MANUAL TECNICO APP MOVIL POR:

Erick Daniel Morales Xicara

Carné: 201930699

Manual Técnico

El software puede ser utilizado en cualquier sistema operativo ya que se utilizó java, también se puede ejecutar en la mayoría de computadoras que tengan java 11 o superior. Se utilizó el JDK 17 de java, y el IDE IntelliJ. A su vez se utilizó el Sdk 33 para el soporte de Android.

El software consiste en el manejo de información para la creación de mundos, enviados en formato json, y pasandolos a formato xml donde el cual es capaz de dibujar un mundo basándose en las posiciones definidas en dicha información enviada el servidor, en este caso se utilizo lenguaje java, un poco de html para el formato xml, y kotlin para el cliente del software.

El software a su vez contiene varias clases y manejadores que permiten iniciar, ejecutar y realizar las peticiones en el sistema antes mencionado.

Cliente:

Paquete Models; Este paquete contiene todas las implementaciones de modelos que servirán para presentar la información que se envía desde el cliente, por ende, el manejo de la información de los reportes.

Board Model: solo contiene la clase constructor que se utilizó para la creación de la matriz que servirá para pintar el tablero.

Construcción Matriz: Esta clase solo tiene un método para crear la matriz, e instancio todas las variables estáticas (java).

hacer matriz: este método recibe un mundo (el que envia el servidor) donde se crea la matriz nueva de board, donde se utiliza el array de box, targets y posición del jugador para setear el array de boards y ver si la casilla es caja,target o jugador para después pintarla.

Errores A: Esta clase tiene dos métodos, ya que es una activity y es el encargado de hacer la tabla para mostrar los reportes de errores.

On Create: este crea el activity y solicita el response que se envia desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer el arreglo de errores que recibe y así colocarlos en la tabla.

Error Mov CLient: este solo tiene una clase constructor para manejar los errores que se reciben al ingresar los movimientos del tablero

Names A: Esta clase tiene dos métodos, ya que es una activity y es el encargado de hacer la tabla para mostrar solo los nombres de los mundos.

On Create: este crea el activity y solicita el response que se envia desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer el arreglo de nombre de mundos que recibe y así colocarlos en la tabla.

Operaciones Mov Report: esta clase solo tiene la clase constructor que se utiliza para generar la información del reporte de los movimientos

Point: esta clase solo tiene el constructor que sirve para jalar los puntos que son definidos en la matriz de board

Report Mov: esta clase solo tiene el constructor para ver el reporte de la cantidad de movimientos que hay en el cliente.

Report Mov Layout: Esta clase tiene dos métodos, ya que es una activity y es el encargado de hacer la tabla para mostrar los reportes de movimientos, solamente el movimiento y la cantidad de veces que se encuentra

On Create: este crea el activity y solicita el response que se envia desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer el arreglo de los movimientos.

Report Mov Layout: Esta clase tiene dos métodos, ya que es una activity y es el encargado de hacer la tabla para mostrar los reportes de los signos de operación.

On Create: este crea el activity y solicita el response que se envía desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer de operadores

Paquete Objects: este paquete tiene los objetos utilizados para ver los movimientos que se harán en el tablero.

Motion: esta clase solo tiene el constructor para crear el objeto de movimiento.

Motion Arr: esta clase solo tiene el constructor para crear el objeto de todos los movimientos, que se muestran en el reporte

Token: clase utilizada para poder reportar los tokens, que se utilizan en el analizador flex y jcup.

Paquete Objects World: aqui estan los objetos utilizados para la creación de los mundos, con la información que regresa del servidor:
Clase A Error, Array Word, Board, Box, Config World, Player, Response, Target, World.

Parser Mov: esta clase contiene las clases generadas por el archivo flexMov.flex, y cupMov.Cup

Las clases siguientes: LexerMov, ParserMov, ParserMovSym

Parser Xml: esta clase contiene las clases generadas por el archivo lexXmlflex, y cupXml.Cup

Las clases siguientes: LexXml, ParserXml, ParserXmlSym

Y las clases

Fina Report Mov: esta clase contiene el activity para mostrar la tabla de los movimientos finales que se harán.

On Create: este crea el activity y solicita el response que se envía desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer el array de movimientos finales.

Layaout Err Mov: esta clase contiene el activity para mostrar la tabla de los errores en los movimientos que se harán.

On Create: este crea el activity y solicita el response que se envía desde el activity anterior, y lo pasa a una variable local, y llama al setTable();

set Table: este método es el encargado de recopilar los encabezados y datos de la tabla, donde llama a los textView para setear sus valores, y luego usa el for each para recorrer el array de los errores de los movimientos .

Paint Panel: esta clase es la que se encarga de pintar el tablero según las especificaciones de la matriz creada.

OnDraw: este método manda a crear la matriz, luego esa matriz la recorre deprimero las filas y luego las columnas, verifica que si la matriz no se ha pintado entonces manda a llamar al canvas donde le pasa el color, y la posición para pintar el tablero, y dependiendo si es box, target o jugador manda a llamar el método donde ya verifica que color tiene especificado en sus atributos , seguidamente de ver si es BRICK o HALL para pasarle el color.

paintUndefinided: este método es el encargado de verificar el color que tiene definido, y seguidamente pinta el cuadro con las especificaciones dadas de su posicion y asi seguidamente con:
paintBox,Paint Target, paintTarget,paintPlayer,paintBrick, paintHall.

Tablero Juego: Esta clase tiene los métodos para cuando ya se tiene la respuesta del cliente, para ver donde tiene que ir según la respuesta.

OnCreate: este método es el encargado de después de parsear el la respuesta del cliente ver que tipo de repuestos es y mandarlo al siguiente activity.

Compiler: este método llama al parser del los movimientos, verifica que si no hay errores, manda a llamar al método cambiar Negativos y añade a lista de movimientos ya los movimientos finales después de todos sus cambios.

Cambiar Negativos: este método verifica si una operación del parser, resultó negativa, entonces le pone valor absoluto y le cambia la dirección al movimiento

GRAMATICA CLIENTE

A continuación se muestra la gramática utilizada para el analisis lexico de los movimientos del cliente:

Y la gramática para lo que regresa el servidor (xml):

```
Gramatica txt prueba 1
expr ---> expr typeMov ;
        | typeMov ;
typeMov---> instr (operacion1)
           |push instr (operacion1)
instr---> up
         |down
         |left
         |right
operacion1---> operacion1 PLUS operacion1
             | operacion1 SUBSTRACTION operacion1
             | operacion2
operacion2---> operacion2 MULTIPLY operacion2
             | operacion2 DIVIDE operacion2
             | operacion3
operacion3---> int
             | CEIL (decimal | operacion1)
             | FLOOR (decimal | operacion1)
operacion4---> DECIMAL
             | operacion1

-----------------> modificacion gramatica
expr ---> expr typeMov ;
        | typeMov ;
typeMov---> instr (operacion1)
           |push instr (operacion1)
instr---> up
         |down
         |left
         |right
operacion1---> operacion1 PLUS operacion1
             | operacion1 SUBSTRACTION operacion1
             | operacion2
operacion2---> operacion2 MULTIPLY operacion2
             | operacion2 DIVIDE operacion2
             | operacion3
operacion3---> int
             | operacion4
operacion4---> DECIMAL
             | (operacion1)
             | CEIL (operacion4)
             | FLOOR (operacion4)
```

```
posTB---> <posX> ENTERO </posX>
        | <posY> ENTERO </posY>
        | <type> typeH </type>


typeH---> HALL
        | BRICK

config---> config configPro
         | configPro

configPro---> <box_color>  palabraColor </box_color>
            | <box_on_target_color>  palabraColor </box_on_target_color>
            | <target_color> palabraColor </target_color>
            | <brick_color>  palabraColor  </brick_color>
            | <hall_color>   palabraColor  </hall_color>
            | <undefined_color>  palabraColor  </undefined_color>
            | <player_color>  palabraColor  </player_color>
/
```

```
/*Gramatica cup para xml
Gramatica cup para xml

inic---> <?xml VERSION = "LITERAL" ENCONDING = "LITERAL" ?> <WORLDS> WorldModel <\WORLDS>

WorldModel----> WorldModel <WorldModel> atri <\WorldModel>
         | <WorldModel> atri <\WorldModel>

atri----> atri producWorld
       | producWorld

producWorld---> <name> PALABRA <\name>
             | <rows> ENTERO <\rows>
             | <cols> ENTERO <\cols
             | <config> config <\config>
             | <board> configBoard <\board>
             | <boxes> configBox </boxes>
             | <targets> configTarget </targets>
             | <player> configPlayer </player>



configBoard---> configBoard posTB
              | posTB


configBox---> configBox posT
            | postT

configTarget-->configTarget postT
            | postT

configPlayer-->configPlayer postT
            | postT

posT--->  <posX> ENTERO </posX>
       | <posY> ENTERO </posY>

posTB---> <posX> ENTERO </posX>
       | <posY> ENTERO </posY>
```

SERVIDOR

Paquete Layout: este paquete solo contiene las clases swing que servirán para ver el reporte de operaciones y para ver las respuestas del cliente y las respuestas del servidor.

Paquete Models: este paquete contiene las clases modelos que se utilizan para la creación del xml por medio de una librería, que permite crear xml a base de objetos ya creados. Dadas las clases: **BoardsModel, Error, ErrorModel, ErrorType, NamWorld,WorldModel,WorldModelName,WorldsModel.**

Paquete Objects: Esta clase tiene todos los objetos que se utilizan en el singleton implementado en el análisis del json y solo tiene sus constructores, getters y setters. dadas las clases: **Board, Box, ConfigWorld, Player, Response, Target,World.**

Paquete Objects Report: Solo tiene una clase llamada **Report Operaciones** la cual es utilizada para reportar el operador que se encuentra.

Paquete Parser Json: tiene las clases generadas por lexer.flex y json.cup

Contiene las clases encargadas del analizador lexico, sintáctico y semántico siendo: **Lexer, ParserJson y ParserJsonSym.**

Paquete Parser Xml: tiene las clases generadas por lexXml.flex y cup Xml.cup

Contiene las clases encargadas del analizador léxico, sintáctico y semántico siendo: **LexXml, ParserXml y ParserXmlSym.**

Paquete Server: Esta clase tiene los convertidores de xml y la clase que se encarga de ver la conexión del servidor con el cliente.

Converter: Esta clase tiene los convertidores de objeto a xml

convertObjectToXml: es el encargado de recorrer el array de errores y convertirlos en xml.

convertObjectToXmlRequestWorld: es el encargado de recorrer el array de de mundo y convertirlos en xml.

convertObjectToXml: es el encargado de verificar si el mundo enviado esta bien, y dado puede escribirlo para guardarlo en el doc .xml o solamente pasarlo a xml y regresarlo para el cliente.

Server: esta clase se encarga de realizar la conexión con el cliente

conect: es el encargado de realizar la conexión, y verificar el tipo de petición que se esta haciendo.

compileJson: es el que parsea lo que llega desde el cliente y mandara a llamar al método según lo que pidió el cliente por medio del objeto response.

verificarWorld: solamente verifica que el mundo no exista.

VWorld: esta clase se encarga de las verificaciones sobre el json enviado.

algunos de los métodos para verificar:

* Verificar que no se este definiendo un target o box en brick

*Verificar que el numero de targets sea igual al de boxes

*Verificar que el numero de columnas y filas este

* Verificar que el Board, Box y Target su posicion si este definido

*Verificar que el box este definido dentro del tablero

*Verificar que el target este definido en el tablero

* Verificar que no este repetido la definicion del box

* Verificar que no este repetido la definicion del board

*Verificar que no este repetido el target

* Modificar el color por defecto en caso no este definido

* Verificar que este definido la posicion de Board,Box y Target en el tablero

GRAMÁTICA SERVIDOR:

A continuación presentamos la gramática utilizada para el analisis del json:

```
/* gramatica txt
inic ::= worldPro:a
                    {:
                       RESULT=a;
                    :};

   worldPro ::=  worldPro:n1 COMA Llave_A atri:a1 Llave_C
                 {:
                 a1=new World(getSingletonInstanceWorld().getName(), getSingletonInstanceWorld().getRows(),getSingletonInstanceWorld().getCols(),getSingletonInstanceWorld().getConfig(),
                                                        getSingletonInstanceWorld().getArrayBoard(),getSingletonInstanceWorld().getArrayBoxes(),getSingletonInstanceWorld().getArrayTarget(),getSingletonInstanceWorld().getPlayer());
                 n1.add(a1);
                 RESULT=n1;
                 :}
              | Llave_A atri:a1 Llave_C {:
               RESULT= new ArrayList<World>();
               RESULT.add(new World(getSingletonInstanceWorld().getName(), getSingletonInstanceWorld().getRows(),getSingletonInstanceWorld().getCols(),getSingletonInstanceWorld().getConfig(),
                                    getSingletonInstanceWorld().getArrayBoard(),getSingletonInstanceWorld().getArrayBoxes(),getSingletonInstanceWorld().getArrayTarget(),getSingletonInstanceWorld().getPlayer()));
                                                                                :};
   atri ::= atri COMA producWorld  {:RESULT= getSingletonInstanceWorld(); :}
          | producWorld  {:RESULT= getSingletonInstanceWorld(); :};

   producWorld ::= COMILLA NAME COMILLA DOS_PUNTOS COMILLA PALABRA COMILLA  {: getSingletonInstanceWorld().setName(a1.getLexeme());:}
                 | COMILLA ROWS COMILLA DOS_PUNTOS COMILLA operacion:a1 COMILLA  {: getSingletonInstanceWorld().setRows(Integer.parseInt(a1.getLexeme())); :}
                 | COMILLA COLS COMILLA DOS_PUNTOS COMILLA operacion:a1 COMILLA  {: getSingletonInstanceWorld().setCols(Integer.parseInt(a1.getLexeme())); :}
                 | COMILLA CONFIG COMILLA DOS_PUNTOS LLAVE_A config:a LLAVE_C   {: getSingletonInstanceWorld().setConfig(new ConfigWorld(getSingletonInstanceConfig().getBox_color(),getSingletonInstanceConfig().getBox_on_target_color(),getSingletonInstanceConfig().getTarget_color(),
                                          getSingletonInstanceConfig().getBrick_color(), getSingletonInstanceConfig().getHall_color(), getSingletonInstanceConfig().getUndefined_color(), getSingletonInstanceConfig().getPlayer_color()));  :}
                 | COMILLA BOARD COMILLA DOS PUNTOS L_CORCHETE configBoard R_CORCHETE    {: getSingletonInstanceWorld().getArrayBoard().add(new Board(getSingletonInstanceBoard().getPosX(),getSingletonInstanceBoard().getPosY(),getSingletonInstanceBoard().getType()));  :}
                 | boxA configBox:a1 boxC          {: getSingletonInstanceWorld().getArrayBoxes().add(new Box(getSingletonInstanceBox().getPosX(),getSingletonInstanceBox().getPosY()));  :}
                 | targetA configTarget:a1 targetC {: getSingletonInstanceWorld().getArrayTarget().add(new Target(getSingletonInstanceTarget().getPosX(),getSingletonInstanceTarget().getPosY())); :}
                 | playerA configPlayer:a1 playerC    {: getSingletonInstanceWorld().setPlayer(new Player(getSingletonInstancePlayer().getPosX(),getSingletonInstancePlayer().getPosY())); :};

   configBoard ::= configBoard COMA LLAVE_A posTB:a1 LLAVE_C {: RESULT=a1; :}
                 | LLAVE_A posTB:a1 LLAVE_C {: RESULT=a1; :};

   configBox ::= configBox COMA LLAVE_A posBox:a1 LLAVE_C{: RESULT=a1; :}
               | LLAVE_A posBox:a1 LLAVE_C
   {: RESULT=a1; :}
   ;
   configTarget ::= configTarget COMA LLAVE_A posT:a1 LLAVE_C {: RESULT=a1; :}
                 | LLAVE_A posT:a1 LLAVE_C
   {: RESULT=a1; :}
   ;
   configPlayer ::= configPlayer COMA LLAVE_A posPlayer:a1 LLAVE_C {: RESULT=a1; :}
                  | LLAVE_A posPlayer:a1 LLAVE_C
   {: RESULT=a1;:}
   ;
   posT ::=  posX_A operacion:a1 posX_C
   {: getSingletonInstanceTarget().setPosX(Integer.parseInt(a1.getLexeme())); :}

          | posY_A operacion:a1 posY_C
   {: getSingletonInstanceTarget().setPosY(Integer.parseInt(a1.getLexeme())); :}
   ;
```

```
posBox ::=  posX_A operacion:a1 posX_C
 {: getSingletonInstanceBox().setPosX(Integer.parseInt(a1.getLexeme())); :}

        | posY_A operacion:a1 posY_C
 {:   getSingletonInstanceBox().setPosY(Integer.parseInt(a1.getLexeme())); :}
;

posPlayer ::=  posX_A operacion:a1 posX_C
{: getSingletonInstancePlayer().setPosX( Integer.parseInt(a1.getLexeme())); :}

        | posY_A operacion:a1 posY_C
{:   getSingletonInstancePlayer().setPosY(Integer.parseInt(a1.getLexeme())); :}
;

posTB ::=  posX_A operacion:a1 posX_C
{:  getSingletonInstanceBoard().setPosX(Integer.parseInt(a1.getLexeme())); :}

         | posY_A operacion:a1 posY_C
{:  getSingletonInstanceBoard().setPosY(Integer.parseInt(a1.getLexeme())); :}

         | typeA typeH:a1 typeC
{:  getSingletonInstanceBoard().setType((Integer)a1); :}
;

typeH ::= HALL:t1
 {: RESULT= t1.getTokenType() ;:}

         | BRICK:t1
{: RESULT=t1.getTokenType() ;:}
;
config ::= config COMA configPro
          | configPro:a1 {: RESULT=a1;:};
configPro ::= COMILLA BOX_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
 {:  getSingletonInstanceConfig().setBox_color(a1.getLexeme());  :}

             | COMILLA BOX_ON_TARGET_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setBox_on_target_color(a1.getLexeme());  :}

             | COMILLA TARGET_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setTarget_color(a1.getLexeme());  :}

             | COMILLA BRICK_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setBrick_color(a1.getLexeme());  :}

             | COMILLA HALL_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setHall_color(a1.getLexeme());  :}

             | COMILLA UNDEFINED_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setUndefined_color(a1.getLexeme());  :}

             | COMILLA PLAYER_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setPlayer_color(a1.getLexeme());  :}
```

```
{:  getSingletonInstanceConfig().setHall_color(a1.getLexeme());  :}

             | COMILLA UNDEFINED_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setUndefined_color(a1.getLexeme());  :}

             | COMILLA PLAYER_COLOR COMILLA DOS_PUNTOS COMILLA PALABRA_COLOR COMILLA
{:  getSingletonInstanceConfig().setPlayer_color(a1.getLexeme());  :}

operacion ::= ENTERO
             | COMILLA operacion5:a1 COMILLA

operacion5 ::= L_PARENT operacion1:a1 R_PARENT
              | operacion1:a

operacion1 ::= operacion1:a1 SUMA operacion1:a2 {:RESULT = a1 + a2 ; :}
           | operacion1:a1 RESTA operacion1:a2 {:RESULT = a1 - a2 ; :}
           | operacion2:a1 {:RESULT = a1 ; :};
operacion2 ::= operacion2:a1 MULTIPLY operacion2:a2 {:RESULT = a1 * a2 ; :}
            | operacion2:a1 DIVISION operacion2:a2 {:RESULT = a1 / a2 ; :}
            | operacion3:a1 {:RESULT = a1 ; :} ;
operacion3 ::= NUM:a1 {:RESULT = Double.parseDouble(a1.getLexeme()); :}
             | operacion4:a1 {:RESULT =a1;:};
operacion4 ::= L_PARENT operacion1:a1 R_PARENT {:RESULT =a1;:}
             | CEIL L_PARENT operacion1:a1  R_PARENT {:RESULT = Math.ceil(a1);:}
             | FLOOR L_PARENT operacion1:a1 R_PARENT {:RESULT = Math.floor(a1);:}
             | DECIMAL:a1 {: RESULT = Double.parseDouble(a1.getLexeme()); :}; */
```