Departamento de Ciencias de la Computación
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
CC3501 Modelación y Computación Gráfica para Ingenieros

IMPLEMENTACIÓN DEL JUEGO FLAPPY BIRD

Reporte T1

Estudiante: Asunción Gómez Colomer
Email: asuncion.gomez@ug.uchile.cl
Número telefónico: +56 9 99 14 09 63

Fecha de entrega: 22 de abril de 2022
Santiago de Chile

1. Solución y especificaciones

Se implementaron tres modelos para el desarrollo del juego: el pájaro, los tubos y el fondo. El personaje principal era el pájaro, el cual puede "volar" con la flecha hacia arriba. Su movimiento es constante en el eje vertical y fija horizontalmente. El objeto del "flappy bird", también tiene un contador de tubos por los que ha pasado, para así conocer los puntos que lleva el usuario.

Los tubos tienen una altura aleatoria, es decir, el espacio vertical entre los pilares varía. Además, cada tubo está separado el uno con el otro. Para el fondo, se utilizó una textura para simular el fondo del juego. Se creó un modelo de este debido a que se le debía aplicar una traslación horizontal hacia la izquierda, por lo que era más fácil manejarlo si existía un objeto.

Además, el juego simulaba las colisiones entre el pájaro con el suelo/cielo y con los pilares. Si el jugador lograba pasar los N tubos, un mensaje aparecía para indicar al usuario que había ganado (ver figura A.1). Lo mismo para el caso contrario, cuando el usuario perdía (ver figura A.2).

Finalmente, gracias a las librerías glfw, openGL, numpy, random y sys, se lograron implementar todas estas funcionalidades. También es importante destacar que fue de gran ayuda los códigos entregados en clases auxiliares como base del programa.

1.1. Arquitectura

Se usó la arquitectura vista en clases: Modelo - Vista - Controlador, en la cual el programa se divide en estas tres grandes capas. En el modelo, se encuentran los objetos, en este caso: el pájaro, los tubos y el fondo. Se define también la lógica para manipularlos (cómo se mueven, se dibujan, etc). Para más detalle ver el código B.1 que muestra el objeto Tubo como ejemplo.

Luego, la Vista es donde se ejecuta el programa principal. Aquí se definen los shaders, se conectan los "callback" del controlador, se dibujan y actualizan los modelos, entre otras cosas. Es todo lo que el usuario ve. Una parte del código se encuentra en el anexo B.2.

Finalmente, el controlador es el encargado de detectar los inputs del usuario, ya sea presionar una tecla o mover el mouse, y actualizar los datos de los modelos. Por ejemplo, cada vez que se presiona la flecha hacia arriba, se le indica al pájaro de aumentar su posición vertical. Cuando se suelta esta tecla, el pájaro se mueve hacia abajo (ver código B.3).

2. Análisis

2.1. Problemas abordados

Las mayores dificultades se tuvieron al momento de detectar las colisiones del pájaro y la traslación del fondo. Para lo primero, se programó el código que se muestra en el anexo B.4. Sin embargo, cuando se puso en marcha, el pájaro perdía antes de colisionar contra el tubo. Para identificar la causa del problema, se decidió quitar las líneas de código que permitían la transparencia en las tex-

turas (glEnable(GLBLEND) y glBlendFunc(GLSRCALPHA, GLONEMINUSSRCALPHA)), para así ver cómo se comportaba el pájaro y los tubos (ver figura A.3). De esta forma se pudo ver que ambas figuras contenían más espacios transparentes de lo esperado, por lo que se tuvo que quitar estos espacios vacíos (gracias a la función B.5) y así, la función que permitía detectar las colisiones funcionó correctamente.

Para la traslación del fondo, se decidió crear un objeto que creara un modelo con la textura de fondo y además, que guardara la posición horizontal. De esta forma, ir actualizando esta variable con el tiempo. También, se creó una textura rectangular de un gran ancho, para luego usar "GL REPEAT" y repetir el fondo varias veces a lo ancho. De esta forma, se logró implementar la traslación del fondo.

2.2. Posibles mejoras

Una implementación que no se logró llevar a cabo fue variar la textura del pájaro para crear una animación en el que el pájaro abría y cerraba las alas. Además, se quería añadir una leve rotación hacia arriba o hacia abajo, para indicar al usuario cuándo el pájaro estaba volando.

También, se buscaría una mejor librería que permitiera dibujar texto y números en la pantalla, en lugar de usar texturas con imágenes de números para mostrar los puntos del juego.

Para una futura versión, se podría añadir también un sonido para indicar al usuario cuándo pasó un pilar o cuando chocó contra uno de este. Asimismo, luego de haber pasado K tubos, hacer el juego más difícil, aumentando al velocidad de los tubos y disminuyendo la distancia entre estos horizontal y verticalmente.

3. Instrucciones de uso

Para jugar, basta con correr el siguiente comando en la terminal (en la carpeta donde se tenga el programa):

```
python flappy_bird.py N
```

El objetivo del juego es hacer pasar al pájaro N tubos. Para esto, se debe presionar la flecha hacia arriba para hacer "volar" el pájaro. Una vez el pájaro haya pasado los N tubos, el usuario habrá ganado. En el caso contrario, si el pájaro colisiona con algún tubo o con el suelo/cielo, entonces el usuario perderá.

Anexo A. Imágenes

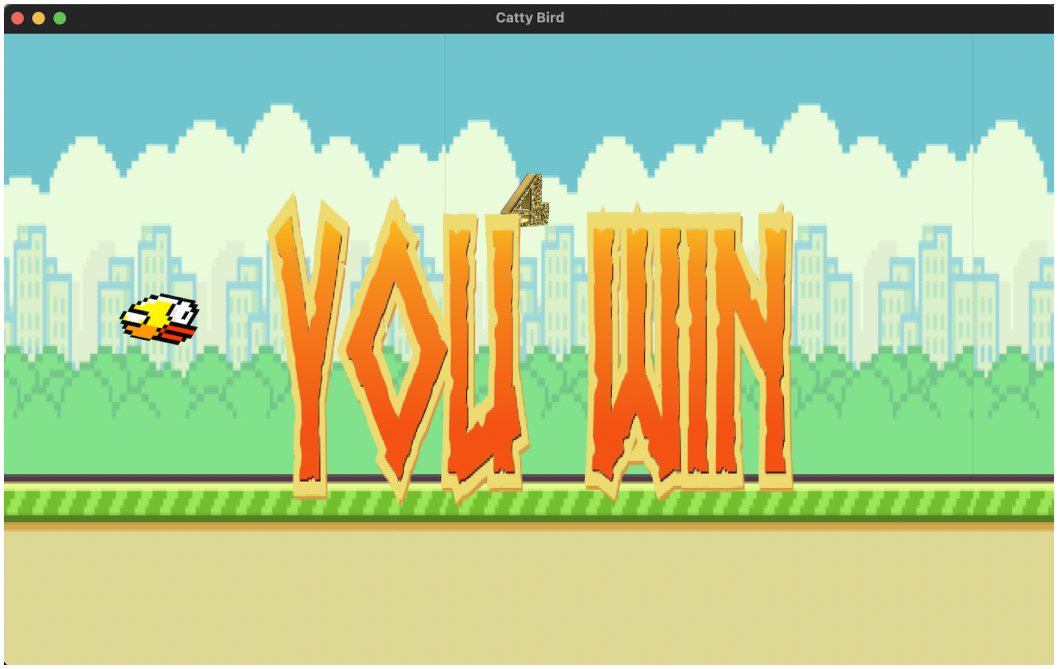

Figura A.1: Usuario gana

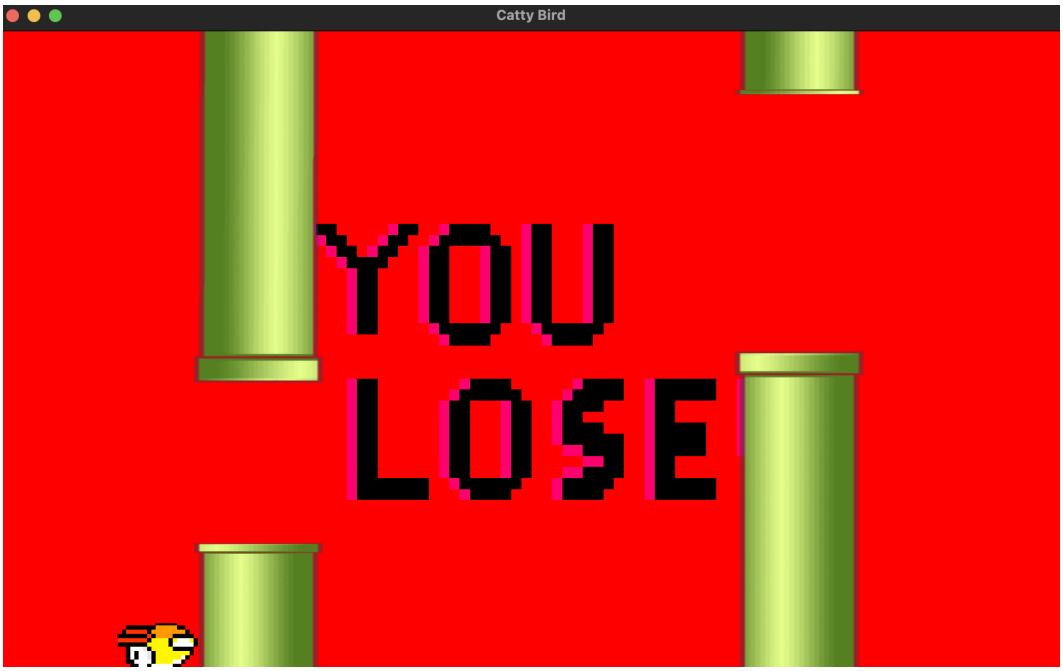

Figura A.2: Usuario pierde

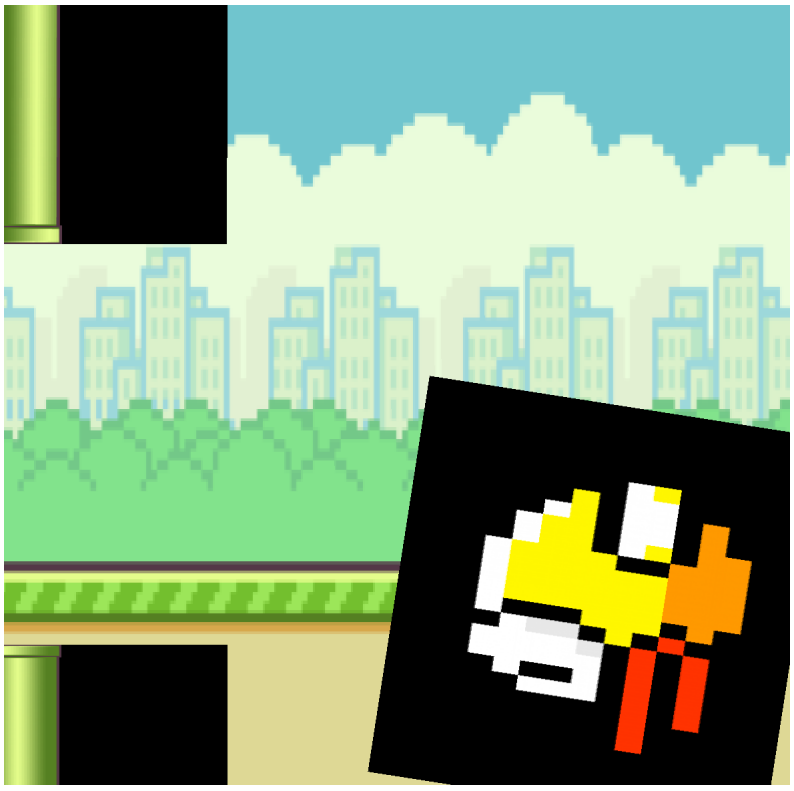

Figura A.3: Juego sin transparencias

Anexo B. Código

Código B.1: Objeto Tube.

```
class Tube(object):

    def __init__(self, pipeline):
        self.pos_x = 1.25
        self.width = 0.25
        # create tube with a random dy
        self.height_inf = choice(arange(0.4, 1.2, 0.1))  # todo make it random
        min_dy = 0.2
        max_dy = 2 - self.height_inf - 0.4
        self.height_sup= choice(arange(min_dy, max_dy, 0.1))

        # tube model
        shape_tube_inf = bs.createTextureQuadAdvance(0.41,0.59,0,1)
        shape_tube_sup = bs.createTextureQuadAdvance(0.41,0.59,0,1)

        gpu_tube_inf = create_gpu(shape_tube_inf, pipeline)
        gpu_tube_sup = create_gpu(shape_tube_sup, pipeline)

        gpu_tube_inf.texture = es.textureSimpleSetup(getImagesPath("tube.png"), GL_REPEAT
    , GL_REPEAT, GL_NEAREST, GL_NEAREST)
```

```
        gpu_tube_sup.texture = es.textureSimpleSetup(getImagesPath("tube.png"), GL_REPEAT
    , GL_REPEAT, GL_NEAREST, GL_NEAREST)

        tube_inf = sg.SceneGraphNode('tube_inf')
        pos_y = -1 + self.height_inf/2 # todo check this translation
        tube_inf.transform = tr.matmul([
            tr.translate(0, pos_y, 0),
            tr.scale(self.width, self.height_inf, 0)
        ])
        tube_inf.childs += [gpu_tube_inf]

        tube_sup = sg.SceneGraphNode('tube_sup')
        pos_y = 1 - self.height_sup/2
        tube_sup.transform = tr.matmul([
            tr.translate(0, pos_y, 0),
            tr.rotationZ(3.14), # todo check this rotation
            tr.scale(self.width, self.height_sup, 0)
        ])
        tube_sup.childs += [gpu_tube_sup]

        tube = sg.SceneGraphNode("tube")
        tube.childs = [tube_inf, tube_sup]

        self.model = tube

    def draw(self, pipeline):
        self.model.transform = tr.translate(self.pos_x, 0, 0)
