**A ideia**

No jogo, há filas de tijolos no topo da tela. O objetivo do jogo é destruir todos os tijolos com uma pequena bola que atravessa a tela. Um tijolo é destruído quando atingido pela bola. A bola é rebatida quando atinge as bordas da tela, os tijolos e o remo. O jogador deve evitar que a bola toque a parte inferior da tela (ou em palavras comuns, caindo no chão). Para evitar que isso aconteça, o jogador usa o remo para devolver a bola para cima, mantendo-a em jogo.

Sua pontuação aumenta quando você destrói tijolos. Por outro lado, você perde uma vida toda vez que a bola cai no chão.

**Estados do jogo**

O jogo tem 4 estados:

1. *Bola no remo* - chegamos nesse estado quando o jogo começa ou quando perdemos a bola.
2. *Jogando* - chegamos nesse estado quando pressionamos o SPACE para lançar a bola.
3. *Fim de jogo* - quando perdemos todas as nossas vidas, chegamos nesse estado.
4. *Ganhou* - nós entramos neste estado quando destruímos todos os tijolos.

Esta é a sequência de eventos. O jogo começa no estado "bola no remo". Nesse estado, a bola é colada ao remo. Podemos mover o remo e, quando o fazemos, a bola segue o remo, porque está colada ao remo. Então, quando pressionamos o SPACE, o jogo entra no estado “Playing”. Neste estado a bola viaja através da tela saltando fora das bordas da tela, tijolos e o remo. Cada vez que a bola atinge um tijolo, aumentamos a pontuação. Por outro lado, se perdermos a bola, diminuímos a contagem de vida e, se for maior que zero, entramos na “bola em estado de remo”. Caso contrário, entramos no estado "game over".

Enquanto no estado "jogando", se destruirmos todos os tijolos, entraremos no estado "vencido". Passamos dos estados “game over” e “won” para o estado “ball in paddle” pressionando a tecla ENTER. Na verdade, isso significa jogar novamente.

**Movimento da pá**

Nós movemos o remo usando as teclas de seta ESQUERDA e DIREITA. Ao mover o remo, garantimos que ele permaneça dentro da tela.

**Movimento da bola**

A bola se move em 4 direções: para cima-direita, para cima-esquerda, para baixo-direita, para baixo-esquerda. Para simular isso, usamos um vetor de velocidade. Espere, não tenha medo dessa expressão. Significa apenas que usamos valores separados para atualizar a posição da bola. Por exemplo, para simular o movimento para cima, usamos velX = 5, velY = -5 (o velY é negativo para fazer a bola subir). Abaixo mostro mais exemplos de valores vetoriais de velocidade.

|  |  |
| --- | --- |
| 1  2  3  4  5 | [X,Y]  [5,-5] --> up-right direction  [-5,-5] --> up-left direction  [-5,5] --> down-right direction  [-5,-5] --> down-left direction |

Temos certeza de que a bola está dentro da tela. Quando ele atinge as bordas esquerda e direita, invertemos o componente X velocity. Então, se fosse para a esquerda, acabaria indo para a direita e vice-versa. Por outro lado, se a bola atingir a raquete, as bordas superior ou inferior ou um tijolo, inverteremos o componente de velocidade Y. Então, se a bola estava subindo, ela acaba caindo e vice-versa.

**Detecção de colisão**

A detecção de colisão é uma técnica usada para saber se dois objetos colidem ou não. Basicamente, dois objetos colidem se eles se sobrepuserem. Empregamos detecção de colisão para determinar se a bola colide com os tijolos e a pá.

**O código**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import pygame    SCREEN\_SIZE   = 640,480   # Object dimensions  BRICK\_WIDTH   = 60  BRICK\_HEIGHT  = 15  PADDLE\_WIDTH  = 60  PADDLE\_HEIGHT = 12  BALL\_DIAMETER = 16  BALL\_RADIUS   = BALL\_DIAMETER / 2    MAX\_PADDLE\_X = SCREEN\_SIZE[0] - PADDLE\_WIDTH  MAX\_BALL\_X   = SCREEN\_SIZE[0] - BALL\_DIAMETER  MAX\_BALL\_Y   = SCREEN\_SIZE[1] - BALL\_DIAMETER    # Paddle Y coordinate  PADDLE\_Y = SCREEN\_SIZE[1] - PADDLE\_HEIGHT - 10    # Color constants  BLACK = (0,0,0)  WHITE = (255,255,255)  BLUE  = (0,0,255)  BRICK\_COLOR = (200,200,0)    # State constants  STATE\_BALL\_IN\_PADDLE = 0  STATE\_PLAYING = 1  STATE\_WON = 2  STATE\_GAME\_OVER = 3 |

Começamos importando o módulo pygame. Em seguida, definimos algumas constantes. A primeira constante define as dimensões da tela. Então, temos constantes que definem as dimensões da raquete, bola e tijolos. Depois disso, definimos constantes que especificam a coordenada X máxima para a bola e o remo. Essas constantes serão usadas mais tarde para impor a pá e a bola dentro da tela. Finalmente, definimos constantes de cor e estado.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **class Bricka:**    **def \_\_init\_\_(self):**  **pygame.init()**    **self.screen = pygame.display.set\_mode(SCREEN\_SIZE)**  **pygame.display.set\_caption("bricka (a breakout clone by codeNtronix.com)")**    **self.clock = pygame.time.Clock()**    **if pygame.font:**  **self.font = pygame.font.Font(None,30)**  **else:**  **self.font = None**    **self.init\_game()** |

Encapsulamos o código do jogo dentro da classe Bricka. No construtor, em primeiro lugar, inicializamos os módulos pygame. Em seguida, criamos a janela do jogo e definimos um título para ela. Em seguida, criamos um objeto de relógio que será usado posteriormente para bloquear nossa taxa de quadros para um valor constante. Em seguida, criamos um objeto de fonte, somente se o módulo de fonte estiver disponível. Este objeto será usado posteriormente para desenhar texto na tela. Finalmente, chamamos a função *init\_game ()*. Esta função é descrita abaixo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | def init\_game(self):      self.lives = 3      self.score = 0      self.state = STATE\_BALL\_IN\_PADDLE        self.paddle = pygame.Rect(300,PADDLE\_Y,PADDLE\_WIDTH,PADDLE\_HEIGHT)      self.ball = pygame.Rect(300,PADDLE\_Y -BALL\_DIAMETER,BALL\_DIAMETER,BALL\_DIAMETER)        self.ball\_vel = [5,-5]      self.create\_bricks()    def create\_bricks(self):      y\_ofs = 35      self.bricks = []      for i in range(7):          x\_ofs = 35          for j in range(8):              self.bricks.append(pygame.Rect(x\_ofs,y\_ofs,BRICK\_WIDTH,BRICK\_HEIGHT))              x\_ofs += BRICK\_WIDTH + 10          y\_ofs += BRICK\_HEIGHT + 5 |

Na função *init\_game ()*, redefinimos algumas variáveis. Começamos com 3 vidas, pontuação 0 e estado definido como STATE\_BALL\_IN\_PADDLE. Em seguida, definimos os retângulos para a raquete e a bola.Posteriormente, usaremos esses retângulos para movimentação, desenho e detecção de colisão. Então, inicializamos a velocidade da bola, configurando-a para ir para cima e para a direita. Finalmente, chamamos a função *create\_bricks ()* que irá criar os tijolos. Os tijolos são mantidos em uma lista.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | def check\_input(self):      keys = pygame.key.get\_pressed()        if keys[pygame.K\_LEFT]:          self.paddle.left -= 5          if self.paddle.left < 0:              self.paddle.left = 0        if keys[pygame.K\_RIGHT]:          self.paddle.left += 5          if self.paddle.left > MAX\_PADDLE\_X:              self.paddle.left = MAX\_PADDLE\_X        if keys[pygame.K\_SPACE] and self.state == STATE\_BALL\_IN\_PADDLE:          self.ball\_vel = [5,-5]          self.state = STATE\_PLAYING      elif keys[pygame.K\_RETURN] and (self.state == STATE\_GAME\_OVER or self.state== STATE\_WON):          self.init\_game() |

A função *check\_input ()* manipula a entrada do teclado. Em primeiro lugar, obtemos uma lista com os estados de todas as chaves. Depois disso, se a tecla de seta ESQUERDA é pressionada, movemos o remo para a esquerda. Da mesma forma, se a tecla de seta para a DIREITA é pressionada, movemos a raquete para a direita. Durante o movimento, certificamo-nos de que o remo fique dentro da tela. Então, se a tecla ESPAÇO for pressionada enquanto estivermos no estado *Ball in Paddle* , mudamos a velocidade para subir de direita e mudamos o estado para *Playing* . Isso faz com que a bola seja lançada. Finalmente, se a tecla ENTER for pressionada enquanto estiver no estado do *jogo* ou *ganha* , chamamos init\_game () para reiniciar o jogo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | def move\_ball(self):      self.ball.left += self.ball\_vel[0]      self.ball.top  += self.ball\_vel[1]        if self.ball.left <= 0:          self.ball.left = 0          self.ball\_vel[0] = -self.ball\_vel[0]      elif self.ball.left >= MAX\_BALL\_X:          self.ball.left = MAX\_BALL\_X          self.ball\_vel[0] = -self.ball\_vel[0]        if self.ball.top < 0:          self.ball.top = 0          self.ball\_vel[1] = -self.ball\_vel[1]    def handle\_collisions(self):      for brick in self.bricks:          if self.ball.colliderect(brick):              self.score += 3              self.ball\_vel[1] = -self.ball\_vel[1]              self.bricks.remove(brick)              break        if len(self.bricks) == 0:          self.state = STATE\_WON        if self.ball.colliderect(self.paddle):          self.ball.top = PADDLE\_Y - BALL\_DIAMETER          self.ball\_vel[1] = -self.ball\_vel[1]      elif self.ball.top > self.paddle.top:          self.lives -= 1          if self.lives > 0:              self.state = STATE\_BALL\_IN\_PADDLE          else:              self.state = STATE\_GAME\_OVER |

