**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Научный руководитель  Приглашенный преподаватель департамента программной инженерии | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Поповкин  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |

**МНОГОПОЛЬЗОВАТЕЛЬСКАЯ ИГРА «КОЛОНИЗАТОРЫ» ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 12 01-1-ЛУ**

**Исполнитель**

Студент группы БПИ173

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / И.Н. Руденко /

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Москва 2018**

**УТВЕРЖДЕН**

**RU.17701729.04.01-01 12 01-1-ЛУ**

**МНОГОПОЛЬЗОВАТЕЛЬСКАЯ ИГРА «КОЛОНИЗАТОРЫ» ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ**

**Текст программы**

**RU.17701729.04.01-01 12 01-1**

**Листов 31**

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**Москва 2018**

**СОДЕРЖАНИЕ**

[1. ТЕКСТ ПРОГРАММЫ 4](#_Toc514322289)

**1. ТЕКСТ ПРОГРАММЫ**

**MainMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MainMenu : MonoBehaviour {

/// <summary>

/// Закрывает игру

/// </summary>

public void ExitGameBtn\_OnClick()

{

Application.Quit();

}

}

**CreatingGameMenu.cs**

using UnityEngine;

using UnityEngine.UI;

public class CreatingGameMenu : MonoBehaviour {

public GameObject \_3Players;

public GameObject \_4Players;

public GameObject player1Input;

public GameObject player2Input;

public GameObject player3Input;

public GameObject player4Input;

// имена игроков

public static string[] playerNames = new string[4];

/// <summary>

/// Добавляет четвертого игрока

/// </summary>

public void AddPlayer()

{

\_3Players.SetActive(false);

\_4Players.SetActive(true);

}

/// <summary>

/// Убирает четвертого игрока

/// </summary>

public void RemovePlayer()

{

\_3Players.SetActive(true);

\_4Players.SetActive(false);

}

/// <summary>

/// Сохраняет введенные имена игроков

/// </summary>

public void SavePlayerNames()

{

playerNames[0] = player1Input.GetComponent<InputField>().text == "" ? "Player 1" : player1Input.GetComponent<InputField>().text;

playerNames[1] = player2Input.GetComponent<InputField>().text == "" ? "Player 2" : player2Input.GetComponent<InputField>().text;

playerNames[2] = player3Input.GetComponent<InputField>().text == "" ? "Player 3" : player3Input.GetComponent<InputField>().text;

if (\_4Players.activeSelf)

{

playerNames[3] = player4Input.GetComponent<InputField>().text == "" ? "Player 4" : player4Input.GetComponent<InputField>().text;

}

}

}

**GameScript.cs**

using UnityEngine;

using UnityEngine.UI;

using GameClassLib;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

public class GameScript : MonoBehaviour

{

// Текстовые поля, отображающие введенные имена игроков

public Text[] playerNames;

public Text[] playerNames\_4Players;

// Текстовые поля, отображающие количество ресурсов у игрока

public Text[] resLabels;

// Объекты игроков

static Player[] players;

public static Player[] Players { get { return players; } }

static int currentPlayerNum;

public static int CurrentPlayerNum { get { return currentPlayerNum; } }

// Объекты поселений и дорог

static List<Road> roads = new List<Road>();

public static List<Road> Roads { get { return roads; } }

static List<Town> towns = new List<Town>();

public static List<Town> Towns { get { return towns; } }

// Цвета для игроков

Color[] playerColors = { Color.red, new Color(0.1f, 0.7f, 1f), new Color(1f, 0.7f, 0.01f), Color.white };

// Спрайты местностей

public Sprite mountains;

public Sprite lake;

public Sprite forest;

public Sprite grassland;

public Sprite fields;

// Массив гексов

public SpriteRenderer[] hexObjects;

Hex[] hexes = new Hex[18];

// Массивы игровых объектов мест для строительства поселений и дорог

public GameObject[] townObjArray;

public GameObject[] roadObjArray;

// Экран уведомлений

public GameObject infoScreen;

public Text infoScreenText;

public Text infoScreenBtnText;

public Button infoScreenButton;

// Экран смены хода в основной фазе

public GameObject changeTurnScreen;

public Text changeTurnPlayerTxt;

public Text changeTurnCubesTxt;

// Экран строительства

public GameObject buildingScreen;

public Button[] buildingScreenBtns;

// Экран подсчета победных очков

public GameObject finalScreen;

public GameObject finalScreen\_player4Obj;

public Button quitBtn;

public Button continueGameBtn;

public Text[] endGamePlayerNames;

public Text[] endGamePlayerScores;

// Интерфейсы для разных стадий хода

public GameObject standartInterface;

public GameObject standartInterface\_3Players;

public GameObject standartInterface\_4Players;

public GameObject buildingInterface;

public Button buildingInterfaceCancelBtn;

// Кнопки игрового меню

public Button endGameBtn;

public Button tradeBtn;

public Button buildBtn;

public Button endTurnBtn;

// Текущая фаза игры

static int phase = 1;

// Количество игроков в данный момент

public static int playersCount;

// Генератор случайных значений

System.Random rand = new System.Random();

// Последнее выпавшее число на кубиках

static int cubesValue;

/// <summary>

/// Обрабатывает постройку нового поселения в фазе основания

/// </summary>

/// <param name="townID"></param>

public void TownBuildedHandler\_SettlePhase(string townID, bool townBuilded)

