Краткое содержание для ориентации по проекту:

1. Задействованные **библиотеки**: Zenject, UniTask, UniRx(заменямо), LeanPool, EasyButtons

2. Общая архитектура на основе **ServiceHolder** с прокидываемыми сервисами (на данный момент такие как **Input, Progress, AssetsProvider**.
Доступ через DI (**Zenject**), возможен переход на синглтоны для более простой работы, но zenject предлагает удобные сигналы (SignalsContainer) для общего геймплей флоу (впринципе заменямо также кастомным мессенджером или с коробки от UniRx (MessageBroker).

3. Уровни:
LevelData (id и сетка блоков BlocksData);
SectionDataContainer (**Section** - логический блок или локация, содержащая левел даты, пример: локация лето: внутри уровни 1 2 3 4, локация зима: внутри уровни 1 2 3..);
SectionsContainer - конфиг для выставления порядка локаций, хранится в Resources/Configs

Создаются через **Window-LevelEditor**, небольшой и примитивный редактор, выбирается тип блока соотвествующий Енаму блока **BlockType** (красный-огонь и т.д.), выбирается уровень (добавляется удаляется копируется), также выбирается локация Section.
Итого: сохраняются файлы в **Json** в Resources/Data по логическим блокам, по итогу с будущей возможностью подгрузкой его с сервера и рантайм обновления у игроков. Основной тип данных для блоков и сеток: **2д массив.**

4. Сохранения:
ProgressService, для левелов отдельный **LevelsProgressHandler**;
LevelProgressData, содержащие пока что булевую Completed и Level id;

SectionProgressDataContainer (те самые локации, содержащие прогресс даты, и енам **LevelSection**);

Сохранения через **PlayerPrefs**, ключ - Section, значение - тот самый контейнер. Не создаем сохранения всех секций сразу (игрок проходит локацию - открывается новая, и создается новый ключ с новым прогрессом). Также идёт сохранение последнего открытого уровня, отдельный тип данных **LevelProgressDataExtended** хранится в отдельном ключе, и содержит преимущественно сетку с блоками. Обновляется через **zenject сигналы**.

5. Получение данных:
AssetsProviderService - методы для загрузки json файлов и ресурсов из Resources по нужному пути, для едитора используется идентичная система EditorResourcesManager. Названия путей расположены в классе **Constants** (важные игровые константы также расположены там)

6. Основной геймплей флоу:
Bootstrapper сцена с инициализацией **ServiceHolder** и как холодная загрузка **GameplayScene**. На игровой сцене расположены сразу MonoBehavior обьекты:
- **GameplayHandler** принимает управление, выбирает необходимый уровень, выдает сигнал LevelStartedSignal с данными.
- **GameFieldCreator** от сигнала создает необходимую сетку и заполняет ее блоками с помощью прокинутых классов и выравнивает по экрану.
- **GridController** - главная логика, содержит 2д массив **GridCell** (статичных ячеек), ячейки принимают **Block** классы. Логика контроллера расставлена по комментариям KEYPOINT, вкратце: свайп жест - свап блоков - проверка на падение - поиск совпадений - уничтожение блоков - заново до тех пор пока есть чему падать. Поиск совпадений через поиск в ширину (пародия на алгоритм заливки), можно через закоменченную рекурсию (на всякий случай оставил, правда нужно подогнать ее под систему).

Ожидания анимаций через UniTask.WhenAll с ожиданием всех тасков, во время анимации логически ставится **IsBusy** булевая, исключающая клетку из всех операций, но все остальные клетки можно дергать.
- Кол-во клеток для уничтожения в **Constants**
- Конфиг для блоков - прокинут в **BlocksSpawner**, находится по пути Configs/BlocksConfig. Там же находится аниматор:
каждый блок имеет свой оверрайдед аниматор по пути Animations, для создания некоторых анимаций используется класс **AnimationCreator**, который принимает спрайты и распределяет их равномерно по фпс (нам нужно получить анимации уничтожения идентичной длительности, для приятного вида).

7. Летающие шары:
BalloonsController - выпускает первые N с задержкой, по колбеку захода за экран - спаунит новые, итого всегда N на экране. Спаун через **LeanPool**.
BalloonsSpawner - берет рандомный спрайт (тип шара), создает Bounds для вмещения в текущий экран, и спаунит за его пределами.
BalloonsConfig - находится в Configs, содержит спрайты шаров и настройки анимации.
Balloon - сама сущность, принимает в себя данные инициализации, ждет пока булевая canMove, и в апдейте двигается к точке назначения. Синус движение через mathf sin, настройки там же в конфиге.

8. UI:
MainCanvas - временный канвас с кнопками вызывающими рестарт или ресет левела (временный потому что в идеале в архитектуре будет DialogService спаунящий скрины или диалоги в какой то контейнер, но здесь не было надобности).