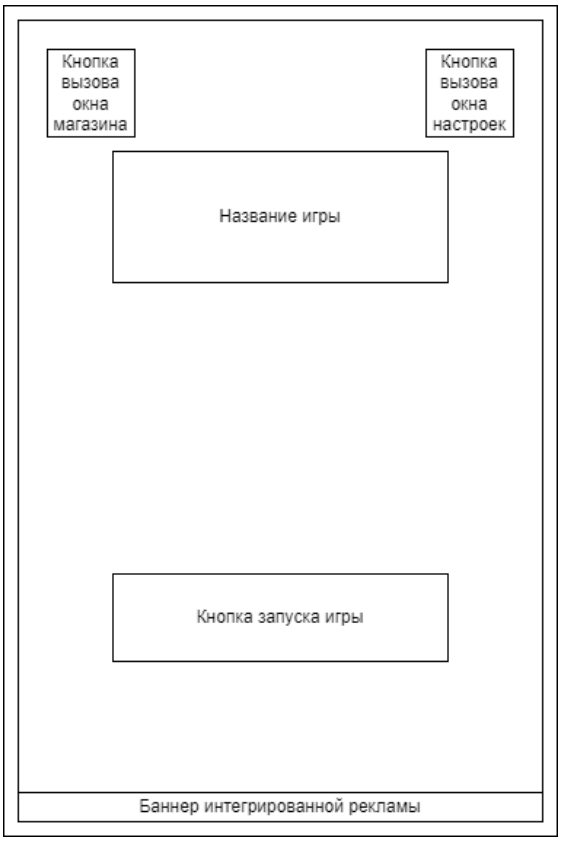
Разработка базового расписания

1) Прототипы экранных форм

- **Прототип главного экрана меню**

  Главное окно, которое отображается при входе в приложение, служит стартовым экраном откуда можно начать саму игру (кнопка запуска игры), войти в магазин (кнопка вызова окна магазина), в настройки приложения (кнопка вызова настроек). Помимо этого, на экране присутствует тайтл игры и перманентный блок под баннер интегрированной рекламы, который активен на протяжении всей игровой сессии.

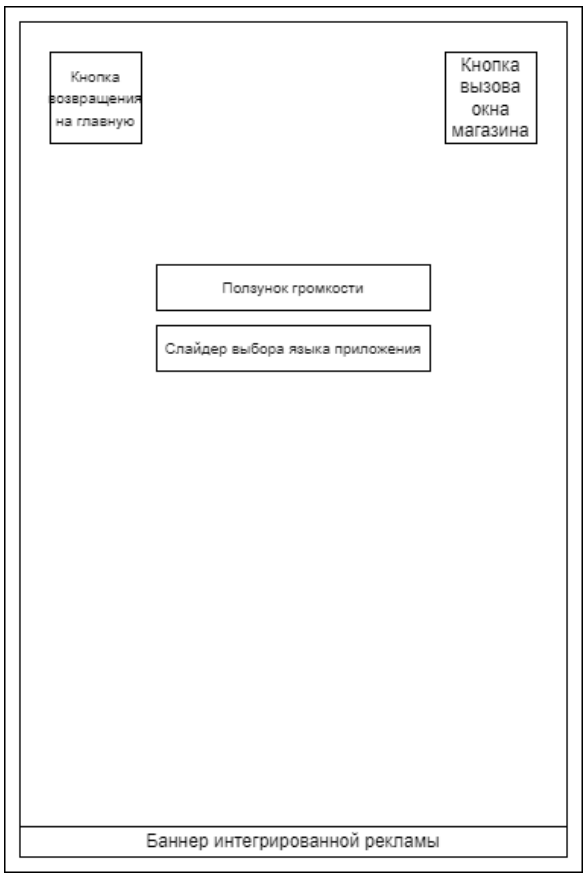


- **Прототип экрана основного игрового процесса**

  Игровая сцена, которая запускается из главного меню, содержит игровое интерактивное поле, на котором происходит весь основной геймплей, кнопки возвращения в главное меню, кнопка вызова паузы, которая в свою очередь приостанавливает игровой процесс и открывает поверх текущего окна новое. Шкала прогресса отображает относительную часть завершения текущего уровня и заполняется постепенно в зависимости от количества верных действий игрока. Количество валюты отображает наличие денежного ресурса игрока и сохраняется между уровнями.

