Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

________ Світлана ПОПЕРЕШНЯК

“___” _________________ 2023 р.

Колекційна карткова гра зі штучним інтелектом

Текст програми

КПІ.ІП-1116.045480.03.12

“ПОГОДЖЕНО”

Керівник роботи:

____________ Світлана ПОПЕРЕШНЯК

Консультант:	Виконавець:
___________ Максим ГОЛОВЧЕНКО	__________ КУЗЬМЕНКОВ

Київ – 2023

Файл AnimateGif.cs

```
using UnityEngine;
using UnityEngine.UI;

public class AnimateGif : MonoBehaviour
{
    public Texture2D[] frames;
    public RawImage backgroundImage;
    private float framesPerSecond = 12.5f;

    private void Update()
    {
        if (backgroundImage != null)
        {
            float index = Time.time * framesPerSecond;
            index = index % frames.Length;
            backgroundImage.texture = frames[(int)index];
        }
    }
}
```

Файл ButtonBehaviour.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;



public class ButtonBehaviourScr : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler, IPointerDownHandler, IPointerUpHandler
{

    public Color normalColor;
    public Color highlightColor;
    public Color pressedColor;
    public float YOffset = 5f;
    public AudioSource audioSource;
    public Button button;




    private Vector2 originalPosition;
    private Vector2 enteredPosition;
    public TextMeshProUGUI buttonText;

    public void Start()
    {

        //audioSource = GetComponent<AudioSource>();
        buttonText = GetComponentInChildren<TextMeshProUGUI>();
        originalPosition = buttonText.rectTransform.anchoredPosition;
        enteredPosition = buttonText.rectTransform.anchoredPosition;
        enteredPosition.y -= YOffset;

        ColorUtility.TryParseHtmlString("#CAC5C1", out normalColor);
        ColorUtility.TryParseHtmlString("#A2A09E", out highlightColor);
        ColorUtility.TryParseHtmlString("#5A5A5A", out pressedColor);
        buttonText.color = normalColor;

        if (button.IsInteractable() == false)
            buttonText.color = pressedColor;
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (button.IsInteractable() == false)
            return;
        buttonText.color = highlightColor;
        buttonText.rectTransform.anchoredPosition = new Vector2(0,
originalPosition.y - YOffset);
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (button.IsInteractable() == false)
            return;
        buttonText.color = normalColor;
        buttonText.rectTransform.anchoredPosition = originalPosition;



    }
    public void OnPointerDown(PointerEventData eventData)
    {

        if (button.IsInteractable() == false)
            return;
        buttonText.color = pressedColor;
        buttonText.rectTransform.anchoredPosition = new Vector2(0, 
originalPosition.y - YOffset * 2);
        audioSource.Play();
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        if (button.IsInteractable() == false)
            return;
        buttonText.color = eventData.hovered.Contains(gameObject) ? 
highlightColor : normalColor;

        buttonText.rectTransform.anchoredPosition = originalPosition;

    }


}
```

Файл MainMenuScr.cs

```
using System.IO;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenuScr : MonoBehaviour
{
    //public Transform menu;
    public Button PlayButton;
    public Button ChangeDeckButton;
    public Button SettingsButton;
    public Button ExitButton;
    public GameObject settingsPanel;

    public GameSettings Settings = new GameSettings();

    private void Awake()
    {
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            Settings = JsonUtility.FromJson<GameSettings>(json);
        }
        else
        {
            Settings.soundVolume = .5f;
            Settings.timer = 120;
            Settings.timerIsOn = true;
            Settings.difficulty = "Normal";
        }
        AudioListener.volume = Settings.soundVolume;
    }

    void Start()
    {
        PlayButton.onClick.AddListener(OnPlayButtonClicked);
        ChangeDeckButton.onClick.AddListener(OnChangeDeckButtonClicked);
        SettingsButton.onClick.AddListener(OnSettingsButtonClicked);
        ExitButton.onClick.AddListener(OnExitButtonClicked);
    }

    public void OnPlayButtonClicked()
    {
        SceneManager.LoadScene("Gameplay");
    }

    public void OnChangeDeckButtonClicked()
    {
        SceneManager.LoadScene("ChangeDeck_Scene");
    }

    public void OnSettingsButtonClicked()
    {
        settingsPanel.SetActive(true);
    }

    public void OnExitButtonClicked()
    {
        Application.Quit();
    }

}
```

Файл SettingsManagerScr.cs

```
using System;
using System.IO;
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

[Serializable]
public class GameSettings
{
    public float soundVolume;
    public int timer;
    public bool timerIsOn;
    public string difficulty; // Easy, Normal, Hard
}

public class SettingsManager : MonoBehaviour
{

    public GameSettings currentSettings = new GameSettings();
    public Slider soundSlider;
    public TextMeshProUGUI soundTxt;
    public ToggleGroup timerToggleGroup;
    public ToggleGroup difficultyToggleGroup;

    public GameObject pausePanel, settingsPanel;

    public AudioSource audioSource;


    private void Awake()
    {
        LoadSettings();
        if (soundSlider != null)
            soundSlider.onValueChanged.AddListener(OnSoundVolumeChanged);
        AddToggleListeners(timerToggleGroup, OnTimerToggleChanged);
        AddToggleListeners(difficultyToggleGroup, OnDifficultyToggleChanged);
    }

    private void AddToggleListeners(ToggleGroup toggleGroup,
UnityAction<bool> callback)
    {
        foreach (Toggle toggle in
toggleGroup.GetComponentsInChildren<Toggle>())
        {
            toggle.onValueChanged.AddListener(callback);
        }
    }

    public void LoadSettings()
    {
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            currentSettings = JsonUtility.FromJson<GameSettings>(json);
        }
        else
        {
            CreateDefaultSettings();
        }

        ApplySettingsToUI();
    }

    public void SaveSettings()
    {
        string json = JsonUtility.ToJson(currentSettings, true);
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        File.WriteAllText(filePath, json);
    }

    void CreateDefaultSettings()
    {
        TextAsset settingsAsset =
Resources.Load<TextAsset>("Settings/Settings");
```

```
        if (settingsAsset != null)
        {
            currentSettings =
JsonUtility.FromJson<GameSettings>(settingsAsset.text);
        }
        else
        {
            currentSettings.soundVolume = .5f;
            currentSettings.timer = 120;
            currentSettings.timerIsOn = true;
            currentSettings.difficulty = "Normal";
        }
        SaveSettings();
    }

    private void ApplySettingsToUI()
    {
        if (soundSlider != null)
        {
            soundSlider.value = currentSettings.soundVolume;
            soundTxt.text = (currentSettings.soundVolume *
100).ToString("F0");
        }


        foreach (Transform toggleTransform in timerToggleGroup.transform)
        {
            Toggle toggle = toggleTransform.GetComponent<Toggle>();
            if (toggle != null)
            {
                toggle.isOn = false;
            }
        }

        Toggle toggleToActivate = null;

        switch (currentSettings.timer)
        {
            case 0:
                toggleToActivate =
timerToggleGroup.transform.Find("OffToggle").GetComponent<Toggle>();
                break;
            case 60:
                toggleToActivate =
timerToggleGroup.transform.Find("60sToggle").GetComponent<Toggle>();
                break;
            case 120:
                toggleToActivate =
timerToggleGroup.transform.Find("120sToggle").GetComponent<Toggle>();
                break;
            case 180:
                toggleToActivate =
timerToggleGroup.transform.Find("180sToggle").GetComponent<Toggle>();
                break;

        }
        if (toggleToActivate != null)
        {
            toggleToActivate.isOn = true;
        }

        if (difficultyToggleGroup != null)
        {
            foreach (Transform toggleTransform in
difficultyToggleGroup.transform)
            {
                Toggle toggle = toggleTransform.GetComponent<Toggle>();
                if (toggle != null)
                {
                    toggle.isOn = false;
                }
            }

            toggleToActivate = null;

            switch (currentSettings.difficulty)
            {
                case "Easy":
                    toggleToActivate =
difficultyToggleGroup.transform.Find("EasyToggle").GetComponent<Toggle>();
                    break;
                case "Normal":
                    toggleToActivate =
difficultyToggleGroup.transform.Find("NormalToggle").GetComponent<Toggle>();
                    break;
                case "Hard":
                    toggleToActivate =
difficultyToggleGroup.transform.Find("HardToggle").GetComponent<Toggle>();
                    break;

            }

            if (toggleToActivate != null)
            {
                toggleToActivate.isOn = true;
            }
        }
    }

    public void OnSoundVolumeChanged(float volume)
    {
        currentSettings.soundVolume = volume;
        AudioListener.volume = volume;
        soundTxt.text = (currentSettings.soundVolume * 100).ToString("F0");
    }

    public void OnTimerToggleChanged(bool firstentry)
    {
        Toggle activeToggle =
timerToggleGroup.ActiveToggles().FirstOrDefault();

        if (activeToggle != null)
        {
            // Обновляем настройку таймера в зависимости от того, какой тоггл
активен
            if (activeToggle.name == "OffToggle")
            {
                currentSettings.timer = 0;
                currentSettings.timerIsOn = false;
            }
            else if (activeToggle.name == "60sToggle")
            {
                currentSettings.timer = 60;
                currentSettings.timerIsOn = true;
            }
            else if (activeToggle.name == "120sToggle")
            {
                currentSettings.timer = 120;
                currentSettings.timerIsOn = true;
            }
            else if (activeToggle.name == "180sToggle")
            {
                currentSettings.timer = 180;
                currentSettings.timerIsOn = true;
            }
        }
    }

    public void OnDifficultyToggleChanged(bool firstentry)
    {
        Toggle activeToggle =
difficultyToggleGroup.ActiveToggles().FirstOrDefault();

        if (activeToggle != null)
        {
            // Обновляем настройку таймера в зависимости от того, какой тоггл
активен
            if (activeToggle.name == "EasyToggle")
            {
                currentSettings.difficulty = "Easy";
            }
            else if (activeToggle.name == "NormalToggle")
            {
                currentSettings.difficulty = "Normal";
            }
            else if (activeToggle.name == "HardToggle")
            {
                currentSettings.difficulty = "Hard";
            }
        }
    }

    public void BackToPause()
    {
        SaveSettings();
        settingsPanel.SetActive(false);
        pausePanel.SetActive(true);

    }

    public void BackToMenu()
    {
        SaveSettings();
        settingsPanel.SetActive(false);
    }

}
```

Файл ButtonManagerScr.cs

```
using System.IO;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class ButtonManagerScr : MonoBehaviour
{
    public GameObject WhatToChangeMenu;
    public GameObject WarningObj;
    public GameObject CardLine;
    public GameObject CardPref;
    public Transform MyDeck;
    public Transform EnemyDeck;
    public Transform MyScrollView;
    public Transform EnemyScrollView;
    public DecksManagerScr DecksManager;
    public TextMeshProUGUI Title;
    public TextMeshProUGUI WarningMsg;
    public TextMeshProUGUI DeckCounter;
    public Button ExitButton;
    public Button MyDeckButton;
    public Button EnemyDeckButton;
    public Button ChangeDeckButton;
    public Transform CardsLine;

    public GameSettings Settings = new GameSettings();

    private void Awake()
    {
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            Settings = JsonUtility.FromJson<GameSettings>(json);
        }
        else
        {
            Settings.soundVolume = .5f;
            Settings.timer = 120;
            Settings.timerIsOn = true;
            Settings.difficulty = "Normal";
            string json = File.ReadAllText(filePath);
            Settings = JsonUtility.FromJson<GameSettings>(json);
        }
        AudioListener.volume = Settings.soundVolume;
    }

    void Start()
    {
        DecksManager = gameObject.GetComponent<DecksManagerScr>();
        Title.text = "";
        DeckCounter.text = "";
        WarningMsg.text = "";
        MyDeck.gameObject.SetActive(false);
        MyScrollView.gameObject.SetActive(false);
        EnemyScrollView.gameObject.SetActive(false);
        WarningObj.SetActive(false);
        ExitButton.onClick.AddListener(OnExitButtonClicked);
        MyDeckButton.onClick.AddListener(OnMyDeckButtonClicked);
        EnemyDeckButton.onClick.AddListener(OnEnemyDeckButtonClicked);
        ChangeDeckButton.onClick.AddListener(OnChangeDeckButtonClicked);
        ShowDeck(MyDeck);
        ShowDeck(EnemyDeck);
        PaintCardsGreen(MyDeck, DecksManager.GetMyDeck());
        PaintCardsGreen(EnemyDeck, DecksManager.GetEnemyDeck());

    }

    public void OnExitButtonClicked()
    {
        EnemyScrollView.gameObject.SetActive(false);
        MyScrollView.gameObject.SetActive(false);
        EnemyDeck.gameObject.SetActive(false);
        MyDeck.gameObject.SetActive(false);
        if (DecksManager.GetEnemyDeck().cards.Count < DecksManager.MaxDeckLen
|| DecksManager.GetMyDeck().cards.Count < DecksManager.MaxDeckLen)
        {
            WarningObj.SetActive(true);

            if (DecksManager.GetMyDeck().cards.Count <
DecksManager.MaxDeckLen)
                WarningMsg.text += "Player deck misses " +
(DecksManager.MaxDeckLen - DecksManager.GetMyDeck().cards.Count).ToString() +
" cards.";
            if (DecksManager.GetEnemyDeck().cards.Count <
DecksManager.MaxDeckLen)
                WarningMsg.text += "\nEnemy deck misses " +
(DecksManager.MaxDeckLen -
DecksManager.GetEnemyDeck().cards.Count).ToString() + " cards.";
            WarningMsg.text += "\nMissing cards will be added
automatically.";
        }
        else
        {
            Exit();
        }
    }

    public void Exit()
    {
        DecksManager.AddMissingCards();
        DecksManager.SaveAllDecks();
        SceneManager.LoadScene("MainMenu_Scene");
    }

    public void OnMyDeckButtonClicked()
    {
        EnemyScrollView.gameObject.SetActive(false);
        MyScrollView.gameObject.SetActive(true);
        EnemyDeck.gameObject.SetActive(false);
        WhatToChangeMenu.SetActive(false);
        Title.text = "My deck";
        DeckCounter.text = DecksManager.GetMyDeck().cards.Count.ToString() +
" / 30";
        MyDeck.gameObject.SetActive(true);
    }

    public void OnEnemyDeckButtonClicked()
    {
        MyScrollView.gameObject.SetActive(false);
        EnemyScrollView.gameObject.SetActive(true);
        MyDeck.gameObject.SetActive(false);
        WhatToChangeMenu.SetActive(false);
        Title.text = "Enemy deck";
        DeckCounter.text = DecksManager.GetEnemyDeck().cards.Count.ToString()
+ " / 30";
        EnemyDeck.gameObject.SetActive(true);

    }

    public void OnChangeDeckButtonClicked()
    {
        Title.text = "";
        DeckCounter.text = "";
        MyDeck.gameObject.SetActive(false);
        EnemyDeck.gameObject.SetActive(false);
        WarningObj.SetActive(false);
        WhatToChangeMenu.SetActive(true);
        MyScrollView.gameObject.SetActive(false);
        EnemyScrollView.gameObject.SetActive(false);



    }

    public void ShowDeck(Transform Deck)
    {
        int NumOfCards = DecksManager.GetAllCards().cards.Count;

        for (int i = 0; i < NumOfCards; i++)
        {
            Transform newCardLine = Instantiate(CardsLine, Deck, false);
            newCardLine.transform.SetParent(Deck.transform, false);
            newCardLine.gameObject.SetActive(true);



            for (int j = 0; j < 8 && i < NumOfCards; j++)
            {
                GameObject newCard = Instantiate(CardPref, newCardLine,
false);
                newCard.SetActive(true);
                newCard.transform.SetParent(newCardLine.transform, false);
                //CardInfoScript cardInfo =
newCard.GetComponent<CardInfoScript>();
                CardController cardC =
newCard.GetComponent<CardController>();
                cardC.Init(DecksManager.GetAllCards().cards[i], true);

                //Debug.Log(cardC.Card.HP);
                if (cardC.Info != null)
                {
                    //CC.Info.ShowCardInfo();
                    cardC.Info.ShowCardInfo();



                }
                i++;
            }



            i--;
        }
    }

    public void ChangeDeck(AllCards Deck, Card card)
    {
        if (Deck.ContainsCard(card))
        {
            DecksManager.DeleteCardFromDeck(Deck, card);
        }
        else
        {
            DecksManager.AddCardToDeck(Deck, card);
        }
    }

    public void PaintCardsGreen(Transform Deck, AllCards cards)
    {
        foreach (Transform cardline in Deck)
        {
            foreach (Transform Card in cardline)
            {

                CardController CC = Card.GetComponent<CardController>();
                if (cards.ContainsCard(CC.Card))
                {
                    CC.Info.PaintGreen();
                }
            }
        }
    }

    public void UpdateDeckCounters()
    {

        Debug.Log("Update called");
        if (MyDeck.gameObject.activeSelf)
        {
            DeckCounter.text = 
DecksManager.GetMyDeck().cards.Count.ToString() + " / 30";
        }
        else if (EnemyDeck.gameObject.activeSelf)
        {
            DeckCounter.text = 
DecksManager.GetEnemyDeck().cards.Count.ToString() + " / 30";
        }
    }





}
```

Файл CardInteractionScr.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class CardInteractionScr : MonoBehaviour, IPointerExitHandler, 
IPointerDownHandler
{
    CardController CC;
    ButtonManagerScr buttonManager;
    UnityEngine.Color OriginalColor;
    public AudioSource audioSource;
    Camera MainCamera;
    DecksManagerScr DecksManager;
    UnityEngine.Color GreenColor;

    void Start()
    {
        GreenColor = new UnityEngine.Color(13f / 255f, 142f / 255f, 0f /
255f, 1f);
        CC = GetComponent<CardController>();
        MainCamera = Camera.allCameras[0];
        buttonManager = MainCamera.GetComponent<ButtonManagerScr>();

        OriginalColor = CC.Info.card_BG.color;



    }

    public void OnPointerExit(PointerEventData eventData)
    {
        CC.Info.PaintAnother(OriginalColor);
    }

    public void OnPointerDown(PointerEventData eventData)
    {

        if (buttonManager.MyDeck.gameObject.activeSelf)
        {
            if ((buttonManager.DecksManager.GetMyDeck().cards.Count <=
buttonManager.DecksManager.MinDeckLen &&
CC.Info.card_BG.color.Equals(GreenColor)) ||
(buttonManager.DecksManager.GetMyDeck().cards.Count >=
buttonManager.DecksManager.MaxDeckLen &&
CC.Info.card_BG.color.Equals(UnityEngine.Color.white)))
            {
                return;
            }
            ChangeCardColor();
            buttonManager.ChangeDeck(buttonManager.DecksManager.GetMyDeck(),
CC.Card);
            buttonManager.UpdateDeckCounters();
        }
        else if (buttonManager.EnemyDeck.gameObject.activeSelf)
        {
            if ((buttonManager.DecksManager.GetEnemyDeck().cards.Count <=
buttonManager.DecksManager.MinDeckLen &&
CC.Info.card_BG.color.Equals(GreenColor)) ||
(buttonManager.DecksManager.GetEnemyDeck().cards.Count >=
buttonManager.DecksManager.MaxDeckLen &&
CC.Info.card_BG.color.Equals(UnityEngine.Color.white)))
            {
                return;
            }
            ChangeCardColor();

buttonManager.ChangeDeck(buttonManager.DecksManager.GetEnemyDeck(), CC.Card);
            buttonManager.UpdateDeckCounters();
        }
    }

    public void ChangeCardColor()
    {
        audioSource.Play();

        if (OriginalColor.Equals(GreenColor))
        {
            CC.Info.PaintWhite();
            OriginalColor = CC.Info.card_BG.color;
        }
        else
        {
            CC.Info.PaintGreen();
            OriginalColor = CC.Info.card_BG.color;
        }
    }
}
```

Файл DecksManagerScr.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using static Card;

[Serializable]
public class Card
```

```
{

    public enum CardClass
    {
        /*0*/
        ENTITY,
        /*1*/
        ENTITY_WITH_ABILITY,
        /*2*/
        SPELL
    }
    public enum AbilityType
    {
        /*0*/
        NO_ABILITY,
        /*1*/
        LEAP,
        /*2*/
        PROVOCATION,
        /*3*/
        SHIELD,
        /*4*/
        DOUBLE_ATTACK,
        /*5*/
        REGENERATION_EACH_TURN,
        /*6*/
        INCREASE_ATTACK_EACH_TURN,
        /*7*/
        HORDE,
        /*8*/
        ADDITIONAL_MANA_EACH_TURN,
        /*9*/
        ALLIES_INSPIRATION,
        /*10*/
        EXHAUSTION
    }

    public enum SpellType
    {
        /*0*/
        NO_SPELL,
        /*1*/
        HEAL_ALLY_FIELD_CARDS,
        /*2*/
        DAMAGE_ENEMY_FIELD_CARDS,
        /*3*/
        HEAL_ALLY_HERO,
        /*4*/
        DAMAGE_ENEMY_HERO,
        /*5*/
        HEAL_ALLY_CARD,
        /*6*/
        SHIELD_ON_ALLY_CARD,
        /*7*/
        PROVOCATION_ON_ALLY_CARD,
        /*8*/
        BUFF_CARD_DAMAGE,
        /*9*/
        DEBUFF_CARD_DAMAGE,
        /*10*/
        SILENCE,
        /*11*/
        KILL_ALL

    }

    public enum TargetType
    {
        NO_TARGET,
        ALLY_CARD_TARGET,
        ENEMY_CARD_TARGET
    }

    public int id;
    public string Title, Description, LogoPath;
    public CardClass Class;
    public int Attack, HP, ManaCost;
    public bool CanAttack;
    public bool IsPlaced;

    public List<AbilityType> Abilities;
    public SpellType Spell;
    public TargetType SpellTarget;
    public int SpellValue;

    public int TimesTookDamage;
```

```
public int TimesDealedDamage;

public bool HasAbility
{
    get { return !Abilities.Exists(x => x == AbilityType.NO_ABILITY); }
}

public bool IsProvocation
{
    get { return Abilities.Exists(x => x == AbilityType.PROVOCATION); }
}

public bool IsSpell
{
    get { return Spell != SpellType.NO_SPELL; }
}

public void GetDamage(int dmg)
{
    if (dmg >= 0)
    {
        if (Abilities.Exists(x => x == AbilityType.SHIELD))
        {
            Abilities.Remove(AbilityType.SHIELD);
            if (Abilities.Count == 0)
            {
                Abilities.Add(AbilityType.NO_ABILITY);
            }
        }
        else
            HP -= dmg;
    }

}

public bool IsAlive()
{
    if (HP > 0)
    {
        return true;
    }
    return false;
}
```

```
    public Card GetCopy()
    {
        Card card = new Card();
        card = this;
        //card.Abilities = new List<AbilityType>(Abilities);
        return card;
    }

    public Card GetDeepCopy()
    {
        Card card = new Card();

        // Копируем простые и перечисляемые типы данных
        card.id = this.id;
        card.Title = this.Title;
        card.Description = this.Description;
        card.LogoPath = this.LogoPath;
        card.Class = this.Class;
        card.Attack = this.Attack;
        card.HP = this.HP;
        card.ManaCost = this.ManaCost;
        card.CanAttack = this.CanAttack;
        card.IsPlaced = this.IsPlaced;
        card.Spell = this.Spell;
        card.SpellTarget = this.SpellTarget;
        card.SpellValue = this.SpellValue;
        card.TimesTookDamage = this.TimesTookDamage;
        card.TimesDealedDamage = this.TimesDealedDamage;

        // Для коллекций создаем новые экземпляры (глубокое копирование)
        card.Abilities = new List<AbilityType>(this.Abilities);

        return card;
    }
}

public class AllCards
{
    public List<Card> cards = new List<Card>();

    public bool ContainsCard(Card CheckedCard)
    {
        foreach (Card card in cards)
        {
            if (card.id == CheckedCard.id)
            {
                return true;
            }
        }
        return false;
    }
}

public class DecksManagerScr : MonoBehaviour
{
    private AllCards allCardsDeck;
    private AllCards MyDeck;
    private AllCards EnemyDeck;
    public int MinDeckLen = 5;
    public int MaxDeckLen = 30;

    public AllCards GetAllCards() { return allCardsDeck; }
    public AllCards GetMyDeck() { return MyDeck; }
    public AllCards GetEnemyDeck() { return EnemyDeck; }
    public AllCards GetMyDeckCopy()
    {
        AllCards deck = new AllCards();
        foreach (Card card in MyDeck.cards)
        {
            deck.cards.Add(card.GetDeepCopy());
        }
        return deck;
    }
    public AllCards GetEnemyDeckCopy()
    {
        AllCards deck = new AllCards();
        foreach (Card card in EnemyDeck.cards)
        {
            deck.cards.Add(card.GetDeepCopy());
        }
        return deck;
    }

    public void Awake()
    {
        allCardsDeck = new AllCards();
        MyDeck = new AllCards();
        EnemyDeck = new AllCards();

        TextAsset allCardsText =
Resources.Load<TextAsset>("CardsInfo/AllCards");
        allCardsDeck = JsonUtility.FromJson<AllCards>(allCardsText.text);

        LoadOrCreateDeck(ref MyDeck, "MyDeck.json");
        LoadOrCreateDeck(ref EnemyDeck, "EnemyDeck.json");
        UpdateDecksInfo();
    }

    private void LoadOrCreateDeck(ref AllCards deck, string fileName)
    {
        string filePath = Path.Combine(Application.persistentDataPath,
fileName);
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            deck = JsonUtility.FromJson<AllCards>(json);
        }
        else
        {

            for (int i = 0; i < Math.Min(30, allCardsDeck.cards.Count); i++)
                deck.cards.Add(allCardsDeck.cards[i]);
            SaveDeck(deck, filePath);
        }
    }

    public void UpdateDecksInfo()
    {
        foreach (Card card in MyDeck.cards)
        {
            Card allCardsDeckCard = allCardsDeck.cards.Find(x => x.id ==
card.id);

            card.Title = allCardsDeckCard.Title;
            card.Description = allCardsDeckCard.Description;
            card.LogoPath = allCardsDeckCard.LogoPath;
            card.Class = allCardsDeckCard.Class;
            card.Attack = allCardsDeckCard.Attack;
            card.HP = allCardsDeckCard.HP;
            card.ManaCost = allCardsDeckCard.ManaCost;
            card.CanAttack = allCardsDeckCard.CanAttack;
            card.IsPlaced = allCardsDeckCard.IsPlaced;
            card.Spell = allCardsDeckCard.Spell;
            card.SpellTarget = allCardsDeckCard.SpellTarget;
            card.SpellValue = allCardsDeckCard.SpellValue;
            card.TimesTookDamage = allCardsDeckCard.TimesTookDamage;
            card.TimesDealedDamage = allCardsDeckCard.TimesDealedDamage;

            card.Abilities = new
List<AbilityType>(allCardsDeckCard.Abilities);
        }

        foreach (Card card in EnemyDeck.cards)
        {
            Card allCardsDeckCard = allCardsDeck.cards.Find(x => x.id ==
card.id);

            card.Title = allCardsDeckCard.Title;
            card.Description = allCardsDeckCard.Description;
            card.LogoPath = allCardsDeckCard.LogoPath;
            card.Class = allCardsDeckCard.Class;
            card.Attack = allCardsDeckCard.Attack;
            card.HP = allCardsDeckCard.HP;
            card.ManaCost = allCardsDeckCard.ManaCost;
            card.CanAttack = allCardsDeckCard.CanAttack;
            card.IsPlaced = allCardsDeckCard.IsPlaced;
            card.Spell = allCardsDeckCard.Spell;
            card.SpellTarget = allCardsDeckCard.SpellTarget;
            card.SpellValue = allCardsDeckCard.SpellValue;
            card.TimesTookDamage = allCardsDeckCard.TimesTookDamage;
            card.TimesDealedDamage = allCardsDeckCard.TimesDealedDamage;

            card.Abilities = new
List<AbilityType>(allCardsDeckCard.Abilities);
        }
    }

    public void SaveAllDecks()
    {
        SaveDeck(MyDeck, Path.Combine(Application.persistentDataPath,
"MyDeck.json"));
```

