy()
    {
        if (Instance == this)
        {
            Instance = null;
        }
    }

    private void Start()
    {
        NewGame();
```

```
    }

    public void NewGame()
    {
        // Reset score (if applicable)
        // Hide game over screen
        SetScore(0);
        hiscoreText.text = LoadHiscore().ToString();

        gameOver.alpha = 0f;
        gameOver.interactable = false;

        // Update board state
        board.ClearBoard();
        board.CreateTile();
        board.CreateTile();
        board.enabled = true;

        FindObjectOfType<Timer>().enabled = true;

        FindObjectOfType<Timer>().ResetTimer();
    }

    //Game Over when Board Full
    public void GameOver()
    {
        board.enabled = false;
        gameOver.interactable = true;

        FindObjectOfType<Timer>().enabled = false; //stop timer if
gameover

        StartCoroutine(Fade(gameOver, 1f, fadeDuration));
    }

    private IEnumerator Fade(CanvasGroup canvasGroup, float to, float
delay = 0f)
    {
        yield return new WaitForSeconds(delay);

        float elapsed = 0f;
        float from = canvasGroup.alpha;
```

```
        while (elapsed < fadeDuration) // Use configurable duration
        {
            canvasGroup.alpha = Mathf.Lerp(from, to, elapsed /
fadeDuration);
            elapsed += Time.deltaTime;
            yield return null;
        }

        canvasGroup.alpha = to;
    }

    public void IncreaseScore(int points)
    {
        SetScore(score + points);
    }

    private void SetScore(int score)
    {
        this.score = score;

        scoreText.text = score.ToString();

        SaveHiscore();
    }

    //Save Score if the new highest
    private void SaveHiscore()
    {
        int hiscore = LoadHiscore ();

        if(score > hiscore)
        {
            PlayerPrefs.SetInt("hiscore", score);
        }
    }

    //Load High Score
    private int LoadHiscore()
    {
        return PlayerPrefs.GetInt("hiscore", 0);
    }

}
```

Volumesettings.cs

```
using UnityEngine.Audio;
using UnityEngine;
using UnityEngine.UI;
using Unity.VisualScripting;

public class VolumeSettings : MonoBehaviour
{
    [SerializeField] private AudioMixer Mymixer;
    [SerializeField] private Slider MusicSlider;

    private void Start(){
        if (PlayerPrefs.HasKey("MasterVolume"))
        {
            LoadVolume();
        }
        else{
            SetVolume();
        }

    }

    //Set Volume with slider
    public void SetVolume()
    {
        float volume = MusicSlider.value;
        Mymixer.SetFloat("Master", Mathf.Log10(volume)*20);
        PlayerPrefs.SetFloat("MasterVolume", volume);
    }

    public void LoadVolume()
    {
        MusicSlider.value=PlayerPrefs.GetFloat("MasterVolume");

        SetVolume();
    }
}
```

Scane.cs

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

public class Scene : MonoBehaviour
{
    [SerializeField] private GameObject settingsMenu; // Reference to
the Settings Menu
    [SerializeField] private GameObject mainMenu;     // Reference to
the Main Menu (Optional)

    // Automatically hide the Settings Menu when the game starts
    public void MainMenuButton()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
- 1);
    }

    // Start the game button
    public void StartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
    }

    // Quit the application
    public void Quit()
    {
        Application.Quit();
    }
}
```

Timer.cs

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

public class Timer : MonoBehaviour
{

    [SerializeField] TextMeshProUGUI timerText;
    float elapsedTime;

    //Timer
    void Update()
    {
        elapsedTime += Time.deltaTime;
        int minutes = Mathf.FloorToInt(elapsedTime / 60);
        int seconds = Mathf.FloorToInt(elapsedTime % 60);
        timerText.text = string.Format("{0:00}:{1:00}", minutes,
seconds);
    }