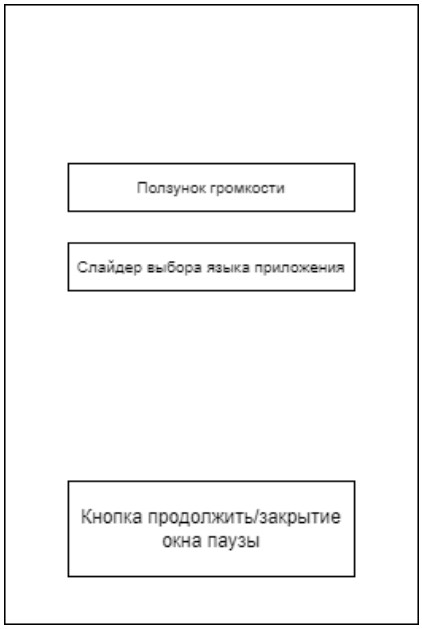
- **Прототип окна настроек игры, запущенного из главного экрана**

  Окно настроек, в котором игрок может изменить уровень громкости или поменять язык, также может выйти обратно на главный экран или перейти в магазин.



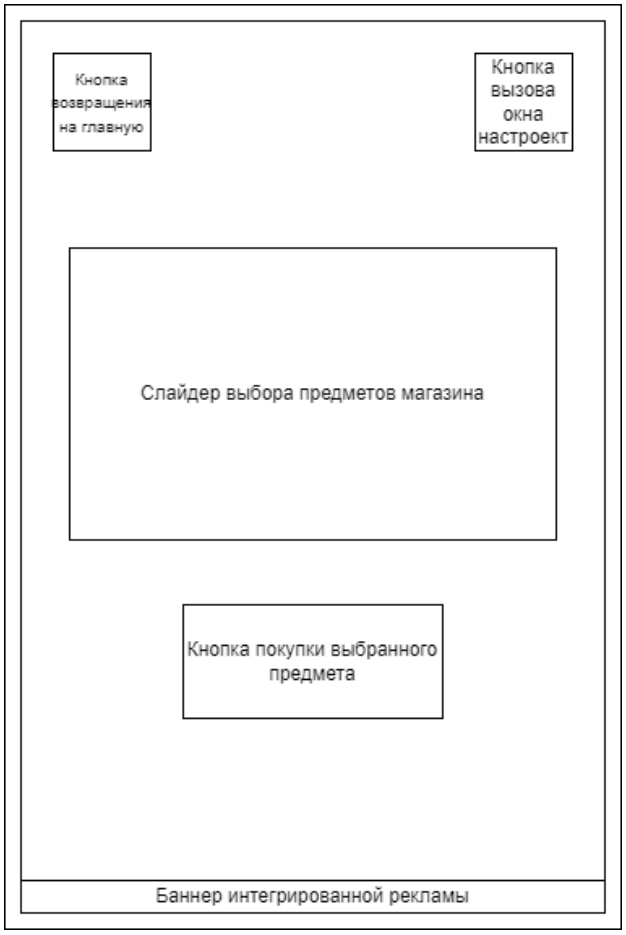
- **Прототип настроек игры (окно паузы), запущенного во время игры**

  Окно паузы/настроек, вызванное из процесса игры, имеет урезанный функционал основного окна настроек без переходов на другие окна, можно настроить громкость и язык или выйти и продолжить игровой процесс.



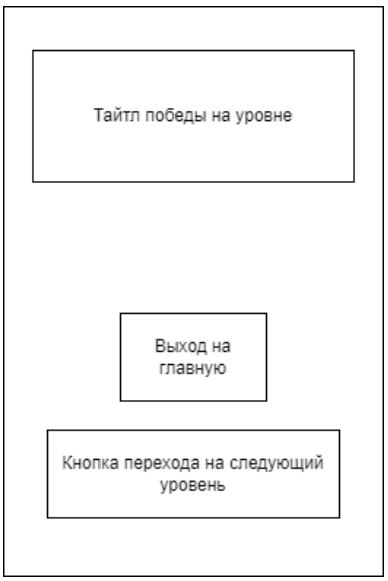
- **Прототип окна магазина**

Окно магазина, в котором можно с помощью слайдера выбрать предмет и купить его за валюту игрока кнопкой ниже, также можно перейти либо назад на главную, либо в меню настроек.



