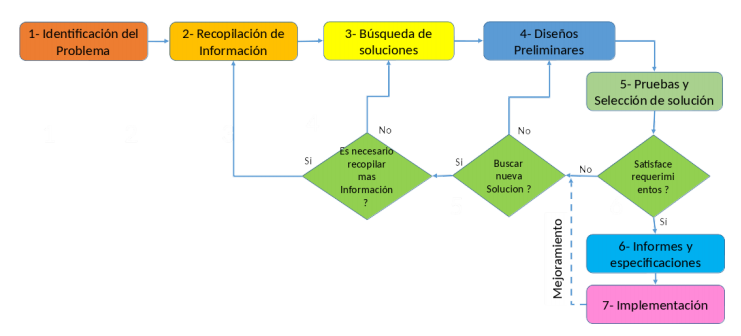
**Contexto problemático**

El juego “Minecraft” es un famoso juego tipo sandbox, uno de sus características más notables es capacidad de construcción con diferentes materiales que posee. Ahora bien, el jugador tiene un sistema de inventario con casillas de un tamaño de 4x9 que incluye una barra de acceso rápido (ultima fila). Los desarrolladores notaron que los jugadores tienen problemas al momento de construir usando bloques, además de que el algoritmo relacionado con la referencia de cada casilla del inventario hacia el contador interno funciona utilizando un algoritmo muy lento.

**Desarrollo Problemático**

Para resolver la situación anteriormente mencionada, se decide utilizar el reconocido Método de la Ingeniería para desarrollar una solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



**Paso 1. Identificación del Problema**

En esta primera instancia se reconocen de manera concreta las necesidades propias de la situación problemática, así como sus síntomas y condiciones bajo las cuales debe ser resuelta.

Identificación de necesidades y síntomas:

* Los usuarios de Minecraft desean un nuevo feature de acceso para facilitar la construcción en el juego.
* No hay un algoritmo eficiente que maneje el inventario.
* El algoritmo que se requiere que se mejore el consumo de RAM.
* Nuevo feature de modalidad de construcción en las barras de acceso rápido.
* Visualización de los nuevos cambios en el sistema (GUI).

Definición del problema:

La empresa Mojang AB. Requiere del desarrollo de un algoritmo eficiente para acceder a los bloques del sistema. Adicionalmente requieren que se desarrollen nuevas características de modalidad de acceso rápido y de construcción. Finalmente desean que estos cambios puedan ser visualizados intuitivamente en una interfaz gráfica.

**Paso 2. Recopilación de Información**

A partir de lo enunciado podemos detectar que el funcionamiento general del algoritmo que maneja el inventario consiste en:

“verificar el tipo de bloque que es usado para luego recorrer uno por uno el total de bloques en el sistema, y de esta manera detectar que, si se obtiene un bloque de piedra, este debe agruparse con los bloques de piedra que están presentes en el inventario.” (Icesi, 2019).

Este proceso hace que el videojuego tenga un alto consumo de memoria RAM.

Con el objetivo de tener claridad en los conceptos involucrados se hace una búsqueda de las definiciones precisas de los elementos utilizados por el juego y su propósito.

Definiciones:

*Inventario*

De forma general se define un inventario como la existencia o conteo de bienes almacenados destinados a realizar una actividad, de operación, compra, venta, alquiler, uso o transformación. (GestioPolis.com, 2002)

Fuente:

GestioPolis.com Experto. (2002, agosto 19). ¿Qué es inventario? Tipos, utilidad, contabilización y valuación. Recuperado de <https://www.gestiopolis.com/que-es-inventario-tipos-utilidad-contabilizacion-y-valuacion/>

*Matriz*

“Una matriz es un arreglo bidimensional de números (llamados entradas de la matriz) ordenados en filas (o renglones) y columnas, donde una fila es cada una de las líneas horizontales de la matriz y una columna es cada una de las líneas verticales.” (Wikimedia, 2019)

Fuente:

<https://es.wikipedia.org/wiki/Matriz_(matemáticas)#Definición>

**Paso 3. Búsqueda de soluciones**

Para este paso se necesita pensar en una solución creativa, por lo que se ha de considerar varias alternativas para desarrollar para cada necesidad del cliente.

Algoritmo para el manejo del inventario

Para este concepto, tenemos en primer lugar una matriz 4x9, que representa el inventario y la barra de acceso rápido. El enunciado nos dice que el programa tiene un contador interno de bloques que puede ser un arreglo 1xn, con n materiales (bloques). Infiero que luego de detectar dicho bloque, luego lo busca en cada una de las casillas del inventario para agruparlo finalmente con los de su mismo tipo. Ahora bien, asumiendo que cada bloque es un objeto que debe ser tomado como único, pero teniendo en común tipo de bloque y otras características que pueden dar la posición y estado de este en cualquier momento.

Alternativa 1

La primera alternativa consiste en agrupar los bloques en el ADT (Abstract Data Type) Stack, que es en síntesis una estructura linear basada en el principio LIFO (Last in First Out)[[1]](#footnote-1). Ahora bien, para hallar los bloques disponibles en el inventario es necesario un algoritmo de búsqueda simple que busque absolutamente todos los slots disponibles, por lo que se utilizaría un método ingenuo (Naive). Y en vez de un contador de bloques en el inventario tradicional, tendremos una tabla hash que tendrá el tipo de bloque como llave (key) y una lista enlazada de Stacks de tamaño máximo 64. De esta forma se agruparán los bloques que se reciban.

Alternativa 2

Esta consiste en agrupar los bloques en el ADT Queue, que es una estructura linear basada en el principio FIFO (First In First Out). Ahora bien, para hallar los bloques disponibles en el inventario es necesario un algoritmo de búsqueda simple que busque absolutamente todos los slots disponibles, por lo que se utilizaría un método ingenuo (Naive). Y en vez de un contador de bloques en el inventario tradicional, tendremos una tabla hash que tendrá el tipo de bloque como llave (key) y una lista enlazada de Queue de tamaño máximo 64. De esta forma se agruparán los bloques que se reciban.

Alternativa 3

La presente alternativa consiste en una aproximación ingenua al problema. Los bloques se agruparían al buscar la totalidad de los bloques y cada una de las casillas serían un arreglo de bloques con tamaño máximo de 64. Un algoritmo de búsqueda simple para encontrar el total de bloques en el inventario. Y en vez de un contador de bloques en el inventario tradicional, tendremos una tabla hash que tendrá el tipo de bloque como llave (key) y el número total de bloques de ese tipo como dato. De esta forma se agruparán los bloques que se reciban.

Modalidad de acceso rápido

Alternativa 1

Esta alternativa consiste que en el usuario elija una casilla del inventario y decida ponerla en la barra de acceso rápido, luego se organizaran todos los bloques del mismo tipo en la barra de acceso rápido

Alternativa 2

El usuario elige el tipo de bloque que desea en la barra de acceso rápido al hacer click sobre una casilla, luego se tomara el tipo del bloque y se dispondrán todos los de ese tipo en la barra.

Alternativa 3

El usuario escribe que tipo de bloque quiere disponer en la barra

Modalidad de Construcción

Se organizarán las barras de acceso rápido en orden alfabético dependiendo del material del bloque, luego al hacer click en la flecha (acción Siguiente), se cambiará a la siguiente barra de acceso rápido.

**Paso 4. Transición de las Ideas a los Diseños Preliminares**

En primer lugar, debemos descartar las ideas que no son factibles. Por esta razón se descarta la Alternativa 3 para el manejo del inventario debido a que esta opción consumiría mucha memoria para poder almacenar todos los datos y referencias que se necesitarían para poner en operación el algoritmo. Adicionalmente, se descarta la Alternativa 1 para la modalidad de acceso rápido, debido a que su similitud con la alternativa 2 es bastante evidente, sin embargo, requiere pasos adicionales que no son necesarios para el objetivo de este proyecto.

La revisión cuidadosa de las otras alternativas nos conduce a:

Manejo del inventario

Alternativa 1

* El ADT Stack es adecuado para la resolución del problema.
* Stack funciona con LIFO, pero no nos importa el orden de salida.
* Ocupa el menor espacio en memoria
* Tabla Hash simplifica los contadores de bloques

Alternativa 2

* El ADT Queue es adecuado para la resolución del problema
* Queue funciona con FIFO, pero no nos importa el orden de salida
* Referencias adicionales para principio FIFO.
* Ocupa ligeramente más espacio en memoria que la alternativa 1.
* Tabla Hash simplifica los contadores de bloques

Modalidad de acceso rápido

Alternativa 2

* Fácil implementación
* Requiere de visualización para poder elegir el bloque.
* Si funciona a base de clics, se necesitaría GUI.

Alternativa 3

* Fácil implementación
* No requiere visualización previa para elegir los bloques
* No necesita GUI.

**Paso 5. Evaluación y Selección de la Mejor Solución**

Criterios

Deben definirse los criterios que permitirán evaluar las alternativas de solución y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios se escogieron en este caso son los que enumeramos a continuación, separados por categoría. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen mayor importancia.

* Criterio A. Consumo de Recursos. Memoria RAM. La idea es optimizar el uso de recursos.
  + [3] Usa pocos recursos
  + [2] Consumo moderado
  + [1] Alto Consumo
* Criterio B. Facilidad en Implementación algorítmica. Se prefiere una implementación sencilla que las otras consideradas.
  + [3] Fácil Implementación
  + [2] Moderada Dificultad
  + [1] Alta Dificultad
* Criterio C. Compatibilidad para la resolución del problema. Que tan competente es el método a utilizar en la solución del problema.
  + [3] Es compatible en su totalidad
  + [2] Es compatible, pero tiene características adicionales innecesarias.
  + [1] No es compatible.
* Criterio D. Importancia para el usuario. Que tan importante es la implementación de la solución para el usuario.
  + [2]Importante
  + [1]Secundario

Evaluación

Evaluando los criterios anteriores en las alternativas que quedan, obtenemos la siguiente tabla:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| Alternativa 1. Inventario | Pocos Recursos. 3 | Dificultad moderada. 2 | Totalmente compatible. 3 | Importante para el usuario. 2 | 10 |
| Alternativa 2. Inventario | Poco Consumo. 3 | Moderado. 2 | Tiene referencias adicionales que no se usan. 2 | Importante. 2 | 9 |
| Alternativa 2. Acceso rápido | 3 | 3 | 3 | 2 | 11 |
| Alternativa 3. Acceso rápido | 3 | 3 | 3 | No es como el juego funciona normalmente. 1 | 10 |

Selección

De acuerdo a la evaluación anterior se debe seleccionar la alternativa 1 para el manejo de inventario y la alternativa 2 para la barra de acceso rápido. Esto de acuerdo a la tabla tienen mayor puntuación de acuerdo con los criterios definidos. Es importante recalcar que en el criterio D para la modalidad se asume que los usuarios prefieren tener la misma dinámica con la que se juega normalmente, es decir, arrastrando o haciéndole click a los bloques gráficamente.

**Paso 6. Preparación de Informes y Especificaciones**

Especificaciones de los problemas (en términos de entrada/salida)

Problema: Implementación del ADT Stacks de forma genérica.

Entradas:

Salida:

Problema: Organizar los bloques en stacks

Entradas:

Salidas:

Consideraciones

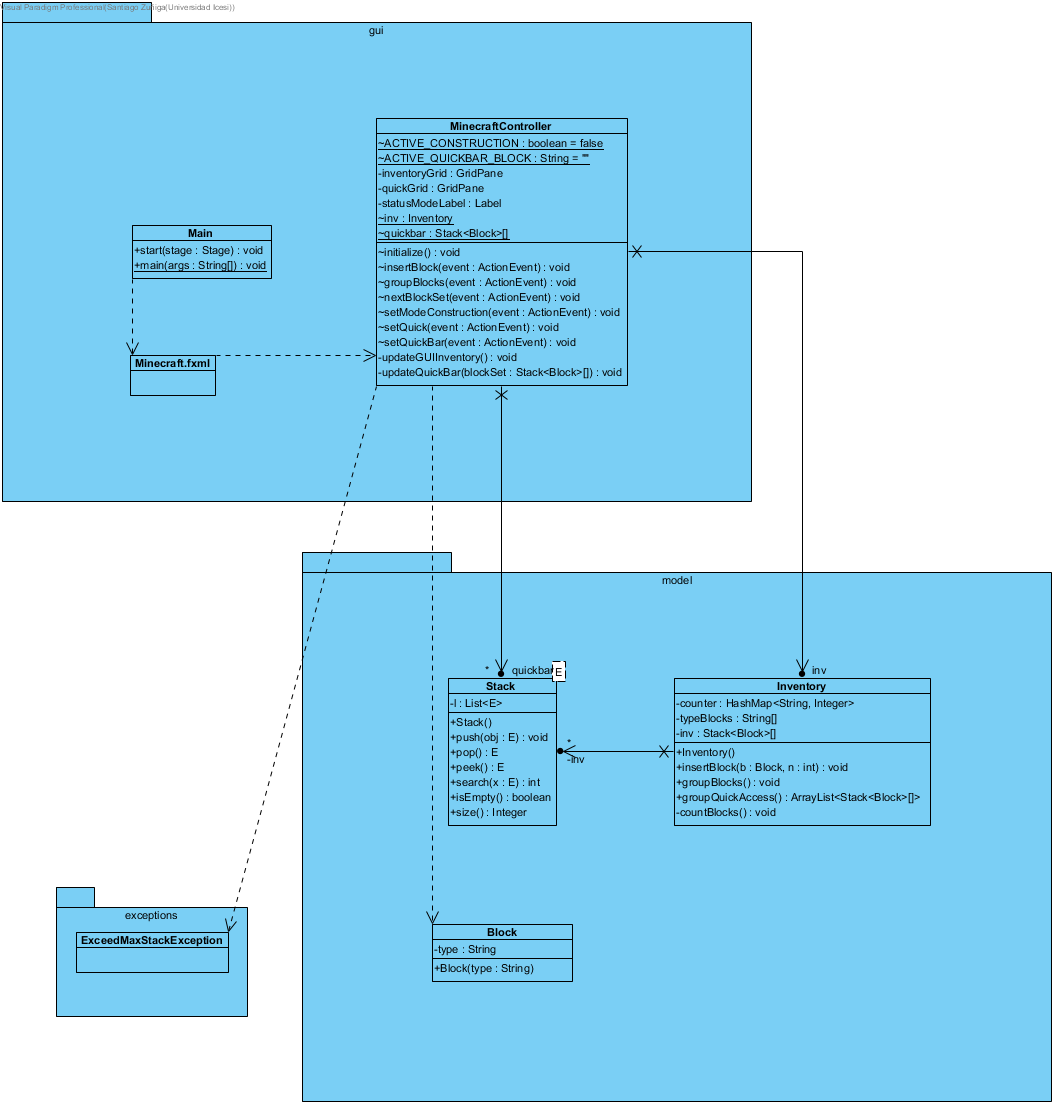
Se deben tener en cuenta los siguientes casos:

1. Por casilla no deben haber más de 64 bloques.
2. Solo se pueden agrupar bloques del mismo tipo
3. No hay ningún bloque en el inventario.
4. Agregar bloques

Especificaciones de las estructuras de datos

|  |
| --- |
| **ADT Stack** |
| Es una estructura lineal de datos que utiliza operaciones de inserción y remoción al inicio. Basado en el principio LIFO (Last in First Out, ultimo que entra, primero que sale). |
| Invariante: |
| Operaciones: |
| Nombre: Push; Entrada:Objeto; Salida: Inserta el objeto al Stack.  Nombre: Pop; Salida: Remueve el objeto del Stack y lo muestra.  Nombre: Peek; Salida: Muestra el primer objeto.  Nombre: Search; Entrada: Valor o objeto; Salida: índice o posición.  Nombre: isEmpty(); Salida: boolean |

Diagrama de Clases de la Solución



**Paso 7. Implementación**

Este proyecto fue implementado en Java y trabajado con control de versiones en GitHub y puede ser revisado en: <https://github.com/Legendary-Overlord/Minelab> .

Lista de tareas a implementar:

1. Agrupar los bloques en Stacks
2. Realizar operaciones en el contador interno
3. Insertar bloques al inventario
4. Agrupar bloques de modalidad rápida

|  |  |
| --- | --- |
| Requirement | 1 |
| Summary | Insert Block |
| Input | Block, number of blocks to be inserted |
| Output | Block inserted into inventory |
| Requirement | 2 |
| Summary | countBlocks |
| Input |  |
| Output | updated the internal block counter |
| Requirement | 3 |
| Summary | groupQuickAccess, groups the blocks for the quick access bar |
| Input |  |
| Output | Set of Blocks |
| Requirement | 4 |
| Summary | groupBlocks, groups all the blocks and organizes them |
| Input |  |
| Output | All blocks grouped and organized in inventory. |
| Requirement | 5 |

Construcción

Escritura del código en un lenguaje de programación (Java en este caso)

Requerimiento 1:

public void insertBlock(Block b,int n) {

Stack<Block> c = new Stack<>();

for(int i=0;i<n;i++)

c.push(b);

for (Stack<Block> s:inv) {

if(s.isEmpty()) {

s=c;

break;

}

}

}

Requerimiento 2:

private void countBlocks() {

counter.clear();

for(Stack<Block> s : inv) {

if(!s.isEmpty()) {

String com = s.peek().getType();

if (counter.containsKey(com)){

counter.replace(com, (counter.get(com)+s.size()));

}else

counter.put(com, s.size());

}

}

}

Requerimiento 3:

public ArrayList<Stack<Block>[]> groupQuickAccess() {

countBlocks();

ArrayList<Stack<Block>[]> qt = new ArrayList<>();

for(String st : typeBlocks) {

int n = counter.get(st);

Stack<Block>[] s = new Stack[(int) Math.ceil(n/64)];

for(Stack<Block> b :s) {

if(b.isEmpty()) {

if(n>=64) {

for(int i =0; i<63;i++)

b.push(new Block("st"));

n-=64;

}else if(n>0) {

for(int i=0;i<n;i++)

b.push(new Block("st"));

break;

}else

break;

}

}

qt.add(s);

}

return qt;

}

Requerimiento 4:

public void groupBlocks() {

countBlocks();

Stack<Block>[] s = new Stack[4\*9];

for(String st : typeBlocks) {

int n = counter.get(st);

for(Stack<Block> b :s) {

if(b.isEmpty()) {

if(n>=64) {

for(int i =0; i<63;i++)

b.push(new Block("st"));

n-=64;

}else if(n>0) {

for(int i=0;i<n;i++)

b.push(new Block("st"));

break;

}else

break;

}

}

}

inv=s;

}

1. BSC. (2019). Stack ADT. Recuperado de http://btechsmartclass.com/data\_structures/stack-adt.html [↑](#footnote-ref-1)