UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA DE SOFTWARE

Diseño y desarrollo de un software integrado (IDE, compilador y máquina virtual), para la enseñanza de la lógica de programación a partir de edades de 15 años, por medio de la creación propia de un lenguaje de programación de rutas instruccionales

T E S I S

QUE PARA OPTAR POR EL TÍTULO DE

LICENCIADO EN INGENIERÍA DE SOFTWARE

P R E S E N T A :

JONATHAN GARCÍA GIL

D I R E C T O R

MTRO. ADALBERTO ROBLES VALADEZ

Ciudad de México, mayo de 2023.

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO

COLEGIO DE CIENCIA Y TECNOLOGÍA

ACADEMIA DE INFORMÁTICA

Licenciatura en Ingeniería de Software

“ APÉNDICE 3: DOCUMENTACIÓN DEL CÓDIGO DESTINO”

TESIS

QUE PARA OPTAR POR EL TÍTULO DE LICENCIATURA EN:

INGENIERÍA DE SOFTWARE

PRESENTA:
JONATHAN GARCÍA GIL

DIRECTOR DE TESIS:
Mtro. Adalberto Robles Valadez
Profesor-Investigador de la Academia de Informática, UACM

Ciudad de México, marzo de 2023

Contenido

1 Documentación de los objetos JSON de cada instrucción

En este capítulo no detallaré cómo realizar la generación de código para el análisis sintáctico porque no acabaría (Razón por la cual fue el subtema anterior). En cambio, solo agregaré las instrucciones que tendrá que generar el Analizador Sintáctico.

1.1 Estructura básica de un programa

La estructura básica del programa es que siempre se agregue la cadena “programa(){}”. Esto definirá en primera instancia que el lenguaje que se creó está siendo bien utilizado. Para esto se modifica la producción inicial de la gramática para que regrese los siguientes valores:

Código de Ejemplo del lenguaje:

```
1. programa(){
2. }
```

JSON de respuesta:

```
1. {
2. "functions":[],
3. "main":[],
4. "message":"success",
5. "status":true
6. }
```

Explicación de la estructura JSON:

status: Indica que el programa ha sido generado con éxito.
message: Indica un mensaje de éxito.
main: Contiene un arreglo de estructuras con las instrucciones del programa principal, será la primera estructura que se lea.
functions: Contiene un arreglo de estructuras dónde se encuentran definidas las distintas funciones disponibles del programa.

1.2 Instrucciones básicas

Las instrucciones básicas del lenguaje son aquellas que solo desean que se ejecuten, no hay nada que resolver para estas instrucciones por lo que solo interesa conocer cuál instrucción se está ejecutando.

Código de Ejemplo del lenguaje:

```
1.      pick_up();
2.      toma_objeto();
3.
4.      put_down();
5.      soltar_objeto();
6.
7.      delete_object();
8.      eliminar_objeto();
9.
10.     disable_bomb();
11.     desactivar_bomba();
12.     disable_kaboom();
13.     desactivar_kaboom();
14.
15.     stop_painting();
16.     dejar_de_pintar();
17.
18.     object_in_front ();
19.     tengo_objeto_delante();
20.
21.     bomb_in_front();
22.     tengo_bomba_delante();
23.     kaboom_in_front();
24.     tengo_kaboom_delante();
25.
26.     in_front_me ();
27.     delante_de_mi();
28.
29.     wall_in_front();
30.     tengo_muro_delante();
```

JSON de respuesta:

```
1.  {"instruction":"TOMAR"},
2.  {"instruction":"SOLTAR"},
3.  {"instruction":"ELIMINAR"},
4.  {"instruction":"DESACTIVARKABOOM"},
5.  {"instruction":"DEJAPINTAR"},
6.  {"instruction":"OBJETODELANTE"},
7.  {"instruction":"KABOOMDEFRENTE"},
8.  {"instruction":"QUETENGODELANTE"},
9.  {"instruction":"MURODELANTE"}
```

Explicación de la estructura JSON:

instruction: Es el nombre de la instrucción a ejecutar, toda instrucción que se genera debe tener este valor. Esto indica exactamente qué instrucción se está ejecutando.

1.3 Instrucciones no tan elaboradas

Las instrucciones no tan elaboradas son las que tienen un parámetro de entrada y no hay nada que resolver.

Código de Ejemplo del lenguaje:

```
1.        paint_floor();
2.        pintar_suelo(azul);
3.        paint_floor(blue);
4.        pintar_suelo(green);
5.        paint_floor(verde);
6.        pintar_suelo(amarillo);
7.        paint_floor(yellow);
8.        pintar_suelo(red);
9.        paint_floor(rojo);
10.       pintar_suelo();
11.
12.       print_var(id);
13.       imprimir_var(id2);
14.       print_variable(id3);
15.       imprimir_variable(id4);
```

JSON de respuesta:

```
1.  {
2.  "color":"DEFAULT",
3.  "instruction":"PINTAR"
4.  },
5.  {"color":"azul","instruction":"PINTAR"},
6.  {"color":"blue","instruction":"PINTAR"},
7.  {"color":"green","instruction":"PINTAR"},
8.  {"color":"verde","instruction":"PINTAR"},
9.  {"color":"amarillo","instruction":"PINTAR"},
10. {"color":"yellow","instruction":"PINTAR"},
11. {"color":"red","instruction":"PINTAR"},
12. {"color":"rojo","instruction":"PINTAR"},
13.
14. {
15. "identifier":"id",
16. "instruction":"IMPRIMIRVARIABLE"
17. }
```

Explicación de la estructura JSON:

Para la estructura de Pintar el Suelo:
color: indica el color que el usuario ha puesto dentro del parámetro de entrada, si no ha puesto nada muestra “DEFAULT”. Debido al soporte de idiomas “color” regresará distintas opciones que indican un color.

Para la estructura imprimir variable:
identifier: Es el nombre de la variable que se desea imprimir.

1.4 Valores de entrada que se deben resolver:

Hay algunos valores de entrada de una instrucción que se deben resolver antes de mostrarle resultados al usuario final. Estas instrucciones no podrán considerarse como instrucciones de acciones, es decir, no son instrucciones que realicen una operación en el tablero del Entorno de Desarrollo.

1.4.1 El valor proviene de un número aleatorio

Esta instrucción genera un número aleatorio y es necesario resolver cuál es el valor que tendrá.

Código de Ejemplo del lenguaje:

```
1.  random();
2.  aleatorio(4,15);
```

JSON de respuesta:

```
1.  {
2.  "instruction": "RANDOM",
3.  "from": -1,
4.  "to": -1,
5.  "between": false
6.  },
7.
8.  {
9.  "instruction": "RANDOM",
10. "from": 4,
11. "to": 9,
12. "between": true
13. }
```

Explicación de la estructura JSON:

between: Significa que existen valores de entrada dentro de esta instrucción, es decir, que el usuario ha puesto un rango de números para generar un número aleatorio entre un número en específico y otro.
from: Si el usuario especificó un rango de valores, esta etiqueta tendrá el valor de partida.
to: Si el usuario especificó un rango de valores, esta etiqueta tendrá el valor de tope.

1.4.2 El valor proviene de una operación aritmética

Cuando el valor proviene de una operación aritmética es necesario tomar en cuenta los paréntesis, que el número provenga de un número aleatorio, de una variable o de un número. Para propósitos del proyecto, solo hay que pensar en que se generará una cadena de valores, más no interesa resolverla.

Código de Ejemplo del lenguaje:

```
algunaInstrucción (

82+(variable*random(1,10))-74

);
```

JSON de respuesta:

```
{
    "random": {},
    "type": 1,
    "value": 82,
    "operation": {
      "random": {},
      "type": 0,
      "value": -1,
      "operation": {
        "random": {
          "random": {},
          "type": 1,
          "value": 74,
          "operation": {},
          "valueFrom": "",
          "operator": "-",
          "isBracket": false
        },
        "type": 0,
        "value": -1,
        "operation": {
          "random": {},
          "type": 2,
          "value": -1,
          "operation": {
            "random": {
              "instruction": "RANDOM",
              "from": 1,
              "to": 10,
              "between": true
            },
            "type": 3,
            "value": -1,
            "operation": {},
            "valueFrom": "",
            "operator": "*",
            "isBracket": false
          },
          "valueFrom": "identificador",
          "operator": "",
          "isBracket": false
        },
        "valueFrom": "",
        "operator": "",
        "isBracket": true
      },
      "valueFrom": "",
      "operator": "+",
      "isBracket": true
    },
    "valueFrom": "",
    "operator": "",
    "isBracket": false
  }
```

Explicación de la estructura JSON:

type: es el tipo de valor que se ha leído, pueden ser los siguientes:

- 0: Indica que el usuario ha abierto un paréntesis.
- 1: Indica que se trata de un número
- 2: Indica que se trata del valor de una variable.
- 3: Indica que se trata de un valor aleatorio.

value: Si se trata de un type: 1 aquí se encontrará el valor numérico. Para el resto tendrá un valor "-1".

valueFrom: Si se trata de un type: 2 aquí se encontrará el nombre de la variable de dónde se deberá tomar el valor que contenga.

random:

- Si se trata de un type: 3 aquí se encontrará la estructura random con sus características para generarlo.
- Si se trata de un type 0 aquí se encontrará la continuación del cierre de paréntesis (En caso de existir)

operator: Si el lugar dónde se encuentra leyendo hay un operador aritmético a su izquierda aquí aparecerá, si no hay nada no aparecerá nada.

operation: Se encontrará la continuación de la operación que se está realizando.

1.5 Declaración de variables

La declaración de variables es una instrucción que declara variables que se pueden utilizar en todo el programa.

En el caso del lenguaje, cada declaración tendrá una instrucción individual aún si se utiliza la estructura para declarar mismas variables a la vez.

Código de Ejemplo del lenguaje:

```
1.        variable id31,
2.                 id32<-78,
3.                 id33<-valor,
4.                 id34<-aleatorio(4,9),
5.                 id35<-(25-12);
6.        var id5;
7.        var id6<-78;
8.        var id7<-random();
9.        var id8<- valor;
```

JSON de respuesta:

```
1.      {
2.        "identifier": "identificador",
3.        "random": { },
4.        "instruction": "DECLARAVARIABLE",
5.        "type": 0,
6.        "value": -1,
7.        "operation": { },
8.        "valueFrom": ""
9.      }
```

Explicación de la estructura JSON:

type: Incluye el tipo de valor que se recibió. Pueden ser los siguientes casos:

- 0: No se incluyó un valor de asignación para la variable.
- 1: La variable recibirá el valor de un valor numérico.
- 2: La variable recibirá el valor de un valor numérico aleatorio.
- 3: La variable recibirá el valor de una operación aritmética.
- 4: La variable recibirá el valor de otro identificador que debió ser previamente declarado.

value: Si se trata de un type: 1 aquí se encontrará el valor numérico asignado, de lo contrario siempre estará con un valor -1.
random: Si se trata de un type: 2 aquí se encontrará la estructura que se definió antes para números aleatorios.
operation: Si se trata de un type: 3 aquí se encontrará la estructura que se definió antes para operaciones aritméticas.
valueFrom: Si se trata de un type: 4 aquí se encontrará el nombre de la variable de la que se quiere tomar el valor numérico.

1.6 Modificación de variables

Muchas veces se intenta modificar los valores con los que se han creado las variables del programa, por lo que se contempla la siguiente estructura

Código de Ejemplo del lenguaje:

```
1.  identificador <- 14;
2.  identificador <- identificador2;
3.  identificador <- random();
4.  identificador <- 14+12;
```

JSON de respuesta:

```
1.      {
2.        "identifier": "identificador",
3.        "instruction": "MODIFICAVARIABLE",
4.        "valueFrom": { }
5.      }
```

Explicación de la estructura JSON:

identifier: Es el nombre de la variable a modificar.
valueFrom: Contiene la misma estructura que la definición anterior, con excepción del campo “instruction”.

1.7 Instrucciones con parámetros de entrada

Existen diversas instrucciones que pueden tener variables de entrada como: Números aleatorios, números naturales, operaciones aritméticas o pueden ir vacías.

Código de Ejemplo del lenguaje:

```
1.        avanzar();
2.        avanzar(random(7,98));
3.        avanzar(aleatorio());
4.        move(5);
5.        avanzar(id);
6.        move(5+78);
7.        move(5+78+(id+96));
8.
9.        espera();
10.       espera(random(7,98));
11.       espera(aleatorio());
12.       wait(5);
13.       espera(id);
14.       wait(5+78);
15.       espera(5+78+(id+96));
16.
17.       turn_left();
18.       turn_left(random(7,98));
19.       turn_left(aleatorio());
20.       gira_izquierda(5);
21.       turn_left(id);
22.       gira_izquierda(5+78);
23.       turn_left(5+78+(id+96));
```

JSON de respuesta:


```
1.  {
2.  "instruction":"AVANZAR",
3.  "parameter":{ }
4.  },
5.  {"instruction":"ESPERA","parameter":{"identifier":"","type":0,"value":-1,"valueFrom":{}}},
6.  {"instruction":"IZQUIERDA","parameter":{"identifier":"","type":0,"value":-1,"valueFrom":{}}},
7.  {"instruction":"DERECHA","parameter":{"identifier":"","type":0,"value":-1,"valueFrom":{}}}
```


Explicación de la estructura JSON:

parameter: contiene información sobre lo que hay dentro del paréntesis de la instrucción. Esta estructura se definirá a continuación.

1.8 Parámetros de entrada de una instrucción

Ahora se definirá cuáles son las instrucciones que puede tener como parámetro la instrucción anterior.

Código de Ejemplo del lenguaje:

```
1.        nombreDeLaInstrucción();
2.        nombreDeLaInstrucción (random(7,98));
3.        nombreDeLaInstrucción (5);
4.        nombreDeLaInstrucción (id);
```

JSON de respuesta:

```
1.        {
2.          "identifier": "",
3.          "type": 0,
4.          "value": -1,
5.          "valueFrom": { }
6.        }
```

Explicación de la estructura JSON:

type: Es el tipo de parámetro que se ha leído.

- 0: No tiene valor, por lo que la instrucción solo se debe ejecutar 1 vez.
- 1: El valor de pase ha sido del valor de una variable.
- 2: El valor de pase ha sido un valor numérico.
- 3: El valor de pase ha sido un número aleatorio
- 4: El valor de pase ha sido una operación aritmética.

identifier: Si la instrucción es type: 1, aquí aparecerá el nombre de la variable.
value: Si la instrucción es type: 2, aquí aparecerá el valor numérico.
valueFrom: Si la instrucción es de type: 3 o type: 4, aquí aparecerá la estructura para resolver un valor aleatorio o una operación aritmética según sea el caso.

1.9 Incrementos a una variable

Si se desea solo sumar un elemento a una variable, se puede utilizar esta instrucción. Es útil para no utilizar una asignación de variable cada vez que se quiera sumar solo 1 elemento.

Código de Ejemplo del lenguaje:

```
1.        id12++;
2.        id48--;
```

JSON de respuesta:

```
1.  {
2.  "identifier":"id12",
3.  "instruction":"INCREMENTAVARIABLE"
4.  },
5.  {
6.  "identifier":"id48",
7.  "instruction":"DECREMENTAVARIABLE"
8.  }
```

Explicación de la estructura JSON:

identifier: Muestra el identificador dónde se realizará la modificación del valor.

1.10 Llamadas a función

Las llamadas a función pueden ser ejecutadas en cualquier momento del programa fuente, estos pueden tener parámetros o ninguno.

Código de Ejemplo del lenguaje:

```
1.        llamadaAFuncion();
2.        llamadaAFuncion(id, random(),id+5);
```

JSON de respuesta:

```
1.      {
2.        "instruction": "LLAMADAAFUNCION",
3.        "parameter": [
4.          {},{},{},{} … {},{},{}
5.        ],
6.        "name": "llamadaAFuncion"
7.      }
```

Explicación de la estructura JSON:

name: El nombre de la función invocada.
parameter: Contiene elementos en forma de lista de los parámetros de entrada, estos están ordenados en el mismo orden que el usuario escribió

1.11 Imprimir concatenaciones entre variables y cadenas de texto

A diferencia de la instrucción para imprimir variables, esta instrucción se debe ocupar si en algún momento se desea crear cadenas que concatenen cadenas de texto y valores de alguna variable, o simplemente para imprimir cadenas o variables. No puede imprimir valores numéricos o valores aleatorios.

Código de Ejemplo del lenguaje:

```
1.        print (id45);
2.        imprimir("Cadena: ");
3.        print (id46+"El ID es ");
4.        imprimir("Valor: "+id48);
5.        print (id49+" es distinto de "+ id50);
6.        imprimir("Resultado: "+id51+ " final");
```

JSON de respuesta:

```
1.      {
2.        "instruction": "IMPRIMIRCADENA",
3.        "parameter": [
4.          {},{},{},{} … {},{},{}
5.        ]
6.      }
```

Explicación de la estructura JSON:

parameter: Se encuentra un arreglo de objetos dónde se encuentran los valores que el usuario escribió en el mismo orden. A continuación, se detallan esos parámetros que puede tener.

1.12 Valores que puede regresar un parámetro

Esta es la estructura que podrá aparecer en los parámetros anteriores:

Código de Ejemplo del lenguaje:

```
imprimir("Resultado: "+id51+ " final");
```

JSON de respuesta:

```
{
  "string": "",
  "type": 0,
  "valueFrom": "id51"
}
```

Explicación de la estructura JSON:

type: Es el tipo de valor que el usuario ingresó, para esta parte solo hay 2 tipos.

- 0: Se trata del valor de un identificador.
- 1: Se trata de una cadena de texto.

string: Si la instrucción es type: 0, aquí se encontrará la cadena que el usuario indicó.
valueFrom : Si la instrucción es type: 1, aquí se encontrará la variable que el usuario indicó.

1.13 Comparaciones lógicas y de valores

A partir de aquí se comenzarán a utilizar con frecuencia las comparaciones. Hay que recordar que existen 2 tipos de comparaciones. Las lógicas solo evaluarán las de tipo and y or. Las comparaciones de valor con las de tipo: mayor que, menor que, distinto a, o igual (con sus derivaciones como menor igual que, y mayor igual que). Además, se tiene un operador de negación que vuelve un resultado verdadero a falso y viceversa.

Código de Ejemplo del lenguaje:

```
estructura(id<78){ }
estructura(!(id<78)){ }
estructura((id<78) or !(random()==25)){ }
```

JSON de respuesta:

```
{
      "identifier":"id",
      "type":0,
      "value":true,
      "valueFrom":{},
      "compareWith",{}
      "isBracket": true,
}
{
      "identifier":"id",
      "type":0,
      "value":0,
      "operator":0,
      "valueFrom":{},
      "compareWith",{}
      "isBracket" false,
}
```

Explicación de la estructura JSON:

type: El tipo de dato que se está analizando en ese momento.

- 0: El valor es un valor booleano (Verdadero o Falso).
- 1: El valor proviene de un identificador.
- 2: El valor es un número.
- 3: El valor se genera de un número aleatorio.
- 4: Es una comparación de paréntesis.
- 5: Comparación lógica

identifier: Si la instrucción es type: 1 aquí se encontrará el nombre de la variable para comparar.

value:

Si la instrucción es type: 2 aquí se encontrará el valor del número

Si la instrucción es type: 4 aquí se encontrará un valor booleano. Si es verdadero el resultado final dentro de ese paréntesis se debe negar. Es decir, si el resultado es verdadero se tendrá que convertir a falso, y viceversa.

valueFrom:

- Si la instrucción es type: 4 aquí se encontrará la evaluación de la expresión de valores.
- Si la instrucción es de type: 5 aquí se encontrará la evaluación de la expresión de valores que se encuentra a la izquierda del operador lógico.

compareWith:

Si la instrucción es type: 1, type: 2, type: 3 aquí se encontrará la estructura con la cual evaluar la expesión.

- Si la instrucción es de type: 5 aquí se encontrará la evaluación de la expresión de valores que se encuentra a la derecha del operador lógico.
- operator: Solo aparecerá si se encuentra dentro de un compareWith, en esta expresión se encontrará el operador de comparación para ese nivel.

isBracket: Solo aparece como verdadero si se ha iniciado la evaluación de una expresión dentro de un paréntesis.

1.14 Estructuras de Control selectivas

Las estructuras de control selectivas valoran una condición para poder ser ejecutadas. Existen 4 tipos de estructuras, if simple, doble if else, if else if anidadas y estructuras case. Observando cómo quedará para el lenguaje cada una de ellas.

1.14.1 Estructura if simple

Código de Ejemplo del lenguaje:

```
1.        if(id<78){
2.                  llamadaAFuncion();
3.                  llamadaAFuncion(id);
4.        }
5.
6.        if(true){
7.                  wall_in_front();
8.                  tengo_muro_delante();
9.        }
```

JSON de respuesta:

```
1.  {
2.  "condition":{ },
3.  "subprogram":[],
4.  "instruction":"SI",
5.  "continue":[]
6.  }
```

Explicación de la estructura JSON:

condition: Aquí se encuentra la estructura que se vieron en el subtema pasado.
subprogram: Aquí se encuentra una lista que contiene las instrucciones del subprograma a ejecutar en caso de que la condición resulte ser verdadera.
continue: Aquí se encontrará una lista con el resto de instrucciones, para el caso de if simple siempre estará vacío. **Es la única estructura que está acomodada al revés, es decir, el siguiente elemento a evaluar estará al final de la lista**.

1.14.2 Estructura if-else

Esta estructura tiene la característica de realizar el último bloque de código si la condición del if simple no se cumple.

Código de Ejemplo del lenguaje:

```
1.        if(id<78){
2.                  llamadaAFuncion();
3.                  llamadaAFuncion(id);
4.        }else{
5.                  wall_in_front();
6.                  tengo_muro_delante();
7.        }
```

JSON de respuesta:

```
1.  {
2.  "subprogram":[],
3.  "instruction":"SINO"
4.  }
```

Explicación de la estructura JSON:

Esta estructura solo tendrá información en subprogram que contendrá lo mismo que el ejemplo anterior.
El resto de caracteres estarán en blanco.

1.14.3 Estructura if-else anidadas

Estas estructuras evalúan otros casos y de no cumplirse podrán ejecutar el último bloque “else” (Sin condición) en caso de haber sido escrito.

Código de Ejemplo del lenguaje:

```
1.  si(id<78){
2.                  wait(1);
3.        }else_if(true){
4.                  wait(2);
5.        }sinosi(random() <= aleatorio(7,9)){
6.                  wait(3);
7.        }else_if(!(id<78)){
8.                  wait(4);
9.        }sinosi((true)){
10.                 wait(5);
11.       }else_if(!(random() <= aleatorio(7,9))){
12.                 wait(6);
13.       }else{
14.                 wait(7);
15.       }
```

JSON de respuesta:

```
1.  {
2.  "condition":{ },
3.  "subprogram":[],
4.  "instruction":"SI",
5.  }
```

Explicación de la estructura JSON:

Contiene las mismas características que el caso if, salvo que no se encuentra el valor de continue.

1.15 Estructura case

La estructura case compara una variable con distintos casos. Para el caso del lenguaje solo se puede comparar una variable con un número natural, además hay que recordar que es obligado para el programador del lenguaje agregar el caso por "*default*".

Código de Ejemplo del lenguaje:

```
1.      comparar(id){
2.              case 7:
3.                      wait(5);
4.                      move(1);
5.                      tengo_muro_delante();
6.              case <7:
7.                      move(2);
8.                      wall_in_front();
9.                      tengo_muro_delante();
10.                     wait(4);
11.                     end;
12.             case random():
13.                     wall_in_front();
14.                     wait(3);
15.                     tengo_muro_delante();
16.                     move(3);
17.                     fin;
18.             case aleatorio(7,12):
19.                     wall_in_front();
20.                     move(4);
21.                     tengo_muro_delante();
22.                     wait(2);
23.                     fin;
24.             default:
25.                     wall_in_front();
26.                     move(5);
27.                     tengo_muro_delante();
28.                     wait(1);
29.                     fin;
30.     }
```

JSON de respuesta:

```
1.  {
2.  "identifier":"id",
3.  "default":{ },
4.  "cases":{ },
5.  "instruction":"COMPARAR"
6.  }
```

Explicación de la estructura JSON:

identifier: contendrá el nombre de la variable de la cual se estará comparando.
cases: contendrá todos los casos en las que se comprueba la variable (se detallará las características de este elemento en el siguiente subtema).
default: es el caso por default de la estructura, siempre deberá estar ahí.

1.15.1 Estructura de comparación case

Código de Ejemplo del lenguaje:

```
case 7:
        wait(5);
        move(1);
        tengo_muro_delante();
case <7:
        move(2);
        wall_in_front();
        tengo_muro_delante();
        wait(4);
        end;
case random():
        wall_in_front();
        wait(3);
        tengo_muro_delante();
        move(3);
        fin;
```

JSON de respuesta:

```
{
"condition":{ },
"subprogram":[],
"instruction":"CASO",
"continue":{ },
,"end":true
}
```

Explicación de la estructura JSON:

condition: Es la condición del caso a comparar.
subprogram: Contendrá las instrucciones a ejecutar en caso de que la condición sea verdadera.
end: Contiene un valor booleano. Si es verdadero indica que en caso de ser veradera la condición se dejará de evaluar los casos. Si es falso indica que se puede continuar leyendo los siguientes casos para ejecutar sus respectivos subprogramas.
continue: Contiene el siguiente caso en caso de resultar negativa la condición o en caso de que se haya cumplido la función pero la llave “end” tenga valor falso.

1.15.2 Estructura default de case

Código de Ejemplo del lenguaje:

```
default:
        wall_in_front();
        move(5);
        tengo_muro_delante();
        wait(1);
        fin;
```

JSON de respuesta:

```
{
"subprogram":[],
"instruction":"DEFAULT",
,"end":true
}
```

Explicación de la estructura JSON:

Las mismas variables que el caso anterior, salvo que en este caso no existe parámetro de comparación o de continuación.

1.16 Estructuras de control iterativas

Las estructuras de control iterativas son aquellas instrucciones que se repetirán hasta que se cumpla una instrucción en específico. Existen 3 tipos de estas estructuras, la estructura for, do while y while.

1.16.1 Estructura For

Para la estructura for se divide en algunas partes: La declaración o inicialización de variables, una condición, una parte para aumentar o disminuir contadores cada que termina de ejecutarse un bloque de código principal y la parte de un bloque principal.

Código de Ejemplo del lenguaje:

```
1.        para(var i<-0, id<-(2*random()) ; i<25 ; i++, a--){
2.                  wall_in_front();
3.                  tengo_muro_delante();
4.        }
```

JSON de respuesta:

```
1.  {
2.  "instructions":[ ],
3.  "condition":{ },
4.  "instruction":"FOR",
5.  "declarations":[ ],
6.  "increments":[ ]
7.  }
```

Explicación de la estructura JSON:

declarations: Contendrá las declaraciones o modificaciones de variables iniciales de la estructura for. Solo se ejecutan 1 vez.
condition: Es la condición que se evalúa en cada iteración.
instructions: Son las instrucciones que se ejecutarán cada que termina la evaluación de la condición (Solo si resulta verdadera).
increments: Son incrementos o decrementos de variables. Cuando termina de ejecutar las instrucciones estas se ejecutarán. Solo se pueden realizar estas instrucciones en este espacio.

1.16.2 Estructura do while

Esta estructura ejecuta el código de usuario una sola vez y para continuar ejecutándose a sí misma deberá evaluar una condición.

Código de Ejemplo del lenguaje:

```
1.        hacer{
2.                  wall_in_front();
3.                  tengo_muro_delante();
4.        }while(falso);
```

JSON de respuesta:

```
1.  {
2.  "instructions":[],
3.  "condition":{ },
4.  "instruction":"HACER"
5.  }
```

Explicación de la estructura JSON:

instructions: Contiene las instrucciones a ejecutar.
condition Contiene la condición que se evaluará en cada iteración.

1.16.3 Estructura While

La estructura while evalúa desde el principio la condición para saber si ejecutarse o no.

Código de Ejemplo del lenguaje:

```
1.        repite_hasta(verdadero){
2.                  wall_in_front();
3.                  tengo_muro_delante();
4.        }
```

JSON de respuesta:

```
1.  {
2.  "instructions":[],
3.  "condition":{ },
4.  "instruction":"REPITEHASTA"
5.  }
```

Explicación de la estructura JSON:

instructions: Contiene las instrucciones a ejecutar.
condition Contiene la condición que se evaluará en cada iteración.

1.17 Funciones

Las funciones son una parte importante de este proyecto. Estas van declaradas por fuera y después de la función principal *main*.

Código de Ejemplo del lenguaje:

```
1.  funcionSimple(){
2.        repite_hasta(verdadero){
3.                  wall_in_front();
4.                  tengo_muro_delante();
5.        }
6.  }
7.
8.  funcionConParametros(var id, var other, var right){
9.        funcionSimple();
10.       turn_right(id);
11.       gira_derecha(other);
12.       turn_right(right);
13. }
```

JSON de respuesta:

```
1.  {
2.  "instructions":[],
3.  "instruction":"FUNCION",
4.  "parameter":[]
5.  }
```

Explicación de la estructura JSON:

parameter: Contiene los parámetros de entrada de la función, es como una declaración de variables.
instructions: Contiene todas las instrucciones que podrá ejecutar la función.

1.18 Parámetros de una función

Por último, se declara la estructura que tendrán los parámetros de una función. Estos parámetros deberán coincidir en número y en posición cuando se llama a una función.

Código de Ejemplo del lenguaje:

```
1.  funcionConParametros(var id, var other, var right){
2.  }
```

JSON de respuesta:

```
1.  {
2.  "identifier":"id",
3.  "instruction":"DECLARAFUNCION"
4.  }
```

Explicación de la estructura JSON:

identifier: Contiene el nombre de la variable que se podrá utilizar en la función.