Reglas del Hackathon 2021-2022: Robot Combat

Celebrado el 1 DE DICIEMBRE DEL 2021

INTRODUCCIÓN

¡Zimatek organiza su primer Hackathon! El principal objetivo es que sea un reto estimulante y disfrutemos en grande con vuestro ingenio para crear robots invencibles. Además, pretendemos promover el trabajo en equipo, las habilidades de análisis y resolución de problemas complejos, y la mejora de las competencias en materia de programación del alumnado de la Facultad de Ciencia y Tecnología (ZTF/FCT) de la UPV/EHU.

El reto planteado consiste en **comprender la interfaz** que hemos desarrollado en Zimatek para reproducir combates de robots virtuales y usar ese conocimiento para **diseñar la lógica que debe seguir el robot para lograr la victoria** en un torneo. Las reglas del concurso se describen a continuación. Recordamos que las bases generales del concurso se encuentran accesibles y que en caso de contradicción las bases prevalecen frente a estas reglas.

Nos gustaría agradecer el apoyo de la ZTF/FCT en la financiación de los premios y el apoyo de infraestructuras.

DESCRIPCIÓN DEL RETO

Por favor, antes de empezar a programar **leed atentamente estas reglas y aseguraos de haber entendido el reto**. La gente de Zimatek estará encantada de ayudaros. Además, es importante **que antes de programar penséis** la idea sobre papel y valoréis la posibilidad de repartir tareas. Recomendamos que empecéis por un planteamiento sencillo antes de complicaros la vida, que ya habrá tiempo.

Descripción del Robot Combat

www.ehu.eus/es/web/zimatek
Tel. 946 015 483
zimatek@ehu.es

En un combate robot se enfrentan dos robots virtuales en una batalla por **recoger monedas y subir de nivel** antes que su oponente.

Con cada subida de nivel el robot obtiene **tres puntos de experiencia** que puede utilizar para mejorar alguno de sus atributos, como la vida, el daño, la armadura… Esto le dará ventajas para seguir coleccionando monedas, que a su vez le permitirán seguir subiendo de nivel. Aunque ten cuidado, coger monedas está muy bien pero si el robot enemigo se pone agresivo tendrás que reaccionar. ¿Lograrás esquivar sus proyectiles o acabar con él antes de que él acabe contigo?

Descripción de la interfaz

El programa está dividido en varias clases que permiten realizar los combates robot.

En el archivo combat.py se encuentra la clase Combat, que es la que se ejecuta para iniciar un combate. Aquí podéis ver cómo se usan los robots y los valores de los atributos con los que se crearán (líneas 227-244). Podéis editarlos durante el desarrollo, pero tened en cuenta que **los robots competirán con los valores que aparecen ahí**. Del mismo modo, **la frecuencia de monedas (cps)**, será la que se use para el campeonato. En la línea 155 para comprobar cómo se usa la función decide que tendréis que programar.

La clase robot.py define todos los atributos del robot y le dedicamos una sección específica.

En el archivo automated_robot.py está la clase AutomatedRobot, de la cuál cualquier robot automático debe *heredar*. La herencia es un concepto en programación orientada a objetos que describe la noción de que una clase se construye a partir de otra, eso facilita crear varios “tipos” sobre una misma base. En este caso cada uno de los grupos vais a crear un robot automático diferente, pero no lo haréis de cero, tomaréis como base esta clase. Más abajo os contamos un ejemplo para que lo veáis.

Los archivos manual_robot.py y robot_hub.py contienen las clases necesarias para poder controlar un robot de forma manual. No es necesario que comprendáis su funcionamiento, pero si tenéis curiosidad nos lo podéis preguntar.

El archivo projectile.py define la clase Projectile y coin.py la clase Coin. Puede ser de ayuda entender los archivos, pero no debéis manejarlas directamente.

La clase Robot

La clase robot.py define la funcionalidad del robot. Es importante que entendáis su funcionamiento general.

ATRIBUTOS RELEVANTES	
Estadísticas que definen las capacidades del robot y se pueden mejorar, comúnmente conocidas como *stats*	
damage	Daño máximo que puede aplicar un proyectil creado por el robot
projectile_initial_speed	Módulo de la velocidad a la que se desplaza un proyectil creado por el robot
projectile_per_second	Cantidad máxima de proyectiles que un robot puede crear
self_speed	Velocidad máxima del robot en el eje vertical y horizontal. Puede adquirir más velocidad al moverse en diagonal
armor	Describe la atenuación del daño que recibe el robot de un proyectil. La atenuación se calcula como armor/(armor+100).
max_health	Vida máxima del robot
health_regen	Describe cuánta vida recupera el robot cada cinco segundos (aunque la vida se recupera cada segundo).
Atributos espaciales	
pos	Vector [x,y] que describe la posición del vértice superior izquierdo del robot.
velocity	Vector [vx,vy] que describe el cambio por segundo de la posición.
width	Anchura del robot (puede ser ligeramente superior a la visual)
height	Altura del robot (puede ser ligeramente superior a la visual)

Otros	
level	Nivel del robot
experience	Cantidad de experiencia acumulada para subir al siguiente nivel
experience_points	Cantidad de puntos de experiencia sin usar
base_<stat>	Cantidad base de una stat
g_<stat>	Cantidad que se usa para calcular el incremento de la stat al mejorarla: g_<stat> * (0.65 + 0.035 * level)
max_self_speed	El tope de la stat self_speed.
max_projectile_initial_speed	El tope de la stat projectile_initial_speed

FUNCIONES RELEVANTES	
que SÍ puedéis usar	
stop(direction)	Para el movimiento del robot en la dirección dada. direction puede ser ‘right’, ‘left’, ‘top’, ‘down’ o ‘all’.
set_velocity(vx,vy)	Establece el vector de velocidad del robot. Si alguno de los ejes supera self_speed se sustituye por + o - self_speed.
upgrade_stat(stat_key)	Solicita mejorarte una stat. stat_key puede ser cualquiera de los nombres de las stats. La función solo ejecutará la mejora si se tienen puntos de experiencia, y no se superan límites máximos.
cast_basic_attack(target_pos)	Crea y devuelve un proyectil a la izquierda o derecha del robot que se moverá hacia la posición indicada. El proyectil no se creará si se supera el límite de ataques por segundo.
flip_x()	Cambia la orientación del robot.
que NO vais a usar	
move(dx,dy,combat)	Ejecuta el desplazamiento del robot y solo se le puede llamar con un objeto combat, por esto no podéis usarla .

get_properties()	Devuelve un diccionario que resume las características del robot.

Descripción del objetivo

Debéis crear un archivo dentro del directorio automated_robots, con el nombre que prefiráis, y que contendrá una clase con el nombre que le queráis poner, pero que debe heredar de AutomatedRobot. Como modelo, podéis ver el archivo de random_robot.py.

La clase que creéis debe tener al menos una función: decide. Esta función tiene una estructura definida en la clase AutomatedRobot. Toma como argumentos un diccionario con toda la información sobre el robot oponente, el conjunto de proyectiles y el conjunto de monedas. La idea es que con esa información la función debe decidir hacia dónde debe moverse el robot (cogiendo monedas y/o esquivando proyectiles), qué estadísticas subirse (si es posible) y si disparar o no y hacia dónde. La función devolverá None o un proyectil.

Para variar la dirección del movimiento se usará la función set_velocity, para subir estadísticas upgrade_stat, y para crear un proyectil cast_basic_attack. Estas son las tres únicas funciones necesarias, aunque hay otras disponibles.

Conjunto de prácticas de obligado cumplimiento

Python es genial, pero no ideal para todos los usos. En este caso, nos vemos obligados a pediros que no uséis ciertas técnicas que en Python no se pueden limitar completamente (es que pediros programar en Java o C++ era too much).

Principalmente, no intentes cambiar los valores de los atributos de tu robot o tu oponente si no es a través de las funciones habilitadas para ello (set_velocity y upgrade_stat). Si intentáis hackear el sistema para curar a vuestro robot por arte de magia o teletransportar a vuestro oponente fuera de la pantalla, quedaréis descalificados. Aun así, si descubrís algún fallo curioso en el sistema contádnoslo, que seguro que nos parece interesante.

No podéis importar ningún archivo del proyecto a excepción de AutomatedRobot (como se ve en el archivo random_robot.py). Esto es con el objetivo de que no creéis objetos que no debéis. Por un lado, las monedas las crea el sistema y vosotros en nigún caso podéis crearlas. Por otro, la única forma de crear

proyectiles es la función cast_basic_attack, que está limitada por la velocidad de ataque de vuestro robot.

Ejemplo: Random robot

Como veis, para que una clase herede de otra ponemos a la segunda ente paréntesis:

```
class RandomRobot(AutomatedRobot):
```

La función decide escoge aleatoriamente del conjunto de atributos (self.stat_keys es una lista de todos los atributos que podéis mejorar al subir de nivel) qué atributos mejorar y solicita mejorarlos. upgrade_stats solo mejorará el atributo si se tienen puntos de experiencia suficientes o el atributo no ha llegado a su límite, pero se le puede llamar tantas veces como queráis.

Después, decide aleatoriamente hacia qué dirección moverse y cambia la velocidad del robot con set_velocity o detiene el movimiento con stop.

Finalmente, decide aleatoriamente si disparar o no y en qué dirección. Solo ejecuta el disparo si el proyectil no apunta hacia sí mismo (random sí, pero tonto no). Podéis ver que si decide no disparar devuelve None, esto no es del todo necesario porque Python lo va a hacer si no le dices que devuelva algo en específico, pero bueno, así más clarito.

DESCRIPCIÓN DEL TORNEO

- Los robots se enfrentarán en una serie de combates por pares a modo de torneo. Los ganadores de cada fase pasarán a competir entre ellos sucesivamente. Finalmente, el robot vencedor será aquel que quede invicto.
- En cada combate, los robots se enfrentan en una serie de tres duelos. El ganador será aquel que gane más duelos.
- Si uno de los dos robots muere durante el duelo, el duelo finaliza. El robot superviviente será el ganador. Si ambos murieran simultáneamente (una cosa rara-rara), ese duelo no tendría ganador.

- Si un robot alcanza el nivel 10, el duelo finalizará en un máximo de 10 segundos. Si el tiempo acaba y ambos robots sobreviven, aquellos que hayan logrado el nivel 10 serán los ganadores.
- En caso de que ambos robots hayan sido declarados ganadores en el mismo número de duelos, el desempate se hará comparando la suma de los puntos de experiencia obtenidos por cada uno en aquellos duelos en los que hayan sido declarados ganadores.
- En caso de que ocurra algo increíblemente impredecible, Zimatek se reserva el derecho a aplicar criterios excepcionales.