- **Прототип окна выигрыша**

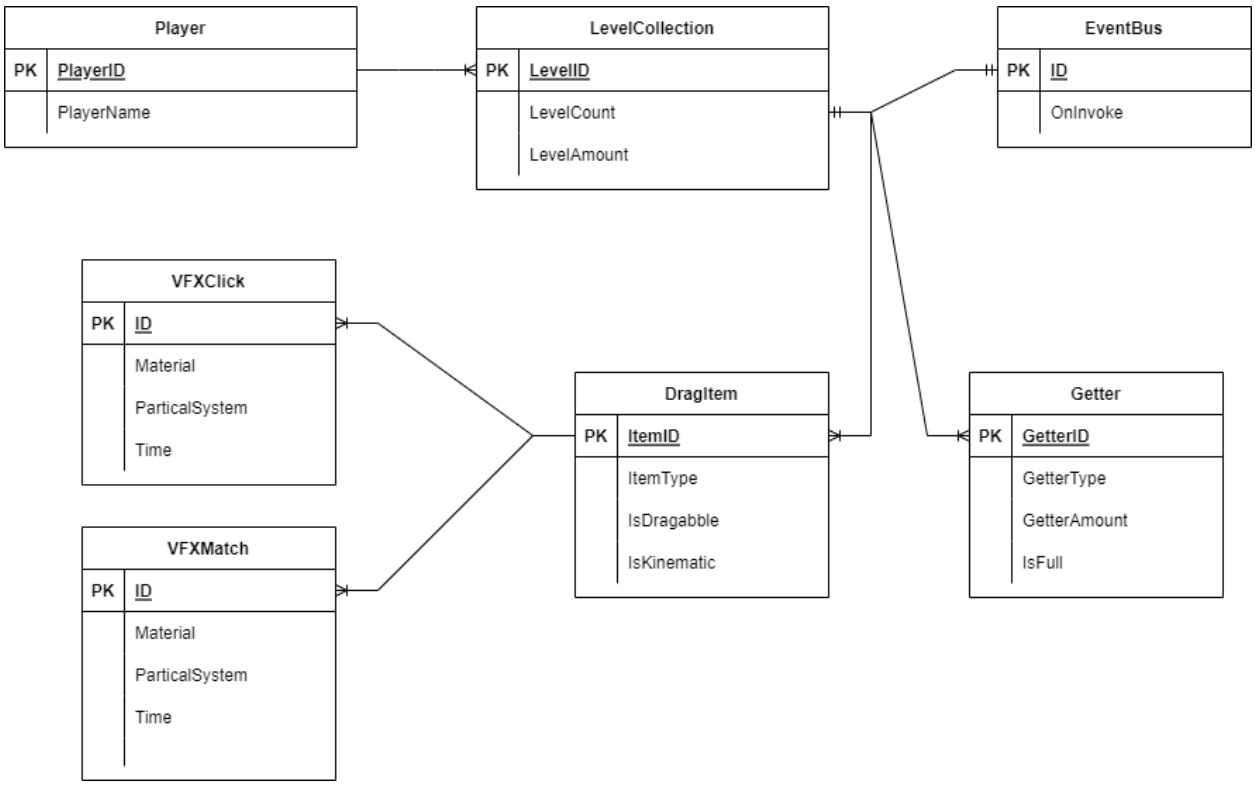
  Окно выигрыша, которое открывается поверх игрового экрана в тот момент, когда игрок успешно завершил уровень, ему высвечивается соответствующая надпись о успехе, далее он может либо выйти в главное меню, либо перейти на следующий уровень



- **Прототип окна проигрыша**

  Схожее с предыдущим окно за исключением того, что высвечивается оно при неудаче и кнопка уже может отправить на перепрохождение проваленного уровня.

2) Диаграмма сущностей



3) Разработка api системы

- **CanvasSwitcher**

Функция предназначена для переключения канвасов пользовательского интерфейса для различных ситуаций (основное меню, окно победы, поражения, и т.д.). Функция переключает компонент SetActive, который отвечает за отображение игрового объекта на камере.

Входные данные: массив объектов типа Canvas, который содержит в себе всевозможные канвасы, которые встречаются по ходу игры.

Выходных данные: определенному канвасу изменяется булевый компонент SetActive.

- **LevelChanger**

Функция предназначена для переключения уровней по их завершению, она уничтожает предыдущий экземпляр уровня и создает экземпляр следующего. Срабатывание функции вызывают встроенные ивенты OnClick, которые вызываются по нажатию.

