Práctica 1

23/11/2022

Sebastian Pasker

Sistemas Distribuidos

Introducción

En esta práctica vamos a crear un MMO con kafka y sockets, la estructura se basará en la demandada por la práctica.

(IMAGEN A)

Guía de despliegue

Para poder desplegar el proyecto lo que tendremos que realizar desde la terminal es lo siguiente:

```
sebas-p ~> # kafka_* = kafka y la versión
sebas-p ~> cp ./properties/server.properties
./bin/kafka_*/config/server.properties
sebas-p ~> cp ./properties/zookeeper.properties
./bin/kafka_*/config/zookeeper.properties
sebas-p ~> ./bin/kafka_*/bin/zookeeper-server-start.sh
./bin/kafka_*/config/zookeeper.properties &> ./logs/zookeeper.log &
sebas-p ~> ./bin/kafka_*/bin/kafka-server-start.sh
./bin/kafka_*/config/server.properties &> ./logs/server.log &
```

También se podrá correr:

```
sebas-p ~> pip3 install -r dependencies.txt
```

Para la base de datos:

```
sebas-p ~> python3 scripts/create_tabler_player.py # Para el engine
sebas-p ~> python3 scripts/create_weather_table.py # Para el weather
```

En diferentes terminales, corremos:

```
sebas-p ~> python3 AA_Register.py <config.json>
sebas-p ~> python3 AA_Weather.py <config.json>
sebas-p ~> python3 AA_Engine.py <config.json>
sebas-p ~> python3 AA_Player.py <config.json>
sebas-p ~> python3 AA_NPC.py <config.json>
```

Funcionamiento

AA_Engine

El AA_Engine se encargará del sistema manejo de datos del juego. Conteniendo funciones para manejar el map, los clientes y los NPCs. Además del manejo principal de kafka, de los sockets y de la base de datos. Podría decirse que es el director de la orquesta.

Funciones base de datos:

- ***save_map()***: Guarda el mapa.
- ***read_map()***: Lee el mapa.
- ***read_client()***: Lee los clientes para el login.
- Otros.

Funciones kafka:

- ***manage_npcs()***: Lee el consumidor de npcs.
- ***recieve_key_clients()***: Lee el consumidor del players.
- ***send_dict_npc()***: Productor de npcs.
- ***send_dict_players()***: Productor de players.
- ***send_map()***: Productor del mapa.
- ***send_weather()***: Productor del Weather.
- Otros.

El director principal del programa se llama *execute_threads_start_game(),* que creando diferentes hilos para el funcionamiento correcto del mapa.

El resto de funciones son para el manejo del Engine, incluyendo funciones de la recepción del weather, el manejo de los comandos del cliente y npc, el login, etc. Para encontrar el funcionamiento **interno** del engine, nos tendremos que ir a *src/.* Incluyendo este utils/ para utilidades generales, */expections* para el manejo de excepciones, Map.py que es una clase del Mapa, Player.py que es el jugador y NPC.py que es para el NPC.

Para el procesado de datos tanto en sockets como en kafka. Hemos utilizado un diccionario creado en src/utils/Sockets_dict.py

AA_Weather

Se encarga de seleccionar 4 ciudades aleatorias de la base de datos *clima.db.* Y enviarlo al AA_Engine con sockets.

AA_Registry

Se encarga del registro y la edición del usuario. Solicitándolo con sockets y guardándolo en la base de datos.

AA_NPC

Se encarga del funcionamiento de los NPCs. Para ello va enviando diferentes direcciones al Engine y este va lo interceptando y procesando. A la vez que va recibiendo diferente información como si el npc está muerto (para dejar de envíar) y más información del engine.

AA_Player

En conjunto con el Engine es la parte más compleja, teniendo como papel recibir el mapa y el player asignado e ir enviando información de los movimientos que va haciendo. Además se encarga de procesar el registro, el login, la edición de usuario, la espera hasta que la partida empiece, el procesado de información del weather para impresión en el mapa, etc.

Funciones kafka:

- ***read_weather_cli()***: Consumidor del clima (enviado desde el engine).
- ***read_map_cli()***: Consumidor del mapa.
- ***evaluate_npc()***: : Consumidor del player.
- ***send_move_cli()***: Productor del movimiento.

Map

Es una clase que se encarga de procesar el funcionamiento interno del juego. Procesa funciones como una pelea, recopilar comida o una mina por el jugador, el movimiento interno del NPC al igual que una pelea del NPC, o el reparto de información y procesado en el mapa.

Funciones:

- ***npc_random_position()****:* Introduce de forma aleatoria npcs en el mapa.
- ***player_random_position()****:* Introduce de forma aleatoria players en el mapa.
- ***raw_string_to_matrix()***: Convierte un string crudo del mapa en una matriz de posiciones (para envío kafka).
- ***to_raw_string()***: Convierte la matriz de posiciones del mapa en un string crudo (para envío kafka).
- ***print_color()***: Función de impresión del mapa.
- ***evaluate_mine()****:* Evalúa una mina.
- ***evaluate_food()***: Evalúa una comida.
- ***evaluate_space()***: Evalúa un espacio (por el jugador).
- ***evaluate_space_npc()***: Evalúa un espacio, comida o mina (por el npc).
- ***sum_weather()****:* Devuelve el EF o EC según el weather.
- ***evaluate_fight()***: Evalúa una pelea entre dos players.
- ***evaluate_npc()***: Evalúa una pelea entre un player a npc.
- ***evaluate_fight_npc()***: Evalua una pelea entre un npc a player.
- ***evaluate_move()***: Director de movimientos del player.
- ***evaluate_move_npc()***: Director de movimientos del NPC.
- ***Setters.***
- ***Getters.***
- ***Otros.***

Player

Gestiona el funcionamiento de un jugador en el juego. Movimiento, estado, muerte, etc.

NPC

Parecido a el player pero con menos funcionalidades.

Características del sistema

Congruencia

El sistema es capaz de interpretar los fallos y procesarlos. Sobre todo en el AA_Player, es capaz de detectar cuando el servidor está caído y reaccionar ante ello volviendo al menú inicial del Player.

Independencia

Todos los módulos están conectados entre sí pero reaccionan como independientes, refiriéndonos a que si uno de ellos no está corriendo, el sistema puede seguir su funcionamiento normal. Esto se puede ver por ejemplo en el AA_Weather, que en el caso que no esté corriendo el AA_Engine procesa las estaciones de cada ciudad como uno que no influencie a la jugabilidad (que esté entre 10 y 25 grados). Lo mismo con el AA_NPC, en el caso de que no esté corriendo el Engine lo percibe pero puede seguir su funcionamiento normal sin él.

Sostenibilidad y escalabilidad.

El proyecto sigue al pie de la letra la resolución de un problema grande en problemas menores. Creando un sistema sostenible debido a que cada parte del sistema es fácilmente identificable. Y escalable, debido a que cada función tiene asignado una tarea concreta, haciéndolo fácil a expandir e implementar nuevas funcionalidades. Además, está creado por un sistema de archivos de configuración JSON de tal manera que sea fácil de ejecutar con diferentes parámetros.

Paralelismo

Se basa en una ejecución de lectura, escritura y procesado. Habiendo una variedad de distintos hilos que se ejecutan de manera paralela en el programa. Esto se puede observar en funciones execute_thread_start_game() del AA_Engine o *start_game()* del AA_Player.

Automatización

La práctica está creada con scripts y dinámicas de automatización que facilita el trabajo del programador. Como la creación de la base de datos, la ejecución de programas como kafka o creación de archivos de configuración.

Optimización

El sistema está creado desde una perspectiva óptima, en la cual todo se ejecuta de la manera más rápida posible pero sin sobrecargar los componentes del sistema. Esto se puede observar en funciones como send_map() del AA_Engine, que envía el mapa cada segundo por si se ha perdido en el transcurso o cuando hay un cambio en el mapa.

Persistencia

Se considera persistente debido a que guarda el estado de la partida. Si se cae el servidor a mitad de partida, seremos capaces de volver a introducirnos de nuevo en la partida con el mismo estado que teníamos.

Resultado

Como resultado obtenemos un juego con pleno funcionalidad en kafka y sockets, además de otras características que lo convierten en un sistema distribuido completo: