Dice: A Programming Language Manual

Prepared by Zach Healy

November 29, 2023

Table of Contents

1. Language Overview

1.1 Introduction to the Language

This is a programming language themed around the ideas and words often associated with gambling. The keywords all have some sort of relation back to one of many various games seen in casinos such as slots, blackjack, poker, and craps. After working all semester to perfect our craps abilities, we thought this would be a fitting end to the semester. This language is designed to be a simple and readable language, without compromising on functionality.

1.2 Discussion of Influences

Our language will be heavily influenced by Python in terms of the simplicity in variable definition and format of various things such as loops. We were inspired also by the readability and relative simplicity of the language, which influences our choice for the language as a whole. We want our language to not only be readable and easy to understand, but also have its own fun spin on languages. So, for inspiration for the language itself, after working with building Craps games all semester, we thought that this would be a fun way to capstone the experience and make it all come full circle with a language centered around casino themed elements. We used keywords such as ‘buyin’ and ‘cashout’ in order to fittingly signify the start and end of a program. The ‘buyin’ kicks off the program. Since running each program is a ‘gamble’ as to whether or not it will run or not, the ‘cashout’ reward is a functional working program. We also decided to implement keywords such as ‘shoot’ and ‘place’ to print and declare variables. Initially we also thought keywords such as ‘spin’ or ‘roll’ would be fun identifiable keywords for loops, however in practice we found it often compromised the readability of the language more than we had wanted.

1.3 BNF Description of Syntax

<program> ::= "buyin" <block> "cashout"

<block> ::= <statement>+

<statement> ::= <assignment>
| <expression>
| <return-statement>
| <control-statement>
| <function-declaration>

<assignment> ::= "place" <variable> "=" <expression>

<expression> ::= <term>
| <expression> "+" <term>
| <array>

<term> ::= <factor>
| <term> "*" <factor>

<factor> ::= <number>
| <string>
| <variable>
| "(" <expression> ")"
| <array>

<return-statement> ::= "return" <expression>

<control-statement> ::= <if-statement>

| <while-loop>

| <for-loop>

<function-declaration> ::= "sitdown" <id> "(" [<parameter-list>] ")" <block> "standup"

<print-statement> ::= "shoot" <expression>

<id> ::= [a-zA-Z]+

<parameter-list> ::= <id> {"," <id>}

<if-statement> ::= "if" <expression> "then" <block> ["else" <block>] "endif"

<while-loop> ::= "while" <expression> "do" <block> "endwhile"

<for-loop> ::= "for" <assignment> "to" <expression> "do" <block> "endfor"

<variable> ::= "place" <id> | <array>

<array> ::= "[" [<expression-list>] "]"

<array-index> ::= <variable> "[" <expression> "]"

<expression-list> ::= <expression> {"," <expression>}

<number> ::= [0-9]+ | [0-9]+ "." [0-9]+

<string> ::= "\"" <characters> "\""

<characters> ::= <character>*

<character> ::= [a-zA-Z0-9_]

<whitespace> ::= " " | "\t" | "\n" | "\r"

2. Example Programs

2.1 Hello World

```
buyin
    shoot “Hello World”
cashout
```

2.2 Bubble Sort

```
buyin
 sitdown bubbleSort(arr)
  place n = arr.length
  place swapped = true

  while swapped do
   place swapped = false

   for place i = 0 to n - 2 do
    if arr[i] > arr[i + 1] then
     place temp = arr[i]
     place arr[i] = arr[i + 1]
     place arr[i + 1] = temp
     place swapped = true
    endif
   endfor
  endwhile

  return arr
 standup

 place myArray = [5, 2, 9, 1, 5, 6]

 place sortedArray = bubbleSort(myArray)
 shoot "Sorted Array: " + sortedArray
cashout
```

2.3 Tower of Hanoi Solver

```
buyin
 sitdown hanoi(n, source, target, auxiliary)
  if n > 0 then
    hanoi(n - 1, source, auxiliary, target)
    shoot “Move disk “ + n + “ from “ + source + “ to “ + target
    hanoi(n – 1, auxiliary, target, source)
  endif
 standup

 place numDisks = 3
 place sourcePeg = “A”
 place targetPeg = “C”
 place auxiliaryPeg = “B

 hanoi(numDisks, sourcePeg, targetPeg, auxiliaryPeg)
cashout
```

3. Language Reference Manual

3.1 Operator Precedence Chart

Operator	Explanation
Buyin	Program start
Cashout	Program end
Sitdown	Function definition
Standup	Function ending
Place	Variable declaration
Shoot	Print Statement
For, while if	Start of each loop respectively
Endfor, endwhile, endif	End of each loop respectively

3.2 Operator Explanations

3.2.1 – buyin

- A keyword used to start the program
- Example:

```
buyin
    shoot 'Hello World'
cashout
```

3.2.2 – cashout

- A keyword used to end the program
- Example:

```
buyin
    shoot 'Hello World'
cashout
```

3.2.3 – sitdown

- A keyword used to declare a function
- Usage: sitdown functionName()
- Example:

```
sitdown helloWorld()
    shoot 'Hello World'
standup
```

3.2.4 – standup

- A keyword used to end a function
- Usage: standup
- Example:
  sitdown helloWorld()
      shoot ‘Hello World’
  standup

3.2.5 – place

- A keyword used to declare variables
- Example:
  buyin
      place sentence = ‘Hello World’
  cashout

3.2.6 – if/endif

- A set of keywords used to start and end if statements
- Example:
  if state = true
      shoot ‘hello!’
  endif

3.2.7 – for/endfor

- A keyword used as a print statement
- Example:
  for place i = 0 to n – 1 do
      shoot ‘hello!’
  endfor

3.2.8 – while/endwhile

- A keyword used as a print statement
- Example:
  while state = true
      shoot ‘hello!’
      state = false
  endwhile

3.3 Semantic Information

3.3.1 Arrays and Indices

Arrays are declared using the same ‘place’ keyword as a variable would be, then using square brackets to include the elements that the array intakes. To access the individual elements, you’ll use the array name followed by more square brackets that contain the number or identifier for the position you want to index. This is very similar to Python in the sense that no explicit array object needs to be called to create an array. It can just be declared and will be recognized as an array.

3.3.2 Type Coercion Rules

Types are not explicitly stated, rather they are declared using the ‘place’ keyword. Like Python, which is the main inspiration for this language, types are implied based off their format when declared. Should two types be combined, such as integer-char addition, or string-int concatenation, an type error will be thrown. To avoid type errors, there will need to be explicit type conversion at the time of the operation. For example, if a character ‘5’ is added to an integer 2, the programmer will need to type int (‘5’) to cast the character to an integer.

3.3.3 Parameter Passing Information

When passing parameters into a function, following the definition of the function the list will be put into paratheses to be passed in for use in the function. The arguments that are passed in should match the order and type of what is in the function declaration. Should the arguments not match the program will throw an error. If a programmer creates a function that has the parameters (int, string, char), then the arguments passed in must match those types. Any arguments not following these parameter types (i.e. ‘five’, 2, 250), will throw an error and cause the program to cease.

3.3.4 Other Semantic Information

When assigning variables, the ‘=’ operator is used to place the value onto the variable name. The language has support for all sorts of conditionals, such as if statements, while loops, and for loops.