MINECRAFT AGENT FRAMEWORK

Técnicas Avanzadas de Programación

Àlex Casanova Margalef
Marius Ionut Ursache

ÍNDICE

1. Objetivo.

El objetivo de esta tarea ha sido la implementación de un framework haciendo uso de Python y la librería "mcpi" que nos permita implementar agentes (bots) dentro de un servidor de Minecraft ejecutado en un entorno "localhost". Algunos ejemplos de implementaciones de agentes han sido:

BotChat: Agente que lee los mensajes del chat y responde con respuestas predefinidas basadas en palabras clave en el chat de Minecraft.

BotGpt: Agente que se conecta a la API de ChatGPT para responder mensajes al chat de Minecraft.

BotInsulto: Agente que envía insultos predefinidos de forma aleatoria cada cierto tiempo en el chat de Minecraft.

BotTnt: Agente que detecta la posición del jugador y coloca y enciende un bloque de TNT en una posición cercana.

BuilderBot: Agente que permite la construcción de torres y paredes en una posición cercana a la del jugador.

StatsBot: Agente que monitorea estadísticas del servidor y del jugador.

2. Manual de instalación.

1) Asegurarse que el servidor ha sido instalado de:

   `git clone https://github.com/AdventuresInMinecraft/AdventuresInMinecraft-PC`

2) Realizar las siguientes instalaciones desde una consola:
   Para hacer uso de las librerías de Minecraft:

   `pip install mcpi`

   (Si no se reconoce el comando `pip`, instalar con `python -m ensurepip –upgrade`)

   Para poder conectarse a la API de OpenAI y poder usar ChatGPT:

   `pip install openai`

3) Configurar la variable de entorno OPENAI_API_KEY con una API KEY de OpenAI:

   Windows:

   - CMD: `set OPENAI_API_KEY="sk-…"`
   - PowerShell: `$env:OPENAI_API_KEY="sk-…"`

   MAC o Linux:

- export OPENAI_API_KEY="sk-…"

4) Ejecutar Minecraft en la versión 1.12.
5) Añadir un servidor a Minecraft con dirección IP “localhost”.
6) Ejecutar el archivo “StartServer.sh”.
7) Una vez inicializado, entrar al servidor.
8) Ejecutar el archivo “main.py” para poder utilizar el framework.

3. Test y cobertura del código.

Para verificar correctamente el funcionamiento de todo el proyecto en conjunto, hemos desarrollado unos Unit Tests sobre los cuales podemos aplicar Code Coverage:

- Test_Agentes.py: Se comprueba el funcionamiento de la clase “AgenteBase.py” como clase abstracta.
- Test_Basico.py: Se comprueba la interacción con el chat y el mundo del servidor de Minecraft.
- Test_Bot_Chat.py: Se comprueba el funcionamiento de BotChat.py, si responde adecuadamente a las preguntas predefinidas y no predefinidas y si envía mensajes al chat correctamente.
- Test_Bot_Insulto.py: Se comprueba que BotInsulto.py envía al menos un insulto.
- Test_Bot_Tnt.py: Se comprueba que BotTnt.py pueda colocar y activar bloques de TNT.
- Test_Builder_Bot.py: Se comprueba que BuilderBot.py pueda construir una torre y una pared.
- Test_Framework.py: Se comprueba que el framework sea capaz de registrar, inicializar y detener agentes.
- Test_Stats_Bot.py: Se comprueba que BotStats.py monitoree correctamente las estadísticas del servidor y del jugador.

Los resultados del último Code Coverage realizado se encuentra en el directorio “/htmlcov/index.html”. Los pasos a seguir para realizarlo son los siguientes:

1) Instalar “coverage”:
   - pip install coverage
2) Situarnos en el directorio actual del proyecto:
   cd /MyAdventures/
3) Ejecutar Code Coverage con:
   coverage run -m unittest discover -s tests
4) Ver los resultados:
   - Para ver los resultados en una terminal:
     coverage report -m
   - Para ver los resultados en un informe HTML más detallado:
     coverage html
     start htmlcov/index.html (Windows)
     open htmlcov/index.html (macOS)
     xdg-open htmlcov/index.html (Linux)

4. Ejemplo de Coverage

Siguiendo los pasos anteriores y generando un informe HTML obtenemos los siguientes resultados:

File ▲	statements	missing	excluded	coverage
agents__init__.py	0	0	0	100%
agents\BotChat.py	41	3	0	93%
agents\BotInsulto.py	29	3	0	90%
agents\BotTnt.py	34	18	0	47%
agents\BuilderBot.py	59	26	0	56%
agents\StatsBot.py	78	23	0	71%
framework__init__.py	0	0	0	100%
framework\AgenteBase.py	11	2	0	82%
framework\Framework.py	23	0	0	100%
mcpi__init__.py	0	0	0	100%
mcpi\block.py	88	6	0	93%
mcpi\connection.py	37	5	0	86%
mcpi\event.py	26	13	0	50%
mcpi\minecraft.py	118	41	0	65%
mcpi\util.py	13	2	0	85%
mcpi\vec3.py	83	57	0	31%
tests\Test_Agentes.py	31	2	0	94%
tests\Test_Basico.py	24	2	0	92%
tests\Test_Bot_Chat.py	51	2	0	96%
tests\Test_Bot_Insulto.py	35	2	0	94%
tests\Test_Bot_Tnt.py	39	2	0	95%
tests\Test_Builder_Bot.py	33	2	0	94%
tests\Test_Framework.py	40	2	0	95%
tests\Test_Stats_Bot.py	44	2	0	95%
Total	**937**	**215**	**0**	**77%**

En este reporte se observan los archivos que han pasado por el coverage, con una columna indicando el nombre, otra indicando el número de línea ejecutables que tiene el código (Statements), otra que indica las líneas de código no ejecutadas por el banco de pruebas (Missing), otra que indica las que han sido excluídas (Excluded) y una última que muestra el porcentaje de líneas ejecutables cubiertas por las pruebas.

5. Creación de agentes.

Dado que tenemos un framework es posible añadir nuevos agentes de forma sencilla y cómoda, y el procedimiento a seguir es el siguiente:

1) Crear un nuevo archivo “Bot{nombre}.py” en el directorio “/agents” que es donde están guardados los que tenemos a disposición actualmente.

2) Hacer que el agente creado herede de la clase “AgenteBase.py” e implementar los métodos heredados “start()”, que debería contener las acciones que va a realizar y “stop()”, que debería detener su ejecución, e implementar de forma opcional funcionalidades extra que puedan ser consideradas necesarias.

   Ejemplo de implementación (se van a mostrar las líneas de código esenciales).

```
from framework.AgenteBase import AgenteBase
class Bot{nombre}(AgenteBase):
        def __init__(self, mc):
                super().__init(“Bot{nombre}”)
                self.mc = mc
                …
        def start(self):
                …
        def stop(self):
                …
```

3) Registrar el agente en el archivo “main.py”. Es el framework el encargado de realizar dicha tarea:

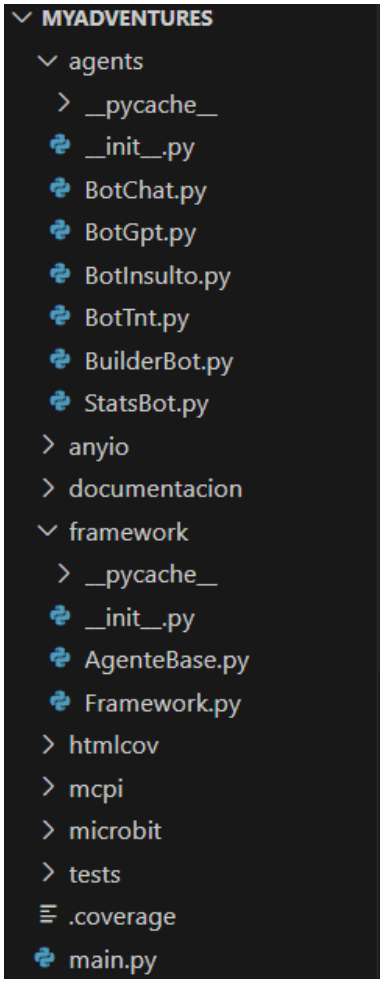
```
from agents.Bot{nombre} import Bot{nombre}
mc = minecraft.Minecraft.create()
framework = Framework()

# Crear
bot_nombre = Bot{nombre}(mc)

# Registrar
Framework.register_agent(bot_nombre)
```

6. Estructura del proyecto.

El proyecto ha sido estructurado de la siguiente forma:



A continuación, se describen los archivos y directorios más relevantes:

- main.py: Es el flujo principal del programa. Inicializa el framework y registra a los agentes, ejecutándolos como threads independientes. Maneja la interrupción con “KeyboardInterrupt” para finalizar el programa.
- AgenteBase.py: Clase abstracta que define la estructura básica que deben seguir los agentes, obligándolos a implementar los métodos “start()” y “stop()”.
- Framework.py: Gestiona la ejecución de agentes como hilos independientes usando la librería “threading”. Permite crear agentes, iniciar y detener su ejecución.
- Directorio /agents/: Contiene el código de los agentes que han sido implementados. Los archivos que contiene han sido explicados en el apartado **Objetivo**.

- Directorio /tests/: Contiene los unit tests que hemos diseñado para verificar el funcionamiento del proyecto. Los archivos que contiene han sido explicados en el apartado **Tests y cobertura del código**.
- Directorio /mcpi/: Contiene las librerías de Minecraft. Venía incluido en el repositorio del servidor.
- Directorio /htmlcov/: Contiene los resultados del Code Coverage aplicado a los unit tests diseñados.

7. Diagrama de la funcionalidad.