**Аннотация**

*Мы начинаем рассматривать игры на клетчатом поле. Это занятие посвящено идеологии клетчатых игр и реализации простейших алгоритмов. Продолжим на следующем занятии.*

**Что такое клетчатое поле?**

В основе очень многих игр лежит клетчатое поле. Оно встречается не только в шахматах, шашках и «крестиках-ноликах». Это и стакан тетриса, и поле змейки. Даже обычная стратегия редко обходится без клеток.



*Age of Empires тоже разделена на клетки-тайлы.*

Поэтому очень важно научиться работать «с клеточками».

**Создание класса**

В любом большом проекте (а игра — это достаточно большая программа) проще мыслить объектно. Поэтому мы будем строить **класс** клетчатого поля. Чаще всего используется прямоугольное поле с квадратными клетками.

Давайте подумаем, какие поля и какие методы будут в проектируемом классе? Что общего между всеми полями всех игр?

На этот вопрос можно отвечать немного по-разному, но неоспоримым фактом остается то, что у поля есть **размер**, который можно измерить в клетках. Кроме того, важные параметры игрового поля — это размер клетки, а также положение его верхнего левого угла на экране.

Поле должно уметь как минимум **создаваться** и **рисоваться** (отображать свое текущее состояние).

Для того чтобы программа была гибкой, реализуем принципы рисования отдельно.

Само поле можно представить двумерным списком. Для большинства задач достаточно хранить в этом списке обычные целые числа.

Класс можно определить примерно так:

class Board:

# создание поля

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

self.board = [[0] \* width for \_ in range(height)]

# значения по умолчанию

self.left = 10

self.top = 10

self.cell\_size = 30

# настройка внешнего вида

def set\_view(self, left, top, cell\_size):

self.left = left

self.top = top

self.cell\_size = cell\_size

*На этом занятии мы не будем рассматривать практически ничего нового из возможностей библиотеки Pygame, и в процессе занятия вам сразу предстоит решать задачи.*

Для упрощения дальнейшей работы стоит отрисовывать поле не линиями, а квадратами. В этом случае мы пройдем по всем клеткам, а ведь именно с клетками и нужно будет работать. Например, такой способ необходим, когда надо нарисовать не «чистый» квадрат, а какую-то осмысленную картинку, как в случае с Age of Empires.

Добавьте в класс Board метод render(screen), принимающий в себя холст так, чтобы следующий фрагмент:

# поле 5 на 7

board = Board(5, 7)

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

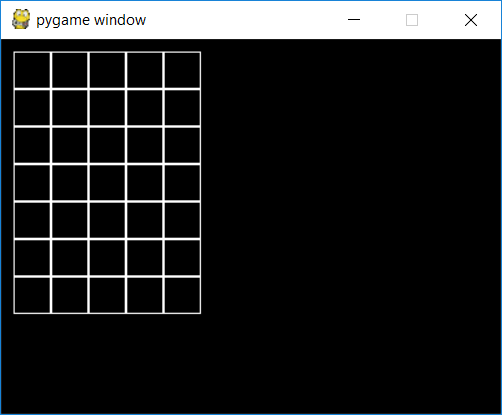
running = False

screen.fill((0, 0, 0))

board.render(screen)

pygame.display.flip()

нарисовал вот такую картинку:

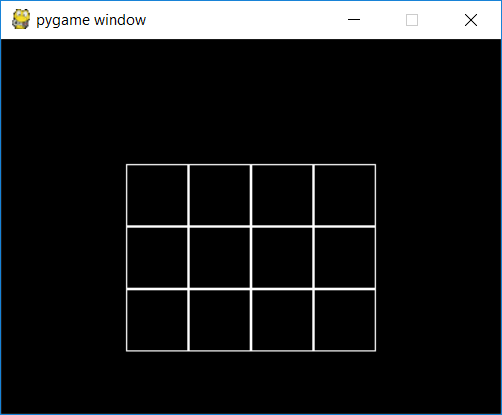


А если заменить строку инициализации board и добавить вызов метода set\_view() следующим образом:

board = Board(4, 3)

board.set\_view(100, 100, 50)

то такую:



Обратите внимание, что внутри поля получается **удвоение** линий. Это естественно, потому что каждая клетка занимает свой собственный размер, и граница входит в размер клетки.

**Реакция поля на события мыши**

При щелчке по клетке поле должно «понять», где именно произошло нажатие, и среагировать на это событие.

Это удобно оформить при помощи трех дополнительных методов:

* Метода get\_cell(self, mouse\_pos), который **возвращает координаты клетки в виде кортежа** по переданным координатам мыши. Он должен вернуть None, если координаты мыши оказались вне поля
* Метода on\_click(self, cell\_coords), который как-то изменяет поле, опираясь на полученные координаты клетки
* Метода get\_click(self, mouse\_pos) — «диспетчера», который получает событие нажатия и вызывает первые два метода

Реализуйте все три описанных метода.

Например, метод get\_click() может быть реализован так:

def get\_click(self, mouse\_pos):

cell = self.get\_cell(mouse\_pos)

self.on\_click(cell)

Следуя описанной выше технологии, можно достаточно легко реализовывать логику различных игр на клетчатом поле. Достаточно получать координаты мыши в главном цикле программы и просто передавать ее полю:

...

if event.type == pygame.MOUSEBUTTONDOWN:

board.get\_click(event.pos)

...

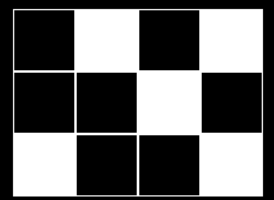
А дальше поле сделает все само!

**Пример**

Теперь надо «оживить» поле. До этого момента мы никак не задействовали список board. Что мы можем с ним делать?

Допустим, что в нашем списке хранятся нули и единицы. Ноль означает, что клетка **черная**, а единица — **белая**. Тогда мы можем, например, менять цвет клетки по нажатию клавиш мышки на противоположный, с черного на белый и обратно.

Получится монохромная интерактивная мозаика:



Кажется, что сделано очень мало. Но это не так. Построен фундамент, и на базе нашего простого класса получится достаточно быстро реализовать самые разные клеточные игры.