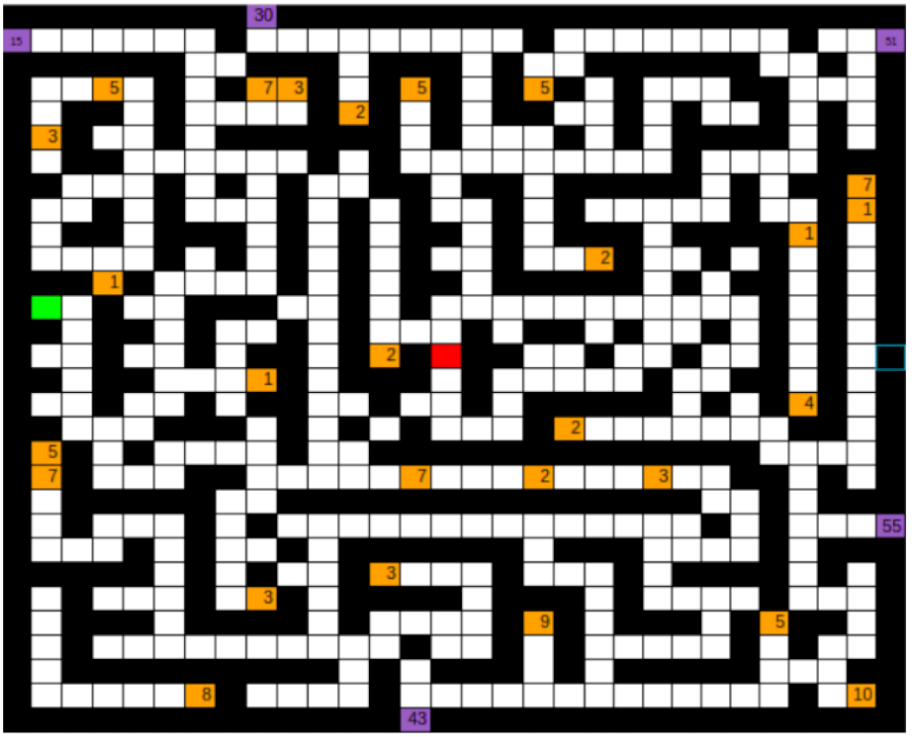
Laberinto en consola_



Juego de laberinto en consola

Manual técnico

Versión: 0100

Fecha: 28/02/2022

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Desarrollador: Cristian Alejandro Vásquez Escobar, 202131936

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Requerimientos para la ejecución del juego

- Una computadora.
- Se necesita tener instalada la máquina virtual de java. Esto se puede hacer desde su página y para cualquier sistema operativo: Descargar Java.
- Alguna terminal para poder interpretar comandos, si se encuentra en windows con el cmd o el windows powershell será suficiente.

Elaboración del programa

Herramientas

- Para la realización del programa “laberinto en caída” se utilizó el lenguaje de java en su versión 17.0.2.
- El ambiente de desarrollo utilizado es el editor de texto Visual Studio Code en su versión 1.64.2.
- Fue realizado desde una laptop hp 15-db0 con windows 10.
- Para una mejor organización se utilizó el sistema de versiones git con github.
- Para la creación del archivo laberinto.jar, se utilizó la consola de comandos git bash.

Diseño

Para facilitar el desarrollo del programa se crearon un total de 10 clases.

Helpers

En esta ubiqué dos funcionalidades las cuales se verían repetidas veces a lo largo del programa.

```java
public class Helpers {

    public static void clear() {
        // System.out.print("\033[H\033[2J");
        // System.out.flush();
    }

    public static int[] localizacionRandom(int _filas, int _columnas) {

        int randomX = (int) Math.floor(Math.random() * ((_columnas - 1) - 0 + 1) + 0);
        int randomY = (int) Math.floor(Math.random() * ((_filas - 1) - 0 + 1) + 0);
        int[] location = { randomY, randomX };

        return location;

    }

}
```

Clear: Era una función que limpiaba la consola, pero al ver que esto sólo funcionaba en ciertas consolas, decidí que lo mejor sería removerlo.
localizacionRandom: Es una función que según las filas y las columnas del mapa, devolvería una posición aleatoria.

Jugador

Esta es una clase para un jugador, donde se definen sus atributos, de los cuales algunos servirán para los reportes.

```
public class Jugador {

    int[] posicion;
    int oroRecolectado;
    int movimientos = 0;
    String estado = "En juego";
    char caracter = 'J';

    public Jugador(int[] _posicion, int _oroRecolectado) {
        this.posicion = _posicion;
        this.oroRecolectado = _oroRecolectado;
    }

    public void getOroRecolectado() {
        System.out.println(
"La cantidad total de oro recolectada es de " + this.oroRecolectado
);
    }

    public void getMovimientos(String _estado) {
        System.out.println(
"La cantidad de movimientos realizados para " + _estado + " es de: "
 + this.movimientos);

    }

}
```

getOroRecolectado: Imprime en consola el oro recolectado.

getMovimientos: Imprime en consola la cantidad de movimientos del jugador realizados.

Oro

Es una clase para cada casilla de oro, la cuál tiene la cantidad de oro que almacena, su posición y un booleano para saber si esta levantada.

```
public class Oro {

    public int cantidadOro;
    public int[] posicion;
    public boolean estaLevantada = false;

    public Oro(int _cantidadOro, int[] _posicion) {

        this.cantidadOro = _cantidadOro;
        this.posicion = _posicion;

    }

}
```

Salida

Una clase para las casillas de salida, tiene un método para imprimir en pantalla el oro que la casilla necesita para poder ser utilizada.

```
public class Salida {

    public int oroNecesario;
    public int[] posicion;

    public Salida(int _oroNecesario, int[] _posicion) {

        this.oroNecesario = _oroNecesario;
        this.posicion = _posicion;

    }

    public void getOroNecesario() {

        System.out.println(
"El oro necesario para esta salida es de : " + this.
oroNecesario);

    }

}
```

Mapa

Es una clase para todos los mapas que se puedan crear, con su método constructor.

```
public class Mapa {

    public String nombre;
    public char[][] matriz;
    public Salida[] salidas;
    public Oro[] listaOro;

    public int columnas;
    public int filas;
    public boolean estaJuegoTerminado = false;
    public int vecesJugado = 0;
    public int vecesGanadas = 0;
    public int vecesPerdidas = 0;

    public Mapa(String _nombre, char[][] _matriz, Salida[] _salidas
, Oro[] _listaOro) {

        this.nombre = _nombre;
        this.matriz = _matriz;
        this.salidas = _salidas;
        this.listaOro = _listaOro;

        this.columnas = this.matriz[0].length;
        this.filas = this.matriz.length;

    }

}
```

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Laberinto

```
public class Laberinto {

    static Mapa[] LISTA_MAPAS = new Mapa[50];
    static int indiceUltimoMapa = 0;

    static Jugador[] LISTA_JUGADORES = new Jugador[50];
    static int indiceUltimoJugador = 0;

    static int totalPartidasGanadas = 0;
    static int totalPartidas = 0;
    static int totalOroPartidas = 0;
    static int totalMovimientos = 0;
    static int totalVecesAtrapado = 0;

    public static void main(String[] args) {
        Helpers.clear();

        char[][] matriz = {
                { '#', '#', '#', '#', '#', '#', '#', '#', 'S', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
                        '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
                { 'S', 'O', 'O', 'O', 'O', 'O', 'O', '#', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '#', 'O', 'O',
                        'O', 'O', 'O', 'O', 'O', 'O', 'O', '#', 'O', 'O', 'S' },
```

El laberinto es la main class y aquí se crean unas variables globales accesibles para todas las clases, además aquí también se inicializan los mapas por defecto.

```
        Salida[] salidas = {
                new Salida(30, new int[] { 0, 8 }),
                new Salida(15, new int[] { 1, 0 }),
                new Salida(51, new int[] { 1, 29 }),
                new Salida(51, new int[] { 21, 29 }),
                new Salida(51, new int[] { 29, 13 }),
        };

        Oro[] listaOro = {
                new Oro(5, new int[] { 3, 3 }),
                new Oro(7, new int[] { 3, 8 }),
                new Oro(3, new int[] { 3, 9 }),
                new Oro(5, new int[] { 3, 13 }),
                new Oro(5, new int[] { 3, 17 }),
```

```
                new Oro(8, new int[] { 28, 6 }),
                new Oro(10, new int[] { 28, 28 }),
        };

        Mapa mapaPorDefecto = new Mapa("Mapa por defecto", matriz, salidas, listaOro);
        LISTA_MAPAS[indiceUltimoMapa] = mapaPorDefecto;
        indiceUltimoMapa++;
```

También se inicia un arreglo con todas las casillas de salida y de oro para luego instanciar el mapa.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```
        menuPrincipal(false);

    }

    public static void menuPrincipal(boolean deNuevo) {

        Helpers.clear();

        if (deNuevo) {
            System.out.println("El número escrito no es una opción valida, por favor vuelva a intentar");
        }

        Scanner scanner = new Scanner(System.in);
        final String[] MENU_PRINCIPAL_OPCIONES = new String[] { "JUGAR", "CREAR", "REPORTES", "VISUALIZAR"
, "SALIR" };
        System.out.println("————————————MENU--PRINCIPAL————————————");
        System.out.println("Por favor eliga una de las siguientes opciones:");
        for (int i = 0; i < MENU_PRINCIPAL_OPCIONES.length; i++) {

            System.out.println((i + 1) + ". " + MENU_PRINCIPAL_OPCIONES[i]);

        }

        int OPCION_MENU_PRINCIPAL = scanner.nextInt();
        switchOpcionesMenuPrincipal(OPCION_MENU_PRINCIPAL);

    }
```

El menú principal se ejecuta y salen las opciones de este.

Juego

```java
public void jugar(Mapa _mapa) {
        comandosFallidos = 0;
        Helpers.clear();
        _mapa.vecesJugado++;
        System.out.println("Empezando juego con el mapa: " + _mapa.nombre);

        int oroRecolectado = 0;

        Jugador jugador = new Jugador(localizarJugadorAleatoriamente(_mapa),
                oroRecolectado);
        _mapa.matriz[jugador.posicion[0]][jugador.posicion[1]] = 'J';
        scanner.nextLine();

        // bot
        boolean esta4Jugador = false;
        boolean estaMismaPosicion = true;
        int[] posicionBot = localizarJugadorAleatoriamente(_mapa);

        if (_mapa.columnas > 5 && _mapa.filas > 5) {
            while (!esta4Jugador) {
                posicionBot = localizarJugadorAleatoriamente(_mapa);
                if (((posicionBot[1] - jugador.posicion[1]) > 4) || ((jugador.posicion[1] - posicionBot[1]) > 4)
                        && ((posicionBot[0] - jugador.posicion[0]) > 4)
                        || ((jugador.posicion[0] - posicionBot[0]) > 4)) {
                    esta4Jugador = true;
                }
            }
        } else {
            do {
                if (Arrays.equals(posicionBot, jugador.posicion)) {
                    posicionBot = localizarJugadorAleatoriamente(_mapa);
                } else {
                    estaMismaPosicion = false;
                }
            } while (estaMismaPosicion);
        }
        Bot bot = new Bot(posicionBot);
        bot.limpiarMatriz(_mapa);

        _mapa.matriz[bot.posicion[0]][bot.posicion[1]] = 'B';

        while (!_mapa.estaJuegoTerminado) {
            turno(_mapa, jugador);
            if (!_mapa.estaJuegoTerminado)
                turnoBot(bot, _mapa, jugador);
        }

        _mapa.estaJuegoTerminado = false;

        Reportes.reportesFinalizarJuego(Laberinto.LISTA_JUGADORES[Laberinto.indiceUltimoJugador - 1], bot);

    }
```

Se crea una instancia del jugador y del bot con una posición aleatoria. Luego se crea el ciclo del juego, el cual termina hasta que se dé por finalizado el juego con la variable “estaJuegoTerminado”.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```
public void turno(Mapa _mapa, Jugador _jugador) {
        System.out.println(
"Su turno, por favor ejecute uno de los comandos descritos en el manual de usuario");
        String opcion = scanner.nextLine();
        switchComandos(opcion, _mapa, _jugador);
    }

    public void turnoBot(Bot _bot, Mapa _mapa, Jugador _jugador) {
        System.out.println("Turno del bot.");

        if (_bot.jugadorALaVista) {
            _bot.simular(_mapa, _jugador);
        } else {
            if (_bot.mirar(_mapa, _jugador)) {
                _bot.jugadorALaVista = true;
                System.out.println(
"El bot detectó que te encuentras cerca de él !Ten cuidado!");
            } else {
                int numeroAleatorio = (int) Math.floor(Math.random() * (4 - 1 + 1) + 1);
                _bot.mover(numeroAleatorio, _mapa, false);
            }
        }

    }
```

Hay 2 métodos llamados turnos. El turno del bot decide qué es lo que el bot hará en su turno, el turno del jugador le pide al usuario qué comando desea ejecutar con el método switch comando.

```
public void levantar(Mapa _mapa, Jugador _jugador) {

    int indiceOroLevantado = esOro(_jugador.posicion, _mapa);

    if (indiceOroLevantado == -1) {
        System.out.println("No se encuentra en una casilla de oro, pierde el turno");
    } else {
        Oro casillaOro = _mapa.listaOro[indiceOroLevantado];
        casillaOro.estaLevantada = true;

        _jugador.oroRecolectado += casillaOro.cantidadOro;

        System.out.println("La accion se realizo correctamente");
        System.out.println("Usted ha levantado " + casillaOro.cantidadOro + " de oro");
        _jugador.getOroRecolectado();
    }

}
```

Este método logra poder levantar oro si es que el jugador está en una casilla de oro.

```
   public void oroRequerido(Mapa _mapa, Jugador _jugador) {

       int indiceSalida = esSalida(_jugador.posicion, _mapa);

       if (indiceSalida == -1) {
           System.out.println("No se encuentra en una casilla de salida, pierde el turno."
);
       } else {
           Salida salida = _mapa.salidas[indiceSalida];

           salida.getOroNecesario();
       }

   }
```

Este comando imprime el oro necesario si es que el jugador se encuentra en una casilla de salida.

```
public void terminarJuegoComandos(Mapa _mapa, Jugador _jugador) {
        _mapa.estaJuegoTerminado = true;
        limpiarMatriz(_mapa, _jugador);
        Laberinto.LISTA_JUGADORES[Laberinto.indiceUltimoJugador] = _jugador;
        Laberinto.indiceUltimoJugador++;
        _jugador.estado = "perder por escribir comandos erroneos";
        for (int i = 0; i < _mapa.listaOro.length; i++) {
            _mapa.listaOro[i].estaLevantada = false;
            _mapa.matriz[_mapa.listaOro[i].posicion[0]][_mapa.listaOro[i].posicion[1]] =
'G';
        }

        _mapa.vecesPerdidas++;
        Laberinto.totalPartidas++;
        Laberinto.totalOroPartidas += _jugador.oroRecolectado;
        Laberinto.totalMovimientos += _jugador.movimientos;
    }
```

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```java
public void mirar5x5(Mapa _mapa, Jugador _jugador) {

    if ((_mapa.columnas ≤ 5) && (_mapa.filas ≤ 5)) {
        mirar(_mapa);
    } else {
        int rango[][] = {
                { -2, 2 }, // rango en y
                { -2, 2 } // rango en x
        };

        final int sumaRangoPosicionX1 = rango[1][0] + _jugador.posicion[1];
        final int sumaRangoPosicionX2 = rango[1][1] + _jugador.posicion[1];
        if (sumaRangoPosicionX1 < 0) {
            rango[1][0] -= sumaRangoPosicionX1;
            rango[1][1] -= sumaRangoPosicionX1;
        } else if (sumaRangoPosicionX2 > (_mapa.columnas - 1)) {
            rango[1][0] -= sumaRangoPosicionX2 - (_mapa.columnas - 1);
            rango[1][1] = Math.abs(sumaRangoPosicionX2 - rango[1][1] - (_mapa.columnas - 1));
        }

        final int sumaRangoPosicionY1 = rango[0][0] + _jugador.posicion[0];
        final int sumaRangoPosicionY2 = rango[0][1] + _jugador.posicion[0];
        if (sumaRangoPosicionY1 < 0) {
            rango[0][0] -= sumaRangoPosicionY1;
            rango[0][1] -= sumaRangoPosicionY1;
        } else if (sumaRangoPosicionY2 > (_mapa.filas - 1)) {
            rango[0][0] -= sumaRangoPosicionY2 - (_mapa.filas - 1);
            rango[0][1] = Math.abs(sumaRangoPosicionY2 - rango[0][1] - (_mapa.filas - 1));

        }

        imprimir5x5(rango, _mapa.matriz, _jugador.posicion);
    }

}

public void imprimir5x5(int[][] _rangos, char[][] _matriz, int[] _posicionJugador) {
    int[] rangoY = _rangos[0];
    int[] rangoX = _rangos[1];

    for (int i = rangoY[0]; i ≤ rangoY[1]; i++) {
        String fila = "";
        for (int j = rangoX[0]; j ≤ rangoX[1]; j++) {
            int y = _posicionJugador[0] + i;
            int x = _posicionJugador[1] + j;

            fila += _matriz[y][x] + " ";
        }
        System.out.println(fila);
    }

}
```

Con estos dos métodos imprimen en consola una visión de 5x5 del jugador.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Bot

```java
public class Bot {

    boolean jugadorALaVista = false;
    int[] posicion;
    int numeroTurno = 1;
    Juego juego = new Juego();
    char caracter = 'B';
    int movimientos = 0;
    int vecesEnVision = 0;

    public Bot(int[] _posicion) {

        this.posicion = _posicion;

    }

    public void getMovimientos() {
        System.out.println(
                "El total de movimientos realizados por el bot para atrapar al jugador es de: " + this.movimientos);
    }

    public void getVecesEnVision() {
        System.out.println(
                "El total de veces que el jugador estuvo en visión del bot es de: " + this.vecesEnVision);
    }
```

Una clase para instanciar bots, con atributos serviciales para los reportes y diferentes métodos para la funcionalidad de este

getMovimientos: Imprime la cantidad de movimientos que el bot ha realizado desde el inicio de la partida.

getVecesEnVicion: Imprime las veces que el bot ha tenido al jugador en su visión.

```java
public void mover(int _direccion, Mapa _mapa, boolean _estaSimulacion) {
        switch (_direccion) {
            case 1:
                // mover norte
                if ((this.posicion[0] - 1) < 0) {
                    if (!_estaSimulacion) {
                        System.out.println(
                                "El bot intentó salir de los límites del laberinto, movimiento denegado.");
                    }
                } else if (_mapa.matriz[this.posicion[0] - 1][this.posicion[1]] == '#') {
                    if (!_estaSimulacion) {
                        System.out.println("El bot intentó moverse a una pared, movimiento denegado");

                    }
                } else {
                    this.posicion[0]--;
                    this.limpiarMatriz(_mapa);
                    _mapa.matriz[this.posicion[0]][this.posicion[1]] = this.caracter;
                    if (!_estaSimulacion) {
                        System.out.println("El bot se movió hacia el norte");
                    }
                    this.movimientos++;
                }
                break;
            case 2:
```

mover: El método recibe la dirección a dónde se quiere mover el bot, si es hacia arriba su posición en y se disminuye en 1, si es hacia abajo, se incrementa en 1 y así sucesivamente. Las direcciones están representadas en números: 1 para el norte, 2 para el sur, 3 para el oeste y 4 para el este.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```
public double distancia(Jugador _jugador) {

    int[] coordenadaJugador = { _jugador.posicion[0] - this.posicion[0], _jugador.posicion[1]
- this.posicion[1] };

    double distancia = Math.sqrt(Math.pow(coordenadaJugador[0], 2) + Math.pow
(coordenadaJugador[1], 2));

    return distancia;

}

public void simular(Mapa _mapa, Jugador _jugador) {

    int[] posicionOriginal = this.posicion.clone();

    this.mover(1, _mapa, true);
    double distanciaNorte = distancia(_jugador);
    if (!Arrays.equals(posicionOriginal, this.posicion))
        this.movimientos--;
    this.posicion = posicionOriginal.clone();
    this.limpiarMatriz(_mapa);
    _mapa.matriz[this.posicion[0]][this.posicion[1]] = this.caracter;
```

simular: Este método simula un movimiento del bot para todas las direcciones y con el método **distancia** encuentra cuál de estos movimientos acerca más al bot al jugador y lo ejecuta.

```
public void terminarJuego(Mapa _mapa, Jugador _jugador) {

    _mapa.estaJuegoTerminado = true;
    this.limpiarMatriz(_mapa);
    Laberinto.LISTA_JUGADORES[Laberinto.indiceUltimoJugador] = _jugador;
    Laberinto.indiceUltimoJugador++;
    _jugador.estado = "perder";
    for (int i = 0; i < _mapa.listaOro.length; i++) {
        _mapa.listaOro[i].estaLevantada = false;
        _mapa.matriz[_mapa.listaOro[i].posicion[0]][_mapa.listaOro[i].posicion[1]] = 'G';
    }

    _mapa.vecesPerdidas++;
    Laberinto.totalVecesAtrapado++;
    Laberinto.totalPartidas++;
    Laberinto.totalOroPartidas += _jugador.oroRecolectado;
    Laberinto.totalMovimientos += _jugador.movimientos;

}
```

El método terminar termina el ciclo principal de turnos dandole un valor de true a la variable del mapa “estaJuegoTerminada”. Luego limpia el mata y efectúa algunas acciones para los reportes generales y que se pueda volver a jugar el mismo mapa.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```
public boolean jugadorALaVista(int[][] _rangos, char[][] _matriz, int[]
_posicionJugador) {
        int[] rangoY = _rangos[0];
        int[] rangoX = _rangos[1];

        for (int i = rangoY[0]; i ≤ rangoY[1]; i++) {
            for (int j = rangoX[0]; j ≤ rangoX[1]; j++) {
                int y = this.posicion[0] + i;
                int x = this.posicion[1] + j;

                if (Arrays.equals(new int[] { y, x }, _posicionJugador)) {
                    this.vecesEnVision++;
                    return true;
                }

            }
        }

        return false;

    }
```

```
public boolean mirar(Mapa _mapa, Jugador _jugador) {

        if ((_mapa.columnas ≤ 5) && (_mapa.filas ≤ 5)) {
            this.vecesEnVision++;
            return true;
        }

        int rango[][] = {
                { -2, 2 }, // rango en y
                { -2, 2 } // rango en x
        };

        final int sumaRangoPosicionX1 = rango[1][0] + this.posicion[1];
        final int sumaRangoPosicionX2 = rango[1][1] + this.posicion[1];
        if (sumaRangoPosicionX1 < 0) {
            rango[1][0] -= sumaRangoPosicionX1;
            rango[1][1] -= sumaRangoPosicionX1;
        } else if (sumaRangoPosicionX2 > (_mapa.columnas - 1)) {
            rango[1][0] -= sumaRangoPosicionX2 - (_mapa.columnas - 1);
            rango[1][1] = Math.abs(sumaRangoPosicionX2 - rango[1][1] - (_mapa.columnas - 1));
        }

        final int sumaRangoPosicionY1 = rango[0][0] + this.posicion[0];
        final int sumaRangoPosicionY2 = rango[0][1] + this.posicion[0];
        if (sumaRangoPosicionY1 < 0) {
            rango[0][0] -= sumaRangoPosicionY1;
            rango[0][1] -= sumaRangoPosicionY1;
        } else if (sumaRangoPosicionY2 > (_mapa.filas - 1)) {
            rango[0][0] -= sumaRangoPosicionY2 - (_mapa.filas - 1);
            rango[0][1] = Math.abs(sumaRangoPosicionY2 - rango[0][1] - (_mapa.filas - 1));

        }

        return jugadorALaVista(rango, _mapa.matriz, _jugador.posicion);

    }
```

El método **mirar** crea un rango para que el bot luego con el método jugadorALaVista verifique si el jugador se encuentra cerca de él.