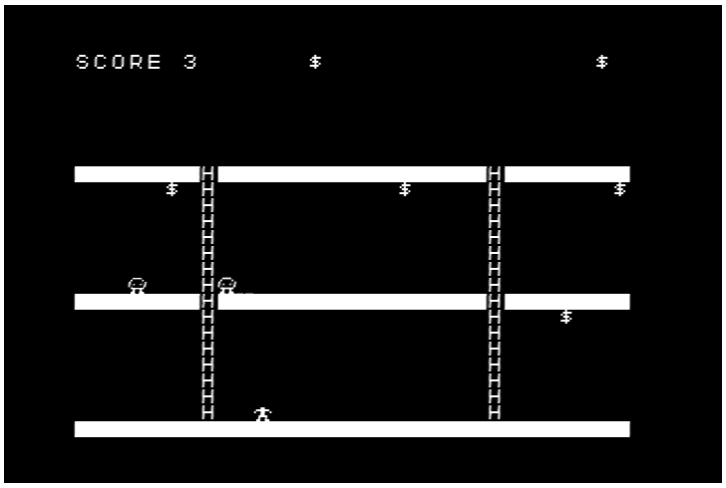
【A-3】IchigoJam でジャンプアクションゲームを作る

●今回の目標

IchigoJam で動くジャンプアクションゲームを作ります。
自分のキャラクターがステージの中を移動したりジャンプしたりするゲームです。



●スコアを表示する

まず、プログラムの初期設定をした後、画面にスコア（点数）を表示します。

```
10 '*Jump Act
20 CLV:CLS:?"SCORE 0"
```

コメントで、プログラムのタイトルを入れる

変数をクリアする:画面をクリアする:スコアを画面左上に表示

入力できたら、「RUN」で実行してみましょう。
画面がクリアされて、左上にスコア「0」が表示されます。

プログラムの内容を説明します。

10 行:コメントで、プログラムのタイトルを入れています。

先頭に「' 」(アポストロフィ、キーボードでは Shift キーを押しながら「7」を押す)を付けると、その行はコメントとなり、何も実行されません。ですから、「' 」のあとは好きな文字を書くことができます。

プログラムを後で見た時にわかりやすくするために、プログラムにいろいろコメントを入れるといいでしょう。

20 行:CLV(シーエルブイ)命令は、**変数**(へんすう)をクリアする命令です。

変数とは、数字を入れる入れ物(箱)と思ってください。

小学校の算数でやる「□」(四角)、中学校の数学でやる「x」と考えればいいです。

今回は、ゲームの最初に、全ての変数をクリアしています。

続く CLS(シーエルエス)命令は、画面をクリアする命令です。

続いて「?"ＳＣＯＲＥ　０"」で、画面左上に「SCORE 0」の文字を表示しています。「?」は「ＰＲＩＮＴ」の略です。

今回はプログラムが長くなるので、こうした省略形を使ったり、「:」(コロン)で複数の命令を続けたりして、プログラムの容量を節約しています。打つ文字数も少なくてすみます。

●床を表示する

次に、ステージの 3 段の床(ゆか)を表示します。
前のプログラムに追加して入力します。

```
10 '*Jump Act
20 CLV:CLS:?"SCORE 0"
30 FOR C=7 TO 23 STEP 8:LC 0,C
40 FOR B=0 TO 30:?CHR$(1);:NEXT
50 NEXT
```

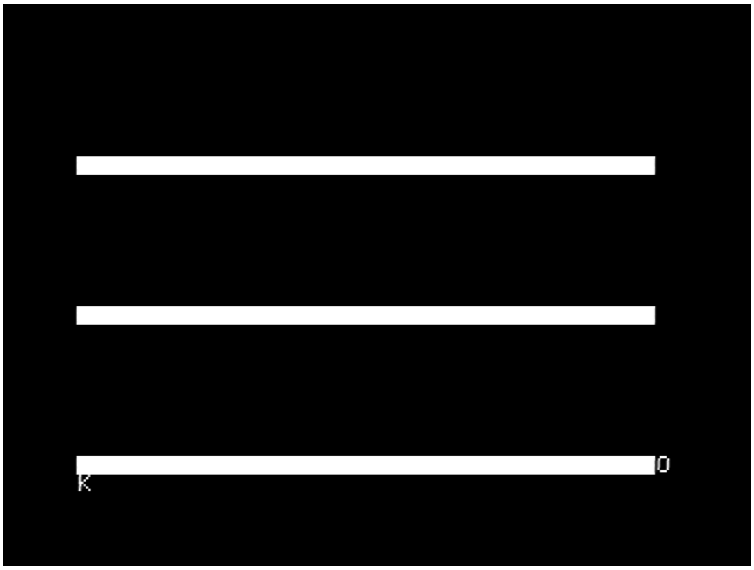
変数 C を7から 23 まで、8 ずつ増やす
:座標(0,C)にカーソルを移動

横に 31 文字の「■」を表示

くりかえし(50 行へもどる)

プログラムを実行してみましょう。
3 段の床が表示されます。

※一番下の床を表示した後に「OK」が表示されて、画面全体が上へスクロールしてしまうのですが、今回は気にしないことにします。



30 行:3 段の床を表示するために、FOR(フォー)命令を使って、カーソルの y 座標(縦座標)を、7,15,23 の 3 種類に変化させてくり返します。

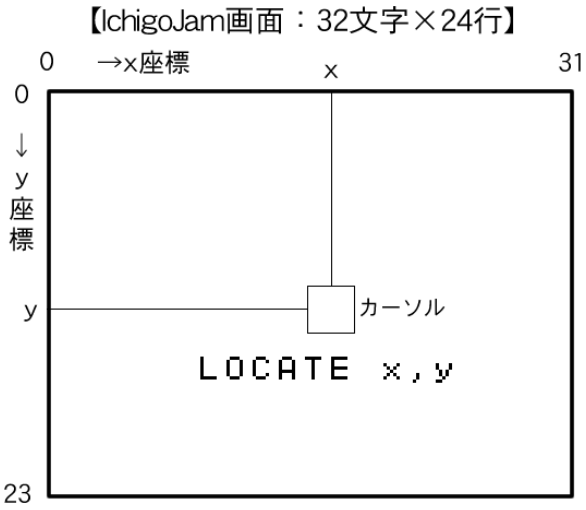
```
FOR  C=7  TO  23  STEP  8
     変数の   変数の      変数の
     初期値   終値        増分
```

変数の初期値	ループ変数の最初の値
変数の終値	ループ変数の最後の値
変数の増分	ループ変数をどのくらい変化させるかの値 STEP 以下を省略すると1ずつ増やす

次の「ＬＣ」は「ＬＯＣＡＴＥ」の略です。
LOCATE（ローケート）命令は、画面に文字を表示する位置（カーソル位置）を設定する命令です。

```
LOCATE  0  ,C
        x 座標   y 座標
```

x 座標	画面の x 座標（0～31）。
y 座標	画面の y 座標（0～23）。



IchigoJam の画面サイズは、横 32 文字×縦 24 行になっています。
横（x 方向）の座標は 0～31、縦（y 方向）の座標は 0～23 になっています。
今回は「ＬＣ　０，Ｃ」として、画面左側の、床を表示しはじめる座標を指定しています。
FOR 命令で、C の値が 7,15,23 と変化するので、カーソルの座標は、1 回目は(0,7)、2 回目は(0,15)、3 回目は(0,23)になります。

40 行：2 個めの FOR～NEXT ループです。床を画面左はじから右はじまで表示するため、変数 B を 0～30 まで変化させます。

「？ＣＨＲ＄（１）；」で、画面に床の 1 文字「■」を表示しています。
後ろに「；」（セミコロン）が付いているので、「■」を表示した後に改行されず、カーソルがその右側にとどまります。それを FOR～NEXT でくり返すので、床が横に連続して表示されます。

50 行：30 行の FOR 命令にもどる NEXT 命令です。
全体として、2 重の FOR～NEXT ループになっています。

ループ②
ループ①

```
10 '*Jump Act
20 CLV:CLS:?"SCORE 0"
30 FOR C=7 TO 23 STEP 8:LC 0,C
40 FOR B=0 TO 30:?CHR$(1);:NEXT
50 NEXT
```

