NERD HPC説明書

2025/12/31
ファームウェアv1.12.0

中田商店

目次

1. NERD HPCとは
2. NERD HPCでできること
3. NERD HPCの機能
4. NERD HPCの使い方
5. NERD HPCの仕様
6. ゲームを作ってみよう←とりあえず触りたい人はここから
7. サンプルプログラム集
8. 付録(命令一覧)
9. 付録(NERD HPC裏話)
10. 各種リンク

1. NERD HPCとは

NERD HPCとは一言で表すなら「次世代ポケコン」。今や大手メーカーが全て撤退してしまったポケットコンピュータを「安くて自由で使いやすい」をコンセプトに作り直してみました。

その中でも一番のアピールポイントは「自由」であること。このNERD HPCはありとあらゆることを許可しています(犯罪を除く)。回路図・基板データからファームウェアまで全てが公開されており、NERD HPC用のBASICプログラムを作成するのはもちろんのこと、拡張パーツを作成するのも良し、ハードウェアを改造するのも良し、GitHubにプルリクエストを送るのも良し、PSPのようなカスタムファームウェアを作成するのも良し、解説サイトを作成するのも良し、同人誌を作って販売するのも良し、互換品・改良品を販売するのも良し、プログラミングスクールを開くのも良し。個人利用から商用利用まで全て許可を取らずに勝手にやって大丈夫です。この自由度がこれまでのポケコンには無かった次世代たるポイントかつNERD HPCの真骨頂です。

このようにNERD HPCは中田商店だけが開発する製品ではなく、持っている人皆で発展させていこうという思いが込められています。この解説本がその第一歩となれば幸いです。

2. NERD HPCでできること

NERD HPCは紛うことなきコンピュータです。コンピュータでできることは大抵できます(多分)。

中でもBASICプログラムを組んで何かを作るというのが一番基本的な使い方です。プログラムを組んで作れるものの例としては以下のようなものが挙げられます。

- ゲーム
- 電子工作
- ピアノ
- キーボード・マクロツール
- 赤外線リモコン
- 仕事ツール

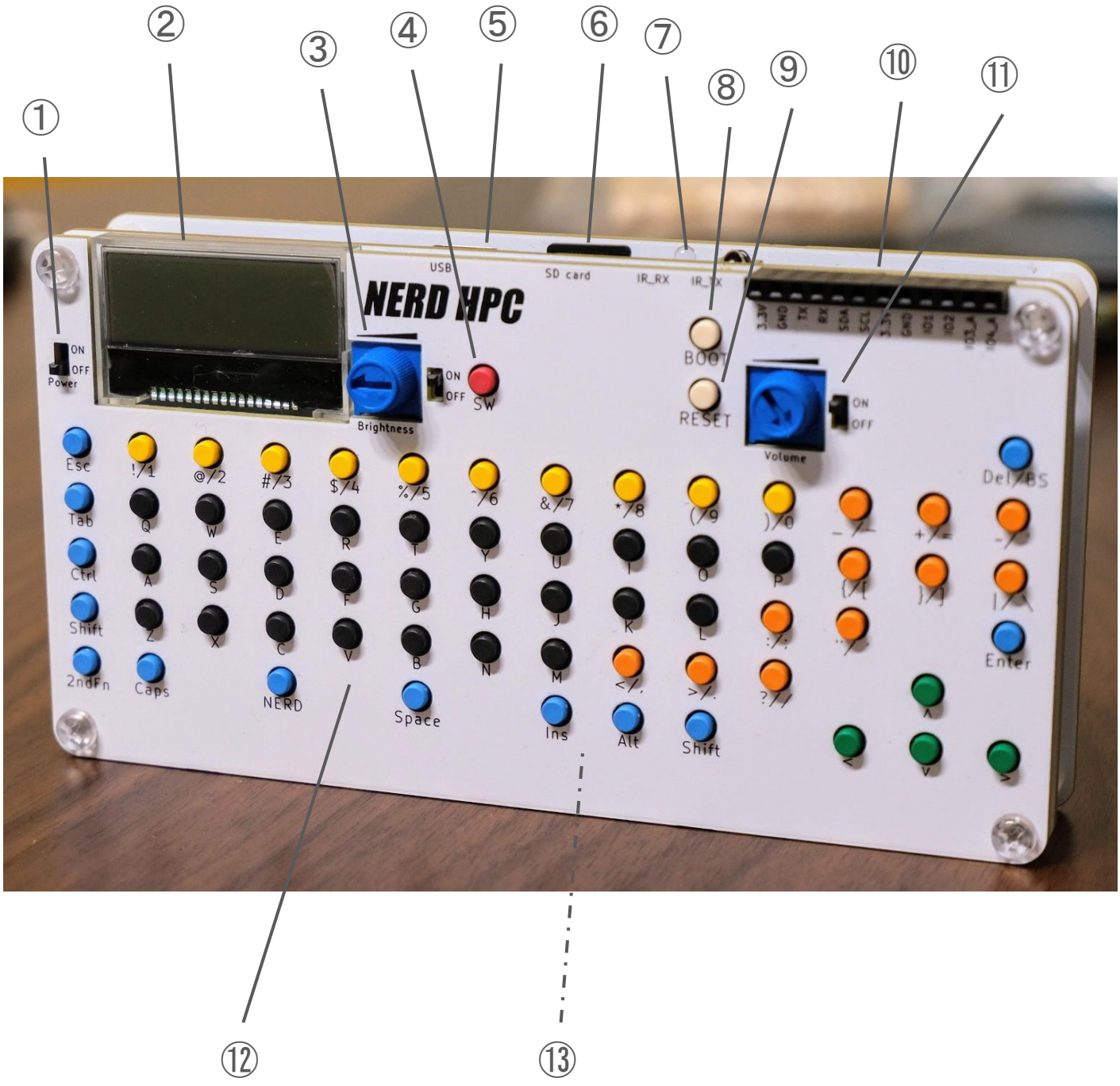
さらにファームウェアに手を加えたり、汎用入出力端子を活用すれば以下のようなこともできます

- オリジナルの命令追加
- ゲームパッドとして利用
- WiFi, Bluetooth拡張

CPUのスペックとしては、元祖FPSのDOOMを移植して実行できるレベルです。高速で動くBASICを楽しみましょう。
↓Youtubeリンク
【RP2040 Doom: DOOM1 Demo/Gameplay on a Raspberry Pi Pico】
https://www.youtube.com/watch?v=eDVazQVycP4

3. NERD HPCの機能

まずはNERD HPCにどういった機能があるのか見てみましょう。



1. 【電源スイッチ】
   ONにすることで電源が入り、OFFにすることで全ての電力が遮断される。

2. 【ディスプレイ】
   48x128ドットのモノクロディスプレイ。この小さなディスプレイをいかに上手く使いこなすかが腕の見せどころ。プログラム実行中は60fpsで動作する。

3. 【バックライトスイッチ&明るさ調整つまみ】
   バックライトのON,OFFを切り替えられる。また、ONの状態でつまみを回すと明るさを調整できる。

4. 【SW】
   謎のボタン。もともとNERD HPCはNERD BOYという自作のワンボタンゲーム機の開発機として開発がスタートし、NERD HPCは当初、NERD DEVと名付けられていた。しかし紆余曲折ありNERD BOYの開発は中断。NERD DEVはNERD HPC(オタクのためのホビーパソコン)と改名し現在に至る。つまりこれはNERD BOYというゲーム機を想定したボタンであるが、今となっては何の機能も割り当てられていない謎のボタンとなってしまった。
   →現在は起動時に押されているとAUTORUN.TXT自動実行無効。

5. 【USB type-C】
   電源供給が可能。PCと接続することでNERD HPCの画面を出力したり、PCのキーボードから入力したり、PCにキー出力できる。

6. 【micro SD cardスロット】
   micro SDカードを入れることでプログラムを保存・読み出しができる。他にもNERD HPCオリジナル形式の動画ファイルを入れることでリアルタイムで読み出しながら再生が可能(開発途中)。micro SDカードはFAT32もしくはexFATでフォーマットしてください。

7. 【赤外線LED&赤外線受信モジュール】
NERD HPC同士の通信やNERD HPCをリモコンとして用いる際に使用する(開発段階)。赤外線受信モジュールは940nmがピーク波長となっており、中心周波数は37.9kHz。

8. 【BOOTスイッチ】
PCに接続した状態でBOOTスイッチを押しながら電源ON、もしくはPCに接続して電源ONの状態でBOOTスイッチを押しながらRESETスイッチを押すことで本体ファームウェア書き換えモードに入れる。本体ファームウェアは日々更新しており、命令の追加、バグ修正が行われるので気が向いたときに更新しておくのが良い。ファームウェアのリンクは各種リンクのGitHubのページ参照。

9. 【RESETスイッチ】
メインのマイコンを再起動するスイッチ。あくまでメインのマイコンを再起動するだけなのであらゆる部品を完全に再起動したい際は電源ボタンのON,OFFがおすすめ(とはいってもたいていのことはRESETスイッチを押せば大丈夫)。先述のファームウェア書き換えモードに入るために使うこともある。

10. 【汎用入出力端子】
UART通信、I2C通信、3.3v供給、入出力に対応。NERD HPC同士の通信やLEDを光らせたりボタン入力ができる。

11. 【ボリュームスイッチ&音量調整つまみ】
ON,OFFで音の有り無しが切り替えられる。また、ONの状態でつまみを回すと音量が調整できる。

12. 【キーボード】
詳しい使い方は後述。一部ダイオードを省いたキーマトリクスで押下されたキーをスキャンしているので同時押しに対応していないキーがあるのは注意。色々な設計思想を盛り込んだので、このキーボードはこだわりポイント。やっぱりCtrlはAの横がいいよね？

13. 【電池BOX】
単4電池 x4本で動作。プラスマイナスの表記が基板に印刷されているので向きを間違えないように注意。充電式電池があると便利かも。

4. NERD HPCの使い方

- キー入力
  - Esc
    プログラムの実行中にEscを押すことでプログラムを中断できます。

  - Caps
    Capsを押すことでShiftを押さなくとも大文字が入力できます。起動時は Capsが押された状態なのでもう一度 Capsを押すと小文字が入力できるようになります。

  - 2ndFn
    2nfFnを押すことでShiftを押さなくとも数字キー、記号キーの /(スラッシュ)の左側の文字が入力できます。もう一度押すと解除されます。

  - Shift
    Shiftを押しながら数字キー、記号キーを押すと /(スラッシュ)の左側の文字が入力できます。Capsを解除している状態だと、Shiftキーを押しながらアルファベットを押すと大文字が入力できます。

  - Del/BS
    入力した文字を削除できます。

  - Enter
    入力を確定します。

  - RESET
    再起動します。

  - BOOT
    PCに接続した状態でBOOTを押しながら電源 ON、もしくはPCに接続して電源ONの状態でBOOTを押しながらRESETを押すと本体ファームウェア書き換えモードに入ります。RPI-RP2という名前でPCに認識されるので、本体ファームウェアファイルをドラッグ・アンド・ドロップすることで更新できます。ファームウェアは各種リンクの GitHubページ参照。

- プログラミング方法
  - プログラムの入力
    命令を入力してEnterを押すことで一行だけその場で実行できます。
    行番号 命令
    と入力してEnterを押すと指定の行番号に命令を入力します。
    (例)
    PRINT ‘hello, world’<Enter>
    10 A=10<Enter>
    20 PRINT A<Enter>

  - プログラムの削除
    NEW
    と入力すると入力済みの全てのプログラムを削除します。
    行番号のみを入力すると指定の行のプログラムを削除します。

