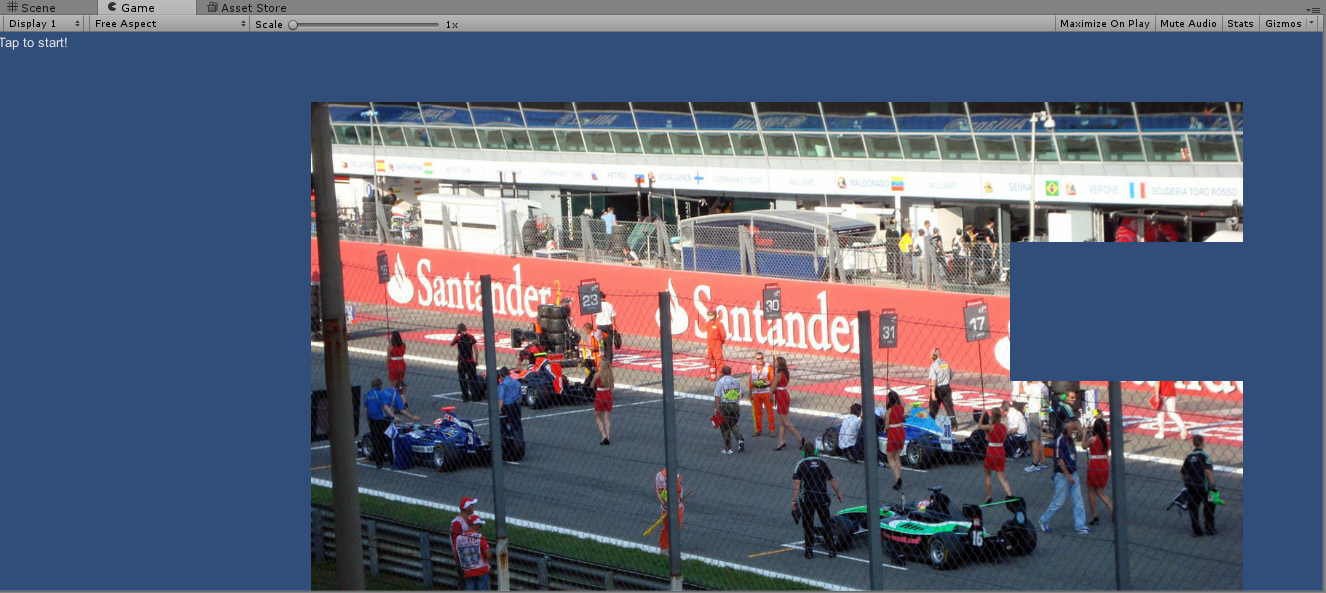
**Внесение вклада в открытый репозиторий на github**

Я выбрал следующий репозиторий - <https://github.com/dgkanatsios/PuzzleGameUnity>

Автор создал небольшую игру на движке Unity про решение пазла. Изображение разделяется на 16 равных прямоугольников, после чего из них удаляется один, прямоугольники перемешиваются, задача игрока – собрать пазл в соответствии с изначальным изображением (фактически «Пятнашки»). Я изучил код, структуру проекта, и решил внести некоторые изменения.





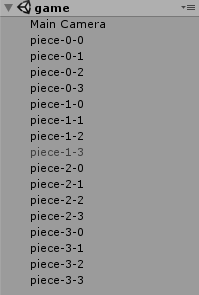
В первую очередь, добавим своё изображение в проект. Для этого найдем в папке с ассетами папку текстур и добавим туда наше изображение.



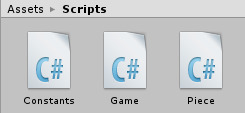
Теперь, с помощью sprite editor разделим наше изображение на 16 блоков (именно столько их должно быть по умолчанию).



Создадим новую сцену, создав копию уже существующей. Важно обратить внимание на структуру сцены. Параметры камеры являются стандартными (впоследствии мы увидим, что они меняются в коде), а также, помимо камеры на сцене расположены 16 объектов с именем piece-x-y, объекты также не имеют никаких особенных параметров, за исключением sprite renderer, изменим изначальные спрайты, заданные автором, на те которые мы только что получили, разделив нашу текстуру.

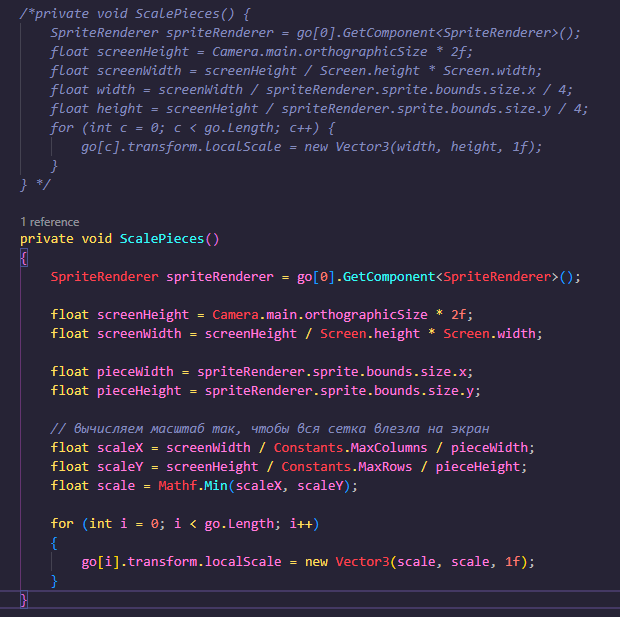


Игра управляется полностью тремя скриптами. Кратко разберем, что находится в каждом из них. Constants – содержит константы, такие как количество кусков пазла по горизонтали, вертикали, их общее количество. Эти значения будут неоднократно применяться в математике игры. Piece – скрипт класса кусочка пазла, в нем ключевыми являются четыре значения, текущая координата по обеим осям, и изначальная координата (та в которую элемент должен вернуться после перетасовки). Game – основной скрипт, содержащий всю логику игры.