{

GameMethods.DeactivateTownObjects(towns);

// Настраивает интерфейс

buildingInterface.GetComponent<Text>().text = "Проложите дорогу от поселения";

// Находит объект построенного поселения

Town buildedVillage;

foreach(Town town in towns)

{

if (town.Name == townID)

{

buildedVillage = town;

// Делает недоступными для строительства соседние перекрестки

GameMethods.DeleteBorderingTownObjects(buildedVillage, towns);

// Активирует соседние объекты дорог

foreach (Road road in buildedVillage.Roads)

{

road.EnableFor.Add(players[currentPlayerNum]);

road.SetActive(true);

}

// Подписывает игрока на события собра ресурсов с гексов

GameMethods.AddHexListeners(townID, hexes, players[currentPlayerNum]);

return;

}

}

}

/// <summary>

/// Обрабатывает постройку дороги в фазе основания

/// </summary>

/// <param name="roadID"></param>

public void RoadBuildedHandler\_SettlePhase(string roadID)

{

GameMethods.DeactivateRoadObjects(roads);

// Настраивает игровой интерфейс

buildingInterface.GetComponent<Text>().text = "Выберите место для поселения";

buildingInterface.SetActive(false);

standartInterface.SetActive(true);

// Сменяет игрока

if (phase == 1)

{

if (currentPlayerNum == playersCount - 1)

phase++;

else

{

++currentPlayerNum;

infoScreenText.text = "Ход игрока " + players[currentPlayerNum].Name;

}

}

else if (phase == 2)

{

if (currentPlayerNum == 0)

{

phase++;

// Настраивает экран уведомления о смене хода

infoScreenText.text = "Этап игры";

// Назначает новый обработчик для кнопки продолжения игры

infoScreenButton.onClick.RemoveAllListeners();

infoScreenButton.onClick.AddListener(StartMainPhase);

}

else

{

--currentPlayerNum;

infoScreenText.text = "Ход игрока " + players[currentPlayerNum].Name;

}

}

}

/// <summary>

/// Обрабатывает постройку нового поселения в основной фазе

/// </summary>

/// <param name="townID"></param>

/// <param name="townBuilded"></param>

public void TownBuildedHandler\_MainPhase(string townID, bool townBuilded)

{

Player currentPlayer = players[currentPlayerNum];

currentPlayer.Score++;

GameMethods.DeactivateTownObjects(towns);

// Отнимает у игрока ресурсы, необходимые для постройки объекта

if (townBuilded)

currentPlayer.Build("town");

else

currentPlayer.Build("village");

GameMethods.LoadResValues(resLabels, currentPlayerNum);

// Выводит стандартный интерфейс

buildingInterface.SetActive(false);

standartInterface.SetActive(true);

// Находит объект построенного поселения

Town buildedTown;

foreach (Town town in towns)

{

if (town.Name == townID)

{

buildedTown = town;

// Делает недоступными для строительства соседние перекрестки

if (buildedTown.Builded == "village")

GameMethods.DeleteBorderingTownObjects(buildedTown, towns);

// Подписывает игрока на события собра ресурсов с гексов

GameMethods.AddHexListeners(townID, hexes, players[currentPlayerNum]);

return;

}

}

}

/// <summary>

/// Обрабатывает постройку новой дороги в основной фазе

/// </summary>

/// <param name="roadID"></param>

public void RoadBuildedHandler\_MainPhase(string roadID)

{

GameMethods.DeactivateRoadObjects(roads);

// Отнимает у игрока ресурсы, необходимые для постройки объекта

players[currentPlayerNum].Build("road");

GameMethods.LoadResValues(resLabels, currentPlayerNum);

// Выводит стандартный интерфейс

buildingInterface.SetActive(false);

standartInterface.SetActive(true);

}

/// <summary>

/// Загружает сцену

/// </summary>

public void Start()

{

// Создает новый остров

GameMethods.CreateIsland(hexObjects, ref hexes, mountains, lake, forest, grassland, fields);

// Инициализирует объекты игроков

playersCount = CreatingGameMenu.playerNames[3] == null ? 3 : 4;

players = new Player[playersCount];

currentPlayerNum = 0;

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

{

players[i] = new Player(CreatingGameMenu.playerNames[i] ?? "Player " + (i + 1), playerColors[i]);

}

// Инициализирует объекты дорог и поселений

for (int i = 0; i < 72; i++)

{

string roadName = roadObjArray[i].name;

Road newRoad = new Road(roadName);

roads.Add(newRoad);

RoadObjectScript.RoadBuilded += roads[i].RoadBuildedHandler;

}

for (int i = 0; i < 54; i++)

{

towns.Add(new Town(townObjArray[i].name, roads));

TownObjectScript.TownBuilded += towns[i].TownBuildedHandler;

}

// Определяет корректный вариант игрового интерфейса

if (playersCount == 4)

{

playerNames = playerNames\_4Players;

standartInterface\_3Players.SetActive(false);

standartInterface\_4Players.SetActive(true);

}

// Загружает интерфейс

GameMethods.LoadPlayerInterface(playerNames, resLabels);

// Добавляет методы-обработчики события появления нового объекта

RoadObjectScript.RoadBuilded += RoadBuildedHandler\_SettlePhase;

TownObjectScript.TownBuilded += TownBuildedHandler\_SettlePhase;

// Добавляет обработчики нажатий для кнопок интерфейса

infoScreenButton.onClick.AddListener(InfoScreenBtn\_OnClick\_SettlePhase);

endTurnBtn.onClick.AddListener(EndTurnBtn\_OnClick\_SettlePhase);

}

/// <summary>

/// Начинает фазу основания

/// </summary>

public void InfoScreenBtn\_OnClick\_SettlePhase()

{

// Настраивает интерфейс

infoScreen.SetActive(false);

infoScreenBtnText.text = "Продолжить";

if (phase != 3)

{

standartInterface.SetActive(false);

buildingInterface.SetActive(true);

}

// Активирует режим строительства поселения

foreach(Town town in towns)

{

if (town.Builded == "none")

{

town.SetActive(true);

}

}

}

/// <summary>

/// Начинает основную фазу игры

/// </summary>

public void StartMainPhase()

{

// Назначает новый обработчик кнопки окончания хода

endTurnBtn.onClick.RemoveAllListeners();

endTurnBtn.onClick.AddListener(EndTurnBtn\_OnClick\_MainPhase);

// Назначает новые методы-обработчики события появления нового объекта

RoadObjectScript.RoadBuilded -= RoadBuildedHandler\_SettlePhase;

RoadObjectScript.RoadBuilded += RoadBuildedHandler\_MainPhase;

TownObjectScript.TownBuilded -= TownBuildedHandler\_SettlePhase;

TownObjectScript.TownBuilded += TownBuildedHandler\_MainPhase;

// Добавляет каждому игроку по два ПО за основанные поселения

foreach (Player player in players)

player.Score = 2;

// Загружает интерфейс игрока

GameMethods.GetResources(hexes, -1);

GameMethods.LoadResValues(resLabels, currentPlayerNum);

// Делает доступными кнопки игрового меню

tradeBtn.interactable = true;

buildBtn.interactable = true;

// Добавляет кнопку "Отмена" в интерфейсе режима строительства

buildingInterfaceCancelBtn.interactable = true;

// Скрывает экран уведомлений

infoScreen.SetActive(false);

}

/// <summary>

/// Обрабатывает нажатие кнопки "продолжить" во время основной фазы

/// </summary>

public void ChangeTurnScreenBtn\_OnClick()

{

GameMethods.GetResources(hexes, cubesValue);

GameMethods.LoadResValues(resLabels, currentPlayerNum);

changeTurnScreen.SetActive(false);

}

/// <summary>

/// Заканчивает ход во время фазы основания

/// </summary>

public void EndTurnBtn\_OnClick\_SettlePhase()

{

infoScreen.SetActive(true);

GameMethods.LoadPlayerInterface(playerNames, resLabels);

}

/// <summary>

/// Заканчивает ход во время основной фазы

/// </summary>

public void EndTurnBtn\_OnClick\_MainPhase()

{

currentPlayerNum = (currentPlayerNum + 1) % playersCount;

// Гененрирует новый случайный результат броска кубиков

int cubeValue1 = rand.Next(1, 7);

int cubeValue2 = rand.Next(1, 7);

cubesValue = cubeValue1 + cubeValue2;

changeTurnPlayerTxt.text = "Ход игрока " + players[currentPlayerNum].Name;

changeTurnCubesTxt.text = "Число на кубиках: " + cubesValue;

changeTurnScreen.SetActive(true);

GameMethods.LoadPlayerInterface(playerNames, resLabels);

}

/// <summary>

/// Загружает экран строительства

/// </summary>

public void BuildBtn\_OnClick()

{

Player currentPlayer = players[currentPlayerNum];

// Делает неактивными кнопки меню строительства

foreach (Button button in buildingScreenBtns)

{

button.GetComponent<Image>().color = Color.white;

button.interactable = false;

}

if (currentPlayer.Wood >= 1 && currentPlayer.Bricks >= 1)

{

// Активирует кнопку строительства дороги

buildingScreenBtns[0].interactable = true;

buildingScreenBtns[0].GetComponent<Image>().color = currentPlayer.Color;

if (currentPlayer.Wheat >= 1 && currentPlayer.Wool >= 1)

{

// Активирует кнопку строительства поселения

buildingScreenBtns[1].interactable = true;

buildingScreenBtns[1].GetComponent<Image>().color = currentPlayer.Color;

}

}

if (currentPlayer.Wheat >= 2 && currentPlayer.Stone >= 3)

{

// Активирует кнопку строительства города

buildingScreenBtns[2].interactable = true;

buildingScreenBtns[2].GetComponent<Image>().color = currentPlayer.Color;

}

buildingScreen.SetActive(true);

}

/// <summary>

/// Обрабатывает нажатие кнопки строительства дороги

/// </summary>

public void BuildRoadBtn\_OnClick()

{

buildingInterface.GetComponent<Text>().text = "Выберите место для дороги";

standartInterface.SetActive(false);

buildingInterface.SetActive(true);

buildingScreen.SetActive(false);

GameMethods.ActivateRoadBuildingMode(roads);

}

/// <summary>

/// Обрабатывает нажатие кнопки строительства поселения

/// </summary>

public void BuildVillageBtn\_OnClick()

{

buildingInterface.GetComponent<Text>().text = "Выберите место для поселения";

standartInterface.SetActive(false);

buildingInterface.SetActive(true);

buildingScreen.SetActive(false);

GameMethods.ActivateVillageBuildingMode(towns);

}

/// <summary>

/// Обрабатывает нажатие кнопки строительства города

/// </summary>

public void BuildTownBtn\_OnClick()

{

buildingInterface.GetComponent<Text>().text = "Выберите место для города";

standartInterface.SetActive(false);

buildingInterface.SetActive(true);

buildingScreen.SetActive(false);

GameMethods.ActivateTownBuildingMode(towns);

}

/// <summary>

/// Обрабатывает нажатие кнопки "Отмена" в интерфейсе строительства

/// </summary>

public void BuildingInterfaceCancelBtn\_OnClick()

{

GameMethods.DeactivateRoadObjects(roads);

GameMethods.DeactivateTownObjects(towns);

buildingInterface.SetActive(false);

standartInterface.SetActive(true);

}

/// <summary>

/// Обрабатывает нажатие кнопки "Отмена" на экране выбора объекта для строительства

/// </summary>

public void BuildingScreenCancelBtn\_OnClick()

{

// Делает кнопки меню выбора объекта для строительства недоступными

foreach(Button button in buildingScreenBtns)

{

button.interactable = false;

}

buildingScreen.SetActive(false);

}

/// <summary>

/// Обрабатывает нажатие кнопки "Закончить игру"

/// Выводит экран подсчета победных очков

/// </summary>

public void EndGameBtn\_OnClick()

{

if (playersCount == 4)

finalScreen\_player4Obj.SetActive(true);

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

{

endGamePlayerNames[i].text = players[i].Name;

endGamePlayerScores[i].text = players[i].Score.ToString();

}

finalScreen.SetActive(true);

}

/// <summary>

/// Обрабатывает нажатие кнопки "продолжить" на экране подсчета очков

/// Скрывает экран уведомлений

/// </summary>

public void FinalScreen\_ContinueBtn\_OnClick()

{

finalScreen.SetActive(false);

}

/// <summary>

/// Обрабатывает нажатие кнопки "Выйти из игры"

/// </summary>

public void QuitBtn\_OnClick()

{

Application.Quit();

}

}

**LoadScene.cs**

using UnityEngine;

using UnityEngine.SceneManagement;

public class LoadScene : MonoBehaviour {

/// <summary>

/// Загружает игровую сцену

/// </summary>

/// <param name="sceneId"></param>

public void Next(int sceneId)

{

SceneManager.LoadScene(sceneId);

}

}

**GameMethods.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using GameClassLib;

public class GameMethods : MonoBehaviour {

/// <summary>

/// Создает случайный остров

/// </summary>

public static void CreateIsland(SpriteRenderer[] hexObjects, ref Hex[] hexes,

Sprite mountains, Sprite lake, Sprite forest, Sprite grassland, Sprite fields)

{

// Значения, при которых гексы приносят урожай

int[] gettingResValues = { 6, 3, 11, 9, 4, 5, 10, 9, 12, 11, 4, 8, 10, 5, 2, 6, 3, 8 };

// Список возможных местностей

List<string> landscapes = new List<string> {

"Mountains", "Mountains", "Mountains",

"Lake", "Lake", "Lake",

"Forest", "Forest", "Forest", "Forest",

"Grassland", "Grassland", "Grassland", "Grassland",

"Fields", "Fields", "Fields", "Fields"};

string[] island = new string[18];

System.Random rand = new System.Random();

// Заполняет список island местностями в случайном порядке

for (int i = 0; i < 18; i++)

{

int landScapeID = rand.Next(18 - i);

island[i] = landscapes[landScapeID];

landscapes.RemoveAt(landScapeID);

}

// Присваивает каждому элементу из списка гексов соответствующий спрайт

// Инициализирует объекты гексов

for (int i = 0; i < 18; i++)

{

int gettingResValue = gettingResValues[i];

switch (island[i])

{

case "Mountains":

hexObjects[i].sprite = mountains;

hexes[i] = new Hex("stone", gettingResValue);

break;

case "Lake":

hexObjects[i].sprite = lake;

hexes[i] = new Hex("bricks", gettingResValue);

break;

case "Forest":

hexObjects[i].sprite = forest;

hexes[i] = new Hex("wood", gettingResValue);

break;

case "Grassland":

hexObjects[i].sprite = grassland;

hexes[i] = new Hex("wool", gettingResValue);

break;

case "Fields":

hexObjects[i].sprite = fields;

hexes[i] = new Hex("wheat", gettingResValue);

break;

}

}

}

/// <summary>

/// Активирует объекты дорог, доступные для стоительства текущим игроком

/// </summary>

/// <param name="roads"></param>

/// <param name="activate"></param>

public static void ActivateRoadBuildingMode(List<Road> roads)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

foreach (Road road in roads)

{

if (!road.Builded && road.EnableFor.Contains(currentPlayer))

road.SetActive(true);

}

}

/// <summary>

/// Активирует объекты перекрестков, доступные для стоительства поселения текущим игроком

/// </summary>

/// <param name="towns"></param>

/// <param name="activate"></param>

public static void ActivateVillageBuildingMode(List<Town> towns)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

foreach (Town town in towns)

{

if (town.Builded == "none" && town.EnableFor.Contains(currentPlayer))

{

town.SetActive(true);

Debug.Log("Town " + town.Name + " activated");

}

}

}

/// <summary>

/// Активирует объекты перекрестков, доступные для строительства города текущим игроком

/// </summary>

/// <param name="towns"></param>

/// <param name="activate"></param>

public static void ActivateTownBuildingMode(List<Town> towns)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

foreach (Town town in towns)

{

if (town.Builded == "village" && town.EnableFor.Contains(currentPlayer))

town.SetActive(true);

}

}

/// <summary>

/// Удаляет объекты городов, граничащие с построенным поселением

/// </summary>

/// <param name="buildedTown"></param>

public static void DeleteBorderingTownObjects(Town buildedTown, List<Town> towns)

{

foreach (Town town in towns)

{

if ((buildedTown.Hexes.Contains(town.Hexes[0]) &&

buildedTown.Hexes.Contains(town.Hexes[1]))

||

(buildedTown.Hexes.Contains(town.Hexes[1]) &&

buildedTown.Hexes.Contains(town.Hexes[2]))

||

(buildedTown.Hexes.Contains(town.Hexes[0]) &&

buildedTown.Hexes.Contains(town.Hexes[2])))

{

if (town != buildedTown)

{

town.Builded = "blocked";

}

}

}

}

/// <summary>

/// Собирает ресурсы с нужных гексов. При значении cubesValue = -1

/// собирает ресурсы с каждого гекса

/// </summary>

/// <param name="hexes"></param>

/// <param name="cubesValue"></param>

public static void GetResources(Hex[] hexes, int cubesValue)

{

foreach(Hex hex in hexes)

{

hex.CheckGettingRes(cubesValue);

}

}

/// <summary>

/// Подписывает метод для сбора ресурсов игроком на событие сбора ресурсов с гексов

/// </summary>

/// <param name="borderingHexes"></param>

/// <param name="hexes"></param>

/// <param name="player"></param>

public static void AddHexListeners(string townID, Hex[] hexes, Player player)

{

string[] townData = townID.Split('\_');

for (int i = 1; i < 4; i++)

{

int hexNumber = int.Parse(townData[i]);

if (hexNumber > 0 && hexNumber < 19)

hexes[hexNumber - 1].GettingRes += player.GettingResHandler;

}

}

/// <summary>

/// Загружает данные о количестве ресурсов у игрока

/// </summary>

/// <param name="player"></param>

/// <param name="resLabels"></param>

public static void LoadResValues(Text[] resLabels, int playerNum)

{

Player player = GameScript.Players[playerNum];

resLabels[0].text = player.Wood.ToString();

resLabels[1].text = player.Bricks.ToString();

resLabels[2].text = player.Stone.ToString();

resLabels[3].text = player.Wool.ToString();

resLabels[4].text = player.Wheat.ToString();

}

/// <summary>

/// Загружает интерфейс пользователя

/// </summary>

/// <param name="playerNames"></param>

/// <param name="resLabels"></param>

public static void LoadPlayerInterface(Text[] playerNames, Text[] resLabels)

{

int playersCount = GameScript.playersCount;

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

{

playerNames[i].text = GameScript.Players[(GameScript.CurrentPlayerNum + i) % playersCount].Name;

}

LoadResValues(resLabels, GameScript.CurrentPlayerNum);

}

/// <summary>

/// Деактивирует объекты дорог

/// </summary>

public static void DeactivateRoadObjects(List<Road> roads)

{

foreach(Road road in roads)

{

road.SetActive(false);

}

}

/// <summary>

/// Деактивирует объекты городов

/// </summary>

public static void DeactivateTownObjects(List<Town> towns)

{

foreach (Town town in towns)

{

town.SetActive(false);

}

}

/// <summary>

/// Обменивает ресурсы игроков

/// </summary>

/// <param name="resValues"></param>

/// <param name="exchangingPlayerNum"></param>

public static void MakeDeal(int[] resValues, int exchangingPlayerNum)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

Player exchangingPlayer = GameScript.Players[exchangingPlayerNum];

// Изменяет количество ресурсов у игроков

currentPlayer.Wood += resValues[0];

exchangingPlayer.Wood -= resValues[0];

currentPlayer.Bricks += resValues[1];

exchangingPlayer.Bricks -= resValues[1];

currentPlayer.Stone += resValues[2];

exchangingPlayer.Stone -= resValues[2];

currentPlayer.Wool += resValues[3];

exchangingPlayer.Wool -= resValues[3];

currentPlayer.Wheat += resValues[4];

exchangingPlayer.Wheat -= resValues[4];

}

}

**GameClassLib.cs**

using System.Collections.Generic;

using UnityEngine;

namespace GameClassLib

{

/// <summary>

/// Представляет объект гекса

/// </summary>

public class Hex

{

// Тип ресурса, который производит гекс

string resType;

public string ResType { get { return resType; } }

// Число на кубиках, при котором гекс приносит урожай

int gettingResValue;

public int GettingResValue { get { return gettingResValue; } }

// Событие, вызываемое при сборе урожая с клетки

public delegate void GettingResHandler(string resType);

public event GettingResHandler GettingRes;

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="newResType"></param>

public Hex(string newResType, int newGettingResValue)

{

resType = newResType;

gettingResValue = newGettingResValue;

}

/// <summary>

/// Вызывает событие сбора урожая

/// </summary>

public void CheckGettingRes(int cubesValue)

{

if (cubesValue == gettingResValue || cubesValue == -1)

if (GettingRes != null)

GettingRes(resType);

}

}

/// <summary>

/// Представляет объект игрока

/// </summary>

public class Player

{

// Имя игрока

string name;

public string Name { get { return name; } }

// Цвет игровых объектов

Color color;

public Color Color { get { return color; } }

// Количество ресурсов каждого вида

public int Wood { get; set; }

public int Bricks { get; set; }

public int Stone { get; set; }

public int Wool { get; set; }

public int Wheat { get; set; }

// Количество победных очков игрока

public int Score { get; set; }

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="newName"></param>

public Player(string newName, Color newColor)

{

name = newName;

color = newColor;

}

/// <summary>

/// Позволяет получать ресурсы с клеток

/// </summary>

/// <param name="resType"></param>

public void GettingResHandler(string resType)

{

if (resType == "wood")

Wood++;

else if (resType == "bricks")

Bricks++;

else if (resType == "stone")

Stone++;

else if (resType == "wool")

Wool++;

else if (resType == "wheat")

Wheat++;

}

/// <summary>

/// Отнимает у игрока ресурсы после постройки объекта

/// </summary>

/// <param name=""></param>

public void Build(string objType)

{

switch (objType)

{

case "road":

Wood--;

Bricks--;

break;

case "village":

Wood--;

Bricks--;

Wheat--;

Wool--;

break;

case "town":

Wheat = Wheat - 2;

Stone = Stone - 3;

break;

}

}

}

/// <summary>

/// Представляет объект дороги

/// </summary>

public class Road

{

string name;

public string Name { get { return name; } }

GameObject gameObject;

// Хранит информацию о том, кто из игроков может построить дорогу на этом месте

public List<Player> EnableFor { get; set; }

public bool Builded { get; set; }

// Хранит номера граничащих гексов

string[] hexes = new string[2];

public string[] Hexes { get { return hexes; } }

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="newName"></param>

public Road(string newName)

{

name = newName;

gameObject = GameObject.Find(newName);

EnableFor = new List<Player>();

// Сохраняет номера граничащих гексов

hexes[0] = newName.Split('\_')[1];

hexes[1] = newName.Split('\_')[2];

}

public delegate void BorderingRoadBuildedHandler(Road road);

public event BorderingRoadBuildedHandler BorderingRoadBuilded;

/// <summary>

/// Обрабатывает событие постройки нового объекта на карте

/// </summary>

public void RoadBuildedHandler(string buildedObjName)

{

if (buildedObjName == name)

{

EnableFor.Clear();

Builded = true;

BorderingRoadBuilded(this);

}

}

/// <summary>

/// Активирует/деактивирует данный объект для строительства

/// </summary>

public void SetActive(bool activate)

{

gameObject.GetComponent<Collider2D>().enabled = activate;

}

}

/// <summary>

/// Представляет объект поселения

/// </summary>

public class Town

{

// Название объекта

string name;

public string Name { get { return name; } }

GameObject gameObject;

// Хранит информацию о том, кто из игроков может построить дорогу на этом месте

List<Player> enableFor = new List<Player>();

public List<Player> EnableFor { get { return enableFor; } }

public string Builded { get; set; }

// Дороги, примыкающие к перекрестку

List<Road> roads = new List<Road>();

public List<Road> Roads { get { return roads; } }

// Граничащие гексы

List<string> hexes = new List<string>();

public List<string> Hexes { get { return hexes; } }

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="name"></param>

/// <param name="roadObjects"></param>

public Town(string newName, List<Road> roadObjects)

{

name = newName;

Builded = "none";

string[] townData = name.Split('\_');

gameObject = GameObject.Find(newName);

// Сохраняет номера граничащих гексов

for (int i = 1; i < 4; i++)

{

hexes.Add(townData[i]);

}

// Сохраняет граничащие дороги

foreach (Road road in roadObjects)

{

if (hexes.Contains(road.Hexes[0]) && hexes.Contains(road.Hexes[1]))

{

roads.Add(road);

road.BorderingRoadBuilded += BorderingRoadBuildedHandler;

}

}

}

/// <summary>

/// Обрабатывает событие постройки нового объекта поселения/города

/// Делает перекресток недоступным для строительства

/// </summary>

/// <param name="buildedObjName"></param>

public void TownBuildedHandler(string buildedObjName, bool townBuilded)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

if (buildedObjName == name)

{

enableFor.Clear();

if (townBuilded)

{

Builded = "town";

}

else

{

Builded = "village";

enableFor.Add(currentPlayer);

}

}

}

/// <summary>

/// Обрабатывает строительство дороги, примыкающей к перекрестку.

/// Делает перекресток доступным для строительства поселения и дорог рядом для игрока,

/// проложившего дорогу.

/// </summary>

/// <param name="buildedRoad"></param>

public void BorderingRoadBuildedHandler(Road buildedRoad)

{

Player currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

if (Builded == "none" && !enableFor.Contains(currentPlayer))

{

enableFor.Add(currentPlayer);

}

foreach (Road road in roads)

{

if (!road.Builded && !road.EnableFor.Contains(currentPlayer))

{

road.EnableFor.Add(currentPlayer);

}

}

}

/// <summary>

/// Активирует/деактивирует данный объект для строительства

/// </summary>

public void SetActive(bool activate)

{

gameObject.GetComponent<Collider2D>().enabled = activate;

}

}

}

**RoadObjectScript.cs**

using UnityEngine;

public class RoadObjectScript : MonoBehaviour {

public Sprite sprite;

// Событие, вызываемое при постройке нового поселения

public delegate void RoadBuildedHandler(string objectID);

public static event RoadBuildedHandler RoadBuilded;

/// <summary>

/// Назначает спрайт объекта

/// </summary>

public void OnMouseDown()

{

GetComponent<SpriteRenderer>().sprite = sprite;

GetComponent<SpriteRenderer>().color = GameScript.Players[GameScript.CurrentPlayerNum].Color;

if (RoadBuilded != null)

RoadBuilded(gameObject.name);

GetComponent<Collider2D>().enabled = false;

}

}

**TownObjectScript.cs**

using UnityEngine;

public class TownObjectScript : MonoBehaviour {

public Sprite villageSprite;

public Sprite townSprite;

// Событие, вызываемое при постройке нового поселения

public delegate void VillageBuildedHandler(string objectID, bool townBuilded);

public static event VillageBuildedHandler TownBuilded;

/// <summary>

/// Назначает спрайт объекта

/// </summary>

public void OnMouseDown () {

SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();

spriteRenderer.sprite = spriteRenderer.sprite == null ? villageSprite : townSprite;

spriteRenderer.color = GameScript.Players[GameScript.CurrentPlayerNum].Color;

if (TownBuilded != null)

TownBuilded(name, spriteRenderer.sprite == townSprite ? true : false);

GetComponent<Collider2D>().enabled = false;

}

}

**TradingScript.cs**

using UnityEngine;

using UnityEngine.UI;

using GameClassLib;

public class TradingScript : MonoBehaviour {

// Текстовые поля, отображающие имена игроков

public Text player1Text;

public Text player2Text;

public Text player1Text\_4players;

public Text player2Text\_4players;

public Text player3Text\_4players;

// Текстовые поля, отображающие количество ресурсов у каждого игрока

public Text[] player1resLabels;

public Text[] player2resLabels;

public Text[] player1resLabels\_4players;

public Text[] player2resLabels\_4players;

public Text[] player3resLabels\_4players;

// Экран торговли для 3-х и 4-х игроков

public GameObject tradingScreen3players;

public GameObject tradingScreen4players;

// Экран обмена ресурсами

public GameObject exchangeScreen;

public Text player1Txt\_ExchangeScreen;

public Text player2Txt\_ExchangeScreen;

public Text[] player1ResLabels\_ExchangeScreen;

public Text[] player2ResLabels\_ExchangeScreen;

// Номер игрока, с которым производится обмен

static int exchangingPlayerNum;

// Количество игроков

static int playersCount;

// Данные о совершаемой сделке

static Player currentPlayer;

static Player exchangingPlayer;

static int[] exchangingResValues;

// Текстовые поля, отображающие количество ресурсов в основном интерфейсе

public Text[] resLabels;

/// <summary>

/// Обрабатывает нажатие кнопки "Торговля" в игровом меню.

/// Выводит меню торговли

/// </summary>

public void TradeBtn\_OnClick()

{

exchangingResValues = new int[] { 0, 0, 0, 0, 0 };

// Сбрасывает информацию о предыдущей сделке

for (int i = 0; i < 5; i++)

{

player1ResLabels\_ExchangeScreen[i].text = "0";

player2ResLabels\_ExchangeScreen[i].text = "0";

}

if (GameScript.Players.Length == 3)

{

playersCount = 3;

int currentPlayer = GameScript.CurrentPlayerNum;

// Загружает информацию о других игроках

int player1Num = (currentPlayer + 1) % 3;

player1Text.text = GameScript.Players[player1Num].Name;

GameMethods.LoadResValues(player1resLabels, player1Num);

int player2Num = (currentPlayer + 2) % 3;

player2Text.text = GameScript.Players[player2Num].Name;

GameMethods.LoadResValues(player2resLabels, player2Num);

// Выводит меню торговли

tradingScreen3players.SetActive(true);

}

else if (GameScript.Players.Length == 4)

{

playersCount = 4;

int currentPlayer = GameScript.CurrentPlayerNum;

// Загружает информацию о других игроках

int player1Num = (currentPlayer + 1) % 4;

player1Text\_4players.text = GameScript.Players[player1Num].Name;

GameMethods.LoadResValues(player1resLabels\_4players, player1Num);

int player2Num = (currentPlayer + 2) % 4;

player2Text\_4players.text = GameScript.Players[player2Num].Name;

GameMethods.LoadResValues(player2resLabels\_4players, player2Num);

int player3Num = (currentPlayer + 3) % 4;

player3Text\_4players.text = GameScript.Players[player3Num].Name;

GameMethods.LoadResValues(player3resLabels\_4players, player3Num);

// Выводит меню торговли

tradingScreen4players.SetActive(true);

}

}

/// <summary>

/// Обрабатывает нажатие кнопки "Отмена" в меню торговли.

/// Скрывает меню торговли.

/// </summary>

public void CancelBtn\_OnClick()

{

if (playersCount == 3)

tradingScreen3players.SetActive(false);

else if (playersCount == 4)

tradingScreen4players.SetActive(false);

}

/// <summary>

/// Загружает экран обмена ресурсами

/// </summary>

/// <param name="playerNum"></param>

public void LoadExchangeScreen(int playerNum)

{

exchangingPlayerNum = playersCount == 3 ? (GameScript.CurrentPlayerNum + playerNum) % 3 :

(GameScript.CurrentPlayerNum + playerNum) % 4;

currentPlayer = GameScript.Players[GameScript.CurrentPlayerNum];

exchangingPlayer = GameScript.Players[exchangingPlayerNum];

// Скрывает меню торговли

CancelBtn\_OnClick();

// Загружает экран обмена ресурсами

player1Txt\_ExchangeScreen.text = currentPlayer.Name;

player2Txt\_ExchangeScreen.text = GameScript.Players[exchangingPlayerNum].Name;

exchangeScreen.SetActive(true);

}

/// <summary>

/// Обрабатывает нажатие кнопки добавления ресурса на экране обмена.

/// Добавляет ресурс текущему игроку.

/// </summary>

public void CurrentPlayerAddBtn\_OnClick(int resIndex)

{

switch (resIndex)

{

case 0:

if (exchangingPlayer.Wood > exchangingResValues[0])

{

player1ResLabels\_ExchangeScreen[0].text = (++exchangingResValues[0]).ToString();

player2ResLabels\_ExchangeScreen[0].text = (-exchangingResValues[0]).ToString();

}

break;

case 1:

if (exchangingPlayer.Bricks > exchangingResValues[1])

{

player1ResLabels\_ExchangeScreen[1].text = (++exchangingResValues[1]).ToString();

player2ResLabels\_ExchangeScreen[1].text = (-exchangingResValues[1]).ToString();

}

break;

case 2:

if (exchangingPlayer.Stone > exchangingResValues[2])

{

player1ResLabels\_ExchangeScreen[2].text = (++exchangingResValues[2]).ToString();

player2ResLabels\_ExchangeScreen[2].text = (-exchangingResValues[2]).ToString();

}

break;

case 3:

if (exchangingPlayer.Wool > exchangingResValues[3])

{

player1ResLabels\_ExchangeScreen[3].text = (++exchangingResValues[3]).ToString();

player2ResLabels\_ExchangeScreen[3].text = (-exchangingResValues[3]).ToString();

}

break;

case 4:

if (exchangingPlayer.Wheat > exchangingResValues[4])

{

player1ResLabels\_ExchangeScreen[4].text = (++exchangingResValues[4]).ToString();

player2ResLabels\_ExchangeScreen[4].text = (-exchangingResValues[4]).ToString();

}

break;

}

}

/// <summary>

/// Обрабатывает нажатие кнопки добавления ресурса на экране обмена.

/// Отнимает ресурс у текущего игрока.

/// </summary>

/// <param name="resIndex"></param>

public void ExchangingPlayerAddBtn\_OnClick(int resIndex)

{

switch (resIndex)

{

case 0:

if (currentPlayer.Wood > -exchangingResValues[0])

{

player1ResLabels\_ExchangeScreen[0].text = (--exchangingResValues[0]).ToString();

player2ResLabels\_ExchangeScreen[0].text = (-exchangingResValues[0]).ToString();

}

break;

case 1:

if (currentPlayer.Bricks > -exchangingResValues[1])

{

player1ResLabels\_ExchangeScreen[1].text = (--exchangingResValues[1]).ToString();

player2ResLabels\_ExchangeScreen[1].text = (-exchangingResValues[1]).ToString();

}

break;

case 2:

if (currentPlayer.Stone > -exchangingResValues[2])

{

player1ResLabels\_ExchangeScreen[2].text = (--exchangingResValues[2]).ToString();

player2ResLabels\_ExchangeScreen[2].text = (-exchangingResValues[2]).ToString();

}

break;

case 3:

if (currentPlayer.Wool > -exchangingResValues[3])

{

player1ResLabels\_ExchangeScreen[3].text = (--exchangingResValues[3]).ToString();

player2ResLabels\_ExchangeScreen[3].text = (-exchangingResValues[3]).ToString();

}

break;

case 4:

if (currentPlayer.Wheat > -exchangingResValues[4])

{

player1ResLabels\_ExchangeScreen[4].text = (--exchangingResValues[4]).ToString();

player2ResLabels\_ExchangeScreen[4].text = (-exchangingResValues[4]).ToString();

}

break;

}

}

/// <summary>

/// Обрабатывает нажатие кнопки "Подтвердить"

/// </summary>

public void ConfirmBtn\_OnClick()

{

GameMethods.MakeDeal(exchangingResValues, exchangingPlayerNum);

GameMethods.LoadResValues(resLabels, GameScript.CurrentPlayerNum);

exchangeScreen.SetActive(false);

}

/// <summary>

/// Обрабатывает нажатие кнопки "Назад"

/// </summary>

public void ExchangeScreenCancelBtn\_OnClick()

{

exchangeScreen.SetActive(false);

// Загружает меню торговли

TradeBtn\_OnClick();

}

}

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |