Факультет компьютерных наук

Департамент программной инженерии

Москва
2024

“Кактусовая бойня” – игра-платформер на Unity

Выполнили: студент группы БПИ228 Кунашев Данил
студент группы БПИ228 Лысин Кирилл

Научный руководитель: Виденин С.А., доцент департамента программной инженерии, кандидат педагогических наук

Описание темы курсового проекта

«Кактусовая бойня» - игра-платформер, в которой игроку предстоит:

- Изучать историю игры по мере продвижения по сюжету
- Взять управление главным героем в битве с кактусами
- Проходить игровые уровни

Актуальность

Разработка игры-платформера является актуальной.
Широкую аудиторию могут привлечь следующие аспекты игр в таком жанре:

- Увлекательный игровой процесс
- Низкий порог вхождения
- Разнообразие игровых механик
- Нарастающая сложность
- Развитие навыков в реальной жизни

Цели и задачи проекта

Цели:

- Разработать игру в жанре платформер на движке Unity, в которой персонаж сражается с кактусами.

Задачи:

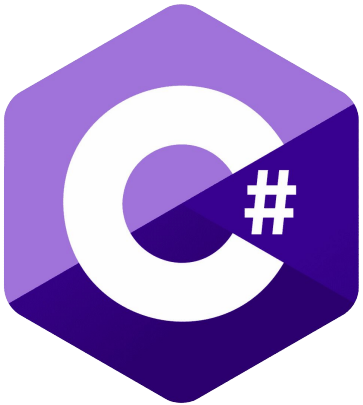
- Создание графики для пользовательского интерфейса
- Создание графики для игры (уровни, персонаж, враги и используемые предметы)
- Создание анимаций для персонажа и врагов
- Проектирование и реализация интерфейса, корректно работающего с разными разрешениями экрана
- Проектирование и реализация игровых уровней
- Дизайн игровых механик
- Создание сюжета и обучения
- Написание скриптов, связанных с интерфейсом и визуальной составляющей игры
- Настройка объектов на сценах в Unity

Стек используемых технологий

Adobe Illustrator

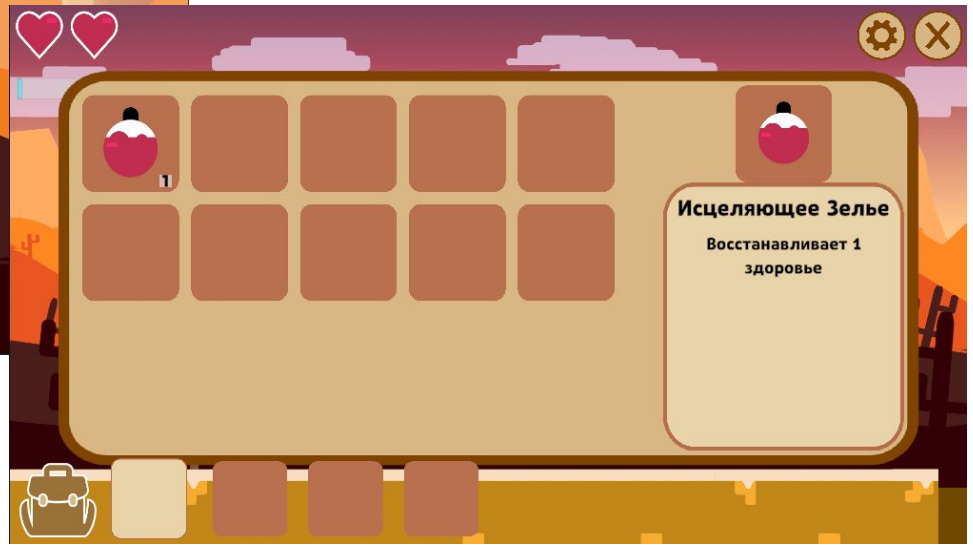
DragonBones

Unity

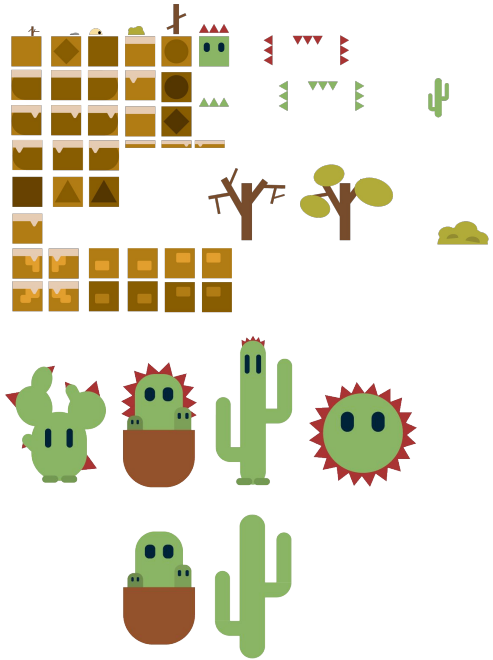


C#

Создание графики для пользовательского

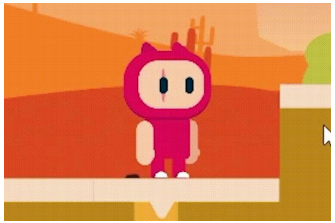


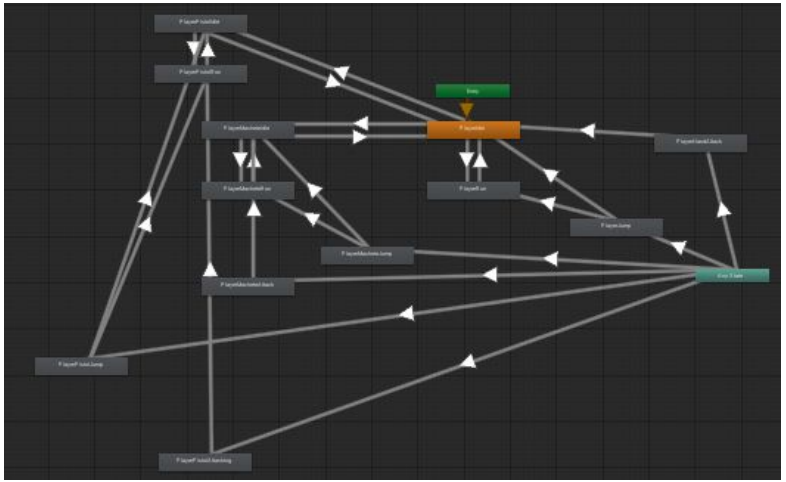
Создание графики для игры (уровни, персонаж, враги и используемые предметы)

