XI Республиканская научно-практическая конференция-конкурс научно-исследовательских работ учащихся средних, средних специальных учебных заведений и студентов вузов «От Альфа к Омеге…» (с международным участием)
Секция 2. Компьютерные науки и программирование
НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЕ РАБОТЫ ШКОЛЬНИКОВ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Государственное учреждение образования
«Гимназия № 2 г. Гродно»

ПРИМЕНЕНИЕ ТЕХНОЛОГИЙ UNITY 3D ПРИ СОЗДАНИИ ИГРЫ ДЛЯ ВЫЯВЛЕНИЯ БОЛЕЕ ПОДХОДЯЩЕЙ АУДИТОРИИ ИГРОКОВ

Чернюк Владислав Сергеевич,
учащийся 10 «А» класса

Козловская Ирина Зигмундовна,
учитель информатики
ГУО «Гимназия № 2 г. Гродно»
высшая кв. категория учителя информатики

Гродно, 2021

ИЗУЧЕНИЕ ВОЗМОЖНОСТИ МЕЖПЛАТФОРМЕННОЙ СРЕДЫ РАЗРАБОТКИ UNITY 3D»

В. С. Чернюк
ГУО «Гимназия № 2 г. Гродно», 10 «А» класс,
Гродно, Беларусь

Научный руководитель – И. З. Козловская, учитель информатики ГУО «Гимназия № 2 г. Гродно», высшая кв. категория учителя информатики.

Работа 25 с., 3 ч., 25 рис., 3 источника, 2 прил.

Ключевые слова: Scripts (скрипты), Scene (игровая сцена), Assets (ассеты), Prefab (префаб), Unity (юнити), функции, методы, компоненты, интерфейс и когнитивный поток.

В работе исследуется разработка компьютерных игр и их практическое применение, приводятся основные факторы испытуемых и их группировка по возрастному признаку, а также произведён анализ полученных данных. Показаны основные методы создания скриптов, расположение объектов в сцене, проектирование уровней и целей игры для поддержания заинтересованности игрока в игровом процессе. Проанализирована информация, полученная в ходе опроса аудитории различных возрастов. Выполнена интерпретация человеческого мышления в поставленной ситуации при различных условиях, и показана возможность практического применения полученной информации в проекте.

Объектом исследования является разработка видео игры 3D пространства Unity и практическое применение игр в современном мире, психологии.

Цель работы – поэтапная разработка проекта и анализ психологического поведения человека.

Работа посвящена изучению возможностей разработки игр на Unity 3D и мониторинг переживания человека в поставленных условиях.

В результате исследования впервые были получены следующие результаты:

1. Нахождение информации в интернет - ресурсах для дальнейшего использования с целью изучения основ и создании скриптов на языке программирования C# для взаимодействия с физикой межплатформенной среды разработки Unity 3D.
2. Построение игрового уровня в сцене.
3. Разработка структуры кода по принципам ООП (объектно-ориентированного программирования) и написание кода для взаимодействия скрипта с процессами игры.
4. Компиляция проекта и финальные настройки игры.
5. Статистика поведения игроков и анализ полученных результатов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Компьютерные игры типа жанра головоломки уже долгое время носят звание игр, требующих умственной работы аналитического плана. Их аналоговый предшественник – шахматы, пазлы и даже дженга, способность которых развивать аналитическое мышление является неоспоримой.

С помощью игр многие учёные мира могут понять работу человеческого мозга, в особенности детского. Чем младше ребёнок, тем сложнее получить от него сведения о том, что он чувствует, думает, как воспринимает взаимодействие с родителями, братьями и сёстрами, другими близкими людьми. Практика показывает недостаточность диагностических средств в этой области. Среди наиболее информативных способов используется игра ребёнка. Она в качестве одного из диагностических средств работы с детьми давно привлекает внимание психологов, так как игровая форма взаимодействия имеет множество преимуществ, благодаря глубокому символическому значению игровых проявлений ребёнка, которые происходят произвольно, на подсознательном уровне.

Почему именно Unity?

Unity – больше, чем движок. Это среда для разработки компьютерных игр, в которой объединены различные программные средства, используемые при создании ПО – это текстовый редактор, компилятор, отладчик и так далее. При этом, благодаря удобству использования, Unity делает создание игр максимально простым и комфортным, а мультиплатформенность движка позволяет охватить как можно большее количество игровых платформ и ОС. [3]

В первую очередь, как уже было упомянуто, движок Unity3D дает возможность разрабатывать игры с небольшим порогом вхождения, не требует для этого каких-либо особых знаний и большего опыта работы в сфере разработки.

На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями. Здесь используется компонентно-ориентированный подход, в рамках которого создаются объекты (например, главного героя) и к ним добавляют различные компоненты (например, визуальное отображение персонажа и способы управления им). Движок позволяет рисовать карты и расставлять объекты в реальном времени и сразу же тестировать получившийся результат.

Сильная сторона Unity 3D – поддержка огромного количества платформ, технологий, API. Созданные на движке игры, можно легко портировать между разными платформами. Физика твердых тел, ragdoll и тканей, система Level of Detail, коллизии между объектами, сложные анимации – все это можно реализовать силами движка. [2]

1 ВЫЯВЛЕНИЕ НЕОБХОДИМОЙ ИНФОРМАЦИИ ПЕРЕД РАЗРАБОТКОЙ

1.1 Планирование проекта, разработка логики игры

Усилия, направленные на создание хорошей игры, прилагаются задолго до этапа кодирования. Планирование, несколько недооцениваемое многими начинающими разработчиками, — это одна из важнейших фаз разработки игрового проекта. Именно на этом этапе будут определенны основные направляющие для всех остальных стадий. Целевым жанром является головоломка.

Головоломка – название жанра компьютерных игр, целью которых является решение логических задач, требующих от игрока логики, стратегии и интуиции или иных случаев некоторого наличия удачи.

Следует сделать небольшие задачи, за решение которых игрок будет получать поощрения (счётчик очков или прохождение уровня с отличием). Чем больше завлекает головоломка, тем более искренние в ней действия будет совершать человек.

Также стоит сделать телепорты, переносящие игрока из одной точки в другую для создания больших вариаций прохождения уровней, а уровни в свою очередь будут состоять из лабиринтов с невысокими стенами.

Решения некоторых головоломок могут показаться слишком простыми, поэтому нужно подгонять игрока ограничением времени, ловушками и иногда простыми противниками.

Отлично, игра будет являться лабиринтом с шаром, появляющимся в центре. Цель игры – собрать все монеты. Вид камеры будет от третьего лица с картой в верхнем правом углу (при возможности можно отключить подсказку) для простоты ориентации в пространстве. Кабинки-телепорты будут подниматься при нахождении игрока рядом, будут пересоздавать тот же объект, но уже в другой точке карты. Меню используется для входа в игру и выхода, настройки качества и разрешения, панель паузы для остановки игры или перезагрузки уровня для удобства и гибкости.

1.1.1 Межплатформенная среда разработки Unity 3D

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет - приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие. [3]

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов.

Проект в Unity делится на сцены (уровни) — отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты — объекты, которые не имеют модели («пустышки»). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент Transform — он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Renderer, делающий модель объекта видимой. [1]

К объектам можно применять коллизии (в Unity т. н. коллайдеры — collider), которых существует несколько типов. Также Unity поддерживает физику твёрдых тел и ткани, а также физику типа Ragdoll. В редакторе имеется система наследования объектов; дочерние объекты будут повторять все изменения позиции, поворота и масштаба родительского объекта. Скрипты в редакторе прикрепляются к объектам в виде отдельных компонентов.

