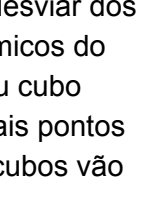






OBJETIVO DO GAME

O objetivo do jogo “cube-crush” é desviar dos cubos que caem em pontos randômicos do tabuleiro. Quanto mais tempo o seu cubo consegue ficar sem ser atingido mais pontos são acumulados e mais rápido os cubos vão sendo atirados no tabuleiro.



Requisitos obrigatórios

OpenGL

Todo o projeto é renderizado em OpenGL.

Interação do usuário

O usuário pode se mover para qualquer direção através das teclas WASD ou através do teclado número. Pelo teclado numérico é possível mover-se pela diagonal através das teclas 7913. Além disso, é possível pular em uma direção através das setas do teclado.

Pontos de Iluminação

O jogo possui luz ambiente e um ponto de luz difusa situado próximo a figura de uma lua.

Requisitos obrigatórios e desejáveis

Aplicação de textura

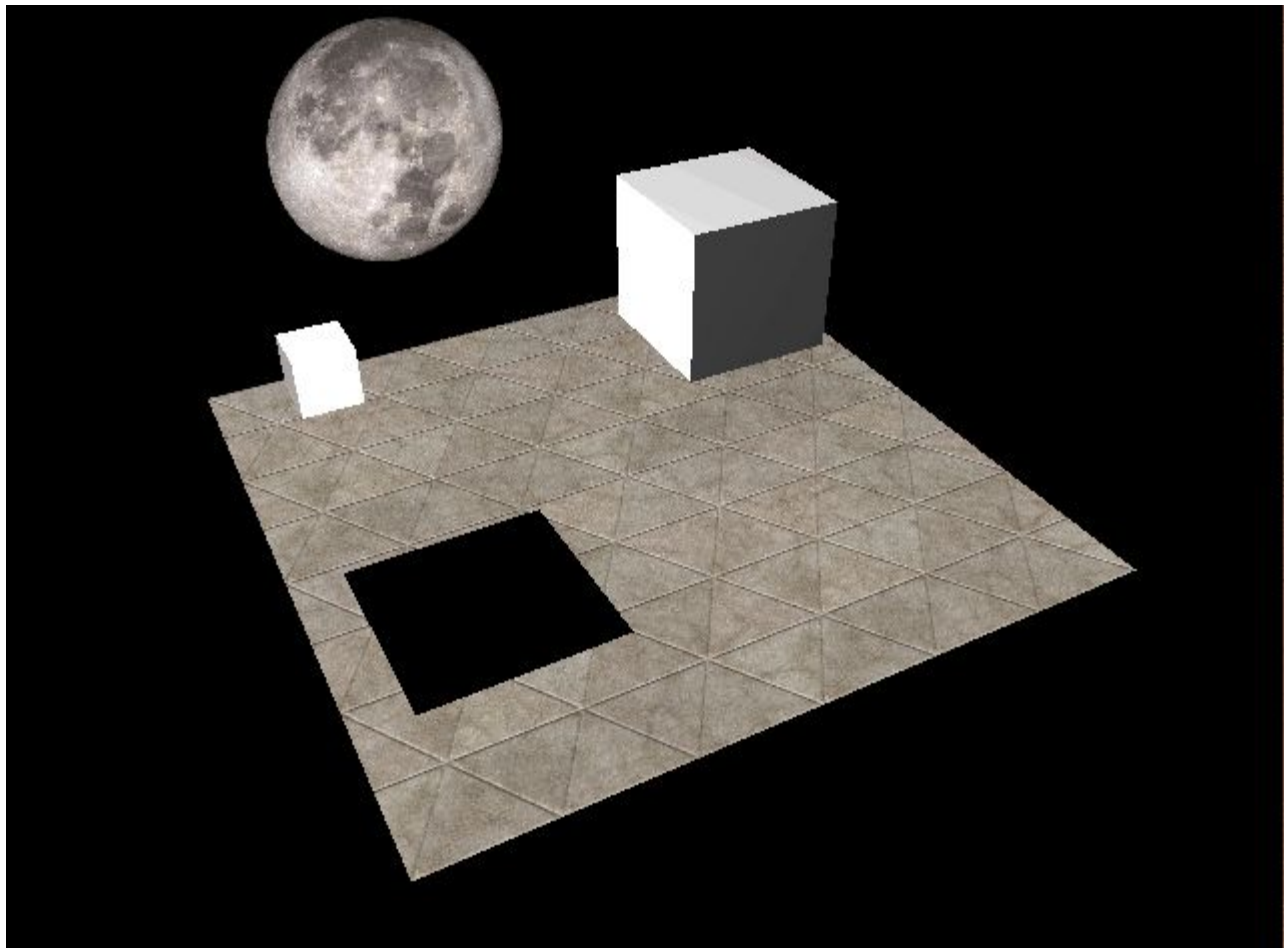
O tabuleiro, a lua e o efeito de buraco possuem textura.

Controle de colisão

A colisão mais óbvia ocorre quando os cubos que estão caindo atingem o cubo principal. É raro ele ser atingido diretamente pois quando os cubos caem eles fazem buracos no tabuleiro, que nada mais são que objetos com textura preta. Se passar o cubo por cima desses objetos o cubo cai do tabuleiro e o jogador perde, bem como também se ultrapassar os limites do tabuleiro.

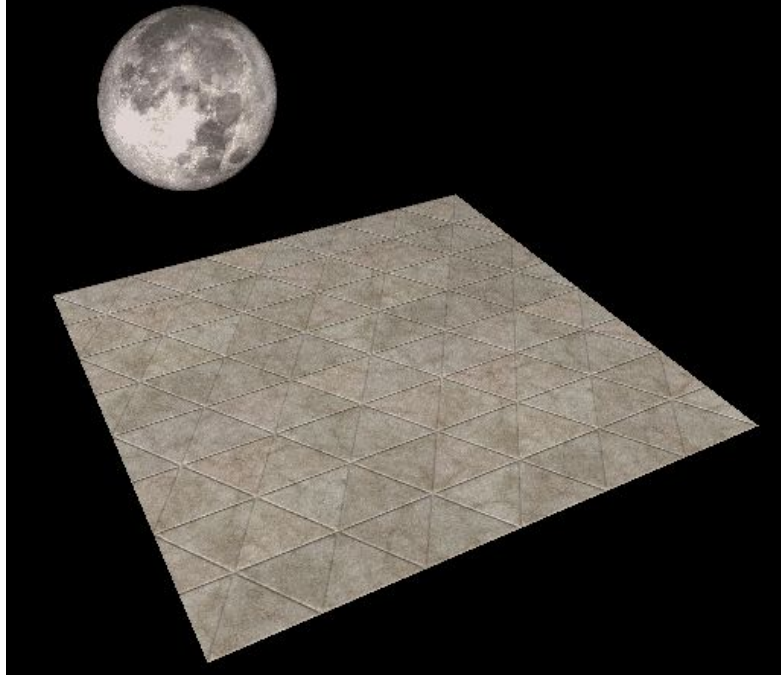
Som

- Foi utilizado recurso de som para dar uma maior experiência ao jogador.



Interação por parte do usuário

```
def render_cube(self):  
    glPushMatrix()  
    glTranslatef(self.coordinates[0],self.coordinates[1],self.coordinates[2])  
    self.cube.render_scene()  
    glPopMatrix()
```



```
def move_forward(self):
    self.coordinates[2] += 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] -= 0.3 * math.sin(math.radians(self.angle))

def move_back(self):
    self.coordinates[2] -= 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] += 0.3 * math.sin(math.radians(self.angle))

def move_left(self):
    self.coordinates[0] += 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[2] += 0.3 * math.sin(math.radians(self.angle))

def move_right(self):
    self.coordinates[0] -= 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[2] -= 0.3 * math.sin(math.radians(self.angle))

def move_top_right(self):
    self.coordinates[2] += 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] += 0.3 * math.cos(math.radians(self.angle))

def move_top_left(self):
    self.coordinates[2] += 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] -= 0.3 * math.cos(math.radians(self.angle))

def move_bottom_right(self):
    self.coordinates[2] -= 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] -= 0.3 * math.cos(math.radians(self.angle))

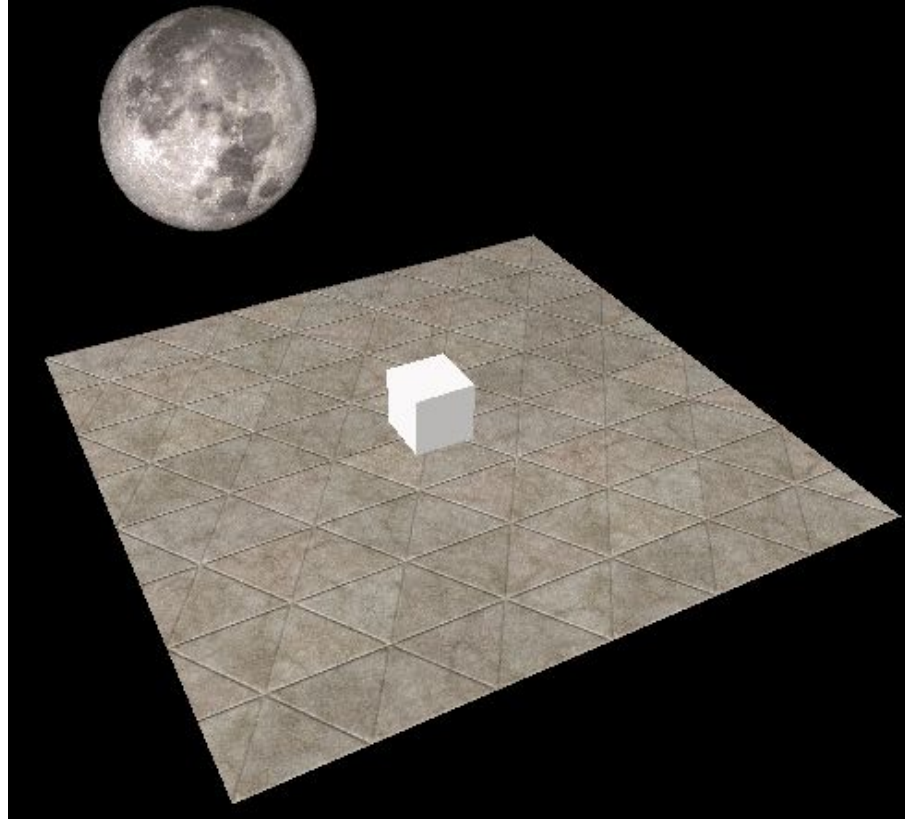
def move_bottom_left(self):
    self.coordinates[2] -= 0.3 * math.cos(math.radians(self.angle))
    self.coordinates[0] += 0.3 * math.cos(math.radians(self.angle))
```



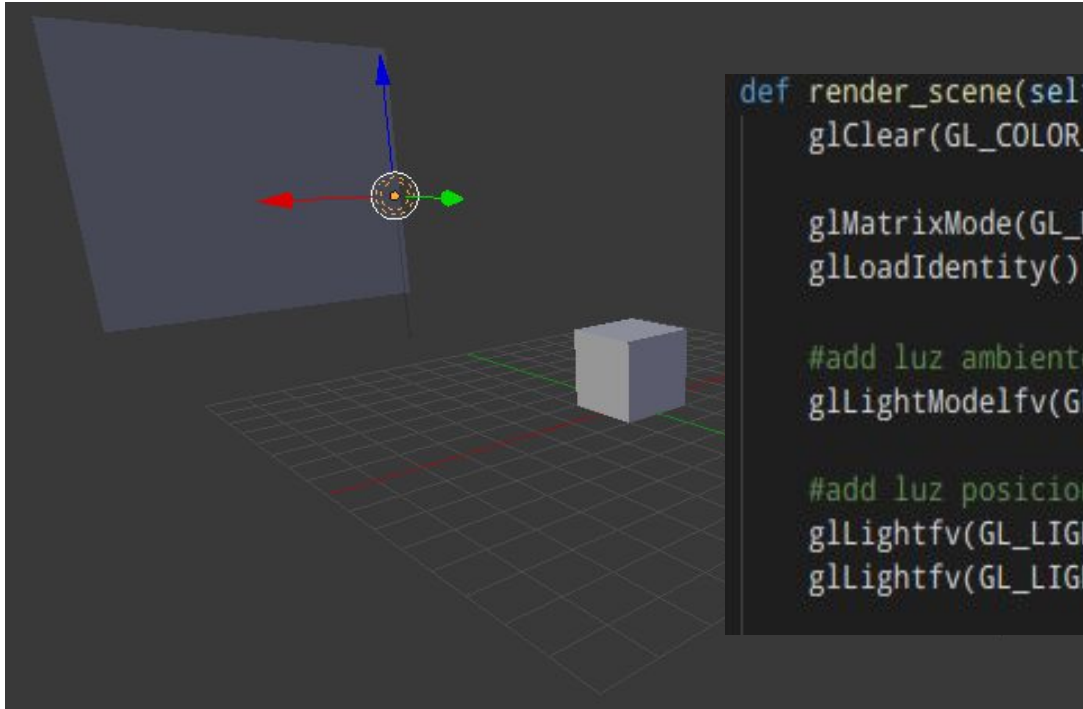
```

def isjumping(self):
    jump = pygame.mixer.Sound("jump.wav")
    if self.coordinates[1] >= 1:
        if self.state == 'JUMPING UP':
            self.coordinates[1] += 0.6
            self.coordinates[2] += 1.8
            if(self.coordinates[1] >= 3):
                jump.play(0)
                self.state = 'DOWN'
        if self.state == 'JUMPING LEFT':
            self.coordinates[1] += 0.6
            self.coordinates[0] += 1.8
            if(self.coordinates[1] >= 3):
                jump.play(0)
                self.state = 'DOWN'
        if self.state == 'JUMPING RIGHT':
            self.coordinates[1] += 0.6
            self.coordinates[0] -= 1.8
            if(self.coordinates[1] >= 3):
                jump.play(0)
                self.state = 'DOWN'
        if self.state == 'JUMPING DOWN':
            self.coordinates[1] += 0.6
            self.coordinates[2] -= 1.8
            if(self.coordinates[1] >= 3):
                jump.play(0)
                self.state = 'DOWN'
        elif self.state == 'DOWN':
            if self.coordinates[1] >= 0:
                self.coordinates[1] -= 0.6
                if self.coordinates[1] == 1:
                    self.state = 'WALKING'
            else:
                self.state = 'WALKING'

```



Pontos de iluminação



```
def render_scene(self):  
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)  
  
    glMatrixMode(GL_MODELVIEW)  
    glLoadIdentity() #começar matriz da origem  
  
    #add luz ambiente:  
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,[0.2,0.2,0.2,1.0])  
  
    #add luz posicionada:  
    glLightfv(GL_LIGHT0,GL_DIFFUSE,[2,2,2,1])  
    glLightfv(GL_LIGHT0,GL_POSITION,[-12,5,0,1])
```

Aplicação de textura

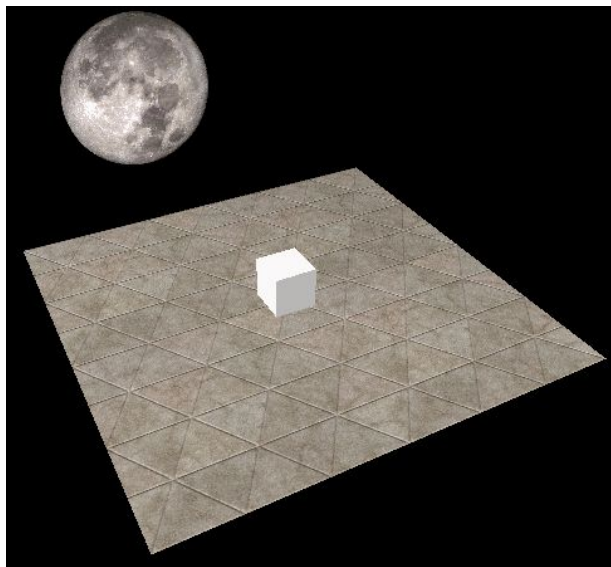
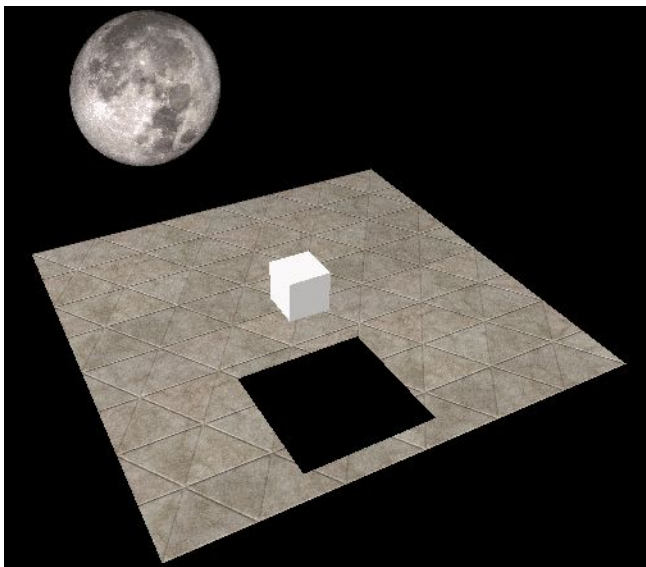
```
def load_texture(filename):  
    """ Essa função irá retornar o id para a texture """  
    textureSurface = pygame.image.load(filename)  
    textureData = pygame.image.tostring(textureSurface, "RGBA", 1)  
    width = textureSurface.get_width()  
    height = textureSurface.get_height()  
    ID = glGenTextures(1)  
    glBindTexture(GL_TEXTURE_2D, ID)  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)  
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, textureData)  
    return ID
```



```
def render_texture(self, textureID, texcoord):
    glEnable(GL_TEXTURE_2D)
    glBindTexture(GL_TEXTURE_2D, textureID)

    glBegin(GL_QUADS)
    for face in self.quad_faces:
        n = face[0]
        normal = self.normals[int(n[n.find("/") + 1:])] - 1
        glNormal3fv(normal)
        for i, f in enumerate(face):
            glTexCoord2fv(texcoord[i])
            glVertex3fv(self.vertices[int(f[:f.find("/")]) - 1])
    glEnd()

    glDisable(GL_TEXTURE_2D)
```



Controle de colisão

```

def meteoro_caindo(self):
    colidiu_play = pygame.mixer.Sound("explosion.wav")
    if self.contador <= 0:
        if self.meteoro_coord[1] >= 1:
            self.meteoro_coord[1] -= self.velocidade_meteoro
            self.meteoro_coord[2] += self.velocidade_meteoro

            if self.colidiu == False:
                colidiu_x=False
                if self.meteoro_coord[0] <= self.coordinates[0] and self.meteoro_coord[0] >= self.coordinates[0]-1 and self.meteoro_coord[1] <= 2:
                    colidiu_x=True
                elif self.meteoro_coord[0] >= self.coordinates[0] and self.meteoro_coord[0] <= self.coordinates[0]+1 and self.meteoro_coord[1] <= 2:
                    colidiu_x=True
                if colidiu_x == True:
                    if self.meteoro_coord[2] <= self.coordinates[2] and self.meteoro_coord[2] >= self.coordinates[2]-1:
                        self.game_over=True
                        self.colidiu=True
                        colidiu_play.play(0)

                    elif self.meteoro_coord[2] >= self.coordinates[2] and self.meteoro_coord[2] <= self.coordinates[2] and self.meteoro_coord[2] <= self.coordinates[2]+1:
                        self.game_over=True
                        self.colidiu=True
                        colidiu_play.play(0)
            else:
                self.z_random = random.randrange(-31,-9,1)
                self.x_random = random.randrange(-11,11,1)
                self.meteoro_coord = [self.x_random, 22, self.z_random]
                self.contador=200-self.desconto
        else:
            self.contador-=1
    print("Pontuacao: "+str(self.pontuacao))

```



```
def caindo(self):
    caindo = pygame.mixer.Sound("caindo.wav")
    #cubo caindo
    #caindo pelo x
    if self.coordinates[0] >= 11 or self.coordinates[0] <= -11:
        self.coordinates[1] -= 1
        self.game_over=True

    #caindo pelo z
    elif self.coordinates[2] >= 11 or self.coordinates[2] <= -11:
        self.coordinates[1] -= 1
        self.game_over=True

    #independente da pos de x e z se y for menor que a pos original entao cai
    elif self.coordinates[1] < 1:
        self.coordinates[1] -= 1
        self.game_over=True

    elif (self.coordinates[2] <= self.buracoz + 3 and self.coordinates[2] >= self.buracoz - 3) and self.buracoz != -12:
        if(self.coordinates[0] <= self.buracox + 3 and self.coordinates[0] >= self.buracox - 3 and self.coordinates[1] <= 1):
            self.coordinates[1] -= 1
            self.game_over=True

    if self.coordinates[1] < 1 and self.coordinates[1] > -1:
        caindo.play(0)
```