●はしごを表示する

次に、3 段の床を上り下りするはしごを表示します。
前のプログラムに追加して入力します。

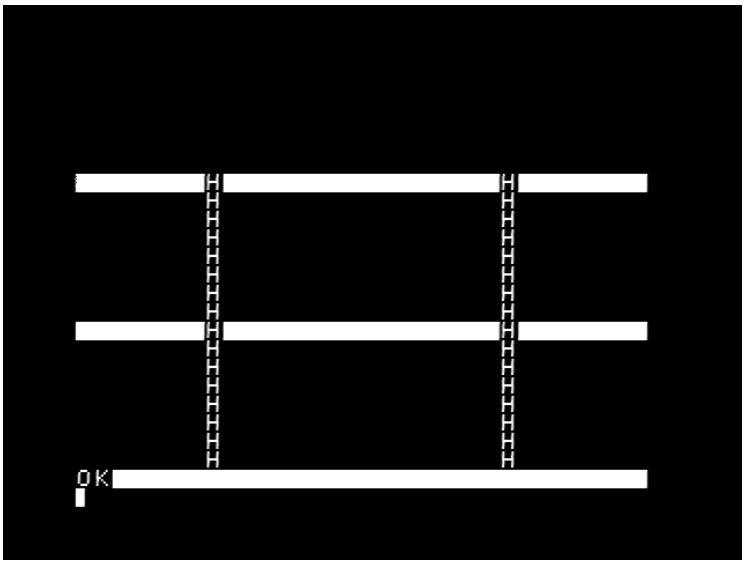
```
10 '*Jump Act
20 CLV:CLS:?"SCORE 0"
30 FOR C=7 TO 23 STEP 8:LC 0,C
40 FOR B=0 TO 30:?CHR$(1);:NEXT
50 NEXT
60 FOR C=7 TO 22
70 LC 7,C:?"H":LC 23,C:?"H"
80 NEXT
```

変数 J を7から 22 まで変化させる

くりかえし

左側と右側のはしごを表示

プログラムを実行してみましょう。
左右 2 本のはしごが表示されます。



60 行:左右のはしごを表示するために、FOR 命令を使って、カーソルの y 座標(縦座標)を、7～22 まで変化させてくり返します。

70 行:「LC」(LOCATE)と「?」(PRINT)を使って、左のはしごと右のはしごの 1 文字を表示します。今回はアルファベットの「H」を表示して、はしごに見せています。

80 行:60 行の FOR 命令にもどってくり返します。これで上から下まではしごがつながります。

70 行の「LC 7,C」「LC 23,C」の数値を変えると、左右のはしごの位置が変わります。いろいろ変えて試してみましょう。

「SAVE 0」でプログラムを保存しておきましょう。

●人間を表示する

ステージはできたので、ここに自分のキャラ(人間)を表示してみましょう。
以下のプログラムを追加します。

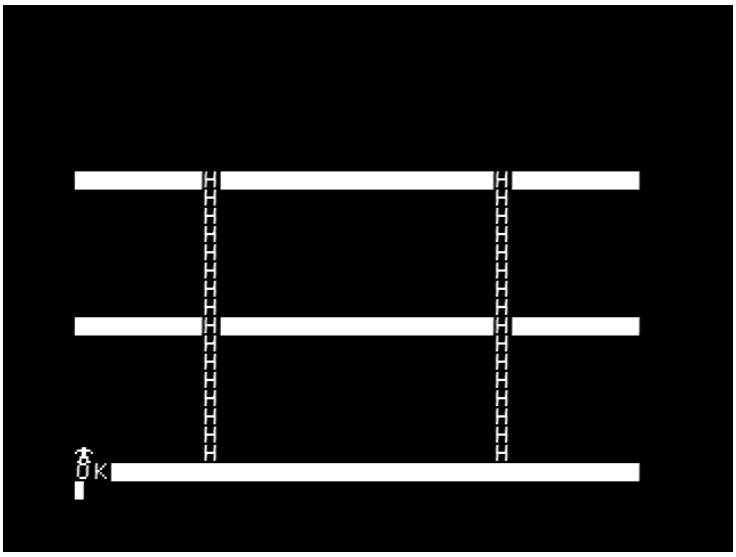
```
10 '*Jump Act
20 CLV:CLS:?"SCORE 0":X=0:Y=22
…(中略)…
60 FOR C=7 TO 22
70 LC 7,C:?"H":LC 23,C:?"H"
80 NEXT
100 '*LOOP
200 LC X,Y:?CHR$(250)
```

人間の x,y 座標を設定

ゲームループの先頭コメント

カーソルを移動:人間のキャラを表示

プログラムを実行してみましょう。
ステージ左下に人間が表示されます。



※あとでいろいろプログラムを追加するので、行番号が「100」の次に「200」へ飛んでいます。

今回は文字コード 250 番を指定して、人間のキャラクターを画面に表示しています。
IchigoJam の文字コードは、右のようになっています。

人間(250 番)

CHR$関数の文字コードを変えると、表示されるキャラクターが変わります。試してみましょう。

●人間を左右に動かす

表示した人間を、キー入力で左右に動かしてみましょう。
以下のプログラムを追加します。

```
…(前略)…
100 '*LOOP
150 U=X-BTN(LEFT)+BTN(RIGHT)
190 LC X,Y:?CHR$(0)
200 LC U,Y:?CHR$(250):X=U
290 WAIT 3:GOTO 100
```

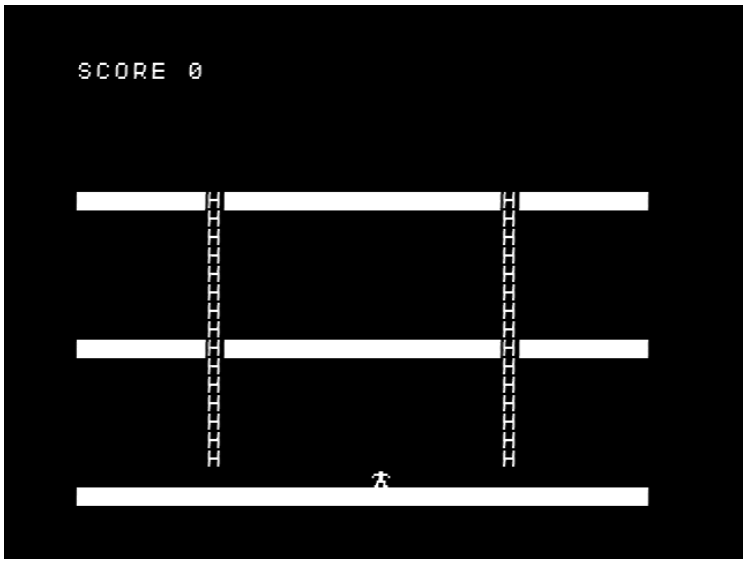
人間の次の x 座標 U をキー入力から計算

前の位置の人間を消去

次の位置に人間を表示
座標を次の位置に更新

時間待ちして、ループ先頭にもどる

プログラムを実行してみましょう。
カーソルキー(矢印キー)の左と右を押すと、人間が左右に動きます。



150 行:BTN(ボタン)関数を使い、左右のカーソルキーの入力を読み取って、人間が移動する先の横座標 U を計算しています。

```
BTN( LEFT)
     キー指定
```

キー指定	LEFT…左矢印キー(←) RIGHT…右矢印キー(→) UP…上矢印キー(↑) DOWN…下矢印キー(↓) SPACE…スペースキー
返り値	キーが押されている＝1 キーが押されていない＝0

指定したキーが押されていると、BTN 関数の値が 1、押されていないと 0 になります。

この BTN 関数を使って、人間が移動する先の横座標 U を計算します。

```
U=X-BTN(LEFT)+BTN(RIGHT)
```

キーが押されていない／押されている場合に分けて表にして、U がどんな値になるか考えてみます。

キー入力	U=X-	BTN(LEFT)	+	BTN(RIGHT)	結果
なし		0		0	U=X-0+0=X
←		1		0	U=X-1+0=X-1
→		0		1	U=X-0+1=X+1

キーが押されていなければ、U=X となり、横座標は X のまま変わらず、
左矢印キー「←」が押されていれば、U=X-1 となり、左どなりの座標、
右矢印キー「→」が押されていれば、U=X+1 となり、右どなりの座標になります。
このように計算式を組み合わせることで、短いプログラムで人間を移動できます。

190 行：元の位置(X,Y)にいる人間を、「CHR$(0)」(文字コード 0)の文字で消しています。

200 行：次の位置(U,Y)に人間を表示した後、「X=U」として、横座標 X を次の位置 U に更新しています。

290 行：WAIT（ウェイト）命令で時間待ちをします。（これを入れないと動きが速すぎるため）

```
WAIT    3
        待ち時間
```

待ち時間	60 分の 1 秒単位で指定する。「60」で 1 秒。

その後、GOTO（ゴートゥー）命令で、ループの先頭（100 行）へもどります。

```
GOTO 100
     行番号
```

指定した行番号へ実行を移します。

このプログラムのままだと、いろいろ問題があります。

- (1)人間が横に移動すると、はしごが消えてしまう。
  移動する人間を、無条件に CHR$(0)(文字コード 0)で消しているからです。

- (2)人間が左はじ・右はじに行くと、おかしな動きをする。
  人間が左はじにいる時に「←」キーを押し続けると、人間は左はじにいて動きませんが、次に「→」キーを押すと、しばらく押さないと右へ動きません。
  逆に人間が右はじへ移動すると、画面がスクロールしておかしな表示になってしまいます。
  これは、人間が左はじ・右はじに行った時に、それ以上は移動しないようにする、いわゆる「はみ出し禁止処理」をしていないからです。

まず(2)のはみ出し禁止処理を追加します。

```
…(前略)…
100 ' *LOOP
150 U=X-BTN(LEFT)*(X>0)+BTN(RIGHT)
*(X<30)
190 LC X,Y:?CHR$(0)
200 LC U,Y:?CHR$(250):X=U
290 WAIT 3:GOTO 100
```

左はじのはみ出し禁止処理

右はじのはみ出し禁止処理

プログラムを実行してみましょう。左はじ・右はじで人間がはみ出さずに収まるようになります。

はみ出し禁止処理のプログラムで、まず左はじの部分を見てみましょう。

キー入力	人間位置	-BTN(LEFT)	*(X>0)	結果
なし	通常	0	1	U=X-0×1=X
←	通常	1	1	U=X-1×1=X-1
なし	左はじ	0	0	U=X-0×0=X
←	左はじ	1	0	U=X-1×0=X

IchigoJam BASIC のプログラムでは、「X>0」などの**数式自体も値を持ちます。**
式が成り立てば「1」、成り立たなければ「0」になります。
人間の位置が通常(左はじではない)の時、キーが押されていない時は次の座標は U=X となって変わりません。「←」キーが押されると U=X-1 となり、左へ 1 個移動します。
しかし人間が左はじにいる時は X=0 となるので、数式「X>0」の値は「0」となり、キーが押されていてもいなくても U=X となって移動しません。

右はじも同様で、人間が右はじにいる時は X=30 となるので、数式「X<30」の値が「0」となり、キーが押されていてもいなくても U=X となって移動しません。

キー入力	人間位置	+BTN(RIGHT)	*(X<30)	結果
なし	通常	0	1	U=X+0×1=X
→	通常	1	1	U=X+1×1=X+1
なし	右はじ	0	0	U=X+0×0=X
→	右はじ	1	0	U=X+1×0=X

「IF X<30 THEN ～」と IF 命令で場合分けすることもできるのですが、このように数式にすることでプログラムを短くできます。

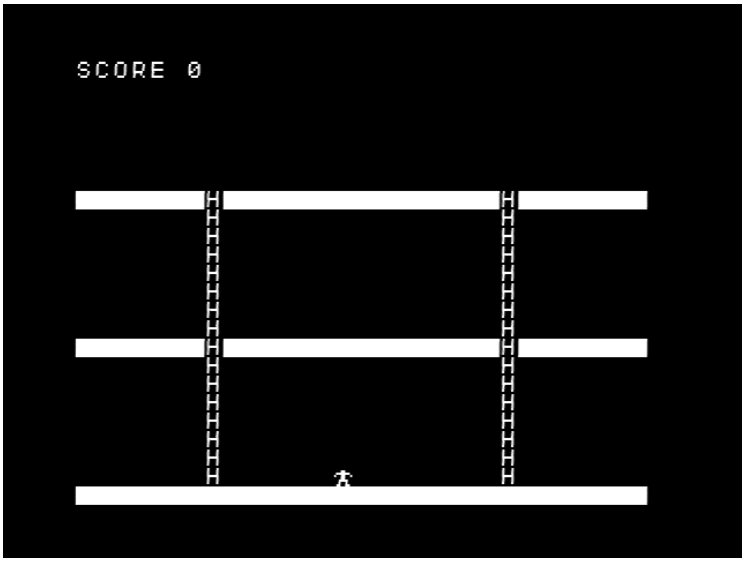
次に、はしごを消さない方法です。
普通に人間を消去してしまうとはしごが消えてしまうので、「移動前にどんな背景があったか」を変数に記憶しておいて、その背景で人間を消すようにします。

```
…(前略)…
100 '*LOOP
150 U=X-BTN(LEFT)*(X>0)+BTN(RIGHT)
*(X<30)
190 LC X,Y:?CHR$(H):H=SCR(U,Y)
200 LC U,Y:?CHR$(250):X=U
290 WAIT 3:GOTO 100
```

記憶していた背景で消す

行き先にある背景を読み取って H に入れる

プログラムを実行してみましょう。
今度は人間を左右に動かしても、はしごが消えません。



画面に表示されている文字を読み取るには、SCR（スクリーン）関数を使います。

```
SCR(  X  ,Y)
```
x 座標　y 座標

x 座標	読み取りたい場所の x 座標。
y 座標	読み取りたい場所の y 座標。
返り値	その場所に表示されている文字の文字コード。 その場所に何も表示されていない時や、座標が画面をはみ出している時は「0」が返る。

今回は、前回記憶していた背景（文字コード H）で人間を消して、行き先にある背景を新たに H に入れています。

記憶していた背景で消す　　行き先にある背景を読み取って H に入れる

```
?CHR$(H):H=SCR(U,Y)
```

プログラム実行直後は、CLV 命令で変数をクリアしているので、H=0 になっています。
そのため最初は文字コード 0（空白）で消し、その後は SCR 関数で読み取った背景で消していきます。

これで人間を左右に動かすプログラムができました。

「SAVE 0」でプログラムを保存しておきましょう。

●人間がはしごを上る･下る

人間を左右に動かすことができたので、今後ははしごを上り下りさせてみましょう。
以下のプログラムを追加します。

```
…(前略)…
100 '*LOOP
110 V=Y-BTN(UP)*(H=72)+BTN(DOWN)*(SCR(X,Y+1)=72)
150 U=X-BTN(LEFT)*(X>0)+BTN(RIGHT)*(X<30)
190 LC X,Y:?CHR$(H):H=SCR(U,V)
200 LC U,V:?CHR$(250):X=U:Y=V
290 WAIT 3:GOTO 110
```

背景がはしごだったら上る。
1 個下の背景がはしごだったら下る。

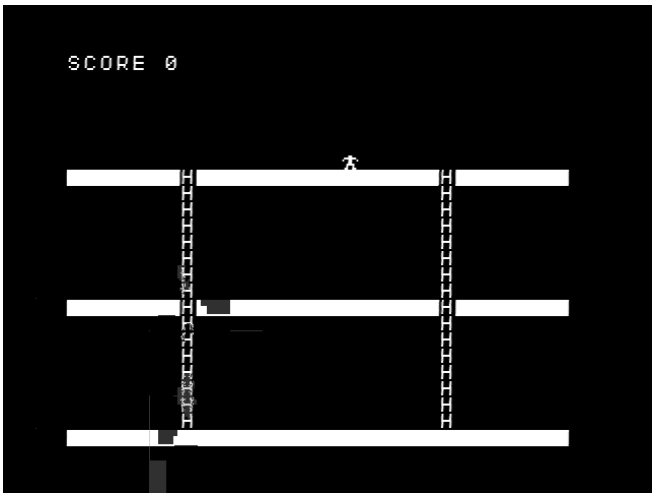
変数を V にする

y 座標を V に更新

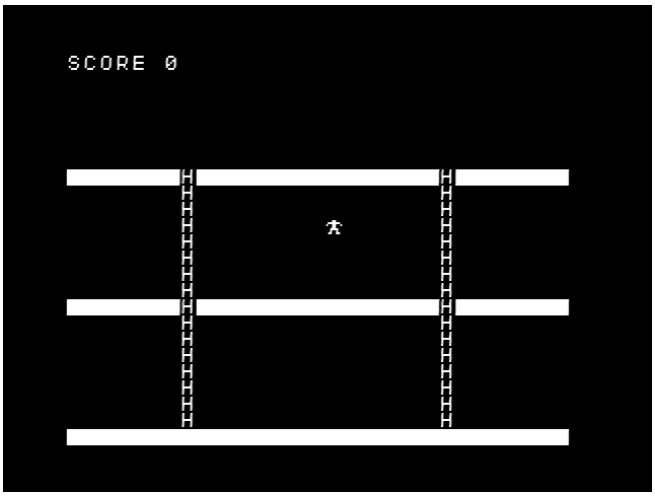
変数を V にする

変数を V にする

プログラムを実行してみましょう。
カーソルキーの上「↑」・下「↓」を押すと、はしごの所で人間が上り下りできます。2 階や 3 階に上がれます。



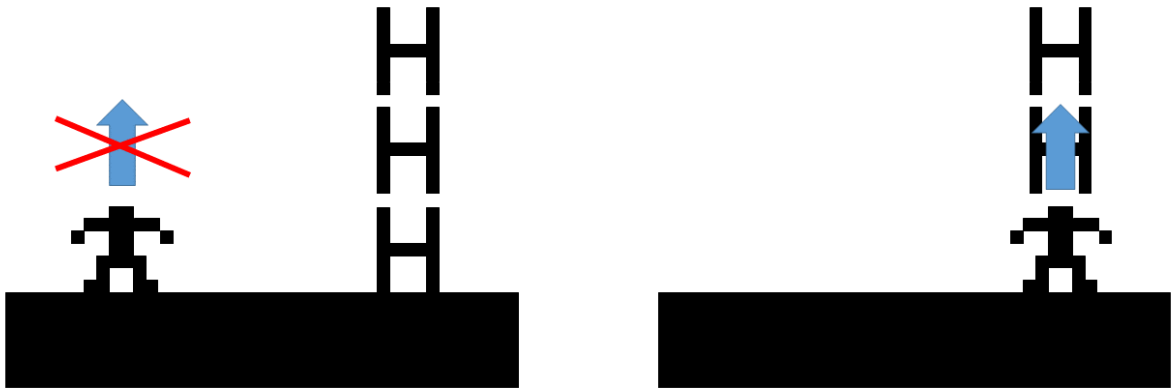
ただし、床が無い所でも左右に動けるので、はしごの途中から空中を歩くことができてしまいます。
(※これは後で直します)



今回追加したプログラムを説明します。

```
110 V=Y-BTN(UP)*(H=72)+BTN(DOWN)*(SCR(X,Y+1)=72)
```

110 行で、はしごを上る・下りる処理をしています。
まず「上矢印キー(↑)を押した時にはしごを上る」処理ですが、「どんな条件の時に上れるか」を考えてみましょう。



はしごが無い場所では上れない　　はしごがある場所へ来れば上れる

上の図にあるように、人間がはしごが無い場所にいる時は上れません。
はしごがある場所へ来れば上れます。
つまり、「人間の背景にはしごがあるか？」を判断すればいいことになります。

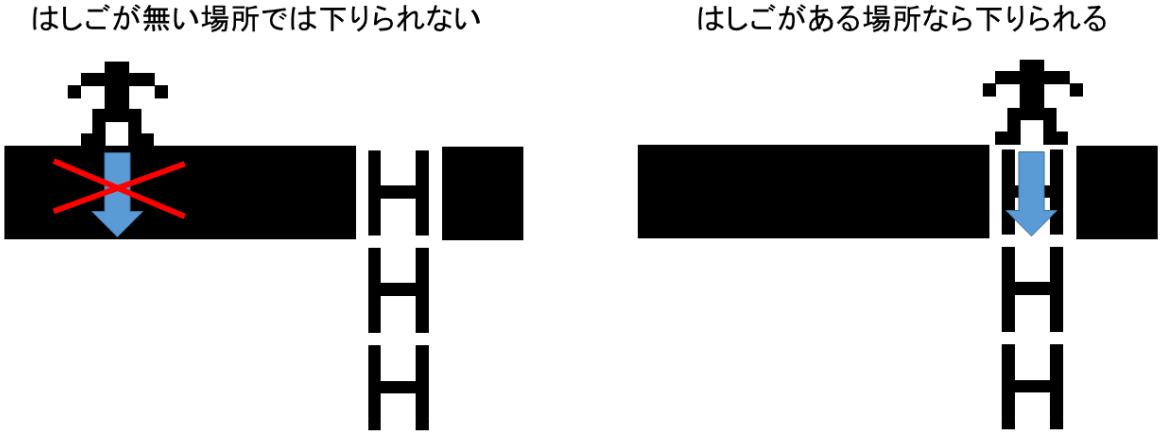
先ほどの「背景を消さずに人間を移動する」プログラムで、背景にあった文字コードを変数 H に記憶しています。それがはしご「H」(文字コード 72)だったら、人間がはしごの場所にいますので、上へ上れます。

人間を左右に動かす時と同様に、キー入力を読み取る BTN 関数と、数式の値を使って、人間の行き先の y 座標(変数 V)の値を決めます。

キー入力	人間位置	-BTN(UP)	*(H=72)	結果
なし	通常	0	0	V=Y-0×0=Y
↑	通常	1	0	V=Y-1×0=Y
なし	はしご	0	1	V=Y-0×1=Y
↑	はしご	1	1	V=Y-1×1=Y-1

「BTN(UP)」は、上矢印キーが押されたかどうかを読み取ります。押されていたら値が「1」、押されていなければ「0」になります。
また、人間がいる場所の背景がはしごだったら、「H=72」が成り立つので、その式の値が「1」になります。はしごでなければ「0」になります。
これで、「上矢印キーが押された」・「人間がはしごの位置にいる」の両方が成り立った時だけ、人間が上へ移動します。

今度は、はしごを下へ下りる時を考えます。どんな条件だったら下りられるでしょうか。



人間の足元が床だったら、下りられません。足元がはしごだったら、下りられます。
人間の1個下に表示されている文字を読み取って、はしご「H」(文字コード 72)だったら下りられる、と判断します。

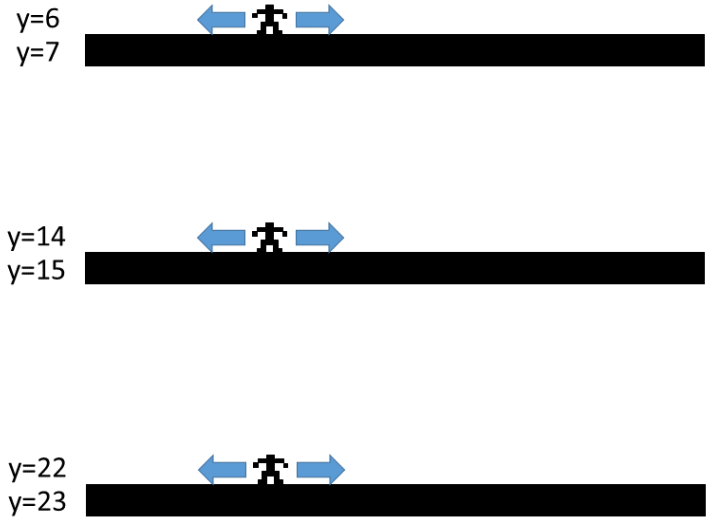
110 行の後半では、キー入力を読み取る BTN 関数と、表示文字を読み取る SCR 関数の値を使って、人間が下りる(＝y 座標を増やす)処理をします。

キー入力	人間位置	+BTN(DOWN)	*(SCR(X,Y+1)=72)	結果
なし	通常	0	0	V=Y+0×0=Y
↓	通常	1	0	V=Y+1×0=Y
なし	はしご	0	1	V=Y+0×1=Y
↓	はしご	1	1	V=Y+1×1=Y+1