```
        SaveDeck(EnemyDeck, Path.Combine(Application.persistentDataPath,
"EnemyDeck.json"));
    }

    private void SaveDeck(AllCards deck, string filePath)
    {
        string json = JsonUtility.ToJson(deck, true);
        File.WriteAllText(filePath, json);
    }

    public void DeleteCardFromDeck(AllCards Deck, Card card)
    {
        for (int i = 0; i < Deck.cards.Count; i++)
        {
            if (card.id == Deck.cards[i].id)
            {
                Deck.cards.RemoveAt(i);
            }
        }
    }

    public void AddCardToDeck(AllCards Deck, Card card)
    {
        Deck.cards.Add(card);
    }

    public void AddMissingCards()
    {
        if (MyDeck.cards.Count < MaxDeckLen)
        {
            foreach (Card card in allCardsDeck.cards)
            {
                if (!MyDeck.cards.Contains(card))
                    AddCardToDeck(MyDeck, card);
                if (MyDeck.cards.Count >= MaxDeckLen)
                    break;
            }
        }
        if (EnemyDeck.cards.Count < MaxDeckLen)
        {
            foreach (Card card in allCardsDeck.cards)
            {
                if (!MyDeck.cards.Contains(card))
```

```
                    AddCardToDeck(EnemyDeck, card);
                if (EnemyDeck.cards.Count >= MaxDeckLen)
                    break;
            }
        }
    }


}
```

Файл AI.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
//using UnityEditor.UIElements;
using UnityEngine;
using static Card;

public class AI : MonoBehaviour
{
    GameState gameState;
    const int NumberOfSimulationsForCast = 1000;
    const int NumberOfSimulationsForSpellTarget = 1000;
    const int NumberOfSimulationsForAttackWithProvocation = 1000;
    const int NumberOfSimulationsForAttack = 1000;

    public bool CourutineIsRunning = false;
    public bool SubCourutineIsRunning = false;
    public bool SubSubCourutineIsRunning = false;
    public void MakeTurn()
    {
        StartCoroutine(EnemyTurn(GameManagerScr.Instance.EnemyHandCards));
    }

    IEnumerator EnemyTurn(List<CardController> cards)
    {
        CourutineIsRunning = true;
        yield return new WaitForSeconds(1);

        //Casting cards
        int targetindex;
        List<CardController> cardsList = cards.FindAll(x =>
GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.ManaCost);

        //int randomCount = UnityEngine.Random.Range(0, cards.Count);
        while (cardsList.Count > 0)
        {
            if (GameManagerScr.Instance.EnemyFieldCards.Count > 5 ||
                GameManagerScr.Instance.CurrentGame.Enemy.Mana == 0 ||
                GameManagerScr.Instance.EnemyHandCards.Count == 0)
                break;

            if (cardsList.Count == 0)
                break;
            int index = FindBestCardToCast(cardsList);
            if (index == -1)
                break;

            if (cardsList[index].Card.IsSpell)
            {

                if (cardsList[index].Card.SpellTarget ==
Card.TargetType.ALLY_CARD_TARGET)
                {
                    if (GameManagerScr.Instance.EnemyFieldCards.Count == 0)
                    {
                        cardsList = cards.FindAll(x =>
GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.ManaCost);
                        cardsList.RemoveAt(index);

                        continue;
                    }
                    else if (GameManagerScr.Instance.EnemyFieldCards.Count ==
1)
                        targetindex = 0;
                    else
                        targetindex = FindBestTargetForSpell(index,
GameManagerScr.Instance.EnemyFieldCards);
                    CastSpell(cardsList[index], targetindex);
                    while (SubCourutineIsRunning)
                        yield return new WaitForSeconds(0.1f);
                }
                else if (cardsList[index].Card.SpellTarget ==
Card.TargetType.ENEMY_CARD_TARGET)
```

```
                {
                    if (GameManagerScr.Instance.PlayerFieldCards.Count == 0)
                    {
                        cardsList = cards.FindAll(x =>
GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.ManaCost);
                        cardsList.RemoveAt(index);
                        continue;
                    }
                    else if (GameManagerScr.Instance.PlayerFieldCards.Count
== 1)
                        targetindex = 0;
                    else
                        targetindex = FindBestTargetForSpell(index,
GameManagerScr.Instance.PlayerFieldCards);
                    CastSpell(cardsList[index], targetindex);
                    while (SubCourutineIsRunning)
                        yield return new WaitForSeconds(0.1f);
                }
                else
                    CastSpell(cardsList[index], -1);
                while (SubCourutineIsRunning)
                    yield return new WaitForSeconds(0.1f);

                UIController.Instance.UpdateHPAndMana();
            }
            else
            {

cardsList[index].GetComponent<CardMovementScr>().MoveToField(GameManagerScr.I
nstance.EnemyField);
                yield return new WaitForSeconds(.51f);

cardsList[index].transform.SetParent(GameManagerScr.Instance.EnemyField);
                cardsList[index].OnCast();
                UIController.Instance.UpdateHPAndMana();
                cardsList = cards.FindAll(x =>
GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.ManaCost);
            }
            cardsList = cards.FindAll(x =>
GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.ManaCost);

        }
        yield return new WaitForSeconds(1);

        //Using cards

        while (GameManagerScr.Instance.EnemyFieldCards.Exists(x =>
x.Card.CanAttack))
        {
            CardController enemy, attacker;
            var activeCards =
GameManagerScr.Instance.EnemyFieldCards.FindAll(x => x.Card.CanAttack);
            bool hasProvocation =
GameManagerScr.Instance.PlayerFieldCards.Exists(x => x.Card.IsProvocation);
            if (hasProvocation)
            {
                int enemyIndex =
GameManagerScr.Instance.PlayerFieldCards.FindIndex(x =>
x.Card.IsProvocation);
                if (activeCards.Count == 1)
                    attacker = activeCards[0];
                else
                    attacker = activeCards[FindBestAttacker(enemyIndex,
activeCards)];
                enemy = GameManagerScr.Instance.PlayerFieldCards[enemyIndex];

                Debug.Log(attacker.Card.Title + " (" + attacker.Card.Attack +
"; " + attacker.Card.HP + ") ---> " +
                          enemy.Card.Title + " (" + enemy.Card.Attack + "; "
+ enemy.Card.HP + ")");


attacker.GetComponent<CardMovementScr>().MoveToTarget(enemy.transform);
                while (SubSubCourutineIsRunning)
                    yield return new WaitForSeconds(0.1f);
                GameManagerScr.Instance.CardsFight(enemy, attacker);
                attacker.Card.CanAttack = false;
            }
            else
            {
                //for (int i = 0; i < activeCards.Count; i++)
                attacker = activeCards[0];
                if (GameManagerScr.Instance.PlayerFieldCards.Count == 0)
                    targetindex = -1;
                else
```

```
                    targetindex = FindBestTargetForEntity(0,
GameManagerScr.Instance.PlayerFieldCards);
                if (targetindex == -1)
                {
                    Debug.Log(attacker.Card.Title + " (" +
attacker.Card.Attack + "; " + attacker.Card.HP + ") ---> Hero");

attacker.GetComponent<CardMovementScr>().MoveToTarget(GameManagerScr.Instance
.PlayerHero.transform);
                    while (SubSubCourutineIsRunning)
                        yield return new WaitForSeconds(0.1f);
                    GameManagerScr.Instance.DamageHero(attacker, false);
                    attacker.Card.CanAttack = false;

                }
                else
                {
                    enemy =
GameManagerScr.Instance.PlayerFieldCards[targetindex];
                    Debug.Log(attacker.Card.Title + " (" +
attacker.Card.Attack + "; " + attacker.Card.HP + ") ---> " +
                    enemy.Card.Title + " (" + enemy.Card.Attack + "; " +
enemy.Card.HP + ")");

attacker.GetComponent<CardMovementScr>().MoveToTarget(enemy.transform);
                    while (SubSubCourutineIsRunning)
                        yield return new WaitForSeconds(0.1f);
                    GameManagerScr.Instance.CardsFight(enemy, attacker);
                    attacker.Card.CanAttack = false;
                }



            }

        }

        yield return new WaitForSeconds(1);

        CourutineIsRunning = false;
        GameManagerScr.Instance.ChangeTurn();
    }
    int FindBestCardToCast(List<CardController> cards)
    {
        List<int> NumOfWins = new List<int>();
        for (int i = 0; i < cards.Count; i++)
        {
            NumOfWins.Add(0);
            for (int sim = 0; sim < NumberOfSimulationsForCast; sim++)
            {
                gameState = new GameState();
                Card card = new Card();
                card = cards[i].Card.GetDeepCopy();
                gameState.AIFieldCards.Add(card);
                gameState.SimulateGame(0);
                if (gameState.Win)
                    NumOfWins[i]++;
            }
            Debug.Log("Card " + cards[i].Card.Title + " HP: " +
cards[i].Card.HP + " has got winrate: " + NumOfWins[i] + "/ " +
NumberOfSimulationsForCast);
        }
        NumOfWins.Add(0);
        for (int sim = 0; sim < NumberOfSimulationsForCast; sim++)
        {
            gameState = new GameState();
            Card card = new Card();
            gameState.SimulateGame(0);
            if (gameState.Win)
                NumOfWins[cards.Count]++;
        }
        Debug.Log("No card has got winrate: " + NumOfWins[cards.Count] + "/ "
+ NumberOfSimulationsForCast);
        int index = 0;
        if (GameManagerScr.Instance.Difficulty == "Hard")
            index = FindBiggestElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Normal")
            index = FindAverageElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Easy")
            index = FindSmallestElementIndex(NumOfWins);

        if (index == cards.Count)
        {

            return -1;
        }
        return index;
    }

    int FindBestTargetForSpell(int cardindex, List<CardController> targets)
    {
        List<int> NumOfWins = new List<int>();
        for (int i = 0; i < targets.Count; i++)
        {
            NumOfWins.Add(0);
            for (int sim = 0; sim < NumberOfSimulationsForSpellTarget; sim++)
            {
                gameState = new GameState();
                if (gameState.AIHandCards[cardindex].SpellTarget ==
Card.TargetType.ALLY_CARD_TARGET)

gameState.CastSpellOnTarget(gameState.AIHandCards[cardindex],
gameState.AIFieldCards[i]);
                else if (gameState.AIHandCards[cardindex].SpellTarget ==
Card.TargetType.ENEMY_CARD_TARGET)

gameState.CastSpellOnTarget(gameState.AIHandCards[cardindex],
gameState.PlayerFieldCards[i]);
                gameState.CastCards(true);
                if (gameState.CheckForVictory())
                    gameState.Win = gameState.ReturnResult();
                else
                {
                    gameState.UseCards(true);
                    if (gameState.CheckForVictory())
                        gameState.Win = gameState.ReturnResult();
                    else
                        gameState.AITurn = false;
                    gameState.SimulateGame(1);
                }
                if (gameState.Win)
                    NumOfWins[i]++;
            }
        }

        if (GameManagerScr.Instance.Difficulty == "Hard")
            return FindBiggestElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Normal")
            return FindAverageElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Easy")
            return FindSmallestElementIndex(NumOfWins);
        return FindBiggestElementIndex(NumOfWins);

    }

    int FindBestTargetForEntity(int attackerIndex, List<CardController>
targets)
    {
        int index = 0;
        List<int> NumOfWins = new List<int>();
        for (int i = 0; i < targets.Count; i++)
        {
            NumOfWins.Add(0);
            for (int sim = 0; sim < NumberOfSimulationsForAttack; sim++)
            {
                gameState = new GameState();
                gameState.CardsFight(gameState.AIFieldCards.FindAll(x =>
x.CanAttack)[attackerIndex], gameState.PlayerFieldCards[i]);
                gameState.UseCards(true);
                if (gameState.CheckForVictory())
                    gameState.Win = gameState.ReturnResult();
                else
                    gameState.AITurn = false;
                gameState.SimulateGame(1);
                if (gameState.Win)
                    NumOfWins[i]++;
            }
        }
        NumOfWins.Add(0);
        for (int sim = 0; sim < NumberOfSimulationsForAttack; sim++)
        {
            gameState = new GameState();
            gameState.DamageHero(true, gameState.AIFieldCards.FindAll(x =>
x.CanAttack)[attackerIndex]);
            if (gameState.CheckForVictory())
                gameState.Win = gameState.ReturnResult();
            else
            {
                gameState.UseCards(true);
                if (gameState.CheckForVictory())
```

```
                    gameState.Win = gameState.ReturnResult();
                else
                    gameState.AITurn = false;
                gameState.SimulateGame(1);
            }
            if (gameState.Win)
                NumOfWins[targets.Count]++;
        }
        if (GameManagerScr.Instance.Difficulty == "Hard")
            index = FindBiggestElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Normal")
            index = FindAverageElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Easy")
            index = FindSmallestElementIndex(NumOfWins);

        if (index == targets.Count)
            return -1;
        return index;

    }

    int FindBestAttacker(int targetIndex, List<CardController> cards)
    {
        if (cards.Count == 0)
            return 0;
        List<int> NumOfWins = new List<int>();
        for (int i = 0; i < cards.Count; i++)
        {
            NumOfWins.Add(0);
            for (int sim = 0; sim <
NumberOfSimulationsForAttackWithProvocation; sim++)
            {
                gameState = new GameState();
                //Debug.Log(cards.Count + " --- " +
gameState.AIFieldCards.FindAll(x => x.CanAttack).Count);
                gameState.CardsFight(gameState.AIFieldCards.FindAll(x =>
x.CanAttack)[i], gameState.PlayerFieldCards[targetIndex]);
                gameState.UseCards(true);
                if (gameState.CheckForVictory())
                    gameState.Win = gameState.ReturnResult();
                else
                    gameState.AITurn = false;
                gameState.SimulateGame(1);
                if (gameState.Win)
                    NumOfWins[i]++;
            }
        }
        if (GameManagerScr.Instance.Difficulty == "Hard")
            return FindBiggestElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Normal")
            return FindAverageElementIndex(NumOfWins);
        else if (GameManagerScr.Instance.Difficulty == "Easy")
            return FindSmallestElementIndex(NumOfWins);
        return FindBiggestElementIndex(NumOfWins);
    }

    int FindBiggestElementIndex(List<int> ints)
    {
        int maxNumber = int.MinValue;
        int maxIndex = -1;
        for (int i = 0; i < ints.Count; i++)
        {
            if (ints[i] > maxNumber)
            {
                maxNumber = ints[i];
                maxIndex = i;
            }
        }
        return maxIndex;
    }

    int FindAverageElementIndex(List<int> ints)
    {
        double average = ints.Average();

        int closestIndex = -1;
        double minDifference = double.MaxValue;

        // Iterate through the list to find the element closest to the
average
        for (int i = 0; i < ints.Count; i++)
        {
            double difference = Math.Abs(ints[i] - average);
            if (difference < minDifference)
            {
                minDifference = difference;
                closestIndex = i;
            }
        }

        return closestIndex;
    }

    int FindSmallestElementIndex(List<int> ints)
    {
        int minNumber = int.MaxValue;
        int minIndex = -1;
        for (int i = 0; i < ints.Count; i++)
        {
            if (ints[i] < minNumber)
            {
                minNumber = ints[i];
                minIndex = i;
            }
        }
        return minIndex;
    }

    void CastSpell(CardController card, int targetindex)
    {
        card.Info.ShowCardInfo();
        switch (card.Card.SpellTarget)
        {
            case Card.TargetType.NO_TARGET:
                switch (card.Card.Spell)
                {
                    case Card.SpellType.HEAL_ALLY_FIELD_CARDS:
                        if (GameManagerScr.Instance.EnemyFieldCards.Count >
0)
                            StartCoroutine(CastCard(card));


                        break;

                    case Card.SpellType.DAMAGE_ENEMY_FIELD_CARDS:
                        if (GameManagerScr.Instance.EnemyFieldCards.Count >
0)
                            StartCoroutine(CastCard(card));
                        break;

                    case Card.SpellType.HEAL_ALLY_HERO:
                        StartCoroutine(CastCard(card));
                        break;

                    case Card.SpellType.DAMAGE_ENEMY_HERO:
                        StartCoroutine(CastCard(card));
                        break;
                }
                break;

            case Card.TargetType.ALLY_CARD_TARGET:
                if (GameManagerScr.Instance.EnemyFieldCards.Count > 0)
                    StartCoroutine(CastCard(card,

GameManagerScr.Instance.EnemyFieldCards[targetindex]));
                break;

            case Card.TargetType.ENEMY_CARD_TARGET:
                if (GameManagerScr.Instance.PlayerFieldCards.Count > 0)
                    StartCoroutine(CastCard(card,

GameManagerScr.Instance.PlayerFieldCards[targetindex]));

                break;
        }
    }

    IEnumerator CastCard(CardController spell, CardController target = null)
    {
        SubCourutineIsRunning = true;
        if (spell.Card.SpellTarget == Card.TargetType.NO_TARGET)
        {
            spell.Info.ShowCardInfo();

spell.GetComponent<CardMovementScr>().MoveToField(GameManagerScr.Instance.Ene
myField);
            while (SubSubCourutineIsRunning)
                yield return new WaitForSeconds(0.1f);

            spell.OnCast();
        }
        else
        {


spell.GetComponent<CardMovementScr>().MoveToTarget(target.transform);

            while (SubSubCourutineIsRunning)
                yield return new WaitForSeconds(0.1f);
            spell.Info.ShowCardInfo();

            GameManagerScr.Instance.EnemyHandCards.Remove(spell);
            GameManagerScr.Instance.EnemyFieldCards.Add(spell);
            GameManagerScr.Instance.ReduceMana(false, spell.Card.ManaCost);

            spell.Card.IsPlaced = true;

            spell.UseSpell(target);

            //yield return new WaitForSeconds(.49f);
        }

        string targetStr = target == null ? "no_target" : target.Card.Title;
        Debug.Log("AI spell cast: " + spell.Card.Title + "---> target: " +
targetStr);
        SubCourutineIsRunning = false;
    }
}

public class GameState
{
    public int AIHP, PlayerHP;
    public List<Card> AIFieldCards = new List<Card>();
    public List<Card> PlayerFieldCards = new List<Card>();
    public List<Card> AIHandCards = new List<Card>();
    public List<Card> PlayerHandCards = new List<Card>();
    public AllCards AIDeckCards;
    public AllCards PlayerDeckCards;

    public DecksManagerScr decksManager;

    Player Player, AI;
    public bool AITurn;
    public bool Win;

    public GameState()
    {
        AITurn = !GameManagerScr.Instance.PlayersTurn;

        decksManager = new DecksManagerScr();
        Player = new Player();
        Player.HP = GameManagerScr.Instance.CurrentGame.Player.HP;
        Player.Mana = Player.Manapool =
GameManagerScr.Instance.CurrentGame.Player.Manapool;

        AI = new Player();
        AI.HP = GameManagerScr.Instance.CurrentGame.Enemy.HP;
        AI.Mana = AI.Manapool =
GameManagerScr.Instance.CurrentGame.Enemy.Manapool;

        AIHandCards = new List<Card>();
        PlayerHandCards = new List<Card>();
        AIFieldCards = new List<Card>();
        PlayerFieldCards = new List<Card>();

        AIDeckCards = new AllCards();
        PlayerDeckCards = new AllCards();

        AIFieldCards =
DeepCopy(CardControllerToCards(GameManagerScr.Instance.EnemyFieldCards));
        PlayerFieldCards =
DeepCopy(CardControllerToCards(GameManagerScr.Instance.PlayerFieldCards));
        AIHandCards =
DeepCopy(CardControllerToCards(GameManagerScr.Instance.EnemyHandCards));
        PlayerHandCards =
DeepCopy(CardControllerToCards(GameManagerScr.Instance.PlayerHandCards));
        AIDeckCards.cards =
DeepCopy(GameManagerScr.Instance.decksManager.GetEnemyDeckCopy().cards);
        PlayerDeckCards.cards =
DeepCopy(GameManagerScr.Instance.decksManager.GetMyDeckCopy().cards);

        int PlayerHandCount = PlayerHandCards.Count;

        PlayerDeckCards.cards.AddRange(PlayerHandCards);
        PlayerHandCards.Clear();

        PlayerDeckCards.cards = ShuffleDeck(PlayerDeckCards.cards);
        AIDeckCards.cards = ShuffleDeck(AIDeckCards.cards);

        for (int i = 0; i < PlayerHandCount; i++)
        {
            PlayerHandCards.Add(PlayerDeckCards.cards[0]);
            PlayerDeckCards.cards.RemoveAt(0);
        }

    }



    List<Card> CardControllerToCards(List<CardController> List)
    {
        List<Card> NewList = new List<Card>();
        for (int i = 0; i < List.Count; i++)
        {
            NewList.Add(List[i].Card.GetDeepCopy());
        }
        return NewList;
    }

    List<Card> DeepCopy(List<Card> source)
    {
        List<Card> list = new List<Card>();
        for (int i = 0; i < source.Count; i++)
        {
            list.Add(source[i].GetDeepCopy());
        }
        return list;
    }

    List<Card> ShuffleDeck(List<Card> Deck)
    {
        Card temp;
        System.Random random = new System.Random();
        // Fisher-Yates shuffle
        for (int i = Deck.Count - 1; i > 0; i--)
        {
            int randomIndex = random.Next(i + 1);
```

```
            temp = Deck[i];
            Deck[i] = Deck[randomIndex];
            Deck[randomIndex] = temp;
        }
        return Deck;
    }

    public void SimulateGame(int turn)
    {
        while (true)
        {

            AITurn = !AITurn;
            if (AITurn)
            {
                if (turn != 0)
                    AI.IncreaseManapool();
                AI.RestoreRoundMana();
                foreach (Card card in AIFieldCards)
                {
                    card.CanAttack = true;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.REGENERATION_EACH_TURN))
                        card.HP += card.SpellValue;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.INCREASE_ATTACK_EACH_TURN))
                        card.Attack += card.SpellValue;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.ADDITIONAL_MANA_EACH_TURN))
                        AI.Mana += card.SpellValue;
                }
            }
            else
            {
                if (turn != 0)
                    Player.IncreaseManapool();
                Player.RestoreRoundMana();
                foreach (Card card in PlayerFieldCards)
                {
                    card.CanAttack = true;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.REGENERATION_EACH_TURN))
                        card.HP += card.SpellValue;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.INCREASE_ATTACK_EACH_TURN))
                        card.Attack += card.SpellValue;
                    if (turn != 0 && card.Abilities.Exists(x => x ==
Card.AbilityType.ADDITIONAL_MANA_EACH_TURN))
                        Player.Mana += card.SpellValue;
                }
            }
            if (turn != 0)
                CastCards(AITurn);
            if (CheckForVictory())
                break;
            UseCards(AITurn);
            if (CheckForVictory())
                break;
            turn++;
        }
        Win = ReturnResult();
    }

    public void CastCards(bool AITurn)
    {
        if (AITurn)
        {
            GiveCardToHand(AIDeckCards.cards, AIHandCards, true);
            int randomCount = UnityEngine.Random.Range(0, AIHandCards.Count);
            for (int i = 0; i < randomCount; i++)
            {
                if (AIFieldCards.Count > 5 ||
                    AI.Mana == 0 ||
                    AIHandCards.Count == 0)
                    break;

                List<Card> cardsList = AIHandCards.FindAll(x => AI.Mana >=
x.ManaCost);

                if (cardsList.Count == 0)
                    break;
                int randomIndex = UnityEngine.Random.Range(0,
cardsList.Count);
                AI.Mana -= cardsList[randomIndex].ManaCost;

                if (cardsList[randomIndex].IsSpell)
                {
                    if (cardsList[randomIndex].SpellTarget ==
Card.TargetType.NO_TARGET ||
                       (cardsList[randomIndex].SpellTarget ==
Card.TargetType.ALLY_CARD_TARGET && AIFieldCards.Count > 0) ||
                       (cardsList[randomIndex].SpellTarget ==
Card.TargetType.ENEMY_CARD_TARGET && PlayerFieldCards.Count > 0))
                        CastSpell(cardsList[randomIndex], true);
                }
                else
                {
                    CastCard(cardsList[randomIndex], true);
                }

            }
        }
        else
        {
            GiveCardToHand(PlayerDeckCards.cards, PlayerHandCards, false);
            int randomCount = UnityEngine.Random.Range(0,
PlayerHandCards.Count);
            for (int i = 0; i < randomCount; i++)
            {
                if (PlayerFieldCards.Count > 5 ||
                    Player.Mana == 0 ||
                    PlayerHandCards.Count == 0)
                    break;

                List<Card> cardsList = PlayerHandCards.FindAll(x => AI.Mana
>= x.ManaCost);

                if (cardsList.Count == 0)
                    break;

                int randomIndex = UnityEngine.Random.Range(0,
cardsList.Count);
                Player.Mana -= cardsList[randomIndex].ManaCost;
                if (cardsList[randomIndex].IsSpell)
                {
                    if (cardsList[randomIndex].SpellTarget ==
Card.TargetType.NO_TARGET ||
                       (cardsList[randomIndex].SpellTarget ==
Card.TargetType.ALLY_CARD_TARGET && PlayerFieldCards.Count > 0) ||
                       (cardsList[randomIndex].SpellTarget ==
Card.TargetType.ENEMY_CARD_TARGET && AIFieldCards.Count > 0))
                        CastSpell(cardsList[randomIndex], false);
                }
                else
                {
                    CastCard(cardsList[randomIndex], false);
                }

            }
        }
    }

    public void CastSpellOnTarget(Card spell, Card target)
    {
        AI.Mana -= spell.ManaCost;
        if (spell.SpellTarget == Card.TargetType.ALLY_CARD_TARGET)
        {
            switch (spell.Spell)
            {
                case Card.SpellType.HEAL_ALLY_CARD:
                    target.HP += spell.SpellValue;
                    break;

                case Card.SpellType.SHIELD_ON_ALLY_CARD:
                    target.Abilities.Add(Card.AbilityType.SHIELD);
                    break;

                case Card.SpellType.PROVOCATION_ON_ALLY_CARD:
                    target.Abilities.Add(Card.AbilityType.PROVOCATION);
                    break;

                case Card.SpellType.BUFF_CARD_DAMAGE:
                    target.Attack += spell.SpellValue;
                    break;
            }
        }
        else if (spell.SpellTarget == Card.TargetType.ENEMY_CARD_TARGET)
        {
            switch (spell.Spell)
            {
                case Card.SpellType.DEBUFF_CARD_DAMAGE:
                    target.Attack -= spell.SpellValue;
                    break;

                case Card.SpellType.SILENCE:
                    target.Abilities.Clear();
                    target.Abilities.Add(AbilityType.NO_ABILITY);
                    break;
            }
        }
        DestroyCard(spell);
    }

    public void UseCards(bool AITurn)
    {
        int AttackerIndex, DefenderIndex;
        List<Card> Attackers, Defenders;
        if (AITurn)
        {
            Attackers = AIFieldCards.FindAll(x => x.CanAttack);
            Defenders = PlayerFieldCards;

        }
        else
        {
            Attackers = PlayerFieldCards.FindAll(x => x.CanAttack);
            Defenders = AIFieldCards;
        }
        foreach (Card card in Attackers)
            card.TimesDealedDamage = 0;
        for (int i = 0; i < Attackers.Count; i++)
        {
            AttackerIndex = UnityEngine.Random.Range(0, Attackers.Count);
            DefenderIndex = UnityEngine.Random.Range(0, Defenders.Count);
            if (!(Defenders.Count == 0))
            {
                for (int j = 0; j < Defenders.Count; j++)
                {
                    if (Defenders[j].IsProvocation)
```

```
                        DefenderIndex = j;
                    }
                }
                if ((UnityEngine.Random.Range(0, 2) == 0 &&
!FieldHasProvocation(Defenders)) || Defenders.Count == 0)
                {
                    DamageHero(AITurn, Attackers[AttackerIndex]);
                    Attackers[AttackerIndex].TimesDealedDamage++;
                    if (CheckForVictory())
                        return;

                }
                else
                {
                    CardsFight(Attackers[AttackerIndex],
Defenders[DefenderIndex]);
                    Attackers[AttackerIndex].TimesDealedDamage++;
                }
                if (!(Attackers[AttackerIndex].Abilities.Exists(x => x ==
AbilityType.DOUBLE_ATTACK) && Attackers[AttackerIndex].TimesDealedDamage <
2))
                    Attackers.RemoveAt(AttackerIndex);
            }
        }

        public void DamageHero(bool AITurn, Card card)
        {

            if (AITurn)
                Player.HP -= card.Attack;
            else
                AI.HP -= card.Attack;
            card.CanAttack = false;
        }



        public void CardsFight(Card attacker, Card defender)
        {
            defender.GetDamage(attacker.Attack);
            attacker.GetDamage(defender.Attack);


            if (attacker.Abilities.Exists(x => x == AbilityType.EXHAUSTION))
            {
                attacker.Attack += attacker.SpellValue;
                defender.Attack -= attacker.SpellValue;
            }
            if (attacker.Abilities.Exists(x => x == AbilityType.HORDE))
            {
                attacker.Attack = attacker.HP;
            }
            if (defender.Abilities.Exists(x => x == AbilityType.HORDE))
            {
                defender.Attack = defender.HP;
            }

            attacker.CanAttack = false;

            CheckForAlive(defender);
            CheckForAlive(attacker);
        }

        bool FieldHasProvocation(List<Card> FieldCards)
        {
            for (int i = 0; i < FieldCards.Count; i++)
            {
                if (FieldCards[i].IsProvocation)
                    return true;
            }
            return false;
        }

        void GiveCardToHand(List<Card> deck, List<Card> hand, bool AI)
        {
            if ((AI && AIHandCards.Count >= 8) || (!AI && PlayerHandCards.Count
    >= 8))
                return;
            if (deck.Count == 0)
                deck = RenewDeck(AI);

            hand.Add(deck[0]);
            deck.RemoveAt(0);

        }

        public List<Card> RenewDeck(bool AI)
        {
            if (AI)
            {
                AIDeckCards.cards = new
    List<Card>(GameManagerScr.Instance.decksManager.GetEnemyDeckCopy().cards);
                AIDeckCards.cards = ShuffleDeck(AIDeckCards.cards);
                return AIDeckCards.cards;
            }
            else
            {
                PlayerDeckCards.cards = new
    List<Card>(GameManagerScr.Instance.decksManager.GetMyDeckCopy().cards);
                PlayerDeckCards.cards = ShuffleDeck(PlayerDeckCards.cards);
                return PlayerDeckCards.cards;
            }
        }


        void CastCard(Card card, bool AITurn)
        {
            if (AITurn)
            {
                foreach (Card fieldcard in AIFieldCards)
                {
                    if (fieldcard.Abilities.Exists(x => x ==
    Card.AbilityType.ALLIES_INSPIRATION))
                    {
                        card.Attack += fieldcard.SpellValue;
                    }
                }
                AIFieldCards.Add(card);
                AIHandCards.Remove(card);

            }
            else
            {
                foreach (Card fieldcard in PlayerFieldCards)
                {
                    if (fieldcard.Abilities.Exists(x => x ==
    Card.AbilityType.ALLIES_INSPIRATION))
                    {
                        card.Attack += fieldcard.SpellValue;
                    }
                }
                PlayerFieldCards.Add(card);
                PlayerHandCards.Remove(card);
            }

            if (card.HasAbility)
            {
                foreach (var ability in card.Abilities)
                {
                    switch (ability)
                    {
                        case Card.AbilityType.LEAP:
                            card.CanAttack = true;
                            break;

                        case Card.AbilityType.ALLIES_INSPIRATION:
                            if (AITurn)
                            {
                                foreach (var fieldcard in AIFieldCards)
                                {
                                    if (fieldcard.id != card.id)
                                    {
                                        fieldcard.Attack += card.SpellValue;
                                    }
                                }
                            }
                            else
                            {
                                foreach (var fieldcard in PlayerFieldCards)
                                {
                                    if (fieldcard.id != card.id)
                                    {
                                        fieldcard.Attack += card.SpellValue;
                                    }
                                }
                            }

                            break;
                    }
                }
            }
        }
```

```
    void CastSpell(Card card, bool AITurn)
    {
        int targetIndex = 0;
        if (card.SpellTarget == Card.TargetType.ALLY_CARD_TARGET && AITurn)
            targetIndex = UnityEngine.Random.Range(0, AIFieldCards.Count);
        else if (card.SpellTarget == Card.TargetType.ALLY_CARD_TARGET &&
!AITurn)
            targetIndex = UnityEngine.Random.Range(0,
PlayerFieldCards.Count);
        else if (card.SpellTarget == Card.TargetType.ENEMY_CARD_TARGET &&
AITurn)
            targetIndex = UnityEngine.Random.Range(0,
PlayerFieldCards.Count);
        else if (card.SpellTarget == Card.TargetType.ENEMY_CARD_TARGET &&
!AITurn)
            targetIndex = UnityEngine.Random.Range(0, AIFieldCards.Count);
        switch (card.Spell)
        {
            case Card.SpellType.HEAL_ALLY_FIELD_CARDS:
                var allyCards = AITurn ?
                                 new List<Card>(AIFieldCards) :
                                 new List<Card>(PlayerFieldCards);
                foreach (Card fieldcard in allyCards)
                    fieldcard.HP += card.SpellValue;
                break;

            case Card.SpellType.DAMAGE_ENEMY_FIELD_CARDS:
                var enemyCards = AITurn ?
                                 new List<Card>(PlayerFieldCards) :
                                 new List<Card>(AIFieldCards);
                foreach (Card fieldcard in enemyCards)
                    GiveDamageTo(fieldcard, card.SpellValue);
                break;
            case Card.SpellType.HEAL_ALLY_HERO:
                if (AITurn)
                    AI.HP += card.SpellValue;
                else
                    Player.HP += card.SpellValue;
                break;
            case Card.SpellType.DAMAGE_ENEMY_HERO:
                if (AITurn)
                    Player.HP -= card.SpellValue;
                else
```

```
                    AI.HP -= card.SpellValue;
                break;

            case Card.SpellType.HEAL_ALLY_CARD:
                if (AITurn)
                    AIFieldCards[targetIndex].HP += card.SpellValue;
                else
                    PlayerFieldCards[targetIndex].HP += card.SpellValue;
                break;

            case Card.SpellType.SHIELD_ON_ALLY_CARD:
                if (AITurn)
                {
                    if (!AIFieldCards[targetIndex].Abilities.Exists(x => x ==
Card.AbilityType.SHIELD))

AIFieldCards[targetIndex].Abilities.Add(Card.AbilityType.SHIELD);
                }
                else
                {
                    if (!PlayerFieldCards[targetIndex].Abilities.Exists(x =>
x == Card.AbilityType.SHIELD))

PlayerFieldCards[targetIndex].Abilities.Add(Card.AbilityType.SHIELD);
                }
                break;

            case Card.SpellType.PROVOCATION_ON_ALLY_CARD:
                if (AITurn)
                {
                    if (!AIFieldCards[targetIndex].Abilities.Exists(x => x ==
Card.AbilityType.PROVOCATION))

AIFieldCards[targetIndex].Abilities.Add(Card.AbilityType.PROVOCATION);
                }
                else
                {
                    if (!PlayerFieldCards[targetIndex].Abilities.Exists(x =>
x == Card.AbilityType.PROVOCATION))

PlayerFieldCards[targetIndex].Abilities.Add(Card.AbilityType.PROVOCATION);
                }
                break;
            case Card.SpellType.BUFF_CARD_DAMAGE:
                if (AITurn)
                {
                    AIFieldCards[targetIndex].Attack += card.SpellValue;
                }
                else
                {
                    PlayerFieldCards[targetIndex].Attack += card.SpellValue;
                }
                break;

            case Card.SpellType.DEBUFF_CARD_DAMAGE:

                if (AITurn)
                {
                    PlayerFieldCards[targetIndex].Attack =
Mathf.Clamp(PlayerFieldCards[targetIndex].Attack - card.SpellValue, 0,
int.MaxValue);
                }
                else
                {
                    AIFieldCards[targetIndex].Attack =
Mathf.Clamp(AIFieldCards[targetIndex].Attack - card.SpellValue, 0,
int.MaxValue);
                }
                break;

            case Card.SpellType.SILENCE:
                if (AITurn)
                {
                    PlayerFieldCards[targetIndex].Abilities.Clear();

PlayerFieldCards[targetIndex].Abilities.Add(AbilityType.NO_ABILITY);
                }
                else
                {
                    AIFieldCards[targetIndex].Abilities.Clear();

AIFieldCards[targetIndex].Abilities.Add(AbilityType.NO_ABILITY);
                }
                break;
```

```
        case Card.SpellType.KILL_ALL:
            while (AIFieldCards.Count != 0)
                DestroyCard(AIFieldCards[0]);
            while (PlayerFieldCards.Count != 0)
                DestroyCard(PlayerFieldCards[0]);
            break;
    }

    DestroyCard(card);
}

void GiveDamageTo(Card card, int damage)
{
    card.GetDamage(damage);
    CheckForAlive(card);
}

void CheckForAlive(Card card)
{
    if (!card.IsAlive())
    {
        DestroyCard(card);
    }
}

void DestroyCard(Card card)
{
    RemoveCardFromList(card, AIHandCards);
    RemoveCardFromList(card, AIFieldCards);
    RemoveCardFromList(card, PlayerHandCards);
    RemoveCardFromList(card, PlayerFieldCards);
}

void RemoveCardFromList(Card card, List<Card> list)
{
    if (list.Exists(x => x == card))
        list.Remove(card);
}

public bool CheckForVictory()
{
    if (Player.HP <= 0 || AI.HP <= 0)
        return true;
        return false;
    }

    public bool ReturnResult()
    {
        if (Player.HP <= 0)
            return true;
        else
            return false;
    }

}
```