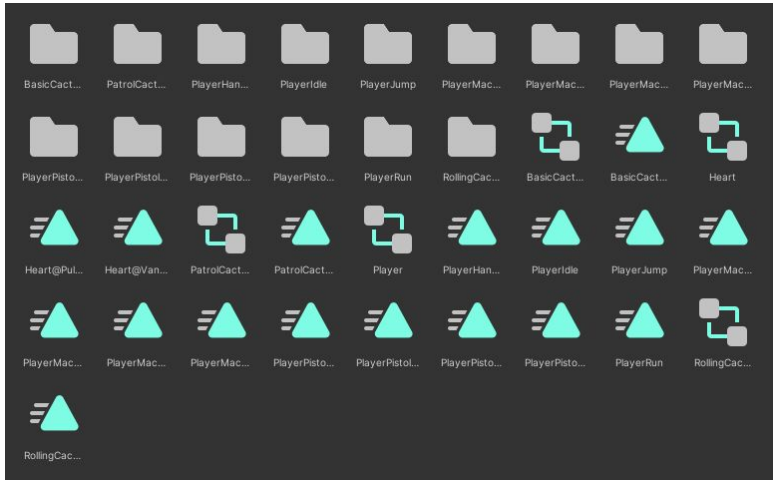


Создание анимаций для персонажа и врагов

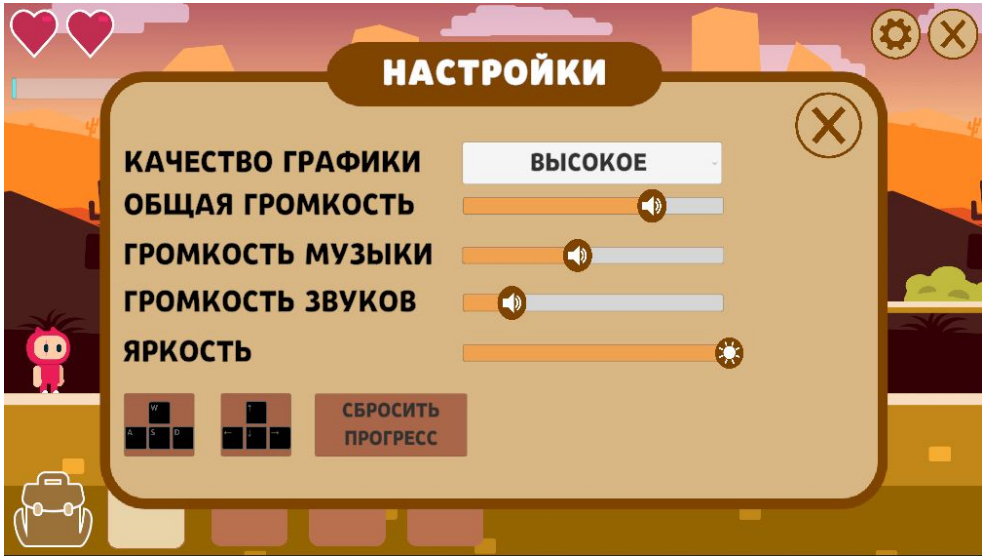


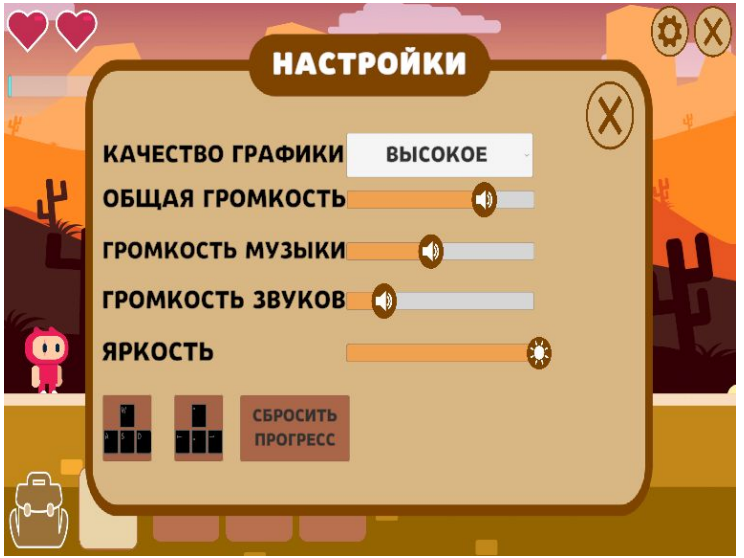







Проектирование и реализация интерфейса, корректно работающего с разными разрешениями экрана





Проектирование и реализация игровых уровней

Исцеляющее зелье

Новый вид кактуса

Новый вид оружия

И так далее…

Дизайн игровых механик

Два типа колючек. Наносят урон (1/2) при соприкосновении с игроком

Трамплин, который подбрасывает игрока вверх

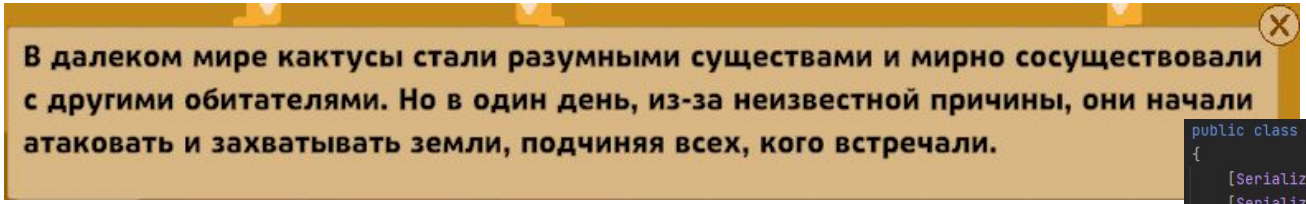
Выбирает рандомную точку и идет к ней (пока не заметит игрока)

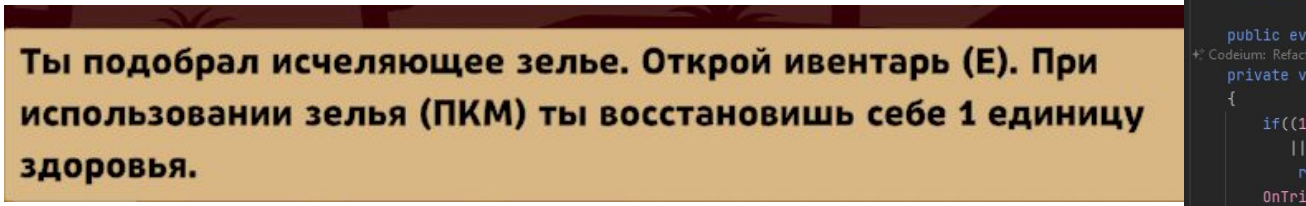
Двигается от точки А до точки Б

Стоит. Как только замечает игрока, начинает быстрое движение в его сторону

Выбирает рандомную точку и идет к ней. Как только замечает игрока - останавливается и начинает стрелять по нему

Создание сюжета и обучения






```
public class StorylineTrigger : MonoBehaviour
{
    [SerializeField] private StoryInsertionAsset storyInsertion;
    [SerializeField] private LayerMask playerLayer;

    public event Action<StoryInsertionAsset> OnTriggerActivated;
Codeium: Refactor  Explain  Docstring
    private void OnTriggerEnter2D(Collider2D other)
    {
        if((1 << other.gameObject.layer &  (int)playerLayer) == 0
           || !other.TryGetComponent(out PlayerState _))
            return;
        OnTriggerActivated?.Invoke(storyInsertion);
        gameObject.SetActive(false);
    }


Codeium: Refactor  Explain  Docstring
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.cyan;
        Gizmos.DrawWireSphere(transform.position,  radius: 0.5f);
    }
}
```

Написание скриптов, связанных с интерфейсом и визуальной составляющей игры

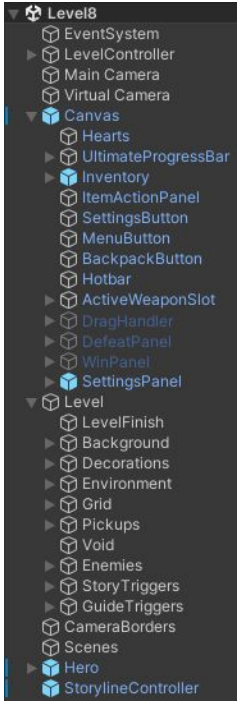
```
public class Scenes : MonoBehaviour
{
    [SerializeField] private SceneField mainMenuScene;


Codeium: Refactor | Explain | Docstring
    public static void ChangeScene(SceneField scene)
    {
        SceneManager.LoadScene(scene.SceneName);
    }


Codeium: Refactor | Explain | Docstring
    public void ReloadActiveScene()
    {
        var sceneName:string = SceneManager.GetActiveScene().name;
        SceneManager.LoadScene(sceneName);
    }

Codeium: Refactor | Explain | Docstring
    public void LoadLastAvailableLevel()
    {
        var progressManager = PlayerProgressManager.Instance;
        var lastAvailableLevelName:string = progressManager.LastAvailableLevelName;
        if (lastAvailableLevelName == null)
            return;
        SceneManager.LoadScene(lastAvailableLevelName);
    }

Codeium: Refactor | Explain | Docstring
    public void LoadMainMenu() => ChangeScene(mainMenuScene);
}
```

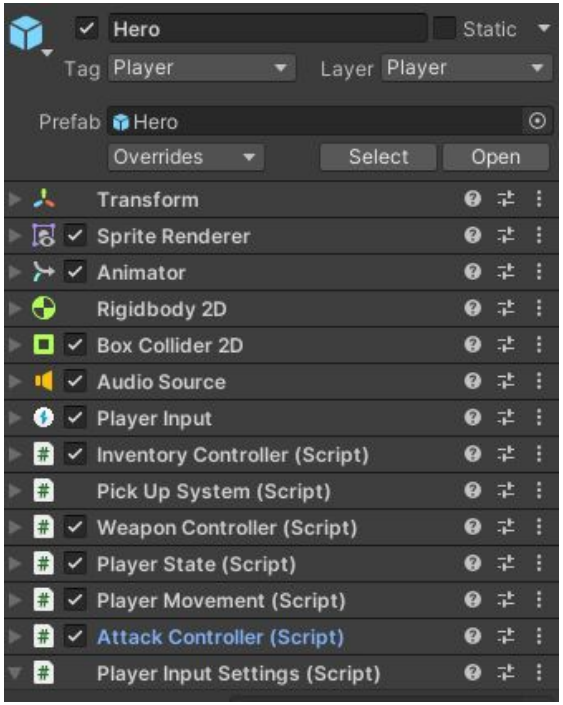
```
public class HeartVisuals : MonoBehaviour
{
    [SerializeField] private GameObject heartPrefab;
    [SerializeField] private PlayerState playerState;
    [SerializeField] private float activateHeartDuration = 1f;

    private int _currentHealth;
    private List<GameObject> Hearts { get; } = new List<GameObject>();

Codeium: Refactor | Explain | Docstring
    private void Start()
    {
        playerState.OnUpdateHealth += HandleHealthUpdate;
        playerState.OnUpdateMaximumHealth += HandleMaxHealthUpdate;

        _currentHealth = playerState.HealthPoints;
        for (int i = 0; i < playerState.MaximumHealth; ++i)
        {
            var heart:GameObject = Instantiate(heartPrefab, transform, worldPositionStays:false);
            heart.transform.SetSiblingIndex(i);
            Hearts.Add(heart);
            if(i >= _currentHealth)
                heart.gameObject.SetActive(false);
        }
    }

Codeium: Refactor | Explain | Docstring
    private void HandleHealthUpdate(int newHealth)
    {
        while (newHealth > _currentHealth && _currentHealth < playerState.MaximumHealth)
        {
            ++_currentHealth;
            ActivateHeart();
        }
```

