EDACraft

Navegación y Tareas Automatizadas en Minecraft

Algoritmos y Estructuras de Datos

ITBA
Segundo Año - Ingeniería Electrónica

9 de mayo de 2025

Introducción y Objetivos

En el presente trabajo práctico se diseñará y construirá un sistema automatizado de juego, mejor conocido como *bot*, en el videojuego **Minecraft**.

El presente trabajo práctico tiene como objetivo integrar conceptos de estructuras de datos, algoritmos de búsqueda, y programación en C++, aplicándolos al control de un *bot* en un entorno tridimensional simulado como lo es **Minecraft**.

1. Arquitectura del Sistema

El sistema está dividido en tres partes que cumplen funciones diferentes:

- **Cliente en C++ (implementado por el estudiante)**: es el núcleo lógico del bot, donde se desarrolla el algoritmo de navegación, y la lógica de toma de decisiones.
- **Servidor de Minecraft**: se ejecuta por separado, corre el mundo donde se desarrolla la simulación.

1.1. Medio de comunicación: JSON

JSON será utilizado como lenguaje común para que el programa en C++ pueda intercambiar información con el bot. Es posible enviarle órdenes al bot y recibir feedback acerca del estado del mismo.

JSON, *JavaScript Object Notation*, es un formato ligero de intercambio de datos que permite representar información estructurada de manera simple y legible tanto para humanos como para máquinas. Aunque su nombre proviene de JavaScript, JSON es independiente del lenguaje y es compatible con casi todos los lenguajes de programación modernos, como C++, Python, Java, y muchos otros.

JSON se basa en dos estructuras fundamentales:

- **Objetos:** Colecciones desordenadas de pares `clave:valor`, similares a los mapas en C++.
- **Arreglos:** Listas ordenadas de valores, equivalentes a vectores o arrays.

Un ejemplo de un objeto JSON es el siguiente:

```
{
  "action": "move",
  "x": 123,
  "y": 64,
  "z": -58
}
```

Este objeto describe un comando con varios atributos: un tipo de acción (cadena de texto *action*), como se trata de la acción *move* se especifica las coordenadas hacia donde debe realizar el movimiento en los siguientes tres atributos x, y y z, todos enteros.

1.1.1. Ventajas de JSON

- Es fácil de leer y escribir.
- Es fácil de generar y procesar automáticamente.
- Es ideal para comunicar procesos que están escritos en distintos lenguajes.

2. A trabajar: tareas y objetivos.

Finalmente llegamos a explicar el núcleo del trabajo: todas las tareas y funcionalidades que deberán implementar en su agente.

En primer lugar deben darle una estructura al funcionamiento de su bot, en otras palabras, diseñar un sistema que permita realizar las acciones correctas en los momentos indicados. Con ese fin, se plantea el uso de una FSM. A partir de los estímulos recibidos del entorno y sabiendo las tareas que es necesario completar deben diseñar una máquina de estado finito que modele el funcionamiento de su agente. Se brinda un código de ejemplo en el *starter code* y como ejemplo didáctico una máquina de estados básica podría ser:

- **Idle**: Estado inicial en el cual el agente espera nuevas instrucciones.
- **Navigate**: El agente se mueve hacia una posición objetivo específica en el mapa.
- **Dig**: Una vez alcanzada la posición objetivo, el agente realiza una acción de minería sobre un bloque objetivo.
- **Return**: El agente regresa a la posición inicial o base tras completar la tarea.

Un aspecto fundamental del bot es cómo navegar y moverse por el entorno. Minecraft es un entorno en 3D, pero que puede ser modelado fácilmente debido al carácter discreto de sus coordenadas. Con el fin de garantizar la correcta navegación por entorno se exige la implementación del algoritmo A estrella.

2.1. Algoritmo A*:

2.1.1. Requerimientos

El algoritmo A* deberá ser implementado por completo por los estudiantes, no se permite utilizar funciones de pathfinding de bibliotecas externas. Además debe funcionar en un entorno 3D cúbico, considerando:

- Obstáculos
- Alturas (el bot puede saltar hasta 1 bloque y caer hasta 6)
- Costo de movimiento (uniforme o basado en el tipo de bloque)

2.1.2. Funcionamiento del algoritmo:

El algoritmo **A*** es uno de los más utilizados para encontrar el **camino más corto** entre un nodo inicial y un nodo objetivo en un grafo, como puede ser un mapa de navegación. Es ampliamente utilizado en videojuegos, robótica y sistemas autónomos.

A* combina dos estrategias:

- **Búsqueda por costo real** (como Dijkstra): minimiza la distancia real desde el inicio.
- **Búsqueda heurística** (como Greedy): prioriza los nodos que "parecen"más cercanos al objetivo.

Para cada nodo n, se calcula una función de costo total:

$$f(n) = g(n) + h(n)$$

donde:

- $g(n)$: costo acumulado desde el nodo inicial hasta n.
- $h(n)$: estimación heurística del costo desde n hasta el nodo objetivo.

2.1.3. Algoritmo A* step by step:

1. Se inicia desde un nodo origen, con $g = 0$.
2. Se calcula $f(n) = g(n) + h(n)$ para los vecinos.
3. Se selecciona el nodo con menor f de una lista llamada **open list**.
4. Se repite el proceso hasta llegar al objetivo o agotar las opciones.
5. Al llegar al nodo objetivo, se reconstruye el camino siguiendo los predecesores.

2.1.4. Heurística

La **heurística** $h(n)$ es fundamental para la eficiencia del algoritmo. Algunas heurísticas comunes:

- **Distancia Manhattan**: $h(n) = |x_1 - x_2| + |y_1 - y_2|$ (cuando sólo se puede mover en líneas rectas).
- **Distancia Euclidiana**: $h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (si se permite movimiento diagonal).

La heurística debe ser:

- **Admisible**: **nunca** sobreestima el costo real.
- **Consistente (monótona)**: para que el algoritmo sea óptimo y eficiente.

2.1.5. Ventajas y Aplicaciones

- Encuentra el camino más corto de forma eficiente.
- Es óptimo si la heurística es admisible.

Se recomienda investigar acerca del algoritmo, una buena referencia inicial es el siguiente video: Computerphile - A*

3. Tareas a Resolver

Se proporcionará un mundo de Minecraft especialmente preparado, con condiciones controladas, iguales para todos los grupos:

- Posición inicial conocida.
- Obstáculos, zonas elevadas, biomas cercanos y estructuras predefinidas.
- Objetos clave colocados en el mapa de manera específica.
- Mundo pacífico, sin entidades hostiles que puedan dañar al bot.

La comunidad de modders de Minecraft ha desarrollado un sinfín de bots con el objetivo de automatizar tareas en los servidores. Sus tareas van desde acciones simples como moderar chats hasta comportamientos complejos como construcción automatizada de estructuras o minado y crafteado automático de objetos. Las tareas que su agente debe realizar sirven para demostrar el correcto funcionamiento de su FSM, su algoritmo A*, y también para demostrar las capacidades más complejas del bot.

Se dividen las tareas en las siguientes categorías:

- Navegación.
- Minería y recolección de recursos.
- Construcción.
- Manejo de inventario.
- Interacción con el entorno.
- Automatización de tareas.

Su programa debe ejecutarse en bucle sin parar y en el readme deben especificar cómo se cumplen las siguientes tareas. No es necesario cumplirlas en el orden en que se mencionan, es decir, se evaluará:

- El cumplimiento (o no) de las tareas.
- Cómo son cumplidas las tareas.

3.1. Navegación

- Ve hacia las coordenadas x, y, z (todo: determinarlas en el mapa)
- Salta.
- Cae a salvo por una caída de 3-5 bloques.
- Camina a través del camino rodeado por lava (sin recibir daño)
- Vuelve a la posición inicial.

3.2. Manejo de Inventario

- Ve hacia el cofre cercano y toma al menos una herramienta y cualquier objeto.
- Equípate una herramienta.
- Tira algún objeto desde el inventario al piso.

3.3. Minería

- Mina un bloque de tierra y agarra el bloque del suelo.
- Mina 3 bloques de madera de roble.
- Mina un cubo de 4x4x4 bloques.
- Deja de minar si te encuentras con un bloque irrompible.

3.4. Construcción

- Coloca alguno de los bloques minados.
- Construye un pilar de 1x3 bloques con los bloques minados.
- Construye una plataforma de 3x3 en el suelo.

3.5. Automatización

Las siguientes tareas será asignadas de manera aleatoria a cada grupo. Solamente deberán implementar la tarea asignada:

3.5.1. Opción 1:

- Mina 3 bloques de madera de roble con la mano.
- Craftea un hacha de madera y una pala de madera.
- Mina 3 bloques de madera de abedul con el hacha.
- Mina con la pala los suficientes bloques de tierra para llegar a la capa de piedra.
- Mina 32 bloques de piedra.
- Sube a la superficie.
- Craftea un horno.

3.5.2. Opción 2:

- Ve hacie el cofre cercano al punto de spawn y toma el pico de netherite.
- Mina todos los bloques delimitados por el area: (todo: definir).
- Escribe un mensjae en el chat listando el tipo y cantidad de bloques minados.

3.5.3. Opción 3:

- Ve hacie el cofre cercano al punto de spawn y toma un diamante.
- Navega hacia el beacon.
- Activa el beacon con velocidad 2.

3.5.4. Opción 4:

- Ve hacie el cofre cercano al punto de spawn y toma el pico de diamante y todos los palos disponibles.
- Haz un agujero en linea recto hacia abajo en la coordenada: (todo), hasta caer en la cueva.
- Mina 32 bloques de carbon.
- Craftea todas las antorchas posibles.
- Ilumina la cueva.

4. BONUS

Minecraft plantea una oportunidad única para la implementación de bonus, debido a que se trata de un juego tipo "sandbox". Se dejan algunas propuestas de BONUS, pero el límite está en su imaginación (y su tiempo).

- Encuentren los easter eggs.
- Autominer (bot de minado automatizado para áreas amplias).
- Bot explorador (bonus si puede volar).
- Bot caver (explorador de cuevas para encontrar diamantes, hierro, etc).
- Bot constructor (por ejemplo pasandole archivos con descripción de estructuras y materiales).
- Bot mula (sigue al jugador llevando el inventario lleno).
- Reconocimiento automático de estructuras (por ejemplo, dungeons o cuevas).
- Memoria persistente entre ejecuciones del bot.
- Visualización en consola de la grilla de exploración.
- Moderador de chat.

Anexo 1: Paquetes JSON aceptados por el bot

A continuación se describe el protocolo de mensajes JSON que permite controlar el bot desde el proceso en C++. Cada mensaje contiene una clave `action` que indica la operación a realizar.

Todos los paquetes se envían como una línea JSON serializada con salto de línea final (`\n`).

Acciones disponibles

`walk_path` Ejecuta una secuencia de pasos relativos desde un origen.

```
{
  "action": "walk_path",
  "path": [{ "x": 1, "y": 0, "z": 0 }, { "x": 0, "y": 0, "z": 1 }]
}
```

`move` Mueve al bot a coordenadas absolutas.

```
{
  "action": "move",
  "x": 123,
  "y": 64,
  "z": -58
}
```

`look` Hace que el bot mire hacia una orientación determinada.

```
{
  "action": "look",
  "yaw": 1.57,
  "pitch": 0.5
}
```

`dig` Minar un bloque en una posición dada.

```
{
  "action": "dig",
  "x": 123,
  "y": 63,
  "z": -58
}
```

Respuesta esperada (éxito):

```
{
  "status": "dug",
  "pos": { "x": 123, "y": 63, "z": -58 }
}
```

Respuesta esperada (error):

```
{
  "status": "error",
  "message": "Can't dig block or block missing"
}
```

`chat` Envía un mensaje al chat del servidor.

```
{
  "action": "chat",
  "message": "Chicken Jockey!"
}
```

Respuesta esperada:

```
{
  "status": "said",
  "message": "Chicken Jockey"!
}
```

`place` Coloca un bloque encima del especificado, usando un ítem del inventario.

```
{
  "action": "place",
  "x": 122,
  "y": 64,
  "z": -59,
  "item": "stone"
}
```

Respuesta esperada: No se envía respuesta.

`jump` Hace que el bot salte una vez.

```
{
  "action": "jump"
}
```

Respuesta esperada: No se envía respuesta.

`attack` Ataca a la entidad hostil más cercana.

```
{
  "action": "attack"
}
```

Respuesta esperada (éxito):

```
{
  "status": "ok",
  "message": "Attacking zombie"
}
```

Respuesta esperada (fallo):

```
{
  "status": "ok",
  "message": "No mob to attack"
}
```

`use` Usa el ítem en la mano (clic derecho).

```
{
  "action": "use"
}
```

Respuesta esperada: No se envía respuesta.

`collect` Recoge ítems sueltos en el mundo cercanos al bot.

```
{
  "action": "collect"
}
```

Respuesta esperada: No se envía respuesta.

`equip` Equipa un ítem del inventario en una ranura determinada.

```
{
  "action": "equip",
  "item": "iron_pickaxe",
  "slot": "hand"
}
```

Respuesta esperada:

```
{
  "status": "ok",
  "message": "Equipped iron_pickaxe in hand"
}
```

`inventory` Solicita el listado de ítems que tiene el bot.

```
{
  "action": "inventory"
}
```

Respuesta esperada:

```
{
  "type": "inventory",
  "items": [
    { "name": "stone", "count": 12 },
    { "name": "iron_pickaxe", "count": 1 }
  ]
}
```

`position` Solicita la posición actual del bot en el mundo.

```
{
  "action": "position"
}
```

Respuesta esperada:

```
{
  "status": "ok",
  "position": {
    "x": 123.5,
    "y": 64.0,
    "z": -58.7
  }
}
```

`entities` Devuelve una lista de entidades (jugadores, mobs, ítems) cercanas al bot.

```
{
  "action": "entities"
}
```

Respuesta esperada:

```
{
  "status": "ok",
  "entities": [
    { "name": "zombie", "type": "mob", "x": 125, "y": 64, "z": -60 },
    { "name": "player123", "type": "player", "x": 120, "y": 64, "z": -59 }
  ]
}
```

`find_block` Busca un bloque con el nombre indicado dentro de un radio determinado.

```
{
  "action": "find_block",
  "blockName": "stone",
  "maxDistance": 20
}
```

Respuesta esperada (éxito):

```
{
  "type": "found_block",
  "block": {
    "name": "stone",
    "position": { "x": 15, "y": 63, "z": -2 }
  }
}
```

Respuesta esperada (fallo):

```
{
  "type": "found_block",
  "message": "Not found"
}
```

`status` Devuelve el estado del bot (salud, comida, uso de ítem).

```
{
  "action": "status"
}
```

Respuesta esperada:

```
{
  "type": "status",
  "health": 20,
  "food": 20,
  "isUsingItem": false
}
```

`block_at` Consulta información del bloque en una posición.

```
{
  "action": "block_at",
  "x": 10,
  "y": 64,
  "z": -5
}
```

Respuesta esperada (éxito):

```
{
  "type": "block_at",
  "name": "dirt",
  "hardness": 0.5,
  "position": { "x": 10, "y": 64, "z": -5 }
}
```

Respuesta esperada (fallo):

```
{
  "type": "block_at",
  "message": "No block found"
}
```

`held_item` Devuelve el ítem actualmente en la mano del bot.

```
{
  "action": "held_item"
}
```

Respuesta esperada (tiene ítem):

```
{
  "type": "held_item",
  "name": "iron_pickaxe",
  "count": 1
}
```

Respuesta esperada (vacío):

```
{
  "type": "held_item",
  "message": "Nothing held"
}
```

Anexo 2: Glosario

A* (A estrella) Algoritmo de búsqueda heurística utilizado para encontrar el camino más corto entre dos puntos en un grafo. En este trabajo práctico, deben implementarlo en C++ para la navegación del bot.

Bot Entidad controlada por software dentro del mundo de Minecraft. En este proyecto, se trata de un agente controlado mediante código en C++ a través de una API en JavaScript.

Mineflayer Librería de Node.js que permite controlar bots en Minecraft. Utilizada como backend para conectar el mundo del juego con el proceso en C++.

IPC (Inter-Process Communication) Modelo de comunicación entre procesos. En este trabajo se usa para comunicar un proceso en C++ con uno en Node.js mediante flujos estándar (stdin/stdout).

JSON Formato de intercambio de datos ligero utilizado para transmitir información estructurada entre el bot en Node.js y el controlador en C++.

Node.js Entorno de ejecución para JavaScript del lado del servidor. Se utiliza para correr el bot de Minecraft mediante Mineflayer y gestionar la comunicación con el proceso en C++.

FSM (Finite State Machine) Modelo computacional basado en estados y transiciones. Se utiliza para organizar el comportamiento del bot, por ejemplo: buscar, caminar, minar, esperar.

Heurística Función utilizada en el algoritmo A* para estimar el costo restante hacia el objetivo. Influye en la eficiencia de la búsqueda.

Nodo Elemento de un grafo que representa una posición en el mapa del mundo de Minecraft. Usado en la implementación de A*.

Vecino Nodo adyacente a otro dentro de la representación del grafo, considerando movimientos válidos (caminar, saltar, caer).

Mapa Representación discreta del mundo 3D de Minecraft como una grilla que puede ser recorrida por el bot.

Grilla Modelo de discretización del mundo de Minecraft, donde cada bloque es considerado una celda con información como tipo, altura o si es transitable.

Camino óptimo Ruta más corta desde el origen hasta un destino según los criterios del algoritmo A*.

Exploración Acción de recorrer el mapa para obtener información de bloques, entidades u objetivos aún no conocidos.

Minado Tarea del bot que consiste en romper bloques específicos del entorno, por ejemplo, carbón, hierro, piedra.

API Conjunto de funciones ofrecidas por el entorno JavaScript que permite a los estudiantes controlar al bot desde el código C++.