



Universidad de La Habana
Facultad de Matemática y Computación

PROYECTO DE COMPILACIÓN + INTENTELEGENCIA
ARTIFICIAL + SIMULACIÓN

SIMULADOR DE UN JEFE TÉCNICO DE MOTOGP

Autores:

Arnel Sánchez Rodríguez Grupo: C312
arnelsanchezrodriguez@gmail.com

Samuel Efraín Pupo Wong Grupo: C312
s.pupo@estudiantes.matcom.uh.cu

Darián Ramón Mederos Grupo: C312
darianrm24@gmail.com

2021-2022

Índice

1. Motociclismo de Velocidad	2
2. Estructura del Paddock	2
3. Pista	3
4. Definición del Problema	3
5. Definición del Lenguaje	4
5.1. Introducción a PSharp (P)	5
5.1.1. Hello world!	5
5.1.2. Estructura del Programa	5
5.1.3. Tipos y Variables	5
5.1.4. Expresiones	6
5.1.5. Declaraciones	7
5.1.6. Tipos	9
5.1.7. Métodos	9
5.1.8. Parámetros	10
5.1.9. Cuerpo del método y variables locales	10
5.1.10. Operadores	10
5.1.11. Matrices	10
5.1.12. Análisis Léxico	11

1. Motociclismo de Velocidad

El motociclismo de velocidad es una modalidad deportiva del motociclismo disputada en circuitos de carreras pavimentados. Las motocicletas que se usan pueden ser prototipos, es decir desarrolladas específicamente para competición, o derivadas de modelos de serie (en general motocicletas deportivas) con modificaciones para aumentar las prestaciones. En el primer grupo entran las que participan en el Campeonato Mundial de Motociclismo, y en el segundo las Superbikes, las Supersport y las Superstock.

Las motocicletas deben presentar una serie de características como son estabilidad, alta velocidad (tanto en recta como en paso por curva), gran aceleración, gran frenada, fácil maniobrabilidad y bajo peso.



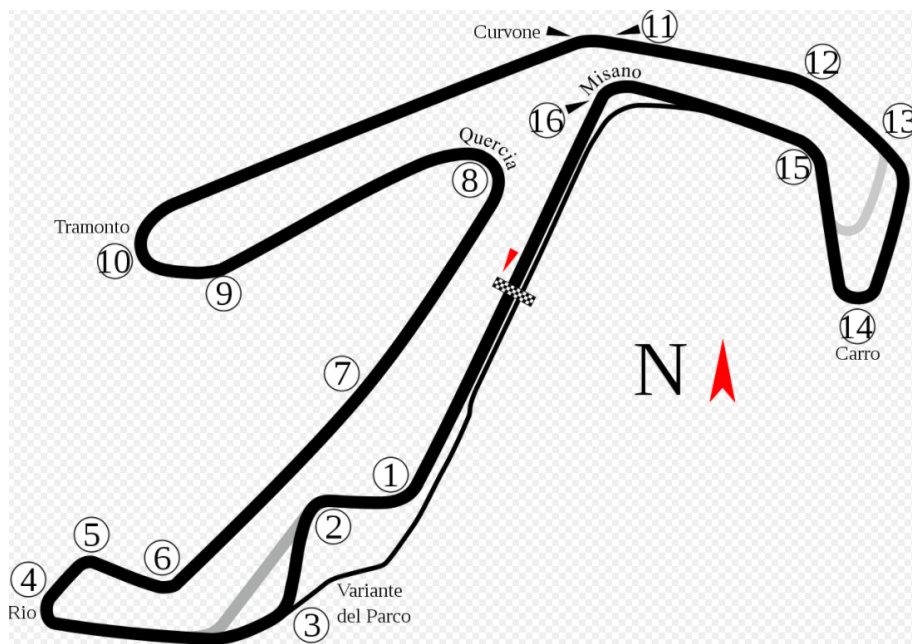
2. Estructura del Paddock

El **Jefe Técnico** de cada estructura se configura como una personalidad de bastante importancia dentro de un box, pues es quién se encarga de dirigir y controlar que todo funcione como un excelente engranaje que gane carreras. De igual importancia es la telemetría dentro de un box en MotoGP. Al fin y al cabo, los **Ingenieros Telemétricos** son las personas que se encargan de analizar, leer y comprender todos los datos proporcionados por el piloto, así como transmitírselos en boca al protago-

nista. Se trata de una figura de la que depende mucha de la información acerca de cualquier cambio realizado en la moto o asumir los puntos más fuertes de sus pilotos. Los **Mecánicos** también desempeñan un papel fundamental a la hora de construir la máquina perfecta.

3. Pista

Se utilizará como referencia el circuito de Misano, Misano World Circuit Marco Simoncelli, autódromo localizado en la fracción de Santa Mónica, comuna de Misano Adriático (provincia de Rímini), región de Emilia-Romaña, Italia.



4. Definición del Problema

Existirán varios pilotos con sus respectivas motos, las cuáles difieren entre sí en cuanto a sus prestaciones. Cada piloto posee su propio método de manejo, siendo algunos más cuidadosos y otros más agresivos. La pista se encuentra influenciada por el accionar del clima, puesto que no es lo mismo el manejo durante un día soleado que bajo la lluvia. Por tanto, el resultado de un piloto se verá condicionado por su moto, su modo de conducción y el clima.

Sin embargo, durante la carrera las condiciones pueden variar y el Jefe Técnico será el encargado de señalar los ajustes necesarios que el piloto deberá hacer para mejorar su rendimiento. Dicho intercambio de información se hará al finalizar cada vuelta utilizando un lenguaje imperativo, mediante el uso de palabras claves para que el piloto no necesite analizar situaciones complejas y pueda concentrarse en pilotar de la forma mas eficiente posible.

De esta manera, la simulación de la carrera será dinámica, puesto que entre las vueltas podrán existir variaciones provocadas por los ajustes propuestos por el Jefe Técnico, el cuál podrá ser una persona o una IA.

5. Definición del Lenguaje

5.1. Introducción a PSharp (P)

5.1.1. Hello world!

```
1  method void main() {  
2      print("Hello World!");  
3  }
```

Los archivos de P# suelen tener la extensión de archivo .pys.

P# permite almacenar el texto fuente de un programa en varios archivos fuente. Cuando se compila un programa P# de varios archivos, todos los archivos de origen se procesan juntos y los archivos de origen pueden hacer referencia libremente entre sí; conceptualmente, es como si todos los archivos de origen estuvieran concatenados en un archivo grande antes de ser procesados.

5.1.2. Estructura del Programa

Los conceptos organizativos clave en P# son programas, tipos y miembros. Los programas P# constan de uno o más archivos fuente. Los programas declaran tipos y miembros. Las motocicletas y los motociclistas son ejemplos de tipos. Los métodos y propiedades son ejemplos de miembros.

5.1.3. Tipos y Variables

Hay dos tipos de tipos en P#: tipos de valor y tipos de referencia. Las variables de tipos de valor contienen directamente sus datos, mientras que las variables de tipos de referencia almacenan referencias a sus datos. Con los tipos de referencia, es posible que dos variables hagan referencia al mismo objeto y, por lo tanto, las operaciones en una variable afecten al objeto al que hace referencia la otra variable. Con los tipos de valor, cada una de las variables tiene su propia copia de los datos, y no es posible que las operaciones en una afecten a la otra.

Categoría	Tipo	Descripción
Tipos por Valor	Tipos Simples	Entero con signo: int
		Punto flotante IEEE: double
		Booleanos: bool
		Cadenas Unicode: string
	Tipos que aceptan valores NULL	Extensiones de todos los demás tipos de valor con un valor nulo
Tipos por Referencia	Tipos de matrices	Unidimensionales, por ejemplo, int []

5.1.4. Expresiones

Las expresiones se construyen a partir de operandos y operadores. Los operadores de una expresión indican qué operaciones aplicar a los operandos. Los ejemplos de operadores incluyen +, -, * y /. Los ejemplos de operandos incluyen literales, variables y expresiones.

Categoría	Expresión	Descripción
Primaria	x(...)	Invocación de método
	x[...]	Acceso a matrices e indexadores
Unaria	+x	Identidad
	-x	Negación
	!x	Negación lógica
Multiplicativa	x * y	Multiplicación
	x / y	División
	x % y	Resto
	x ** y	Exponenciación
Aditiva	x + y	Adición y concatenación de strings
	x - y	Substracción
Relacionales	x < y	Menor que
	x > y	Mayor que
	x <= y	Menor o igual que
	x >= y	Mayor o igual que
Igualdad	x == y	Igual
	x != y	Distinto
Condicionales AND	x && y	Evalúa y si y sólo si x es verdadera
Condicionales OR	x y	Evalúa y si y sólo si x es falsa
Condicionales XOR	x ^ y	
Asignación	x = y	Asignación
	x op= y	Asignación compuesta; los operadores admitidos son *= /= %= **= += -= &&= = ^=

5.1.5. Declaraciones

Las acciones de un programa se expresan mediante declaraciones. P# admite varios tipos diferentes de declaraciones, algunas de las cuales se definen en términos de declaraciones integradas.

Un **bloque** permite escribir múltiples declaraciones en contextos donde se permite una sola declaración. Un bloque consta de una lista de declaraciones escritas entre

los delimitadores{y}.

La **declaración de inclusión** se utiliza para obtener y luego disponer de un recurso.

```
1 include test;
```

Las sentencias de **declaración** se utilizan para declarar variables, valga la redundancia.

```
1 method void example() {  
2     int a = 1;  
3 }
```

Las **declaraciones de expresión** se utilizan para evaluar expresiones. Las expresiones que se pueden usar como declaraciones incluyen invocaciones de métodos, asignaciones que usan = y los operadores de asignación compuesta.

```
1 method void example() {  
2     int a = 1;  
3     print(a + 2);  
4 }
```

La **instrucción de selección** se utiliza para seleccionar una de varias declaraciones posibles para su ejecución en función del valor de alguna expresión. Este es el caso de la sentencia **if**.

```
1 method void example(int a) {  
2     if (a < 5) {  
3         print(a);  
4     }  
5     else {  
6         print(a % 5);  
7     }  
8 }
```

La **instrucción de iteración** se utiliza para ejecutar repetidamente una instrucción incorporada. Este es el caso de la instrucción **while**.

```
1 method void example(int a) {  
2     while (a > 5) {  
3         print(a);  
4         a -= 1;  
5     }  
6 }
```

Las **sentencias de salto** se utilizan para transferir el control. En este grupo están las declaraciones de **break**, **continue** y **return**.

```

1  method int example() {
2      while (true) {
3          x = input()
4          if (x == "x") {
5              continue;
6          }
7          elif (x == "") {
8              break;
9          }
10         print(x);
11     }
12     return 0;
13 }

```

5.1.6. Tipos

Los **tipos** son los elementos más importantes de P#. Un tipo es una estructura de bloques compuesta por acciones (métodos). Un tipo proporciona una definición para casos de, por ejemplo, motociclistas o motocicletas. Su declaración comienza con un encabezado que especifica qué tipo se va a crear y el nombre que se le dará a esta instancia. El encabezado va seguido del cuerpo del tipo, que consiste en una lista de declaraciones de miembros escritas entre los delimitadores{y}.

```

1  rider Rossi() {
2      method int example() {
3          return 0;
4      }
5
6      method void curves() {
7          ...
8      }
9      ...
10 }

```

5.1.7. Métodos

Un **método** es un miembro que implementa un cálculo o acción que se puede realizar por tipo. Los métodos tienen una lista (posiblemente vacía) de **parámetros**, que representan valores o referencias de variables pasadas al método, y un tipo de retorno, que especifica el tipo de valor calculado y devuelto por el método. El tipo de retorno de un método es nulo si no devuelve un valor.

5.1.8. Parámetros

Los **parámetros** se utilizan para pasar valores o referencias de variables a métodos. Los parámetros de un método obtienen sus valores reales de los **argumentos** que se especifican cuando se invoca el método. Las modificaciones de un valor de parámetro no afectan el argumento que se pasó para el parámetro.

5.1.9. Cuerpo del método y variables locales

El cuerpo de un método especifica las declaraciones que se ejecutarán cuando se invoca el método. El cuerpo de un método puede declarar variables que son específicas de la invocación del método. Estas variables se denominan variables locales. Una declaración de variable local especifica un nombre de tipo, un nombre de variable y un valor inicial.

5.1.10. Operadores

Un **operador** es un miembro que define el significado de aplicar un operador de expresión particular. Se pueden definir tres tipos de operadores: operadores unarios y operadores binarios.

Operadores Unarios:

```
1    !true;
```

Operadores Binarios:

```
1    3+5;  
2    true && false;
```

5.1.11. Matrices

Una matriz es una estructura de datos que contiene una serie de variables a las que se accede a través de índices calculados. Las variables contenidas en una matriz, también llamadas elementos de la matriz, son todas del mismo tipo, y este tipo se denomina tipo de elemento de la matriz. Los tipos de matriz son tipos de referencia y la declaración de una variable de matriz simplemente reserva espacio para una referencia a una instancia de matriz. Los índices de los elementos de una matriz van de 0 a longitud - 1.

```
1    int[] a = [1, 2, 3, 4, 5];
```

5.1.12. Análisis Léxico

```
input
: input_element* new_line
| directive
;
input_element
: whitespace
| comment
| token
;
Terminadores de línea
new_line
: '<Caracter de retorno (U+000D)>'
| '<Caracter de avance de línea (U+000A)>'
;
whitespace
: '<Cualquier personaje con clase Unicode Zs>'
| '<Caracter de tabulación horizontal (U+0009)>'
;
Comentarios
comment
: '#' comment_section '#'
;
Tokens
token
: identifier
| keyword
| literal
| operator_or_punctuator
;
Identificadores
identifier
: '<Un identificador que no es una palabra clave>'
| identifier_start_character identifier_part_character*
;
identifier_start_character
: letter_character
| '<Caracter guión bajo (U+005F)>'
```

```

;
identifier_part_character
: letter_character
| decimal_digit
| '<Caracter gui3n bajo (U+005F)>'
;
letter_character
: uppercase_letter_character
| lowercase_letter_character
;
uppercase_letter_character
: 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H'
| 'I' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q'
| 'R' | 'S' | 'T' | 'V' | 'X' | 'Y' | 'Z'
;
lowercase_letter_character
: 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h'
| 'i' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q'
| 'r' | 's' | 't' | 'v' | 'x' | 'y' | 'z'
;
decimal_digit
: '0' | '1' | '2' | '3' | '4'
| '5' | '6' | '7' | '8' | '9'
;
Palabras Claves
keyword
: 'bool'
| 'break'
| 'continue'
| 'double'
| 'elif'
| 'else'
| 'false'
| 'if'
| 'include'
| 'int'
| 'method'
| 'null'

```

```

| 'return'
| 'string'
| 'true'
| 'void'
| 'while'
| 'motorcycle'
| 'rider'
| 'track'
| 'weather'
| 'angle'
| 'brakes'
| 'lap'
| 'length'
| 'speed'
| 'temperature'
| 'skill'
| 'tyres'
;

```

Literales

literal

```

: boolean_literal
| integer_literal
| double_literal
| string_literal
| null_literal
;

```

Literales Booleanos

boolean_literal

```

: 'true'
| 'false'
;

```

Literales Enteros

integer_literal

```

: decimal_digit
;

```

Literales flotantes

double_literal

```

: decimal_digit+ ' ' decimal_digit+

```

```

;
Literales de Cadenas
string_literal
: `` string_literal_character* ``
;
string_literal_character
: '<Cualquier caracter, excepto "(U+0022)';
;
Literales Nulos
null_literal
: 'null'
;
Operadores y signos de puntuación
operator_or_punctuator
: '{'
| '}'
| '['
| ']'
| '('
| ')'
| '.'
| ','
| ':'
| ';'
| '+'
| '-'
| '*'
| '/'
| '%'
| '**'
| '!'
| '='
| '<'
| '>'
| '&&'
| '||'
| '=='
| '!='

```

```

| '<='
| '>='
| '+='
| '-='
| '*='
| '/='
| '%='
| '**='
| '&&='
| '||='
| '^='
;
Directivas
directive
: 'include' '<Nombre_del_archivo.pys>' ';'
;

```