        sg.drawSceneGraphNode(self.model, pipeline, 'transform')

    def update(self, dt):
        self.pos_x -= dt * 0.5

    def clear(self):
        self.model.clear()
```

Código B.2: Vista Flappy Bird (extracto).

```
# Application loop
while not glfw.window_should_close(window):

    # Using GLFW to check for input events
    glfw.poll_events()

    # Clearing the screen in both, color and depth
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) # todo check this with the
    background

    # Using the time as the x_0 parameter
    if flappy_bird.alive:
        ti = glfw.get_time()
        ci = ti # distance
        dc = ci - c0 # dx distance
```

```
        dt = ti - t0
        t0 = ti
    else:
        dt = 0 # stop the game

    # create tubes with a distance between them
    if(dc > 2):
        #print("create tube c1: ", ci)
        tubeCreator.create_tube(pipeline)
        c0 = ci

    # update position
    tubeCreator.update(dt)
    flappy_bird.update(dt)
    background.update(dt)

    # check if flappy collide with a tube
    flappy_bird.game_lost(tubeCreator)

    # draw the models and background
    if flappy_bird.alive:
        # Setting up the background
        #draw_image(pipeline,2,2,"background")
        background.draw(pipeline)
        # draw the points
        if(flappy_bird.points < 10):
            draw_points(pipeline, flappy_bird.points)
        else:
            draw_points_2(pipeline, flappy_bird.points)
        if(flappy_bird.points == n_tubes): # todo fix this
            #print("WIN!!!!!")
            flappy_bird.win = True
            draw_image(pipeline,1,1,"win")
    else:
        glClearColor(1, 0, 0, 1.0)
        draw_image(pipeline,1,1,"lose")

    tubeCreator.draw(pipeline)
    flappy_bird.draw(pipeline)

    # Once the render is done, buffers are swapped, showing only the complete scene.
    glfw.swap_buffers(window)

# freeing GPU memory
controller.clear_gpu()
glfw.terminate()
```

Código B.3: Controlador Flappy Bird (extracto)

```

class Controller():

```

```
    def on_key(self, window, key, scancode, action, mods):

        #print(key, action)
        # 0 release
        # 1 press

        if not (action == glfw.PRESS or action == glfw.RELEASE):
            return

        if key == glfw.KEY_ESCAPE and action == glfw.PRESS:
            glfw.set_window_should_close(window, True)

        elif (key == glfw.KEY_UP or key == glfw.KEY_SPACE) and action == glfw.PRESS:
            #print("move up")
            if self.flappy_bird.alive: self.flappy_bird.move_up()

        elif (key == glfw.KEY_UP or key == glfw.KEY_SPACE) and action == glfw.RELEASE:
            #print("move down")
            if self.flappy_bird.alive: self.flappy_bird.move_down()

        elif key == glfw.KEY_DOWN and action == glfw.RELEASE:
            #print("key down")
            if self.flappy_bird.alive: self.flappy_bird.move_down()

        else:
            print('Unknown key')
```

Código B.4: Coalición Flappy Bird.

```
for i, tube in enumerate(tubes):
    #print("tube: ", i)
    # tubes axis position
    tube_x_left = tube.pos_x - tube.width/2
    tube_x_right = tube.pos_x + tube.width/2
    tube_y_inf = -1 + tube.height_inf # punto alto del tubo inf
    tube_y_sup = 1 - tube.height_sup # punto bajo del tubo sup

    # checking height: bird same height as the tube
    if((bird_y_inf < (tube_y_inf + alpha_error)) or ((bird_y_sup > (tube_y_sup -
    alpha_error)))):
        # bird collide passing throw the tube
        if((bird_x_left > tube_x_left) and (bird_x_left < tube_x_right)):
            self.alive = False
            self.pos_y = -1 + self.size_bird/2
            tube_creator.die()

        # bird collide at the begining of the tube
        elif((bird_x_right > tube_x_left) and (bird_x_right < tube_x_right)):
            self.alive = False
            self.pos_y = -1 + self.size_bird/2
            tube_creator.die()

```

```
    # different height bird and tube
    else:
        # bird passing throw the tube (at the end)
        if((bird_x_left > tube_x_left + tube.width/2) and (bird_x_left < tube_x_right)):
            if not tube in self.tubes:
                self.tubes.append(tube)
```

Código B.5: Texture Quad Advance

```
def createTextureQuadAdvance(nx0, nx1, ny0, ny1):
    """
    create Texture Quad
    from nx0 to nx1
    from ny0 to ny1
    """

    # Defining locations and texture coordinates for each vertex of the shape
    vertices = [
    #   positions        texture
        -0.5, -0.5, 0.0,  nx0, ny1, # esq inf izq
         0.5, -0.5, 0.0, nx1, ny1, # esq inf der
         0.5,  0.5, 0.0, nx1, ny0, # esq sup der
        -0.5,  0.5, 0.0,  nx0, ny0] # esq sup izq

    # Defining connections among vertices
    # We have a triangle every 3 indices specified
    indices = [
         0, 1, 2,
         2, 3, 0]

    return Shape(vertices, indices)
```

Código B.6: Background

```
class Background(object):

    def __init__(self, pipeline):
        alpha_trans = 100000
        shape_bg = bs.createTextureQuad(alpha_trans, 1)
        gpu_bg = create_gpu(shape_bg, pipeline)
        gpu_bg.texture = es.textureSimpleSetup(getImagesPath("background.png"), GL_REPEAT
    , GL_REPEAT, GL_NEAREST, GL_NEAREST)

        background = sg.SceneGraphNode('background')
        background.transform = tr.scale(alpha_trans,2, 0)
        background.childs = [gpu_bg]

        bg_final = sg.SceneGraphNode('bg_final')
        bg_final.childs += [background]

        self.pos_x = 0
        self.model = bg_final
```

```
18
19
20      def draw(self, pipeline):
21          self.model.transform = tr.translate(self.pos_x, 0, 0)
22          sg.drawSceneGraphNode(self.model, pipeline, 'transform')
23
24      def update(self, dt):
25          self.pos_x -= dt * 0.5
```