「BTN(DOWN)」は、下矢印キーが押されたかどうかを読み取ります。押されていたら値が「1」、押されていなければ「0」になります。
また、「SCR(X,Y+1)」は、人間がいる場所の1個下の文字(足元の背景)を読み取ります。
背景がはしごだったら文字コードが 72 になるので、「SCR(X,Y+1)=72」の式の値が「1」になります。はしごでなければ「0」になります。
これで、「下矢印キーが押された」・「人間の足元がはしご」の両方が成り立った時だけ、人間が下へ移動します。

次に、人間が空中を歩いてしまう問題を解決します。
要は、「人間が床の上にいる時だけ、左右に歩ける」ようにすればいいです。

いろいろな解決方法が考えられますが、今回は人間のｙ座標を使って判断します。
右の図のように、3段の床の ｙ 座標は、7,15,23 になっています。
人間が床の上にいる時は、ｙ座標が 1 つ上の 6,14,22 のどれかなので、その時だけ左右に動けるようにします。



プログラムを以下のように修正します。

```
…(前略)…
100 '*LOOP
110 V=Y-BTN(UP)*(H=72)+BTN(DOWN)*(SCR
(X,Y+1)=72)
150 IF Y%8=6 U=X-BTN(LEFT)*(X>0)+BTN(R
IGHT)*(X<30)
…(後略)…
```

もし人間のｙ座標を 8 で割った余りが 6 だったら(＝6,14,22 だったら)
人間を左右に動かす処理をする

プログラムを実行してみましょう。人間が床の上しか左右移動しなくなります。

新しく追加した IF(イフ)命令の文法は、以下のとおりです。

```
IF Y%8=6 THEN ～ ELSE ～
```

条件式 / 条件が成り立つ時に実行 / 条件が成り立たない時に実行

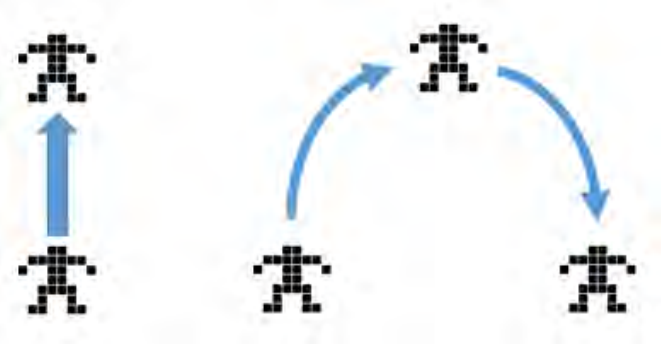
条件式	条件を判断する式。
THEN(ゼン)の後	条件が成り立つ時に実行するプログラム。
ELSE(エルス)の後	条件が成り立たない時に実行するプログラム。ELSE 以下は省略可能。

今回は「IF Y%8=6 U= ～」と、THEN を省略しています。このように省略することもできます。

「SAVE 0」でプログラムを保存しておきましょう。

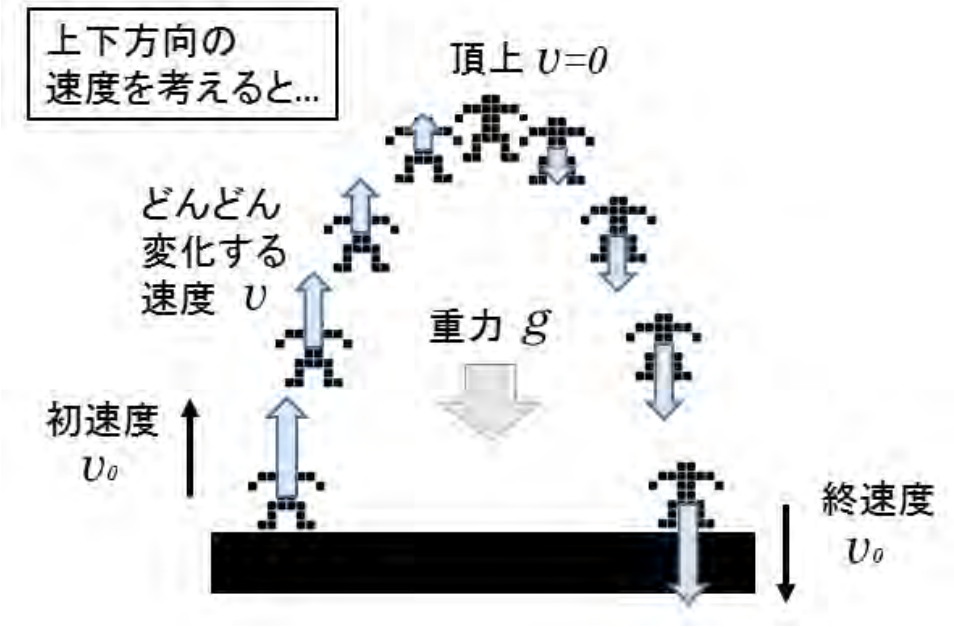
●人間をジャンプさせる

ここまでで、背景にそって人間を動かすことができました。
今度は、人間をジャンプさせる動きを作ってみましょう。
人間をジャンプさせると、アクションの動きが広がっておもしろくなります。



★そもそもジャンプの動きって？

プログラムからちょっとはなれて、上へジャンプする動きがどんなものか考えてみましょう。



ここでは上下方向の速度だけ考えます。
最初に地面をけってジャンプすると、ある速度で上へ飛び出します。
その後は、下向きに重力がずっとかかるので、だんだん速度が遅くなり、頂上では速度が 0 になります。
今度は下向きに速度が増えていって、地面についた時には最初の速度と同じ（ただし下向き）になります。

> 高校の物理で習いますが、このように一定方向に力（重力）がかかる時の動きは「等加速度運動（とうかそくどうんどう）」と言います。
> 初速度を v_0、重力加速度を g とすると、t 秒後の速度 v は、
>
> $$v = v_0 + gt$$

★プログラムではどうする？

この上下方向のキャラクターの動きを、どうやってプログラムにするか考えてみましょう。
「速度」と考えると難しいので、「人間を一度にどのくらい動かすか」と考えます。

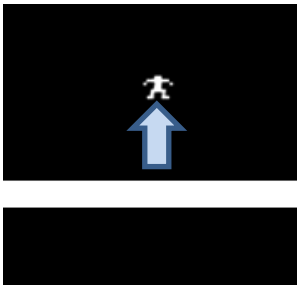
```
…(前略)…
100 '*LOOP
110  V=Y-BTN(UP)*(H=72)+BTN(DOWN)*(SCR
(X,Y+1)=72)
120 IF Y<>V GOTO 190
130 IF BTN(SPACE) AND Y%8=6 J=1:G=-3
140 IF J V=Y+G:G=G+1
150 IF Y%8=6 U=X-BTN(LEFT)*(X>0)+BTN(RIG
HT)*(X<30)
190 LC X,Y:?CHR$(H):H=SCR(U,V)
200 LC U,V:?CHR$(250):X=U:Y=V
290 WAIT 3:GOTO 110
```

もし、上下方向へ移動していたら
ジャンプと横移動の処理を飛ばす

もし、スペースキーが押されている
かつ、床に立っていたら、ジャンプ開始、
重力による変化量を-3 にする

もし、ジャンプ中だったら
人間の y 座標を G だけ変化させ、G を 1 加算

プログラムを実行してみましょう。
スペースキーを押すと、人間がジャンプします。
ただし、床についても着地できなくて、下へ落ちていってしまいます。
(※後で直します)



まず 130 行が、スペースキーが押された時に、ジャンプを開始する処理です。

```
130 IF BTN(SPACE) AND Y%8=6 J=1:G=-3
```

- BTN 関数「BTN(SPACE)」の値は、スペースキーが押されていたら「1」になります。
  条件式の値が「1」なら、IF 命令は「条件が成り立っている」と判断します。
- 次の「AND Y%8=6」は、2 つ目の条件式です。人間が空中を歩かないようにする対策でも使いましたが、「Y%8=6」が成り立つ＝「人間が床の上(の座標)にいる」ということです。
  IF 命令全体では、「スペースキーが押された、かつ、人間が床の上にいるなら、ジャンプの処理(**J=1：G=-3**)をする」という意味になります。
  人間がはしごを上っている途中でスペースキーが押された場合、そこでジャンプしてしまうのはおかしいので、このようにしています。
- ジャンプの処理では、まず変数 J を 1 にします。J は「今ジャンプしているかどうか」を設定する変数で、「**ジャンプ中=1**」「**ジャンプしていない=0**」という値にします。
  そして y 座標の変化量 G(一度に動かす量)を-3 にします。上へのジャンプは y 座標が減る方向なので、マイナスの値にします。

次の 140 行が、ジャンプの動きの処理です。

```
140 IF J V=Y+G:G=G+1
```

- IF 命令の条件式が「J」だけになっていますが、これは「J=1」(J が1だったら→ジャンプ中だったら)と同じです。130 行と同様に、条件式の値が「1」だったら、「条件が成り立った」と判断します。
- 「V=Y+G」以下は、ジャンプ中の処理です(この前の「THEN」が省略されています)。人間の次の y 座標 V を「V=Y+G」として、今の位置から G だけ変化させます。最初は G=-3 なので、V=Y-3 となり、3 個上の座標になります。
- そして「G=G+1」として、G の値を 1 だけ増やします。
  (「G=G+1」は、「G の値に1加えて、それを G に入れる」という意味です。「G と G+1 が等しい」という意味ではないので、注意してください)
  これによって G の値が 1 増えていくので、だんだん上向きの速度が減り、やがて下向きの動きに変わります。

では、ジャンプした後、床でちゃんと着地するようにしてみましょう。

```
…(前略)…
100 '*LOOP
110  V=Y-BTN(UP)*(H=72)+BTN(DOWN)*(SCR(
X,Y+1)=72)
120 IF Y<>V GOTO 190
130 IF BTN(SPACE) AND Y%8=6 THEN J=1:G=-3
140 IF J V=Y+G:G=G+1
150 IF Y%8=6 U=X-BTN(LEFT)*(X>0)+BTN(RI
GHT)*(X<30)
190 LC X,Y:?CHR$(H):H=SCR(U,V)
200 LC U,V:?CHR$(250):X=U:Y=V
210 IF Y%8=6 J=0
290 WAIT 3:GOTO 110
```

もし、人間の y 座標が床の上だったら
ジャンプをやめて着地する

プログラムを実行してみましょう。
床で着地して、ジャンプが止まるようになります。

210 行では、ジャンプ開始の時と同じように、人間の位置が床の上かどうかを判断しています。
もし床の上だったら、ジャンプ変数 J を 0 にしてジャンプをやめています。

「SAVE 0」でプログラムを保存しておきましょう。

●敵を登場させる

ここまでで、人間のアクションはできました。
今度は、敵のキャラクターを登場させてみましょう。

```
…(前略)…
190 LC X,Y:?CHR$(H):H=SCR(U,V)
200 LC U,V:?CHR$(250):X=U:Y=V
210 IF Y%8=6 J=0
250 IF [0]=0 GSB 400
290 WAIT 3:GOTO 110
400 '*ENEMY
410 IF RND(50)>0 RTN
420 [0]=1:[1]=30:[2]=RND(3)*8+6
450 LC [1],[2]:?CHR$(237):RTN
```

もし敵がいなかったら
敵発生サブルーチンを呼ぶ

もし 50 分の 1 の確率に入っていなかったら
敵を発生させずにもどる

敵の初期設定

敵を表示してもどる

プログラムを実行してみましょう。
画面右側に敵が表示されます。
実行する度に、1 階・2 階・3 階のどこかの床に出現します。
今の所は、表示されるだけで動きません。

プログラムを解説します。
敵のいろいろな値を管理するのに、**配列変数**(はいれつへんすう)を使います。

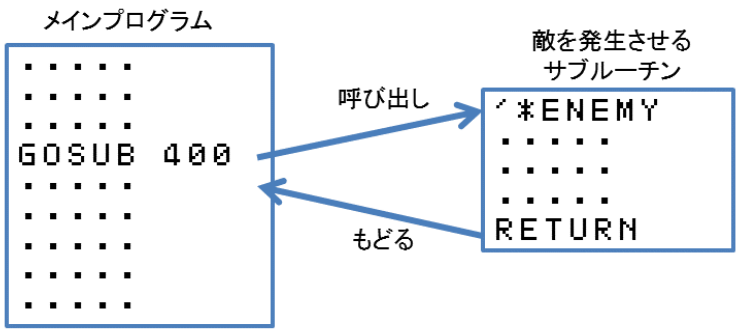
```
250 IF [0]=0 GSB 400
```

配列変数は、番号がついている変数です。
[0]、**[1]**、**[2]** …と、中かっこと数字であらわします。
IchigoJam では、[0]～[101]の 102 個の配列変数が使えます。

ここでは、配列変数[0]を「敵がいるかどうか」を表す変数にしています。
値は「**敵が存在している=1**」「**敵が存在していない=0**」にします。

今回は、「もし敵が存在していなかったら([0]=0 だったら)、敵を発生するサブルーチンを呼ぶ」としています。

サブルーチンとは、メインプログラムから呼び出す処理のことです。
メインプログラムからは「GOSUB」(ゴーサブ)命令でサブルーチンへジャンプします。
サブルーチンからは「RETURN」(リターン)命令でもどります。もどった後は、GOSUB 命令の続きへプログラムの処理が移ります。



今回は、敵を発生させる処理をあとでまた使うため、このようにサブルーチンにします。

また、プログラムを短くするために、「GOSUB」→「GSB」、「RETURN」→「RTN」の省略形を使っています。

400 行から、敵を発生するサブルーチンになっています。
410 行では、敵を発生させるかどうかを、乱数を使って判断しています。

```
410 IF RND(50)>0 RTN
```

「RND(50)」で 0～49 の乱数が出ます。それが 0 より大きかったら「RTN」でメインプログラムへもどります。
つまり乱数が 0 の時だけ(＝50 分の 1 の確率)、これ以降の敵発生を行うことになります。

420 行は、敵の初期設定です。

```
420 [0]=1:[1]=30:[2]=RND(3)*8+6
```

敵の存在変数[0]を 1 にして、「敵が発生している」状態にします。
次に敵の x 座標[1]、y 座標[2]を設定します。
y 座標[2]は、3 段ある床の上のどれかに、乱数で決めます。

0	1	2	3
敵:存在	敵:x座標	敵:y座標	敵:背景

450 行では、設定した座標に敵を表示して、メインプログラムへもどります。

```
450 LC [1],[2]:?CHR$(237):RTN
```

●敵を動かす

画面右側に登場した敵を、左の方へ動かしてみましょう。
敵を移動するサブルーチンを追加します。

```
…(前略)…
250 IF [0] GSB 500 ELSE GSB 400
290 WAIT 3:GOTO 110
400 '*ENEMY
410 IF RND(50)>0 RTN
420 [0]=1:[1]=30:[2]=RND(3)*8+6
450 LC [1],[2]:?CHR$(237):RTN
500 D=[1]-1:E=[2]
510 LC [1],[2]:?CHR$([3])
520 IF D<0 [0]=0:RTN
530 [3]=SCR(D,E)
540 LC D,E:?CHR$(237)
590 [1]=D:[2]=E:RTN
```

もし敵が発生していたら
敵移動サブルーチンを呼ぶ

敵の移動先の座標を計算

敵を消去

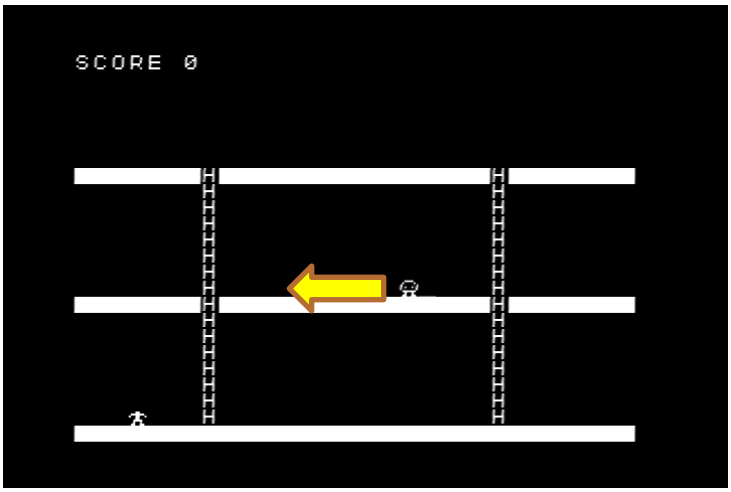
もし移動先の x 座標が 0 より小さいなら、敵消滅

移動先の背景を読み取る

移動先に敵を表示

敵の座標を更新して、もどる

プログラムを実行してみましょう。
しばらく待っていると、画面右側に発生した敵が、左へ移動していきます。
画面の左はじへ行くと消えます。



追加したプログラムを説明します。
250 行では、敵発生変数[0]が 1 だったら(＝敵が画面にいるなら)敵移動サブルーチン(500 行～)を呼ぶ、[0]が 0 だったら敵発生サブルーチン(400 行～)を呼んでいます。

```
250 IF [0] GSB 500 ELSE GSB 400
```

[0]だけの条件式にして、その値が 1 か 0 かで判断しています。

500 行目からは敵移動サブルーチンです。
500 行では、敵の座標[1],[2]から、移動先の座標 D,E を計算しています。

```
500 D=[1]-1:E=[2]
```

左へ 1 個移動するので、x 座標は「D=[1]-1」としています。

510 行では、敵を消去しています。

```
510 LC [1],[2]:?CHR$([3])
```

人間を消す時に、もともと背景にあったもの（変数 H）で消しましたが、同じように配列変数[3]を背景を記憶する変数として使い、[3]の文字コードで敵を消しています。はしごが消えてしまうと困るからです。

520 行では、敵が左はじへ行ったかどうかを判断しています。

```
520 IF D<0 [0]=0:RTN
```

敵が左はじ（のさらに左）へ行った時は、移動先の x 座標 D が 0 より小さくなるので、その時は敵発生変数[0]を 0（発生していない状態）にして、メインプログラムへもどります。
直前の 510 行で敵を消しているので、敵が消えたままでもどることになります。

530 行では、移動先の背景を読み取っています。

```
530 [3]=SCR(D,E)
```

SCR 関数で移動先の背景を読み取って、配列変数[3]に入れています。
510 行で説明したように、移動元ではこの[3]の文字コードで敵を消しているので、はしごを通りすぎる時ははしごが表示されます（＝はしごが消えない）。

540 行では、移動先の座標 D,E に敵を表示しています。

```
540 LC D,E:?CHR$(237)
```

590 行では、敵の座標を新しい位置に更新して、メインプログラムへもどります。

```
590 [1]=D:[2]=E:RTN
```

「SAVE 0」でプログラムを保存しておきましょう。

●人間と敵の当たり判定をする

このままだと、人間と敵が当たっても何も起きません。
敵に当たった時はゲームオーバーになるようにしましょう。
当たるケースとしては、

- 人間が移動して敵に当たる
- 敵が移動して人間に当たる

の 2 つのパターンが考えられます。
まず「人間が移動して敵に当たる」当たり判定をして、もし当たっていたらゲームオーバーの処理をします。

```
…(前略)…
190 LC X,Y:?CHR$(H):H=SCR(U,V)
200 LC U,V:?CHR$(250):X=U:Y=V
210 IF Y%8=6 J=0
220 IF H=237 M=1
250 IF [0] GSB 500 ELSE GSB 400
290 WAIT 3:IF M=0 GOTO 100
300 LC 11,11:?"GAME OVER":END
…(後略)…
```

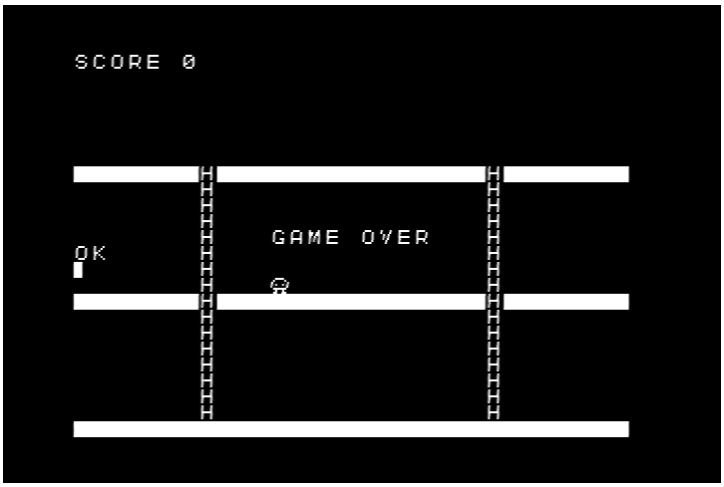
もし背景が敵だったら
ミス変数を 1 にする

もしミス変数が 0 だったら
ループの先頭にもどる

ゲームオーバーを表示して終了

プログラムを実行してみましょう。
敵が出てきたら、人間を動かして敵に当たってみましょう。
ゲームオーバーになります。

※人間から敵へ当たりにいかないとゲームオーバーになりません。
人間が立ち止まって待っていると、敵がすり抜けてしまいます。



プログラムの内容を説明します。
220 行では、背景の文字コードを記憶している変数 H を使って、当たり判定をしています。

```
220 IF H=237 M=1
```

背景の文字コード H が 237（敵）だったら、ミス変数 M を1にします。

290 行で、この M を使って、ゲームオーバーかどうかを判断しています。

```
290 WAIT 3:IF M=0 GOTO 100
```

これまでは「**GOTO 100**」と無条件でループ先頭へもどっていたのですが、IF 命令を使って「もし M=0 だったら（ミスをしていなかったら）、100 行へもどる」と変更します。
M=1 だったらミスをしているので、300 行のゲームオーバーの処理へ移ります。

```
300 LC 11,11:?"GAME OVER":END
```

画面中央に「GAME OVER」の文字を表示して、END 命令でプログラムを終了します。

次に、敵移動サブルーチンを改造して、「敵が移動して人間に当たった場合」の当たり判定をします。

```
…(前略)…
500 D=[1]-1:E=[2]
510 LC [1],[2]:?CHR$([3])
520 IF D<0 [0]=0:RTN
530 [3]=SCR(D,E)
540 LC D,E:?CHR$(237)
550 IF [3]=250 M=1
590 [1]=D:[2]=E:RTN
```

もし移動先の背景が人間なら
ミス変数 M を 1 にする

プログラムを実行してみましょう。
今度は人間が立ち止まっていても、敵が当たるとゲームオーバーになります。

550 行では、敵の移動先にある背景文字コード[3]を使って、もし背景が人間（250）ならミス変数 M を 1 にしています。

「SAVE 0」でプログラムを保存しておきましょう。

●敵をたくさん出す

これまでは敵が 1 体しか出てきませんでしたが、1 体ではゲームがかんたんすぎるので、敵を 4 体に増やしてみましょう。
敵の存在や座標を配列変数で設定していましたが、これを 4 体分用意します。

```
…(前略)…
210 IF Y%8=6 J=0
220 IF H=237 M=1
240 FOR A=0 TO 12 STEP 4
250 IF [A] GSB 500 ELSE GSB 400
260 NEXT
290 WAIT 3:IF M=0 GOTO 100
…(後略)…
```

敵の配列の先頭番号を表す A を 0,4,8,12 と 4 体分設定する

敵存在配列[A]を使って敵発生サブルーチン、移動サブルーチンを呼ぶ

FOR にもどってくりかえし

ゲームループ部分で、敵発生サブルーチンと敵移動サブルーチンを呼んでいる所を、FOR～NEXT で変数 A を変化させて、敵の数の分だけ呼び出します。
敵を設定する配列変数は以下のようになっています。

0	1	2	3
敵1:存在	敵1:x座標	敵1:y座標	敵1:背景

4	5	6	7
敵2:存在	敵2:x座標	敵2:y座標	敵2:背景

8	9	10	11
敵3:存在	敵3:x座標	敵3:y座標	敵3:背景

12	13	14	15
敵4:存在	敵4:x座標	敵4:y座標	敵4:背景

敵1の先頭が 0 番、敵 2 の先頭が 4 番、敵 3 の先頭が 8 番、敵 4 の先頭が 12 番となっています。
そのため、「`FOR A=0 TO 12 STEP 4`」と、A を 0 から 4 刻みに 12 まで変化させています。

続いて、敵発生サブルーチン、敵移動サブルーチンを、A で参照するように改造します。

```
…(前略)…
400 ' *ENEMY
410 IF RND(50)>0 RTN
420 [A]=1:D=30:E=RND(3)*8+6
430 IF SCR(D,E) RTN
450 LC D,E:?CHR$(237):[A+1]=D:[A+2]
=E:RTN
500 D=[A+1]-1:E=[A+2]
510 LC [A+1],[A+2]:?CHR$([A+3])
520 IF D<0 [A]=0:RTN
530 [A+3]=SCR(D,E)
540 LC D,E:?CHR$(237)
550 IF [A+3]=250 M=1
590 [A+1]=D:[A+2]=E:RTN
```

敵発生を、配列変数 A と変数 D,E を使って設定

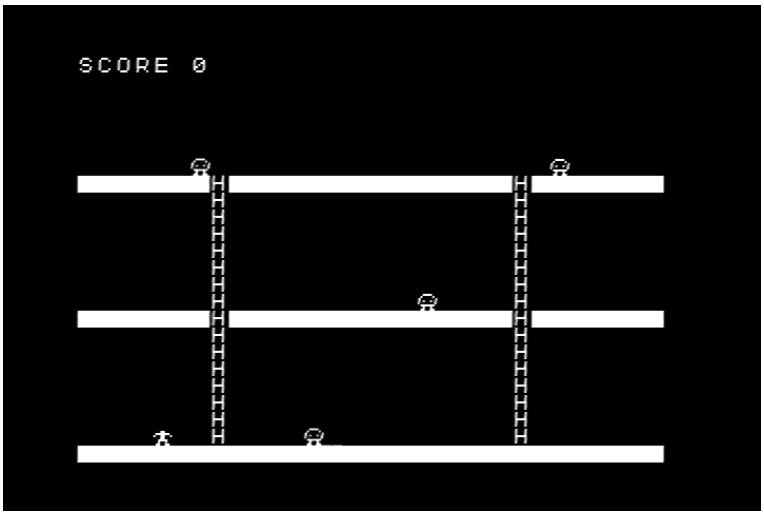
すでに同じ場所に敵がいたら、発生せずにもどる

D,E を配列変数に移す

配列変数を A を使って指定

配列変数の部分を、A で指定するように書きかえます。
430 行は、敵を発生する時に、もしすでに同じ場所に敵がいたら、発生させずに **RTN**（RETURN）でもどります。こうしないと敵同士がかち合ってしまい、移動する時におかしな動きをしてしまいます。
なお、420～450 行で変数 D,E を使っているのは、[A+1],[A+2]といちいち書いているとプログラムが長くなっていまい、後でメモリが足りなくなるからです。

プログラムを実行してみましょう。
敵が 4 体まで出てきます。



「SAVE 0」でプログラムを保存しておきましょう。

●コインを出す

敵ばかりだと人間が得になることが何もないので、コイン「$」を表示して、それを取ると得点が入るようにしましょう。
まずゲームループの部分を改造して、コインを発生するサブルーチン、コインをゲットするサブルーチンを呼ぶようにします。

```
…(前略)…
210 IF Y%8=6 J=0
220 IF H=237 M=1
230 IF H=36 GSB 650
240 FOR A=0 TO 12 STEP 4
250 IF [A] GSB 500 ELSE GSB 400
260 NEXT
270 IF RND(50)=0 GSB 600
290 WAIT 3:IF M=0 GOTO 100
…(後略)…
```

もし背景の文字コードが 36($)なら
コインゲットのサブルーチンを呼ぶ

50 分の 1 の確率で、コインを
発生するサブルーチンを呼ぶ

プログラムの最後に、コインを発生するサブルーチン、コインをゲットするサブルーチンを追加します。

```
…(前略)…
600 '*COIN
610 B=RND(31):C=RND(3)*8
620 IF SCR(B,C) GOTO 610
630 LC B,C:?"$":RTN
650 S=S+1:H=0:LC 6,0:?S:RTN
```

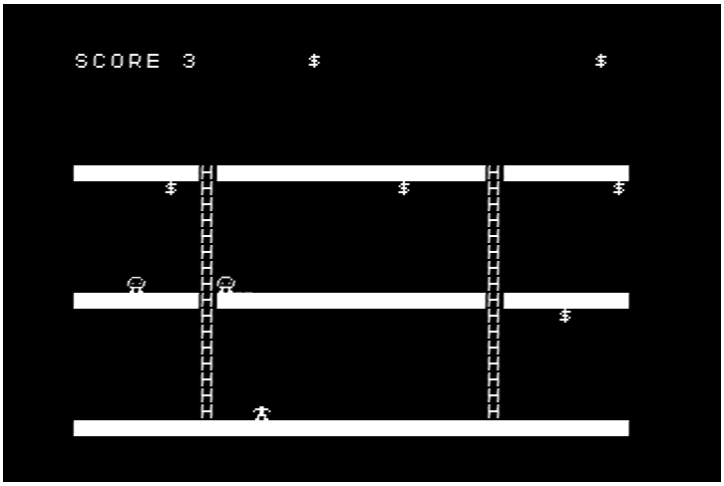
コインの位置を乱数で決める

同じ位置にコインがあったら
位置決めをやり直し

コインを表示して、もどる

スコアを 1 点加算する:背景変数 H を 0 にする:スコアを表示してもどる

プログラムを実行してみましょう。
コインが表示されます。
ジャンプしてコインを取ると、スコアが 1 点増えます。



プログラムの内容を説明します。
ゲームループの中では、コインを発生するサブルーチン、コインをゲットするサブルーチンを呼んでいます。

```
230 IF H=36 GSB 650
```

230 行では、人間の背景を読み取った変数 H を使って、もし背景がコイン（文字コード 36）なら、コインをゲットするサブルーチンを呼んでいます。

```
270 IF RND(50)=0 GSB 600
```

270 行では、**RND(50)** で 0～49 の乱数が出るので、それが 0 になった時（＝50 分の 1 の確率）にコイン発生サブルーチンを呼んでいます。

600～630 行は、コイン発生サブルーチンです。

```
600 '*COIN
610 B=RND(31):C=RND(3)*8
620 IF SCR(B,C) GOTO 610
630 LC B,C:?"$":RTN
```

- 610 行で、乱数を使ってコインを出す座標を決めています。
  x 座標 B は、**RND(31)** で 0～30 のどれかになります。
  y 座標 C は、**RND(3)** で 0～2 が出るので、それを「***8**」（×8）として 0,8,16 のどれかになります。これは人間がジャンプした時の最高地点です。
- 620 行では、610 行で決めた座標にもし何かあれば（他のコインが重なっていれば）、610 行へもどって、位置を決め直します。
- 630 行では、決めた座標にコインを表示して、ゲームループにもどります。

650 行は、コインをゲットした時のサブルーチンです。

```
650 S=S+1:H=0:LC 6,0:?S:RTN
```

スコア変数 S を1加算します。
また、コインをゲットすると消えてなくなるので、背景の変数 H を 0 にします。
加算したスコアを画面に表示して、ゲームループにもどります。

「SAVE 0」でプログラムを保存しておきましょう。

●効果音を出す

これまで静かにゲームが動いていましたが、効果音を出してみましょう。
コインをゲットした時と、ゲームオーバーになった時に、音を出します。

```
650 S=S+1:H=0:BEEP:LC 6,0:?S:RTN
```

コインをゲットした時に音を出す

これで、コインをゲットした時に音が出ます。

ゲームオーバーの時に
音を出す

```
300 LC 11,11:?"GAME OVER":BEEP 30,30:END
```

これで、ゲームオーバーの時に音が出ます。

音を出すには BEEP（ビープ）命令を使います。
BEEP 命令の文法は以下のとおりです。

```
BEEP   30     ,30
       音の高さ  音の長さ
```

音の高さ	1～255 で指定する。省略可能。
音の長さ	60 分の 1 秒単位で指定する。「60」で 1 秒。省略可能。

数字を変えると、音の高さや長さが変わります。いろいろ変えて試してみましょう。

「SAVE 0」でプログラムを保存しておきましょう。