Quiz выбрать цифру/букву по заданию

Критерии оценивания

1. Качество кода. Включает в себя грамотность применения знаний разработки в рамках ООП/КОП, а также архитектурный подход;

2. Качество выполнения. Включает в себя полноту выполнения ТЗ и внимательность к техническим требованиям.

Обязательное к применению в проекте

- Использовать библиотеку DOTween для визуальных эффектов. Преимущественно для скейла (bounce), fade, movement;

- Версия unity должна быть 2019.4.31f1;

- Платформа проекта не важна, но мы игры делаем на платформе IOS и Android (Билд тестового задания не обязателен!);

- Проект ориентирован на Landscape Orientation.

- Пользоваться UnityEvent в ходе разработки/настройки;

- Bounce эффект

Геймплей

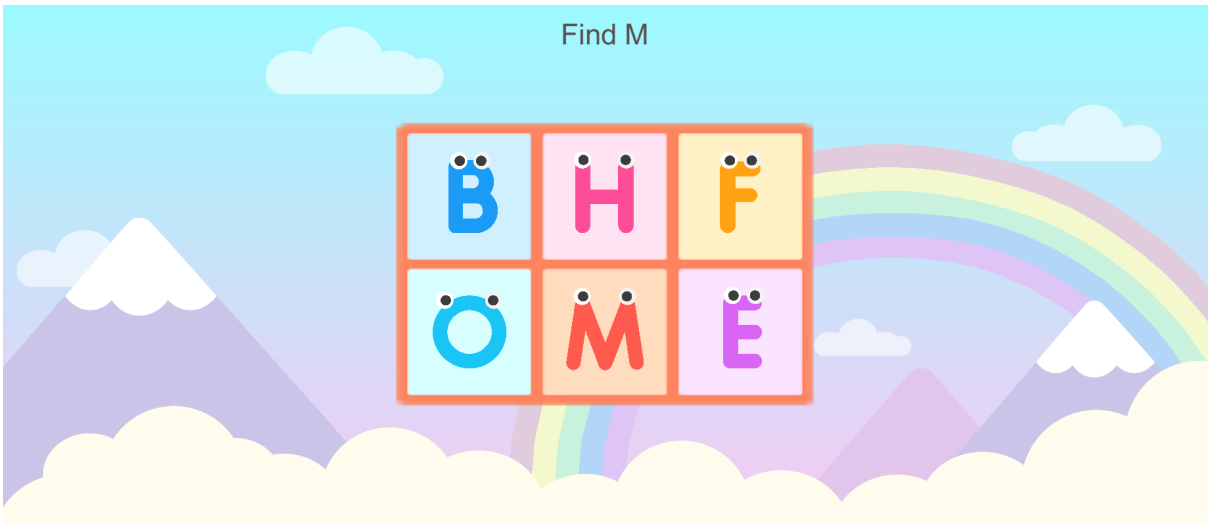
Цель игры: выбрать правильный вариант ответа, соответствующий заданию.

Игровой процесс по шагам

- В игре присутствует 3 уровня сложности. Легкий - 3 ячейки. Средний - 6 ячеек. Сложный - 9 ячеек.
- При завершении Легкого уровня игрок переходит на средний, а затем на сложный.

1-й Уровень - Легкий.

2-й Уровень - Средний.



3-й Уровень - Сложный

- При запуске сцены:
  - Через bounce эффект появляются ячейки с объектами (с цифрами/буквами);
  - Через fade in эффект появляется UI текст с заданием, выбрать правильный вариант ответа;

- При тапе на неправильный ответ:
  - Объект внутри карточки дергается туда-сюда (easeInBounce)

- При тапе на правильный ответ:
  - bounce объекта внутри карточки
  - Появляются партиклы звездочки

- При окончании всех уровней:
  - В центре экрана появляется кнопка **Restart,** нажав на нее можно начать игру заново с легкого уровня.
  - Все элементы в игре не должны быть кликабельным.
  - Должно эффектом **FadeIn/FadeOut** появиться затемнение экрана, но она не должна перекрывать кнопку **Restart.**

- При нажатии кнопки **Restart**:
  - Должен через эффект **FadeIn/FadeOut** появиться загрузочный экран (можно просто белой/черной текстурой), игровые объекты удалятся и появятся новые, после этого загрузочный экран пропадает.
  - Все должно начинаться с пункта “При запуске сцены”.

Условия генерации данных

- При смене уровня
    - Цель задания (правильный ответ) выбирается случайным образом. Она не повторяется в рамках сессии, где сессия - запущенный Play Mode. Т.е. если в задании говорится найти цифру 4, то на следующих уровнях цифра 4 не выберется
    - эффект появления ячеек НЕ происходит. Старые объекты в ячейках удаляются, появляются новые (мгновенно. Новые ячейки также появляются мгновенно).
    - мгновенно меняется текст задания в UI. Пример: Find F или Find 7
    - **Важное требование.** При смене уровня может подставится один из двух видов визуализаций (из разных наборов данных): вариант с цифрами, либо с буквами, соответственно и цель задания меняется вместе с набором объектов. Данная конфигурация должна быть максимально простой и понятной. Это необходимо для того, чтобы в дальнейшем без участия разработчика можно было в данную игру подставить фрукты, овощи, виды машин и пр.

- Во время генерации уровня учитывать, что правильный вариант ответа только один.

- В коде не должно быть разделения на букву/цифру. Механика должна работать с любыми объектами, вне зависимости от того что это. Код одинаковый для обоих случаев, различаются только наборы данных и визуальная составляющая. В общем виде будет выглядеть так: есть набор цифр от 1 до 9 включительно и есть набор букв от A до Z. Первым делом выбирается состав данных, который как минимум включает в себя необходимый набор объектов в ячейках и цель задания.

- Префаб ячейки должен быть единственным в проекте

- Gameplay не должен разделяться на разные сцены, все должно работать на одной сцене.

Технические требования

- Разработка должна вестись в рамках подхода SOLID. Отдельное внимание уделить OCP и SRP;

- В скриптах не должно быть классов, отвечающих за всю игру целиком (GameController.cs, GameManager.cs, Menu.cs, Controller.cs, LevelController.cs и прочие). **Разделяйте игровую логику по разным классам**. Пример: есть спавнер, есть другой скрипт который вызывает логику спавна в нужный момент. Либо есть отдельно логика визуализации скейла, фейда через DOTween. **Не нужно все смешивать** в один скрипт;

- Не использовать ключевое слово static помимо методов расширений. Т.е. если вы используете static чтобы получить доступ к некоторому полю - это будет грубым нарушением;

- Не применять паттерн синглтона, т.к. в большинстве случаев он приводит к антипаттерну синглетонизма. Вдобавок в данной игре можно обойтись без подобного рода доступа к классам

- Если видите, что у вас есть публичные поля, то скорее всего вы нарушаете инкапсуляцию. Пересмотрите архитектуру, уделите этому внимание

- В коде не должно быть хардкода, помимо данных визуализации твинов. Конкретнее: можно оставлять магические числа в визуализации по типу позиций, скейлов, времени выполнения и пр. В остальных случаях хардкод, магические числа не приемлемы.

- Не должно быть привязки в коде на ячейки 3, 6, 9 и каждый раз проверять какой сейчас уровень. Все должно быть построено на данных. К примеру мы хотим последовательность уровней (3 ячейки, 6, 9), то в таком случае должно быть следующее: 1-й набор данных включает 1 строку, 3 столбца; 2-й набор для среднего уровня 2 строки и 3 столбца. И после того как мы создали все нужные данные в виде Scriptable Object занесли их в некоторый массив. Т.е. понятие перехода от легкого к среднему и далее к сложному базируются на понятиях смены итераций. Последовательно из списка выбрали нужные данные для сетки на экране, а не в коде пишем что на каждый новый уровень +3 ячейки, чтобы в итоге получить 3, 6 и 9. Эти данные относятся к сетке и ячейкам сетки, что появится в этих ячейках уже другой вопрос.

- Сетка с ячейками должна быть гибкой. Можно сделать кнопками UI, но приоритетнее те работы, которые разрабатывают именно сетку для ячеек, GameObject’ов

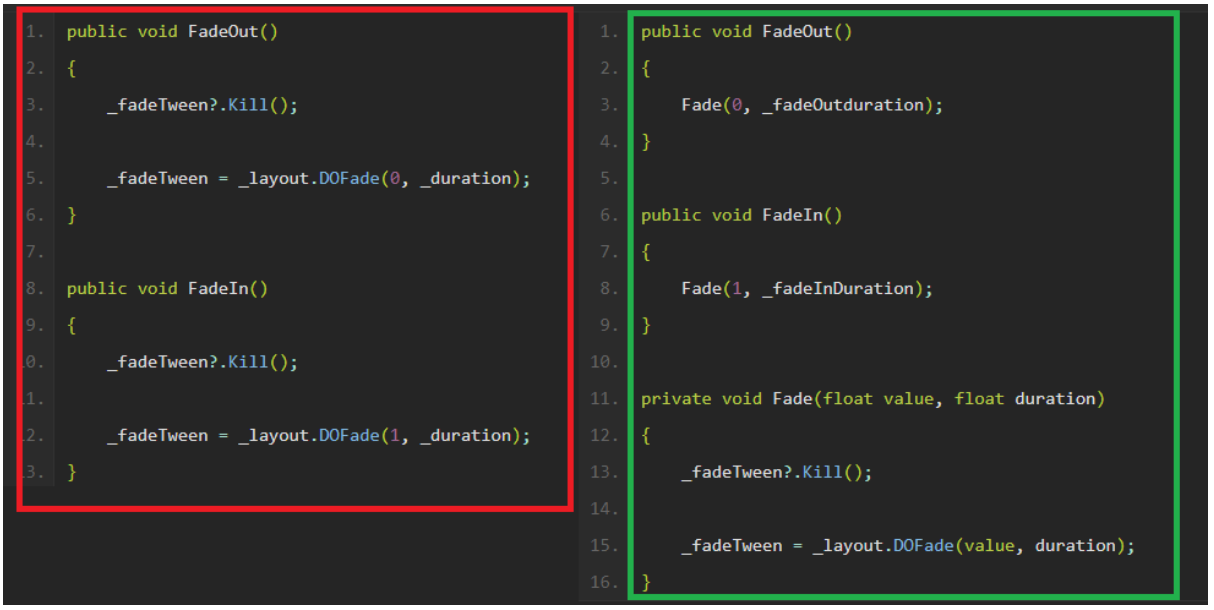
- Набор данных должен быть в отдельных файлах Scriptable Object. При замене этих SO разными данными логика должна отрабатывать. Т.е. есть один набор на цифры, второй на буквы. Если подставить третий набор с фруктами, то все должно отрабатывать корректно. Пример набора данных:

```
1.      [Serializable]
2.      public class CardData
3.      {
4.          [SerializeField]
5.          private string _identifier;
6.
7.          [SerializeField]
8.          private Sprite _sprite;
9.
10.         public string Identifier => _identifier;
11.
12.         public Sprite Sprite => _sprite;
13.     }
```

```
1.  [CreateAssetMenu(fileName = "New CardBundleData", menuName = "Card Bundle Data", order = 10)]
2.  public class CardBundleData : ScriptableObject
3.  {
4.      [SerializeField]
5.      private CardData[] _cardData;
6.
7.      public CardData[] CardData => _cardData;
8.  }
```

- Проверьте стилистику кода перед сдачей проекта. Не должно быть пустых методов, лишних переносов строк, некорректных наименований в коде. Пример: не называть переменые *int c, string abs, var btn* и пр. Проверьте, что везде все корректно. Если где-то указан явно модификатор доступа private, а где-то нет, то разработчик скорее всего не проверил код перед сдачей проекта

- Избегайте копипаста. Пример:



```
public void FadeOut()
{
    _fadeTween?.Kill();

    _fadeTween = _layout.DOFade(0, _duration);
}

public void FadeIn()
{
    _fadeTween?.Kill();

    _fadeTween = _layout.DOFade(1, _duration);
}
```

```
public void FadeOut()
{
    Fade(0, _fadeOutduration);
}

public void FadeIn()
{
    Fade(1, _fadeInDuration);
}

private void Fade(float value, float duration)
{
    _fadeTween?.Kill();

    _fadeTween = _layout.DOFade(value, duration);
}
```

- Оптимизация в приложении не требуется, но это не означает что нужно каждый раз при удобном случае вызывать GetComponent<T>(). Разделяйте инициализацию и кешируйте все необходимое