

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

- Из сторонних решений можно использовать ТОЛЬКО библиотеки для визуализации (пр. DOTween). Т.е. нельзя использовать DI контейнеры, Bootstrapper и прочие решения;
- Использовать библиотеку DOTween для визуальных эффектов. Преимущественно для скейла (bounce), fade, movement;
- Версия unity 2019/2020 года;
- Платформа проекта не важна;
- Проект должен быть ориентирован на Landscape Orientation;
- [Bounce эффект](#) (объект становится больше, затем меньше и в конце стандартный размер)

Геймплей

Цель игры: выбрать правильный вариант ответа, соответствующий заданию.

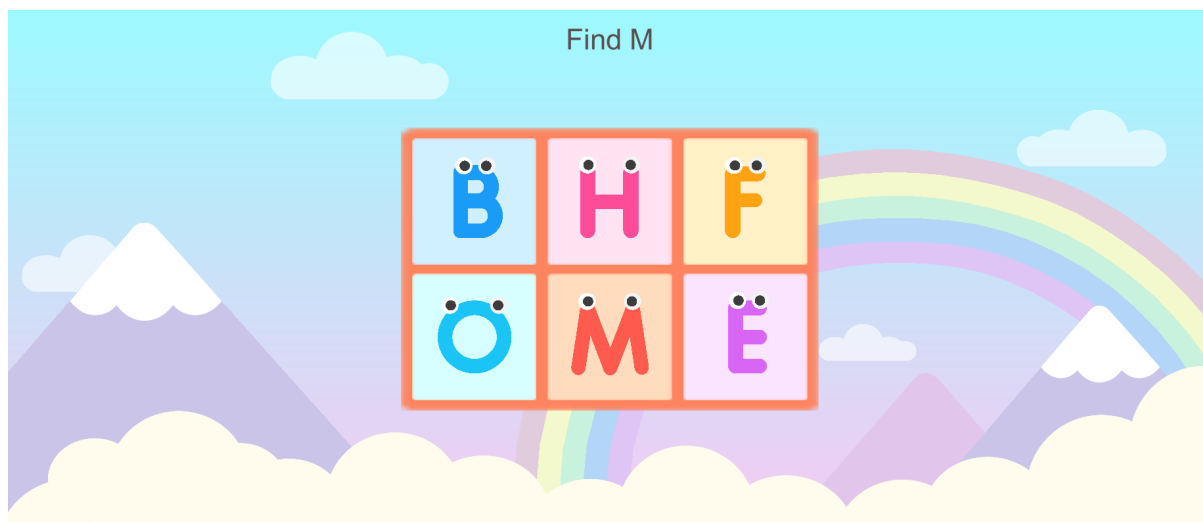
Игровой процесс по шагам

- 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);
 - Эффектом fade in появляется UI текст с заданием, выбрать правильный вариант ответа.
- При тапе на неправильный ответ:
 - Объект внутри карточки дергается туда-сюда ([easeInBounce](#)) влево-вправо.
- При тапе на правильный ответ:
 - bounce объекта внутри карточки. Эффект bounce описан выше;
 - Появляются частицы звездочки на правильном ответе (не просто где-то на экране).
- 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**. Учитываем, что затемнение не полное, а такое, чтобы было видно сетку с ячейками под ним.
- 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 подставиться 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

- Git репозиторий.
 - В репозитории **НЕ ДОЛЖНО** быть папки Library и билда с игрой.
Используйте .gitignore (генерируется автоматически в github при создании репозитория [можно выбрать unity из списка])
 - Обязательно проверьте не добавили ли вы .meta файлы в .gitignore. Без них проект нельзя будет проверить
- 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 синглтонизма. 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 вводить** в коде понятие сложности (пример: enum Difficulty). Также **не привязывайтесь** к тому, что в каждом уровне 3 столбца (т.е. в коде += 3). Количество строк и столбцов должно задаваться из данных ScriptableObject.
 - 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. Приоритетность не означает, что работы с использованием UI не рассматриваются, однако грамотная реализация на сетке для GameObject'ов ценится выше.
- Набор данных должен быть в отдельных файлах Scriptable Object. При замене этих SO разными данными логика должна обрабатывать. Т.е. есть один набор на цифры, второй на буквы. Если подставить третий набор с фруктами, то все должно обрабатывать 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.
 - Требования к стилистике:

- не должно быть лишних переносов строк (обычно двойные переносы, пустые строки в конце классов)
- не должно быть пустых методов (а если метод пуст и идет от интерфейса - то это уже нарушение ISP, а не стилистики)
- не должно быть некорректных наименований в коде. Пример: не называть переменные *int c*, *string abs*, *var btn* и пр. Наименование класса, поля, метода и т.п. должно отражать суть. К примеру GameManager.cs или ButtonScript.cs ни о чем не говорят
- **Обязательно** используйте namespace'ы
- Не пишите код слитно, разделяйте переносами строк
- Указывайте модификаторы доступа в явном виде

- Избегайте копипаста. Пример:

```

1. public void FadeOut()
2. {
3.     _fadeTween?.Kill();
4.
5.     _fadeTween = _layout.DOFade(0, _duration);
6. }
7.
8. public void FadeIn()
9. {
10.    _fadeTween?.Kill();
11.
12.    _fadeTween = _layout.DOFade(1, _duration);
13. }

```

```

1. public void FadeOut()
2. {
3.     Fade(0, _fadeOutduration);
4. }
5.
6. public void FadeIn()
7. {
8.     Fade(1, _fadeInDuration);
9. }
10.
11. private void Fade(float value, float duration)
12. {
13.     _fadeTween?.Kill();
14.
15.     _fadeTween = _layout.DOFade(value, duration);
16. }

```

- Уделить внимание оптимизации. **Не использовать ресурсоемких операций** (примеры: .Find, .FindGameObjectWithTag, .FindGameObjectByType, Resources.Load). Разделяйте инициализацию и кешируйте все GetComponent в тех скриптах, где эти объекты используются повторно.

F.A.Q.

1. Перевернутые спрайты - что делать? Нужно самостоятельно перевернуть в фотошопе? **Ответ:** изменять атлас не нужно. Но развернуть элементы в нормальное состояние конечно же нужно, а как именно решать вам (решение программное).
2. Не могу реализовать сетку для GameObject'ов. **Ответ:** Ничего страшного, используйте сетку UI
3. Я сделал работу в указанный срок, но вижу что остались моменты не по ТЗ. Что делать? **Ответ:** за то, что отдельные части работы выполнены не по заданию снижаются баллы. Напишите HR'у и попросите больше времени

4. На всякий случай скачайте свой репозиторий перед сдачей работы. Если видите ошибки в консоли, поправьте их. **Проекты, которые не запускаются не рассматриваются**