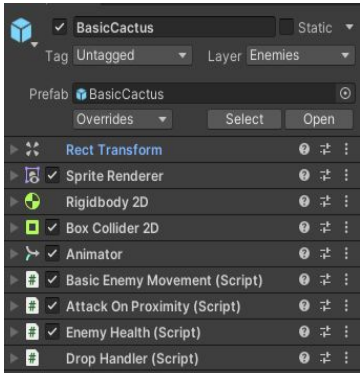
```
public class ParallaxBehaviour : MonoBehaviour
{
    [SerializeField] private Transform followingTarget;
    [SerializeField, Range(0f, 1f)] private float parallaxStrength = 0.1f;
    [SerializeField] private bool disableVerticalParallax;
    private Vector3 targetPreviousPosition;

Codeium: Refactor | Explain | Docstring
    private void Start()
    {
        if (followingTarget) return;
        if (Camera.main != null) followingTarget = Camera.main.transform;

        targetPreviousPosition = followingTarget.position;
    }

Codeium: Refactor | Explain | Docstring
    private void FixedUpdate()
    {
        var delta:Vector3 = followingTarget.position - targetPreviousPosition;
        if (disableVerticalParallax) delta.y = 0f;
        targetPreviousPosition = followingTarget.position;
        transform.position += delta * parallaxStrength;
    }
}
```

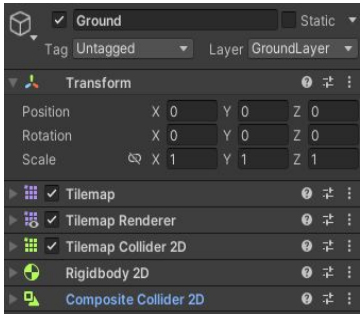
```
public class Finish : MonoBehaviour
{
    [SerializeField] private GameObject canvas;
    [SerializeField] private GameObject winPanel;
    [SerializeField] private GameObject player;
    [SerializeField] private LayerMask playerLayer;
Codeium: Refactor | Explain | Docstring
    private void OnTriggerEnter2D(Collider2D other)
    {
        if ((1 << other.gameObject.layer & playerLayer) == 0)
            return;
        foreach (Transform child in canvas.transform)
        {
            child.gameObject.SetActive(false);
        }
        winPanel.SetActive(true);
        player.SetActive(false);

        var activeScene = SceneManager.GetActiveScene();
        PlayerProgressManager.Instance.ChangeLevelStatus(activeScene.name, LevelStatus.Completed);
    }
}
```

Настройка объектов на сценах в Unity









Демонстрация игры

Планы на будущее

Я собираюсь продолжить разработку игры, реализовав данные вещи:

- Разместить проект в онлайн-сервисе дистрибуции цифрового контента (Steam, Origin, Epic Games)
- Больше уникальных игровых механик
- Больше врагов
- Добавление уникальных боссов
- Больше уровней
- Развитие сюжета игрового мира
- Добавить систему достижений
- Добавить статистику прохождения уровней (количество прохождений, лучшее время)
- Выбор языка (перевод на английский)
- Адаптация игры под мобильные устройства


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •