



Departament d'Enginyeria  Informàtica i Matemàtiques  UNIVERSITAT ROVIRA I VIRGILI	Tècniques avançades de programació
	TAP
	Curso 24/25
	Primera Convocatoria
	Pràctica: Minecraft Agent Framework

Pràctica TAP (Minecraft Agent Framework)

EQUIPO:

Mireya Marín Calvo
Nayra Li Luna Rodríguez

Index

Minecraft Agent Framework-----	2
Interacción con Minecraft-----	2
Implementación de Agentes-----	2
Software Testing-----	3
UnitTest-----	3
Coverage-based-----	3
MAIN-----	3
Conclusión-----	4

Minecraft Agent Framework

El **Minecraft Agent Framework** es una herramienta innovadora diseñada para facilitar el desarrollo y la ejecución de agentes en servidores de Minecraft. Este framework permite a los desarrolladores crear agentes en Python que pueden interactuar con el entorno del juego, incluyendo movimiento, interacción con bloques, y comunicación a través del chat. Además, soporta paradigmas de programación funcional y reflectiva, brindando gran flexibilidad y potencia a los desarrolladores.

Interacción con Minecraft

El framework proporciona capacidades avanzadas para interactuar con el mundo de Minecraft:

- **Movimiento:** Controla el desplazamiento de los agentes en el entorno.
- **Interacción con bloques:** Los agentes pueden manipular, destruir, o colocar bloques en el mundo.
- **Interacción con el chat:** Los agentes pueden enviar y recibir mensajes a través del chat del servidor, facilitando la comunicación.

Implementación de Agentes

A continuación, se listan los agentes implementados:

1. **InsultBot:** Genera respuestas humorísticas o sarcásticas en el chat.
2. **TNTBot:** Interactúa con bloques de TNT para crear eventos explosivos controlados.
3. **OracleBot:** Responde preguntas y proporciona información.
4. **RainbowBot:** Crea patrones coloridos en el mundo.
5. **HouseBot:** Construye estructuras tipo casa.
6. **EmotionsBot:** Capta las emociones y da una respuesta basada en ellas.

Ahora se añade un resumen de cada bot implementado.

EmotionsBot usa **programación funcional** implementando un filter. Este filtra una lista de mensajes de chat para extraer aquellos que contienen alguna palabra de la lista indicada.

En este caso la lista `positive_words`.

```
positive_responses = list(filter(lambda x: any(word in x.message.lower() for word in
positive_words), chat_messages))
```

Esta línea sigue la siguiente estructura, donde la función se pasa como una **lambda** (**paradigma funcional**), y el iterable es `chat_messages`.

filter(función, iterable)

Tanto **RainbowBot** como **HouseBot** permiten la construcción en Minecraft mediante operaciones matemáticas.

TNTBot coloca bloques TNT cerca del jugador y los activa colocando bloques de fuego.

Para estos 3 bots es esencial saber la posición del jugador con la función

mc.player.getTilePos().

OracleBot se basa en usar una estructura de datos de tipo **diccionario**. En ella se guardan parejas de tipo `key:value`. Cuando el jugador hace una pregunta, se accede al diccionario por la clave:

dictionary.get(keyname, value)

En nuestro caso `keyname` es la pregunta y `value` es un mensaje por defecto que se devuelve si no se encuentra la clave.

InsultBot tiene una implementación muy sencilla. Tiene una lista con respuestas. Cuando se llama a la función **insult()** se coge una respuesta aleatoria de dicha lista.

Software Testing

UnitTest

Para probar el funcionamiento correcto de los bots, utilizamos `unittest` para crear tests específicos para cada agent y ver sus comportamientos.

Coverage-based

Utilizamos la opción de añadir workflow que proporciona gitHub. Para esto hemos creado una carpeta "github" en el proyecto. Y dentro de esta otra carpeta "workflows" con el fichero `test.yml`. En este fichero indicamos que queremos ejecutar la carpeta test de la rama main con `pytest`. Para que esto funcione también se ha añadido un fichero `requirements.txt` con todas las dependencias que usamos.

MAIN

Para que se ejecuten los bots se ha implementado una clase main, que funciona como un menú. Al inicio se importan las librerías y los bots.

Se crea una clase **CommandHandler** que permite registrar, ejecutar, y mostrar comandos personalizados.

- **`_init_(self, mc)`**
Se usa como constructor. Crea la conexión con Minecraft y crea un diccionario (`self.commands`) donde guardar los comandos
- **`register_command(self, name, function, description="No description")`**
Añade al diccionario el nuevo comando. Si no se le pasa descripción se añade "No description" por defecto.
- **`execute_command(self, name, *args)`**
S'executa el comand que tingui el nom passat com argument. Si no existeix es llança un missatge d'error.
- **`show_help(self)`**
Executa una iteració imprimint tots els comandos amb la seva descripció.

Clase main():

Se crea una instancia de `CommandHandler` a la que se le añaden los comandos de la siguiente manera:

```
handler.register_command("insultbot", lambda: InsultBot(mc).start_insulting(1), "Start InsultBot")
```

La función se pasa como una lambda para evitar crear código extra.

Se imprime por pantalla el mensaje:

Welcome to Minecraft! Type '--help' to see available commands.

Finalmente se realiza un bucle infinito donde se espera la respuesta del jugador. Con **mc.events.pollChatPosts()** se captura la respuesta y se trata para extraer el comando y sus argumentos. Luego se ejecuta el pedido.

Conclusión

El Minecraft Agent Framework proporciona una plataforma sólida y flexible para crear agentes personalizados en Minecraft, combinando interacción avanzada con el entorno y paradigmas modernos de programación. Con su enfoque en pruebas unitarias y capacidades innovadoras, ofrece un entorno ideal para el desarrollo y la exploración.