

**Instituto Tecnológico y de Estudios Superiores de Monterrey  
Campus Mty**



**Tecnológico  
de Monterrey**

**Materia**

**Modelación de sistemas multiagentes con gráficas computacionales**

**Nombre del archivo**

Evidencia 1. Actividad Integradora

Raymundo Iván Díaz Alejandro - A01735644

Erwin Porras Guerra - A01734881

Daniela Lozada Bracamontes - A01736549

Gerardo Deustúa Hernández - A01736455

**Grupo 401**

29 de Septiembre del 2023

En el marco de la actividad integradora realizada, se llevó a cabo la conjunción de dos tecnologías clave en el desarrollo de aplicaciones gráficas y de simulación: Mesa y OpenGL. Este proyecto se basó en la plataforma previamente desarrollada para la gestión de recolección de basura, la cual emplea las herramientas de OpenGL y el lenguaje de programación Python. El objetivo central de esta iniciativa era poner a prueba y demostrar nuestros profundos conocimientos en OpenGL y Mesa, y, al mismo tiempo, fusionar ambas tecnologías para crear una aplicación con representaciones tridimensionales de vehículos recolectores de basura y los propios residuos.

Una de las tareas fundamentales que se abordaron en este proyecto fue la implementación de un algoritmo de pathfinding. Esta función permitió a los recolectores de basura navegar de manera eficiente a través del entorno simulado, evitando de manera inteligente cualquier obstáculo que fueran encontrando en su camino, especialmente los desechos generados en el canvas.

El diseño integral de este sistema se compone de dos partes fundamentales, cada una desempeñando un papel esencial en el funcionamiento del proyecto. En primer lugar, encontramos el backend, que constituye la parte lógica del sistema y se ha desarrollado en Mesa, en ella se ejecutan todas las instrucciones y operaciones de los agentes encargados de la recolección de cajas. Además, en el backend se implementan y gestionan las variables y parámetros necesarios para garantizar una comunicación fluida y eficiente con la parte gráfica del proyecto.

Por otro lado, el frontend del proyecto asume la parte gráfica. Este componente, desarrollado en OpenGL, se encarga de generar modelos tridimensionales que reflejan el funcionamiento y la interacción de los agentes del backend. La colaboración estrecha entre Mesa y OpenGL en este proyecto ha culminado en un sistema ejecutable.

Dentro del **backend** se encuentran dos archivos importantes.

#### **robot2.py**

En este archivo se crearon las instrucciones de mesa sobre el funcionamiento del sistema. dividiendo el archivo en 3 agentes diferentes agente bot: con todas los métodos del robot recolector tales como pathfinding, movetopos, pickup, sweep, moves y step; agente Trash: que es la generación de basura de forma random en el grid, con métodos como move que permite el movimiento de esta para la ejecución del incinerador; agente Incinerador: que es interacciona directamente con la basura y los robots con métodos como checkTrash, burn, choosebot. Uniendo a todos los agentes en la función floor que se encarga de su multigeneración.

```
mesaWrapper / back / robot2.py ↑ Top

Code Blame Raw Copy Download Edit Search

19
20 > class Bot(Agent):
334     self.trash.move(self.pos[0], self.pos[1])
335
340
341 > class Incinerador(Agent):
380     self.condition = self.free # yellow
381
382
383 > class Trash(Agent):
396     self.model.grid.move_agent(self, (x, y))
397
398
399 basura_total = 0
400
401
402 > class Floor(Model):
493     self.datacollector.collect(self)
```

## backend2.py

En este archivo se crea la comunicación entre el frontend y backend. pasando las variables y parámetros de cada agente necesarios para el funcionamiento de OpenGL. os datos se transmiten de manera eficiente, permitiendo que el frontend traduzca y represente de manera visual los procesos ejecutados por los agentes en el backend.

```
mesaWrapper / back / backend2.py Copy

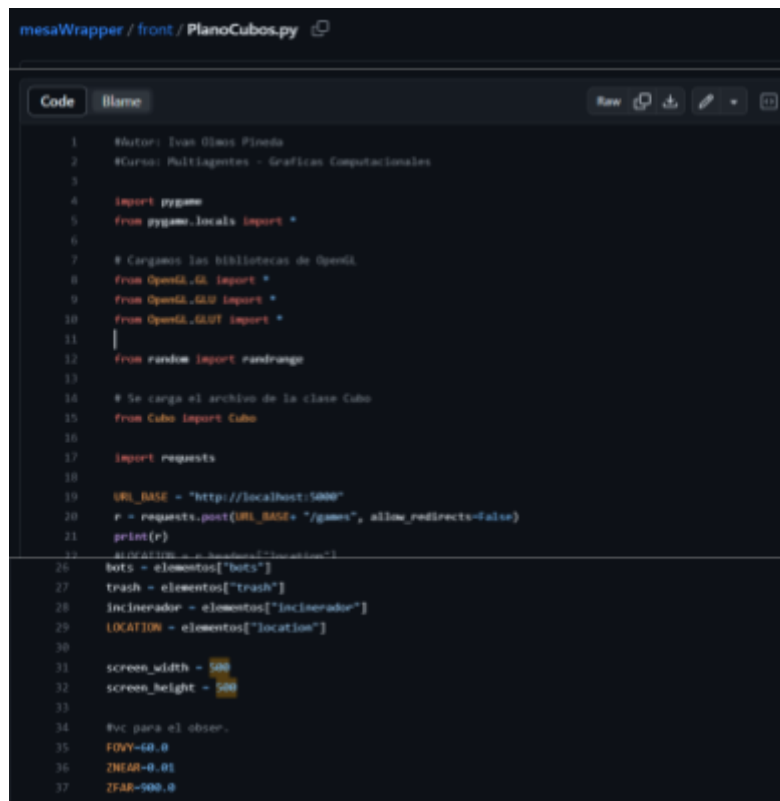
Code Blame Raw Copy Download Edit Search

1 import flask
2 from flask.json import jsonify
3 import uuid
4 from robot2 import Floor
5 from robot2 import Bot
6 from robot2 import Trash
7 from robot2 import Incinerador
8 games = {}
9
10 app = flask.Flask(__name__)
11
12 @app.route("/games", methods=["POST"])
13 > def create():
27     return jsonify({"bots": bots, "trash": trash, "incinerador": incinerador, 'location': f"/games/{id}"})
28
29 @app.route("/games/<id>", methods=["GET"])
30 > def queryState(id):
44     return jsonify({"bots": bots, "trash": trash, "incinerador": incinerador})
45
```

Dentro del **frontend** se encuentran tres archivos importantes.

### planoCubos.py

Dentro de este archivo se implementa la visualización del entorno gráfico mediante una cámara de visualización fija que permite al usuario observar cada detalle de la ejecución del sistema. También en este apartado se crean los objetos a partir de la clase cubo de los diferentes agentes(bots, trash e incinerador), se diseña el ambiente gráfico como luces y canvas del tablero. Finalmente y aún más importante es el archivo del frontend en el que se hace conexión con la parte de backend.



```
1 # Autor: Ivan Olmos Pineda
2 # Curso: Multiagentes - Graficas Computacionales
3
4 import pygame
5 from pygame.locals import *
6
7 # Cargamos las bibliotecas de OpenGL
8 from OpenGL.GL import *
9 from OpenGL.GLUT import *
10 from OpenGL.GLU import *
11
12 from random import randrange
13
14 # Se carga el archivo de la clase Cubo
15 from Cubo import Cubo
16
17 import requests
18
19 URL_BASE = "http://localhost:5000"
20 r = requests.post(URL_BASE + "/games", allow_redirects=False)
21 print(r)
22
23 # Se cargan los modelos 3D
24
25 bots = elementos["bots"]
26 trash = elementos["trash"]
27 incinerador = elementos["incinerador"]
28 LOCATION = elementos["location"]
29
30
31 screen_width = 500
32 screen_height = 500
33
34 # Func para el obser.
35 FOVY=60.0
36 ZNEAR=0.01
37 ZFAR=500.0
```

