**Drone Language Manual**

**George Brink gb2280**

**Xiaotong Chen xc2230**

**Shuo Qiu sq2144**

**Xiang Yao xy2191**

Drone language is a stack-based imperative language. The stack accepts only integers, booleans, and flags. Integers can be used as arithmetic operands or parameters of the functions. Booleans are subject to stack manipulation operations and as parameter for conditional jump operators. Flags are subject to stack manipulation operations and parameters for special functions which check is the flag is of the expected kind and leave Boolean true or false on the stack.

Language syntax consists of "words" separated by white space. A word is a sequence of arbitrary characters (except white space).

E.g., each of the following lines contains exactly one word:

word

!@#$%^&\*()

1234567890

5!a

Language is case insensitive.

E.g. each of the following words is exactly the same:

Word

wOrd

WORD

WoRd

Each word read from the source code is either comment, integer, boolean call to a user defined function, label, variable, or operator.

Single line comments, start with a word // and continue to the end of the line. E.g. each of the following lines contains a comment

E.g.

// whole line can be a comment

2 2 + // or comment can start after some compilable words

// any word appeared after // is still a comment

Multi line comments, start with a word /\* and continue to the first \*/ word. The nested comments are not supported.

E.g.

/\* Inside here is a comment \*/

Integer is word which consists solely from characters 0-9.

E.g.

123 // one integer

1 2 3 // three integers

These words put the specified integer directly on the stack.

Booleans are two words "true" and "false" which represent the logical values and are subject to logical operations and conditional jumps.

User functions are marked with a word which starts with a "sub" followed by a function name and ends with "esub".

E.g.

sub foo

these words a body of a function

esub

sub add

+

esub

The first example defines a function named "foo" with some arbitrary words and the second example defines an alias "add" for a simple arithmetic operation. The call to the previously defined function is just its name. E.g. the next two lines will do exactly the same:

2 2 +

2 2 add

Functions cannot have sub-functions. For example, the next line shows an illegal code:

sub foo 1 2

sub bar

3 4

esub

esub

Labels are any set of characters ended with colon:

this\_is\_label:

this-is/also.label:

123456:

Of course, the white-space character any sequence of characters into sequence of words and the next line will be understood as four words and a label with the name 'label': and this is not a label:

Operation "unconditional jump to the label" is marked by adding, "jump" to the name. The next three lines are jumps to the labels defined in the previous example:

this\_is\_label jump

this-is/also.label jump

123456 jump

The conditional jump is marked by adding "jump if" the name. Conditional jump happens if and only if the top of the stack contains boolean "true" value. If top of the stack is not a boolean value, then drone freezes.

Labels are visiblity are restricted to the function. For example:

sub foo

2

lbl1: 2 +

lbl1 jump

esub

lbl2: lbl1 jumpif

Here, label lbl1 is defined inside a function foo and jump to it is allowed. The label lbl2 is defined in the main program and jump to it is allowed from any where from the main program, but not from the inside of user defined function. Conversely, the conditional jump to lbl1 will fail since the label is defined inside of the function, but the jump is attempted from the main program.

Variables are words that directly proceeded by keywords "store" or "read". The first one take the top of the stack and stores it into the variable (creating the variable in the process if necessary). The second one reads variable and puts its contents on the stack. For example, if you need to store top value in the variable and leave it on the stack for the future use, you can do this:

E.g.

abc store

abc read

Operators are always taking some number of values from the stack and return some values back on the stack:

E.g.

**Arithmetic operators:**

+ a b -> (a + b)

- a b -> (a - b)

\* a b -> (a \* b)

/ a b -> (a / b)

mod a b -> (a mod b)

^ a b -> (a ^ b)

**Logic operators:**

and a b -> (a and b)

or a b -> (a or b)

not a -> (not a)

**Logic constants**

true -> true

false -> false

**Conditions:**

= a b -> (a = b)

< a b -> (a < b)

> a b -> (a > b)

**Stack manipulation:**

drop a b c -> a b

dropall a b c ->

dup a b c -> a b c c

swap a b c -> a c b

over a b c -> a b c b

rot a b c -> b c a

**Prefixes:**

>name a ->

Store value into variable "name", create the variable if necessary.

<name -> a

Read value from variable "name". Die if such variable does not exist.

:name ->

Start definition of function "name". Function definition ends with single

character ':'.

**Suffixes:**

name: ->

Define a label.

name> ->

Unconditional jump to the label.

name? a ->

Jump to the specified label if top of the stack contains boolean true. Continue execution of the next word if there was false. Die if top of the stack was not a boolean value.

**Game specific operators:**

move direction ->

Start moving in the specified direction

stop ->

Stop moving

shoot distance direction -> bool

Shoot in the specified direction. Projectile will explode after traveling the specified distance. Returns boolean value:

true = shooting was successful and projectile is on its way

false = cannon did not have enough time to cool-down

look direction -> END dist-1 dir-1 type-1 [... dist-n dir-n type-n ]

Look for other drones and walls in the specified direction. Returns one or

more triplets (distance direction type) which represent distance to the object,

the exact direction to the object and type of the object. Type of the object is

a flag from the set: FOE, ALLY, WALL. Under the last triplets there would be

a special flag END.

isFoe flag -> bool

Checks is the top of the stack contains a flag FOE and returns

corresponding boolean value.

isAlly flag -> bool

Checks is the top of the stack contains a flag ALLY and returns

corresponding boolean value.

isWall flag -> bool

Checks is the top of the stack contains a flag WALL and returns

corresponding boolean value.

isEnd flag -> bool

Checks is the top of the stack contains a flag END and returns

corresponding boolean value.

wait milliseconds ->

Do nothing for specified numer of milliseconds.

getHealth -> health

Put current drone's health on the stack

random a b -> r

Make a random integer in the range [a,b] (inclusive) and return it.