Редактор Unity поддерживает написание и редактирование шейдеров. Редактор Unity имеет компонент для создания анимации, но также анимацию можно создать предварительно

в 3D-редакторе и импортировать вместе с моделью, а затем разбить на файлы. При компиляции проекта создается исполняемый (.exe) файл игры (для Windows), а в отдельной папке — данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

1.2 **Работа с объектами и их компонентами**

В первую очередь необходимо познакомиться с интерфейсом самого Unity. При запуске пустого проекта появиться окно новой сценой посередине с трехмерными объектами по умолчанию: основная камера и направленный источник света (см. рисунок 1.1).

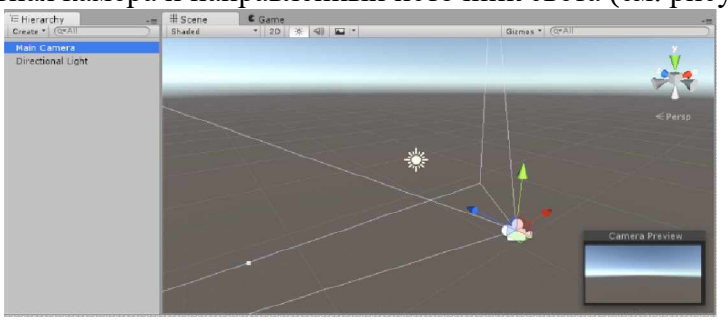


Рисунок 1.1 Сцена и иерархия

В левой стороне расположена иерархия сцены, а окно сцены по центру. Сцены содержат окружение и меню игры. Каждый новый файл сцены будет являться уникальным уровнем в игре или меню. В каждой Сцене можно размещать свое окружение, препятствия и декорации, по сути, проектируя и строя свою игру по частям, подобно конструктору. Unity сохраняет сцены как Assets в папке Assets проекта. Это означает, что они отображаются в окне проекта вместе с остальными файлами проекта. Папку проекта можно увидеть внизу проекта Unity (см. рисунок 1.2).

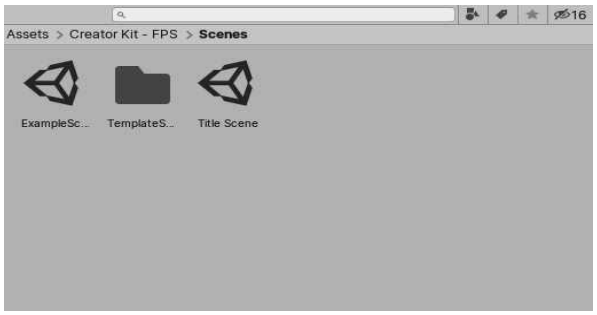


Рисунок 1.2 Сохраненные сцены, видимые в окне проекта

GameObject - самая важная концепция в редакторе Unity. Каждый объект в игре - это GameObject, от персонажей и коллекционных предметов до источников света, камер и спецэффектов. Однако GameObject ничего не может делать сам по себе; для начала необходимо присвоить ему свойства, прежде чем он сможет стать персонажем, средой или специальным эффектом (см. рисунок 1.3).

Рисунок 1.3 Анимированный персонаж, свет, дерево и источник звука.

Чтобы дать GameObject свойства, необходимые для того, чтобы он стал источником света, деревом или камерой, нужно добавить к нему компоненты. В зависимости от того, какой объект будет создан, необходимо добавлять различные комбинации компонентов в GameObject. Unity имеет множество различных типов встроенных компонентов, и также можете создавать свои собственные компоненты с помощью Unity Scripting API.

GameObject - это фундаментальные объекты в Unity, которые представляют персонажей, реквизит и декорации. Сами по себе они не дают многого, но действуют как контейнеры для компонентов, реализующие реальную функциональность. Твердый кубический объект имеет компоненты Mesh Filter и Mesh Renderer для рисования поверхности куба и компонент Box Collider для представления твердого объема объекта с точки зрения физики (см. рисунок 1.4).

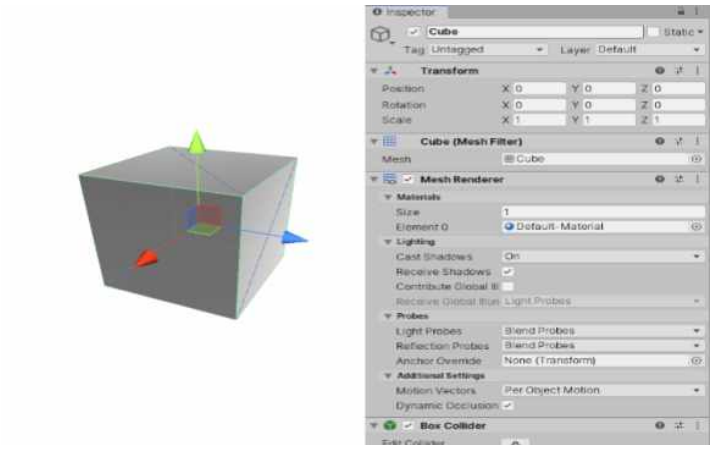


Рисунок 1.4 Простой Cube GameObject с несколькими компонентами

GameObject всегда имеет прикрепленный компонент Transform (для представления положения и ориентации), и удалить его невозможно. Другие компоненты, которые придают объекту его функциональность, могут быть добавлены из меню «Компонент» редактора или из сценария. Также есть много полезных готовых объектов.

Создание сценариев - это написание собственных дополнений к функциям редактора Unity в коде с использованием API сценариев Unity. Если после создания скрипт прикрепить к GameObject, скрипт появляется в инспекторе GameObject, как встроенный компонент. Это потому, что скрипты становятся компонентами, при сохранении их в своем проекте.

Система Prefab Unity позволяет создавать, настраивать и хранить GameObject вместе со всеми его компонентами, значениями свойств и дочерними GameObject в качестве повторно используемого актива. Prefab Asset действует как шаблон, из которого можно создавать новые экземпляры Prefab в Scene. Если в проекте будет вновь использован GameObject, настроенный определенным образом - например, неигрового персонажа (NPC), опоры или части декораций - в нескольких местах сцены или в нескольких сценах проекта, нужно преобразовать его в Prefab. Это лучше, чем просто копировать и вставлять GameObject, потому что система Prefab позволяет автоматически синхронизировать все копии.

2 СОЗДАНИЕ ИГРЫ «THE MAZE»

2.1 Построение объектов в сцене

В игровой сцене был создан примитивный куб с увеличенным параметром Scale в компоненте Transform и переименован как “Ground” (см. рисунок 2.1). Объект Ground выступает в роли поверхности, на которой будет расположена игровая зона, и будет использован в программе лишь один раз, поэтому для него не стоит создавать отдельный Prefab. В компоненте Mesh Renderer для придания контрастности был изменён материал с Default-Material на Ground. Под поверхность был подобран простой лабиринт из префабов Wall разных размеров.

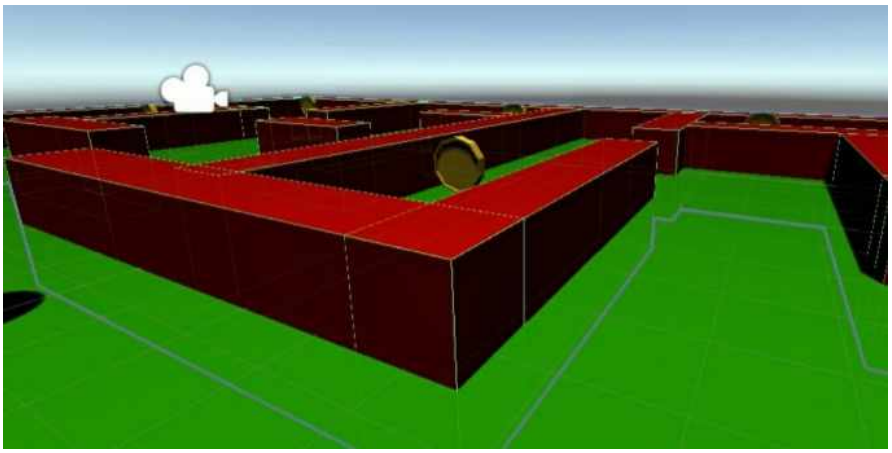


Рисунок 2.1 Сцена в Unity

В центр лабиринта помещён Prefab PlayerObject, созданный из примитивной сферы с компонентами: Rigidbody, Mesh Filter, Mesh Renderer, Sphere Collider и одним скриптом (см. рисунок 2.2). К нему будет следовать камера, и передвижение объекта будет реализовано при помощи физики игрового движка.

Рисунок 2.2 Компоненты PlayerObject

Rigidbody - это основной компонент, подключающий физическое поведение для объекта. С прикреплённым Rigidbody объект немедленно начнёт реагировать на гравитацию.

Если добавлен один или несколько компонентов Collider, то при коллизиях (столкновениях) объект будет передвигаться.

Составные коллайдеры приближают форму GameObject, сохраняя при этом низкую нагрузку на процессор. Их удобно использовать для придания более гибких условий для индивидуальных настроек в дочерних GameObjects. Например, можно вращать блоки относительно локальных осей родительского GameObject.

По карте был размещён Prefab Coin, монеты будут использоваться как цель в игре, а также как игровые очки (см. рисунок 2.3). Coin изначально был создан в Blender 3D и был импортирован в Unity. Для сохранения .blend файлов необходимо их переместить в папку Assets нужного проекта путём использования любого проводника или же при помощи Drag&Drop.

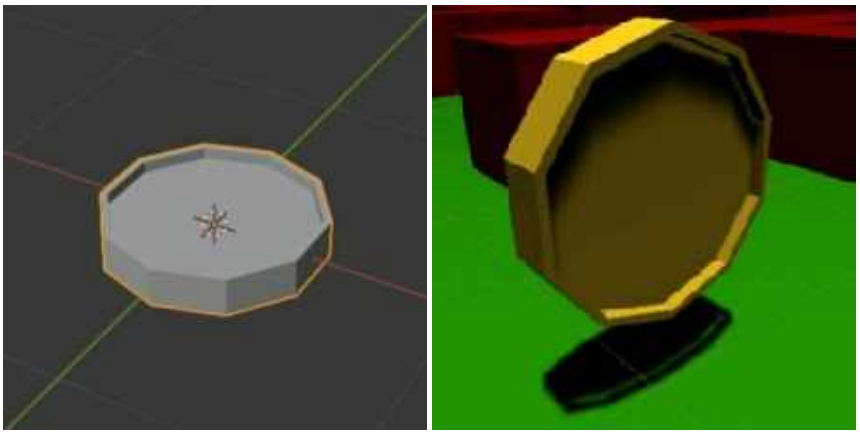


Рисунок 2.3 Модель Coin в Blender и Unity

Blender — профессиональное свободное и открытое программное обеспечение для создания трёхмерной компьютерной графики, включающее в себя средства моделирования, скульптинга, анимации.

В игру добавлена новая модель, но уже с готовой анимацией (см. рисунок 2.4). Создан префаб с важным компонентом Animator. Контроллер аниматора создается Unity и позволяет руководить набором анимаций для персонажа или предмета и переключаться между ними, когда выполняется некоторое условие. Например, можно переключиться от анимации ходьбы к прыжку при нажатии клавиши пробела. Контроллер управляет переходами между анимациями, используя так называемую машину состояний (State Machine). [1]

Когда игровой объект окажется внутри телепорта, он должен будет создать ощущения мгновенного перемещения из одной точки карты в другую с небольшой задержкой. Один из способов реализации - поменять координаты объекта на новые, но так как подобное перемещение в реальной жизни на данный момент невозможно, можно “пересобирать” игрока. То есть подконтрольный шарк будет удаляться и пересоздаваться в другой точке, а камера лишь будет менять объект, к которому закреплена.

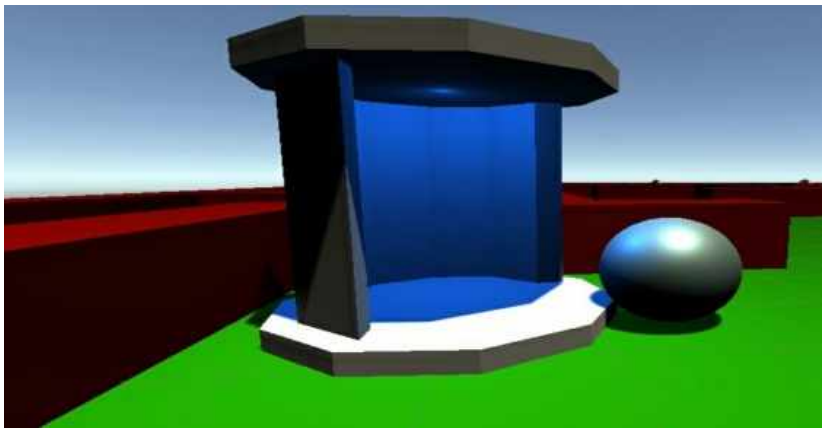


Рисунок 2.4 Телепорт в игре

2.2 Написание скриптов

Скриптинг - необходимая составляющая всех игр. Даже самые простые игры нуждаются в скриптах для реакции на действия игрока и организации событий геймплея. Кроме того, скрипты могут быть использованы для создания графических эффектов, управления физическим поведением объектов или реализации пользовательской ИИ (искусственный интеллект) системы для персонажей игры.

В отличии от других ассетов, скрипты обычно создаются непосредственно в Unity. Создать скрипты можно используя меню Create в левом верхнем углу панели Project или выбрав Assets > Create > C# Script (или JavaScript/Boo скрипт) в главном меню. Новый скрипт будет создан в выбранной папке панели Project. Имя нового скрипта будет выделено, предлагая ввести новое имя (см. рисунок 2.5). Лучше ввести новое имя скрипта сразу после создания, чем изменять его потом. Имя будет использовано, чтобы создать начальный текст в скрипте.

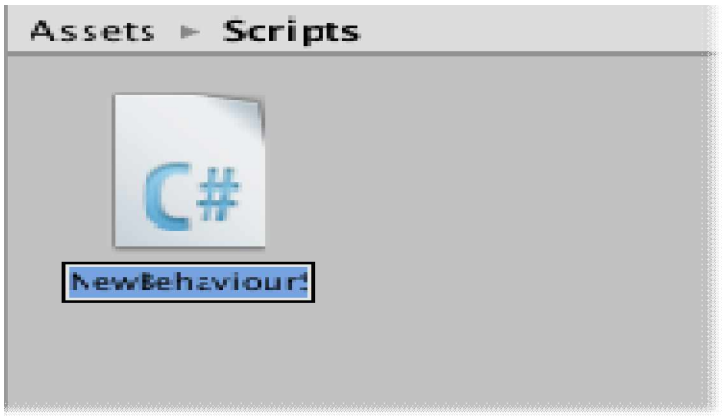


Рисунок 2.5 Unity предлагает ввести имя

Скрипт взаимодействует с внутренними механизмами Unity за счет создания класса, наследованного от встроенного класса, называемого MonoBehaviour. Представить класс можно как своего рода план для создания нового типа компонента, который может быть прикреплен к игровому объекту. При создании нового файла можно увидеть две функции Update и Start, определенные внутри класса (см. рисунок 2.6). Функция Update - это место для размещения кода, который будет обрабатывать обновление кадра для игрового объекта. Это может быть движение, срабатывание действий и ответная реакция на ввод пользователя, в основном всё, что должно быть обработано с течением времени в игровом процессе. Функция Start будет вызвана Unity до начала игрового процесса (т.е. до первого вызова функции Update), и это идеальное место для выполнения инициализации переменных.

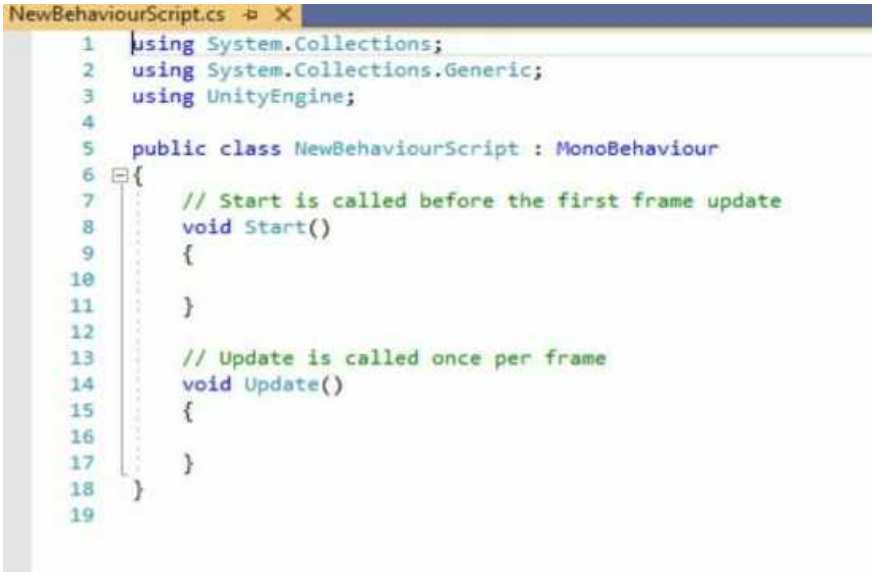


Рисунок 2.6

Класс PlayerController реализует движение объекта по двум векторам с использованием функции AddForce (см. рисунок 2.7). Эта функция добавляет силу Rigidbody. Сила прикладывается непрерывно в направлении вектора силы. Указание режима ForceMode позволяет изменить тип силы на ускорение, импульс или изменение скорости. Сила может быть применена только к активному телу. Если GameObject неактивен, AddForce не действует. Кроме того, Rigidbody не может быть кинематическим. Так же публичная переменная speed позволяет регулировать ускорение объекта прямо в инспекторе.

```
void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");

    Vector3 movement = new Vector3 (moveHorizontal, 0f, moveVertical);

    rb.AddForce(movement * speed);
}
```

Рисунок 2.7 Использование функции AddForce в коде

Также в сценарии PlayerController принимает две переменные speed и countToWin. Переменная speed отвечает за множитель ускорение объекта, а countToWin – за количество монет, которые надо поднять для победы (см. рисунок 2.8). Позже счётчик монет, что остались на карте, будет выводиться на игровой интерфейс.

```
public float speed = 12;
public int countToWin = 10;
private Text counter , winner;
private Rigidbody rb;
private int count = 0;

void Start()
{
    counter = GameObject.Find("CountText").GetComponent<Text>();
    winner = GameObject.Find("WinText").GetComponent<Text>();
    winner.text = "";
    rb = GetComponent<Rigidbody>();
    UpdateCount();
}
```

Рисунок 2.8 Инициализация переменных класса PlayerController

В методе OnTriggerEnter реализовано “поднятие” монеты, до которых игрок прикоснулся, а в методе UpdateCount счётчик в интерфейсе игрока обновляется, и он же проверяет, не победил ли игрок (см. рисунок 2.9).

```
void OnTriggerEnter (Collider other){
    if (other.tag == "Coin"){
      Destroy(other.gameObject);
      count++;
      UpdateCount();
    }
}

void UpdateCount(){
    counter.text = "Coins left: " + (countToWin - count).ToString();
    if (count == countToWin){
      winner.text = "You win!";
    }
}
```

Рисунок 2.9 Методы OnTriggerEnter и UpdateCount

Когда GameObject сталкивается с другим GameObject, Unity вызывает OnTriggerEnter. OnTriggerEnter происходит в функции FixedUpdate, когда два GameObject сталкиваются.

Класс CameraController принимает расстояние от камеры до игрового объекта и непосредственно положение самого объекта. Отступ, на котором будет расположена камера от координат объекта, называется offset. На координаты offset в функции LateUpdate будет перемещаться камера (см. рисунок 2.10). LateUpdate вызывается после того, как были вызваны все функции обновления. Это полезно для порядка выполнения скрипта. Например, камера слежения всегда должна быть реализована в LateUpdate, поскольку она отслеживает объекты, которые могли перемещаться внутри Update.

```
public class CameraController : MonoBehaviour
{
    public GameObject player;
    private Vector3 offset;

    void Start()
    {
        offset = transform.position - player.transform.position;
    }

    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

Рисунок 2.10 Реализация класса CameraController

Монеты на карте на данном этапе всё ещё статичны и никак не привлекают внимание игрока. Для этого был создан класс Rotate, в котором прописана простая анимация вращения через изменения значений поворота по осям в компоненте transform (см. рисунок 2.11). Она придаёт ощущения интерактивности с этим объектом.

```
public class Rotate : MonoBehaviour
{
    void Update()
    {
        transform.Rotate(new Vector3 (15,20,25) * Time.deltaTime);
    }
}
```

Рисунок 2.11 Реализация класс Rotate

Скрипт Teleport отвечает за логику телепорта. В этом классе были использованы корутины для реализации процесса “пересоздания” (см. рисунок 2.12). Корутины в Unity — простой и удобный способ запускать функции, которые должны работать параллельно в течение некоторого времени. Корутины представляют собой простые C# итераторы, возвращающие **IEnumerator** и использующие ключевое слово **yield**. В Unity корутины регистрируются и выполняются до первого **yield** с помощью метода **StartCoroutine**. Дальше Unity опрашивает зарегистрированные корутины после каждого вызова Update и перед вызовом LateUpdate, определяя по возвращаемому в yield значению, когда нужно переходить к следующему блоку кода. [2]

Метод Instatiate клонирует оригинал объекта и возвращает клон. Эта функция создает копию объекта аналогично команде «Дублировать» в редакторе. Так же при клонировании можно указать положение и поворот объекта.

Destroy удаляет игровой объект, компонент или актив. Объект уничтожается сразу после текущего цикла обновления или через t секунд, если указано время. Если переданное значение является Component, этот метод удаляет компонент из GameObject и уничтожает его. Если значение - GameObject, он уничтожает GameObject, все его компоненты и все дочерние элементы преобразования GameObject. Фактическое разрушение объекта всегда откладывается до завершения текущего цикла обновления, но всегда выполняется перед отрисовкой.

```
public class Teleport : MonoBehaviour
{
    public GameObject secondTeleport;
    public TeleportAnimator teleportAnimator;
    public TeleportAnimator secondTeleportAnimator;
    public GameObject prefab;
    private int count;

    public IEnumerator OnTriggerEnter(Collider other){
     if (other.tag == "Player"){
     secondTeleportAnimator.SetIsNear(true);
       yield return new WaitForSeconds(0.5f);
       GameObject gameObject = Instantiate(prefab, secondTeleport.transform.position + new Vector3 (0, 1f, 0), secondTeleport.transform.rotation);
       gameObject.GetComponent<Rigidbody>().AddForce(-secondTeleport.transform.forward * 250f);
       GameObject.Find("MainCamera").GetComponent<CameraController>().player = gameObject;
       count = other.gameObject.GetComponent<PlayerController>().GetCount();
       Destroy(other.gameObject);
       teleportAnimator.SetIsNear(false);
       yield return new WaitForSeconds(1);
       gameObject.AddComponent<PlayerController>();
       gameObject.GetComponent<PlayerController>().SetCount(count);
       gameObject.tag = "Player";
     }
    }
}
```

Рисунок 2.12 Класс Teleport

Для того, чтобы человек, недолго думая, понял, что с этим объектом делать, был добавлен простой скрипт TeleportAnimator с логикой анимации телепорта (см. рисунок 2.13).

Сценарий проверяет, не находится ли рядом с телепортом игрок, если да, то он запускает одну анимацию, если нет, то проигрывается другая.

```
public class TeleportAnimator : MonoBehaviour
{

    public Animator animator;

    public void Start(){
     animator = GetComponent<Animator>();
    }

    public void OnTriggerEnter(Collider other){
      if (other.tag == "Player"){
        SetIsNear(true);
      }
    }

    public void OnTriggerExit(Collider other){
      if (other.tag == "Player"){
        SetIsNear(false);
      }
    }

    public void SetIsNear(bool isNear){
      animator.SetBool("isNear", isNear);
    }
}
```

Рисунок 2.13 Класс TeleportAnimator

Необходимо также написать логику для меню и интерфейса игры. Класс MenuController содержит три метода, каждый из которых будет вызываться при нажатии нужной клавиши на интерфейсе (см. рисунок 2.14). PauseController отвечает за кнопки в панели паузы во время игры (см. рисунок 2.15). Переменная timeScale класса Time отвечает за масштаб, в котором проходит время. Это можно использовать для эффектов замедленного движения. Когда timeScale равен 1.0, время проходит так же быстро, как и в реальном времени. Когда timeScale равен 0,5, время проходит в 2 раза медленнее, чем в реальном времени. Когда timeScale установлен на ноль, то игра в основном приостанавливается, если все функции не зависят от частоты кадров. В данном случае игра останавливается в то время, когда активна панель паузы. [2]

Settings отвечает за панель настроек внутри игры (см. рисунок 2.16).

```
public class MenuController : MonoBehaviour
{
    public void PlayPressed(){
      SceneManager.LoadScene("Maze");
    }

    public void MenuPressed(){
      SceneManager.LoadScene("MainMenu");
    }

    public void ExitPressed(){
    Debug.Log("Exit pressed!");
      Application.Quit();
    }
}
```

Рисунок 2.14 Методы в классе MenuController

```
public class PauseController : MonoBehaviour
{
    public GameObject panel;
    public GameObject setting;

    void LateUpdate()
    {
        if(Input.GetKeyDown(KeyCode.Escape)){
          panel.SetActive(true);
          Time.timeScale = 0;
        }
        if(!panel.activeSelf && !setting.activeSelf){
          Time.timeScale = 1;
        }
    }

    public void SettingPressed(){
      setting.SetActive(true);
      panel.SetActive(false);
    }
}
```

Рисунок 2.15 Класс PauseController

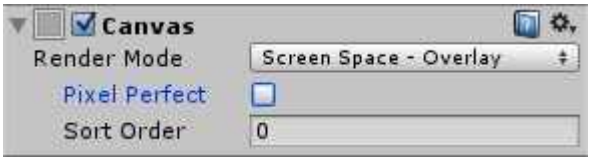
```
public class Settings : MonoBehaviour
{
    public Dropdown dropdawn;
    public AudioMixer audioMixer;
    private bool isFullScreen = true;
    Resolution[] resolutions;
    List<string> resolutionList;

    public void Awake(){
       resolutionList = new List<string>();
       resolutions = Screen.resolutions;
       foreach (var i in resolutions){
         resolutionList.Add(i.width + "x" + i.height);
       }
       dropdawn.ClearOptions();
       dropdawn.AddOptions(resolutionList);
    }

    public void FullScreenToggle(){
       isFullScreen = !isFullScreen;
       Screen.fullScreen = isFullScreen;
       Debug.Log(isFullScreen);
    }

    public void AudioVolume(float sliderValue){
       audioMixer.SetFloat("masterVolume", sliderValue);
    }

    public void Quality(int quality){
       QualitySettings.SetQualityLevel(quality);
    }

    public void Resolution(int res){
       Screen.SetResolution(resolutions[res].width, resolutions[res].height, isFullScreen);
    }
}
```

Рисунок 2.16 Класс Settings

2.3 Разработка пользовательского интерфейса и главного меню игры

Интерфейс — «общая граница» между отдельными системами, через которую они взаимодействуют; совокупность средств и правил, обеспечивающих взаимодействие

отдельных систем. В играх интерфейс находится не на последнем месте по значимости. Благодаря этому человек может легко и быстро получать важную информацию, делать быстрые решения и действия, что упрощает погружение.



Интерфейс в Unity представлен “холстом” и тем, что находится внутри него. Компонент Canvas представляет собой абстрактное пространство, в котором производится настройка и отрисовка UI. Все UI-элементы должны быть потомками игровых объектов, к которым присоединён Canvas. Сначала следует создать Canvas и настроить его отображение поверх других объектов в сцене, за это отвечает Render Mode.

Из важной информации, что находится на интерфейсе этой игры, - дочерние объекты CountText, WinText, RawImage, Pause и Settings (см. рисунок 2.17). CountText отображает количество оставшихся монет и является простым текстом на холсе. WinText уведомляет об окончании игры, тоже выражен текстом. RawImage является мини-картой в реальном времени, а также это 2D изображение с текстурой, которую передаёт камера над уровнем. Pause – это панель со своими объектами различных текстов (как и Settings), кнопок, флажков и выпадающих панелей, останавливает игру и позволяет открыть меню с настройками игры, Settings.

Рисунок 2.17 UI объекты

В Canvas можно настроить задний фон, в меню паузы это ни к чему, но в главном меню игры это играет важную роль, так как в новой сцене Menu нет никаких объектов помимо UI. Задний фон придаёт ощущение наполненности пространства на экране (см. рисунок 2.18).

Рисунок 2.18 Главное меню и меню настроек

Нажатие на каждую кнопку UI имеет свои действия, которые были заданы через Inspector при помощи скриптов. Так, например, кнопка “Exit” завершит работу программы, но для того, чтобы игра выключилась, её следует скомпилировать.

2.4 Компиляция проекта и финальные настройки игры

Финальным и одним из важных пунктов после создания игры являет компиляция проекта, без неё невозможно будет использовать приложение по назначению. Компиляция – это преобразование всех файлов исходного кода высокого уровня в эквивалентную программу на низком уровне. [3] Компиляция игры предусмотрена самим движком Unity по готовым сценариям для разных платформ. Для этого необходимо открыть панель File в верхнем левом углу и выбрать Build Settings (см. рисунок 2.19).

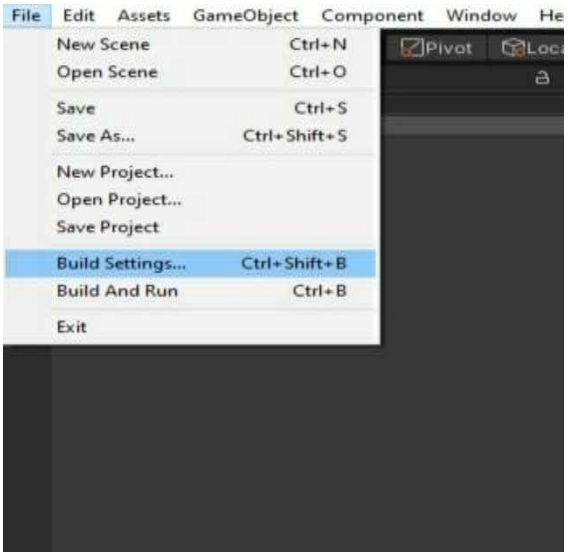


Рисунок 2.19 Контекстное меню File

Это необходимо, чтобы сделать последние настройки игры перед компиляцией проекта. [Приложение Б] В открытом окне Build Settings красным цветом отмечены сцены, которые будут включены в программу, синим - кнопка Add Open Scenes, что позволяет добавить недостающие сцены в необходимый пул, зелёным цветом отмечена самая популярный сценарий – универсальный (см. рисунок 2.20). Последним пунктом будет выбрать кнопку Build And Run, что на картинке отмечена фиолетовым, так игра пройдёт компиляцию и запуститься в новом окне. Жёлтым цветом отмечена кнопка Player Settings. При нажатии на кнопку Player Settings открывается новое окно, в котором можно делать точечные настройки (см. рисунок 2.21)

- Audio позволяет работать со звуковыми настройками.
- Editor позволяет настраивать компоненты игры, что связаны с тенями или их методами просчёта.
- Graphics позволяет настроить графику камеры.
- Input Manager позволяет задать параметры ввода.
- Package Manager позволяет предоставить материал по ссылке.
- Physics и Physics 2D позволяют настроить переменные физики движка.
- Player позволяет настроить такие параметры как: спрайт курсора, спрайт ярлыка приложения, лого при запуске, рендеринг и свет.
- Preset Manager позволяет добавить пресеты в игру.
- Quality позволяет добавить уровни детализации текстур и освещения, а также отвечает за качество картинки.
- Script Execution Order позволяет настроить переменные взаимодействия сценариев.

- Tags and Layers позволяет добавлять и изменять теги, слои.
- TextMesh Pro позволяет добавить новый шрифт в проект.
- Time позволяет регулировать время благодаря переменным.
- VFX позволяет добавлять визуальные эффекты.
- XR Plugin Management предоставляет управление и предлагает помощь с загрузкой, инициализацией, настройками и поддержкой сборки для подключаемых модулей XR.

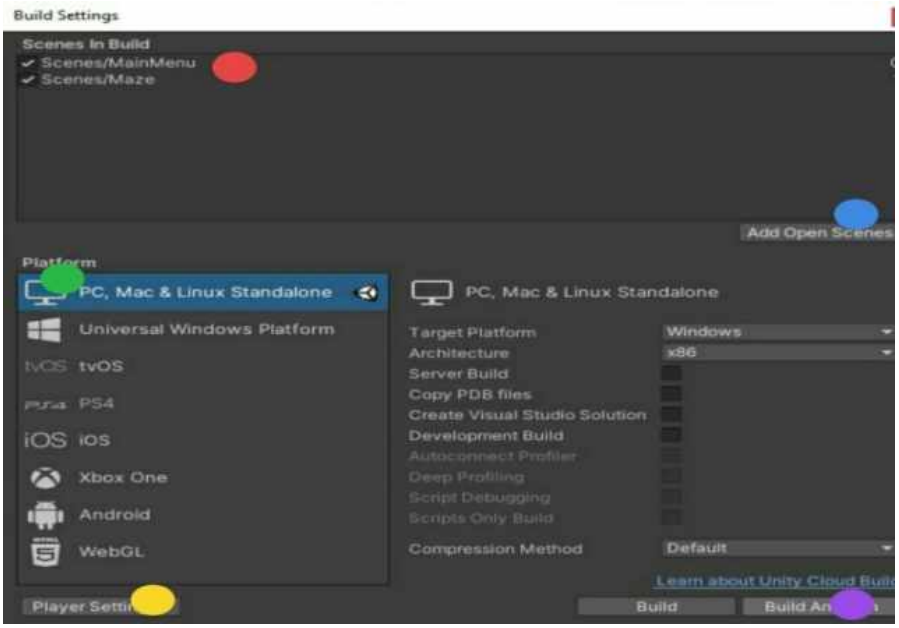


Рисунок 2.20 Окно Build Settings

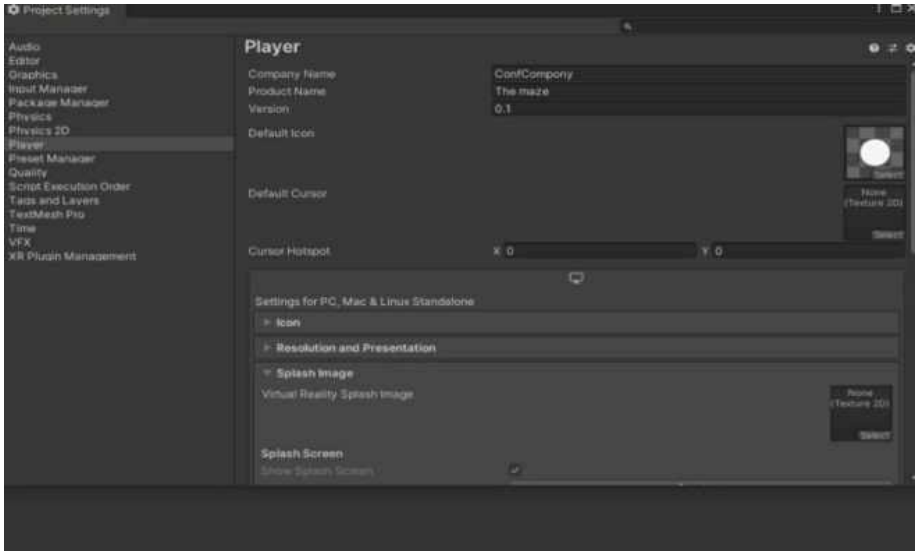


Рисунок 2.21 Окно настроек проекта

1. Во вкладке Player, под пунктом Default Icon была изменена иконка приложения.
2. Во вкладке Company Name задано название ConfCompony.
3. Во вкладке Product Name задано название The maze.
4. Добавлена загрузка лога игры. [Приложение А]

3 ПСИХОЛОГИЧЕСКИЙ АНАЛИЗ ИСПЫТУЕМЫХ

Было отобрано 10 человек разных возрастных категорий для проведения опыта:

1. Мальчик 8 лет.
2. Девочка 12 лет.
3. Парень 15 лет.
4. Парень 16 лет.
5. Девушка 16 лет.
6. Мужчина 31 года.
7. Мужчина 32 лет.
8. Мужчина 39 лет.
9. Женщина 38 лет.
10. Мужчина 51 года.

Первая подгруппа людей относится к молодой аудитории (№ 1, 2). Вторая подгруппа является подростками (№ 3, 4, 5). Третья – люди средних лет (6, 7, 8, 9). Четвёртая - люди в возрасте (№ 10).

Каждый новый опыт прохождения игры проходил под контролем и чётко документировался, все добровольцы были в равных условиях и не знали об эксперименте. Никто из добровольцев также не знал целей и задач игры. Было проверено несколько вариантов расположения монет, телепортов. Каждому после прохождения первого уровня игры предоставлялась возможность оставить включённой миникарту или же отключить её.

Первая подгруппа проявила высокую заинтересованность в опыте:

1. Наблюдалась чрезмерная активность у девочки под № 2.
2. После прохождения уровней люди дети задавали большое количество вопросов.
3. Отмечены попытки креативных подходов к достижению целей как непосредственно игровых, так и надуманных, собственных.
4. Мальчик под № 1 сам понял, что делать в игре. Задавал меньше других вопросов, так как игра его заинтересовала, но он почти всё время молчал, это явный признак налаживания когнитивного потока.
5. Передвижения игрового объекта испытуемыми были небрежные, шар часто сталкивался со стенами лабиринта, дети не контролировали импульс при разгоне шарика.
6. Проходили одни и те же уровни несколько раз, пытаясь найти «секреты».
7. Девочка под номером №2 пыталась «забраться» на телепорт, так как подумала, что «он может поднять её наверх».

Вторая подгруппа сразу поняла цель:

1. Было отмечено нетипичное действие, девочка под номером № 5 долго пыталась разобраться с управлением и понять «смысл» игры, но в итоге интерес получить ответы на свои вопросы затягивал испытуемую. Единственное, что её заинтересовало с самого начала – это подбор крутящихся монет.
2. Для всех участников данной группы сразу была понятна механика телепорта.
3. После завершения опыта каждый посчитал нужным оставить свой отзыв об игре. Обратная связь была как позитивной, так и негативной, но при этом комментарии были объективны.

Третья подгруппа людей средних лет отнеслась серьёзно к своей задаче:

1. У подгруппы было сильное желание узнать «чёткие границы» или «правила игры» перед началом опыта - что делать можно, а что нельзя.
2. Время прохождения одного уровня с картой и без карты у подгруппы увеличилось, также увеличилось количество получаемой информации.

3. Значительно уменьшилось количество столкновений со стенами лабиринта, исключена проблема с торможением.
4. Мужчина № 6 подбору настройки графики и разрешение экрана уделил большое внимание, также в первую очередь, перед нажатием кнопки «Play», его интересовал интерфейс, меню и панель настройки.
5. Женщина под № 9 показала противоположный результат в отличии от мужчины под № 6. Она нуждалась в подсказках и не знала, что в играх по умолчанию передвижение игрока задаётся клавишами «WASD» или стрелками в правой нижней части клавиатуры.

Четвертая подгруппа показала самые слабые результаты опыта в сравнении с предыдущими группами волонтёров:

1. Мужчина под № 10 самостоятельно выполнил задачи лишь первого уровня, а во втором уровне была оказана помощь и руководство по продвижению, так как управление в игре было «непонятно».
2. Передвижение шара было неровное, большое количество столкновений со стенами лабиринта замедлило прохождение уровня. Следовательно, проблема с управлением являлась главным препятствием на пути завершения конечной задачи.

В заключение анализа можно с полной уверенностью заявить, что игра, созданная в ходе работы, отлично подходит для первых двух групп людей. Когнитивный поток у людей школьных возрастов сильно отличался от взрослых людей, закончивших школу, в лучшую сторону, а это значит, что игру следует специализировать и дорабатывать, ссылаясь на ожидания детей и их вовлечённость. Из-за небольшой целевой нагрузки в игре начала пробуждаться фантазия, а она является сильнейшим инструментом для сохранения прочного когнитивного потока и интереса к игре. Следующим действием в разработке игры будет её улучшение и повышения качества продукта.

Испытуемые желали ухватиться за любую информацию, что может быть полезной или даст преимущество в будущих испытаниях. По статистике полученной информации из проведённого опыта можно предложить, что целевая аудитория данной игры – это дети от 7 до 14 лет. Это означает, что когнитивный поток у данной группы ярко выраженный, следовательно, время, проводимое в игре, может приблизиться к возможному максимуму. А будет ли человек вспоминать игру и хотеть сыграть в неё еще раз, напрямую зависит от проведённого в ней времени. У детей фантазия навязывает свой игровой азарт, который вызывает позитивные чувства после окончания игры. Это следует использовать в своих наработках.

Взрослые люди из всех остальных групп не подходят под целевую аудиторию, потому что зрелый человек привык анализировать все “за” и “против”. Много игр в жанре головоломок не позволяет игроку видеть всю информацию сразу. Игроки предпочитают получать так много информации, что они не помещаются в их поле зрения. Вернее, всё-таки можно уместить, но для этого придется ее так сильно сжать, что станет невозможно различить, что на ней происходит. Образ получения информации окажется настолько маленьким, что его будет сложно понять. И этим стремлением можно управлять. Для этого и создаются интерактивные подсказки, панели, задачи или лабиринты. В начале игры почти всё для игрока – «черный ящик». Некоторую информацию о взаимосвязях дают описание и система подсказок. В основном игрок строит предположения, исходя из здравого смысла. Иногда они оправдываются, иногда – нет. И это желание получать всё больше и больше информации движет по сюжету игры, заставляя тратить больше времени для решения различных задач и формирует когнитивный поток. А это в свою очередь доставляет моральное удовольствие от их решения, затягивая человека.

Свобода действий это хорошо, но если бы никто не знал, куда нужно направляться, навигация по игровому миру отсутствовала бы, то вряд ли игроки были бы этому рады, а

мотивация играть канула бы в небытие. Миникарты выступают в качестве большой подсказки, которая помогает игроку до самого конца своей игры. На первом уровне игры она обхватывает почти всю карту и является большой подсказкой, а на втором уровне – лишь небольшую зону вокруг игрока. Угол обзора и ракурс находятся не на последнем месте. Так человек подсознательно видит через небольшие стены вращающиеся монеты и на этом строит своё продвижение, но он не осознаёт, что такое облегчение игрового процесса создано специально. Игра не должна быть полностью честной, лёгкой или до боли сложной. Хорошая игра постоянно бросает вызов игроку, но не позволяет ему утонуть в проблемах. Главное – не показывать человеку того, что игра постоянно ему поддаётся.

Поэтому по карте были разбросаны небольшие «мотиваторы» - монеты и телепорты. Во второй части лабиринта из монет аккуратно проложен путь от одной части карты в другую, заворачивая и пропуская игрока по каждому углу, с преодолением непроходимых препятствий помогает телепорт. Монеты выступают в роли небольших задач (добраться до самой монеты) и вознаграждений (добавление счётчика в игре), которые заставляют двигаться всё дальше, не теряя интерес к игре, так же если игрок случайно заберётся в тупик, он обязательно надёт монету и посчитает, что сделал всё верно, а не «налажал». Складывается невольное ощущение возможности игроку прийти к конечному результату свои путём, хоть игра и линейная. Но наш мозг готов пойти на мизерные риски без серьёзных последствий ради получения больших эмоций, поэтому это работает. Красные цвета дают подсознательную информацию, что его касаться не следует, а зелёный - наоборот, поэтому стены стилизованы под красный оттенок, а дорога в лабиринте зелёного цвета.

Следует отметить, что люди, незнакомые с играми, не могут знать неформальные обозначения, такие как вращающаяся монета, поднимающийся телепорт или шар, который обязательно должен катиться. К неформальным обозначениям следует приписать и стандартные клавиши управления или главные принципы игр (проблема → решение → вознаграждение).

ЗАКЛЮЧЕНИЕ

Целью данной работы было создание простой программы, которая послужила в роли инструмента для исследования человеческого мышления, его подсознательные решения и действия при определённых условиях, образы восприятия, реакцию на получаемую информацию, а также анализ полученных данных. В результате психологического анализа была выявлена более подходящая аудитория для игры и исследования, а также были описаны самые главные аспекты по созданию и проектированию игр.

В работе были поставлены задачи: разработать, изучить и усвоить полученную информацию по работе в сфере разработки игр. Разработанная мною игра может быть использована для детей школьного возраста (7-14 лет). Она позволяет детям мыслить шире, развивает фантазию и логическое мышление. Данная работа может послужить материалом для понимания психологии человека одним из самых эффективных способов - при помощи компьютерных программ и игр.

Тем не менее, игры имеют большое влияние на человеческое общество. Они занимают одну из лидирующих позиций на международной арене в сфере развлечений. Как и любая потенциальна коммерческая идея, влияние игры на человеческое общество может послужить росту частного бизнеса, иногда и государственного бюджета. Благодаря полученным знаниям и анализам работы всегда можно сделать успешный, креативный и правильный проект по типу «трипл-эй» игр, ориентируюсь на ожидания людей, реализуя те желания, о работе которых потенциальный игрок даже и не подозревает.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity - Manual: Unity User Manual 2020.3 (LTS)[Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/Manual/index.html. – Дата доступа: 03.01.2021.
2. Unity - Scripting API [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ScriptReference/index.html. – Дата доступа: 15.01.2021.
3. Unity (игровой движок) — Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA). – Дата доступа: 04.01.2021.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Скомпилированный проект - https://github.com/Vladislav-2/TheMaze;

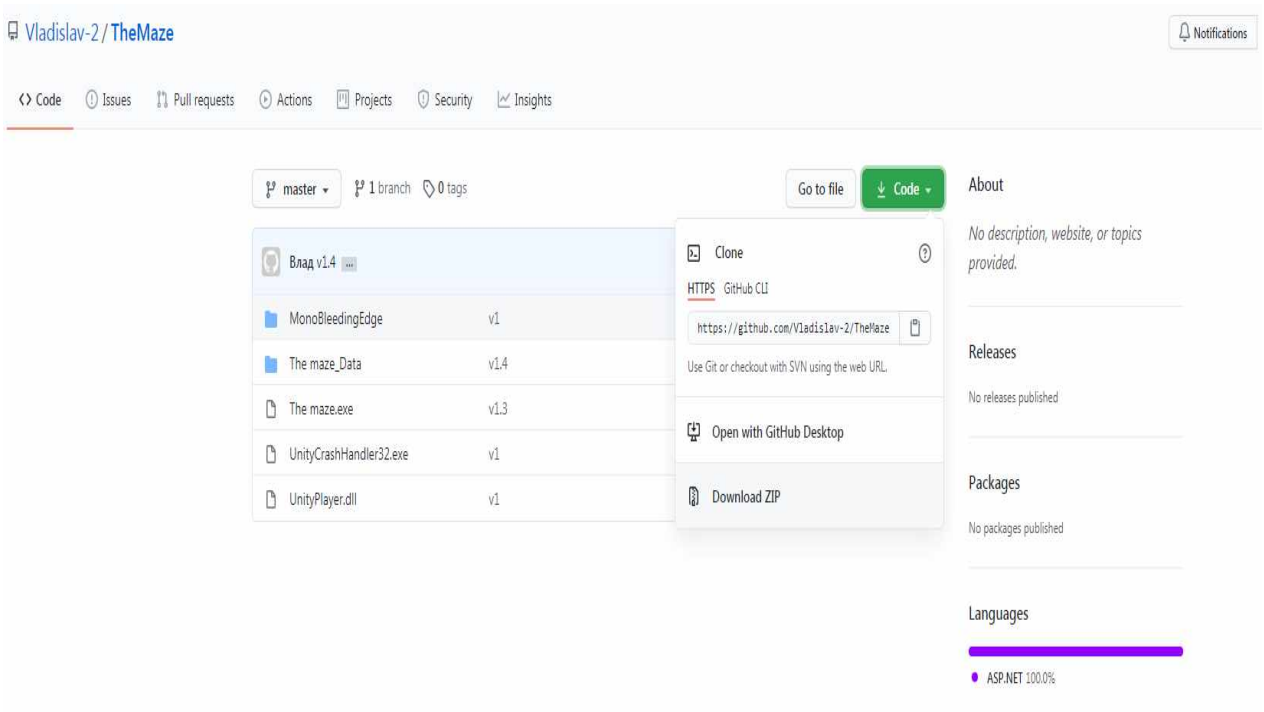


Рисунок А.1

Исходный код - https://github.com/Vladislav-2/TheMazeSource;

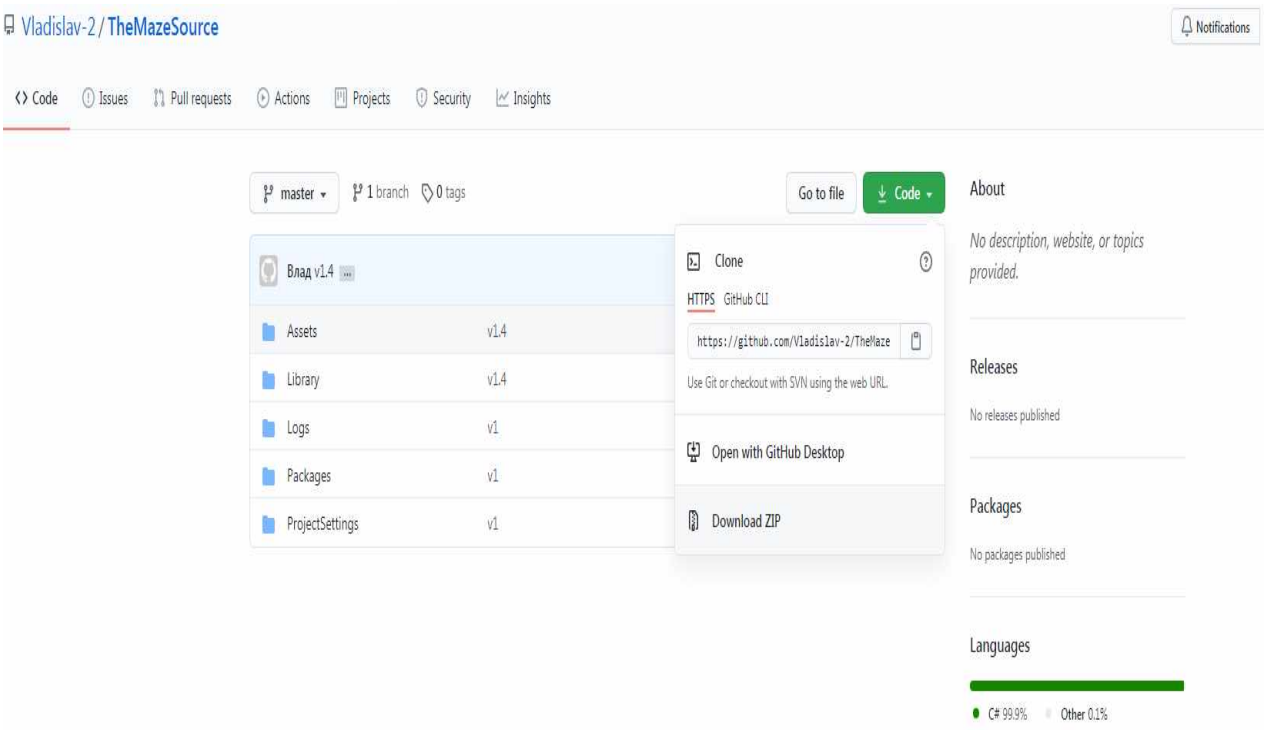


Рисунок Б.1