РАЗРАБОТКА 2D-ИГРЫ ПОД МОБИЛЬНЫЕ УСТРОЙСТВА: ОТ ИДЕИ ДО ПРОТОТИПА

САВКИНА Анастасия Васильевна
кандидат технических наук, доцент
КОТЬКИН Егор Александрович
студент
Мордовский государственный университет им. Н. П. Огарёва
г. Саранск, Россия

В статье рассматривается процесс разработки прототипа мобильной двухмерной игры в среде Unity. Основное внимание уделяется архитектурным решениям, реализации ключевых игровых систем, таких как спавн объектов, система перетаскивания, управление состоянием уровня и итеративному подходу к балансировке сложности. Описывается эволюция проекта от базового функционала до комплексного прототипа с нелинейным возрастанием трудности.

Ключевые слова: Unity, геймдизайн, мобильная разработка, 2D, архитектура игровых систем, итеративная разработка, балансировка сложности.

Разработка студентами игр играет важную роль в освоении информационных технологий, поскольку позволяет повысить мотивацию в освоении современных технологий программирования, применять теоретические знания на практике, стимулировать творчество и креативность, развивать комплекс навыков, формировать цифровую грамотность [2]. Целью работы являлась разработка функционального прототипа мобильной 2D-игры, совмещающей элементы головоломки и динамического экшена. Основной игровой механикой определено перетаскивание управляемого объекта с последующим поглощением статических

целей для набора очков в условиях ограниченного времени. В статье последовательно анализируются этапы реализации и интеграции ключевых систем проекта.

Система генерации игровых объектов (спавна). Была реализована система процедурного размещения объектов на основе предопределенных шаблонов (паттернов). Паттерны хранятся в формате JSON-файлов, содержащих координаты позиций для основного и второстепенных объектов [3]. Алгоритм спавна на каждом игровом этапе выполняет следующие действия:

Загрузка случайного паттерна из доступного для текущего уровня набора объектов реализуется с помощью скрипта. Для управления свойствами объектов, а также для их взаимодействия, объявлен экземпляр класса одного скрипта в другом со ссылкой на объект на сцене, для обеспечения подсчета количества набранных очков [4].

Далее, объявляем класс куба, у которого две переменные, одна типа integer, другая Text. Text является классом из библиотеки UintyEngine.UI. Переменная этого типа управляет всеми параметрами текста: шрифт, цвет, заполнение и т.д.

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

public class EnemyHP : MonoBehaviour
{
    public int EnPoints;
    public Text PointsText;
}
```

С помощью строчки кода EnemyHP=Enemy.GetComponent<EnemyHP>(), можно получаем доступ к переменным в классе EnemyHP (EnPoints, PointsText).

Часть кода, которая отвечает за спавн объектов и появления их в случайном порядке на заготовленных позициях (далее Паттерны). Объекты появляются

циклически, при этом, сначала меняется текст на Объекте, который будет создан, затем создаётся сам Объект с помощью функции Instantiate, которая принимает в качестве входных параметров объект, позицию на которой нужно его создать, вращение и родительский объект.

```
for (int i = 0; i < 5; i++)
{
    HP.PointsText.text = HP.EnPoints + "";
    random = Random.Range(0, spawner.Count-1);
    Instantiate(Enemy,spawner[random].position,Quaternion.iden-
    tityspawner[random]);
    HP.EnPoints *= 2;
    spawner.RemoveAt(Random);
}
```

В этом коде Spawner является переменной List типа Transform. Transform хранит все данные о позиции, размере, масштабе. Далее изменяется количество очков на следующем объекте и из списка удаляется та позиция, которая уже занята. Таким образом, инстанциирование префабов объектов на сцене осуществляется в соответствии с координатами из паттерна с использованием метода Instantiate.

Инициализация параметров каждого объекта (количество очков, графическое представление) происходит через межскриптовое взаимодействие. Для связи используется паттерн инжекции зависимостей: ссылка на компонент EnemyHP (содержащий поля EnPoints (integer) и PointsText (UnityEngine.UI.Text)) экземпляра объекта передается в управляющий скрипт спавна.

Система управления игровым процессом (Drag-and-Drop). Взаимодействие игрока реализовано через кастомную систему перетаскивания (Drag-and-Drop), основанную на классах библиотеки UnityEngine.EventSystems. Система обрабатывает события:

- OnBeginDrag: Инициализация перетаскивания.

- OnDrag: Непрерывное обновление позиции объекта. Для корректного преобразования координат касания в мировые координаты камеры используется метод RectTransformUtility.ScreenPointToWorldPointInRectangle с учетом поправочного коэффициента для различных разрешений экрана.
- OnEndDrag: Завершение взаимодействия.

Логика столкновения и поглощения объектов реализована в методах слота (Slot), принимающего объект. При срабатывании события OnDrop выполняется проверка на наличие дочернего объекта в слоте, сравниваются значения очков игрока и целевого объекта. В случае успешного условия, целевой объект уничтожается, а его очки добавляются к счету игрока. Ниже представлен класс объекта перетаскивания и его функции, которые реагируют на события начала перемещения, дальнейшего его движения и окончания перемещения.

```
public class DragItem: MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
public void OnBeginDrag (PointerEventData eventData)
public void OnDrag (PointerEventData eventData)
public void OnEndDrag (PointerEventData eventData)
```

Для того, чтобы обеспечить работу игры на экране телефонов разных моделей, была введена вспомогательная функция, определённая в RectTransformUtility, которая позволяет преобразовывать точку касания на экране в точку касания на основной камере игры, а затем изменить позицию перетаскиваемого объекта с учётом смещения. Класс слота, в который можно поместить объект игрока, и его функция которая реагирует на помещение этого объекта.

Далее, она проверяет наличие дочернего объекта в слоте, сопоставляет количество очков игрока и на объекте, а затем обрабатывается разрушение объекта в слоте и изменение количества очков игрока.

```
Destroy(transform.GetChild(2).gameObject);
DragItem.dragItem.PlayerPoints+= HP.EnPoints;
DragItem.dragItem.PPText.text=DragItem.dragItem.PlayerPoints + "";
```

```
spawner.DestroyedEnem++;
```

Для взаимодействия классов слота и объекта перемещения, в качестве родительских классов выбраны хендлеры, которые доступны в библиотеке UnityEngine.EventSystems.

Прогрессия уровней и мета-игра. Игровой процесс структурирован в виде башен (уровней сложности), каждая из которых состоит из последовательности этажей. Для прохождения этажа требуется уничтожить пять объектов, а, чтобы пройти один из уровней необходимо пройти энное количество этажей. Для этого спавн объектов был выведен в отдельную функцию для неоднократного обращения к ней и для осуществления сброса очков при переходе на следующие этажи. Обьявляем public void SpawnEnem() и вызываем метод FixedUpdate через фиксированное количество кадров, то есть в независимости от того сколько у игрока кадров в секунду.

```
HP.EnPoints = (int)Mathf.Round((Player.PlayerPoints * 0.5f));
int playerPointsPrev = Player.PlayerPoints + HP.EnPoints;
for (int i = 0; i < 5; i++)
{
    HP.PointsText.text = HP.EnPoints + "";
    random = Random.Range(0, spawner.Count - 1);
    Instantiate(Enemy, spawner[random].position, Quaternion.iden-
tity, spawner[random]);
    HP.EnPoints = (int)Mathf.Round((playerPointsPrev)
*(0.6f+i*0.1f))
    playerPointsPrev += HP.EnPoints;
    spawner.RemoveAt(random);
}
```

В этом методе идёт проверка на количество сломанных объектов и Invocation функции SpawnEnem (Invoke позволяет вызвать функцию через заданное количество секунд, для чего будет рассказано позже). В этой функции можно заметить, что логика распределения очков уже изменилась на другую. Очки рас-

пределяются так: общее число очков на каждой итерации уменьшается на процент, который, в свою очередь уменьшается на 10% (начальное значение процента – 50%).

Заключение и перспективы. В результате работы был создан полнофункциональный прототип, демонстрирующий жизнеспособность основной игровой механики и архитектурную масштабируемость проекта. Проведенная итеративная балансировка позволила сформировать кривую сложности, обеспечивающую плавное введение игрока в механику с последующим ее усложнением [1].

Направлениями для дальнейшей разработки определены:

- Введение динамических препятствий
- Реализация босс-уровней с проверкой реакции игрока.
- Введение дополнительных модификаторов (порталы, меняющие объекты местами).

СПИСОК ЛИТЕРАТУРЫ

1. Машнин Т. Введение в облачные и распределенные информационные системы. – Екатеринбург: Ridero, 2020. – 138 с.

2. Мэтиган Д. Психология видеоигр. Взгляд психолога на видеоигры. – М.: Изд-во БОМБОРА, 2023. – 352 с.

3. Тоттен К. Левел-дизайн. Архитектурный подход и пространственное проектирование. – М.: Эксмо, 2025. – 592 с.

4. Харрисон Ф. Изучаем C# через разработку игр на Unity. – Санкт-Петербург. Питер, 2022. – 400 с.

DEVELOPMENT OF A 2D GAME FOR MOBILE DEVICES: FROM IDEA TO PROTOTYPE

SAVKINA Anastasiia Vasilievna

Candidate of Technical Sciences, Associate Professor

KOTKIN Egor Alexandrovich
Student
Mordovian State University named after. N.P. Ogareva
Saransk, Russia

This article examines the process of developing a prototype for a two-dimensional (2D) mobile game within the Unity engine. The primary focus is placed on architectural solutions and the implementation of key game systems, such as object spawning, a drag-and-drop system, level state management, and an iterative approach to difficulty balancing. The paper describes the project's evolution from basic functionality to a comprehensive prototype featuring non-linear difficulty progression.

Keywords: Unity, game design, mobile development, 2D, game systems architecture, iterative development, difficulty balancing.