Файл AttackedCard.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class AttackedCard : MonoBehaviour, IDropHandler
{
    public void OnDrop(PointerEventData eventData)
    {

        if (!GameManagerScr.Instance.PlayersTurn)
            return;
        Debug.Log("OnDrop Called");
        CardController attacker =
eventData.pointerDrag.GetComponent<CardController>(),
                       defender = GetComponent<CardController>();

        if (attacker &&
            attacker.Card.CanAttack &&
            defender.Card.IsPlaced)
        {
            if (GameManagerScr.Instance.EnemyFieldCards.Exists(x =>
x.Card.IsProvocation) &&
                !defender.Card.IsProvocation)
                return;
            if (attacker.IsPlayerCard)
                attacker.Info.PaintWhite();

            GameManagerScr.Instance.CardsFight(attacker, defender);
        }
    }
}
```

Файл AttackedHero.cs

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class AttackedHero : MonoBehaviour, IDropHandler
{
    public enum HeroType
    {
        ENEMY,
        PLAYER
    }
    public HeroType Type;
    public Color NormalColor, TargetColor;

    public void OnDrop(PointerEventData eventData)
    {
        if (!GameManagerScr.Instance.PlayersTurn)
            return;

        CardController card = 
eventData.pointerDrag.GetComponent<CardController>();

        if (card &&
           card.Card.CanAttack &&
           Type == HeroType.ENEMY &&
           !GameManagerScr.Instance.EnemyFieldCards.Exists(x => 
x.Card.IsProvocation))
        {
            GameManagerScr.Instance.DamageHero(card, true);
        }
    }

    public void HighlightAsTarget(bool highlight)
    {
        GetComponent<Image>().color = highlight ? TargetColor : NormalColor;
    }
}
```

Файл CardAbility.cs:

```
using UnityEngine;

public class CardAbility : MonoBehaviour
{
    public CardController CC;
    public GameObject Shield, Provocation;

    public void OnCast()
    {
        foreach (var ability in CC.Card.Abilities)
        {
            switch (ability)
            {
                case Card.AbilityType.LEAP:
                    CC.Card.CanAttack = true;
                    if (CC.IsPlayerCard)
                        CC.Info.HighliteUsableCard();
                    break;

                case Card.AbilityType.SHIELD:
                    Shield.SetActive(true);
                    break;

                case Card.AbilityType.PROVOCATION:
                    Provocation.SetActive(true);
                    break;

                case Card.AbilityType.ALLIES_INSPIRATION:
                    if (CC.IsPlayerCard)
                    {
                        foreach (var card in CC.gameManager.PlayerFieldCards)
                        {
                            if (card.Card.id != CC.Card.id)
                            {
                                card.Card.Attack += CC.Card.SpellValue;
                                card.Info.RefreshData();
                            }
                        }
                    }
                    else
                    {
                        foreach (var card in CC.gameManager.EnemyFieldCards)
                        {
                            if (card.Card.id != CC.Card.id)
                            {
                                card.Card.Attack += CC.Card.SpellValue;
                                card.Info.RefreshData();
                            }
                        }
                    }

                    break;
            }
        }
    }

    public void OnDamageDeal(CardController defender = null)
    {
        foreach (var ability in CC.Card.Abilities)
        {
            switch (ability)
            {
                case Card.AbilityType.DOUBLE_ATTACK:
                    if (CC.Card.TimesDealedDamage == 1)
                    {
                        CC.Card.CanAttack = true;
                        if (CC.IsPlayerCard)
                            CC.Info.HighliteUsableCard();
                    }
                    break;

                case Card.AbilityType.EXHAUSTION:
                    if (defender != null && defender.Card.Attack > 0)
                    {
                        CC.Card.Attack += CC.Card.SpellValue;
                        CC.Info.RefreshData();
                        defender.Card.Attack =
Mathf.Clamp(defender.Card.Attack - CC.Card.SpellValue, 0, int.MaxValue);
                        defender.Info.RefreshData();
                    }
                    break;



            }
        }
    }

    public void OnDamageTake(CardController attacker = null)
    {
        Shield.SetActive(false);

        foreach (var ability in CC.Card.Abilities)
        {
            switch (ability)
            {
                case Card.AbilityType.SHIELD:
                    Shield.SetActive(true);
                    break;



                case Card.AbilityType.HORDE:
                    CC.Card.Attack = CC.Card.HP;
                    CC.Info.RefreshData();
                    break;
            }
        }
    }

    public void OnNewTurn()
    {

        CC.Card.TimesDealedDamage = 0;

        foreach (var ability in CC.Card.Abilities)
        {
            switch (ability)
            {
                case Card.AbilityType.REGENERATION_EACH_TURN:
                    CC.Card.HP += CC.Card.SpellValue;
                    CC.Info.RefreshData();
                    break;



                case Card.AbilityType.INCREASE_ATTACK_EACH_TURN:
                    CC.Card.Attack += CC.Card.SpellValue;
                    CC.Info.RefreshData();
                    break;
                case Card.AbilityType.ADDITIONAL_MANA_EACH_TURN:
                    if (CC.IsPlayerCard &&
CC.gameManager.CurrentGame.Player.Mana <
CC.gameManager.CurrentGame.Player.GetMaxManapool())
                        CC.gameManager.CurrentGame.Player.Mana +=
CC.Card.SpellValue;
                    else if (!CC.IsPlayerCard &&
CC.gameManager.CurrentGame.Enemy.Mana <
CC.gameManager.CurrentGame.Enemy.GetMaxManapool())
                        CC.gameManager.CurrentGame.Enemy.Mana +=
CC.Card.SpellValue;
                    UIController.Instance.UpdateHPAndMana();
                    break;

                case Card.AbilityType.ALLIES_INSPIRATION:
                    if (CC.IsPlayerCard)
                    {
                        foreach (var card in CC.gameManager.PlayerFieldCards)
                        {
                            if (card.Card.id != CC.Card.id)
                            {
                                Card OriginalCard =
CC.gameManager.decksManager.GetMyDeck().cards.Find(Card => Card.id ==
card.Card.id);
                                if (card.Card.Attack == OriginalCard.Attack)
                                {
                                    card.Card.Attack += CC.Card.SpellValue;
                                    card.Info.RefreshData();
                                }
                            }
                        }
                    }
                    else
                    {
                        foreach (var card in CC.gameManager.EnemyFieldCards)
                        {
                            if (card.Card.id != CC.Card.id)
                            {
                                Card OriginalCard =
CC.gameManager.decksManager.GetMyDeck().cards.Find(Card => Card.id ==
card.Card.id);
                                if (card.Card.Attack == OriginalCard.Attack)
                                {
                                    card.Card.Attack++;
                                    card.Info.RefreshData();
                                }
                            }
                        }
                    }

                    break;

                case Card.AbilityType.HORDE:
                    if (CC.Card.Attack > CC.Card.HP)
                        CC.Card.HP = CC.Card.Attack;
                    else
                        CC.Card.Attack = CC.Card.HP;

                    CC.Info.RefreshData();
                    break;
            }
        }
    }
}
```

Файл CardController.cs

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

public class CardController : MonoBehaviour
{
    public Card Card;

    public bool IsPlayerCard;

    public CardInfoScript Info;
    public CardMovementScr Movement;

    public GameManagerScr gameManager;
    public CardAbility Ability;

    public void Init(Card card, bool isPlayerCard)
    {
        Card = card;
        gameManager = GameManagerScr.Instance;
        IsPlayerCard = isPlayerCard;

        if (isPlayerCard)
        {
            Info.ShowCardInfo();
            GetComponent<AttackedCard>().enabled = false;
        }
        else
            Info.HideCardInfo();
    }

    public void OnCast()
    {
        if (Card.IsSpell && Card.SpellTarget != Card.TargetType.NO_TARGET)
            return;
        if (IsPlayerCard)
        {
            gameManager.PlayerHandCards.Remove(this);
            gameManager.PlayerFieldCards.Add(this);
            gameManager.ReduceMana(true, Card.ManaCost);
            gameManager.CheckCardForManaAvailability();
        }
        else
        {
            gameManager.EnemyHandCards.Remove(this);
            gameManager.EnemyFieldCards.Add(this);
            gameManager.ReduceMana(false, Card.ManaCost);
            Info.ShowCardInfo();
        }
        Card.IsPlaced = true;

        if (Card.HasAbility)
            Ability.OnCast();

        if (Card.IsSpell)
            UseSpell(null);
        UIController.Instance.UpdateHPAndMana();
    }

    public void OnTakeDamage(CardController attacker = null)
    {
        CheckForAlive();
        Ability.OnDamageTake(attacker);
    }

    public void OnDamageDeal(CardController defender = null)
    {
        Card.TimesDealedDamage++;
        Card.CanAttack = false;
        Info.PaintWhite();

        if (Card.HasAbility)
            Ability.OnDamageDeal(defender);
    }

    public void UseSpell(CardController target)
    {
        switch (Card.Spell)
        {
            case Card.SpellType.HEAL_ALLY_FIELD_CARDS:
                var allyCards = IsPlayerCard ?
                                gameManager.PlayerFieldCards :
                                gameManager.EnemyFieldCards;

                foreach (var card in allyCards)
                {
                    card.Card.HP += Card.SpellValue;
                    card.Info.RefreshData();
                }
                break;

            case Card.SpellType.DAMAGE_ENEMY_FIELD_CARDS:
                var enemyCards = IsPlayerCard ?
                                 new
List<CardController>(gameManager.EnemyFieldCards) :
                                 new
List<CardController>(gameManager.PlayerFieldCards);
                foreach (var card in enemyCards)
                    GiveDamageTo(card, Card.SpellValue);
                break;

            case Card.SpellType.HEAL_ALLY_HERO:
                if (IsPlayerCard)
                    gameManager.CurrentGame.Player.HP += Card.SpellValue;
                else
```

```
                    gameManager.CurrentGame.Enemy.HP += Card.SpellValue;
                UIController.Instance.UpdateHPAndMana();
                break;

            case Card.SpellType.DAMAGE_ENEMY_HERO:
                if (IsPlayerCard)
                    gameManager.CurrentGame.Enemy.HP -= Card.SpellValue;
                else
                    gameManager.CurrentGame.Player.HP -= Card.SpellValue;
                UIController.Instance.UpdateHPAndMana();
                gameManager.CheckForVictory();
                break;

            case Card.SpellType.HEAL_ALLY_CARD:
                target.Card.HP += Card.SpellValue;
                break;

            case Card.SpellType.SHIELD_ON_ALLY_CARD:
                if (!target.Card.Abilities.Exists(x => x ==
Card.AbilityType.SHIELD))
                    target.Card.Abilities.Add(Card.AbilityType.SHIELD);
                break;

            case Card.SpellType.PROVOCATION_ON_ALLY_CARD:
                if (!target.Card.Abilities.Exists(x => x ==
Card.AbilityType.PROVOCATION))
                    target.Card.Abilities.Add(Card.AbilityType.PROVOCATION);
                break;

            case Card.SpellType.BUFF_CARD_DAMAGE:
                target.Card.Attack += Card.SpellValue;
                break;

            case Card.SpellType.DEBUFF_CARD_DAMAGE:
                target.Card.Attack = Mathf.Clamp(target.Card.Attack -
Card.SpellValue, 0, int.MaxValue);
                break;

            case Card.SpellType.SILENCE:
                target.Card.Abilities.Clear();
                target.Card.Abilities.Add(AbilityType.NO_ABILITY);
                target.Card.Description = "";
                target.Info.ShowCardInfo();
                target.Ability.Provocation.SetActive(false);
                target.Ability.Shield.SetActive(false);
                break;

            case Card.SpellType.KILL_ALL:
                while (gameManager.PlayerFieldCards.Count != 0)
                    gameManager.PlayerFieldCards[0].DestroyCard();
                while (gameManager.EnemyFieldCards.Count != 0)
                    gameManager.EnemyFieldCards[0].DestroyCard();
                break;

        }

        if (target != null)
        {
            target.Ability.OnCast();
            target.CheckForAlive();
        }
        DestroyCard();
    }

    void GiveDamageTo(CardController card, int damage)
    {
        card.Card.GetDamage(damage);
        card.CheckForAlive();
        card.OnTakeDamage();
    }

    public void CheckForAlive()
    {
        if (Card.IsAlive())
            Info.RefreshData();
        else
            DestroyCard();
    }

    void DestroyCard()
    {
        Movement.OnEndDrag(null);

        RemoveCardFromList(gameManager.EnemyFieldCards);
        RemoveCardFromList(gameManager.EnemyHandCards);
        RemoveCardFromList(gameManager.PlayerFieldCards);
```

```
        RemoveCardFromList(gameManager.PlayerHandCards);

        Destroy(gameObject);
    }

    void RemoveCardFromList(List<CardController> list)
    {
        if (list.Exists(x => x == this))
            list.Remove(this);
    }
}
```

Файл CardInfoScript.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;
//using UnityEngine.WSA;

public class CardInfoScript : MonoBehaviour
{
    public CardController CC;

    public Image card_BG;
    public Image title_BG;
    public Image descr_BG;
    //public Card SelfCard;
    public Image Logo;
    public Image ClassLogo;
    public Sprite EntityClassLogo;
    public Sprite SpellClassLogo;
    public TextMeshProUGUI Title;
    public TextMeshProUGUI Description;
    public TextMeshProUGUI ManaCost;
    public TextMeshProUGUI HP;
    public TextMeshProUGUI Attack;
    public GameObject HideObj;
    public GameObject ManaCostIndicator;
    public GameObject HPIndicator;
    public GameObject AttackIndicator;
    //Sprite CardLogo;
    //public bool IsPlayer;

    public void HideCardInfo()
    {
        HideObj.SetActive(true);
        ManaCostIndicator.SetActive(false);
        HPIndicator.SetActive(false);
        //ShowCardInfo();
    }

    public void ShowCardInfo()
    {
        //IsPlayer = isPlayer;
        HideObj.SetActive(false);
        card_BG.gameObject.SetActive(true);
        ManaCostIndicator.SetActive(true);
        HPIndicator.SetActive(true);
        //SelfCard = card;

        Logo.sprite = Resources.Load<Sprite>(CC.Card.LogoPath);
        Logo.preserveAspect = true;
        Title.text = CC.Card.Title;
        Description.text = CC.Card.Description;
        ManaCost.text = CC.Card.ManaCost.ToString();
        HP.text = CC.Card.HP.ToString();
        Attack.text = CC.Card.Attack.ToString();
        if (card_BG != null)
        {
            card_BG.color = UnityEngine.Color.white;
        }
        if (title_BG != null)
        {
            title_BG.color = UnityEngine.Color.white;

        }
        if (descr_BG != null)
        {
            descr_BG.color = UnityEngine.Color.white;
        }

        if (CC.Card.Class == Card.CardClass.ENTITY || CC.Card.Class ==
Card.CardClass.ENTITY_WITH_ABILITY)
        {
            ClassLogo.sprite = EntityClassLogo;
        }
        else if (CC.Card.Class == Card.CardClass.SPELL)
        {
            ClassLogo.sprite = SpellClassLogo;
        }

        if (CC.Card.IsSpell)
        {
            HPIndicator.SetActive(false);
            AttackIndicator.SetActive(false);
        }

    }

    public void RefreshData()
    {
        Attack.text = CC.Card.Attack.ToString();
        HP.text = CC.Card.HP.ToString();
        ManaCost.text = CC.Card.ManaCost.ToString();
    }

    public void PaintGreen()
    {
        float red = 13f / 255f;
        float green = 142f / 255f;
        float blue = 0f / 255f;
        float alpha = 1f;

        card_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        title_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        descr_BG.color = new UnityEngine.Color(red, green, blue, alpha);



    }

    public void PaintWhite()
    {
        card_BG.color = UnityEngine.Color.white;
        title_BG.color = UnityEngine.Color.white;
        descr_BG.color = UnityEngine.Color.white;
    }

    public void PaintAnother(UnityEngine.Color color)
    {
        card_BG.color = color;
```

```
        title_BG.color = color;
        descr_BG.color = color;
    }

    public void HighliteUsableCard()
    {
        if (card_BG == null)
            return;
        float red = 134f / 255f;
        float green = 47f / 255f;
        float blue = 255f / 255f;
        float alpha = 1f;

        card_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        title_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        descr_BG.color = new UnityEngine.Color(red, green, blue, alpha);
    }

    public void HighlightManaAvaliability(int currentMana)
    {
        GetComponent<CanvasGroup>().alpha = currentMana >= CC.Card.ManaCost ?
1 : .75f;

    }

    public void HighlightAsTarget(bool highlight)
    {
        if (card_BG == null)
            return;
        if (!highlight)
            PaintWhite();
        else
        {
            float red = 255f / 255f;
            float green = 127f / 255f;
            float blue = 129f / 255f;
            float alpha = 1f;

            card_BG.color = new UnityEngine.Color(red, green, blue, alpha);
            title_BG.color = new UnityEngine.Color(red, green, blue, alpha);
            descr_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        }
    }
    public void HighlightAsSpellTarget(bool highlight)
    {
        if (card_BG == null)
            return;
        if (!highlight)
            if (CC.Card.CanAttack)
                HighliteUsableCard();
            else
                PaintWhite();
        else
        {
            float red = 66f / 255f;
            float green = 45f / 255f;
            float blue = 255f / 255f;
            float alpha = 1f;

            card_BG.color = new UnityEngine.Color(red, green, blue, alpha);
            title_BG.color = new UnityEngine.Color(red, green, blue, alpha);
            descr_BG.color = new UnityEngine.Color(red, green, blue, alpha);
        }
    }

}
```

