Informe Técnico – Pokeminmax!

Objetivos

Elaborar un programa en Python que permite simular un combate estratégico entre dos entrenadores Pokémon.

Implementar el algoritmo minimax con poda alfa-beta para que la IA tome decisiones inteligentes durante el enfrentamiento de los pokemones.

Desarrollo del Proyecto

1. Estructura de Datos (Módulos pokemon.py y trainer.py)

Para representar el estado del combate de manera efectiva, se definieron dos clases fundamentales:

- **Pokemon**: Esta clase representa a cada criatura individual. Almacena atributos cruciales como el nombre, el tipo (elemental), los puntos de salud (HP) actuales y máximos, y una lista de ataques disponibles. Cada ataque es una instancia de la clase Attack, que a su vez contiene su nombre, tipo y poder. Es una representación directa y clara de los Pokémon, permitiendo una fácil manipulación de sus estados durante el combate.

- **Trainer**: Esta clase representa a un entrenador Pokémon, ya sea el jugador humano o la IA. Contiene el nombre del entrenador, una lista de Pokémon bajo su posesión y el active_pokemon, que es el Pokémon que está actualmente en combate. Métodos como choose_attack y has_available_pokemon permiten interactuar con el estado del entrenador y sus Pokémon. La inclusión de switch_pokemon es un buen punto, aunque en esta versión, el cambio no está permitido durante el turno, es una base para futuras expansione

2. Lógica del Combate (Módulo battle.py)

La clase Battle es el corazón de la simulación, orquestando las interacciones entre los entrenadores y sus Pokémon.

- **Inicialización y Turnos**: La clase Battle se inicia con dos objetos Trainer y un indicador de turnos para controlar quién tiene la iniciativa.
- **Tabla de Tipos Simplificada**: Se implementó una type_chart para calcular la efectividad de los ataques. Aunque simplificada para la primera generación, esta tabla maneja la lógica de efectividad, súper efectividad y no efectividad (en este caso, 0.5x, 1x y 2x de daño, sin considerar ineficacia o inmunidad). Es importante señalar que esta tabla se complementa en la GUI (gui.py) para incluir el tipo "Electric" y asegurar que esté completa, lo cual es una consideración práctica dado el alcance del proyecto.
- **Cálculo de Daño (calculate_damage)**: Esta función toma un ataque, el atacante y el defensor para determinar el daño infligido, aplicando los modificadores de tipo. La simplificación de "siempre aciertas" y la omisión de cambios de estado agilizan la simulación y permiten centrarse en la lógica central del algoritmo.
- **Ejecución del Ataque (perform_attack)**: Simplemente aplica el daño calculado al HP del Pokémon defensor, asegurando que el HP no caiga por debajo de cero.
- **Control de Estado (check_fainted, battle_over, get_winner)**: Estas funciones son cruciales para determinar si un Pokémon ha sido debilitado, si el combate ha terminado y quién es el ganador.

3. Inteligencia Artificial con Minimax (Módulo ai.py)

La implementación de la IA es el componente más complejo y definitorio de este proyecto. La clase AI encapsula el algoritmo Minimax con poda Alpha-Beta.

- **Minimax con Poda Alpha-Beta**: El método minimax es la implementación central de este algoritmo. Recorre un árbol de estados de combate simulados hasta una depth predefinida. La poda Alpha-Beta es fundamental para optimizar la búsqueda, eliminando ramas del árbol que no afectarán la decisión final, lo que mejora significativamente el rendimiento, especialmente a mayores profundidades. La lógica maximizing_player y minimizing_player (representada por True y False para la IA y el jugador humano, respectivamente) es crucial para alternar entre maximizar el puntaje de la IA y asumir que el oponente minimizará ese mismo puntaje.
- **Función de Evaluación Heurística (evaluate_battle)**: Esta función es vital para el algoritmo Minimax, ya que asigna un valor numérico a cualquier estado del combate. Mi función de evaluación considera:
    - **PS restantes**: La diferencia total de HP entre los Pokémon de la IA y los del jugador. Un mayor HP en los Pokémon de la IA y un menor HP en los del oponente resultan en un valor más alto.
    - **Pokémon restantes**: Se le da un peso significativo (multiplicado por 50) a la cantidad de Pokémon que cada entrenador tiene disponibles. Esto incentiva a la IA a priorizar debilitar los Pokémon del oponente y mantener los suyos. La

combinación de estos factores permite a la IA tomar decisiones que buscan asegurar una ventaja a largo plazo en el combate.

- **Elección del Mejor Ataque (choose_best_attack)**: Este método inicia la búsqueda Minimax desde el estado actual del combate, iterando sobre todos los ataques posibles del Pokémon activo de la IA. Por cada ataque, simula el movimiento, cambia el turno y luego llama a minimax para evaluar el resultado de esa secuencia. Finalmente, elige el ataque que conduce al mejor eval (valor máximo) desde la perspectiva de la IA.
- **Clonación del Estado del Combate (clone_battle, clone_pokemon)**: Para que el algoritmo Minimax pueda explorar diferentes futuros sin alterar el estado actual del juego, es imprescindible clonar el objeto Battle y todos sus componentes (Trainer y Pokemon). Las funciones clone_battle y clone_pokemon aseguran que las simulaciones se realicen sobre copias independientes, manteniendo la integridad del estado real del juego. Este es un detalle técnico crucial para el correcto funcionamiento de Minimax.

4. Interfaz de Usuario (Módulo gui.py)

La implementación de la GUI utilizando tkinter proporciona una experiencia de usuario interactiva, lo que era un requisito fundamental del proyecto.

- **Selección Inicial de Pokémon**: La interfaz comienza con una pantalla donde el jugador humano puede elegir su Pokémon inicial. Esto personaliza el inicio de cada partida.
- **Actualización Dinámica**: Funciones como update_ui se encargan de mantener la información del combate (HP de los Pokémon, turno actual, etc.) visible y actualizada para el jugador.
- **Manejo de Ataques**: Los ataques del Pokémon activo del jugador se presentan como botones, haciendo la interacción intuitiva. El método on_attack maneja la lógica cuando el jugador elige un ataque, actualiza el estado del combate y luego pasa el turno a la IA.
- **Turno de la IA (ai_turn)**: Una vez que el jugador ha actuado, se programa una pequeña pausa (root.after) antes de que la IA tome su decisión. Esto simula un "pensamiento" de la IA y mejora la fluidez de la interfaz. La IA llama a choose_best_attack y ejecuta su movimiento, actualizando también la interfaz.
- **Notificaciones**: Se utilizan messagebox para notificar al jugador sobre eventos importantes como la debilitación de un Pokémon o el fin del combate.
- **Gráfico de Tipos Completo para GUI**: Es relevante que la función ensure_complete_type_chart se llama en el módulo gui.py para extender la tabla de tipos definida en battle.py. Esto es una solución pragmática para incluir el tipo "Electric" y asegurar que la IA, al clonar batallas, tenga una tabla de tipos completa para sus evaluaciones.

Análisis Final

El pokeminmax permitió realizar una simulación de combate Pokémon, destacando la implementación del algoritmo **Minimax con poda Alpha-Beta** como centro de la Inteligencia Artificial oponente. El planteamiento modular del pokeminmax estructurado en clases como Pokemon, Attack, Trainer, Battle, AI y GUI, facilitó la implementación de la la batalla de los pokemones e integración de la ia oponente para la elección de ataques más óptimos en cada turno.

La clase AI encapsula la lógica del Minimax, empleando una recursión profunda hasta un depth configurable, permitiendo la exploración del árbol de juego. La **poda Alpha-Beta** optimiza significativamente esta exploración al descartar ramas que no pueden influir en la decisión óptima, reduciendo la complejidad computacional de O(bd) a O(bd/2) en el mejor de los casos (donde b es el factor de ramificación y de la profundidad de búsqueda). Esta optimización es crítica para mantener la interactividad de la GUI, especialmente a profundidades de búsqueda mayores.

La **función de evaluación heurística** (evaluate_battle) se define como una combinación lineal de dos métricas clave: la diferencia total de HP entre los equipos de la IA y el oponente, y la diferencia en el número de Pokémon restantes. La asignación de un peso mayor a la segunda métrica (* 50) refleja una estrategia que prioriza la eliminación de Pokémon enemigos sobre la mera conservación de HP, un enfoque común en juegos de combate. La precisión de esta heurística es fundamental para la calidad de las decisiones de la IA, ya que un valor más alto para la IA indica un estado más favorable.

Un aspecto importante es la implementación de la **clonación de objetos** (clone_battle, clone_pokemon); ya quee esa copia del estado del juego en cada nodo del árbol de búsqueda permite guardar la integridad de las evaluaciones del Minimax.Al manejar las copias de forma controlada, nos aseguramos de que cada simulación del combate sea independiente. Esto permite que el algoritmo pruebe diferentes escenarios posibles sin alterar el estado real de la batalla.

En conclusión, este proyecto demuestra que los algoritmos de búsqueda adversarial como Minimax con poda Alfa-Beta pueden aplicarse de forma efectiva en juegos por turnos. El código refleja un buen manejo del diseño orientado a objetos y la IA resultante logra tomar decisiones acertadas dentro de las reglas simplificadas del juego. En el futuro, se podrían probar heurísticas más avanzadas, aumentar la profundidad del análisis (teniendo en cuenta el impacto en el rendimiento) y ajustar la tabla de tipos para que se parezca más a la del juego original de Pokémon..

Fuente De Pokemones Usados Para Enfrentamientos:

https://pokeapi.co/