Minecraft Python Framework con mcpi

Índice

Iván García Pallarès

Objetivo

Desarrollar un framework en Minecraft, ejecutado en un servidor localhost, que permita implementar y gestionar funcionalidades interactivas utilizando Python y la librería MCPI. El objetivo principal es integrar características dinámicas que mejoren la experiencia dentro del juego, incluyendo:

Creación de un conjunto de dinamita para realizar explosiones de forma controlada en el entorno de Minecraft.
Gestión de una lista de almacenamiento de insultos, proporcionando una funcionalidad para registrar y consultar elementos en tiempo real desde el juego.
Implementación de una API conectada con ChatGPT, permitiendo realizar consultas y recibir respuestas directamente en el chat del juego.
Adicionalmente, el framework hace uso de funciones clave como la impresión de menús interactivos en el chat y la lectura en tiempo real de las interacciones de los jugadores, fomentando una integración intuitiva entre el usuario y las herramientas desarrolladas.

Pasos para la instalación y configuración:

Asegurar que el servidor esté instalado de:

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

Instalaciones necesarias desde consola bash:

- Para hacer uso de las funciones del Minecraft
  ```
  pip install mcpi
  ```
- Para asignar variables de entorno en un archivo .env
  ```
  pip install python-dotenv
  ```
- Para hacer uso de la versión correcta del ChatGPT
  ```
  pip install openai==0.28
  ```
- Crear archivo .env en la carpeta src con la siguiente linea de codigo para conectar con la API del chatgpt, añadiendo la API_Key Personal:
  ```
  OPENAI_API_KEY="sk-..."
  ```
- Tener el Minecraft instalado.
- Abrir Minecraft con la versión 1.12 (Recomendación resolución 1024x768).
- Añadir un servidor en Minecraft en el apartado de Multijugador con ip "localhost".
- Abrir la carpeta del servidor, acceder a Server/ y ejecutar server.exe.
- Entrar dentro del servidor recientemente creado desde Minecraft.
- Ejecutar el archivo Main.py para implementar el Framework.

Control sobre el Código

Para Tener un control sobre el código creado tenemos unos test que son los archivos:

- Interactive_Test.py (Test interactivo con el usuario para probar impresiones por chat y la lectura de este).
- Unit_Test.py (Test sobre funciones automáticas sin necesidad de interacción directa del usuario).

Para controlar el código que tenemos y que ejecutamos podemos hacer un Coverage con:

- Instalación del Coverage

```
pip install coverage
```

- Ejecución del Coverage:
  - Situarnos en la carpeta donde se encuentra el proyecto.

```
cd ../Minecraft-Agent-Framework/src
```

  - Ejecutamos el Coverage en el Unit_Test.

```
coverage run -m unittest Unit_Test.py
```

  - Revisamos el reporte proporcionado.

```
coverage report
```

Ejemplo:

Siguiendo los pasos anteriores para ejecutar el análisis sobre el código eligiendo el test Unitario obtenemos los siguientes resultados:

Ejecución del Coverage:

```
PS C:\Users\playe\Desktop\Trabajos\Bots_Minecraft_Server\Minecraft-Agent-Framework\src>
 coverage run -m unittest .\Unit_Test.py
Test con ChatGPT:
Contexión ha sido:
..Se ha intentado escribir el insulto 60, cuando solo tenemos 6 elementos en la lista.
rata Se ha añadido a la lista de insultos global.
rata Ya existe dentro de la lista.
.1  entidad/es disponible/s.
.
----------------------------------------------------------------------
Ran 4 tests in 1.578s

OK
```

Como podemos ver se han ejecutado los 4 test principales automáticos de manera correcta en un tiempo de 1.578 segundos los cuales es un poco alto ya que dentro del código tenemos algunos `time.sleep(0.2)` para organizar la salida de textos, sobre todo los menús.

Reporte del Coverage:

```
PS C:\Users\playe\Desktop\Trabajos\Bots_Minecraft_Server\Minecraft-Agent-Framework\src>
 coverage report
Name                        Stmts   Miss  Cover
-----------------------------------------------
Acciones\chatgpt.py            22      8    64%
Acciones\dinamita.py           21      8    62%
Acciones\insult_bot.py         43     15    65%
Acciones\lectura_chat.py       12      9    25%
Acciones\textos.py             35      0   100%
Unit_Test.py                   26      1    96%
-----------------------------------------------
TOTAL                         159     41    74%
```

En este reporte vemos los archivos por los que ha pasado la ejecución el código del test seleccionado en la ejecución del Coverage. Donde en cada columna tenemos la siguiente información:

- **Name**
  Esta columna indica el nombre del archivo que han sido analizados.
- **Stmts (Statements)**
  Representa la cantidad de líneas de código ejecutables que tiene cada archivo.
- **Miss**
  Representa las líneas de código ejecutables no ejecutadas por el banco de pruebas.
- **Cover**
  Es el porcentaje de líneas ejecutables que fueron cubiertas por las pruebas.

Dentro de nuestro caso podemos ver que en los tenemos algunas líneas no ejecutadas en algunos archivos, esto corresponde porque tenemos una función concreta que es: `def Lectura_chat()`, la cual hace una espera activa hasta que el jugador escribe algo por el chat, y este contenido es el que devuelve como resultado. De ahí que tenemos otro test para probar a parte esta función.

Creación de un nuevo agente

Como este proyecto trata de ser un Framework hay que tener en cuenta que podemos añadir nuevos agentes o implementaciones de manera sencilla y cómoda para el un usuario cualquiera.

Ejemplo de nuevo agente:

- Crear un nuevo archivo (teletransporte.py) en la carpeta Acciones, que es donde tenemos todos los Agentes.
  Contenido del Agente (teletransporte.py):

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import random

mc = minecraft.Minecraft.create("localhost")


def teleport():
  pos = mc.player.getTilePos()
   mc.player.setPos(random.randint((pos.x-10000), (pos.x+10000)), pos.y,
random.randint((pos.z-10000), (pos.z+10000)))
```

- Añadir la función al archivo Main.py:
  Contenido del Main.py:

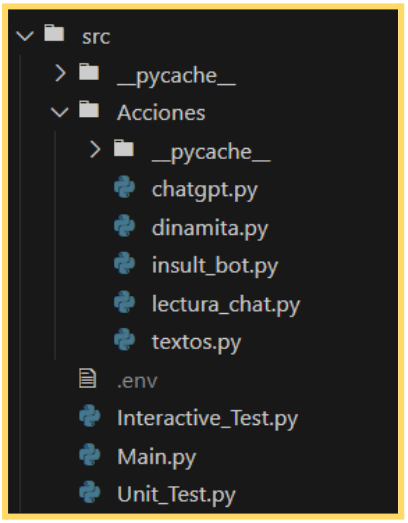
```
from Acciones.teletransporte import *
```

- Añadir en el archivo Main.py la llamada a la función del nuevo Agente dentro del bucle que interactuar con el jugador:
  Contenido del Main.py:

```
elif decision == "4":
    mc.postToChat("Teletransporte")
    teleport()
    print_Menu_Acciones()
```

Archivos

El proyecto está organizado de la siguiente manera:



1. Main.py

El archivo principal del framework.
Define el punto de entrada del programa y coordina la ejecución de las distintas funcionalidades incluidas en la carpeta `Acciones`. Este archivo establece las conexiones, inicializa los módulos necesarios y maneja la interacción con el jugador.

2. Carpeta de Acciones

Contiene los módulos principales del framework que implementa las funcionalidades específicas.

a. chatgpt.py

- Propósito:
  - Conecta el framework con la API de ChatGPT.
  - Permite que el jugador haga peticiones directamente desde el chat de Minecraft y obtenga respuestas contextuales.
- Características principales:
  - Procesa los mensajes enviados por el jugador desde el chat.
  - Realiza solicitudes a la API de OpenAI.
  - Envía las respuestas generadas al chat del juego.

b. dinamita.py

- Propósito:
  - Crea estructuras explosivas (dinamitas) con dimensiones personalizadas definidas por el jugador en el chat.
- Características principales:
  - Recibe las dimensiones deseadas a través del chat.
  - Regula la distancia en base al tamaño para evitar daños al jugador.
  - Genera las dinamitas automáticamente en el mundo del juego.

c. insult_bot.py

- Propósito:
  - Gestiona un sistema de insultos interactivo.
  - Proporciona un menú donde el jugador puede:
    - Ver la lista de insultos.
    - Añadir nuevos insultos.
    - Pedir que el bot lo insulte.
- Características principales:
  - Interfaz sencilla a través del chat del juego.
  - Base de datos simple de insultos personalizables.
  - Función para seleccionar y mostrar insultos aleatorios.

d. lectura_chat.py

- Propósito:
  - Facilita la programación funcional al implementar una espera activa que captura lo que el jugador escribe en el chat del juego.
- Características principales:
  - Espera de manera constante a que el jugador escriba un comando o mensaje.
  - Retorna el texto ingresado como parámetro para su procesamiento posterior.
  - Fundamental para la interacción dinámica en tiempo real.

e. textos.py

- Propósito:
  - Almacena los menús y textos predefinidos que se muestran en el chat del juego para guiar al jugador en el uso de los comandos disponibles.
- Características principales:
  - Define los menús de interacción para las funciones principales.
  - Centraliza la gestión de textos para facilitar la traducción o modificaciones.

3. Test

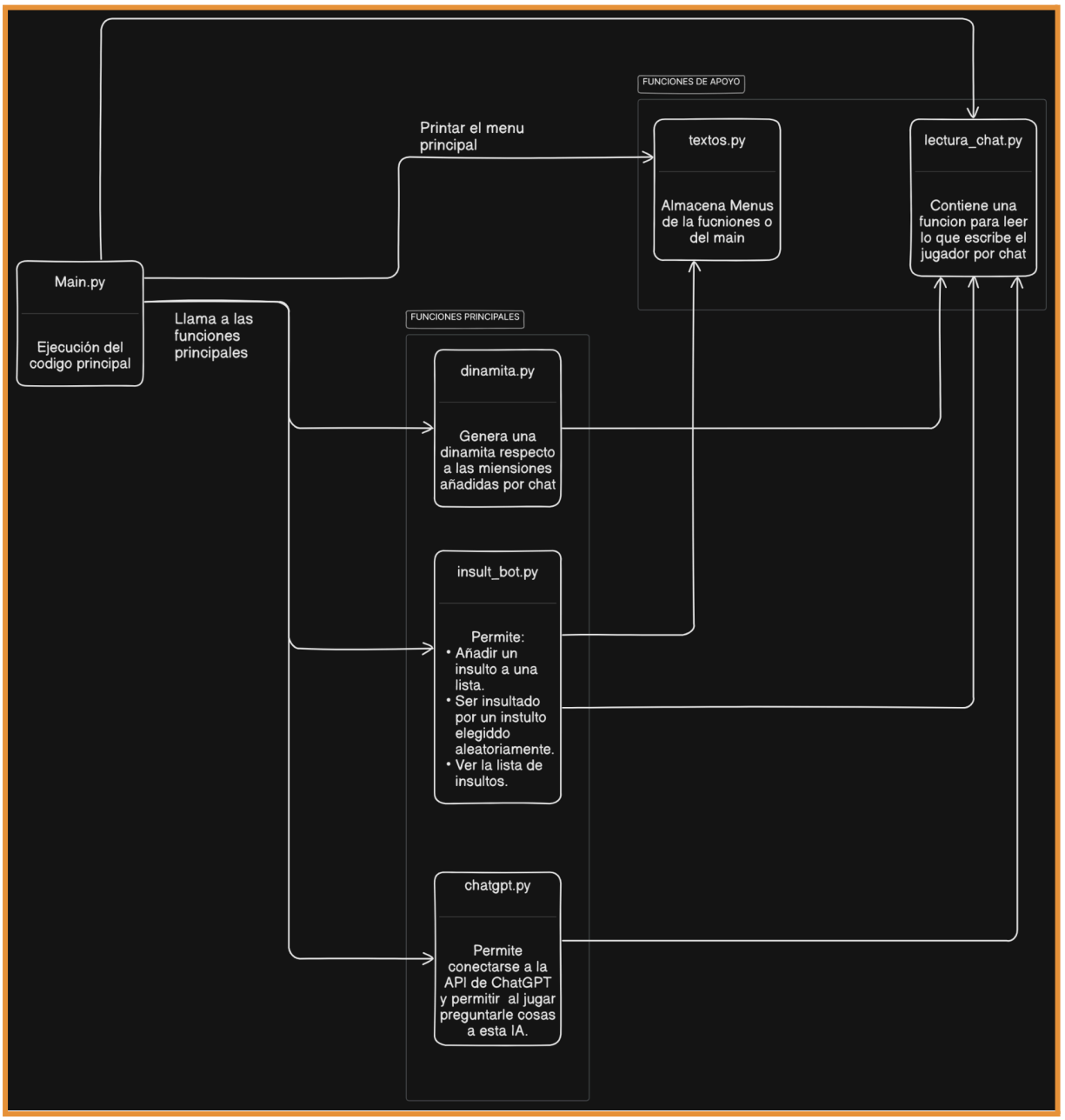
a. Interactive_Test.py

- Propósito:
    - Realiza pruebas manuales para verificar la funcionalidad de la función lectura_chat().
- Características principales:
    - Permite al usuario interactuar directamente con el chat del juego durante las pruebas.
    - Verifica la capacidad de la función para capturar correctamente la entrada del jugador.
    - Diseñado para pruebas más flexibles y en entornos reales.

b. Unit_Test.py

- Propósito:
    - Implementa pruebas automáticas utilizando el módulo unittest para validar las funciones principales del framework.
- Características principales:
    - Comprueba la correcta ejecución y los resultados esperados de las funciones clave.
    - Asegura la estabilidad del código frente a cambios o nuevas implementaciones.

Diagrama de la funcionalidad



Este diagrama ilustra la estructura y las relaciones entre los archivos del proyecto:

1. **Main.py**:
    - Es el punto de inicio del programa.
    - Llama a las **funciones principales** para ejecutar las funcionalidades del framework.
2. **Funciones principales**:
    - **dinamita.py**:
      Genera dinamitas según las dimensiones indicadas por el jugador en el chat.
    - **insult_bot.py**:
      Gestiona una lista de insultos, permitiendo añadir nuevos, visualizar la lista o recibir un insulto aleatorio.
    - **chatgpt.py**:
      Conecta con la API de ChatGPT para que el jugador interactúe con la IA desde el chat del juego.
3. **Funciones de apoyo**:
    - **textos.py**:
      Almacena los menús y mensajes que se mostrarán al jugador en el chat.
    - **lectura_chat.py**:
      Captura los mensajes del jugador en el chat y los devuelve como entrada para las demás funciones.
4. **Relaciones clave**:
    - `textos.py` y `lectura_chat.py` son utilizados por todas las funciones principales para interactuar con el jugador.
    - El menú principal se gestiona desde `Main.py` y se conecta a los módulos principales para ejecutar cada funcionalidad.