Файл CardMovementScr.cs

```
using DG.Tweening;
using System.Collections;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class CardMovementScr : MonoBehaviour, IBeginDragHandler,
IDragHandler, IEndDragHandler
{
    public CardController CC;

    Camera MainCamera;
    Vector3 offset;
    public Transform DefaultParent, DefaultTempCardParent;
    GameObject TempCardGO;
    public bool IsDraggable;
    int startID;
    void Awake()
    {
        MainCamera = Camera.allCameras[0];
        TempCardGO = GameObject.Find("TempCardGO");
    }
    public void OnBeginDrag(PointerEventData eventData)
    {
        offset = transform.position - 
MainCamera.ScreenToWorldPoint(eventData.position);
        DefaultParent = DefaultTempCardParent = transform.parent;

        IsDraggable = GameManagerScr.Instance.PlayersTurn &&
                      (
                      (DefaultParent.GetComponent<DropPlaceScr>().Type == 
FieldType.SELF_HAND &&
                      GameManagerScr.Instance.CurrentGame.Player.Mana >= 
CC.Card.ManaCost) ||
                      (DefaultParent.GetComponent<DropPlaceScr>().Type == 
FieldType.SELF_FIELD &&
                      CC.Card.CanAttack)
                      );

        if (!IsDraggable)
            return;

        startID = transform.GetSiblingIndex();

        if (CC.Card.IsSpell || CC.Card.CanAttack)
            GameManagerScr.Instance.HightLightTargets(CC, true);

        TempCardGO.transform.SetParent(DefaultParent);
        TempCardGO.transform.SetSiblingIndex(transform.GetSiblingIndex());

        transform.SetParent(DefaultParent.parent);

        GetComponent<CanvasGroup>().blocksRaycasts = false;
    }

    public void OnDrag(PointerEventData eventData)
    {
        if (!IsDraggable)
```

```
            return;

        Vector3 newPos = MainCamera.ScreenToWorldPoint(eventData.position);
        transform.position = newPos + offset;

        if (!CC.Card.IsSpell)
        {

            if (TempCardGO.transform.parent != DefaultTempCardParent)
                TempCardGO.transform.SetParent(DefaultTempCardParent);

            if (DefaultParent.GetComponent<DropPlaceScr>().Type !=
FieldType.SELF_FIELD)
                CheckPosition();
        }
    }

    public void OnEndDrag(PointerEventData eventData)
    {

        if (!IsDraggable)
            return;

        GameManagerScr.Instance.HightLightTargets(CC, false);

        transform.SetParent(DefaultParent);
        GetComponent<CanvasGroup>().blocksRaycasts = true;

        transform.SetSiblingIndex(TempCardGO.transform.GetSiblingIndex());
        TempCardGO.transform.SetParent(GameObject.Find("Canvas").transform);
        TempCardGO.transform.localPosition = new Vector3(2362, 0);
    }

    void CheckPosition()
    {
        int newIndex = DefaultTempCardParent.childCount;
        for (int i = 0; i < DefaultTempCardParent.childCount; i++)
        {
            if (transform.position.x <
DefaultTempCardParent.GetChild(i).position.x)
            {
                newIndex = i;
                if (TempCardGO.transform.GetSiblingIndex() < newIndex)
                {
                    newIndex--;
                }
                break;
            }
        }

        if (TempCardGO.transform.parent == DefaultParent)
            newIndex = startID;

        TempCardGO.transform.SetSiblingIndex(newIndex);
    }

    public void MoveToField(Transform field)
    {
        transform.SetParent(GameObject.Find("Canvas").transform);
        transform.DOMove(field.position, .5f).SetEase(Ease.InOutSine);

        HorizontalLayoutGroup layout =
transform.parent.GetComponent<HorizontalLayoutGroup>();
        if (layout != null)
        {
            layout.enabled = false;
            layout.enabled = true;
        }

        //RebuildLayout();
    }

    public void MoveToTarget(Transform target)
    {
        StartCoroutine(MoveToTargetCor(target));

        //RebuildLayout();
    }

    IEnumerator MoveToTargetCor(Transform target)
    {
        GameManagerScr.Instance.EnemyAI.SubSubCourutineIsRunning = true;

        Vector3 pos = transform.position;
        Transform parent = transform.parent;
        int index = transform.GetSiblingIndex();
        HorizontalLayoutGroup layout =
transform.parent.GetComponent<HorizontalLayoutGroup>();
        if (layout != null) layout.enabled = false;

        transform.SetParent(GameObject.Find("Canvas").transform);

        // Начало анимации с плавным стартом и завершением
        Tween moveTween = transform.DOMove(target.position,
.5f).SetEase(Ease.InOutSine);

        // Ожидание завершения анимации
        yield return moveTween.WaitForCompletion();

        // Возможно, вам захочется добавить небольшую паузу здесь
        yield return new WaitForSeconds(0.5f);

        // Обратное перемещение
        moveTween = transform.DOMove(pos, .5f).SetEase(Ease.InOutSine);

        // Ожидание завершения обратного перемещения
        yield return moveTween.WaitForCompletion();

        // Восстановление исходной иерархии
        transform.SetParent(parent);
        transform.SetSiblingIndex(index);

        if (layout != null) layout.enabled = true;

        GameManagerScr.Instance.EnemyAI.SubSubCourutineIsRunning = false;
    }


}
```

Файл DropPlaceScr.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public enum FieldType
{
    SELF_HAND, SELF_FIELD,
    ENEMY_HAND, ENEMY_FIELD
}

public class DropPlaceScr : MonoBehaviour, IDropHandler,
IPointerEnterHandler, IPointerExitHandler
{
    public FieldType Type;
    public void OnDrop(PointerEventData eventData)
    {

        if (Type != FieldType.SELF_FIELD)
        {
            return;
        }
        CardController card =
eventData.pointerDrag.GetComponent<CardController>();

        if (card &&
            GameManagerScr.Instance.PlayersTurn &&
            GameManagerScr.Instance.CurrentGame.Player.Mana >=
card.Card.ManaCost &&
            !card.Card.IsPlaced)
        {
            if (!card.Card.IsSpell)
                card.Movement.DefaultParent = transform;
            card.OnCast();
        }
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (eventData.pointerDrag == null || Type == FieldType.ENEMY_FIELD ||
Type == FieldType.ENEMY_HAND ||
            Type == FieldType.ENEMY_HAND || Type == FieldType.SELF_HAND)
            return;

        CardMovementScr card =
eventData.pointerDrag.GetComponent<CardMovementScr>();

        if (card)
        {
            card.DefaultTempCardParent = transform;
        }
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (eventData.pointerDrag == null)
            return;
        CardMovementScr card =
eventData.pointerDrag.GetComponent<CardMovementScr>();

        if (card && card.DefaultTempCardParent == transform)
        {
            card.DefaultTempCardParent = card.DefaultParent;
        }
    }

}
```

Файл GameManagerScr.cs

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

public class Game : MonoBehaviour
{
    public Player Player, Enemy;
    public DecksManagerScr DecksManager;
    public List<Card> EnemyDeck, PlayerDeck;
    public int StarterCardsNum = 4;
    public GameSettings Settings;

    public Game(DecksManagerScr decksManager)
    {
        DecksManager = decksManager;

        EnemyDeck = new List<Card>(DecksManager.GetEnemyDeckCopy().cards);
        PlayerDeck = new List<Card>(DecksManager.GetMyDeckCopy().cards);
        List<Card> ShuffledDeck = ShuffleDeck(EnemyDeck);
        EnemyDeck = ShuffledDeck;
        ShuffledDeck = ShuffleDeck(PlayerDeck);
        PlayerDeck = ShuffledDeck;
        Player = new Player();
        Enemy = new Player();

        Settings = new GameSettings();
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            Settings = JsonUtility.FromJson<GameSettings>(json);
        }
        else
        {
            Settings.soundVolume = .5f;
            Settings.timer = 120;
            Settings.timerIsOn = true;
            Settings.difficulty = "Normal";
        }
    }

    public List<Card> ShuffleDeck(List<Card> Deck)
    {
        Card temp;
        System.Random random = new System.Random();
        // Fisher-Yates shuffle
        for (int i = Deck.Count - 1; i > 0; i--)
        {
            int randomIndex = random.Next(i + 1);

            temp = Deck[i];
            Deck[i] = Deck[randomIndex];
            Deck[randomIndex] = temp;
        }
        return Deck;
    }
}

public class GameManagerScr : MonoBehaviour
{
    public static GameManagerScr Instance;

    public Game CurrentGame;
    public Transform EnemyHand, PlayerHand,
                  EnemyField, PlayerField;
    public GameObject CardPref;
    public DecksManagerScr decksManager;
    public int Turn = 1, TurnTime, OriginalTurnTime;
    public bool TimerIsOn, PlayerIsFirst, PlayersTurn;

    public string Difficulty;

    public AttackedHero EnemyHero, PlayerHero;
    public AI EnemyAI;
    public List<CardController> PlayerHandCards = new List<CardController>(),
                                PlayerFieldCards = new
List<CardController>(),
                                EnemyHandCards = new List<CardController>(),
                                EnemyFieldCards = new List<CardController>();

    public GameSettings Settings = new GameSettings();

    public void Awake()
    {
        string filePath = Path.Combine(Application.persistentDataPath,
"Settings.json");
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            Settings = JsonUtility.FromJson<GameSettings>(json);
        }
        else
        {
            Settings.soundVolume = .5f;
            Settings.timer = 120;
            Settings.timerIsOn = true;
            Settings.difficulty = "Normal";
        }
        AudioListener.volume = Settings.soundVolume;

        if (Instance == null)
            Instance = this;
    }

    void Start()
    {
        StartGame();
    }

    public void BackToMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu_Scene");
    }

    public void PauseGame()
    {
        UIController.Instance.PauseGame();
    }

    public void ResumeGame()
    {
        UIController.Instance.ResumeGame();
    }

    public void RestartGame()
    {
        StopAllCoroutines();

        foreach (var card in PlayerHandCards)
            Destroy(card.gameObject);
        foreach (var card in PlayerFieldCards)
            Destroy(card.gameObject);
        foreach (var card in EnemyHandCards)
            Destroy(card.gameObject);
        foreach (var card in EnemyFieldCards)
            Destroy(card.gameObject);

        PlayerHandCards.Clear();
        PlayerFieldCards.Clear();
        EnemyHandCards.Clear();
        EnemyFieldCards.Clear();

        UIController.Instance.pausePanel.SetActive(false);
        UIController.Instance.ResumeGame();

        StartGame();
    }

    void StartGame()
```

```
{
    Time.timeScale = 1f;

    decksManager = GetComponent<DecksManagerScr>();
    CurrentGame = new Game(decksManager);

    OriginalTurnTime = CurrentGame.Settings.timer;
    TimerIsOn = CurrentGame.Settings.timerIsOn;
    Difficulty = CurrentGame.Settings.difficulty;

    UIController.Instance.EnableTurnTime(TimerIsOn);
    PlayerIsFirst = FlipCoin();
    PlayersTurn = PlayerIsFirst;

    UIController.Instance.EnableTurnTime(TimerIsOn);

    GiveHandCards(CurrentGame.EnemyDeck, EnemyHand, false);
    GiveHandCards(CurrentGame.PlayerDeck, PlayerHand, true);

    UIController.Instance.WhoseTurnUpdate();
    UIController.Instance.EnableTurnBtn();



    if (PlayersTurn)
        GiveCardToHand(CurrentGame.PlayerDeck, PlayerHand, true);
    else
        GiveCardToHand(CurrentGame.EnemyDeck, EnemyHand, false);

    Turn = 0;

    CurrentGame.Player.Mana = CurrentGame.Player.Manapool = 1;
    CurrentGame.Enemy.Mana = CurrentGame.Enemy.Manapool = 1;

    UIController.Instance.UpdateHPAndMana();

    UIController.Instance.StartGame();

    StartCoroutine(TurnFunc());
}

void GiveHandCards(List<Card> deck, Transform hand, bool player)
{
    int i = 0;
```

```
        while (i++ < CurrentGame.StarterCardsNum)
        {
            GiveCardToHand(deck, hand, player);
        }
    }
    void GiveCardToHand(List<Card> deck, Transform hand, bool player)
    {
        if ((player && PlayerHandCards.Count >= 8) || (!player &&
EnemyHandCards.Count >= 8))
            return;
        if (deck.Count == 0)
            return;

        CreateCardPref(deck[0], hand);

        deck.RemoveAt(0);

    }

    void CreateCardPref(Card card, Transform hand)
    {
        GameObject cardGO = Instantiate(CardPref, hand, false);
        cardGO.SetActive(true);
        CardController cardC = cardGO.GetComponent<CardController>();

        cardC.Init(card, hand == PlayerHand);
        if (cardC.IsPlayerCard)
            PlayerHandCards.Add(cardC);
        else

            EnemyHandCards.Add(cardC);
    }

    IEnumerator TurnFunc()
    {
        foreach (var card in PlayerFieldCards)
            card.Info.PaintWhite();

        if (TimerIsOn)
        {
            TurnTime = OriginalTurnTime;
            UIController.Instance.UpdateTurnTime(TurnTime);
        }
        else
            TurnTime = int.MaxValue;

        CheckCardForManaAvailability();

        if (PlayersTurn)
        {
            foreach (var card in PlayerFieldCards)
            {
                card.Card.CanAttack = true;
                card.Info.HighliteUsableCard();
                card.Ability.OnNewTurn();
                Debug.Log(card.Card.CanAttack);
            }

            while (TurnTime-- > 0)
            {
                UIController.Instance.UpdateTurnTime(TurnTime);
                yield return new WaitForSeconds(1);
            }
            ChangeTurn();
        }
        else
        {
            foreach (var card in EnemyFieldCards)
            {
                card.Card.CanAttack = true;
                card.Ability.OnNewTurn();
            }



            StartCoroutine(EnemyAITurn());
            while (TurnTime-- > 0)
            {
                UIController.Instance.UpdateTurnTime(TurnTime);
                yield return new WaitForSeconds(1);
            }

            //ChangeTurn();
        }
    }

    IEnumerator EnemyAITurn()
    {
        EnemyAI.MakeTurn();
        yield return null; // Это нужно, чтобы корутина корректно завершилась
    }



    public void RenewDeck(bool playerdeck)
    {
        if (playerdeck)
        {

            CurrentGame.PlayerDeck = new
List<Card>(decksManager.GetMyDeckCopy().cards);
            CurrentGame.PlayerDeck =
CurrentGame.ShuffleDeck(CurrentGame.PlayerDeck);
        }
        else
        {
            CurrentGame.EnemyDeck = new
List<Card>(decksManager.GetEnemyDeckCopy().cards);
            CurrentGame.EnemyDeck =
CurrentGame.ShuffleDeck(CurrentGame.EnemyDeck);
        }
    }

    public void ChangeTurn()
    {
        StopAllCoroutines();
        Turn++;
        PlayersTurn = !PlayersTurn;
        UIController.Instance.EnableTurnBtn();
        UIController.Instance.WhoseTurnUpdate();


        if (PlayersTurn)
        {
            if (CurrentGame.PlayerDeck.Count == 0)
                RenewDeck(true);
            GiveCardToHand(CurrentGame.PlayerDeck, PlayerHand, true);
            if (Turn != 1)
                CurrentGame.Player.IncreaseManapool();
            CurrentGame.Player.RestoreRoundMana();
        }
        else
        {
            if (CurrentGame.EnemyDeck.Count == 0)
                RenewDeck(false);
            GiveCardToHand(CurrentGame.EnemyDeck, EnemyHand, false);
            if (Turn != 1)
                CurrentGame.Enemy.IncreaseManapool();
            CurrentGame.Enemy.RestoreRoundMana();
        }
        StartCoroutine(TurnFunc());
    }

    public bool FlipCoin()
    {
        System.Random random = new System.Random();
        return random.Next(2) == 1;
    }

    public void CardsFight(CardController attacker, CardController defender)
    {
        defender.Card.GetDamage(attacker.Card.Attack);
        attacker.OnDamageDeal(defender);
        defender.OnTakeDamage(attacker);

        attacker.Card.GetDamage(defender.Card.Attack);
        attacker.OnTakeDamage();

        attacker.CheckForAlive();
        defender.CheckForAlive();
    }



    public void ReduceMana(bool playerMana, int manacost)
    {
        if (playerMana)
            CurrentGame.Player.Mana -= manacost;
        else
            CurrentGame.Enemy.Mana -= manacost;
        UIController.Instance.UpdateHPAndMana();
    }
```

```
    public void DamageHero(CardController card, bool isEnemyAttacked)
    {
        if (isEnemyAttacked)
            CurrentGame.Enemy.GetDamage(card.Card.Attack);
        else
            CurrentGame.Player.GetDamage(card.Card.Attack);

        UIController.Instance.UpdateHPAndMana();
        card.OnDamageDeal();
        CheckForVictory();
    }

    public void CheckForVictory()
    {
        if (CurrentGame.Enemy.HP == 0 || CurrentGame.Player.HP == 0)
        {
            StopAllCoroutines();
            Time.timeScale = 0f;
            UIController.Instance.ShowResult();
        }
    }

    public void CheckCardForManaAvailability()
    {
        foreach (var card in PlayerHandCards)
            card.Info.HighlightManaAvaliability(CurrentGame.Player.Mana);


    }

    public void HightLightTargets(CardController attacker, bool highlight)
    {
        List<CardController> targets = new List<CardController>();

        if (attacker.Card.IsSpell)
        {
            if (attacker.Card.SpellTarget == Card.TargetType.NO_TARGET)
                targets = new List<CardController>();
            else if (attacker.Card.SpellTarget ==
Card.TargetType.ALLY_CARD_TARGET)
                targets = PlayerFieldCards;
            else
                targets = EnemyFieldCards;
        }
        else
        {
            if (EnemyFieldCards.Exists(x => x.Card.IsProvocation))
                targets = EnemyFieldCards.FindAll(x => x.Card.IsProvocation);
            else
            {
                targets = EnemyFieldCards;
                EnemyHero.HighlightAsTarget(highlight);
            }
        }


        foreach (var card in targets)
        {
            if (attacker.Card.IsSpell)
                card.Info.HighlightAsSpellTarget(highlight);
            else
                card.Info.HighlightAsTarget(highlight);
        }

    }
}
```

Файл Player.cs

```
using UnityEngine;

public class Player
{
    public int HP, Mana, Manapool;
    public const int MAX_MANAPOOL = 10;

    public Player()
    {
        HP = 30;
        Mana = Manapool = 1;
    }

    public void RestoreRoundMana()
    {
        Mana = Manapool;
        UIController.Instance.UpdateHPAndMana();
    }
    public void IncreaseManapool()
    {
        Manapool = Mathf.Clamp(Manapool + 1, 0, MAX_MANAPOOL);
        UIController.Instance.UpdateHPAndMana();
    }

    public void GetDamage(int damage)
    {
        HP = Mathf.Clamp(HP - damage, 0, int.MaxValue);
        UIController.Instance.UpdateHPAndMana();
    }

    public int GetMaxManapool()
    {
        int MaxMana = MAX_MANAPOOL;
        return MaxMana;
    }
}
```

Файл SpellTarget.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class SpellTarget : MonoBehaviour, IDropHandler
{
    public void OnDrop(PointerEventData eventData)
    {

        if (!GameManagerScr.Instance.PlayersTurn)
            return;
        CardController spell =
eventData.pointerDrag.GetComponent<CardController>(),
                       target = GetComponent<CardController>();

        if (spell &&
            spell.Card.IsSpell &&
            spell.IsPlayerCard &&
            target.Card.IsPlaced &&
            GameManagerScr.Instance.CurrentGame.Player.Mana >=
spell.Card.ManaCost)
        {
            if ((spell.Card.SpellTarget == Card.TargetType.ALLY_CARD_TARGET
&&
                target.IsPlayerCard) ||
                (spell.Card.SpellTarget == Card.TargetType.ENEMY_CARD_TARGET
&&
                !target.IsPlayerCard))
            {
                GameManagerScr.Instance.ReduceMana(true,
spell.Card.ManaCost);
                spell.UseSpell(target);
                GameManagerScr.Instance.CheckCardForManaAvailability();
            }
        }
    }
}
```

Файл UIController.cs

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

public class UIController : MonoBehaviour
{
    public static UIController Instance;

    public TextMeshProUGUI PlayerMana, EnemyMana;
    public TextMeshProUGUI PlayerHP, EnemyHP;

    public Sprite ActiveManaPoint, InactiveManaPoint;
    public List<GameObject> PlayerManaPoints, EnemyManaPoints;

    public GameObject ResultGO;
    public GameObject pausePanel, settingsPanel;
    public TextMeshProUGUI ResultTxt;

    public TextMeshProUGUI TurnTimeTxt, WhoseTurn;
    public Button EndTurnButton;
    public Button PauseButton;

    private void Awake()
    {
        if (!Instance)
```

```
            Instance = this;
        else
        {
            Destroy(gameObject);
            return;
        }
        //DontDestroyOnLoad(this);
    }

    public void TogglePause()
    {
        if (Time.timeScale == 0f)
        {
            ResumeGame();
        }
        else
        {
            PauseGame();
        }
    }

    public void PauseGame()
    {
        Time.timeScale = 0f; // Остановка игры
        pausePanel.SetActive(true); // Показ окна паузы

    }

    public void ResumeGame()
    {
        Time.timeScale = 1f;
        pausePanel.SetActive(false);
    }

    public void StartGame()
    {
        EndTurnButton.interactable = true;
        ResultGO.SetActive(false);
    }

    public void OpenSettings()
    {
        pausePanel.SetActive(false);
        settingsPanel.SetActive(true);

    }

    public void CloseSettings()
    {
        pausePanel.SetActive(true);
        settingsPanel.SetActive(false);

    }

    public void UpdateHPAndMana()
    {
        //Updating mana
        PlayerMana.text =
GameManagerScr.Instance.CurrentGame.Player.Mana.ToString() + " / " +
GameManagerScr.Instance.CurrentGame.Player.Manapool.ToString();
        if (GameManagerScr.Instance.CurrentGame.Player.Mana != 0)
        {
            for (int i = 0; i <
GameManagerScr.Instance.CurrentGame.Player.Mana; i++)
            {
                PlayerManaPoints[i].GetComponent<Image>().sprite =
ActiveManaPoint;
            }
        }
        if (GameManagerScr.Instance.CurrentGame.Player.Mana !=
GameManagerScr.Instance.CurrentGame.Player.GetMaxManapool())
        {
            for (int i = GameManagerScr.Instance.CurrentGame.Player.Mana; i <
GameManagerScr.Instance.CurrentGame.Player.GetMaxManapool(); i++)
            {
                PlayerManaPoints[i].GetComponent<Image>().sprite =
InactiveManaPoint;
            }
        }

        EnemyMana.text =
GameManagerScr.Instance.CurrentGame.Enemy.Mana.ToString() + " / " +
GameManagerScr.Instance.CurrentGame.Enemy.Manapool.ToString();
        if (GameManagerScr.Instance.CurrentGame.Enemy.Mana != 0)
        {
            for (int i = 0; i <
GameManagerScr.Instance.CurrentGame.Enemy.Mana; i++)
            {
                EnemyManaPoints[i].GetComponent<Image>().sprite =
ActiveManaPoint;
            }
        }
        if (GameManagerScr.Instance.CurrentGame.Enemy.Mana !=
GameManagerScr.Instance.CurrentGame.Enemy.GetMaxManapool())
        {
            for (int i = GameManagerScr.Instance.CurrentGame.Enemy.Mana; i <
GameManagerScr.Instance.CurrentGame.Enemy.GetMaxManapool(); i++)
            {
                EnemyManaPoints[i].GetComponent<Image>().sprite =
InactiveManaPoint;
            }
        }

        //Updating HP
        PlayerHP.text =
GameManagerScr.Instance.CurrentGame.Player.HP.ToString();
        EnemyHP.text =
GameManagerScr.Instance.CurrentGame.Enemy.HP.ToString();
    }

    public void ShowResult()
    {
        ResultGO.SetActive(true);

        if (GameManagerScr.Instance.CurrentGame.Enemy.HP == 0)
            ResultTxt.text = "Hooraaaay! You won!";
        else
            ResultTxt.text = "Womp-womp... You lost.";
    }

    public void EnableTurnTime(bool enable)
    {
        if (TurnTimeTxt != null)
            TurnTimeTxt.enabled = enable;
    }

    public void UpdateTurnTime(int Time)
    {
        TurnTimeTxt.text = Time.ToString();
    }

    public void WhoseTurnUpdate()
    {
        if (GameManagerScr.Instance.PlayersTurn)
            WhoseTurn.text = "Your turn";
        else
            WhoseTurn.text = "Enemy turn";
    }

    public void EnableTurnBtn()
    {
        EndTurnButton.interactable = GameManagerScr.Instance.PlayersTurn;
    }


}
```