UNIVERSIDAD NACIONAL DE INGENIERIA

“FACULTAD DE INGENIERIA ELÉCTRICA Y ELECTRÓNICA”

“PANAMERICANA CRASH COURSE”

Integrantes:

• Escajadillo Gaspar José Fernando L4 (20231351H)

• Lopez Chauca Aaron Rubén L3 (20230108B)

Profesor: Tello Canchapoma Yuri Oscar

Lima, 2024

Índice

Introducción:

En " **PANAMERICANA CRASH COURSE** ", el jugador se embarca en un emocionante viaje a lo largo de una de las carreteras más icónicas de Perú: la **Panamericana Norte**. El objetivo principal es avanzar por la carretera mientras evitas coches que aparecen a medida que aumentas de velocidad. La experiencia del juego simula el frenético tráfico de la Panamericana, con sus característicos autos y camiones que viajan a gran velocidad. El jugador controla un vehículo que se desplaza por un carril central, pudiendo moverse a la izquierda o derecha para evitar choques.

Objetivos:

Desarrollar un juego funcional en Python: El objetivo principal es crear un juego que sea completamente jugable, con todas las mecánicas y funcionalidades implementadas de manera efectiva. Esto incluye el desarrollo de la lógica de juego, el control del personaje y la interacción con el entorno. El uso de Python permitirá aprovechar su simplicidad y flexibilidad, al mismo tiempo que se implementan técnicas de programación avanzadas, como la programación orientada a objetos.

Implementar mecánicas de juego interactivas: Que permitan al jugador tener una experiencia entretenida y dinámica. La interacción debe proporcionar una experiencia fluida y atractiva, manteniendo el interés del jugador durante toda la partida.

Utilizar bibliotecas especializadas para gráficos y eventos: Se hará uso de bibliotecas especializadas, como Pygame, que facilitan la creación de videojuegos en Python. Estas bibliotecas permiten la gestión eficiente de gráficos 2D, animaciones, sonidos y eventos (teclado, ratón, controladores). El objetivo es aprovechar estas herramientas para crear una buena experiencia de juego que tenga buen contenido visual y auditivo, sin sacrificar el rendimiento ni la simplicidad del código.

Crear una interfaz gráfica intuitiva y amigable: Un buen diseño de interfaz es fundamental para que el jugador se sienta cómodo. La interfaz debe ser clara, fácil de entender y permitir que el jugador navegue por los menús.

Proyectos similares subidos a GIT-HUB:

Juegos básicos de Python: FirstGame:

Este juego trata de un sencillo **juego de carreras 2D**, diseñado en Python, en el que controlas un automóvil que debe esquivar bloques que caen desde la parte superior de la pantalla. A medida que avanzas, la velocidad de los bloques aumenta, y el objetivo es esquivar tantos bloques como sea posible antes de que ocurra una colisión.

Donde el creador dice que es un juego de esquivar obstáculos con coches escrito a partir de un tutorial de YouTube.

https://github.com/TheSahilGit/Basic-Python-Games-FirstGame

Dodge It!:

El presente juego **"Dodge It!"**, en el que controlas un coche que debe esquivar obstáculos (bloques negros) mientras avanzas en el juego. El objetivo principal es evitar chocar con los bloques que caen desde la parte superior de la pantalla. A medida que esquivas más bloques, la velocidad de los obstáculos aumenta, lo que incrementa la dificultad del juego.

- Tiene una interfaz que en su menú principal sale: **Play**: Comienza el juego y **Quit**: Cierra el juego
- Sus elementos principales: El movimiento del auto (las teclas de flecha izquierda y derecha), Obstáculos (bloques) y Pausa del Juego (pulsa la tecla P)
- Sonidos y Efectos: Sonido de choque (car_crash.wav) y Música de fondo (game_music.wav)

https://github.com/apaar97/dodge-it

F1 GAME:

juego de carreras de autos en Turtle llamado "F1 GAME", en el que dos jugadores (jugador A y jugador B) compiten moviendo sus coches en una pista. El objetivo es maniobrar los coches sin chocar con los obstáculos, y el juego es controlado con teclas del teclado para ambos jugadores. Los obstáculos cambian de posición a medida que los jugadores avanzan en la pista, lo que añade dificultad al juego.

https://github.com/giacomotampella/f1_game_python

Car Game:

Este código implementa un juego de conducción en 2D utilizando la biblioteca *Pygame*, donde el jugador controla un coche que debe evitar obstáculos en forma de conos de tráfico. La pantalla tiene un fondo negro con líneas de carril que simulan una carretera. El coche del jugador puede moverse a la izquierda y derecha, mientras los conos de tráfico descienden de manera aleatoria desde la parte superior de la pantalla. El objetivo es esquivar estos conos para evitar colisiones, acumulando puntos a medida que se avanza en el juego. Los obstáculos se generan dinámicamente y aumentan la dificultad del juego conforme el jugador sobrevive más tiempo.

Este proyecto es similar al que estoy desarrollando, ya que ambos tratan de evitar obstáculos en una carretera, aunque este se enfoca más en un estilo arcade simplificado.

https://github.com/KrishnaAkshay7/python-car-game

Librerías:

Durante el trayecto de realización de este proyecto hicimos uso de ciertas librerías, las cuales se encargan de realizar determinadas funciones que mostraremos ahora:

1. Pygame:
   Es una librería diseñada para realizar juegos en Python, proporciona herramientas para poder hacer uso de gráficos, audio y entrada de usuario con el uso del teclado.

2. Sys:
   Proporciona funciones y variables que interactúan con el sistema operativo como lo puede ser el finalizar programas y acceder a la línea de comandos.

3. Random:
   Se usa para generar números aleatorios y asignar variables aleatorias en determinadas partes del juego.

4. Numpy:
   Se usa para trabajar con matrices o arreglos (Arrays) y en conjunto con el Q table, permite realizar cálculos números de forma eficiente.

5. Keyboard:
   Se usa para detectar cuando se presiona una determinada tecla.

Para las pruebas unitarias hicimos uso de algunas librerías adicionales:

1. Unittest:
   Permite organizar las pruebas en clases, de modo que se pueda analizar de forma individual el correcto funcionamiento de las clases.

2. Pandas:
   Proporciona estructuras de datos más rápidas, permitiendo manejar y manipular grandes cantidades de datos de forma sencilla, en este caso haciendo uso de un cuadro de Excel.

3. Numpy:
   Se mencionó anteriormente y es que se encarga del cálculo numérico en Python.

Tecnologías:

Durante el desarrollo de este proyecto hicimos uso de una tecnología muy importante como lo es el Machine Learning, que es una rama de la inteligencia artificial que permite a una máquina mejorar y aprender a partir de la experiencia, esto se ve de mejor manera al implementar el Aprendizaje Reforzado.

Gracias al Aprendizaje por Refuerzo el sistema aprende a base de prueba y error, mejorando de forma que no se cometan los mismos errores, permitiendo la automatización del programa ejecutado. Esto se favorece a su vez haciendo uso de una Q table, la cual es una tabla que almacena los valores de acciones realizadas por un agente, favoreciendo este aprendizaje automático.

Hiperparámetros:

En el proyecto presente hicimos uso de múltiples hiperparámetros los cuales se encargan de controlar el comportamiento del modelo de aprendizaje, estos parámetros configuran su valor antes de entrenar al modelo. Los hiperparámetros utilizados son los siguientes:

1. Tasa de aprendizaje:
   Su valor en este proyecto es de 0.4 y se encarga de controlar qué tanto influye el nuevo conocimiento sobre el valor de Q, un valor más alto permite aprender más rápido, pero tendría probabilidades de fallar más seguido.

2. Factor descuento:
   Su valor en este proyecto es de 0.97 y se encarga de controlar la anticipación con respecto al entorno, a un valor más cercano a 1 se suele dar más importancia al futuro, mientras que un valor más cercano a 0 hace que se considere la instantaneidad.

3. Episodios:
   El valor en este proyecto es de 500 y es el número de iteraciones en los que el agente entrenará y hace que el agente experimente el proceso de toma de decisiones en base al entorno.

4. Qtable:
   Los valores en este proyecto se presentan como np.random.uniform(low=0, high=2, size=[3, 3, 3]) donde la tabla Q inicia con los valores de un rango de 0 a 2, donde se almacenan las estimaciones del valor Q para el estado y la acción.

Aparte de estos hiperparámetros se usan otros parámetros menores los cuales son la acciones y recompensas por realizar un esquive exitoso, también la velocidad de incremento, la dificultad y el valor de enemigos en pantalla a la vez.

Diseño y solución:

Para el desarrollo de este juego implementamos el siguiente diagrama de clases, mediante el cual podremos ver las funciones y atributos que corresponden al desarrollo del juego, como los coches enemigos, la dificultad, entre otros.

@startuml

class Juego {

- jugador: CarroJugador
- enemigos: list
- pista: Pista
- puntaje: int
- dificultad: int
- imagen_fondo_game_over: pygame.Surface
- imagen_fondo: pygame.Surface
- carril_posiciones: list
- estado: int
- qtable: np.ndarray
- tasa_aprendizaje: float
- factor_descuento: float
- episodios: int
- accion: int
- recompensa: int
- modo: int
- train: bool
- future_q: float
- future_riel: int
- actual_q: float
+ __init__(jugador: CarroJugador, enemigos: list, pista: Pista)
+ iniciarJuego(): void
+ acumularPuntos(puntos: int): void
+ gameOver(): void
+ pantalla_inicio(): void

```
    + seleccion_dificultad(): void
    + dibujar_boton(texto: string, color_normal: Color, color_hover: Color, rect: pygame.Rect): void
}

class CarroJugador {
    - imagen: pygame.Surface
    - x: int
    - y: int
    - velocidad: int
    - direccion: int
    + __init__(imagen: pygame.Surface, x: int, y: int)
    + mover(): void
}

class CarroEnemigo {
    - imagen: pygame.Surface
    - x: int
    - y: int
    - velocidad: int
    + __init__(imagen: pygame.Surface, x: int, y: int)
    + mover(): void
}

class Pista {
    - longitud: int
    - carriles: list
    + __init__(longitud: int, carriles: list)
    + mostrar(): void
}
```

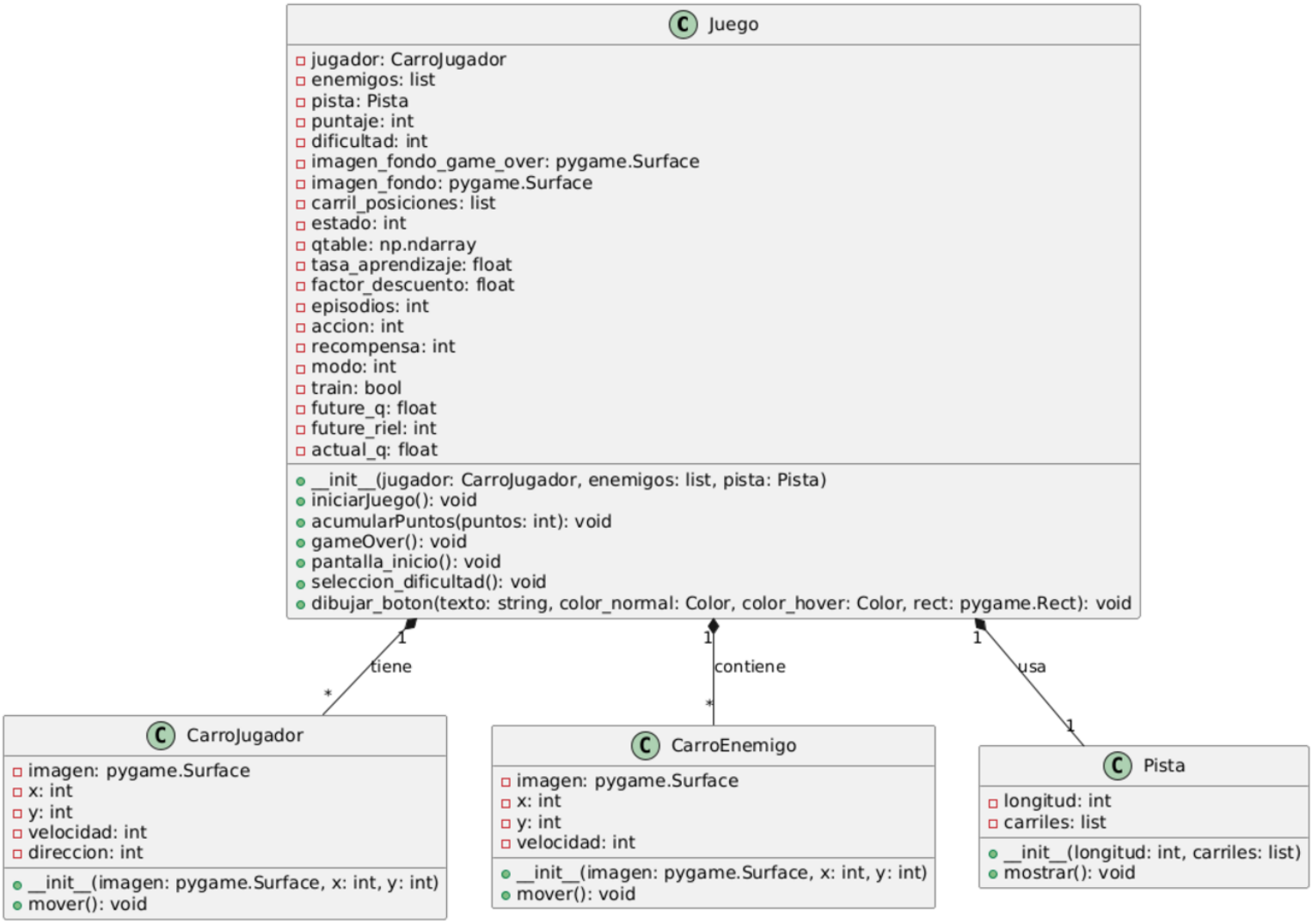
Juego "1" *-- "*" CarroJugador : tiene

Juego "1" *-- "*" CarroEnemigo : contiene

Juego "1" *-- "1" Pista : usa

@enduml

El cual tras ser ejecutado deja con el diagrama presente:



Cronograma:

RESPONSABLE	TAREAS	SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 5
Aaron Lopez	*1.Revisión del modelo*					
José Escajadillo	*2.Diseño de los controles y sistema de puntuación*					
Aaron Lopez	*3.Implementación del sistema de colisiones y efectos de audio*					
José Escajadillo	*4.Desarrollo del sistema de niveles y velocidades*					
Aaron Lopez	*5.Creación del entorno gráfico (escenarios y coches enemigos)*					

RESPONSABLE	TAREAS	SEMANA 6	SEMANA 7	SEMANA 8
José Escajadillo	*6.Automatización*			
Aaron Lopez	*7.Pruebas automatizadas*			
José Escajadillo	*8.Publicación y resultados finales*			

Código por clases:

1. **Carro:**
   Es la clase base para las subclases Carro en el juego.
   Atributos: carril, y, velocidad, ancho y alto.

   Método: mover(dirección_x).

   1.1. **CarroEnemigo:**
   Atributos: type, imagen_auto

   Método: mover()

   1.2. **CarroJugador:**
   Atributos: cantidadPuntos, riel_player, dir_back1, dir_back2, imagen_auto, listado_recompensas, max_speed, speed, carril_act, ocupado, modo.

   Métodos: acumularPuntos(puntos, choque), mover(direccion_x, direccion_xaut, carril_posiciones)

2. **Pista:**
   Representa la pista del juego.
   Atributos: carriles.

3. **Juego:**
   Es la lógica principal del juego, controla su flujo, el puntaje, movimientos, colisiones y otras funciones.

   Atributos: jugador, enemigos, pista, puntaje, dificultad, musica_menu, musica_juego, sonido_choque, screen, imagen_fondo, fuente.
   Otros atributos importantes y que comprenden el entrenamiento y prueba de la automatización son:
   qtable, tasa_aprendizaje, factor_descuento, episodios.

   Métodos: iniciarjuego(), gameover()