### Cubo.py

En este archivo se declaró la clase cubos donde se asignaron cada uno de los atributos que llevará cada modelo de los robots realizado así como sus métodos para los distintos comportamientos que estos tienen, como el moverse al centro con el recolector, recoger basura o rotar. A la par se hace conexión con el archivo “objloader.py” para que dentro del entorno gráfico se puedan cargar modelos 3D.

```

20     class Cubo:
26         def __init__(self, dim, velocidad):
27
28             #####OP
29             #Estados
30             self.objetivo = None
31             self.recolectando = False #No se uso
32             self.carrying_basura = None
33
34             self.angulo = 0
35             self.direction = (0, 0, 1) #Dirección inicial
36             self.objetivo_direction = (0, 0, 1)
37             self.valid = 0
38
39
40             #Se inicializan las coordenadas de los vertices del cubo
41             self.vertexCoords = [
42                 1,1,1,  1,1,-1,  1,-1,-1,  1,-1,1,
43                 -1,1,1, -1,1,-1, -1,-1,-1, -1,-1,1 ]
44             #Se inicializan los colores de los vertices del cubo
45             self.vertexColors = [
46                 1,1,1,  1,0,0,  1,1,0,  0,1,0,
47                 0,0,1,  1,0,1,  0,0,0,  0,1,1 ]
48             #Se inicializa el arreglo para la indexación de los vertices
49             self.elementArray = [
50                 0,1,2,3, 0,3,7,4, 0,4,5,1,
51                 6,2,1,5, 6,5,4,7, 6,7,3,2 ]
52
53             self.DimBoard = dim
54

```

```

def update(self, new_x, new_z):
    self.Position[0] = new_x
    self.Position[2] = new_z

def updatevalid(self,value):
    self.valid = value

```

### objloader.py

Este archivo contiene todas las funciones para realizar el renderizado de los objetos 3D junto con la iluminación de los mismos. Los objetos que se cargarán tendrán que tener la terminación “.obj” donde se definirán cada vértice que conforma al objeto 3D, a la par es necesario tener un

archivo con terminación “.mtl” que será el encargado de describir el comportamiento de las luces que tiene el modelo 3D.

```
Code Blame Raw Copy Download Edit Search

1  import os
2  import pygame
3  from OpenGL.GL import *
4
5
6  class OBJ:
7      generate_on_init = True
8      @classmethod
9  >  def loadTexture(cls, imagefile):
18  """    return texid
19
20      @classmethod
21  >  def loadMaterial(cls, filename):
41  """    return contents
42
43  >  def __init__(self, filename, swapyz=False):
90  """    self.generate()
91
92  >  def generate(self):
117  """    glEndList()
118
119      def render(self):
120          glCallList(self.gl_list)
121
122      def free(self):
123          glDeleteLists([self.gl_list])
```

## Instrucciones de Ejecución

En una terminal acceder al directorio back y ejecutar backend2.py

```
cd back
```

```
python3 backend2.py
```

En otra terminal acceder al directorio front y ejecutar PlanoCubos.py

```
cd front
```

```
python3 PlanoCubos.py
```

Asegurarse de correr en un ambiente conda que contenga las extensiones de MESA y OpenGL necesarias.

## Problemas

Error al borrar objetos en el ambiente gráfico: Ocasionalmente cuando una basura es depositada al incinerador o es aproximada a este, se genera una discrepancia que ocasiona que esta no se borre del ambiente gráfico. Este error solo se refleja en el ambiente gráfico, desde la parte lógica se logra borrar correctamente el objeto y este no es considerado por los agentes.

## Reflexiones

### Erwin Porras Guerra - A01734881:

Las contribuciones que yo hice al código vienen más del lado gráfico de OpenGL, ya que en un inicio ayude a corregir ciertos problemas que había con los objetos que no respetaban los límites establecidos por el plano, otro de los aportes que realice fue en el movimiento del modelo inicial del coche para que este pueda moverse al centro del plano y que las basuras se eliminaran una vez que estas lleguen al centro. Los últimos aportes realizados por mi parte fueron el suavizar la rotación de los modelos cuando estos cambiaban de dirección y por último el cambiar el modelo inicial de los coches por un objeto 3D modelado con anterioridad.

En un comienzo tuve muchos problemas al realizar la suavización de la rotación del coche, ya que tuve que trabajar con la función `rotate()` de OpenGL y fue una función con la que no estaba tan familiarizado, a la par tuvimos problemas con la versión inicial del código ya que aunque los coches ya podían recoger las basuras del plano, estos solo la movían al centro y no se eliminaban, lo que nos causó ciertos conflictos en la ejecución ya que los robots seguían agarrando los cubos que ya se encontraban en el centro del plano.

### Daniela Lozada Bracamontes- A01736594:

Iniciando con mis implementaciones al proyecto, en la parte lógica de mesa implemente que el incinerador tenga estados dependiendo si se encuentra en uso o no y que los robots respeten los estados para que no se amontonen en caso de que todos basura por desechar. Para la parte gráfica estuvimos trabajando en el movimiento, comportamiento y generación base de los agentes en el entorno gráfico, tanto robots con posiciones aleatorias como las basuras que los robots recolectarán igual en posiciones aleatorias. Por otra parte, implemente la función necesaria para que los planos que conforman al vehículo tengan texturas personalizadas. A la par, realicé el dibujo y funcionalidad de la plataforma para que esta pueda subir y bajar dependiendo si es que el vehículo está transportando basura o no.

De las implementaciones más complicadas de todas las que hice fue la de poder hacer que la plataforma suba o baje sin que el modelo del vehículo se vea afectado por el mismo, ya que la plataforma sigue estando en la clase de los cubos y estuvo complicado el darle un comportamiento independiente a plataforma. También tuve problemas al implementar las texturas al modelo del coche ya que lo estábamos trabajando como un objeto únicamente en lugar de dividirlo por los distintos planos que conforman al coche.

## **Raymundo Iván Díaz Alejandro - A01735644:**

Por mi parte, yo estuve trabajando en la funcionalidad principal de los vehículos que era el movimiento de los mismos, la recolección de basura y hacer que lleve la basura al plano que representa al incinerador, a la vez trabaje junto con mi compañera en la parte gráfica de las plataformas para que una vez que la basura sea recogida, esta se vaya a la parte de la plataforma y se mueva junto con el vehículo. También realicé la primera versión de la rotación de los vehículos cuando cambian dirección y el pathfinding de los coches junto con mi compañero Erwin para delimitar ciertos comportamientos en la ejecución.

Más que nada tuve problemas al implementar la rotación inicial de los vehículos, ya que en las primeras versiones tenían un movimiento muy abrupto y no podía suavizar esa rotación. A su vez estuvimos un buen rato mi compañera y yo a que una vez que el vehículo encontrará una basura y levantara su plataforma y la basura cambiara de posición a donde se encontraba la plataforma.

## **Gerardo Deustúa Hernández - A01736455:**

En mi caso, me encargué en un inicio junto a Daniela Lozada a realizar las bases del archivo generando los cubos en un plano con color sólido y que estos cubos se movieran respetando los límites establecidos en el tablero que en el caso del código fue de un tamaño de 200 x 200 unidades de medida. La segunda contribución principal de mi parte fue el modelado inicial de los coches recolectores de basura por medio de distintos planos que conformarán al vehículo y como de todas las texturas que envuelven a cada plano respectivo para que tenga la apariencia de un vehículo .

En donde más tuve problemas fue al inicio del código ya que pasar de trabajar OpenGL en C++ a python fue un cambio muy fuerte, ya que aunque la estructura era muy parecida al primer lenguaje y en python es más fácil trabajar en OpenGL, tuve que volver a aprender a ocupar esta herramienta desde 0 y las bases de como crear objetos que puedan reaccionar al entorno digital entre distintos objetos o con el entorno en general. A la par en el modelado de la primera versión de los coches me costó un poco el entender cómo es que OpenGL procesa las matrices de vértices y la forma en la que se ponen texturas a estos planos.

## **Repositorios:**

### **Liga de repositorio con contribuciones:**

<https://github.com/ErwinPo/Cubos/commits/master>

### **Liga de repositorio final:**

<https://github.com/ErwinPo/mesaWrapper>