Входные данные: список префабов (объект, который хранит в себе множество игровых объектов и их компонентов) уровней, номер уровня типа данных int.

Выходные данные: с игрового поля удаляется экземпляр предыдущего объекта типа GameObject и передается экземпляр следующего.

- **DragItem**

Функция, которая отвечает за свойство перемещения игроком игровых объектов. Она обрабатывает конкретный объект, который был взят игроком и при помощи встроенных ивентов Unity, присваивает ему свойства и отключает их (например: при взятии отключение своевольного перемещения, отключение физики, при отпуске объекта эти свойства обратно включаются).

Входные данные: экземпляр объекта типа GameObject, его компонент Rigidbody, классификация объекта (пользовательский тип данных TypeItem)

Выходные данные: выбранному объекту типа GameObject передаются новые значения их свойств.

- **ScoreHandler**

Функция отвечает за обрабатывание текущего прогресса на уровне, она отслеживает, сколько предметов было отсортировано на каждом контейнере и на каждый добавляет очки прогресса, значение которых потом отправляет на визуальный прогресс бар. Также она следит за тем, когда достаточное количество предметов было помещено в каждый контейнер, когда все они заполнены вызывает ивент для вызова завершения уровня.

Входные данные: массив пользовательского типа данных Getters, который состоит из всех типов контейнеров.

Выходные данные: на визуальный объект, который висит на игровом канвасе передается значение счета типа int, которое преобразуется в визуальное отображение прогресса.

- **ItemSpawner**

Функция предназначена для создания экземпляров объектов типа GameObject на игровом поле, которая рандомно распределяет их по площади.

Входные данные: префаб типа данных GameObject, количество предметов, которые нужно создать типа int, и площадь, внутри которой будет размещение объектов типа данных Vector3.

Выходные данные: все экземпляры объектов типа GameObject, которые нужно создать на игровом поле

- **ItemGetter**

Функция отвечает за обработку попадания в контейнер объектов, при срабатывании триггера вызывается ивент, который уничтожает экземпляр объекта, который попал внутрь.

Входные данные: пользовательские типы данных ItemType – тип предмета, который попал в контейнер, GetterType – тип контейнера.

Выходные данные: количество предметов типа int, пройденных через контейнеры передается в функцию ScoreHandler.

- **FlowingItems**

Функция отвечает за перемещение игровых объектов по реке, им придается физическая сила, которая их толкает в определенном направлении, имитируя поток реки.

Входные данные: точка перемещения, к которой стремятся объекты типа Transform, скорость перемещения типа float.

Выходные данные: на все экземпляры игровых объектов передается физическое воздействие.

- **ItemClickFX**

Функция отвечает за отображение визуального эффекта при клике на предмет. Создает вокруг объекта окружность, которая сигнализирует о поднятии объекта и показывает какой конкретно был взят.

Входные данные: шаблонная система частиц встроенного типа ParticleSystem

Выходные данные: на игровом поле создается экземпляр объекта системы частиц, после небольшого времени экземпляр удаляется.

- **ItemHideFX**

Функция отвечает за отображение визуального эффекта при помещении предмета в контейнер. Запускает частицы в виде небольшого конфетти в зоне контейнера и показывает, что предмет был помещен в верный контейнер.

Входные данные: шаблонная система частиц встроенного типа ParticleSystem

Выходные данные: на игровом поле создается экземпляр объекта системы частиц, после небольшого времени экземпляр удаляется.

- **ItemTrail**

Функция отвечает за отображение визуального эффекта при перетаскивании предмета. Создает за перетаскиваемым объектом утихающий шлейф.

Входные данные: шаблонная система частиц встроенного типа ParticleSystem

Выходные данные: на игровом поле создается экземпляр объекта системы частиц, после небольшого времени экземпляр удаляется.

