**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа «Программная инженерия»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СОГЛАСОВАНО  Внештатный преподаватель (по ГПХ)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.М.Сосновский  «13» \_\_\_мая\_\_\_\_\_\_\_\_\_\_ 2023 г. | |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_» \_\_\_\_мая\_\_\_\_\_\_\_\_\_ 2023 г. | |
| |  |  | | --- | --- | | *Подп. и дата* |  | | *Инв. № дубл.* |  | | *Взам. инв. №* |  | | *Подп. и дата* |  | | *Инв. № подл* |  | | **Валидный Сапер**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.05.03-01 12 01-1-ЛУ** | | | | | | |
|  | |  | | | | |
| Исполнитель  студент группы БПИ213  \_\_\_\_\_\_\_\_\_ / Фам Ч.Х. /  «13» \_\_\_\_\_мая\_\_\_ 2023 г. | | | | |
|  | | | | | | |
|  | | | | |  | |

**Москва 2023**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.05.03-01 12 01-1 |  | |  | |
| |  |  | | --- | --- | | *Подп. и дата* |  | | *Инв. № дубл.* |  | | *Взам. инв. №* |  | | *Подп. и дата* |  | | *Инв. № подл* |  | | **Валидный Сапер**  **Текст программы**  **RU.17701729.05.03-01 12 01-1**  **Листов 20** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2023**

**Содержание**

**1. ТЕКСТ ПРОГРАММЫ…………………………………………………3**

**1.1. BackGroundInstanceControllerComponent.cs…………………………3**

**1.2. Board.cs……………………………………………………………………5**

**1.3. BoardSetter.cs……………………………………………………………..5**

**1.4. Cell.cs………………………………………………………………………5**

**1.5. DataHolder.cs……………………………………………………………...6**

**1.6. DifficultSetter.cs…………………………………………………………...6**

**1.7. Game.cs…………………………………………………………………….6**

**1.8. MainMenu.cs……………………………………………………………..16**

**1.9. SoundController.cs………………………………………………………16**

**1.10. UIManager.cs…………………………………………………………….19**

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ…………………………...20**

# **ТЕКСТ ПРОГРАММЫ**

Программа написана на языке C# в среде разработки Microsoft Visual Studio Code. Программа состоит из 10 классов, учтены только классы, написанные вручную.

# **BackGroundInstanceControllerComponent.cs**

using UnityEngine;

public class BackgroundInstanceControllerComponent : MonoBehaviour

{

[Header("Tags")]

[Tooltip("Unique Object Tag")]

[SerializeField] private string createdTag;

private void Awake()

{

GameObject obj = GameObject.FindWithTag(this.createdTag);

if (obj != null)

{

Destroy(this.gameObject);

}

else

{

this.gameObject.tag = this.createdTag;

DontDestroyOnLoad(this.gameObject);

}

}

}

* 1. **Board.cs**

using UnityEngine;

using UnityEngine.Tilemaps;

public class Board : MonoBehaviour

{

public Tilemap tilemap {get; private set;}

public Tile tileUnknown;

public Tile tileEmpty;

public Tile tileMine;

public Tile tileExploded;

public Tile tileFlag;

public Tile tileNum1;

public Tile tileNum2;

public Tile tileNum3;

public Tile tileNum4;

public Tile tileNum5;

public Tile tileNum6;

public Tile tileNum7;

public Tile tileNum8;

private void Awake() {

tilemap = GetComponent<Tilemap>();

}

public void Draw(Cell[,] state) {

int width = state.GetLength(0);

int height = state.GetLength(1);

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

Cell cell = state[x, y];

tilemap.SetTile(cell.position, GetTile(cell));

}

}

}

private Tile GetTile(Cell cell) {

if (cell.revealed) {

return GetRevealedTile(cell);

} else if (cell.flagged) {

return tileFlag;

} else {

return tileUnknown;

}

}

private Tile GetRevealedTile(Cell cell) {

switch(cell.type) {

case Cell.Type.Empty: return tileEmpty;

case Cell.Type.Mine: return cell.exploded ? tileExploded : tileMine;

case Cell.Type.Number: return GetNumberTile(cell);

default: return null;

}

}

private Tile GetNumberTile(Cell cell) {

switch(cell.number) {

case 1: return tileNum1;

case 2: return tileNum2;

case 3: return tileNum3;

case 4: return tileNum4;

case 5: return tileNum5;

case 6: return tileNum6;

case 7: return tileNum7;

case 8: return tileNum8;

default: return null;

}

}

}

* 1. **BoardSeter.cs**

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class BoardSetter : MonoBehaviour

{

[SerializeField] private TMP\_Text width;

[SerializeField] private Slider widthSlider;

[SerializeField] private TMP\_Text height;

[SerializeField] private Slider heightSlider;

[SerializeField] private TMP\_Text mines;

[SerializeField] private Slider minesSlider;

void Update()

{

width.text = widthSlider.value.ToString();

height.text = heightSlider.value.ToString();

minesSlider.maxValue = Mathf.Floor(widthSlider.value \* heightSlider.value / 5.7f);

mines.text = Mathf.Floor(minesSlider.value).ToString();

}

public void StartCustomLevel()

{

DataHolder.width = (int)widthSlider.value;

DataHolder.height = (int)heightSlider.value;

DataHolder.mines = (int)minesSlider.value;

}

}

* 1. **Cell.cs**

using UnityEngine;

public struct Cell

{

public enum Type {

Invalid,

Empty,

Mine,

Number

}

public Type type;

public Vector3Int position;

public int number;

public bool revealed;

public bool flagged;

public bool exploded;

}

* 1. **DataHolder.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class DataHolder

{

public static int width;

public static int height;

public static int mines;

}

* 1. **DifficultSetter.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

public class DifficultSetter : MonoBehaviour

{

public int width;

public int height;

public int mines;

public void StartLevelButton()

{

DataHolder.width = width;

DataHolder.height = height;

DataHolder.mines = mines;

}

}

* 1. **Game.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class Game : MonoBehaviour

{

public int width;

public int height;

public int mineCount;

private Board board;

private Cell[,] state;

private bool gameOver;

private bool firstMove;

public TMP\_Text minesText;

public TMP\_Text timerText;

public float timeStart = 0;

private int flaggedCnt = 0;

private bool panelClicked = false;

private bool panelResume = true;

private bool resume = true;

private bool restartPanelOn = false;

[SerializeField] private TMP\_Text time;

private void Awake()

{

board = GetComponentInChildren<Board>();

width = DataHolder.width;

height = DataHolder.height;

mineCount = DataHolder.mines;

if (height <= width)

{

Camera.main.orthographicSize = (height + width) / 4 + System.Math.Max(height / width, width / height);

} else

{

Camera.main.orthographicSize = height / 2 + 2f;

}

}

private void Start()

{

NewGame();

}

private void SetTime()

{

time.text = (Mathf.Round(timeStart \* 100f) / 100f).ToString();

}

public void NewGame()

{

state = new Cell[width, height];

gameOver = false;

firstMove = false;

minesText.text = "MINES: " + mineCount;

timerText.text = "TIME: 0,00";

Time.timeScale = 1;

flaggedCnt = 0;

timeStart = 0;

GenerateCells();

Camera.main.transform.position = new Vector3(width / 2.5f, height / 2f, -10f);

board.Draw(state);

}

private void GenerateCells()

{

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

Cell cell = new Cell();

cell.position = new Vector3Int(x, y, 100);

cell.number = 0;

cell.type = Cell.Type.Empty;

state[x, y] = cell;

}

}

}

private void GenerateMines(int cellPositionX, int cellPositionY)

{

for (int i = 0; i < mineCount; ++i)

{

int x = Random.Range(1, width - 1);

int y = Random.Range(1, height - 1);

while (state[x, y].type == Cell.Type.Mine || (System.Math.Abs(x - cellPositionX) + System.Math.Abs(y - cellPositionY)) < 4)

{

++x;

if (x >= width - 1) {

x = 1;

++y;

if (y >= height - 1) {

y = 1;

}

}

}

state[x, y].type = Cell.Type.Mine;

}

}

private void GenerateNumbers()

{

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

if (state[x, y].type == Cell.Type.Empty) {

Cell cell = state[x, y];

cell.number = CountMines(x, y);

if (cell.number > 0)

{

cell.type = Cell.Type.Number;

}

state[x, y] = cell;

}

}

}

}

private int CountMines(int cellX, int cellY)

{

int count = 0;

for (int adjacentX = -1; adjacentX <= 1; ++adjacentX) {

for (int adjacentY = -1; adjacentY <= 1; ++adjacentY)

{

int x = cellX + adjacentX;

int y = cellY + adjacentY;

if (!isValid(x, y)) {

continue;

}

if (GetCell(x, y).type == Cell.Type.Mine)

{

++count;

}

}

}

return count;

}

private int CountFlags(int cellX, int cellY)

{

int count = 0;

for (int adjacentX = -1; adjacentX <= 1; ++adjacentX) {

for (int adjacentY = -1; adjacentY <= 1; ++adjacentY)

{

int x = cellX + adjacentX;

int y = cellY + adjacentY;

if (!isValid(x, y)) {

continue;

}

if (GetCell(x, y).flagged)

{

++count;

}

}

}

return count;

}

public void RestartButton()

{

panelClicked = true;

restartPanelOn = false;

panelResume = true;

}

public void ResumeButton()

{

panelResume = true;

}

private void Update()

{

if (panelClicked)

{

NewGame();

panelClicked = false;

}

else if (Input.GetKeyDown(KeyCode.Escape) && panelResume && !restartPanelOn)

{

panelResume = false;

resume = false;

restartPanelOn = false;

Camera.main.GetComponent<UIManager>().PauseOn();

}

else if (Input.GetKeyDown(KeyCode.Escape) && !panelResume)

{

panelResume = true;

resume = true;

Camera.main.GetComponent<UIManager>().PauseOff();

}

else if (panelResume && !resume)

{

resume = true;

}

else if (!gameOver && panelResume && resume)

{

if (firstMove)

{

timeStart += Time.deltaTime;

timerText.text = "TIME: " + System.Math.Round(timeStart, 2).ToString();

}

if (Input.GetMouseButtonUp(1))

{

Flag();

}

else if (Input.GetMouseButtonUp(0))

{

Reveal();

}

}

}

private void Reveal()

{

Vector3 worldPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);

Vector3Int cellPosition = board.tilemap.WorldToCell(worldPosition);

if (!firstMove && isValid(cellPosition.x, cellPosition.y))

{

GenerateMines(cellPosition.x, cellPosition.y);

GenerateNumbers();

OpenBoundCells();

board.Draw(state);

firstMove = !firstMove;

}

Cell cell = GetCell(cellPosition.x, cellPosition.y);

if (cell.type == Cell.Type.Invalid || cell.flagged)

{

return;

}

switch (cell.type) {

case Cell.Type.Mine:

Explode(cell);

break;

case Cell.Type.Empty:

Flood(cell);

break;

case Cell.Type.Number:

if (cell.revealed)

{

bool isOpened = OpenNumberTiles(cell);

if (!isOpened)

{

cell.revealed = true;

state[cellPosition.x, cellPosition.y] = cell;

}

}

else

{

cell.revealed = true;

state[cellPosition.x, cellPosition.y] = cell;

}

break;

default:

cell.revealed = true;

state[cellPosition.x, cellPosition.y] = cell;

break;

}

ChechWinCondition();

board.Draw(state);

}

private void OpenBoundCells()

{

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

if (x == 0 || x == width - 1 || y == height - 1 || y == 0) {

if (state[x, y].type == Cell.Type.Empty)

{

Flood(state[x, y]);

}

else

{

state[x, y].revealed = true;

}

}

}

}

}

private bool OpenNumberTiles(Cell cell)

{

if (CountFlags(cell.position.x, cell.position.y) == cell.number) {

for (int adjacentX = -1; adjacentX <= 1; ++adjacentX)

{

for (int adjacentY = -1; adjacentY <= 1; ++adjacentY)

{

int x = cell.position.x + adjacentX;

int y = cell.position.y + adjacentY;

if (!isValid(x, y)) {

continue;

}

if (!GetCell(x, y).flagged)

{

Cell opening = GetCell(x, y);

if (opening.type == Cell.Type.Mine)

{

Explode(opening);

}

else if (opening.type == Cell.Type.Empty) {

Flood(opening);

}

else

{

opening.revealed = true;

state[x, y] = opening;

}

}

}

}

return true;

}

return false;

}

private void ChechWinCondition()

{

if (gameOver) {

return;

}

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

Cell cell = state[x, y];

if (cell.type != Cell.Type.Mine && !cell.revealed)

{

return;

}

}

}

Camera.main.GetComponent<UIManager>().DelayWon();

SetTime();

minesText.text = "MINES: " + 0;

gameOver = true;

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

Cell cell = state[x, y];

if (cell.type == Cell.Type.Mine)

{

cell.flagged = true;

state[x, y] = cell;

}

}

}

}

private void Explode(Cell cell)

{

gameOver = true;

cell.revealed = true;

cell.exploded = true;

state[cell.position.x, cell.position.y] = cell;

board.Draw(state);

for (int x = 0; x < width; ++x)

{

for (int y = 0; y < height; ++y)

{

cell = state[x, y];

if (cell.type == Cell.Type.Mine)

{

cell.revealed = true;

state[x, y] = cell;

}

}

}

restartPanelOn = true;

Camera.main.GetComponent<UIManager>().DelayLose();

}

private void Flood(Cell cell)

{

if (cell.revealed || cell.type == Cell.Type.Mine || cell.type == Cell.Type.Invalid)

{

return;

}

cell.revealed = true;

state[cell.position.x, cell.position.y] = cell;

if (cell.type == Cell.Type.Empty)

{

Flood(GetCell(cell.position.x + 1, cell.position.y));

Flood(GetCell(cell.position.x - 1, cell.position.y));

Flood(GetCell(cell.position.x, cell.position.y + 1));

Flood(GetCell(cell.position.x, cell.position.y - 1));

Flood(GetCell(cell.position.x + 1, cell.position.y + 1));

Flood(GetCell(cell.position.x - 1, cell.position.y - 1));

Flood(GetCell(cell.position.x - 1, cell.position.y + 1));

Flood(GetCell(cell.position.x + 1, cell.position.y - 1));

}

}

private void Flag()

{

Vector3 worldPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);

Vector3Int cellPosition = board.tilemap.WorldToCell(worldPosition);

Cell cell = GetCell(cellPosition.x, cellPosition.y);

if (cell.type == Cell.Type.Invalid || cell.revealed)

{

return;

}

flaggedCnt += (cell.flagged) ? -1 : 1;

minesText.text = "MINES: " + (mineCount - flaggedCnt);

cell.flagged = !cell.flagged;

state[cellPosition.x, cellPosition.y] = cell;

board.Draw(state);

}

private Cell GetCell(int x, int y)

{

if (isValid(x, y))

{

return state[x, y];

}

else

{

return new Cell();

}

}

private bool isValid(int x, int y)

{

return !(x < 0 || y < 0 || x >= width || y >= height);

}

}

* 1. **MainMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

public void PlayGame()

{

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

}

public void ToMenu()

{

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

}

public void QuitButton()

{

Application.Quit();

}

}

* 1. **SoundConroller.cs**

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class SoundController : MonoBehaviour

{

[Header("Components")]

[Tooltip("Audio Source Does Тot Connect Automatically")]

[SerializeField] private AudioSource \_audio;

[Tooltip("Slider Search Using A Tag")]

[SerializeField] private Slider slider;

[Tooltip("Text Search Using A Tag")]

[SerializeField] private TMP\_Text text;

[Header("Keys")]

[Tooltip("Save Data PlayerPrefs Key")]

[SerializeField] private string saveVolumeKey;

[Header("Tags")]

[Tooltip("Volume Control Slider Tag")]

[SerializeField] private string sliderTag;

[Tooltip("Volume Control Text Tag")]

[SerializeField] private string textVolumeTag;

[Header("Parameters")]

[Tooltip("Sound Volume Value")]

[SerializeField] [Range(0.0f, 1.0f)] private float volume;

public void Awake()

{

// Checks whether a save is available in the registry.

if (PlayerPrefs.HasKey(this.saveVolumeKey))

{

this.volume = PlayerPrefs.GetFloat(this.saveVolumeKey);

this.\_audio.volume = this.volume;

// Search for and connect the slider.

GameObject sliderObj = GameObject.FindWithTag(this.sliderTag);

if (sliderObj != null)

{

this.slider = sliderObj.GetComponent<Slider>();

this.slider.value = this.volume;

}

}

else

{

// Setting the default volume.

this.volume = 0.5f;

PlayerPrefs.SetFloat(this.saveVolumeKey, this.volume);

this.\_audio.volume = this.volume;

}

}

private void LateUpdate()

{

// Search for and connect the slider.

GameObject sliderObj = GameObject.FindWithTag(this.sliderTag);

if (sliderObj != null)

{

this.slider = sliderObj.GetComponent<Slider>();

this.volume = slider.value;

// Check the volume, is stored in the registry if different.

if (this.\_audio.volume != this.volume)

{

PlayerPrefs.SetFloat(this.saveVolumeKey, this.volume);

}

// Search and connect text for volume output.

GameObject textObj = GameObject.FindWithTag(this.textVolumeTag);

if (textObj != null)

{

this.text = textObj.GetComponent<TMP\_Text>();

this.text.text = Mathf.Round(this.volume \* 100) + "%"; // Converts the volume value to a percentage. (from 0% to 100%)

}

}

this.\_audio.volume = this.volume;

}

}

* 1. **UIManager.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class UIManager : MonoBehaviour

{

[SerializeField] private GameObject panelPause;

[SerializeField] private GameObject panelWon;

[SerializeField] private GameObject panelLose;

public void PauseOn()

{

panelPause.SetActive(true);

Time.timeScale = 0;

}

public void PauseOff()

{

panelPause.SetActive(false);

Time.timeScale = 1;

}

public void Won()

{

panelWon.SetActive(true);

Time.timeScale = 0;

}

public void DelayWon()

{

Invoke("Won", 2f);

}

public void Lose()

{

panelLose.SetActive(true);

Time.timeScale = 0;

}

public void DelayLose()

{

Invoke("Lose", 2f);

}

}

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. Технические рекомендации движка Unity [Электронный ресурс]. Автор: Unity Technologies, Дата последнего изменения 29.10.2021 URL: https://docs.unity3d.com/2021.1/Documentation/Manual/system-requirements.html#player (Дата обращения 10.05.2023)

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |