

Tarea 5
Sistemas operativos
Brayan Poloche- Karen Garcia

- **Actividad 1**

- **Seguidor de Línea con Tkinter**

1. Se procedió a crear la estructura de carpetas para organizar adecuadamente el desarrollo del proyecto del seguidor de línea.

```
brayan@brayan-laptop:~$ mkdir -p ~/proyecto_algoritmos/seguidor_linea
```

- **mkdir**: Comando para crear directorios (carpetas).
 - **-p**: Opción que crea toda la ruta completa, incluyendo subdirectorios que aún no existan (por ejemplo, si ~/proyecto_algoritmos no existe, lo crea junto con seguidor_linea).
 - **~/**: Representa el directorio home del usuario actual.
 - **proyecto_algoritmos/seguidor_linea**: Ruta de carpetas a crear, donde se almacenará el proyecto del carro seguidor de línea.
- 2. Una vez creada la carpeta, se accedió a ella mediante el comando:

```
cd ~/proyecto_algoritmos/seguidor_linea
```

- **cd**: Comando para **cambiar de directorio** (entrar a una carpeta).
 - **~/proyecto_algoritmos/seguidor_linea**: Ruta a la carpeta recién creada.
- 3. Una vez ubicado en la carpeta del proyecto, se procedió a clonar el repositorio que contiene el código base del seguidor de línea.

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ git clone https://github.com/dialejobv/Sistemas_Operativos.git
Clonando en 'Sistemas_Operativos'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 0), reused 11 (delta 0), pack-reused 0 (from 0)
Recibiendo objetos: 100% (14/14), 6.15 KiB | 6.15 MiB/s, listo.
```

- **git**: Comando que invoca el sistema de control de versiones Git.
 - **clone**: Copia un repositorio remoto (en línea) hacia tu máquina local.
 - **https://github.com/...**: URL del repositorio que se desea clonar. El contenido se descarga a una carpeta llamada **Sistemas_Operativos** dentro del directorio actual.
4. Se movieron los archivos desde la subcarpeta al directorio actual utilizando el comando **mv**, con el objetivo de reorganizar la estructura del proyecto y dejar únicamente los archivos necesarios en el directorio principal.

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ mv Sistemas_Operativos/"2) Carro_tkinter"/*.
```

- **mv**: Comando para mover (o renombrar) archivos o carpetas.
 - **Sistemas_Operativos/"2) Carro_tkinter"/***: Ruta de origen. Se está accediendo a todos los archivos (*) dentro del directorio **"2) Carro_tkinter"** ubicado en **Sistemas_Operativos**.
 - **..**: Representa el directorio actual. Es decir, los archivos serán movidos a la carpeta actual (**~/proyecto_algoritmos/seguidor_linea**).
5. Posteriormente, se eliminó la carpeta innecesaria mediante el comando **rm -rf**, con el fin de mantener una estructura de archivos limpia y ordenada.

```
rm -rf Sistemas_Operativos
```

- **rm**: Comando para eliminar archivos o carpetas.

- **-r**: Elimina recursivamente, es decir, incluye subdirectorios y sus archivos.
- **-f**: Fuerza la eliminación sin pedir confirmación, incluso si los archivos están protegidos contra escritura.
- **Sistemas_Operativos**: Carpeta a eliminar, ya vacía tras mover su contenido útil.

6. Finalmente, se utilizó el comando **ls** para listar el contenido del directorio y confirmar que el archivo **main.py** fue trasladado correctamente al directorio principal del proyecto.

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ ls
main.py
```

- **ls**: Lista nombres de archivos y carpetas

7. Se instala el módulo Tkinter para Python 3.

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ sudo apt install python3-tk
[sudo] contraseña para brayan:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  blt libtk8.6 tk8.6-blt2.5
Paquetes sugeridos:
  blt-demo tk8.6 tix python3-tk-dbg
Se instalarán los siguientes paquetes NUEVOS:
  blt libtk8.6 python3-tk tk8.6-blt2.5
0 actualizados, 4 nuevos se instalarán, 0 para eliminar y 101 no actualizados.
Se necesita descargar 1.516 kB de archivos.
Se utilizarán 4.929 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://co.archive.ubuntu.com/ubuntu noble/main amd64 libtk8.6 amd64 8.6.14-1build1 [779 kB]
```

- **sudo**: Ejecuta el comando como administrador/root. Necesario para instalar software.
- **apt**: Es el gestor de paquetes en sistema Ubuntu
- **install**: Indica que se va a realizar una instalación de paquetes.
- **python3-tk**: Es el nombre del paquete que contiene Tkinter para Python 3.

8. Se abre el archivo **main.py** en el editor Nano para escribir código en él.

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ nano main.py
```

- **nano**: es un editor de texto de línea de comandos que permite crear o editar archivos directamente desde la terminal.
- **main.py**: es el nombre del archivo Python que se desea editar. Si el archivo no existe, el comando lo creará automáticamente.

9. Se edita el archivo main.py para definir las clases del controlador PID y del carro seguidor de línea, que incluye los parámetros, sensores y funciones de control.

```
GNU nano 7.2 main.py
import tkinter as tk
import math
import time

class PIDController:
    def __init__(self, Kp, Ki, Kd):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.prev_error = 0
        self.integral = 0

    def compute(self, error, dt):
        self.integral += error * dt
        derivative = (error - self.prev_error) / dt if dt > 0 else 0
        output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
        self.prev_error = error
        return output

class LineFollowerCar:
    def __init__(self, canvas):
        self.canvas = canvas
        self.car_x = 100
        self.car_y = 530
        self.car_angle = 0
        self.speed = 1
        self.sensor_distance = 10
        self.sensor_offset = 15
        self.pid = PIDController(0.6, 0.001, 0.1)
        self.last_time = time.time()

        self.body = self.create_car_body()
        self.left_wheel = self.create_wheel()
        self.right_wheel = self.create_wheel()
        self.sensor_left = self.create_sensor()
        self.sensor_right = self.create_sensor()
        self.update_car()

    def create_car_body(self):
        return self.canvas.create_polygon(
            [0, 0, 50, 0, 50, 30, 0, 30],
```

- **tkinter**: crear la interfaz gráfica (dibujar el carro, sensores, etc.).

- `math`: realizar operaciones matemáticas.
- `time`: medir intervalos de tiempo para el controlador PID.
- `Kp`, `Ki`, `Kd`: constantes del control proporcional, integral y derivativo.
- `compute(...)`: calcula la salida del PID a partir del error actual.
- `self.canvas`: referencia al área de dibujo.
- `self.car_X`, `self.car_Y`: posición inicial.
- `self.sensor_distance`, `sensor_offset`: configuración de los sensores.
- `self.pid`: se crea un objeto del controlador PID con parámetros específicos.
- `self.body`, `self.left_wheel`, etc.: componentes gráficos del carro.
- `self.canvas.create_polygon(...)` es una función de `tkinter` que dibuja ese rectángulo sobre el área visual del programa.
- `create_wheel`: Crea las ruedas.
- `create_sensor`: Crea los sensores.

```

GNU nano 7.2 main.py *
50, 491,
51, 492,
52, 493,
53, 493,
54, 494,
55, 495,
56, 495,
56, 496,
57, 496,
58, 496,
58, 497,
59, 497,
60, 497,
60, 498,
60, 498,
60, 498,
61, 498,
62, 498,
63, 498,
64, 498,
64, 498,
65, 498,
66, 498,
67, 498,
]

# Crear pista en canvas (usa la lista plana)
track = canvas.create_line(*track_coords, width=6, fill='#34495E', smooth=True, splinesteps=36)

# Inicializar carro
car = LineFollowerCar(canvas)

def game_loop():
    car.move()
    canvas.create_oval(car.car_x-2, car.car_y-2, car.car_x+2, car.car_y+2, fill='#F1C40F', outline='')
    window.after(30, game_loop)

game_loop()
window.mainloop()

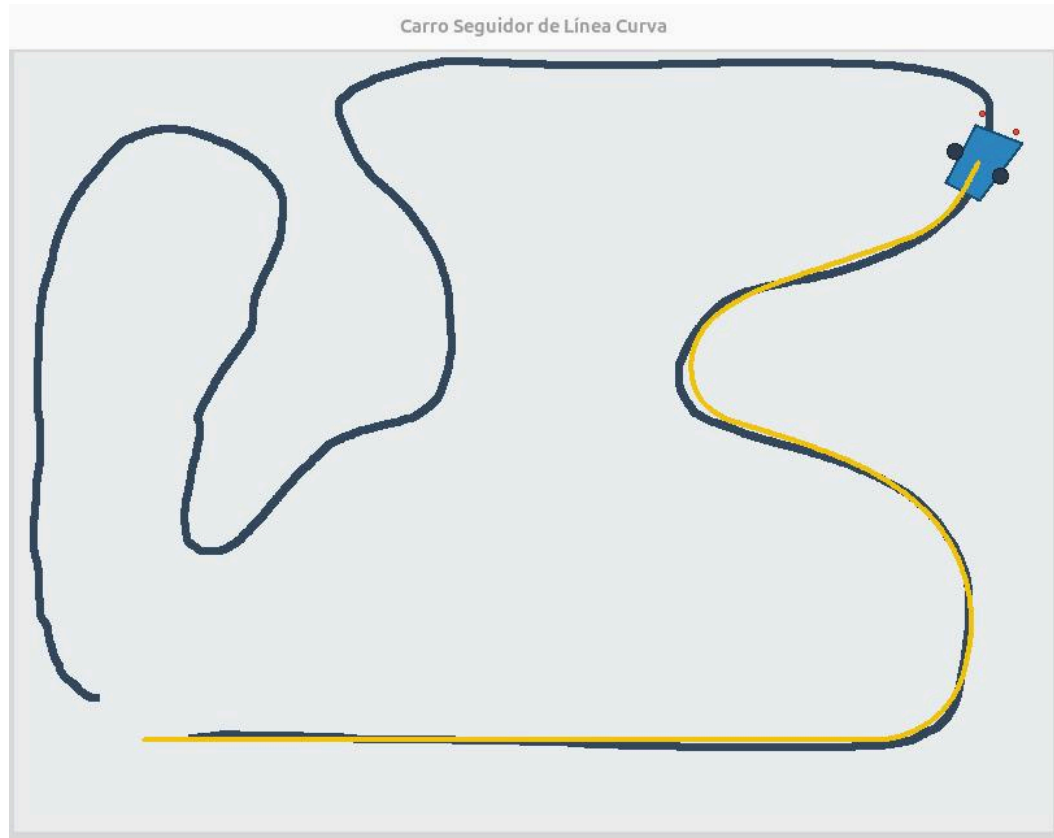
```

- La lista representa una serie de **coordenadas (x, y)** que definen el camino (línea) que seguirá el carro.
- Se usa para dibujar la **trayectoria curva o línea guía** en la que se moverá el vehículo.
- **canvas.create_line(...)**: dibuja una línea en el lienzo **canvas**.
- ***track_coords**: pasa todos los puntos de la lista como argumentos (lista plana).
- **width=6**: grosor de la línea (la pista).
- **fill='#34495E'**: color de la pista (gris azulado).
- **smooth=True**: suaviza la línea (la convierte en curva).

- `splinsteps=36`: define la suavidad de las curvas (más alto = más suave).
- Se crea un objeto car de la clase LineFollowerCar y se le pasa el canvas para que pueda dibujarse en la pista.
- `car.move()`: mueve el carro (asume que el método `move()` controla el desplazamiento y sensores).
- `canvas.create_oval(...)`: dibuja un pequeño punto amarillo en la posición actual del carro.
- `window.after(30, game_loop)`: ejecuta `game_loop` nuevamente después de 30 milisegundos (**actualización periódica** → animación del carro).
- `game_loop()`: arranca el movimiento del carro.
- `window.mainloop()`: mantiene la ventana abierta y actualiza constantemente el GUI de `tkinter`.

10. Se ejecuta el editor Nano y se visualiza el resultado final del funcionamiento del carro seguidor de línea

```
brayan@brayan-laptop:~/proyecto_algoritmos/seguidor_linea$ python3 main.py
```



- **Actividad 2**

- **Naves Espaciales**

1. Se clonó el repositorio del juego de nave espacial desde GitHub. Luego, se accedió al directorio correspondiente y se editó el archivo fuente principal, donde se realizaron las siguientes modificaciones:


```

import pygame
import random
import os

# Inicializar Pygame
pygame.init()

# Configuración de pantalla
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("SpaceMax Defender")

# Definir tamaños de sprites
PLAYER_SIZE = (100, 100)
ENEMY_SIZE = (50, 50)
BULLET_SIZE = (50, 20)

# Cargar imágenes
current_path = os.path.dirname(__file__)
assets_path = os.path.join(current_path, 'assets')
img_path = os.path.join(assets_path, 'images')

player_img = pygame.transform.scale(
    pygame.image.load(os.path.join(img_path, 'player.png')).convert_alpha(),
    PLAYER_SIZE
)

enemy_imgs = [
    pygame.transform.scale(pygame.image.load(os.path.join(img_path, 'enemy1.png')).convert_alpha(), ENEMY_SIZE),
    pygame.transform.scale(pygame.image.load(os.path.join(img_path, 'enemy2.png')).convert_alpha(), ENEMY_SIZE),
    pygame.transform.scale(pygame.image.load(os.path.join(img_path, 'enemy3.png')).convert_alpha(), ENEMY_SIZE)
]

bullet_img = pygame.transform.scale(
    pygame.image.load(os.path.join(img_path, 'bullet.png')).convert_alpha(),
    BULLET_SIZE
)

bullet_img = pygame.transform.rotate(bullet_img, -270)

```

- **pygame**: biblioteca para gráficos, sonido y entrada de usuario.
- **random**: para generar valores aleatorios (por ejemplo, posiciones o enemigos).
- **os**: Para manejar rutas de archivos.
- **pygame.init()**: Inicializa todos los módulos de pygame necesarios (pantalla, audio, etc.).
- **WIDTH, HEIGHT = 800, 600**: Define el tamaño de la ventana del juego (800x600 píxeles).
- **set_mode()**: Crea la ventana del juego.

- `set_caption()`: Establece el título que se ve en la barra de la ventana.
- `PLAYER_SIZE = (100, 100)`, `ENEMY_SIZE = (50, 50)`, `BULLET_SIZE = (50, 20)`: Tamaños de las imágenes del jugador, enemigos y balas, respectivamente.
- `current_path = os.path.dirname(__file__)` , `assets_path = os.path.join(current_path, 'assets')`, `img_path =` Obtiene la ruta del directorio actual y construye la ruta hacia assets/images.
- `player_img = pygame.transform.scale(pygame.image.load(os.path.join(img_path, 'player.png')).convert_alpha(), PLAYER_SIZE)`: Carga la imagen del jugador (player.png) y la redimensiona.
 - `convert_alpha()`: mantiene la transparencia.
- `enemy_imgs = [pygame.transform.scale(pygame.image.load(...'enemy1.png')..., ENEMY_SIZE), pygame.transform.scale(pygame.image.load(...'enemy2.png')..., ENEMY_SIZE), pygame.transform.scale(pygame.image.load(...'enemy3.png')..., ENEMY_SIZE),]`: Lista con imágenes de enemigos, cada una redimensionada.
- `bullet_img = pygame.transform.scale(pygame.image.load(os.path.join(img_path, 'bullet.png')).convert_alpha(), BULLET_SIZE)`
`bullet_img = pygame.transform.rotate(bullet_img, -270)`: Carga la imagen de la bala, la redimensiona y la rota para apuntar hacia arriba.

```

        laser_sound.play()

# Crear enemigos cada cierto tiempo
enemy_spawn_timer += 1
if enemy_spawn_timer >= 60: # Cada 1 segundo (60 FPS)
    spawn_enemy()
    enemy_spawn_timer = 0

# Actualizar sprites
all_sprites.update()

# Colisiones balas-enemigos
hits = pygame.sprite.groupcollide(enemies, bullets, True, True)
for hit in hits:
    if explosion_sound:
        explosion_sound.play()
    player.score += 100

# Colisiones jugador-enemigos
if pygame.sprite.spritecollide(player, enemies, True):
    player.lives -= 1
    if player.lives <= 0:
        print(";Game Over!")
        running = False

# Dibujar todo
screen.blit(background, (0, 0))
all_sprites.draw(screen)

font = pygame.font.Font(None, 36)
text = font.render(f"Score: {player.score} Lives: {player.lives}", True, (255, 255, 255))
screen.blit(text, (10, 10))

pygame.display.flip()

pygame.quit()

```

- `enemy_spawn_timer += 1`
`if enemy_spawn_timer >= 60:`
`spawn_enemy()`
`enemy_spawn_timer = 0:` Cada 60 frames (1 segundo si estás a 60 FPS), genera un nuevo enemigo.
- `all_sprites.update()`: Llama al método `update()` de todos los sprites registrados para que cambien su estado (por ejemplo, moverse).
- `hits = pygame.sprite.groupcollide(enemies, bullets, True, True)`
`for hit in hits:`
`explosion_sound.play()`
`player.score += 100:`
 - Detecta colisiones entre enemigos y balas.

- Si hay impacto, reproduce sonido y suma puntos.
- Elimina tanto el enemigo como la bala (True, True).

```
- if pygame.sprite.spritecollide(player, enemies, True):
    player.lives -= 1
    if player.lives <= 0:
        print("¡Game Over!")
        running = False:
```

- Si un enemigo toca al jugador, pierde una vida.
- Si se queda sin vidas, el juego termina (running = False).

```
- blit: dibuja la imagen de fondo.
```

```
- draw: dibuja todos los sprites.
```

```
- font = pygame.font.Font(None, 36)
text = font.render(f"Score: {player.score} Lives: {player.lives}", True,
(255, 255, 255))
screen.blit(text, (10, 10)):
```

- Crea una fuente (None = por defecto).
- render: genera una imagen del texto.
- (255, 255, 255): color blanco.
- blit(text, (10, 10)): dibuja el texto en la esquina superior izquierda.

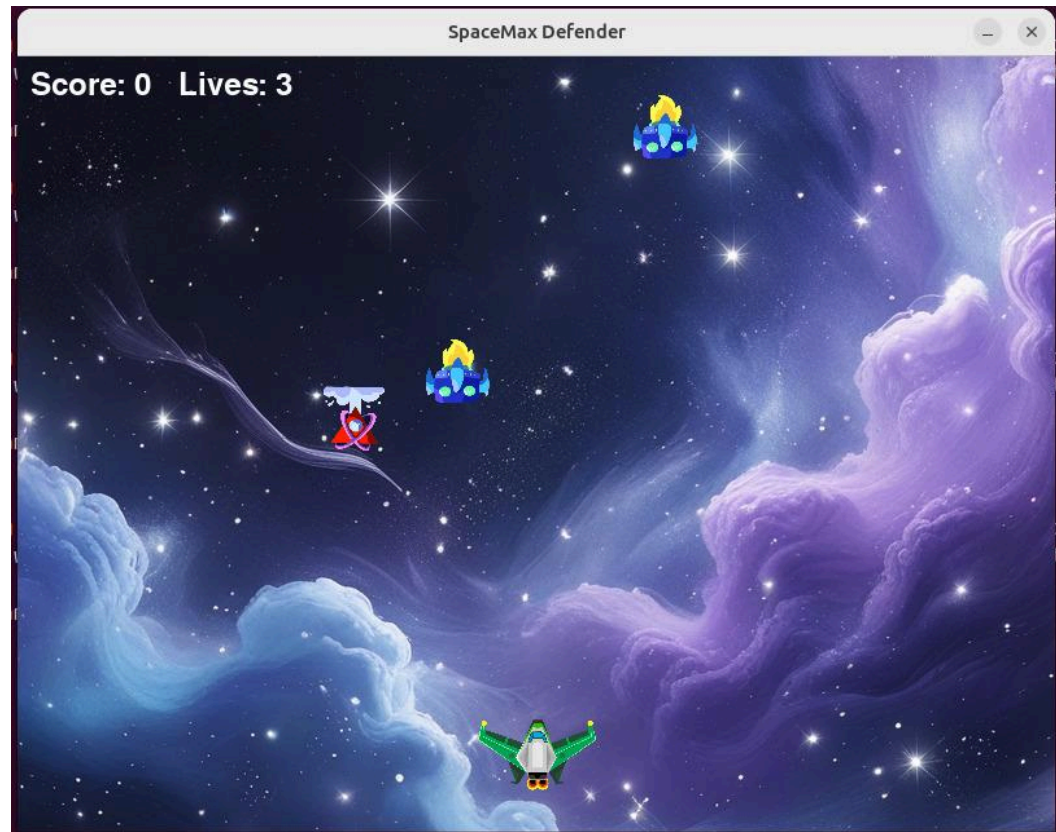
```
- pygame.display.flip(): Actualiza toda la pantalla con los nuevos
elementos dibujados.
```

```
- pygame.quit(): Cierra el juego y libera los recursos usados por
Pygame.
```

2. Se ejecuta el archivo del videojuego (**juego.py**) utilizando Python 3 desde la terminal y se visualiza el juego.

```
brayan@brayan-laptop:~$ python3 juego.py
<frozen importlib._bootstrap>:488: RuntimeWarning: Your system is avx2 capable but pygame was not built with support for it. The performance of some of your blits could be adversely affected. Consider enabling compile time detection with environment variables like PYGAME_DETECT_AVX2=1 if you are compiling without cross compilation.
pygame 2.5.2 (SDL 2.30.6, Python 3.12.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

- `ppython3 juego.py`: Ejecuta el archivo `juego.py` usando Python 3.



- **Actividad 3**

- **ROS**

1. Se accede al directorio del proyecto llamado `ros_turtlesim` desde la terminal. El objetivo es verificar que dicho directorio contenga los archivos necesarios para construir y ejecutar un contenedor Docker.

```
brayan@brayan-laptop:~$ cd ros_turtlesim
brayan@brayan-laptop:~/ros_turtlesim$ ls
docker-compose.yml  Dockerfile  entrypoint.sh
```

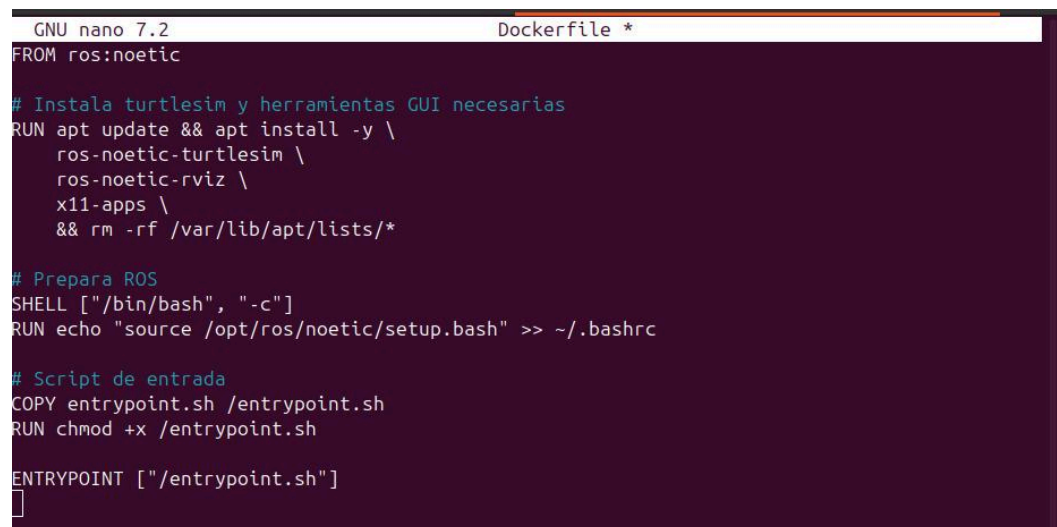
- **cd ros_turtlesim**: Cambia al directorio ros_turtlesim desde la ubicación actual.
- **ls**: Lista el contenido del directorio actual (ros_turtlesim).

2. Se accede al archivo Dockerfile mediante el nano.

```
brayan@brayan-laptop:~/ros_turtlesim$ nano Dockerfile
```

- **nano Dockerfile**: Abre el archivo Dockerfile usando el editor de texto nano.

3. Se modifica el nano para construir una imagen Docker personalizada con ROS Noetic y el simulador turtlesim.



```
GNU nano 7.2 Dockerfile *
FROM ros:noetic

# Instala turtlesim y herramientas GUI necesarias
RUN apt update && apt install -y \
    ros-noetic-turtlesim \
    ros-noetic-rviz \
    x11-apps \
    && rm -rf /var/lib/apt/lists/*

# Prepara ROS
SHELL ["/bin/bash", "-c"]
RUN echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

# Script de entrada
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

- **FROM ros:noetic**: Define la imagen base desde la cual se construirá esta nueva imagen. En este caso, se utiliza una imagen oficial de ROS Noetic.
- **RUN apt update && apt install -y ...**: Ejecuta la actualización de los paquetes del sistema e instala:
 - **ros-noetic-turtlesim**: simulador visual para ROS.
 - **ros-noetic-rviz**: herramienta de visualización 3D.

- **x11-apps**: utilidades necesarias para mostrar interfaces gráficas.
- **rm -rf /var/lib/apt/lists/**: Limpia los archivos temporales de instalación para reducir el tamaño de la imagen Docker.
- **SHELL ["/bin/bash", "-c"]**: Establece **bash** como intérprete de comandos para los comandos posteriores.
- **RUN echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc**: Agrega al archivo de configuración **.bashrc** el comando para cargar el entorno ROS automáticamente cada vez que se inicie un shell interactivo.
- **COPY entrypoint.sh /entrypoint.sh**: Copia el archivo **entrypoint.sh** desde el directorio local al sistema de archivos del contenedor.
- **RUN chmod +x /entrypoint.sh**: Da permisos de ejecución al script.
- **ENTRYPOINT**: Define el script que se ejecutará automáticamente cuando se inicie un contenedor basado en esta imagen.

4. Se abre el archivo `entrypoint.sh` mediante el editor de texto `nano`.

```
brayan@brayan-laptop:~/ros_turtlesim$ nano entrypoint.sh
```

5. Se modifica el `nano` con el código que permite automatizar el arranque del entorno ROS y la ejecución del nodo simulado.

```
GNU nano 7.2          entrypoint.sh *
#!/bin/bash
source /opt/ros/noetic/setup.bash
exec "$@"
```

- `#!/bin/bash`: Es una "shebang". Le indica al sistema que el archivo debe ser ejecutado usando Bash.
 - `source /opt/ros/noetic/setup.bash`: Carga el entorno de ROS Noetic en el shell actual, estableciendo las variables de entorno necesarias.
 - `exec "$@"`: Ejecuta cualquier comando que se pase como argumento al contenedor, reemplazando el proceso actual.
6. Se procede a la edición y configuración del archivo `docker-compose.yml`, el cual permite definir y manejar fácilmente los servicios necesarios para ejecutar el contenedor ROS con soporte gráfico.

```
brayan@brayan-laptop:~/ros_turtlesim$ nano docker-compose.yml
```

7. Se agregó el código en el nano.


```
GNU nano 7.2                                docker-compose.yml
version: "3"
services:
  roscore:
    build: .
    container_name: roscore
    network_mode: host
    environment:
      - DISPLAY=${DISPLAY}
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix
    command: bash -c "roscore"

  turtlesim:
    build: .
    container_name: turtlesim
    network_mode: host
    environment:
      - DISPLAY=${DISPLAY}
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix
    depends_on:
      - roscore
    command: bash -c "roslaunch turtlesim turtlesim_node"

  teleop:
    build: .
    container_name: teleop
    network_mode: host
    stdin_open: true
    tty: true
    depends_on:
      - roscore
    command: bash -c "roslaunch turtlesim turtle_teleop_key"
```

- **version: "3"**: Define la versión del esquema de Docker Compose utilizada.
- **roscore**: Contenedor base que inicia el servidor maestro de ROS.
- **turtlesim**: Servicio que ejecuta el nodo gráfico turtlesim_node, encargado de mostrar la tortuga en pantalla.
- **teleop**: Servicio que permite controlar la tortuga desde el teclado mediante el nodo turtle_teleop_key.
- **stdin_open: true**: Se habilita la entrada estándar

8. Se procedió a la construcción de las imágenes Docker

```
brayan@brayan-laptop:~/ros_turtlesim$ xhost +
access control disabled, clients can connect from any host
brayan@brayan-laptop:~/ros_turtlesim$ docker compose build
WARN[0000] /home/brayan/ros_turtlesim/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 2/5 (24/26)
=> [roscore internal] load build definition from Dockerfile 0.0s
=> == transferring dockerfile: 453B 0.0s
=> [teleop internal] load metadata for docker.io/library/ros:noetic 2.2s
=> [roscore auth] library/ros:pull token for registry-1.docker.io 0.0s
=> [roscore internal] load .dockerignore 0.0s
=> == transferring context: 2B 0.0s
=> [turtlesim 1/5] FROM docker.io/library/ros:noetic@sha256:6465a56f03f7 0.0s
=> [roscore internal] load build context 0.0s
=> == transferring context: 96B 0.0s
=> CACHED [teleop 2/5] RUN apt update && apt install -y ros-noetic-t 0.0s
=> CACHED [teleop 3/5] RUN echo "source /opt/ros/noetic/setup.bash" >> - 0.0s
=> CACHED [roscore 4/5] COPY entrypoint.sh /entrypoint.sh 0.0s
=> CACHED [roscore 5/5] RUN chmod +x /entrypoint.sh 0.0s
=> [roscore] exporting to image 0.0s
=> == exporting layers 0.0s
=> == writing image sha256:3d9f1ad88065266c37928ab2311f45eb0e0740fff23b5 0.0s
=> == naming to docker.io/library/ros_turtlesim-roscore 0.0s
=> [roscore] resolving provenance for metadata file 0.0s
=> [teleop internal] load build definition from Dockerfile 0.0s
=> == transferring dockerfile: 453B 0.0s
=> [turtlesim internal] load build definition from Dockerfile 0.0s
=> == transferring dockerfile: 453B 0.0s
=> [teleop internal] load .dockerignore 0.0s
=> == transferring context: 2B 0.0s
=> [turtlesim internal] load .dockerignore 0.0s
```

```
[+] Building 3/3
✔ roscore Built 0.0s
✔ teleop Built 0.0s
✔ turtlesim Built 0.0s
```

- **xhost +:** Habilita el acceso a la interfaz gráfica (X11) desde cualquier host.
- **docker compose build:** Construye las imágenes de Docker definidas en el archivo docker-compose.yml.

9. Se levantaron los contenedores Docker para los servicios roscore y turtlesim.

```
brayan@brayan-laptop:~/ros_turtlesim$ docker compose up roscore turtlesim
WARN[0000] /home/brayan/ros_turtlesim/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
✔ Container roscore Running 0.0s
✔ Container turtlesim Created 0.0s
Attaching to roscore, turtlesim
turtlesim | QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
turtlesim | libGL error: MESA-LOADER: failed to retrieve device information
turtlesim | libGL error: MESA-LOADER: failed to open andgpu: /usr/lib/dri/andgpu_dri.so: cannot open shared object file: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:$S(ORIGIN)/dri:/usr/lib/dri, suffix_dri)
turtlesim | libGL error: failed to load driver: andgpu
turtlesim | libGL error: failed to open /dev/dri/cardi: No such file or directory
turtlesim | [WARN] [1748265438.893384387]: Oh no! I hit the wall! (Clamping from [x=3.672377, y=-0.007619])
turtlesim | [WARN] [1748265438.893384387]: Oh no! I hit the wall! (Clamping from [x=3.702047, y=-0.011988])
turtlesim | [WARN] [1748265438.999722552]: Oh no! I hit the wall! (Clamping from [x=3.731716, y=-0.011988])
turtlesim | [WARN] [1748265438.92311559]: Oh no! I hit the wall! (Clamping from [x=3.761386, y=-0.011988])
turtlesim | [WARN] [1748265438.941627343]: Oh no! I hit the wall! (Clamping from [x=3.791055, y=-0.011988])
turtlesim | [WARN] [1748265438.956973390]: Oh no! I hit the wall! (Clamping from [x=3.820725, y=-0.011988])
turtlesim | [WARN] [1748265438.973253023]: Oh no! I hit the wall! (Clamping from [x=3.850394, y=-0.011988])
turtlesim | [WARN] [1748265438.989580246]: Oh no! I hit the wall! (Clamping from [x=3.880064, y=-0.011988])
turtlesim | [WARN] [1748265439.005892420]: Oh no! I hit the wall! (Clamping from [x=3.909734, y=-0.011988])
turtlesim | [WARN] [1748265439.021931270]: Oh no! I hit the wall! (Clamping from [x=3.939403, y=-0.011988])
turtlesim | [WARN] [1748265439.037206422]: Oh no! I hit the wall! (Clamping from [x=3.969073, y=-0.011988])
turtlesim | [WARN] [1748265439.053576647]: Oh no! I hit the wall! (Clamping from [x=3.998742, y=-0.011988])
```

- **docker compose up roscore turtlesim:** Ejecuta los servicios roscore y turtlesim definidos en docker-compose.yml.
- **roscore:** Lanza el núcleo de ROS (gestor de nodos, parámetros, etc.).

- **turtlesim**: Inicia el nodo gráfico que simula la tortuga en ROS.

10. Se verificó que el entorno del proyecto contiene los archivos esenciales para la ejecución en contenedores, como docker-compose.yml, Dockerfile y entrypoint.sh.

```
brayan@brayan-laptop:~/ros_turtlesim$ ls
docker-compose.yml  Dockerfile  entrypoint.sh
```

- **ls**: Lista el contenido del directorio actual. Muestra archivos y carpetas.

11. se crea una imagen personalizada de Docker para el entorno ros_turtlesim a partir de un archivo de configuración ubicado en el directorio actual. La imagen generada incluye todos los componentes necesarios para ejecutar ROS Noetic, además de un script que define cómo iniciar el entorno dentro del contenedor.

```
brayan@brayan-laptop:~/ros_turtlesim$ docker build -t lordbasto/ros_turtlesim .
[+] Building 1.8s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 453B                                0.0s
=> [internal] load metadata for docker.io/library/ros:noetic      1.8s
=> [auth] library/ros:pull token for registry-1.docker.io         0.0s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/ros:noetic@sha256:6465a56f03f72033905bd7 0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 34B                                       0.0s
=> CACHED [2/5] RUN apt update && apt install -y ros-noetic-turtlesim 0.0s
=> CACHED [3/5] RUN echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc 0.0s
=> CACHED [4/5] COPY entrypoint.sh /entrypoint.sh                 0.0s
```

- **docker build**: Comando para construir una imagen Docker a partir de un Dockerfile.
- **-t lordbasto/ros_turtlesim**: Etiqueta la imagen resultante con el nombre lordbasto/ros_turtlesim.

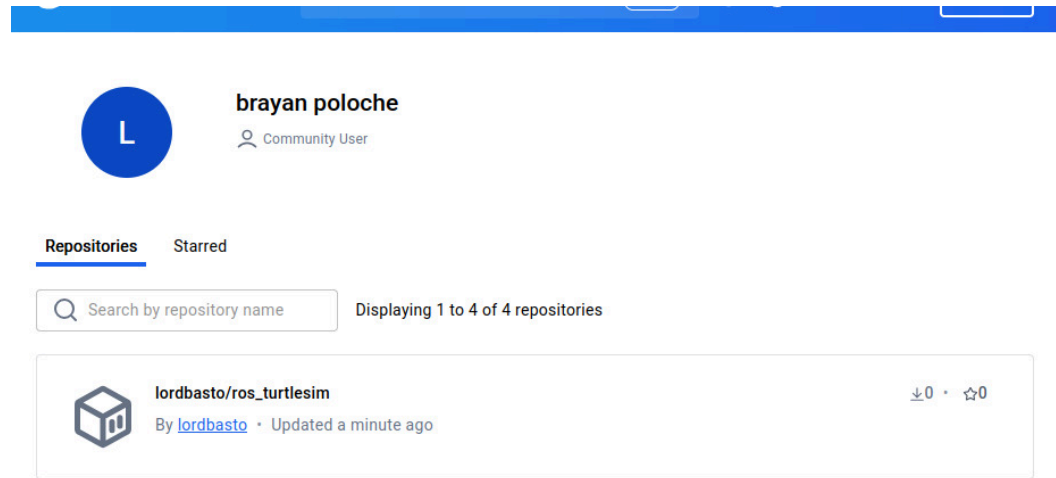
12. Se publica la imagen Docker personalizada previamente construida en el repositorio en línea de Docker Hub. Esto permite almacenar la imagen en la nube y facilita su descarga y uso en otros sistemas o entornos que necesiten ejecutar el entorno ros_turtlesim.

```

brayan@brayan-laptop:~/ros_turtlesim$ docker push lordbasto/ros_turtlesim
Using default tag: latest
The push refers to repository [docker.io/lordbasto/ros_turtlesim]
4ce69375b70d: Pushed
b97b2fba7948: Pushed
2222cd7b2b13: Pushed
781e54328690: Pushed
160021d04398: Mounted from osrf/ros
54e2ce558285: Mounted from osrf/ros
afdddf35f23: Mounted from osrf/ros
8ef2d16cd8ff: Mounted from osrf/ros
19a445f5186d: Mounted from osrf/ros
83f6fd4e23eb: Mounted from osrf/ros
b5d65e98a167: Mounted from osrf/ros
1129f49c3ee7: Mounted from osrf/ros
560e997ec295: Mounted from osrf/ros
470b66ea5123: Mounted from library/ubuntu
latest: digest: sha256:bcd8c3ec6e6ef4fcd84f3bca8eeadd26b062ecf9e05f27db1dfc71683
d1e710e size: 3253

```

- **docker**: Es la herramienta principal para gestionar contenedores e imágenes en Docker.
 - **push**: Es el subcomando que permite subir ("empujar") una imagen Docker local a un repositorio remoto como Docker Hub.
 - **lordbasto/ros_turtlesim**: Es el nombre completo de la imagen que se va a subir, donde:
 - **lordbasto** es el nombre del usuario o la organización en Docker Hub.
 - **ros_turtlesim** es el nombre de la imagen dentro del repositorio de ese usuario.
13. Se confirma visualmente que la imagen Docker personalizada `ros_turtlesim` fue correctamente subida al repositorio del usuario `lordbasto` en Docker Hub.



14. Se accedió al directorio del proyecto `ros_turtlesim` y se ejecutó el comando para permitir al usuario mover la tortuga con las teclas de flecha (`← ↑ ↓ →`) y salir con la tecla `q`.

```
no configuration file provided: not found
brayan@brayan-laptop:~$ cd ~/ros_turtlesim
brayan@brayan-laptop:~/ros_turtlesim$ docker compose run --rm teleop
WARN[0000] /home/brayan/ros_turtlesim/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Creating 1/1
✓ Container roscore Running 0.0s
Reading from keyboard
-----
Use arrow keys to move the turtle. 'q' to quit.
```

- `cd ~/ros_turtlesim`: Cambia el directorio actual al directorio `ros_turtlesim` ubicado en el home del usuario.
- `docker compose run --rm teleop`: Ejecuta el servicio `teleop` definido en el `docker-compose.yml`.
 - `run`: Ejecuta un servicio de manera interactiva, como si fuera un contenedor independiente.
 - `--rm`: Elimina automáticamente el contenedor después de que el proceso termina (no lo guarda).
 - `teleop`: Nombre del servicio que permite controlar la tortuga con el teclado

15. Se visualiza el juego.

