**Класс GameObject**  
GameObject представляет собой визуальный объект, знающий о том, как себя рендерить, сохранять свои границы и перемещаться. GameObject предназначен для того, чтобы быть базовым классом для других объектов. Он непосредственно раскрывает множество свойств его прямоугольника self.bounds, а в своём методе update() он перемещает объект в соответствии с его текущей скоростью. Он ничего не делает в своём методе draw(), который должен быть переопределён подклассами.

**Класс Game**

Класс Game — это ядро игры. Он выполняется в основном цикле. Метод \_\_init\_\_() инициализирует сам Pygame, систему шрифтов и звуковой микшер. Три разных вызова нужны, так как не во всякой игре на Pygame используются все компоненты, поэтому можно контролировать подсистемы, которые мы используем, и инициализировать только нужные с соответствующими параметрами. Метод создаёт фоновое изображение, основную поверхность (на которой всё отрисовывается) и игровой таймер.

Элемент self.objects хранит все игровые объекты, которые должны рендериться и обновляться. Методы update() и draw() обходят все управляемые игровые объекты и вызывают соответствующие им методы. Если два объекта накладываются друг на друга на экране, то порядок списка объектов определяет, какой из них будет рендериться первым, а остальные будут частично или полностью его перекрывать.

Метод handle\_events() слушает события, генерируемые Pygame, такие как события клавиш и мыши.

Метод run() выполняет основной цикл. Он выполняется до тех пор, пока элемент game\_over не принимает значение True. В каждой итерации он рендерит фоновое изображение и вызывает по порядку методы handle\_events(), update() и draw(). Он вызывает метод clock.tick() для управления тем, когда будет вызвана следующая итерация.

**Класс TextObject**  
Класс TextObject предназначен для отображения текста на экране. Класс TextObject создаёт объект шрифта. Он рендерит текст на отдельную текстовую поверхность, которая затем копируется на основную поверхность.

**Полноэкранный режим**

Сначала вызывается init() Pygame, а затем создаются основная поверхность рисования и таймер. Затем выполняется основной цикл, который постоянно заполняет экран однотонным серым цветом и вызывает метод таймера tick() с частотой кадров.

**Фоновое изображение**  
Pygame очень хорошо работает с изображениями. Сначала он перед основным циклом загружает фоновое изображение с помощью функции pygame.image.load(). Затем вместо того, чтобы заливать экран цветом, он выполняет блиттинг (копирование битов) изображения на экран в позицию (0,0). В результате на экране отображается изображение.

**Кирпичи(клетки)**

Кирпичи — это просто прямоугольники. В Pygame есть функция pygame.draw.rect(), получающая поверхность, цвет и объект Rect и копирующая прямоугольник. класс Brick является подклассом GameObject и получает все его свойства, но также имеет и цвет, который обрабатывает самостоятельно (потому что могут существовать игровые объекты, имеющие несколько цветов).

**Шарик(вирус)**

В Pygame есть функция pygame.draw.circle(), получающая цвет, центр, радиус и дополнительный параметр ширины, который по умолчанию равен нулю. Ball тоже является подклассом GameObject. Так как шарик всегда движется (в отличие от кирпичей), он также имеет скорость, которая передаётся для обработки базовому классу GameObject.

**Ракетка**(двигующийся прямоугольник, двигающийся влево и вправо в ответ на нажатия игроком клавиш со стрелками.)

**Класс Breakout**  
В классе Breakout реализуется большинство знаний о том, как управляется игра.

*# Регистрация метода handle\_mouse\_event() объекта кнопки*

self.mouse\_handlers.append(b.handle\_mouse\_event)

*# Регистрация метода handle() ракетки для обработки событий клавиш*

self.keydown\_handlers[pygame.K\_LEFT].append(paddle.handle)

self.keydown\_handlers[pygame.K\_RIGHT].append(paddle.handle)

self.keyup\_handlers[pygame.K\_LEFT].append(paddle.handle)

self.keyup\_handlers[pygame.K\_RIGHT].append(paddle.handle)

 При нажатии или отпускании клавиши вызывается его метод handle(). Объекту Paddle не нужно знать, было ли это событие нажатия или отпускания клавиши, потому что он управляет текущим состоянием с помощью пары булевых переменных: moving\_left и moving\_right. Если moving\_left равна True, то значит, была нажата клавиша «влево», и следующим событием будет отжатие клавиши, которое сбросит переменную. То же самое относится и к клавише «вправо».

Необходимость приостановки игры: например, при отображении сообщения, которое игрок должен прочитать и чтобы при этом ничего его не отвлекало. Метод show\_message() класса Breakout использует такой подход и вызывает time.sleep().

Методе update(), здесь выполняются определённые вычисления, если ракетка находится близко к левой или правой границе экрана. Мы не хотим, чтобы ракетка выходила за границы экрана. Мяч создаётся, когда игрок теряет жизнь, в методе create\_ball(), он получает скорость со случайным горизонтальным компонентом в промежутке от -2 до 2 и вертикальным компонентом, задаваемым в модуле config.py.

В методе handle\_ball\_collisions() есть встроенная функция под названием intersect(), которая используется для проверки того, ударился ли шарик об объект, и того, где он столкнулся с объектом. Она возвращает 'left', 'right', 'top', 'bottom' или None, если шарик не столкнулся с объектом.

Когда шарик стукается об ракетку, он отскакивает. Если он ударяется о верхнюю часть ракетки, то отражается обратно вверх, но сохраняет тот же компонент горизонтальной скорости.  
  
Но если он ударяется о боковую часть ракетки, то отскакивает в противоположную сторону (влево или вправо) и продолжает движение вниз, пока не столкнётся с полом. В коде используется функция intersect().

Когда шарик ударяется об стены или потолок, он просто отскакивает от них. Когда ракетка пропускает шарик на пути вниз (или шарик ударяется об ракетку сбоку), то шарик продолжает падать и затем ударяется об пол. В этот момент игрок теряет жизнь и шарик создаётся заново, чтобы игра могла продолжаться. Игра завершается, когда у игрока заканчиваются жизни. Когда шарик ударяется об кирпич, это является основным событием игры Breakout — кирпич исчезает, игрок получает очко, шарик отражается назад

В Breakout есть простое меню с двумя кнопками, 'PLAY' и 'QUIT'. Меню отображается в начале игры и пропадает, когда игрок нажимает на 'PLAY'.Функция create\_menu() создаёт меню с двумя кнопками с текстом 'PLAY' и 'QUIT'. Она имеет две встроенные функции, on\_play() и on\_quit(), которые она передаёт соответствующей кнопке. Каждая кнопка добавляется в список objects (для отрисовки), а также в поле menu\_buttons.

Рано или поздно игра должна закончиться. В этой версии Breakout игра заканчивается одним из двух способов: игрок или теряет все свои жизни, или разрушает все кирпичи.

Полю game\_over класса Game присваивается значение False в методе \_\_init\_\_() класса Game. Основной цикл продолжается до тех пор, пока переменная game\_over не изменит значение на True. Breakout показывает традиционное сообщение 'GAME OVER!', а когда игрок выигрывает, она показывает сообщение 'YOU WIN!'

В классе Breakout все звуковые эффекты загружаются из config в объект pygame.mixer.Sound и хранятся в словаре.