As funções *movem\_ball ()* cuidam de mover a bola. Primeiro, atualiza as coordenadas de posição, adicionando os componentes de velocidade. Depois disso, ele verifica se a bola atingiu a borda esquerda ou direita da tela. Se verdadeiro, o componente de velocidade X é invertido, fazendo-o saltar. Finalmente, ele verifica se a bola atingiu a borda superior, invertendo o componente de velocidade Y, se verdadeiro.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | def handle\_collisions(self):      for brick in self.bricks:          if self.ball.colliderect(brick):              self.score += 3              self.ball\_vel[1] = -self.ball\_vel[1]              self.bricks.remove(brick)              break        if len(self.bricks) == 0:          self.state = STATE\_WON        if self.ball.colliderect(self.paddle):          self.ball.top = PADDLE\_Y - BALL\_DIAMETER          self.ball\_vel[1] = -self.ball\_vel[1]      elif self.ball.top > self.paddle.top:          self.lives -= 1          if self.lives > 0:              self.state = STATE\_BALL\_IN\_PADDLE          else:              self.state = STATE\_GAME\_OVER |

A função *handle\_collisions ()* determina se a bola colidiu com um tijolo, a pá ou caiu no chão. Primeiro, verifica se a bola colidiu com um tijolo. Se for verdade, aumenta a pontuação em 3 unidades e remove o tijolo da lista de tijolos. Depois de fazer o teste de colisão de tijolos, ele verifica se há tijolos remanescentes. Caso contrário, altera o estado para *Won* . Finalmente, verifica se a bola bateu no remo. Se for verdade, a bola é reposicionada de modo que fique logo acima da raquete e o componente de velocidade Y seja invertido. Caso contrário, verifica se a bola está abaixo do remo (indo para o chão), e se for verdade, diminui a contagem de vida. Se a contagem cair para zero, muda o estado para *Bola no remo.*Caso contrário, altera o estado para *Bola no remo.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | def show\_stats(self):      if self.font:          font\_surface = self.font.render("SCORE: " + str(self.score) + " LIVES: "+ str(self.lives), False, WHITE)          self.screen.blit(font\_surface, (205,5))    def show\_message(self,message):      if self.font:          size = self.font.size(message)          font\_surface = self.font.render(message,False, WHITE)          x = (SCREEN\_SIZE[0] - size[0]) / 2          y = (SCREEN\_SIZE[1] - size[1]) / 2          self.screen.blit(font\_surface, (x,y)) |

Essas duas funções desenham textos na tela. O *show\_stats ()* mostra pontuação e informações de vida enquanto show\_message () é usado para mostrar mensagens relacionadas ao estado do jogo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | def run(self):      while 1:          for event in pygame.event.get():              if event.type == pygame.QUIT:                  sys.exit            self.clock.tick(50)          self.screen.fill(BLACK)          self.check\_input()            if self.state == STATE\_PLAYING:              self.move\_ball()              self.handle\_collisions()          elif self.state == STATE\_BALL\_IN\_PADDLE:              self.ball.left = self.paddle.left + self.paddle.width / 2              self.ball.top  = self.paddle.top - self.ball.height              self.show\_message("PRESS SPACE TO LAUNCH THE BALL")          elif self.state == STATE\_GAME\_OVER:              self.show\_message("GAME OVER. PRESS ENTER TO PLAY AGAIN")          elif self.state == STATE\_WON:              self.show\_message("YOU WON! PRESS ENTER TO PLAY AGAIN")            self.draw\_bricks()          # Draw paddle          pygame.draw.rect(self.screen, BLUE, self.paddle)            # Draw ball          pygame.draw.circle(self.screen, WHITE, (self.ball.left + BALL\_RADIUS, self.ball.top + BALL\_RADIUS), BALL\_RADIUS)            self.show\_stats()          pygame.display.flip() |

Este é o loop do jogo. Primeiro, lidamos com os eventos da janela. Se uma solicitação para encerrar o aplicativo existir, nós o encerramos. Depois de manipular os eventos, usamos o objeto clock para bloquear a taxa de quadros para 50 FPS. Então, lidamos com a entrada do teclado. Depois disso, a próxima ação depende do estado do jogo. Se estivermos no estado *Jogando*, movemos a bola chamando *move\_ball ()* e manipulamos as colisões chamando *handle\_collisions ()* . Caso contrário, imprimimos uma mensagem com instruções. No estado *Bola na pá* , asseguramos que a bola esteja colada à pá. Depois disso, desenhamos a raquete, a bola e exibimos a pontuação e o texto da vida. Finalmente, chamamos *pygame.display.flip ()*para exibir tudo o que foi desenhado no quadro.

|  |  |
| --- | --- |
| 1  2 | if \_\_name\_\_ == "\_\_main\_\_":      Bricka().run() |

Finalmente, este trecho de código cria uma instância da classe de jogo e a executa. Isso garante que o arquivo seja executado diretamente e não importado de um módulo.

[Clique aqui](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt-BR&u=http://goo.gl/MSfvl&xid=17259,1500004,15700023,15700186,15700190,15700248&usg=ALkJrhgkq_zjNf6MuZndbj5RtXMbV8PX0A) para baixar o código-fonte completo.