    //Reset Timer For every new game
    public void ResetTimer()
    {
        elapsedTime = 0f;
        timerText.text = "00:00";
    }
}
```

Tile.cs

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

public class Tile : MonoBehaviour
{
    public TileState state { get; private set; }
    public TileCell cell { get; private set; }
    public bool locked { get; set; }

    private Image background;
    private TextMeshProUGUI text;

    private void Awake()
    {
        background = GetComponent<Image>();
        text = GetComponentInChildren<TextMeshProUGUI>();
    }

    //Set Number on tile
    public void SetState(TileState state)
    {
        this.state = state;

        background.color = state.backgroundColor;
        text.color = state.textColor;
        text.text = state.number.ToString();
    }

    //Random Tile Spawn On board
    public void Spawn(TileCell cell)
    {
        if (this.cell != null) {
            this.cell.tile = null;
        }

        this.cell = cell;
        this.cell.tile = this;

        transform.position = cell.transform.position;
    }

    //Moving behaviour
    public void MoveTo(TileCell cell)
    {
        if (this.cell != null) {
            this.cell.tile = null;
        }

        this.cell = cell;
        this.cell.tile = this;

        StartCoroutine(Animate(cell.transform.position, false));
    }

    //Merge Tile
    public void Merge(TileCell cell)
    {
        if (this.cell != null) {
            this.cell.tile = null;
        }

        this.cell = null;
        cell.tile.locked = true;

        StartCoroutine(Animate(cell.transform.position, true));
    }

    //Animation For smooth Moving
    private IEnumerator Animate(Vector3 to, bool merging)
    {
        float elapsed = 0f;
        float duration = 0.1f;

        Vector3 from = transform.position;

        while (elapsed < duration)
        {
            transform.position = Vector3.Lerp(from, to, elapsed /
duration);
            elapsed += Time.deltaTime;
            yield return null;
        }

        transform.position = to;

        //When
        if (merging) {
            Destroy(gameObject);
        }
    }

}
```

TileBoard.cs

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

public class TileBoard : MonoBehaviour
{
    [SerializeField] private Tile tilePrefab;
    [SerializeField] private TileState[] tileStates;

    private TileGrid grid;
    private List<Tile> tiles;
    private bool waiting;

    private void Awake()
    {
        grid = GetComponentInChildren<TileGrid>();
        tiles = new List<Tile>(16);
    }

    //Clear the board for every new game
    public void ClearBoard()
    {
        foreach (var cell in grid.cells) {
            cell.tile = null;
        }

        foreach (var tile in tiles) {
            Destroy(tile.gameObject);
        }

        tiles.Clear();
    }

    //Tile Creation
    public void CreateTile()
    {
        Tile tile = Instantiate(tilePrefab, grid.transform);
        tile.SetState(tileStates[0]);
        tile.Spawn(grid.GetRandomEmptyCell());
        tiles.Add(tile);
    }

    //Key For movement
```

```
    private void Update()
    {
        if (waiting) return; //Prevent Movement when animation not done

        if (Input.GetKeyDown(KeyCode.W) ||
Input.GetKeyDown(KeyCode.UpArrow)) {
            Move(Vector2Int.up, 0, 1, 1, 1);
        } else if (Input.GetKeyDown(KeyCode.A) ||
Input.GetKeyDown(KeyCode.LeftArrow)) {
            Move(Vector2Int.left, 1, 1, 0, 1);
        } else if (Input.GetKeyDown(KeyCode.S) ||
Input.GetKeyDown(KeyCode.DownArrow)) {
            Move(Vector2Int.down, 0, 1, grid.Height - 2, -1);
        } else if (Input.GetKeyDown(KeyCode.D) ||
Input.GetKeyDown(KeyCode.RightArrow)) {
            Move(Vector2Int.right, grid.Width - 2, -1, 0, 1);
        }
    }

    private void Move(Vector2Int direction, int startX, int incrementX,
int startY, int incrementY)
    {
        bool changed = false;

        for (int x = startX; x >= 0 && x < grid.Width; x += incrementX)
        {
            for (int y = startY; y >= 0 && y < grid.Height; y +=
incrementY)
            {
                TileCell cell = grid.GetCell(x, y);

                if (cell.Occupied) {
                    changed |= MoveTile(cell.tile, direction);
                }
            }
        }

        if (changed) {
            StartCoroutine(WaitForChanges());
        }
    }

    //Move Behaviour
```

```
    private bool MoveTile(Tile tile, Vector2Int direction)
    {
        TileCell newCell = null;
        TileCell adjacent = grid.GetAdjacentCell(tile.cell, direction);

        while (adjacent != null)
        {
            if (adjacent.Occupied)
            {
                if (CanMerge(tile, adjacent.tile))
                {
                    MergeTiles(tile, adjacent.tile);
                    return true;
                }

                break;
            }

            newCell = adjacent;
            adjacent = grid.GetAdjacentCell(adjacent, direction);
        }

        if (newCell != null)
        {
            tile.MoveTo(newCell);
            return true;
        }

        return false;
    }

    private bool CanMerge(Tile a, Tile b)
    {
        return a.state == b.state && !b.locked;
    }

    //Merge Tiles State
    private void MergeTiles(Tile a, Tile b)
    {
        tiles.Remove(a);
        a.Merge(b.cell);
```

```
        int index = Mathf.Clamp(IndexOf(b.state) + 1, 0,
tileStates.Length - 1);
        TileState newState = tileStates[index];

        b.SetState(newState);
        GameManager.Instance.IncreaseScore(newState.number);
    }

    private int IndexOf(TileState state)
    {
        for (int i = 0; i < tileStates.Length; i++)
        {
            if (state == tileStates[i]) {
                return i;
            }
        }

        return -1;
    }

    private IEnumerator WaitForChanges()
    {
        waiting = true;

        yield return new WaitForSeconds(0.1f);

        waiting = false;

        foreach (var tile in tiles) {
            tile.locked = false;
        }

        if (tiles.Count != grid.Size) {
            CreateTile();
        }

        if (CheckForGameOver()) {
            GameManager.Instance.GameOver();
        }
    }

    //Check when the Board is Full and not able to move
    public bool CheckForGameOver()
```

```
    {
        if (tiles.Count != grid.Size) {
            return false;
        }

        foreach (var tile in tiles)
        {
            TileCell up = grid.GetAdjacentCell(tile.cell,
Vector2Int.up);
            TileCell down = grid.GetAdjacentCell(tile.cell,
Vector2Int.down);
            TileCell left = grid.GetAdjacentCell(tile.cell,
Vector2Int.left);
            TileCell right = grid.GetAdjacentCell(tile.cell,
Vector2Int.right);

            if (up != null && CanMerge(tile, up.tile)) {
                return false;
            }

            if (down != null && CanMerge(tile, down.tile)) {
                return false;
            }

            if (left != null && CanMerge(tile, left.tile)) {
                return false;
            }

            if (right != null && CanMerge(tile, right.tile)) {
                return false;
            }
        }

        return true;
    }

}
```

TileGrid.cs

```
using System.Collections;
using UnityEngine;

public class TileGrid : MonoBehaviour
{
    public TileRow[] rows { get; private set; }
    public TileCell[] cells { get; private set; }

    public int Size => cells.Length;
    public int Height => rows.Length;
    public int Width => Size / Height;

    private void Awake()
    {
        rows = GetComponentsInChildren<TileRow>();
        cells = GetComponentsInChildren<TileCell>();

        for (int i = 0; i < cells.Length; i++) {
            cells[i].coordinates = new Vector2Int(i % Width, i /
Width);
        }
    }

    public void Start()
    {
        for (int y = 0; y < rows.Length; y++)
        {
            for (int x = 0; x < rows [y].cells.Length; x++)
            {
                rows[y].cells[x].coordinates = new Vector2Int(x, y);
            }
        }
    }

    public TileCell GetCell(int x, int y)
    {
        if (x >= 0 && x < Width && y >= 0 && y < Height)
        {
            return rows[y].cells[x];
        }
        else
        {
            return null;
```

```
        }
    }

    public TileCell GetCell(Vector2Int coordinates)
    {
        return GetCell(coordinates.x, coordinates.y);
    }

    public TileCell GetAdjacentCell(TileCell cell, Vector2Int
direction)
    {
        Vector2Int coordinates = cell.coordinates;
        coordinates.x += direction.x;
        coordinates.y -= direction.y;

        return GetCell(coordinates);
    }

    public TileCell GetRandomEmptyCell()
    {
        int index = Random.Range(0, cells.Length);
        int startingIndex = index;

    while (cells[index].Occupied) //error occopied //fixed to Occupied
    {
        index++;

        if (index >= cells.Length){
            index = 0;
        }
        if (index == startingIndex){
            return null;
        }
    }
        return cells[index];
    }

}
```

TileRow.cs

```
using UnityEngine;

public class TileRow : MonoBehaviour
{
    public TileCell[] cells { get; private set; }

    private void Awake()
    {
        cells = GetComponentsInChildren<TileCell>();
    }

}
```

TileCell.cs

```
using UnityEngine;

public class TileCell : MonoBehaviour
{
    public Vector2Int coordinates { get; set; }
    public Tile tile { get; set; }

    public bool Empty => tile == null;
    public bool Occupied => tile != null;
}
```

TileState.cs

```
using UnityEngine;

//Tile Number
[CreateAssetMenu(menuName = "Tile State")]
public class TileState : ScriptableObject
{
    public int number;
    public Color backgroundColor;
    public Color textColor;
}
```

THANKS YOU OOP