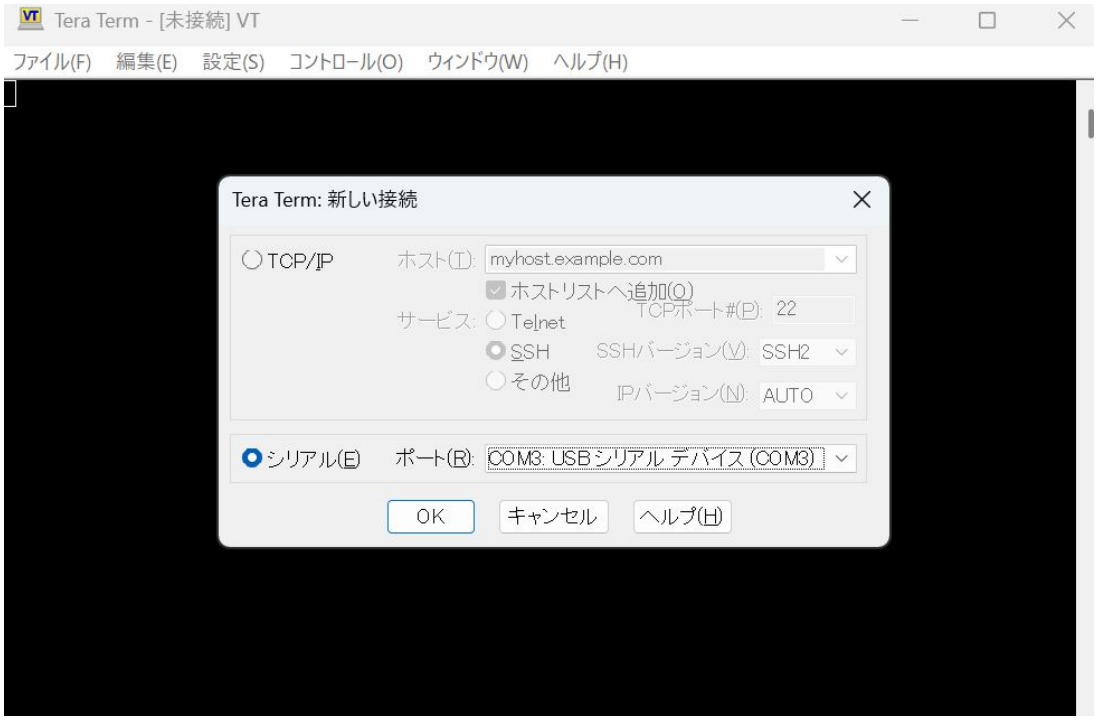
  - プログラムの実行
    RUN
    と入力すると行番号の小さい順から実行します。

  - プログラムの表示
    LIST
    と入力すると行番号が一番小さいプログラムを表示します。右下の矢印キーの上下を押すと表示する行を変更できます。Escを押すとプログラム入力モードに戻れます。
    LIST 行番号
    と入力すると指定の行番号から表示できます (指定の行番号が存在しない場合は繰り上げた行番号から表示)。
    LIST ALL
    と入力すると全てのプログラムを表示する。NERD HPC単体だと表示できる行数が限られているが、後述の PC画面出力を用いるとプログラム全体が確認できて便利です。

- プログラムの保存
  micro SDカードを挿入した状態で、
  SAVE ‘任意のファイル名 .TXT’
  と入力するとSDカードにプログラムが保存できます。.TXTの部分は自由に変更可能ですが、.TXT形式だとSDカードリーダーでファイルを読み取った際に、メモ帳などで開けるので便利です。

- プログラムの読み込み
  プログラムが保存されたmicro SDカードを挿入した状態で、
  LOAD ‘任意のファイル名 .TXT’
  と入力するとSDカードからプログラムを読み込めます。.

- PCとの接続
  NERD HPCはPCに接続することでPCに画面を出力したり、PCのキーボードを使ってプログラムを入力できるのでより快適になります。
  - windowsでの接続方法
    - Tera Termをインストールする。
      https://teratermproject.github.io/
    - 起動し、シリアルにチェックを入れてポートを選択した後に OKを押すと接続できる。



  - MacOSでの接続方法
    - ターミナルを起動し、
      screen /dev/tty.usbmodem1234561 15200
      と入力すると接続できる。/dev/tty.usbmodem1234561の部分は環境によって変わる可能性あり。
    - 終了したいときはCtrl-a k yの順で入力するとで終了できる。
  - Linux(Ubuntu)での接続方法
    - sudo usermod -a -G dialout $(whoami)
      と入力し、シリアル通信にアクセスできるようにする。
    - screen /dev/ttyACM0 115200
      と入力すると接続できる。/dev/ttyACM0の部分は環境によって変わる可能性あり。
    - 終了したいときはCtrl-a k yの順で入力するとで終了できる。

- アップデート方法
  - PCに接続した状態でBOOTボタンを押しながら電源を入れる、もしくは PCに接続して電源を入れた状態で BOOTボタンを押しながらRESETボタンを押すことでPCにRPI-RP2と認識される。
  - NERD HPCのGitHubページからアップデートファイル (.uf2ファイル)をダウンロードし、RPI-RP2にドラッグ・アンド・ドロップすることでアップデートできる。
- その他のプログラミング方法まとめ

  NERD HPCは本体のキーボードで直接プログラムを入力する以外にもプログラミングを行う方法がある。以下にプログラミング方法をまとめる。
  - 本体のキーボードで直接入力

    手軽さはあるが、画面やキーボードが小さいため快適にプログラミングを行うのであれば以下の方法をおすすめする。

  - PCに接続して入力

    前述のPCとの接続を行うとPCの画面に出力し、PCのキーボードから入力することができる。プログラム一覧を表示する LIST ALLコマンドも見やすい。コピー&ペーストもできるのでPC上にテキストファイルを作成し、メモ帳でプログラムを書いて、コピペするというのもおすすめである。

  - SDカードにプログラムを書き込む

    SDカードリーダーライターを用いて SDカードのテキストファイルを直接編集するということも可能。すでに完成したプログラムをコピーしてくる際におすすめ。
- その他機能
  - microSDカードにAUTORUN.TXTというファイルを置くと、起動時に自動的にAUTORUN.TXTの内容を実行する。起動時に SWキーを押していると自動実行を無効。

- エラー一覧

OK	正常終了
Devision by zero	0で除算しようとした
Overflow	値が-32767~32767の範囲を超えている
Subscript out of range	配列の添字が所定の値を超えている
Icode buffer full	指示が長すぎる
List full	プログラム保存領域におさまらない
GOSUB too many nested	GOSUB~RETURNが所定のネスト数を超えた
RETURN stack underflow	RETURNに対応するGOSUBが存在しない
FOR too many nested	FOR~NEXTが所定のネスト数を超えた
NEXT without FOR	NEXTに対応するFORが存在しない
NEXT without counter	NEXTのあとに変数が記述されていない
NEXT mismatch FOR	FORとNEXTの変数が一致しない
FOR without variable	FORのあとに変数が記述されていない
FOR without TO	FORのあとにTOが記述されていない
LET without variable	LETのあとに変数が記述されていない
IF without condition	IFのあとに条件が記述されていない
Undefined line number	分岐先の行が存在しない
'(' or ‘)’ expected	「(」または「)」が記述されていない
'=' expected	「=」が記述されていない
illegal command	プログラムが命令を含んでいる
Syntax error	文法に従っていない
Internal error	豊四季タイニーBASICのバグにより実行できない
Abort by [ESC]	プログラムが[ESC]キーによって中断された
SD card mount error	SDカードが挿入されていない
File not found	指定したファイルが存在しない

5. NERD HPCの仕様

- ハードウェア的仕様
  - MCU: RP2040
    - 32bit(ARM_Cortex-M0+)
    - 133MHz
    - 264kByte RAM
    - 2MByte ROM
  - LCD: AQM1248A
    - 48x128ドット
- BASIC仕様(今後アップデートされる可能性あり)
  - 搭載BASIC: 豊四季タイニーBASIC(書籍「タイニーBASICをCで書く」鈴木哲哉著、GNU General Public License)を拡張したもの
  - 一行に入力できる文字数 : 128文字
  - プログラム保存領域 : 32KByte
  - 配列領域: 255個
  - GOSUBネスト数: 9
  - FORネスト数: 9
- 画面仕様
  - NERD HPCはテキスト画面とグラフィック画面が重なる形で表示されている。テキスト画面は命令を入力する画面、グラフィック画面は絵を描く画面とも言い換えられる。Gから始まる命令はグラフィック画面に対する命令となっている。

6. ゲームを作ってみよう

この章では実際にゲームを作りながらNERD HPCのプログラミング方法に慣れていっていただきたいと思います。

まずはNERD HPCの裏に単4電池4本を入れましょう。基板にプラスマイナスの表記があるので向きに注意して入れましょう。

電池を入れ終わったら、左上のPowerスイッチをONの方向にスライドしましょう。すると以下の表示が出ると思います。

```
TOYOSHIKI TINY BASIC
NERD HPC EDITION
version x.x.x

OK
>
```

これで、あなたはNERD HPCの開発者の一歩を踏み出しました。
「NERD HPCの開発者？自分はただNERD HPCで遊ぶだけなんだけど…」と思ったかもしれません。自身がNERD HPCを触っていただく方を開発者と呼んでいるのはNERD HPCは自分ひとりだけで開発するものではなく、皆さんと共に開発・発展させていくものという思想で作られているからです。例えば、ゲームを作って共有してみる・誰かが作ったゲームを遊んで感想を伝えてみる・ただただ部屋に飾ってTwitterに上げてみるといった行動全てがNERD HPCという「コンテンツの開発・盛り上がり」に繋がります。NERD HPCは今が完成形というわけではありません。ユーザーのフィードバックを受けてどんどん成長を続けていくこれまでのポケコンにはない要素を持っています。また、飽きたら机の奥にでもしまっておいて大丈夫です。数年後ふと思い出して取り出したとき、温かく迎え入れられるように昔と形が変わらないままのNERD HPCと、進化したソフトウェア・発展したコミュニティを用意して待っておきます。

これから順にゲームの作り方を解説していきますが、「ちまちまやってられねェ！」と思った方はこの章の最後に完成したプログラム全体を載せていますのでそちらを入力していただけるとすぐにゲームで遊べます。自分はそっちのタイプです。

……

この行を読んでいるということはあなたは辛抱強いタイプですね？であれば、まずはこちらの画面から説明をしたいと思います。一、二行目の「TOYOSHIKI TINY BASIC NERD HPC EDITION」という文字。こちらは豊四季タイニーBASICという、小さなコンピュータ向けに作られたBASIC実行環境をNERD HPC向けに調整・拡張したものだという意味です。その下のversion x.x.xというのは本体ファームウェア(NERD HPCに組み込まれたBASIC実行環境のこと)のバージョンを意味しています(もし表紙のバージョンより低ければアップデート推奨)。そして準備OKという表示があり、その下の「>」はプログラムが入力可能であるということを表しています。

```
TOYOSHIKI TINY BASIC
NERD HPC EDITION
version x.x.x

OK
>
```

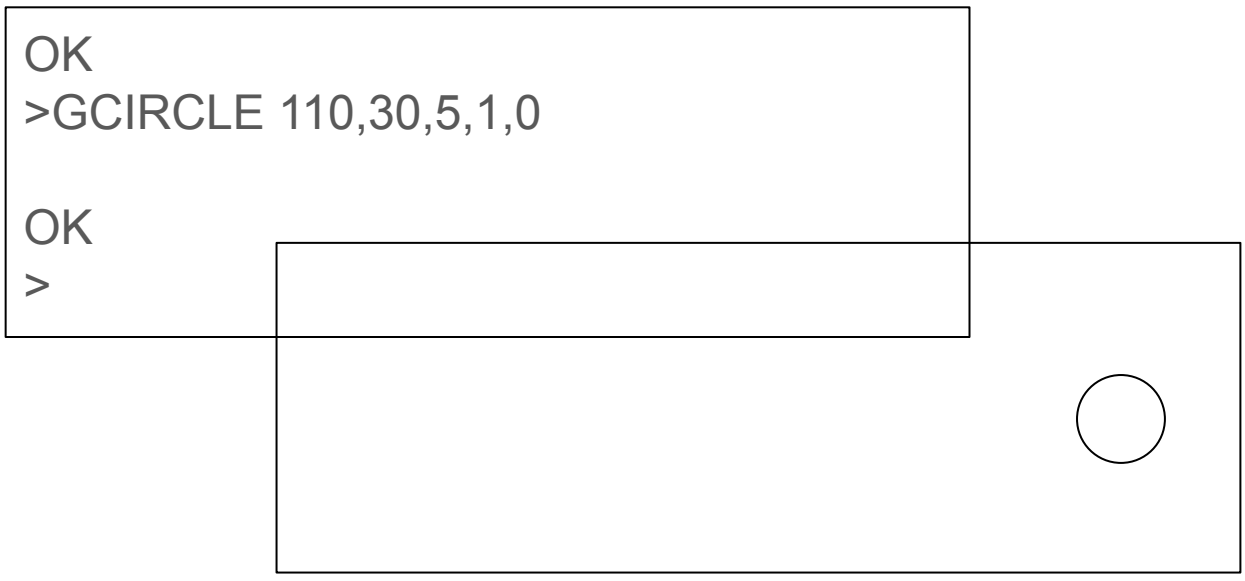
NERD HPCは、れっきとしたコンピュータです。命令を与えることで初めて動きます。試しに以下の「>」の後の文字を打ち込んでEnterキーを押してみてください。

```
TOYOSHIKI TINY BASIC
NERD HPC EDITION
version x.x.x

OK
>GCIRCLE 110,30,5,1,0
```

すると以下のように画面の右のあたりに丸が出現したのではないでしょうか。

そうです、このGCIRCLEという命令は丸を描画する命令です。なぜCIRCLEではなく"G"CIRCLE？と思ったかもしれません。鋭いですね。NERD HPCは2つの画面を重ね合わせて表示しています。どういうことかと言うと、一つはテキスト画面、OKや>GCIRCLE…といった表示が出ているのがそれです。そしてもう一つがグラフィック画面。丸が表示されているのがそれにあたります。テキスト画面は命令を出す画面、グラフィック画面はお絵描きをする画面と言い換えることもできます。この2つの画面を重ね合わせており、先程のGCIRCLEの"G"は、このグラフィック画面に対する命令という意味だったのです。

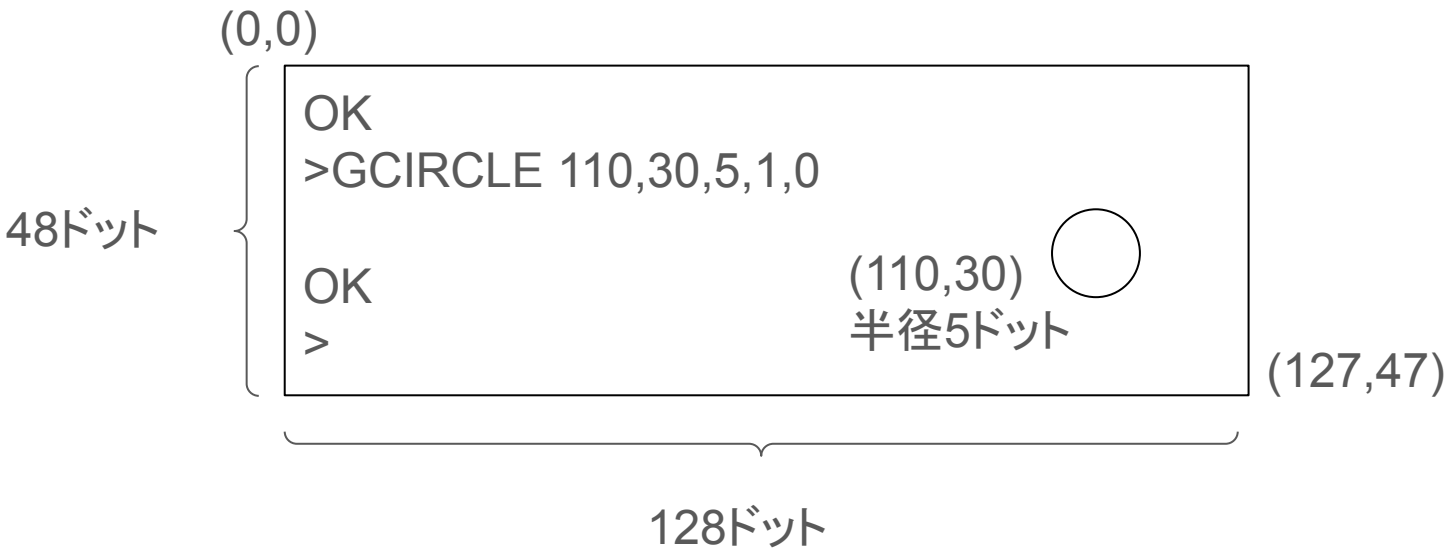
テキスト画面



グラフィック画面

続いて、GCIRCLEの後ろに付いている数字についても解説していきます。命令の後ろに付ける数値や文字のことを引数を呼び、これは順に、円の中心X座標, 円の中心Y座標, 円の半径, 円の色, 塗りつぶすかどうか、を指定しています。

NERD HPCの画面は横128ドット、縦48ドットとなっており、左上が(0,0)右下が(127,47)となっています。円の色というのは円の外形及び、塗りつぶしの色のことを指しており、0で白、1で黒が指定できます。また、最後の引数である塗りつぶしは0で塗りつぶさない、1で塗りつぶすと指定することができます。



続いて、グラフィック画面の内容を削除したい時の命令を教えます。
GCLSと入力してEnterを押してみてください

```
OK
>GCLS

OK
>
```

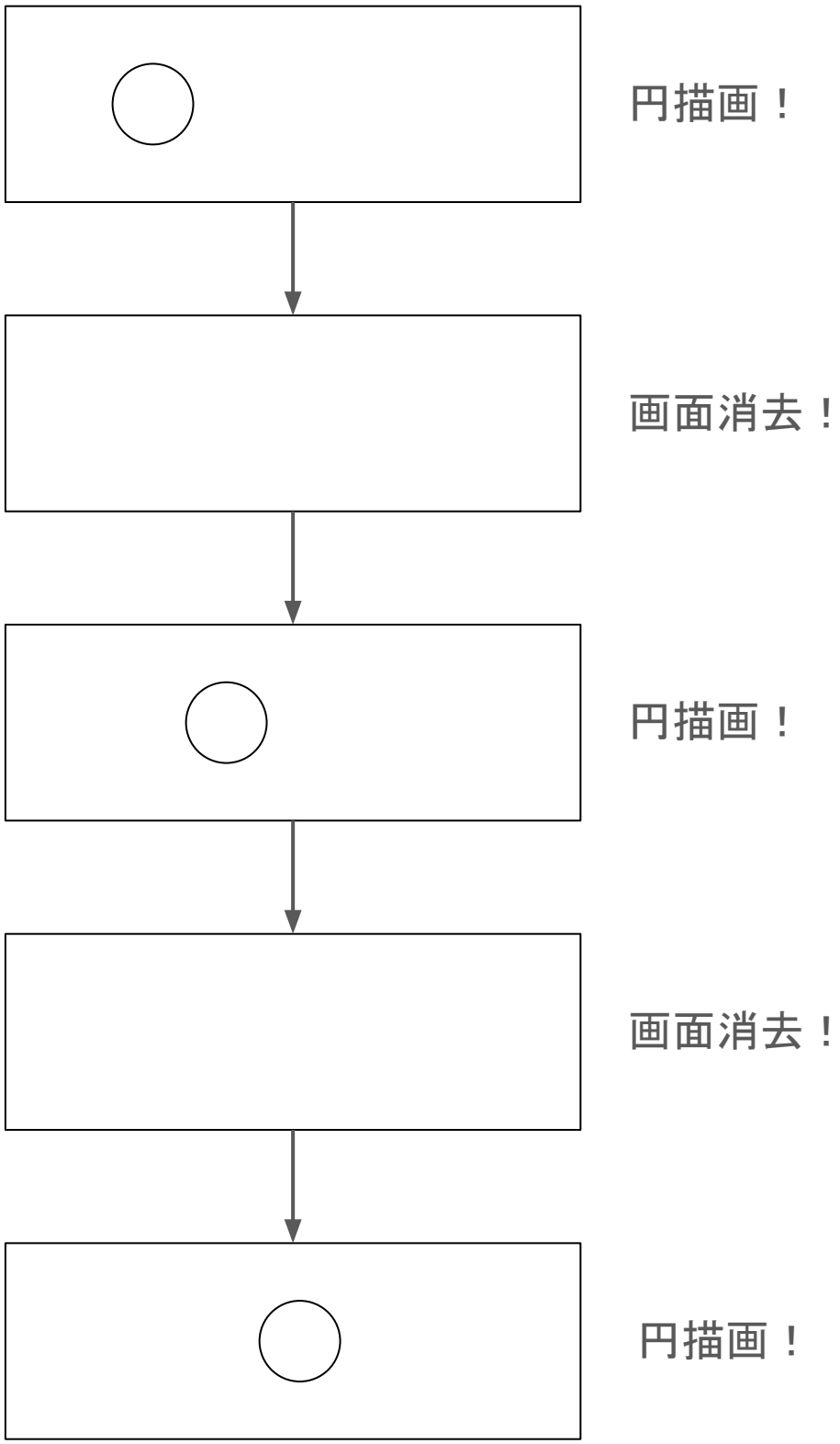
丸が消えたのが分かると思います。先程のGCLSという命令は"G"がついているのでグラフィック画面に対して画面消去(CLSはclear screenの略)を行う命令です。

今度はCLSと入力してEnterを押してみてください。

```
OK
>
```

こちらはテキスト画面に対して画面消去を行う命令なのでテキスト画面がきれいになりました。このCLS, GCLSという命令はゲームを作るうえで非常に重要です。例えば円を動かしたいとなった時、画面消去とちょっとずつずらしながらの円描画を交互に行うことで人間の目には円が動いているように見えます。ちょうどパラパラ漫画のような感じです。

これを高速で行うことで絵が動いているように見せています。



さらに深堀りして表示の仕組みについても解説したいと思います。先程の例だと「円が描画されている間に真っ白な画面が挟まるため、円が点滅するような感じでチカチカするのでは？」と思った方がいるかもしれません。はい、実はこれをそのまま表示すると円がチラついて見える可能性がありますが、対策もあります。まず表示の仕組みを説明すると、NERD HPCは常にリアルタイムでテキスト画面・グラフィック画面の内容を表示しているのではなく1秒間に60回の間隔でテキスト画面・グラフィック画面の内容を表示しているのです。つまりその1秒間に60回のタイミングに間に合うように絵を完成させれば画面消去を行ったとしても気づかれないのです。例えるならジャンプのような週刊で出しているような漫画家さんはYoutubeのライブ配信でリアルタイムで漫画を描いているところを見せているわけではなく、週1回の締め切りのタイミング(NERD HPCにおいては1秒間に60回の表示タイミング)までに描き上げることで読者は漫画家が消したり描いたりしているところを見ることなく完成した漫画だけが見れるというわけです。このように1秒間に60回のタイミングで表示しても問題ない画面を用意することできれいなアニメーションになるのです。その具体的な方法は後ほど説明します。

続いて連続で命令を実行したい場合どのようにすればすれば良いか解説します。連続で命令を実行したい場合、命令の前に数字を入れます。次のように入力してみてください

```
OK
>10 A=1+1
>20 PRINT A
>RUN
```

するとRUNの後に2という表示が出たと思います。実はこれでA=1+1という命令とPRINT Aという命令を連続で実行することができました。

```
>20 PRINT A
>RUN
2

OK
>
```

RUNという命令を実行すると命令の前に入力した数字が小さい順に実行していきます。ではここでプログラム全体を見てみましょう。LIST ALLと入力してみてください。

```
>20 PRINT A
>RUN
2

OK
>LIST ALL
```

するとLIST ALLに続いてプログラム全体が表示されました。

```
>LIST ALL
10 A=1+1
20 PRINT A

OK
>
```

このようにプログラム全体を確認したい場合はLIST ALLと入力してみてください。

また、プログラムを修正したい場合、同じように修正したい番号の後に続いて命令を入力することでその番号の命令を上書きできます。また修正したい番号のみを入力した場合、その番号の命令を削除することもできます。

```
>LIST ALL
10 A=1+1
20 PRINT A

OK
>5 B=2+3 ←プログラムの追加
>5 B=10+10 ←プログラムの修正
>5 ←プログラムの削除
```

ここでふと、番号を10飛ばしで入力していることに疑問を持った方がいるかもしれません。なぜ1,2というように入力しないのか。その理由はプログラムを追加する時にあります。例えばA=1+1とPRINT Aの間にC=2+2 PRINT Cという命令を追加したいとします。10,20と番号を付けていれば

>13 C=2+2
>17 PRINT C

というように10～20の間の番号を適当に振って簡単に追加できますが、1,2と番号をつけていた場合、先程のプログラムの追加に加えてPRINT Aの番号を4番に振り直す必要があります。

>2 C=2+2
>3 PRINT C
>4 PRINT A

これが一行だけであればまだ大丈夫ですが、数十行、数百行になってくるとプログラムを一行追加するだけで番号振り直しの作業がとてつもないことになります。それゆえ行番号は10飛ばしにすることがおすすめとされているのです。

では試しに先程のプログラムを追加してプログラム全体を確認してみましょう。

```
OK
>13 C=2+2
>17 PRINT C
>LIST ALL
```

おや？10 A=1+1が表示されていませんね。NERD HPCの画面サイズの関係上、一度に表示できる行数に上限があるのです。

```
13 C=2+2
17 PRINT C
20 PRINT A

OK
>
```

そこでLISTとだけ入力してみてください。

```
13 C=2+2
17 PRINT C
20 PRINT A

OK
>LIST
```

すると次のように表示されたと思います。

```
10 A=1+1
```

LISTとだけ入力すると一行ずつ確認するモードに入り、キーボード右下の上下ボタン(^ v)を押すことで一行ずつ確認することができます。このモードから抜けるためにはキーボード左上のEscキーを押すことで抜けることができます。

また、途中の行から表示したい場合LISTの後に番号を入力します。

```
OK
>LIST 17
```

これで17番から確認することができました。

```
17 PRINT C
```

これにて一通りのプログラムの入力方法を説明し終えたので、いよいよゲームプログラムをつくっていきます。先程の画面をEscキーで抜けてください。その後NEWと入力することで先程入力したプログラムを全て削除できます。

```
OK
>NEW

OK
>
```

ではとりあえず以下のプログラムをポチポチ入力していってください。行頭の番号も忘れずに。入力し終えたらLIST命令で正しく入力できたか確認してみるのも良いでしょう。途中小文字が出てきていますが、こちらはキーボードの左下のCapsキーを押すことで小文字が入力できるようになります。再度大文字を打つにはもう一度Capsキーを押すか、Shiftを押しながらアルファベットを打つことで大文字が打てます。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'

1000 REM 'INITIALIZE'
1999 RETURN

2000 REM 'LOOP'
2010 GOSUB 3000; REM 'input'
2020 GOSUB 4000; REM 'update'
2030 GOSUB 5000; REM 'output'
2990 VSYNC
2999 GOTO 2000

3000 REM 'INPUT'
3999 RETURN

4000 REM 'UPDATE'
4999 RETURN

5000 REM 'OUTPUT'
5999 RETURN
```

これはゲームプログラムのための雛形です。大抵のゲームプログラムは数値や画面の初期化後、入力→状態更新→出力という処理をループするという流れになっており、先程のプログラムはその流れを書いたものです。

新たに出てきたコマンドを説明すると、

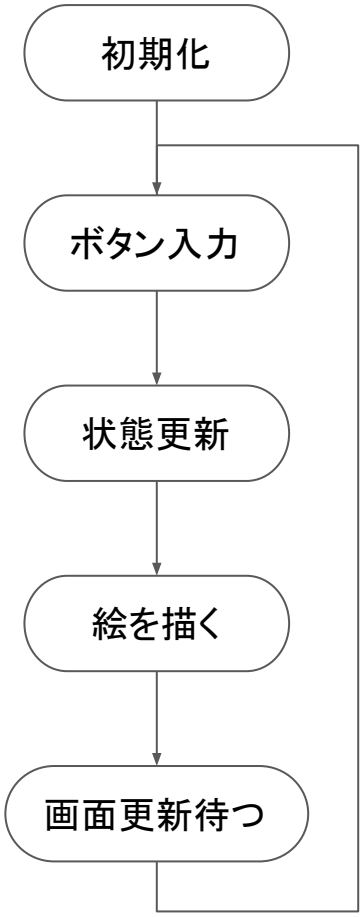
REMという命令はその後の文字を無視します。コメントを書く際に使います。コメントだと分かりやすいように「’ ‘」で囲っていますが、「’ ’」は無くても大丈夫です。

GOSUBというのは後に指定した行までジャンプしてそこから実行するというものです。RETURN命令を受けると元の場所まで戻ってきます。

「;」を付けることで一行に複数の命令を書くこともできます。ここでは10 GOSUB 1000; REM ‘initialize’とすることでその行が何をしているか説明するためのコメントを残しています。

2990にVSYNCという命令がありますが、これは以前説明した1秒間に60回ある画面更新のタイミングまで待つという命令です。画面更新のタイミングで絵を描いていると未完成の絵が原因で画面がチラつくことになります。2030 GOSUB 5000: REM 'output'で画面を描き終えたら新たに消して描いたりするような処理に進まず大人しく画面更新を待つというわけです。

図にするとこのような感じです。



では初期化(REM ‘INITIALIZE’)、ボタン入力(REM ‘INPUT’)、状態更新(REM ‘UPDATE’)、絵を描く(REM ‘OUTPUT’)の部分に処理を追加してゲームを作っていきたいと思います。

その前に、長いプログラムを入力したので努力が水の泡にならないうちにプログラムを保存しておきましょう。micro SDカードをNERD HPCに差し込んだ後に次のように入力しましょう。

```
>SAVE ‘GAME.TXT’

OK
>
```

OKと表示されてプログラムが保存されました。次にNEW命令でプログラムを削除した後にプログラムをSDカードからロードしてみましょう。LIST命令を実行するとプログラムが読み込めていることが確認できるはずです。

```
OK
>NEW

OK
>LOAD ‘GAME.TXT’

OK
>LIST
```

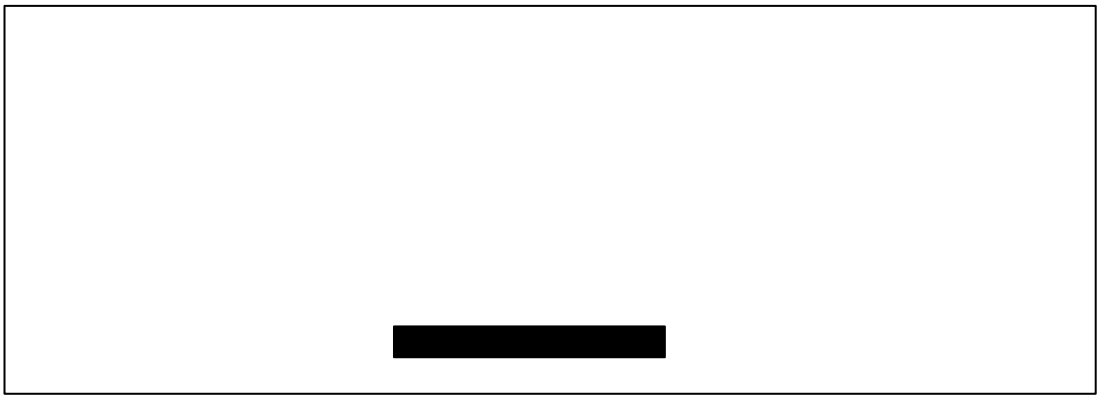
プログラムはSDカードにテキストファイル形式で保存されているのでSDカードリーダーを使ってmicro SDカードをパソコンで読み込むとwindowsメモ帳などを使って直接編集することもできます。

以降任意のタイミングでプログラムを保存してください。保存しないうちにうっかり電源を切らないように注意です。

ではプログラムの追加に話を戻し、新たに追加した赤文字のところを入力していってください。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'

1000 REM 'INITIALIZE'
1010 CLS
1020 GCLS
1030 X=50; REM 'bar left up x'
1040 Y=40; REM 'bar left up y'
1050 W=30; REM 'bar width'
1060 H=3; REM 'bar height'
1999 RETURN

2000 REM 'LOOP'
2010 GOSUB 3000; REM 'input'
2020 GOSUB 4000; REM 'update'
2030 GOSUB 5000; REM 'output'
2990 VSYNC
2999 GOTO 2000

3000 REM 'INPUT'
3010 A=GETKEY(0)
3999 RETURN

4000 REM 'UPDATE'
4010 IF (A=249) * (X+W<127) X=X+1
4020 IF (A=250) * (0<X) X=X-1
4999 RETURN

5000 REM 'OUTPUT'
5010 GCLS
5020 GRECT X,Y,X+W,Y+H,1,1
5999 RETURN
```

実行時にフリーズすることもあるのでこまめな保存がおすすめです。問題なければRUNで実行してみてください。



何やら横長の棒が表示されましたね。キーボード右下の左右キー(< >)を押してみてください。この棒が左右に動くことが確認できるかと思います。これで入力→状態更新→出力の流れに沿って動く小さなプログラムを作成できました。

先程の追加した部分を解説すると、最初に1010 CLS, 1020 GCLSで画面をきれいにしておきます。1030～1060で変数に初期値をセットします。変数はA-Zまであり、数値や文字を格納することができます。変数を用いることで図形を動かしたり、一部を変更するだけでその変更をプログラム全体に反映させることができます。3010 A=GETKEY(0)ではキーボードで入力した文字の情報をAにセットしています。4010 IF (A=249) * (X+W<127) X=X+1 ではIF文というものが使われています。IFの後の数値が0以外の時、後ろの命令を実行します。今回の場合、A=249(入力されたキーが右キーである場合)とX+W<127(バーの右端が画面右端に接触していない場合)の掛け算(プログラミング言語では掛け算を「*」で表します)を行っているので、このA=249とX+W<127という2つの式を同時に満たす時、後ろのX=X+1(バーを右に動かす)という命令が実行されます。いわゆる論理積というやつです。とりあえずA=249といった条件式は満たした場合に1に、満たさなかった場合に0に置き換わるということを覚えておいてください。また、2つの式どちらか一方だけ条件を満たす場合に後ろの命令を実行したいという場合、「+」で2つの式を繋ぐとそれが実現できます。これを論理和と言ったりもします。

同様に、4020 IF (A=250) * (0<X) X=X-1はA=250(入力されたキーが左キーの場合)と0<X(バーの左端が画面左端に接触していない場合)の両方を満たす場合、X=X-1(バーを左に動かす)という命令を実行します。これらの条件を考慮せず画面外にバーを動かそうとするとフリーズするためこのようにバーを動かす前に条件式で判定しています。そして5010 GCLSで画面を消去したのち、5020 GRECT X,Y,X+W,Y+H,1,1でバーを描画しています。

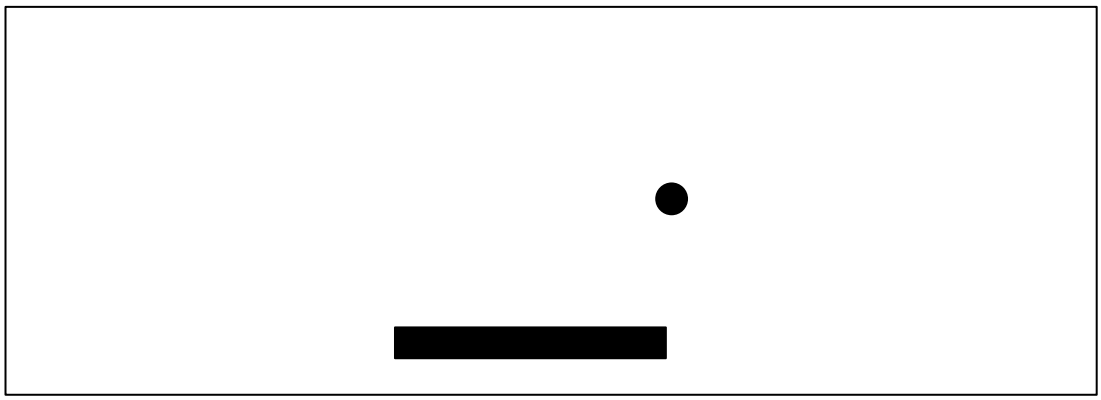
このように入力→状態更新→出力という流れになっていることが分かりましたでしょうか？今後もこの流れに沿ってゲームを作っていきます。

さて、左右に動く棒を見たらなんだか反射する玉を追加したくなってきました。同様に反射する玉を追加していきましょう。Escキーでプログラムを中断できます。棒の表示が残っているので邪魔であればGCLSで消去しておきましょう。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'

1000 REM 'INITIALIZE'
1010 CLS
1020 GCLS
1030 X=50; REM 'bar left up x'
1040 Y=40; REM 'bar left up y'
1050 W=30; REM 'bar width'
1060 H=3; REM 'bar height'
1070 S=64; REM 'ball x'
1080 T=24; REM 'ball y'
1090 R=2; REM 'ball r'
1100 P=1; REM 'ball vx'
1110 Q=1; REM 'ball vy'
1120 C=2; REM 'ball vx/?'
1130 D=0; REM 'for ball vx/? calc'
1140 E=2; REM 'ball vy/?'
1150 F=0; REM 'for ball vy/? calc'
1999 RETURN
```

```
2000 REM 'LOOP'
2010 GOSUB 3000; REM 'input'
2020 GOSUB 4000; REM 'update'
2030 GOSUB 5000; REM 'output'
2990 VSYNC
2999 GOTO 2000

3000 REM 'INPUT'
3010 A=GETKEY(0)
3999 RETURN

4000 REM 'UPDATE'
4010 IF (A=249) * (X+W<127) X=X+1
4020 IF (A=250) * (0<X) X=X-1
4030 IF (S-R+P<0)+(127<S+R+P)P=-P
4040 IF (T-R+Q<0)+(47<T+R+Q)Q=-Q
4050 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)Q=-ABS(Q)
4060 D=D+1
4070 IF C=D S=S+P
4080 IF C=D D=0
4090 F=F+1
4100 IF E=F T=T+Q
4110 IF E=F F=0
4999 RETURN

5000 REM 'OUTPUT'
5010 GCLS
5020 GRECT X,Y,X+W,Y+H,1,1
5030 GCIRCLE S,T,R,1,1
5999 RETURN
```

では保存して実行してみましょう。RUNっと…



なにやら見たことある感じの画面が出てきましたね。もうお分かりかと思いますが、今回はブロック崩しを作っていきます。

では先程追加したプログラムの解説をしていきます。

1070～1150でボールに関する変数を用意します。ボールのX座標(S)、Y座標(T)、半径(R)、横方向速度(P)、縦方向速度(Q)、C,D,E,Fはボールの速度を1以下にするために用いります。小数が扱えないためこれらの変数を用いて擬似的に少数を再現します。P/C、Q/Eがボールの速度となります。少数を再現する処理はこの後解説します。4030 IF (S-R+P<0)+(127<S+R+P)P=-Pで画面横端にボールが当たった際に横方向の速度を反転させています。
4040 IF (T-R+Q<0)+(47<T+R+Q)Q=-Qで画面縦端にボールが当たった際に縦方向の速度を反転させています。4050 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)Q=-ABS(Q)でボールの下部がバーに当たったり、めり込んでいる場合、ボールが上に反射するようにしています。ABS()というのは括弧内の数値を正の数に変換する命令です。例えばABS(-1)は1ということになります。4060～4110で少数を再現する処理を行っています。計算用変数D,Fに1を加算し、速度の分母であるC,Eと一致したときだけ速度の分子のP,Qをボールの位置に加算するという処理になっています。もし分からなければ処理を追って実際に各変数の動きを見てみれば理解できるかと思います。そして5030 GCIRCLE S,T,R,1,1でボールを描画します。これで追加した処理は以上です。

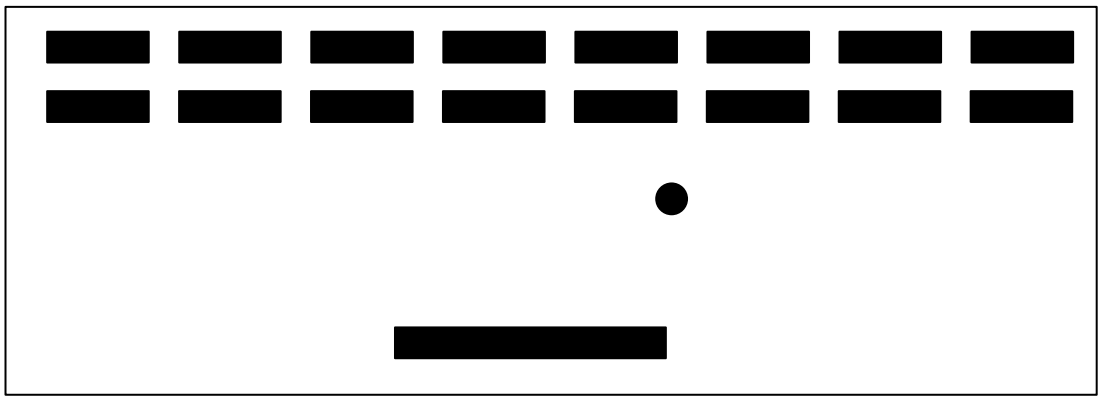
ここまでくればあと一息です。棒と玉があって後足りないものは…そうです、崩せるブロックです。しかしここで問題となってくるのが変数の数です。変数はA~Zの26文字しか使えず、現状13個使用済みです。ブロックを2行x8個=16個用意して、棒と同じく一つのブロックにX座標、Y座標、幅、高さ、それに加えてブロックが崩れたかの判定の5つの変数を利用すると13+16×5=93個変数が必要となり、到底足りない計算になります。そこで工夫をして利用する変数を減らしていくことにします。とりあえずブロックを追加したプログラムがあるので入力していってください。それを元に解説していきます。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'

1000 REM 'INITIALIZE'
1010 CLS
1020 GCLS
1030 X=50; REM 'bar left up x'
1040 Y=40; REM 'bar left up y'
1050 W=30; REM 'bar width'
1060 H=3; REM 'bar height'
1070 S=64; REM 'ball x'
1080 T=24; REM 'ball y'
1090 R=2; REM 'ball r'
1100 P=1; REM 'ball vx'
1110 Q=1; REM 'ball vy'
1120 C=2; REM 'ball vx/?'
1130 D=0; REM 'for ball vx/? calc'
1140 E=2; REM 'ball vy/?'
1150 F=0; REM 'for ball vy/? calc'
1160 J=2; REM 'block left up x'
1170 K=2; REM 'block left up y'
1180 L=12; REM 'block width'
1190 M=3; REM 'block height'
1200 FOR I=0 TO 15
1210 @(I)=1; REM 'block display flg'
1220 NEXT I
1999 RETURN

2000 REM 'LOOP'
2010 GOSUB 3000; REM 'input'
2020 GOSUB 4000; REM 'update'
2030 GOSUB 5000; REM 'output'
2990 VSYNC
2999 GOTO 2000
```

```
3000 REM 'INPUT'
3010 A=GETKEY(0)
3999 RETURN

4000 REM 'UPDATE'
4010 IF (A=249) * (X+W<127) X=X+1
4020 IF (A=250) * (0<X) X=X-1
4030 IF (S-R+P<0)+(127<S+R+P)P=-P
4040 IF (T-R+Q<0)+(47<T+R+Q)Q=-Q
4050 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)Q=-ABS(Q)
4060 D=D+1
4070 IF C=D S=S+P
4080 IF C=D D=0
4090 F=F+1
4100 IF E=F T=T+Q
4110 IF E=F F=0
4120 FOR I=0 TO 7
4130 @(31)=J+I*16<=S
4140 @(30)=S<=J+I*16+L
4150 @(29)=K<=T
4160 @(28)=T<=K+M
4170 IF @(I)*@(31)*@(30)*@(29)*@(28) Q=-Q
4180 IF @(I)*@(31)*@(30)*@(29)*@(28) @(I)=0
4190 @(29)=K+7<=T
4200 @(28)=T<=K+7+M
4210 IF @(I+8)*@(31)*@(30)*@(29)*@(28) Q=-Q
4220 IF @(I+8)*@(31)*@(30)*@(29)*@(28) @(I+8)=0
4230 NEXT I
4999 RETURN
```

```
5000 REM 'OUTPUT'
5010 GCLS
5020 GRECT X,Y,X+W,Y+H,1,1
5030 GCIRCLE S,T,R,1,1
5040 FOR I=0 TO 7
5050 IF @(I) GRECT J+I*16,K,J+I*16+L,K+M,1,1
5060 IF @(I+8) GRECT J+I*16,K+7,J+I*16+L,K+7+M,1,1
5070 NEXT I
5999 RETURN
```

保存して実行してみましょう。RUN RUN♪



はい、ブロック崩しが完成しました。お疲れ様です。

…え？クリア画面もゲームオーバー画面もない？いいんです、とりあえず完成させたという気持ちになることが大事です。メインストーリーやっているときよりエンドコンテンツやっているときのほうがやるべきことをやって余生を楽しんでいる気分になれますから。そういった機能は後でちまちま実装していきましょう。

では先程のプログラムを解説していきます。

1160～1190の部分でブロックの一番左上のX座標、Y座標、幅、高さを決めています。左上の座標さえ決まればあとは等間隔で配置していけばよいのでブロックの個数分の座標は不要です。同様に幅、高さも全て同じなので個数分の変数は不要です。1200～1220の部分で16個ある各ブロックが消えたかどうかのフラグを作っています。1200 FOR I=0 TO 15、1220 NEXT Iという文はIという変数を0から15まで変化させながらNEXT Iという文で囲まれた部分を繰り返し実行するという意味です。囲まれた部分に1210 @(I)=1; REM 'block display flg'という文があると思いますが、@()というのは配列というものです。簡単に言えば変数の集まりだと思ってください。ただし変数と違うのは括弧内の数字を変えながら中身にアクセスできるという点です。このフラグを変数で管理しようと思うと、A=1、B=1、C=1…というように16行に渡って処理を書かないといけませんが、先程のFOR,NEXT文と組み合わせることで配列の0番目から15番目までまとめて1を設定することができます。

実は2進数という概念を使えば2つの変数だけで16個のブロックの状態を管理できるのですが、わかりやすさを重視して配列16個分使うことにしました(本当は処理を考えるのが大変だっただけ)。気が向いたらチャレンジするのも面白いかもしれません。ちなみに配列は@(0)～@(31)までの32個が使えます。続いて4120～4230の部分、ここはブロックと玉が当たったか判定し、当たっていれば先程のブロック表示のフラグを0にし、上下方向の速度を反転させます。本当は上下から当たった場合、左右から当たった場合、角から当たった場合に分けて反射させ方を変えるのが一般的なのですが、プログラムが長くなってしまうのでボールの中心がブロックの中に入った時、上下方向が切り替わるという簡易的な処理にしました。ここも時間がある時処理を変えてみるといいかもしれません。4170 IF @(I)*@(31)*@(30)*@(29)*@(28) Q=-Qにて配列を使っているのは一行に入力できる文字数の限界があるためです。配列の空きを利用して、事前に条件の判定結果を@(28)～@(31)に格納しておくことで文字数制限の問題を回避しています。最後に5040～5070でブロックの表示を行っています。@(0)～@(7)で上の段、@(8)～@(15)で下の段の表示を管理しているので左から順に上下に2個ずつ表示していっているイメージです。

…ということでNERD HPCを触ってみていかがでしたでしょうか？現代の高級プログラミング言語に触ったことある方だと制限も多く、中々面倒くさいなと思ったかもしれません。しかしながらこのプログラムを動かしているチップは驚くべきことに100円程度のチップで、搭載している記憶容量は現在のスマホカメラで取った写真一枚にも満たない2メガByteなのです。普通のPCのCPUが数万円し、ギガByte級の記憶容量を備えていることを考えると頑張っている方だと思いませんか？これらの制限があるのも限られた資源を最大限活用するための制約なのです。この制限の中で可能性を探っていく、それがNERD HPCの面白さなのです。

では、ここから後はエンドコンテンツです。先程完成したブロック崩しに機能を付け加えていきます。もしくは付録の命令一覧を見て自分のプログラムを作り始めてもいいかもしれません。

それでは何の機能を実装していきましょうか。ゲームオーバー画面？クリア画面？画面を触るのもいいですが、NERD HPCにはスピーカーが搭載されています。せっかくなのでこれを利用してみましょう。反射した際に音が鳴るようにしてみたいと思います。プログラムが長くなってきたので追加する部分以外は省略しています。

```
(省略)
1220 NEXT I
1230 N=0; REM 'reflect flg calc'
1240 O=0; REM 'reflect flg'
1250 V=0; REM 'for STPSND'
1999 RETURN
(省略)
4000 REM 'UPDATE'
4002 N=P+Q
4005 O=0
4007 V=V+1
4010 IF (A=249) * (X+W<127) X=X+1
(省略)
4230 NEXT I
4240 IF N=P+Q O=0
4250 IF N#P+Q O=1
4999 RETURN
(省略)
5070 NEXT I
5080 IF O PLYSND RND(2000)+300,0
5090 IF O V=0
5100 IF V>=7 STPSND
5999 RETURN
```

ではスピーカーの電源をONにしてVolumeを調整して実行してみてください。何かにぶつかるたびに音が鳴ると思います。

追加した部分を解説していきます。NはP+Q(横方向の速度と縦方向の速度の和)を状態更新処理の最初と最後で監視し、反射すればP+Qの値が変化するので、それに応じてOにフラグをセットします。もし反射していればPLYSNDという命令で音を鳴らします。PLYSNDは1つ目の引数で鳴らす音の周波数、2つ目の引数で鳴らす秒数(ミリ秒)を指定します。特定の周波数の音を鳴らすのも良いですが、せっかくなのでランダムな音が鳴るようにしました。RND()という命令は1～括弧内で指定した数値をランダムで返します。また、RND(2000)に300を加えることで少し高い音だけが鳴るようにしました。2つ目の引数で任意の秒数を指定したいところですが、秒数を指定した場合音が鳴り終わるまで次の命令に進まないようになっています。しかし2つ目の引数に0を指定した場合は音を鳴らし続けながら次の処理に進めるようになっているので一定時間後にSTPSNDという音を止める命令を出すという流れになっています。一定時間後に止めるために変数Vを利用しています。音を鳴らし始めたと同時にVを0にリセットします。状態更新処理が1回行われる度にVを1ずつ増やしていき、Vが6に到達するとSTPSND命令が出るという仕組みになっています。状態更新処理は1秒間に60回行われるので、音を鳴らしてから1/60 × 6=0.1秒後に音が止められることになります。

続いてゲームオーバーの処理を追加します。同様に追加していきましょう。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'
30 IF G GOTO 6000; REM 'game over'
(省略)
1250 V=0; REM 'for STPSND'
1260 G=0; REM 'game over flg'
1999 RETURN
(省略)
2030 GOSUB 5000; REM 'output'
2040 IF G RETURN
2990 VSYNC
(省略)
4250 IF N#P+Q O=1
4260 IF 47<=T+R G=1
4999 RETURN
(省略)
5070 NEXT I
5075 IF G RETURN
5080 IF O PLYSND RND(2000)+300,0
(省略)
6000 REM 'GAME OVER'
6010 PRINT ‘’
6020 PRINT ‘’
6030 PRINT '(空白6つ)GAME OVER'
6040 PRINT '(空白3つ)Enter = restart'
6050 PLYSND 400,200
6060 SLEEPMS 100
6070 PLYSND 400,400
6080 IF GETKEY(0)=13 GOTO 0
6090 GOTO 6080
```

プログラムの解説です。ゲームオーバーを判定するためのフラグGを用意します。Gはボールの下部が画面下端に触れるとフラグが立ちます。そしてOUTPUTの処理に差し掛かり、描画が終わったタイミングでRETURNを何度か呼び出し、一番初めのMAINの処理まで戻っていきます。そして6000～のゲームオーバーの処理に飛び、GAME OVERと表示します。PRINTという命令はその後に「'」で囲った文字を表示する命令で、表示後改行を行います。3行目からGAME OVERと表示するために2回PRINT "(←クォーテーション「"」ではなくアポストロフィ「'」2回)で改行しています。以降はキー入力を監視し、Enterが押されると0番目に飛び、ゲームをリセットします。RETURNでわざわざMAINまで戻ってきているのはGOSUBでサブルーチンに飛んでいる状態は2重までという制限があるからです(今後のアップデートで変わる可能性あり)。RETURNで戻らず再度0行目から実行しようとすると10 GOSUB 1000; REM 'initialize'という行でGOSUB too many nestedというGOSUBで飛びすぎというエラーが出てしまいます。そこでGOSUBで飛んでいない状態まで戻すためにRETURNでMAINまで戻ってきます。こうすることで問題なくゲームをリセットできるわけです。6060 SLEEPMS 100という命令がありますが、SLEEPMSというのは後ろに指定したミリ秒処理を停止するという命令です。ゲームオーバーのブッブーという音の2回音が鳴る間のディレイのために入れています。

同様にクリア画面も実装しましょう。

```
20 GOSUB 2000; REM 'loop'
30 IF G GOTO 6000; REM 'game over'
40 IF Z GOTO 7000; REM 'game clear'
(省略)
1260 G=0; REM 'game over flg'
1270 Z=0; REM 'game clear flg'
1999 RETURN
(省略)
2030 GOSUB 5000; REM 'output'
2040 IF G+Z RETURN
2990 VSYNC
(省略)
4260 IF 47<=T+R G=1
4270 Z=1
4280 FOR I=0 TO 15
4290 IF @(I) Z=0
4300 NEXT I
4999 RETURN
(省略)
5070 NEXT I
5075 IF G+Z RETURN
5080 IF O PLYSND RND(2000)+300,0
(省略)
7000 REM 'GAME CLEAR'
7010 PRINT ‘’
7020 PRINT ‘’
7030 PRINT '(空白6つ)GAME CLEAR'
7040 PRINT '(空白3つ)Enter = restart'
7050 PLYSND 392,250
7060 PLYSND 494,250
7070 PLYSND 587,250
7080 PLYSND 698,250
7090 PLYSND 1047,600
7100 IF GETKEY(0)=13 GOTO 0
7110 GOTO 7100
```

やっていることしては状態更新処理の最後で全ブロックのフラグをチェックし、全てのブロックが壊されていればゲームクリアの画面を出すという流れです。これまでの知識があれば読み解けるかと思います。

では次の機能追加に進みます。これがこの説明書における最後の機能追加です。今は棒のどの位置に当たっても同じように跳ね返りますがこれだと誰がやってもゲームオーバーにならない限り同じ跳ね返り回数でクリアすることになります。これではいささかつまらないので棒の当たる位置によって跳ね返り方を変えたいと思います。具体的には棒の右に当たると右に速度が加算され、左に当たると左に速度が加速され、真ん中に当たるとそのまま跳ね返るというような処理にします。これで早くブロックを消すためにバーの当て方も考えつつ跳ね返すというゲーム性が生まれるかと思います。ではラストスパート頑張りましょう。

```
(省略)
1060 H=3; REM 'bar height'
1070 S=44; REM 'ball x'
1080 T=18; REM 'ball y'
1090 R=2; REM 'ball r'
1100 P=1; REM 'ball vx'
1110 Q=1; REM 'ball vy'
1120 C=4; REM 'ball vx/?'
1130 D=0; REM 'for ball vx/? calc'
(省略)
4050 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)Q=-ABS(Q)
4051 @(31)=(X<=S)*(S<=X+W/3)*(0<P)
4052 @(31)=@(31)+((X+W*2/3<=S)*(S<=X+W)*(P<0))
4053 @(31)=@(31)*(C<=4)*(Y<=T+R)*(T+R<=Y+H)
4054 IF @(31) C=C*2
4055 @(31)=(X<=S)*(S<=X+W/3)*(P<0)
4056 @(31)=@(31)+((X+W*2/3<=S)*(S<=X+W)*(0<P))
4057 @(31)=@(31)*(2<=C)*(Y<=T+R)*(T+R<=Y+H)
4058 IF @(31) C=C/2
4059 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)T=Y-R-1
4060 D=D+1
4070 IF C<=D S=S+P
4080 IF C<=D D=0
4090 F=F+1
4100 IF E<=F T=T+Q
4110 IF E<=F F=0
4120 FOR I=0 TO 7
```

新しい命令等は無いですが、今回行った工夫として4170 IF @(I)*@(31)*@(30)*@(29)*@(28) Q=-Qのように配列を何個も用いるのではなく@(31)だけを使って順に条件を計算することで配列を使う数を減らしています。4051～4059では加速する場合と減速する場合に分けて、条件を判定します。無限に加速、減速しないように速度の分母であるCの値は1～8までに制限します。あとはボールを射出する位置と初期速度を微調整することでゲームがスタートしてすぐにゲームオーバーにならないように調整しました。

お疲れ様でした。これにてゲームの作り方講座は終了です。自分でオリジナルのゲームを作り始めても良いですし、このブロック崩しをさらに進化させていくというのも面白いと思います。今回紹介した命令以外にも様々な命令があり、今後も追加される予定です。付録として命令一覧を載せたのでバイブル的に使ってください。また、こういった機能が欲しいというものがあれば5chスレッドやX(Twitter)のDMで連絡いただければ、検討したのち追加するかもしれません(現時点でかなりの追加予定の機能があるので追加まで時間がかかるかもですが…)。

ここまでありがとうございました。良きNERD HPCライフを！

※この後のページにプログラム全体を載せておきます。

```
0 REM 'MAIN'
10 GOSUB 1000; REM 'initialize'
20 GOSUB 2000; REM 'loop'
30 IF G GOTO 6000; REM 'game over'
40 IF Z GOTO 7000; REM 'game clear'

1000 REM 'INITIALIZE'
1010 CLS
1020 GCLS
1030 X=50; REM 'bar left up x'
1040 Y=40; REM 'bar left up y'
1050 W=30; REM 'bar width'
1060 H=3; REM 'bar height'
1070 S=44; REM 'ball x'
1080 T=18; REM 'ball y'
1090 R=2; REM 'ball r'
1100 P=1; REM 'ball vx'
1110 Q=1; REM 'ball vy'
1120 C=4; REM 'ball vx/?'
1130 D=0; REM 'for ball vx/? calc'
1140 E=2; REM 'ball vy/?'
1150 F=0; REM 'for ball vy/? calc'
1160 J=2; REM 'block left up x'
1170 K=2; REM 'block left up y'
1180 L=12; REM 'block width'
1190 M=3; REM 'block height'
1200 FOR I=0 TO 15
1210 @(I)=1; REM 'block display flg'
1220 NEXT I
1230 N=0; REM 'reflect flg calc'
1240 O=0; REM 'reflect flg'
1250 V=0; REM 'for STPSND'
1260 G=0; REM 'game over flg'
1270 Z=0; REM 'game clear flg'
1999 RETURN
```

```
2000 REM 'LOOP'
2010 GOSUB 3000; REM 'input'
2020 GOSUB 4000; REM 'update'
2030 GOSUB 5000; REM 'output'
2040 IF G+Z RETURN
2990 VSYNC
2999 GOTO 2000

3000 REM 'INPUT'
3010 A=GETKEY(0)
3999 RETURN

4000 REM 'UPDATE'
4002 N=P+Q
4005 O=0
4007 V=V+1
4010 IF (A=249) * (X+W<127) X=X+1
4020 IF (A=250) * (0<X) X=X-1
4030 IF (S-R+P<0)+(127<S+R+P)P=-P
4040 IF (T-R+Q<0)+(47<T+R+Q)Q=-Q
4050 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)Q=-ABS(Q)
4051 @(31)=(X<=S)*(S<=X+W/3)*(0<P)
4052 @(31)=@(31)+((X+W*2/3<=S)*(S<=X+W)*(P<0))
4053 @(31)=@(31)*(C<=4)*(Y<=T+R)*(T+R<=Y+H)
4054 IF @(31) C=C*2
4055 @(31)=(X<=S)*(S<=X+W/3)*(P<0)
4056 @(31)=@(31)+((X+W*2/3<=S)*(S<=X+W)*(0<P))
4057 @(31)=@(31)*(2<=C)*(Y<=T+R)*(T+R<=Y+H)
4058 IF @(31) C=C/2
4059 IF (X<=S)*(S<=X+W)*(Y<=T+R)*(T+R<=Y+H)T=Y-R-1
4060 D=D+1
4070 IF C<=D S=S+P
4080 IF C<=D D=0
4090 F=F+1
4100 IF E<=F T=T+Q
4110 IF E<=F F=0
```

```
4120 FOR I=0 TO 7
4130 @(31)=J+I*16<=S
4140 @(30)=S<=J+I*16+L
4150 @(29)=K<=T
4160 @(28)=T<=K+M
4170 IF @(I)*@(31)*@(30)*@(29)*@(28) Q=-Q
4180 IF @(I)*@(31)*@(30)*@(29)*@(28) @(I)=0
4190 @(29)=K+7<=T
4200 @(28)=T<=K+7+M
4210 IF @(I+8)*@(31)*@(30)*@(29)*@(28) Q=-Q
4220 IF @(I+8)*@(31)*@(30)*@(29)*@(28) @(I+8)=0
4230 NEXT I
4240 IF N=P+Q O=0
4250 IF N#P+Q O=1
4260 IF 47<=T+R G=1
4270 Z=1
4280 FOR I=0 TO 15
4290 IF @(I) Z=0
4300 NEXT I
4999 RETURN

5000 REM 'OUTPUT'
5010 GCLS
5020 GRECT X,Y,X+W,Y+H,1,1
5030 GCIRCLE S,T,R,1,1
5040 FOR I=0 TO 7
5050 IF @(I) GRECT J+I*16,K,J+I*16+L,K+M,1,1
5060 IF @(I+8) GRECT J+I*16,K+7,J+I*16+L,K+7+M,1,1
5070 NEXT I
5075 IF G+Z RETURN
5080 IF O PLYSND RND(2000)+300,0
5090 IF O V=0
5100 IF V>=6 STPSND
5999 RETURN
```

```
6000 REM 'GAME OVER'
6010 PRINT ‘’
6020 PRINT ‘’
6030 PRINT '(空白6つ)GAME OVER'
6040 PRINT '(空白3つ)Enter = restart'
6050 PLYSND 400,200
6060 SLEEPMS 100
6070 PLYSND 400,400
6080 IF GETKEY(0)=13 GOTO 0
6090 GOTO 6080

7000 REM 'GAME CLEAR'
7010 PRINT ‘’
7020 PRINT ‘’
7030 PRINT '(空白6つ)GAME CLEAR'
7040 PRINT '(空白3つ)Enter = restart'
7050 PLYSND 392,250
7060 PLYSND 494,250
7070 PLYSND 587,250
7080 PLYSND 698,250
7090 PLYSND 1047,600
7100 IF GETKEY(0)=13 GOTO 0
7110 GOTO 7100
```

7. サンプルプログラム集

キーボードプログラム

NERD HPCでパソコンのキーボードを作ることができます。ここで使われているSNDKCD命令を使えばショートカットや定型文をワンボタンで繰り出すこともできます。

```
10 CLS
20 GCLS
30 GKOPT 0,0
40 FOR I=0 TO 5
50 A=0
60 @(I)=GETKEY(I)
70 IF (97<=@(I))*(@(I)<=122)A=@(I)-93; REM a~z
80 IF (49<=@(I))*(@(I)<=57)A=@(I)-19; REM 1~9
90 IF @(I)=48 A=39; REM 0
100 IF @(I)=45 A=45; REM _/-
110 IF @(I)=61 A=46; REM +/=
120 IF @(I)=96 A=53; REM ~/`
130 IF @(I)=91 A=47; REM {/[
140 IF @(I)=93 A=48; REM {/[
150 IF @(I)=92 A=49; REM |/\
160 IF @(I)=59 A=51; REM :/;
170 IF @(I)=39 A=52; REM "/'
180 IF @(I)=44 A=54; REM </,
190 IF @(I)=46 A=55; REM >/,
200 IF @(I)=47 A=56; REM ?//
```

```
210 IF @(I)=249 A=79; REM right
220 IF @(I)=250 A=80; REM left
230 IF @(I)=251 A=81; REM down
240 IF @(I)=252 A=82; REM up
250 IF @(I)=14 A=225; REM L-Shift
260 IF @(I)=15 A=229; REM R-Shift
270 IF @(I)=13 A=40; REM Enter
280 IF @(I)=9 A=43; REM Tab
290 IF @(I)=27 A=41; REM Esc
300 IF @(I)=254 A=57; REM Caps
310 IF @(I)=32 A=44; REM Space
320 IF @(I)=247 A=224; REM L-Ctrl
330 IF @(I)=248 A=230; REM R-Alt
340 IF @(I)=8 A=42; REM BS
350 IF @(I)=246 A=73; REM Ins
360 IF @(I)=245 A=227; REM NERD
370 @(I)=A
380 NEXT I
390 SNDKCD @(0),@(1),@(2),@(3),@(4),@(5)
400 GOTO 40
```

ピアノプログラム

NERD HPCでピアノを作ってみるのはどうでしょうか？数字の1～8でドレミファソラシドを鳴らせます。

```
10 CLS
20 GCLS
30 A=GETKEY(0)
40 IF A=49 PLYSND 262,0
50 IF A=50 PLYSND 294,0
60 IF A=51 PLYSND 330,0
70 IF A=52 PLYSND 349,0
80 IF A=53 PLYSND 392,0
90 IF A=54 PLYSND 440,0
100 IF A=55 PLYSND 494,0
110 IF A=56 PLYSND 523,0
120 IF A=0 STPSND
130 GOTO 30
```

ルーレットプログラム

こちらはNERD HPCの抽選販売を行った際に使ったプログラムです。全体人数と当選人数を入力することで抽選を行います。

```
10 CLS
20 GCLS
30 INPUT 'ZENTAI='A
40 INPUT 'TOUSEN='B
50 FOR I=0 TO 50
60 PLYSND 1000,20
70 SLEEPMS 20
80 NEXT I
90 FOR I=0 TO 25
100 PLYSND 1000,40
110 SLEEPMS 40
120 NEXT I
130 FOR I=0 TO 8
140 PLYSND 1000,100
150 SLEEPMS 100
160 NEXT I
170 FOR I=0 TO 3
180 PLYSND 1000,200
190 SLEEPMS 200
200 NEXT I
210 SLEEPMS 700
220 PLYSND 1500,1200
221 FOR I=0 TO B-1
222 @(I)=0
229 NEXT I
230 FOR I=0 TO B-1
240 @(I)=RND(A)
250 FOR J=0 TO I
```

```
260 IF J=I GOTO 310
270 C=0
280 IF @(I)=@(J) C=1
290 IF C=1 @(I)=RND(A)
300 IF C=1 J=-1
310 NEXT J
320 NEXT I
330 FOR I=0 TO B-1
340 PRINT @(I),',',
350 NEXT I
```

キーコード出力プログラム

GETKEYで出力されるキーコードを事前に確認したい時に便利です。

```
10 GCLS
20 CLS
30 PRINT GETKEY(0)
40 VSYNC
50 GOTO 20
```

IO端子入力プログラム

IO1~4の状態を出力します。GNDが入力されると1→0に変化します。外部コントローラーみたいなのを作っても面白いかもしれません。

```
10 GCLS
20 CLS
30 IOSD 1,1
40 IOSD 2,1
50 IOSD 3,1
60 IOSD 4,1
70 IOPU 1
80 IOPU 2
90 IOPU 3
100 IOPU 4
110 CLS
120 PRINT IOGET(1),IOGET(2),IOGET(3),IOGET(4)
130 SLEEPMS 100
140 GOTO 110
```

IO端子出力プログラム

IO1~4の順に3.3vを出力します。LEDを接続するときれいです。LEDを接続する際は抵抗を忘れないようにしましょう(LEDが燃えます)。

```
10 GCLS
20 CLS
30 IOSD 1,0
40 IOSD 2,0
50 IOSD 3,0
60 IOSD 4,0
70 IOPUT 1,1
80 IOPUT 2,1
90 IOPUT 3,1
100 IOPUT 4,1
110 IOPUT 4,1
120 IOPUT 1,0
130 SLEEPMS 300
140 IOPUT 1,1
150 IOPUT 2,0
160 SLEEPMS 300
170 IOPUT 2,1
180 IOPUT 3,0
190 SLEEPMS 300
200 IOPUT 3,1
210 IOPUT 4,0
220 SLEEPMS 300
230 GOTO 110
```

ランダムドットプログラム

画面上にランダムに点を打ちます。ライフゲーム作りたくなりますね。

```
10 CLS
20 GCLS
30 X=RND(128)
40 Y=RND(48)
50 GPSET X,Y,1
60 SLEEPMS 1
70 GOTO 30
```

8. 付録(命令一覧)

(基本命令)
LIST [行番号|ALL]
NEW
RUN
PRINT [文字列|#桁数|式][,文字列|#桁数|式...]
INPUT [文字列][変数|配列][,[文字列][変数|配列]...]
[LET] 変数|配列=式
REM [コメント]
IF 式 [文]
GOTO 式
GOSUB 式、RETURN
FOR 変数=開始値TO終了値[STEP増分]、NEXT 変数
STOP
SIZE()
ABS(式)
RND(式)
CLS
SLEEPMS 定数
SLEEPUS 定数
SAVE "ファイル名"
LOAD "ファイル名"
DELETE “ファイル名”
SETF 文字コード,フォントデータ
RESETF
CHR(文字コード, …)
LOCATE X座標,Y座標
SCR(X座標,Y座標)

(記号類)
+ - * / %
= #
>= > <= <
A~Z
@(0)~@(255)
"文字列"、'文字列'
,
:
;
`
$
& | ^ ~ << >>

(グラフィック命令)
GCLS
GPSET X座標,Y座標,色
GLINE X0座標,Y0座標,X1座標,Y1座標,色
GCIRCLE X座標,Y座標,半径,色,塗りつぶし
GRECT X0座標,Y0座標,X1座標,Y1座標,色,塗りつぶし
GTRI X0座標,Y0座標,X1座標,Y1座標,X2座標,Y2座標,色,塗りつぶし
GPUTC X座標,Y座標,文字コード,色,透過
GPGET X座標,Y座標
VSYNC

(キーボード命令)
GETKEY(index)
GKOPT Shift有効,Esc有効
SNDKCD HIDキーコード1,HIDキーコード2,HIDキーコード3,HIDキーコード4,HIDキーコード5,HIDキーコード6,

(スピーカー命令)
PLYSND 周波数,ミリ秒
STPSND

(汎用IO命令)
IOSD IO番号,入出力
IOPU IO番号
IOPD IO番号
IODP IO番号
IOPUT IO番号,出力
IOGET(IO番号)
IOUS 数字
IOUR()
IOIIM
IOIIS
IOIS 数字
IOIR()
IRPUT 出力
IRGET()

各命令の解説です。具体的な使用例はHPの方に記載しています。

(基本命令)

- LIST [行番号|ALL]
  - 指定の行番号からプログラムを表示する。
    矢印の上下で表示行移動。LISTのみで最初の行から表示。LIST ALLで全行一括で表示。EscでLIST命令モードを抜ける。
- NEW
  - 現在入力されているプログラムを削除する。
- RUN
  - 行番号の順にプログラムを実行する。
- PRINT [文字列|#桁数|式][,文字列|#桁数|式…]
  - 文字列・式の値を表示する。
    桁数を指定することで右詰めで表示可能。末尾が [,]だと改行しない。
  - 使用例
    >PRINT 'HELLO, WORLD'
    >PRINT 'Say "ok" to Nakata.'
    >LET A=3;LET B=14
    >PRINT "PAT=",A,"."B
    >PRINT #2,123,#5,456,7890
    >PRINT (1+2)/3-4*2
    >PRINT 0<1=(A#B)
- INPUT [文字列][変数|配列][,[文字列][変数|配列]…]
  - 端末上で変数に定数を入力する。
    入力は数字と符号と改行のキーだけを受け付ける。変数または配列の前に文字列を記述するとそれをプロンプトとして表示する。
  - 使用例
    >INPUT A,B,@0
    >INPUT "YEAR="A
- [LET] 変数|配列=式
  - 変数または配列に式の値を代入する。
    LETは省略可能。
- REM [コメント]
  - 行末までの記述を無効にする。
    実行時間はかかるので少し間を置くために使うことがある。
- IF 式 [文]
  - 式の値が0以外のとき後ろの文を実行する。
    式以外にも定数、変数、配列、関数、計算式も記述可能。

- GOTO 式
  - 式が表す行番号へ分岐する。
    1~32767が指定可能。
- GOSUB 式、RETURN
  - 式が表す行番号のサブルーチンを呼び出す。RETURNはサブルーチンを終了して呼び出し元へ復帰する。
    呼び出したサブルーチンから別のサブルーチンを呼び出す回数には仕様で定める上限がある。
- FOR 変数=開始値TO終了値[STEP増分]、NEXT 変数
  - 変数をカウンタにして繰り返す。
    STEP増分を省略すると1ずつ変数を増やす。変数は -32768~32767の範囲で必要がある(終了の時点で超えてもいけない)。FORのネスト回数は仕様で定める上限がある。
- STOP
  - プログラムの実行を終了する。
- SIZE()
  - プログラムの保存領域の残りをバイト数で返す。
    保存領域の大きさはファームウェアのカスタマイズで変更可。
- ABS(式)
  - 式の値の絶対値を返す。
- RND(式)
  - 1~式の値の範囲でランダムな値を返す。
    式の値は1~32767の範囲でなければならない。
- CLS
  - テキスト画面を消去する。
- SLEEPMS 定数
  - 指定したミリ秒処理を待つ。
- SLEEPUS 定数
  - 指定したマイクロ秒処理を待つ。
- SAVE "ファイル名"
  - 現在入力されているプログラムを SDカードに保存する。
    txtファイル形式にするとPCで読み書きできるので便利。
- LOAD "ファイル名"
  - SDカードに保存されているプログラムを読み出す。

- DELETE “ファイル名”
  - 指定したファイルをSDカードから削除する。
- SETF 文字コード(0~255),フォントデータ (“0000000000~FFFFFFFFFF”)
  - 指定した文字コードのフォントデータを変更する。フォントデータは横 5x縦8のドットデータで構成されている。左列のドットデータから 1byteずつフォントデータを記述していく(縦8ドットの順は上が0bit目、下が7bit目となっている)。
- RESETF
  - 全てのフォントデータをリセットする。
- CHR(文字コード, …)
  - 文字コードから文字に変換する。カタカナ等は直接キーボードから入力できないのでCHR命令を使ってPRINT文の中に組み込む。
- LOCATE X座標(0~20),Y座標(0~5)
  - PRINT文のカーソル位置を変更する。
- SCR(X座標(0~20),Y座標(0~5))
  - 指定したテキスト画面の座標にある文字の文字コードを取得する。

(記号類)

- + - * / %
  - 加減乗除余り。
- = #
  - 一致、不一致。
- >= > <= <
  - 大小関係。
- A~Z
  - 変数。-32767~32767まで代入可能。
- @(0)~@(255)
  - 配列。
- "文字列"、'文字列'
  - 文字列
- ,
  - 引数の区切り
- ;
  - 文の区切り
- `
  - 2進数表記。`0~`1111111111111111まで表記可能。
- $
  - 16進数表記。$0～$FFFFまで表記可能。
- & | ^ ~ << >>
  - ビット演算。AND, OR, XOR, ビット反転, 左シフト, 右シフト。

(グラフィック命令)

- GCLS
  - グラフィック画面を消去する。
- GPSET X座標,Y座標,色(0=白|1=黒)
  - X座標、Y座標にドットを打つ
- GLINE X0座標,Y0座標,X1座標,Y1座標,色((0=白|1=黒))
  - X0座標、Y0座標からX1座標、Y1座標に線を描画する。
- GCIRCLE X座標,Y座標,半径,色(0=白|1=黒),塗りつぶし(0=なし|1=あり)
  - 指定のX座標、Y座標に円を描画する。
- GRECT X0座標,Y0座標,X1座標,Y1座標,色(0=白|1=黒),塗りつぶし(0=なし|1=あり)
  - 画面に四角を描画する
- GTRI X0座標,Y0座標,X1座標,Y1座標,X2座標,Y2座標,色(0=白|1=黒),塗りつぶし(0=なし|1=あり)
  - 指定した3点を結ぶ三角形を描画する。
- GPUTC X座標(0~47),Y座標(0~127),文字コード(0~255),色(0=反転|1=そのまま),透過(0=重ねない|1=重ねる)
  - 文字コードはANK文字コード準拠。フォントデータは横 5x縦8のデータとなっており、透過を無効にすると5x8のデータでその場所を上書きする。透過を有効にすると背景と重ねて表示する。
- GPGET X座標,Y座標
  - 指定した座標のドットの状態を取得する。白 =0、黒=1
- VSYNC
  - 画面の更新まで待機する。
    画面は60fpsで更新されている。

(キーボード命令)

- GETKEY(index:0～7)
  - 命令実行時に押下しているキーコードを取得する。
    8個まで同時押しに対応しており、押下しているキーコードを保存している配列よりindexで指定した番号のキーコードを取得する。キーマトリクス回路上同時押しすると意図しないキーコードに化けるものもあるので注意 (修正予定)。
- GKOPT Shift有効(0=無効|1=有効),Esc有効(0=無効|1=有効)
  - GETKEYで得られるキーコードにShiftを押した状態を加味するか設定する。プログラム動作中にEscによるプログラムの中断を有効にするか設定する。
    Shiftが有効だと、Shift+aを押した際にGETKEYでAのキーコードが得られる。
    Shiftが無効だと、Shift+aを押した際にGETKEYでaのキーコードが得られる。
    Escを無効にすることでEscを使うプログラムでプログラムが中断しないようにできる。デフォルトだとShiftもEscも有効。
- SNDKCD HIDキーコード1,HIDキーコード2,HIDキーコード3,HIDキーコード4,HIDキーコード5,HIDキーコード6,
  - パソコンに指定のHIDキーコードを送る。
    同時に6つまでHIDキーコードを送れる。

(スピーカー命令)

- PLYSND 周波数,ミリ秒
  - 指定の周波数の音を出す。
    ミリ秒を0にすると音を鳴らし続ける。
- STPSND
  - 現在鳴っている音を止める。
    PLYSNDでミリ秒を0に指定し、鳴らし続けた音を止めるのに有効

(汎用IO命令)

- IOSD IO番号(1~4),入出力(0=出力|1=入力)
  - IO1~4を入力または出力に設定する。
    IOPUT, IOGETを使用する前に一度設定する必要あり。
- IOPU IO番号(1~4)
  - IO1~4に対し内蔵プルアップ抵抗を有効にする。
    プルアップ抵抗を有効にすることで Hi,Loどちらも入力されていないときの入力が安定する。
- IOPD IO番号(1~4)
  - IO1~4に対し内蔵プルダウン抵抗を有効にする。
    プルダウン抵抗を有効にすることで Hi,Loどちらも入力されていないときの入力が安定する。
- IODP IO番号(1~4)
  - IO1~4に対し内蔵プルアップ・プルダウン抵抗を無効にする。
- IOPUT IO番号(1~4),出力(0=0v|1=3.3v)
  - IO1~4の電圧を変更する。
- IOGET(IO番号)
  - IO1~4に0vが入力されているか 3.3vが入力されているか判定する。
    0=0v,1=3.3v

- IOUS 数字(0~255)
  - UART通信を用いてデータを送信する。
    NERD HPCのGNDを相手側のGND、TXを相手側のRX、RXを相手側のTXに接続すると通信ができる。
- IOUR()
  - UART通信を用いてデータを受信する。
    NERD HPCのGNDを相手側のGND、TXを相手側のRX、RXを相手側のTXに接続すると通信ができる。
- IOIIM
  - I2C通信のマスターに設定する。
    NERD HPCのGNDを相手側のGND、SDAを相手側のSDA、SCLを相手側のSCLに接続し、相手側をIOIISでスレーブに設定するとデータを送れる状態になる。起動時はマスターに設定されている。
- IOIIS
  - I2C通信のスレーブに設定する。
    NERD HPCのGNDを相手側のGND、SDAを相手側のSDA、SCLを相手側のSCLに接続し、相手側をIOIIMでマスターに設定するとデータを受信できる状態になる。起動時はマスターに設定されている。
- IOIS 数字(0～255)
  - I2C通信を用いてデータを送信する。
    NERD HPCのGNDを相手側のGND、SDAを相手側のSDA、SCLを相手側のSCLに接続し、相手側をIOIISでスレーブに設定するとデータを送信できる。
- IOIR()
  - I2C通信を用いてデータを受信する。
    NERD HPCのGNDを相手側のGND、SDAを相手側のSDA、SCLを相手側のSCLに接続し、相手側をIOIIMでマスターに設定するとデータを受信できる。
    v1.7.1にてIOIR→IOIR()に修正
- IRPUT 出力(0=出力しない|1=出力する)
  - 本体から38kHzのPWMで赤外線LEDを出力する
- IRGET()
  - 赤外線受信モジュールの状態を取得する。0=赤外線受信なし|赤外線受信中。本体の赤外線受信モジュールは 38kHzでオン・オフする赤外線 LEDに反応する。また、赤外線受信モジュールの特性として変化のない 38kHzの赤外線LED出力は一定時間経過すると反応しなくなる。

9. 付録(NERD HPC裏話)

この度はNERD HPCに興味を持っていただきありがとうございます。イベントで販売するにあたってNERD HPCの説明書が無いというのは非常にマズイと思い、急きょ作成しました。NERD HPCは皆さんの支えがあり何とか存続できている節があり、他の方がプログラムを公開して盛り上げていただいたり、解説サイトを作成していただいたり、自身に足りないところを補っていただいているような状況です。ついにはセガ和辞典出版局 WATさんが「こんばんは NERD HPC "超"初級編」という解説本も出してくださいました。こちらはこの無機質な説明書と違って画像も分かりやすく付け加えられており紹介されているプログラムも面白いです。要点が抑えられおり素早くNERD HPCの使い方を習得できますのでぜひこちらも見ていただきたいと思います。こうやって自分の作品の同人誌が生まれるって非常に嬉しいものですね。皆でNERD HPCを発展させていっているというのは「NERD HPCとは」の項目で述べたNERD HPCのあるべき姿だと思い非常に感慨深いです(自身が怠惰であることを除いて)。

NERD HPCには「安くて自由で使いやすい」というテーマとは別に様々な思いが込められています。一つ挙げるとすると「長く使える」というものがあります。これはハードウェア的に強固という意味もありますが、それとは別に、引き出しの奥から存在を忘れていたNERD HPCが数年ぶりに出てきたときに、現役でコミュニティに参加できるという意味も含まれます。どういったことかと言うと、このNERD HPCのハードウェア構成は部品の廃盤といったことがない限り極力仕様を変えずに作り続け、昔買った人が最近買った人と問題なく交流できるようにしたいと考えています。そのためにハードウェア的な工夫の一例として、劣化するバッテリーではなく電池駆動にし、WiFi,Bluetoothも乗せようかと検討しましたが無線関係は規格が古くなることを考慮し、汎用入出力端子を使って各々が拡張するという形を取りました。

ですのでNERD HPCに足りないと思う部分はどんどん自身の手で拡張・改良していっていただければと思います。イメージとしてはVimのような感じでしょうか。足りないと思う部分はプラグイン(NERD HPCにおいては汎用入出力を使った拡張基板)を作成し、それでも解決しない部分はNeovimのようにNERD HPC自体を作り変える試みをしても面白いかと思います。また、そういったものはどんどん販売していただいて大丈夫です。皆さんの手でNERD HPCの世界を拡張していきましょう。

そして、このNERD HPCは自身の様々な経験に影響を受けています。先程のVimもそうですが、一番は学研の大人の科学マガジン「4ビットマイコン」です。これは7個のLEDと1つの7セグとスピーカーを出力とし、プログラムをアセンブリで考え、16進数で入力するというなんとも玄人向けのハードウェアなのですが、小学3年生のときに書店で見つけ、なんか面白そうという理由だけで親におねだりしてそのときは買ってもらえなかったのですが、後日学校から帰ると買ってプレゼントしてくれました。普段おねだりして買ってもらえることはなかったのですが、なぜかこれは買ってもらえました。振り返ると教育に良さそうという理由だったのかもしれませんが、当時の定価を調べてみると2500円程度。安かったからなのかもしれません笑。そしてこれが自身のプログラミングの始まりでした。買ってもらったはいいものの使い方を読むとちんぷんかんぷん。しかしながら分からないなりにも試行錯誤し、プログラムが動いたときには感動しました。このように「4ビットマイコン」から受けた影響は大きく、テーマの「安い」と「使いやすい」はこの経験から来ています。安いことで気軽に手にとってもらえる&アセンブリは難しすぎたのでBASICにしようということです。そしてこのNERD HPCも誰かに影響を与えられるような製品になれば冥利に尽きます。

そしてNERD HPCの開発の動機についてですが、自分が長く熱中できるものが欲しかったというのが動機です。自分が飽き性だということもあり、色々なことをやってみるのですが一時的に熱中することはあっても中々長続きしないということに悩んでいました。そこで自分が心から面白いと思えるものを作ろうと思い立ったのがNERD HPCの始まりでした。まずテーマとして設定したのが「安いゲーム機を作ろう」でした。今ゲーム機自体が非常に高価なものになっていて気軽に買えなくなっている現状があると思い、数千円で買えるようなゲーム機を設計しはじめました(Youtubeに一個だけ上がっている動画がそれです笑)。安くするためにボタンは一つ、画面もモノクロ液晶を採用しました。こういった製品によくあるのがPCでプログラミングを行い、本体に書き込んでゲームを楽しむというものですが、この方法だとPCが必須となることからPCを持っていない人からするとむしろ普通にゲーム機を買うより高くなるというのが問題だと考えました。そこでゲーム機単体でプログラミングができるようにしようという発想に至りました。そしてゲーム機でプログラミングを行うということで思い浮かんだのがプチコンシリーズでした。プチコンとはDSやwii Uやswitch向けに配信されている、BASICが実行できるソフトです。この存在を思い出し、BASICを乗せようということになりました。この時点で、BASICが実行できちゃう、ちょっとオタク向けの匂いがするGAME BOYっぽいゲーム機ということで「NERD BOY」と名付けました。しかしながらNERD BOYはボタンが一つしか無いためどのようにしてプログラムを入力するかということについて考える必要がありました。そこで設計したのがNERD BOYに接続するキーボード「NERD KEY」でした。これをNERD BOYに接続することでプログラムを打ち込めるようにしようと思いましたが、基板を発注する直前に、一体型にしてはどうだろうという考えがよぎりました。一体型にすることによってNERD BOYとNERD KEYを別々に用意するよりコストが抑えられ、よりコンパクトになるのではないかと考えました。試しに設計してみると非常に上手くまとまり、これをNERD DEV(開発を意味するdevelopmentから)と名付けました。

NERD DEVの開発を進めていくうちにこれは開発機というよりパソコンとして開発したほうが面白いと思い、NERD DEVはNERD HPC(Hobby Personal Computer)と改名し、一旦NERD BOYの開発は中断してこちらに注力する流れとなりました。ここまでの流れが開発スタートから約半年間くらいの話になるのですが、途中他の用事との兼ね合いで手を付けられないことがあっても熱量は冷めることなくC105コミケにて世に出すことができました。当時、コミケの出展自体初めてで、コミケに行ったのも出展の半年前のコミケが初という右も左も分からない初心者だったのですが、ブースが隣ということで知り合ったゲームインパクトさんの縁がありNERD HPCは勢い付いてくれたといっても過言ではありません。そして今、2024年5月の開発スタートからおよそ一年になるのですがまだまだこのNERD HPCは自分を楽しませてくれそうです。そして皆さんも楽しませることができるように日々発展させていきたい所存です。

10. 各種リンク

- 公式サイト: 中田商店
  - https://nakatashoten-hp.web.app/product/nerdhpc/index.html
- 5chスレッド: 新型ゲーム機開発してたはずが…
  - https://itest.5ch.net/mevius/test/read.cgi/gamedev/1727809542
- X(Twitter): 中田商店(@nakata_shoten)
  - https://x.com/nakata_shoten
- Youtube: 中田商店(@nakata_shoten)
  - https://www.youtube.com/@nakata_shoten

公式サイト

5chスレッド

X(Twitter)

YouTube

- 皆さんの作例: X(Twitter)
  - https://x.com/nakata_shoten/status/1942163772115734709
- GitHub
  - https://github.com/TTtensan/NERD-HPC
- 命令一覧
  - https://nakatashoten-hp.web.app/product/nerdhpc/docs/instruction/

作例 X(Twitter)

GitHub

命令一覧