Introducción a la Inteligencia Artificial
Proyecto sobre aprendizaje reforzado
Prof. Carlos B. Ogando M.

PROYECTO GENETIC DRIFT

HONESTIDAD ACADÉMICA

Como de costumbre, se aplica el código de honor estándar y la política de probidad académica. Las presentaciones isomórficas a (1) las que existen en cualquier lugar en línea, (2) las enviadas por sus compañeros de clase, o (3) las enviadas por los estudiantes en semestres anteriores serán consideradas plagio.

INSTRUCCIONES

En este proyecto entrenarán un agente inteligente que por aprendizaje reforzado pueda ganar un juego de drifting tipo arcade + endless. Para ello crearán una serie de bots controlados por una red neuronal con optimización por algoritmos genéticos.

I. Introducción

II. Diseño del agente

III. Qué necesita enviar

IV. Recomendaciones de abordaje

I. Introducción

1.1 Endless runners

Un endless runner, corredor sin fin o corredor infinito es un subgénero del videojuego de plataformas en el que el personaje del jugador corre o se mueve por un campo o espacio infinito durante un tiempo indeterminado mientras evita obstáculos. El objetivo del jugador es alcanzar una puntuación alta sobreviviendo el mayor tiempo posible. El método por el cual el nivel del juego o el entorno aparecen continuamente ante el jugador es un ejemplo de generación procedimental. El género explotó en las plataformas móviles tras el éxito de Temple Run, con Jetpack Joyride, Subway Surfers y Canabalt sirviendo como otros ejemplos populares. Su popularidad se atribuye a su modo de juego simple que funciona bien en dispositivos con pantalla táctil.

Para este proyecto estarán realizando un juego similar a estos del género endless runner, pero inspirado en el estilo de drifters arcade.

1.2 Drifters arcade

Un drifter arcade (a veces llamado arcade drifting game) es un tipo de juego de conducción simplificado en el que el objetivo principal no es la simulación realista, sino sentir la emoción del derrape (“drift”) con controles muy intuitivos y una jugabilidad fluida y estilizada.

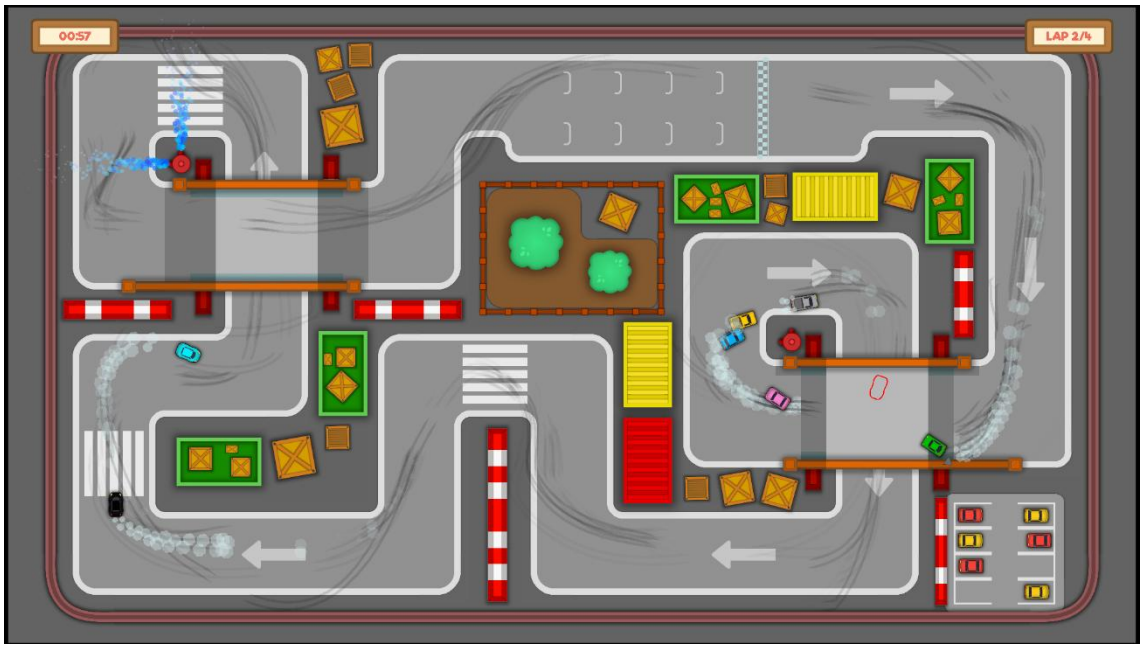
Algunas características de estos juegos son:

- El carro se mueve automáticamente.
- Solo controlas cuándo girar (tap = curva).
- Las curvas están diseñadas para que mantengas un flujo continuo.
- El reto es mantener el drift sin chocar o salirte.

Algunos ejemplos de estos juegos son:

- Super Sprint
- Super Offroad
- Total Arcade Racing

Pueden probar uno en el siguiente enlace:
https://sodarocketstudios.itch.io/endless-drift



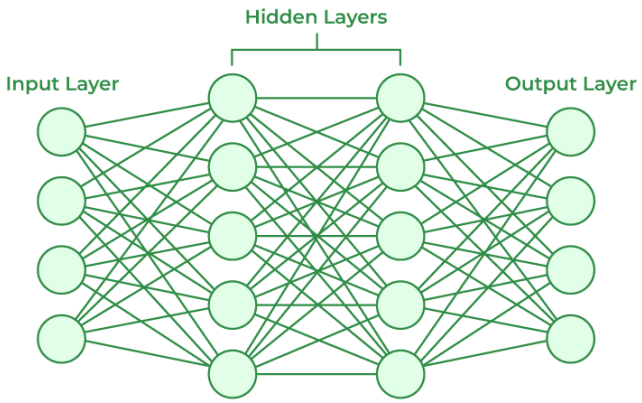
Plantear el agente inteligente desde sus sensores y actuadores es fundamental para poder implementar el mismo. Los actuadores representan las acciones del juego y los sensores las observaciones del entorno.

II. Diseño del agente

El objetivo de este proyecto es crear un agente inteligente que pueda ganar el juego tipo Drifter. Para lograr esto, utilizaremos una combinación de técnicas avanzadas de inteligencia artificial: aprendizaje por refuerzo (RL) y redes neuronales optimizadas mediante algoritmos genéticos.

2.1. Componentes del diseño

1. **Sensores del juego (entradas de la red neuronal)**: el agente debe ser capaz de percibir su entorno en el juego para tomar decisiones informadas. Para ello, se utilizarán varios sensores que indicarán el estado actual del jugador. Algunos ejemplos de qué puede medirse son:
    - **Distancia al obstáculo:** La distancia desde el jugador cualquier obstáculo en diferentes direcciones.
    - **Distancia a un booster:** La distancia desde el jugador hasta el próximo booster.
    - **Velocidad actual:** La velocidad a la que se está moviendo el jugador.
2. **Actuadores (salidas de la red neuronal)**: el agente actuará en su entorno a través de la única acción posible en el juego que es hacer click.
3. **Red Neuronal (cerebro del agente)** La red neuronal será la encargada de procesar la información recibida de los sensores y determinar la acción a tomar. La estructura de la red neuronal incluirá:
    - **Capa de entrada:** Una neurona para cada sensor del juego.
    - **Capas ocultas:** varias capas intermedias que procesarán la información de manera no lineal.
    - **Capa de salida:** Dos neuronas que darán la salida indicando si no hacer nada, doblar a la izquierda o doblar a la derecha.



4. **Aprendizaje por refuerzo (RL):** El agente aprenderá a jugar Jetpack Joyride mediante el aprendizaje por refuerzo implícito en los algoritmos genéticos. Este enfoque se basa en la idea de que el agente recibirá una recompensa o penalización según su desempeño en el juego (ser seleccionado o no). El objetivo del agente es maximizar la recompensa acumulada a lo largo del tiempo. Las principales características del aprendizaje por refuerzo son:
    - **Estado:** La información recibida de los sensores en un momento dado.
    - **Acción:** La decisión de saltar o no, basada en la salida de la red neuronal.
    - **Recompensa:** Una puntuación que refleja el desempeño del agente (por ejemplo, +1 por cada moneda obtenida, -1 por colisionar).
5. **Optimización con algoritmos genéticos** Para mejorar el desempeño de la red neuronal, utilizaremos algoritmos genéticos, los cuales imitan el proceso de evolución natural. Este método permite optimizar los pesos de la red neuronal a través de:
    - **Selección:** Los mejores agentes (redes neuronales con mayor desempeño) son seleccionados para reproducirse.
    - **Cruce:** Combinación de las características de dos agentes seleccionados para crear nuevos agentes.
    - **Mutación:** Introducción de pequeñas modificaciones aleatorias en algunos agentes para mantener la diversidad genética y explorar nuevas soluciones.

2.2. Proceso de entrenamiento

1. **Inicialización:** Crear una población inicial de agentes con redes neuronales con pesos aleatorios.
2. **Simulación:** Permitir que cada agente juegue múltiples partidas de Jetpack Joyride, registrando su desempeño.
3. **Evaluación:** Calcular la recompensa acumulada de cada agente y seleccionar los mejores.
4. **Reproducción:** Aplicar los algoritmos genéticos (selección, cruce y mutación) para crear una nueva generación de agentes.
5. **Repetición:** Repetir el proceso de simulación, evaluación y reproducción durante muchas generaciones hasta que el agente muestre un desempeño óptimo.
6. **Validación:** Validar el desempeño obtenido en niveles nunca antes vistos por el agente.

III. Qué necesita enviar

Su trabajo en esta tarea es desarrollar un agente inteligente de IA con python que aprenda a jugar un juego tipo Drifter empleando aprendizaje reforzado a través de una red neuronal optimizada por algoritmos genéticos.

3.1. Requerimientos de juego e IA:

1. El juego será de running constante tipo Drifter en vista Top Down y estilo 2D.
2. Los circuitos pueden o no estar bordeados por paredes.
3. Deben haber dos tipos de terreno: asfalto (máxima velocidad) y offroad (jugador pierde velocidad)
4. Debe haber un obstáculo tipo "roca" que "mate" al jugador cuando colisione con el mismo.
5. Debe haber un obstáculo tipo "aceite" que haga que el jugador pierda el control por unos segundos.
6. Debe haber un “booster” que le de una aceleración temporal al jugador cuando pase por encima del mismo.
7. Debe morir el jugador cuando colisione de frente a un bloque sólido.
8. Crearán 3 niveles en donde se pueda ver el funcionamiento de todo lo requerido.
9. El juego debe verse gráfica y sonoramente adecuado.

3.2. Requerimientos del optimizador:

- Debe existir al menos un actuador.
- Deben existir al menos cinco sensores. Ejemplo: ¿está cerca de una pared?, distancia a offroad, distancia a una curva, ¿está en modo aceleración?, etc.
- La red neuronal debe tener 1 capa de entrada con los sensores creados, 1 capa de salida con los outputs deseados, y mínimo 1 hidden layer. (3 capas mínimo en total)
- La optimización debe ser usando algoritmos genéticos. Las funciones de cruce y mutación se deciden de forma arbitraria.
- La función de rendimiento debe ser en base a qué tan lejos en el nivel llega el bot.
- Debe probar al menos con 1 función de cruce.
- Debe probar al menos con 2 funciones de mutación.
- Debe probar al menos con 2 funciones de selección.

3.3. Requerimientos de la validación:

1. Para el entendimiento y el análisis de su desarrollo se deberán crear los siguientes niveles:
    a. 2 niveles de dificultad baja con pocas mecánicas del juego.
    b. 2 niveles de dificultad media con mecánicas intermedias del juego.
    c. 2 niveles de dificultad media-alta con todas las mecánicas del juego.
    d. 1 nivel avanzado que conjugue todo.

e. 1 nivel infinito.

2. Deberán correr el optimizador con los niveles base que crearon a modo de entrenamiento de forma incremental. La idea es: correr el bot en el nivel 1, luego el ganador del nivel 1 usarlo como población inicial del nivel 2 y así sucesivamente.
3. Realizarán pruebas incrementales con los niveles de validación mencionados en el acápite 1, tomando en cuenta que compartan las mecánicas y la dificultad de los niveles de training uno a uno. Verificar el rendimiento del bot. (overffited?)
4. Para la validación final usarán el nivel de testing avanzado que integre todas las mecánicas presentadas en los niveles de training en un setup diferente (que no haya sido usado para "entrenar" el bot).
5. Realizarán un benchmark cualitativo del agente inteligente en su proceso de experimentación, describiendo el rendimiento de este en los diferentes escenarios y hallazgos importantes.
6. Realizarán un benchmark cuantitativo en donde compararán:
    a. Métricas:
        i. Velocidad de convergencia de w. (tiempo de convergencia)
        ii. Número de generaciones de convergencia.
        iii. Porcentaje de completitud de niveles de validación
        iv. Tasa de éxito en niveles de validación.
    b. Variables para comparar:
        i. Número de pobladores.
        ii. Tasa de mutación (probabilidad).
        iii. Configuración de peso inicial. (aleatorio, pesos iguales, pesos nulos…)
        iv. Tasa de selección.
        v. Técnicas de mutación. (con sus parámetros respectivos) (2 o más)
        vi. Técnicas de selección. (con sus parámetros respectivos) (1 o más)
        vii. Técnicas de cruce. (con sus parámetros respectivos) (1 o más)
        viii. Técnica de thresholding de la la red neuronal para activar las acciones. (1 o más)
    c. Ejemplo de cómo se puede ver:

NIVEL TIPO FÁCIL	Velocidad de convergencia de w	Número de generaciones de convergencia	Porcentaje de completitud nivel val 1	…	Tasa de éxito en validación
Número de pobladores = 10					
Número de pobladores = 20					
…					

Prob de mutación = 0.1					
Prob de mutación = 0.2					

Nota: No limitarse a este esquema. Pueden hacer una tabla nivel vs variable bajo una sola métrica ordenado por dificultad.

Tasa de éxito en validación	Nivel tipo fácil	Nivel tipo medio	Nivel tipo medio-alto	...	Nivel avanzado
Número de pobladores = 10					
Número de pobladores = 20					
...					
Prob de mutación = 0.1					
Prob de mutación = 0.2					

Pueden acompañar el análisis de los algoritmos con gráficas o cualquier otro elemento que consideren importante.

IV. Recomendaciones de abordaje

1. Lo ideal es que el proceso de entrenamiento del agente se realice de forma incremental nivel a nivel. Al iniciar el entrenamiento de un nivel tomar el mejor agente del anterior.
2. Para lograr convergencia más rápido, frisar la optimización de los sensores relacionados a mecánicas no relacionadas con el nivel es muy recomendable.
3. También probar con una red neuronal shallow (solo capa de entrada y de salida) es sumamente recomendable.
4. Importante recordar normalizar las salidas de las neuronas para que las ponderaciones tengan sentido.