CC3501 Modelación y Computación Gráfica para Ingenieros
Primavera 2017

Implementación de un juego en OpenGL

María Teresa Valdivia

Profesora: María Cecilia Rivara

Auxiliares: Sergio Leiva, Cristóbal Muñoz

Ayudantes: Darío Cáceres, Javier Díaz

Introducción

- Objetivo: implementar escenas en 2D usando OpenGL
- Python 2.7
- Juego elegido: Icy Tower

Soluciones por etapas

Diseño de gráficos

- Personaje principal: pez (Clase Jugador)
- Muros: piedras en fondo gris (Clase Muros)
- Plataformas: Piedra, liana y madera (Clase Plataforma)
- Reloj: se dibuja con el segundero avanzando (Clase Reloj)

GL_LINES, GL_QUADS, GL_TRIANGLES, GL_TRIANGLE_FAN
Uso de CC3501Utils de su clase Figura (uso de **herencia**)

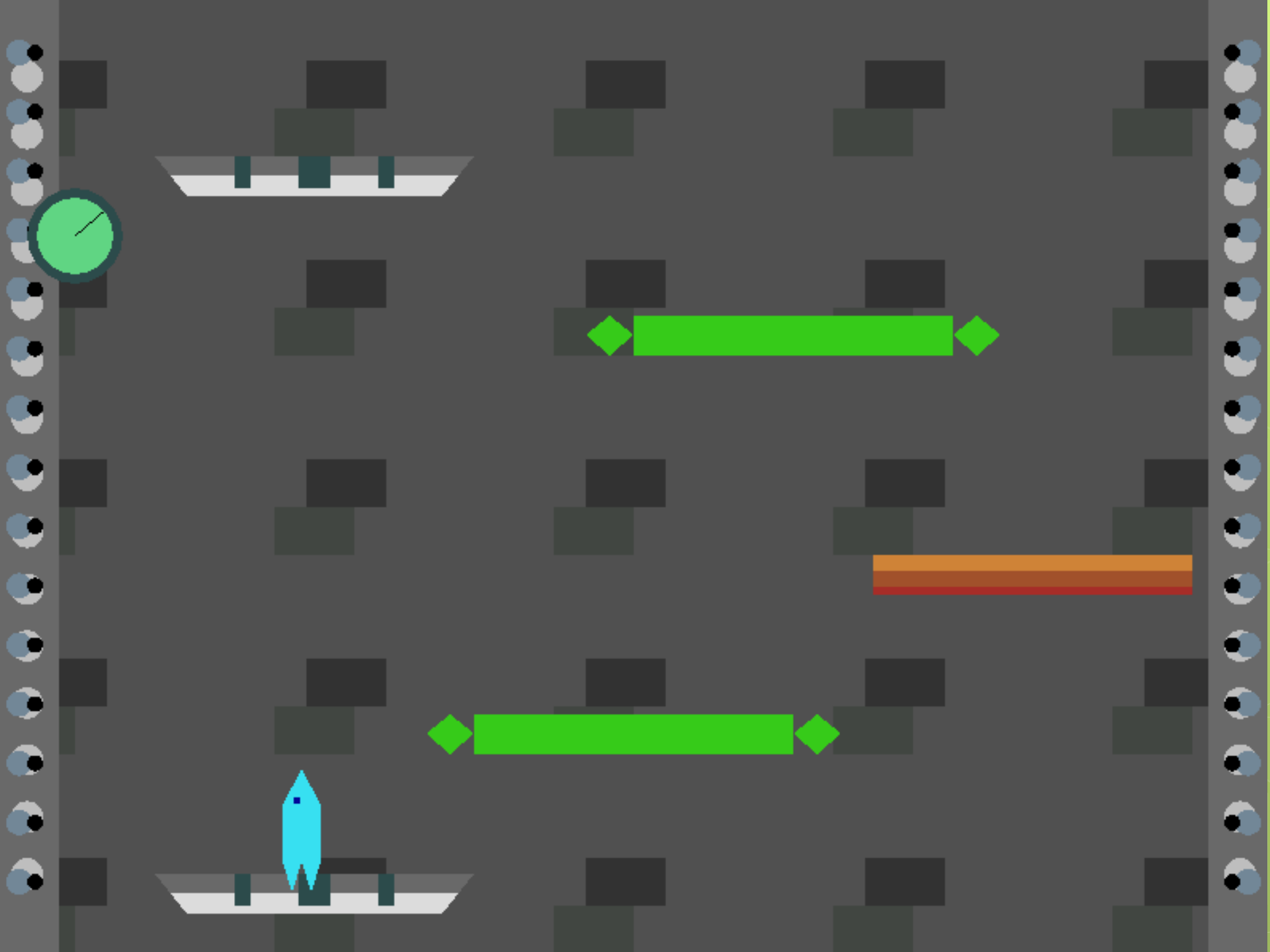
```
pi = 3.1415
fps = 60
clock = pygame.time.Clock()

class Reloj(Figura):
    #variables de instancia del palito
    tamano = 24.
    segundo = 0.
    x_palito = 0
    y_palito = tamano #posiciones iniciales
    seg_final = 30

    def __init__(self, pos=Vector(0, 0), rgb=(1.0, 1.0, 1.0)):
        super(Reloj, self).__init__(pos, rgb)

    ...

    def update(self, fps):
        self.segundo += 1./fps
        if self.segundo >= self.seg_final:
            return
        self.x_palito = self.tamano * sin(pi * self.segundo / 30.)
        self.y_palito = self.tamano * cos(pi * self.segundo / 30.)
```

Hereda método dibujar()

Método usado para actualizar las variables

Método llamado por la función dibujar()

```
def figura(self):

    #Base circular
    glBegin(GL_TRIANGLE_FAN)
    glColor3f(47 / 255.0, 79 / 255.0, 79 / 255.0)
    glVertex2f(0.0, 0.0)

    radio1 = 30
    ang = 2 * pi / 20
    for i in range(20):
        ang_i = ang * i
        glVertex2f(cos(ang_i) * radio1, sin(ang_i) * radio1)

    glVertex2f(1.0 * radio1, 0.0)
    glEnd()

    #Circulo interior
    glBegin(GL_TRIANGLE_FAN)
    glColor3f(97 / 255.0, 218 / 255.0, 137 / 255.0)
    glVertex2f(0.0, 0.0)

    radio2 = 24
    ang = 2 * pi / 20
    for i in range(20):
        ang_i = ang * i
        glVertex2f(cos(ang_i) * radio2, sin(ang_i) * radio2)

    glVertex2f(1.0 * radio2, 0.0)
    glEnd()

    #palillo: que sea actualizable con variables de instancia afuera
    glBegin(GL_LINES)
    glColor3f(0 / 255.0, 0 / 255.0, 0 / 255.0) #negro?
    glVertex2f(0.0, 0.0)
    glVertex2f(self.x_palito, self.y_palito)
    glEnd()
```



Mecánica simple de juego

- El personaje posee velocidad, choca contra las murallas y se para en las plataformas. Hay gravedad.
- Uso de teclas flecha derecha, izquierda y espacio
- Si el personaje se cae, se dibuja solo el muro
- Cámara sigue al personaje sólo cuando sube más arriba de la mitad de la pantalla (Clase Camara)
- 50 plataformas de demo (Clase ControladorPlataforma)

```
class Jugador(Figura):
    vel_x = 0
    vel_y = 0

    ac_y = 1 #gravedad
    ac_x = 0.5
    max_vel_y = -10
    max_vel_x = 10

  def update(self, platformList, camara):
      self.pos += Vector(self.vel_x, self.vel_y - camara.y)
      self.ac_y = 1#reseteo fuera de la plataforma
      self.vel_y -= self.ac_y
      #control muros
      if self.pos.x + 12 >= 760:
          self.pos.x = 760 - 12
          self.vel_x = -self.vel_x

      if self.pos.x - 12 <= 40:
          self.pos.x = 40 + 12
          self.vel_x = -self.vel_x
      #aceleracion en eje y
      if self.vel_y <= self.max_vel_y:
          self.vel_y = self.max_vel_y
      if self.vel_y <= 0 and self.estaSobreAlgunaPlataforma(platformList):
          self.vel_y = 0
          self.ac_y = 0

      else: self.vel_y -= self.ac_y

      #fin de camara
      if self.fueraDePantalla(camara):
          self.vel_y = 0
          self.ac_y = 0
```

Lista de plataformas

Cámara

Al llegar a los muros rebota

¿Cómo se actualiza la velocidad? Dentro del archivo Escena (Juego)

```
keys = pygame.key.get_pressed()

if keys[K_SPACE] and not p.is_jumping():
    p.vel_y += 25
if keys[K_RIGHT]:
    p.vel_x += 1
    if p.vel_x > p.max_vel_x:
        p.vel_x = p.max_vel_x
if keys[K_LEFT]:
    p.vel_x += -1
    if p.vel_x < -p.max_vel_x:
        p.vel_x = -p.max_vel_x
```

```
class Camara:
    def __init__(self, player):
        self.y = 0
        self.player = player

    def update(self):
        if self.player.pos.y - self.y >= 600./2:
            self.y = self.player.pos.y - 600./2
        else:
            self.y = 0
```

Toma al jugador de referencia

Actualiza su posición sólo cuando el jugador supera la mitad de la pantalla

```
class ControladorPlataforma:
    def __init__(self):
        self.lista = []
        self.generar()

    def generar(self):
        #20 plataformas de piedra
        alturas = np.linspace(150, 2100, 19)
        mu = 400
        sigma = 100
        self.lista.append(PlataformaPiedra(Vector(180 + 12, 50)))
        for a in alturas:
            new = PlataformaPiedra(Vector(np.random.normal(loc=mu, scale=sigma), a))
            self.lista.append(new)
        #20 de liana
        alturas +=2050
        for a in alturas:
            new = PlataformaLiana(Vector(np.random.normal(loc=mu, scale=sigma), a))
            self.lista.append(new)
        #y 10 de madera
        alturas += 2050
        for a in alturas:
            new = PlataformaMadera(Vector(np.random.normal(loc=mu, scale=sigma), a))
            self.lista.append(new)
```

Guarda lista de plataformas

Genera plataformas en tres etapas: piedra, liana y madera

Carácterísticas avanzadas

- Movimiento acelerado con velocidad máxima
- Fin de juego: pantalla de fondo luego de 30 segundos

En clase Jugador:

```
#aceleracion en eje x
if self.vel_x > 0:
    self.vel_x -= self.ac_x
elif self.vel_x < 0:
    self.vel_x += self.ac_x
```

En clase ControladorPlataforma (generar()):

```
    for a in alturas:
        new = PlataformaPiedra(Vector(np.random.normal(loc=mu, scale=sigma), a))
        self.lista.append(new)
    #20 de liana
    alturas +=2050
    for a in alturas:
        new = PlataformaLiana(Vector(np.random.normal(loc=mu, scale=sigma), a))
        self.lista.append(new)
    #y 10 de madera
    alturas += 2050
    for a in alturas:
        new = PlataformaMadera(Vector(np.random.normal(loc=mu, scale=sigma), a))
        self.lista.append(new)
```

Dificultades encontradas

- PyOpenGL para Python 3.x no capta el teclado
- Errores al cambiar variables hicieron necesario el uso de un Version controller (Git y GitHub)

Conclusiones

- Se logró el objetivo de la tarea
- El juego quedó en la demo
- OpenGL para dibujar formas complejas requiere muchas líneas de código, pero a partir de primitivas se pueden lograr diversas figuras.