4) Иерархическая структура работ

1. Разработка технического задания
    1.1. Сбор требований;
    1.2. Определение стадий и этапов разработки
        1.2.1. Определение стадий разработки;
        1.2.2. Определение сроков разработки;
    1.3. Общее описание
        1.3.1. Назначение продукта;
        1.3.2. Взаимодействие продукта;
        1.3.3. Допущения и ограничения продукта;
        1.3.4. Определение функций продукта;
2. Разработка приложения
    2.1. Backend-разработка
        2.1.1. Проектирование базы данных
            2.1.1.1. Определение структуры базы данных;
            2.1.1.2. Определение связей между сущностями;
            2.1.1.3. Определение взаимодействия с базой данных;
        2.1.2. Разработка API приложения
            2.1.2.1. CanvasSwitcher;
            2.1.2.2. LevelChanger;
            2.1.2.3. DragItem;
            2.1.2.4. ScoreHandler;
            2.1.2.5. ItemSpawner;
            2.1.2.6. ItemGetter;
            2.1.2.7. FlowingItems;
            2.1.2.8. ItemClickFX;
            2.1.2.9. ItemHideFX;
            2.1.2.10. ItemTrail;
        2.1.3. Сетевое взаимодействие
            2.1.3.1. Определение протокола взаимодействия;
            2.1.3.2. Обеспечение защищённости соединения;
            2.1.3.3. Определение местоположения хранения данных;
        2.1.4. Взаимодействие с UI
            2.1.4.1. Определение модели привязки данных;
            2.1.4.2. Создание механизма обновления данных;
    2.2. Frontend-разработка
        2.2.1. Дизайн
            2.2.1.1. Разработка макетов окон
                2.2.1.1.1. Разработка макета главного окна;
                2.2.1.1.2. Разработка макета игровой сцены;
                2.2.1.1.3. Разработка макета окна настроек;
                2.2.1.1.4. Разработка макета окна магазина;
                2.2.1.1.5. Разработка макета победного окна;

2.2.1.1.6. Разработка макета проигрышного окна;
2.2.1.1.7. Разработка макета окна паузы;
2.2.1.2. Взаимодействие с backend
2.2.1.2.1. Определение модели привязки данных;
2.2.1.2.2. Настройка механизма обновления данных;
2.2.1.2.3. Определение динамического взаимодействия с данными;
2.2.1.3. Разработка общего стиль-кода приложения
2.2.1.3.1. Создание логотипа приложения;
2.2.1.3.2. Создание иконок элементов;
2.2.1.3.3. Определение стиля и размера шрифтов;

3. Приемо-сдаточные испытания
   3.1. Подготовка и проведение демонстрации;
   3.2. Проведение испытаний;
4. Размещение приложения
   4.1. Согласование с издателем;
   4.2. Развёртывание приложения;
   4.3. Размещение в магазине приложений;
5. Поддержка приложения
   5.1. Мониторинг работоспособности;
   5.2. Получение и обработка обратной связи;
   5.3. Улучшение работы приложения
   5.4. Добавление новой функциональности;

5) Оценка время выполнения проекта по методу PERT

Работы	Количество	Оптимистичные трудозатраты	Пессимистичные трудозатраты	Наиболее вероятные трудозатраты
Создание сущностей	7	2	9	3
Создание макетов	7	15	60	30
Создание методов API	10	40	70	55

Посчитаем средние трудозатраты по каждой работе:

$$\text{Создание сущностей} = \frac{9 + 4 \times 3 + 2}{6} = 3.8 \text{ чел.} * \text{ час.}$$

$$\text{Создание макетов} = \frac{60 + 4 \times 30 + 15}{6} = 32.5 \text{ чел.} * \text{ час.}$$

$$\text{Создание методов } API = \frac{70 + 4 \times 55 + 40}{6} = 55 \text{ чел.} * \text{ час.}$$

Посчитаем среднеквадратичную оценку:

$$\text{Создание сущностей} = \frac{9 - 2}{6} = 1.67 \text{ чел.} * \text{ час.}$$

$$\text{Создание макетов} = \frac{60 - 15}{6} = 7.5 \text{ чел.} * \text{ час.}$$

$$\text{Создание методов } API = \frac{70 - 40}{6} = 5 \text{ чел.} * \text{ час.}$$

Посчитаем $\text{Е}_{\text{общ.}}$:

$$\text{Е}_{\text{общ.}} = 7 * 3.8 + 7 * 32.5 + 10 * 55 = 804.1 \text{ чел.} * \text{ час.}$$

Посчитаем $\text{СКО}_{\text{общ.}}$:

$$\text{СКО}_{\text{общ.}} = \sqrt{7 * 1.67^2 + 7 * 7.5^2 + 10 * 5^2} = 25.8 \text{ чел.} * \text{ час.}$$

Оценка суммарной трудоёмкости проекта с вероятностью 95%:

$$\text{Е}_{95\%} = 804.1 \ + 2 * 25.8 = 855.7 \text{ чел.} * \text{ час.}$$