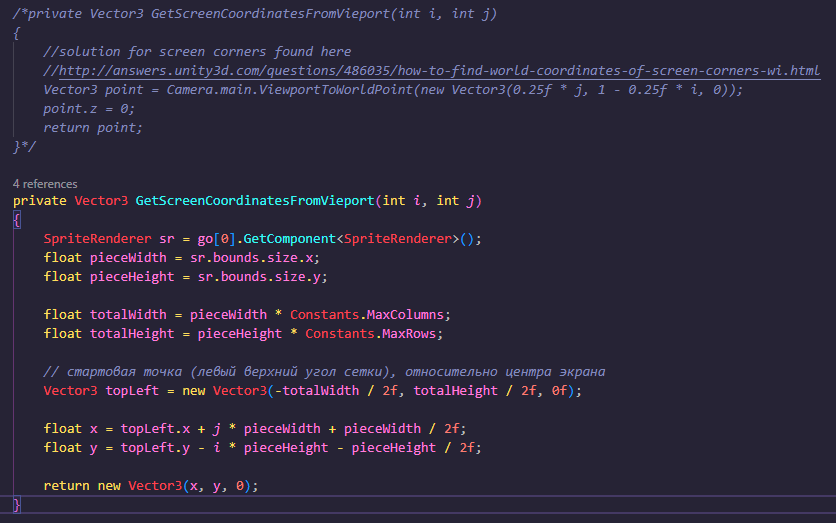


Казалось бы, что все необходимое сделано и игра должна работать корректно, но этого не произойдет, так как автор написал логику, адаптированную только лишь для его изображения. Займемся внесением изменений.

Первая проблема, с которой я столкнулся – камера неправильно захватывает изображение (из-за этого часть элементов не влезала) и масштабирование элементов также было написано с недостатками (из-за чего элементы моего пазла в оригинале которого идеальный квадрат растягивали на совершенно ненужные прямоугольники). Я изменил две функции. В функции ScalePieces помимо общего улучшения читаемости кода я меняю принцип масштабирования элементов. Теперь увеличение в ширину и в длину одинаково, соответственно, будут получаться ровные квадраты вместо прямоугольников.



Следующее изменение я внес в функцию GetScreenCoordinatesFromViewport, предыдущий ее вариант был слишком простым и рассчитывал позиции элементов в отношение экрана, не учитывая важных параметров, таких как размер элемента.

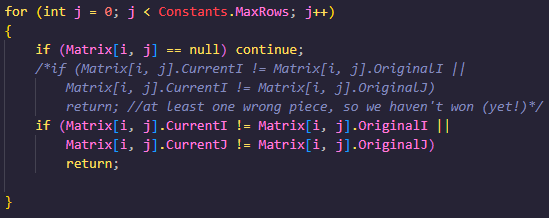


Теперь при запуске игры в Unity все элементы пазла отображаются на экране корректно.





Следующую проблему я обнаружил, когда собрал пазл полностью (оказалось пятнашки не так уж и легко собираются). Игра не заканчивалась, не появлялось соответствующего сообщения с предложением нажать на экран чтобы начать заново. Я быстро нашел ошибочную строчку кода в скрипте Game, автор по случайности два раза написал проверку для одной и той же координаты. Исправляем!

****

Следующее добавление связано с математической логикой, стоящей за этой, казалось бы, безумно простой игрой. Дело в том, что половина комбинаций в игре неразрешимы. В то же время, наша цель генерировать расположение элементов так, чтобы головоломку можно было решить. Для понимания приведу таблицу:

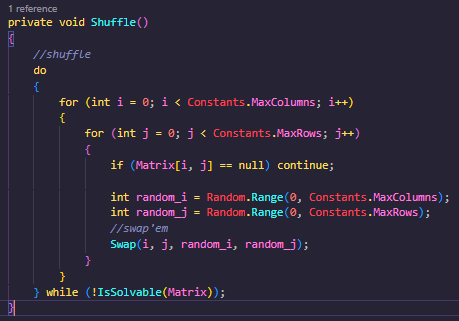
\*Четность инверсий – количество всех пар фишек, где a идет перед b, при этом a > b. Четное или нечетное количество.

|  |  |  |
| --- | --- | --- |
| **Четность инверсий** | **Пустая клетка** | **Решаемость** |
| Четное количество | В четной строке | Решаемо |
| Нечетное количество | В нечетной строке | Решаемо |
| Четное количество | В нечетной строке | Не решаемо |
| Нечетное количество | В четной строке | Не решаемо |

С помощью функции будем проверять каждую генерацию игры, и если условие решаемости не выполняется, то будем генерировать снова. Этот метод имеет нормальную эффективность, так как ровно 50% вариантов будут решаемыми или нерешаемыми.



Теперь доработаем созданную автором функцию shuffle.



Вот пример работы игры, где я решаю головоломку:



Как мы видим, всё отображается корректно, головоломка сгенерировалась решаемая, игра закончилась корректно (маленьким шрифтом в верхнем левом углу), ну и самое главное – в игре используется собственное изображение!