U-16
旭川プログラミングコンテスト
資料

目次

はじめに

今日は「U-16　旭川プログラミングコンテスト」で

扱う事になる競技用プログラムについて、

条件分岐や、繰り返しの処理の作り方に加え、

キャラクターを動かすために何を書けば良いのか、

基本の内容をこの資料では説明します。

今回説明する処理を組み合わせて、自分だけの

誰にも負けないプログラムを作っていきましょう。

変数

```
2
3 ;←この記号から右側は全てコメントとして扱われます
4 ;  コメントはコンピュータからは一切見られません
5
6 a = 0    ;変数を宣言する
7
8 b = 5
9
10 c = a + b    ;変数の足し算
11
12
```

変数名 = 値

値に名前を付け、自由に値を変更することができます。
変数は普通の数字のように四則演算を行うことができます。
変数には、文字を「"」で囲むことで文字も変数として扱えます。

意味	意味
A = B	A に B を**代入する**
A = A + B または **A += B**	A に B を加える（**足し算**）
A = A − B または **A −= B**	A から B を引く（**引き算**）
A = A * B または **A *= B**	A に B をかける（**掛け算**）
A = A / B または **A /= B**	A を B で割る（**割り算**）
A ++（**A --**）	A に**1を加える**（A から**1を引く**）

コメントアウト

; コメント文

「;」から右側の文章はコンピュータがプログラムとして認識しません。
後から確認しやすいようにメモを残しておく場合に使用します。

また、プログラムはコメントアウト外でも、**半角スペース**を全て無視して処理します。全角スペースは無視しないので注意してください。

インデント

tab キーを押すとプログラムに段落を付けることができます。

```
2 a = 5
3 if (a == 5){
4 mes "5です"
5 a += 2
6 };インデントを使わない場合
7
8 a = 5
9 if (a == 5){
10     mes"5です"
11     a += 2
12 }                ;インデントを使用した場合
13
```

配列

```
15 dim value, 9
16
17 value@.0 = 0
18 value@.1 = 5
19 value@.2 = 20
20
21 dim multi, 2, 3
22
23 multi(0,0) = 0
24 multi(0,1) = 5
```

dim 変数名 , 配列の大きさ

変数のまとまりを作ることができます。1つの名前で大きさ分の変数を持つことができるので、いくつか関係のある変数がほしいとき、
配列を用いると沢山の変数を宣言する必要がありません。

変数名@.要素の番号

と表現することで点線が表示され、普通の変数と同じように扱えます。
但し注意点は、配列の大きさが **n 個**だった場合、呼び出すときに用いる数字は **0 から n-1 番**までという点です。

また、配列の大きさをカンマ(,)で区切って2個書くことで、2次元配列と呼ばれる配列を作ることができます。
左図の配列を例に図に表した場合、下の通りです。

配列 value 大きさ 9

0	1	2	3	4	5	6	7	8

配列 multi 大きさ 2 , 3

(0 , 0)	(0 , 1)	(0 , 2)
(1 , 0)	(1 , 1)	(1 , 2)

乱数

```
rnd(20) ;0～19までのランダムな値を出す

randomize ;ランダムのパターンをリセットする場合使う
          ;但したくさん呼ぶとうまく乱数が出来ません
```

rnd(値)

0 から(値-1)までのランダムな値(乱数)を出すことができます。プログラムを実行してから一定の決まったパターンでランダムな値を出すので、本当にランダムな値を出したい時は、

randomize

と書くことでパターンをランダムにします。但し何回も randomize を記入すると正しく乱数ができなくなるので、多用しないようにすること。

If 文「条件分岐」

```
1 pos 0,0
2
3 a = 5
4
5 if( a == 5 ){
6     mes "5です！"
7 }else : if ( a > 5 ){
8     mes "5じゃないけど5より大きい！"
9 }else{
10    mes "5でもないし5よりも大きくない！"
11 }
12
13
```

if(*条件式*) { 処理 }

・*条件式*が満たされている場合、中括弧内の処理を実行します。

満たされていない場合、中括弧内の処理を無視します。

if (*条件式 1*) { 処理 1 } else : if (*条件式 2*){ 処理 2 }

・*条件式 1* が満たされた場合は処理 1 を実行し、
条件式 1 が満たされなかったが、*条件式 2* が満たされた場合は
処理 2 を実行する。

if(*条件式*){処理 1}else{処理 2}

・*条件式* が満たされなかった場合、処理 2 を実行する。
else : if と組み合わせた場合、すべての *条件式* が満たされなかった場合、else の後の中括弧内の処理を実行する。

書き方	条件式の意味
A == B　または　A = B	A と B が**等しい**
A > B	A が B **より大きい**
A >= B	A が B **以上**
A < B	A が B **より小さい**
A <= B	A が B **以下**
A != B	A が B **以外**
条件式 A　||　条件式 B	条件式 A か条件式 B どれか**1つでも満たしているとき**
条件式 A　&&　条件式 B	条件式 A と 条件式 B を**どちらも満たしているとき**

While 文「繰り返し処理」

```
1 pos 0, 0
2
3 a = 0
4
5 while a < 10
6     mes ""+a
7
8     a++
9 wend
```

while ***条件式***

処理

Wend

・*条件式* が満たされている間、while から wend の間に囲まれた処理を条件式が満たされなくなるまで繰り返して実行する。

※無限ループについて

while は while から wend までの間で条件式が満たされなくなるまで繰り返す処理だが、繰り返してもなおずっと条件が満たされている場合、その処理は無限に繰り返され、強制停止を　行わない限り止めることができなくなってしまう。

この現象を**無限ループ**といい、コンピュータのフリーズを引き起こすためとても危険です。そのため、無限ループを起こさないように気を付けてプログラムを作りましょう。

```
2 a = 0
3
4 while a < 5
5     a --
6 wend
7
```

無限ループの一例

while～wend の中で
a<5 を満たし続けるため、
永遠に繰り返し続けてしまう。

簡単な AI の作り方

対戦する上で一番大事な動きは、

・壁に重ならないように移動する

ことです。壁に重なってしまうとルールで負けになってしまいます。

```
mode = "up"

while (checkgame == 1 )      ;ゲームが続いている間回り続ける
    getReady
    ;----------------
    if (mode == "up" && Value@.2 == 2){
        mode = "left"
    }
    if (mode == "left" && Value@.4 == 2){
        mode = "down"
    }
    if (mode == "down" && Value@.8 == 2){
        mode = "right"
    }
    if (mode == "right" && Value@.6 == 2){
        mode = "up"
    }

    if (mode == "up"){
        walkUp
    }else : if(mode == "left"){
        walkLeft
    }else : if(mode == "down"){
        walkDown
    }else : if(mode == "right"){
        walkRight
    }
```

左図のプログラムは壁に重ならない動きをする一例です。
動いている向きを変数でおいて、動いている方向に壁があったら、
変数の値を変えて向きを変えます。ここで気を付けておきたいことは、

・getReady を連続で 2 回行うと負けになる
・各種行動を getReady を行う前に行うと負けになる

この2つです。長いプログラムを作ると起きやすい現象なので、
長いプログラムになる前に対策する必要があります。そのため、
以下の手順を守って作っていくとエラーを抑えることができ、
エラーが発生してもどこで起きたか分かりやすいです。

①getReady はループ開始に 1 個だけ置く
②進む、アイテムを置く、周囲を調べるかどうかの判断をする
③判断を行ったら、どの動作をするか、変数に置き換える
④最後に変数をもとに行動する

これで簡単な AI を持ったプログラムは完成です。
ただし、これだけでは壁にぶつかるごとに反時計回りに回るだけで、
アイテムや敵を見つけても素通りします。

```
if (Value@.2 == 1){
    putmode = "up"
}else : if(Value@.4 == 1){
    putmode = "left"
}else : if(Value@.6 == 1){
    putmode = "down"
}else : if(Value@.8 == 1){
    putmode = "right"
}else{
    putmode = "off"
}

if(putmode == "off"){
    if (movemode == "up"){
        walkUp
    }else : if(movemode == "left"){
        walkLeft
    }else : if(movemode == "down"){
        walkDown
    }else : if(movemode == "right"){
        walkRight
    }
}else{
    if (putmode == "up"){
        putUp
    }else : if(putmode == "left"){
        putLeft
    }else : if(putmode == "down"){
        putDown
    }else : if(putmode == "right"){
        putRight
    }
}
```

勝つために大事な動き

相手と戦って勝つためにはアイテムを多くとるか、敵の上に壁を置く、
敵の四隅を壁で囲むといった行動を取る必要があります。

・アイテムを見つけたら取る
・敵を見つけたら敵めがけてブロックを置く

左のプログラムでは、6 ページで紹介したプログラムの一例に書き加え、
敵めがけてブロックを置く処理を書いています。

基本的な動作を追加しても、対策を取らない限り、ある場所で同じ動き
を延々と繰り返して抜け出せない状態（**ループ**）が発生してしまいます。
多くのアイテムを取るためにループは抜け出さなくてはならないので、

・何度か同じ動きをしたら違う方向に動く
・向きを変更する時にランダム要素を持たせる
・ループしている道のりに壁を設置してループを崩す
・Look や Search を用いて遠くを調べて向きを変える

など、さまざまな対策が考えられます。
プログラムによって様々な対策が考えられ、1つの最高な対策といった
ものはありません。よって、サンプルプログラムはあえて記述しません。