# Objetivos Generales

* Familiarizar al estudiante con la herramienta JFlex y CUP.
* Familiarizar al estudiante con los formatos JSON y XML.
* Aplicar conocimientos de análisis léxico y sintáctico.

# Objetivos Específicos

* Crear archivos de configuración con JFlex y CUP.
* Combinar la funcionalidad de JFlex y CUP en aplicaciones reales.
* Familiarizar al estudiante con el lenguaje de programación Kotlin.
* Familiarizar al estudiante con el desarrollo de aplicaciones móviles Android.

# Descripción de la Actividad

El mercado de aplicaciones móviles está en constante crecimiento debido a que se tiene la capacidad de llegar a más público, lo que puede generar ingresos significativos, esto lleva a una alta demanda de desarrolladores de aplicaciones, específicamente con Kotlin, el lenguaje oficial recomendado por Google para el desarrollo de aplicaciones móviles Android.

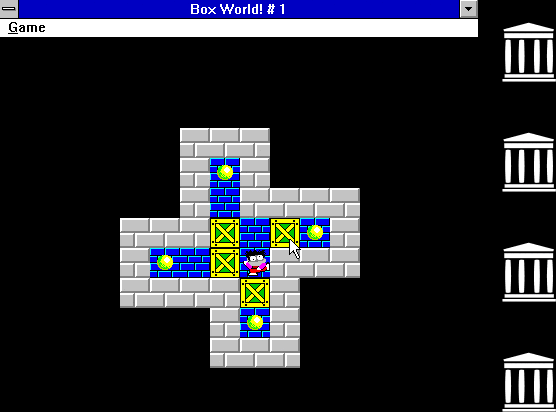
Basándose en lo anterior, usted ha sido contratado como Ingeniero de Software por la empresa App Design para desarrollar la siguiente versión del famoso juego de los años 80, llamado Boxworld o conocido también como Sokoban. El juego estará formado por una aplicación móvil o cliente y una aplicación de escritorio que tendrá las funciones de un servidor.

# Aplicación móvil

Boxworld es un juego de rompecabezas en el cual el jugador está atrapado en un almacén dividido en espacios cuadrados, mueve cajas tratando de llevarlas a lugares de almacenamiento llamados objetivos(targets).

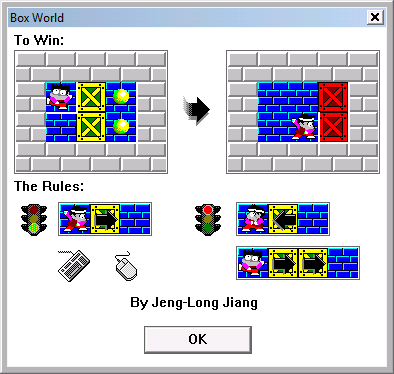
Las reglas del juego son bastante sencillas:

* Cada cuadrado del almacén o tablero puede ser una pared, una caja o un área de almacenamiento de las cajas. (*Ver imagen 1*)
* Un movimiento del jugador solo es posible si la casilla de destino está vacía (es decir, que no es una pared o una caja; sin embargo, puede ser un área de almacenamiento).
* Si un cuadrado contiene una caja y la siguiente en la misma dirección está vacía o es un área de almacenamiento, el movimiento también está permitido.
* Cuando se empuja la caja hacia un cuadro vacío o un lugar de almacenamiento, el jugador simplemente toma el lugar de la caja.
* Un nivel se resuelve o termina cuando todas las cajas están en un lugar de almacenamiento. En la versión original el color de las cajas son amarillo, pero cuando están en un lugar de almacenamiento, estás son rojas.
* Las reglas no permiten saltar una caja y/o atravesar las paredes.
* El juego se pierde si el jugador se queda sin movimientos posibles.



*Imagen 1.*

En la *imagen 1*, se puede ver que hay cuatro cajas, y cuatro puntos de almacenamiento, el objetivo del juego, como ya se describió anteriormente, sería mover las cuatro cajas hacia los lugares de almacenamiento.



*Imagen 2.*

Su objetivo como programador es desarrollar una gramática libre de contexto para el lenguaje utilizado para describir los tableros o almacenes para el juego, para su posterior renderización en la aplicación móvil.

Y posteriormente le corresponderá desarrollar una gramática libre de contexto para las instrucciones que se usarán para jugar y resolver los distintos niveles o mundos que el juego ofrezca. (de ahí el nombre boxworld).

## Lenguaje usado para definir los tableros o almacenes:

El siguiente ejemplo describe el tablero mostrado en la imagen 2.

|  |
| --- |
| {   "name": "tatoine",  "rows": "5 \* 5 - 20",  "cols": "10 / 5 + 2",  "config": {  "box\_color": "#ffff00",  "box\_on\_target\_color": "#fe0002",  "target\_color": "#d6fe0a",  "brick\_color": "#bdbebf",  "hall\_color": "#0001ff",  "undefined\_color": "#303030",  "player\_color": "#ff00f7"  },  "board": [  {  "posX": 0,  "posY": "2 - 2",  "type": "BRICK"  },  {  "posX": 0,  "posY": "3 / 3",  "type": "BRICK"  },  {  "posX": 0,  "posY": "2 \* 1",  "type": "BRICK"  },  {  "posX": 0,  "posY": "9 / 3 \* 1",  "type": "BRICK"  },  {  "posX": 0,  "posY": "10 / 2 - 1",  "type": "BRICK"  },  {  "posX": "2 / 2",  "posY": 0,  "type": "BRICK"  },  {  "posX": "10 - 3 \* 3",  "posY": "3 \* 2 - 5",  "type": "HALL"  },  {  "posX": 1,  "posY": "2 \* 1",  "type": "HALL"  },  {  "posX": 1,  "posY": "12 / 4 \* 1",  "type": "HALL"  },  {  "posX": "17 - 4 \* 4",  "posY": "10 / 2 - 1",  "type": "BRICK"  },  {  "posX": "2 / 2 + 1",  "posY": 0,  "type": "BRICK"  },  {  "posX": "10 - 2 \* 4",  "posY": "3 \* 2 - 5",  "type": "HALL"  },  {  "posX": 2,  "posY": "2 \* 1",  "type": "HALL"  },  {  "posX": 2,  "posY": "12 / 4 \* 1",  "type": "HALL"  },  {  "posX": "20 - 6 \* 3",  "posY": "10 / 2 - 1",  "type": "BRICK"  },  {  "posX": 3,  "posY": 0,  "type": "BRICK"  },  {  "posX": 3,  "posY": "5 / 5 \* 6 - 5",  "type": "BRICK"  },  {  "posX": 3,  "posY": "2 \* 1",  "type": "BRICK"  },  {  "posX": "9 / 3",  "posY": "9 / 3 \* 1",  "type": "BRICK"  },  {  "posX": 3,  "posY": "10 / 2 - 1",  "type": "BRICK"  }  ],  "boxes": [  {  "posX": 2,  "posY": 1  },  {  "posX": 2,  "posY": "1 + 1"  }  ],  "targets": [  {  "posX": "4 / 2 + 1",  "posY": 1  },  {  "posX": "4 / 2 + 1",  "posY": "1 \* 1 + 1"  }  ],  "player": {  "posX": 1,  "posY": "(5 + 2) / (3 + 4)"  } } |

Cómo se puede observar, el lenguaje usado sigue la estructura JSON (JavaScript Object Notation) que es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluidos C, C++, C#, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen de JSON un lenguaje de intercambio de datos ideal.

Cómo se mostrará más adelante este texto en formato JSON, será enviado al servidor para su análisis léxico, sintáctico y semántico.

### Consideraciones a tomar en cuenta en el lenguaje que define los tableros de juego

|  |  |
| --- | --- |
| Palabra reservada | Descripción |
| name | Nombre del nivel, tablero o mundo que se está describiendo, se usaran únicamente caracteres alfanuméricos, el nombre debe de empezar con una letra minúscula o guion bajo, seguido de una letra, seguido de cualquier carácter alfanumérico en mayúscula o minúscula. |
| rows | Número de filas que tendrá el tablero. Puede ser cualquier número entero entre comillas o no. O puede ser cualquier operación aritmética entre comillas. |
| cols | Número de columnas que tendrá el tablero. Puede ser cualquier número entero entre comillas o no. O puede ser cualquier operación aritmética entre comillas. |
|  | |
| config | Objeto JSON que define las configuraciones básicas que tendrá el tablero de juego. **Este campo es opcional.** |
| **Campos que pueden venir dentro de config** | |
| box\_color | Color que tendrán las cajas dentro del campo de juego que no estén sobre un objetivo o lugar de almacenamiento. El valor debe ser un color en formato hexadecimal. **Campo opcional**. (Eje: #FCFF33, #F00, #fcff00). |
| box\_on\_target\_color | Color en formato hexadecimal, que tendrán las cajas que se encuentran en un lugar de almacenamiento. **Campo opcional**. |
| target\_color | Color en formato hexadecimal, que tendrán los lugares de almacenamiento. **Campo opcional**. |
| brick\_color | Color en formato hexadecimal, que tendrán los ladrillos o paredes dentro del campo de juego. **Campo opcional**. |
| hall\_color | Color en formato hexadecimal, que tendrá el pasillo o camino en el campo de juego. **Campo opcional**. |
| undefined\_color | Color en formato hexadecimal, que tendrán las casillas que no están definidas como pasillo, pared o ladrillo, caja, lugar de almacenamiento o jugador. **Campo opcional**. |
| player\_color | Color en formato hexadecimal, que tendrá el jugador dentro del campo de juego. **Campo opcional**. |
|  | |
| board | Arreglo que definirá la estructura del almacén o tablero de juego. |
| **Campos incluidos en el arreglo de objetos que definen el tablero** | |
| posX | Posición en el eje “X” o columna que identifica a la casilla. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
| posY | Posición en el eje “Y” o filas que identifica a la casilla. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
| type | El tipo de casilla, que puede ser pared o ladrillo (BRICK). Pasillo o camino (HALL). |
|  | |
| boxes | Arreglo que definirá la posición inicial de las cajas dentro del tablero de juego. |
| **Campos incluidos en el arreglo de objetos que definen la posición inicial de las cajas** | |
| posX | Posición en el eje “X” o columna que identifica a la casilla donde se ubicará una caja al inicio de una partida. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
| posY | Posición en el eje “Y” o fila que identifica a la casilla donde se ubicará una caja al inicio de una partida. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
|  | |
| targets | Arreglo que definirá la posición de los lugares de almacenamiento dentro del tablero de juego. |
| **Campos incluidos en el arreglo de objetos que definen la posición de los lugares de almacenamiento** | |
| posX | Posición en el eje “X” o columna que identifica a la casilla donde se ubicará el lugar de almacenamiento. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
| posY | Posición en el eje “Y” o fila que identifica a la casilla donde se ubicará el lugar de almacenamiento. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
|  | |
| player | Objeto donde se definirá la posición del jugador al inicio de la partida |
| **Campos incluidos en el objeto que define la posición inicial del jugador** | |
| posX | Posición en el eje “X” o columna que identifica a la casilla donde se ubicará el jugador al inicio de la partida. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |
| posY | Posición en el eje “Y” o fila que identifica a la casilla donde se ubicará el jugador al inicio de la partida. Puede ser un número entero entre comillas o no, o una operación aritmética entre comillas. |

##### Importante:

1. Los campos o parámetros pueden venir en cualquier orden y no necesariamente en el mismo orden del ejemplo.
2. Para los campos que aceptan valores numéricos u operaciones, en caso de ser un único número (positivo o negativo) puede o no venir entre comillas. En caso de ser una operación, debe venir entre comillas.
3. Se deberá validar que dos campos no se definan más de una vez, incluidos los opcionales.
4. Las casillas son 0-indexed, es decir, el conteo de la posición de una casilla empieza en cero (0).
5. Para las casillas de juego de tipo “BRICK” o “HALL”, se deberá de validar que cada casilla solo se defina una vez. En caso de no definir una casilla, esta será de tipo undefined.
6. Se debe de validar que la posición de las cajas, lugares de almacenamiento y la posición inicial del jugador sea en una casilla válida. (No de tipo ladrillo o “BRICK” o undefined).
7. Se debe de validar que el número de cajas sea el mismo que el número de lugares de almacenamiento.
8. Para los campos de tipo posición (“posX” y “posY”) se deberá de validar que estén dentro del tablero.
9. En los campos que van entre comillas, se deben de ignorar espacios y saltos de línea. (Incluir también los caracteres “\n”, “\t”).
10. Los parámetros opcionales son los de configuración del tablero (***config***) en el caso de no ser incluidos, los colores por defecto a usar son:

|  |  |  |
| --- | --- | --- |
| campo | Valor | ejemplo: |
| box\_color | #ffff00 |  |
| box\_on\_target\_color | #fe0002 |  |
| target\_color | #d6fe0a |  |
| brick\_color | #bdbebf |  |
| hall\_color | #0001ff |  |
| undefined\_color | #303030 |  |
| player\_color | #ff00f7 |  |

1. Para las operaciones aritméticas, se pueden incluir también números decimales, pero el resultado final debe de ser de tipo entero, para ello se incluyen las siguientes funciones:

|  |  |  |  |
| --- | --- | --- | --- |
| **Función** | **Descripción** | **Ejemplo** | **Resultado** |
| FLOOR(<número>|<operación>) | Redondea hacia abajo el valor decimal o entero especificado y lo devuelve. | FLOOR(4.3) | 4 |
| CEIL(<número>|<operación>) | Redondea hacia arriba el valor decimal o entero especificado y lo devuelve. | CEIL(4.3) | 5 |

1. Las operaciones aritméticas pueden incluir paréntesis y los siguientes operadores matemáticos.

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Precedencia de mayor a menor |
| FLOOR(<número>|<operación>) | Ver descripción en la tabla anterior | 4 |
| CEIL(<número>|<operación>) | Ver descripción en la tabla anterior | 4 |
| **( )** | Paréntesis | 3 |
| **/** | División | 2 |
| **\*** | Multiplicación | 2 |
| **+** | Suma | 1 |
| **-** | Resta | 1 |

## Lenguaje usado para el movimiento del jugador

Para el movimiento del jugador, se empleará la siguiente sintaxis:

|  |
| --- |
| *# movimiento hacia arriba* up(2 \* 3 + 10); down(6 + 1 - 10); left(12 + 3 \* 2 - 1); right(3 \* 2 - 25); *# movimiento hacia la izquierda 20 casillas*  *# empujando la caja hacia la izquierda 7 casillas* push left (FLOOR (4.2) \* 3 - 5);  *# fin del juego* |

|  |  |
| --- | --- |
| **Instrucción** | **Descripción** |
| up (<número>|<operación>) | Número de casillas que el jugador se moverá hacia arriba. |
| down (<número>|<operación>) | Número de casillas que el jugador se moverá hacia abajo. |
| left (<número>|<operación>) | Número de casillas que el jugador se moverá hacia la izquierda. |
| right (<número>|<operación>) | Número de casillas que el jugador se moverá hacia la derecha. |
| push <dirección> (<número>|<operación>) | Número de casillas que el jugador empujará una caja hacia la dirección especificada. |
| <comentarios\_de\_linea> | El comentario de línea empezará con el símbolo numeral (#). |
| **;** | Se usará el punto y coma como final de sentencia. |

##### Importante:

1. El resultado de la operación o el número pasado como parámetro a las instrucciones de movimiento debe de ser un número entero. Es posible usar las funciones **FLOOR** & **CEIL** que también formaran parte de este lenguaje.
2. El resultado de la operación pasada como parámetro a las instrucciones de movimiento deberá ser un número positivo. En caso de ser un número negativo, la dirección del movimiento cambiará hacia la dirección contraria (up -> down, left -> right, etc.).
3. Para mover una caja hacia cualquier dirección se debe de agregar la palabra reservada push a la instrucción de movimiento.
4. Para el movimiento de las cajas, únicamente es permitido empujar una caja hacia una dirección específica (siempre y cuando las reglas del juego así lo permitan) y no está permitida la acción de “jalar”, “tirar” o “atraer” una caja.
5. En caso de que una acción no tenga sentido o no sea posible ejecutar el movimiento (por haber una pared o ladrillo, o no poder mover la caja), se pasara a la siguiente instrucción disponible.
6. Se deberá intentar ejecutar todas las instrucciones posibles (siempre y cuando las reglas del juego así lo permitan). Por lo tanto, habrá instrucciones que únicamente se ejecutarán parcialmente. Por ejemplo, si se indica mover hacia la derecha cierta cantidad de casillas, pero sí el espacio disponible no es suficiente, el jugador únicamente se moverá las casillas que así lo permitan.
7. Se deberá presentar un reporte de las instrucciones que no se hayan ejecutado, así como una descripción de porque no se ejecutó el movimiento.
8. El juego termina cuando todas las cajas estén en un lugar de almacenamiento o cuando las instrucciones hayan terminado.

El siguiente ejemplo es una posible combinación para resolver el tablero de la imagen 2.

|  |
| --- |
| *# hacia abajo 1 posición* down (2 \* 5 / 10);  *# empujando la caja hacia la derecha* push right (6 - 5);  *# hacia la izquierda una casilla* left (6 / 2 - 2);  *# hacia arriba una casilla* down (-1);  *# empujando la caja hacia la izquierda, 1 casilla* push left (3 / 3);  *# fin* |

### 

# Reportes:

## Reporte de ejecución:

Se deberá mostrar un reporte de ejecución de aquellas instrucciones que, por las reglas del juego, no se hayan podido ejecutar total o parcialmente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrucción** | **Línea** | **Columna** | **Tipo** | **Descripción** |
| push | 3 | 4 | Total | No se ha podido mover la caja porque la casilla (2,3) es una pared. |
| left | 5 | 6 | Parcial | No se ha podido mover hacia la izquierda por la presencia de una caja. |

## Reporte de operadores matemáticos

|  |  |  |
| --- | --- | --- |
| **Operador** | **Línea** | **Columna** |
| Suma | 7 | 6 |
| Resta | 9 | 6 |
| CEIL | 101 | 8 |

## Reporte de instrucciones parcial o totalmente ejecutadas

|  |  |  |  |
| --- | --- | --- | --- |
| **Instrucción** | **Línea** | **Columna** | **Tipo** |
| left | 1 | 1 | Total |
| right | 4 | 5 | Parcial |
| push | 35 | 3 | Total |

## Reporte de movimientos

|  |  |
| --- | --- |
| **Dirección** | **Cantidad** |
| left | 100 |
| right | 124 |
| up | 121 |
| down | 75 |

## Reporte de errores:

Se deberá presentar el siguiente reporte de errores en la aplicación móvil.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lexema** | **Línea** | **Columna** | **Tipo** | **Descripción** |
| @$% | 2 | 13 | Léxico | Símbolo no existe en el lenguaje |
| push | 3 | 1 | Sintáctico | Se esperaba “;” (punto y coma). |
|  | 6 | 7 | Semántico | En la instrucción para la casilla (0,3), esta ya fue definida anteriormente. |

# 

# Servidor:

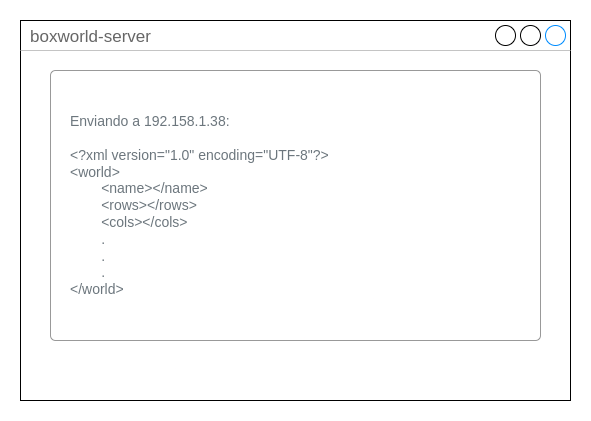
Los tableros o mundos del juego que se ingresaran a la aplicación móvil serán enviados al servidor, y este último será el encargado de analizar léxico, gramática y semánticamente el archivo JSON para así construir los diversos niveles o mundos que compondrán al juego y también será el encargado de almacenar o guardar estos niveles en un archivo llamado **sokoban.xml**, según el nombre del archivo podemos ver que el formato para guardar los niveles del juego será en formato XML, a continuación un ejemplo:

|  |
| --- |
| **<?**xml version="1.0" encoding="UTF-8"**?>** <worlds>  <world>  <name>tatoine</name>  <rows>5</rows>  <cols>4</cols>  <config>  <box\_color>#ffff00</box\_color>  <box\_on\_target\_color>#fe0002</box\_on\_target\_color>  <target\_color>#d6fe0a</target\_color>  <brick\_color>#bdbebf</brick\_color>  <hall\_color>#0001ff</hall\_color>  <undefined\_color>#303030</undefined\_color>  <player\_color>#ff00f7</player\_color>  </config>  <board>  <posX>0</posX>  <posY>0</posY>  <type>BRICK</type>  </board>  <board>  <posX>0</posX>  <posY>1</posY>  <type>BRICK</type>  </board>  <board>  <posX>0</posX>  <posY>2</posY>  <type>BRICK</type>  </board>  <board>  <posX>0</posX>  <posY>3</posY>  <type>BRICK</type>  </board>  <board>  <posX>0</posX>  <posY>4</posY>  <type>BRICK</type>  </board>  <board>  <posX>1</posX>  <posY>0</posY>  <type>BRICK</type>  </board>  <board>  <posX>1</posX>  <posY>1</posY>  <type>HALL</type>  </board>  <board>  <posX>1</posX>  <posY>2</posY>  <type>HALL</type>  </board>  <board>  <posX>1</posX>  <posY>3</posY>  <type>HALL</type>  </board>  <board>  <posX>1</posX>  <posY>4</posY>  <type>BRICK</type>  </board>  <board>  <posX>2</posX>  <posY>0</posY>  <type>BRICK</type>  </board>  <board>  <posX>2</posX>  <posY>1</posY>  <type>HALL</type>  </board>  <board>  <posX>2</posX>  <posY>2</posY>  <type>HALL</type>  </board>  <board>  <posX>2</posX>  <posY>3</posY>  <type>HALL</type>  </board>  <board>  <posX>2</posX>  <posY>4</posY>  <type>BRICK</type>  </board>  <board>  <posX>3</posX>  <posY>0</posY>  <type>BRICK</type>  </board>  <board>  <posX>3</posX>  <posY>1</posY>  <type>BRICK</type>  </board>  <board>  <posX>3</posX>  <posY>2</posY>  <type>BRICK</type>  </board>  <board>  <posX>3</posX>  <posY>4</posY>  <type>BRICK</type>  </board>  <board>  <posX>3</posX>  <posY>5</posY>  <type>BRICK</type>  </board>  <boxes>  <posX>2</posX>  <posY>1</posY>  </boxes>  <boxes>  <posX>2</posX>  <posY>2</posY>  </boxes>  <targets>  <posX>3</posX>  <posY>1</posY>  </targets>  <targets>  <posX>3</posX>  <posY>2</posY>  </targets>  <player>  <posX>1</posX>  <posY>1</posY>  </player>  </world>  <world>  .  .  .  </world>  <world>  .  .  .  </world> </worlds> |

El servidor guardará todos los tableros en el mismo archivo según se puede observar en el ejemplo anterior y será el encargado de enviarle la configuración para cada tablero que el jugador solicite en formato XML, para su posterior renderización en la aplicación móvil.

Se deberá validar que el nombre de cada nivel del juego sea único, para que no existan confusiones al momento de solicitar la configuración de un nivel en específico.

La aplicación de escritorio también tendrá que tener una vista para mostrar las peticiones recibidas de la aplicación móvil o cliente, así como las respuestas enviadas hacia la aplicación móvil.



La comunicación entre la aplicación móvil y el servidor será mediante sockets, esto significa que el servidor estará continuamente aceptando la conexión de clientes, en este caso la aplicación móvil.

# 

# Ejemplos de posibles vistas móviles

|  |  |
| --- | --- |
|  |  |
| *Imagen 2. Editor para el lenguaje que define un tablero.* | *Imagen 3. Vista del tablero de juego y editor para las instrucciones* |

|  |
| --- |
|  |
| *Imagen 4. Reporte de errores.* |

# Comportamiento del Sistema:

Después de describir cada una las aplicaciones o módulos que conforman el sistema, se describirá brevemente el comportamiento de todo el sistema:

## Aplicación móvil

* Aplicación Android escrita con Kotlin.
* Comunicación al servidor por medio de sockets.
* JFlex y CUP para cualquier tarea relacionada con el reconocimiento de texto.

## Servidor:

* Aplicación de escritorio, desarrollada con Java.
* Exposición de servicios por medio de sockets.
* JFlex y CUP para cualquier tarea relacionada con el reconocimiento de texto.

La aplicación móvil, que recibe las instrucciones para crear los diferentes niveles de boxworld en formato JSON, envía este texto al servidor para su análisis léxico, sintáctico y semántico. El servidor deberá devolver una confirmación si todo el análisis ha sido correcto o devolverá los errores que hayan sido encontrados durante el análisis del texto.

En caso de tener errores, estos deben ser devueltos en formato XML al cliente, y este debe de encargarse de analizar este texto y mostrarlo en un formato más conveniente para informar al usuario.

|  |
| --- |
| **<?**xml version="1.0" encoding="UTF-8"**?>** <errors>  <error>  <lexema>#@@</lexema>  <line>1</line>  <column>5</column>  <type>LEXICO</type>  <description>  El texto no se encuentra definida en la gramatica  </description>  </error> </errors> |

El servidor deberá mostrar el texto recibido del cliente y el texto que ha sido enviado hacia el cliente. Y también será el encargado de almacenar los niveles en un archivo XML llamado **sokoban.xml**.

El cliente deberá solicitar un listado de los niveles guardados en el servidor y este último deberá enviar una respuesta en formato XML con los nombres de los diferentes niveles.

Solicitud del cliente:

|  |
| --- |
| {  "worlds": "all" } |

Respuesta del servidor:

|  |
| --- |
| **<?**xml version="1.0" encoding="UTF-8"**?>** <worlds>  <world>tatoine</world>  <world>alderaan</world>  <world>yavin\_iv</world> </worlds> |

Para jugar o usar un nivel de boxworld, se deberá de pedirle al servidor la configuración de un nivel en específico, con el formato siguiente:

|  |
| --- |
| {  "world": "tatoine" } |

La respuesta del servidor para esta petición se mostró en la sección que describe el comportamiento del servidor.

El cliente será el encargado de renderizar o pintar el tablero y también será el encargado de analizar las instrucciones del juego ingresadas por el usuario y generar los reportes.

Toda comunicación entre cliente y servidor será mediante sockets, el cliente envía solicitudes en formato JSON y el servidor envía las respuestas en formato XML.

## Arquitectura del Sistema

|  |
| --- |
|  |
| *Imagen 5. Diagrama del sistema* |

# 

# Consideraciones del proyecto:

* La aplicación móvil debe de ser desarrollada para la plataforma Android usando Kotlin como lenguaje de programación.
* El servidor debe ser desarrollada con el lenguaje de programación Java y deberá tener interfaz gráfica, usar swing o algún similar.
* Usar las herramientas JFlex y CUP para todo tipo de análisis léxico y sintáctico.
* Proyecto obligatorio para tener derecho a la siguiente práctica.
* Las copias obtendrán una nota de cero y se notificará a coordinación.
* Es posible usar código de internet siempre y cuando se entienda la funcionalidad del mismo para que se tome como válido.

## Entrega:

* Fecha de entrega: Viernes 17 de marzo.
* Componentes a entregar:
  + Código fuente (repositorio en GitHub).
  + Ejecutables (apk y jar).
  + Manual técnico, detallando la organización del proyecto, descripción de las expresiones regulares y gramática utilizada.
  + Manual de usuario.

## Calificación:

Pendiente de definir.