RPGOS
NPC 独自アセンブリ
プログラミング

［著］大神祐真

第十二回技術書同人誌博覧会 新刊
2025 年 10 月 26 日　 ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

はじめに

本書を手に取っていただき、ありがとうございます。本書では、「RPG のような世界で動く OS」として制作している「RPGOS」の NPC（Non Player Character）を、独自な言語でプログラミングできるようにした「NPC プログラミング環境」を紹介し、実際に簡単なプログラミングを行ってみます。

まだできることは限られますが、「NPC が動いて機能が実現される」という世界観を楽しんでいただければ幸いです。

表紙・裏表紙について

今回、表紙・裏表紙をくろあんず様に作っていただきました！ くろあんず様はグラフィックデザインやタイポグラフィでご活躍されている方で、ビジュアルノベル制作サークル「D10RAMA（ジオラマ）」でデザインを担当されています。

- くろあんず様：
  - X：`https://x.com/B1ackApric0t`

コミックマーケット 106 に当サークルが出展した際、通路をはさんで正面で出展されていて、当サークルの活動にご興味を持っていただいたきお声がけいただいたご縁で、この度、表紙・裏表紙をデザインしていただくことになりました！[*1]

D10RAMA 様では、2025 年 9 月現在、ビジュアルノベルゲームを 4 作品公開されており、中にはゲームのコンテストやイベントで受賞されている作品もあるスゴいサークル様です！ そしてなんと、そのようなビジュアルノベルゲームを無料で公開されています！[*2]

- D10RAMA 様：
  - ウェブサイト：`https://www.d10rama.com/`

[*1] 当サークルとしては表紙・裏表紙のデザインを他の方にお願いするのは初で、これは記念すべき 1 冊になりました！

[*2] 私はまだ、その内の 1 作品（「みえない二等星」）しかプレイできていないのですが、心温まる、そして人生を応援してくれるような素晴らしい作品でした！

サンプルコードについて

本書で紹介するサンプルコードは、以下の GitHub リポジトリで公開しています。サンプルコードの他に独自な命令[*3]のリファレンス資料も参照できます。

- https://github.com/cupnes/rpgos_npc_programming_samples

本書を読み進める際に、実際のコードを参照したり、手元でコピー&ペーストして実行したりする際にご活用ください。

本書の更新情報などについて

本書を含め、当サークルの同人誌・同人作品の情報は下記の筆者ウェブページにまとめています。

- https://yuma.ohgami.jp/

本書の内容について訂正や更新があった場合もこちらのページに記載します。何かおかしな点があった場合などは、まずこちらのページをご覧ください。

[*3] 「アクション」と呼称しています。詳しくは後述します。

目次

第 1 章　NPC プログラミング環境の使い方と初めてのプログラム

この章では、「RPGOS」と「NPC プログラミング環境」について説明し、その使い方を紹介します。そして、NPC プログラミング環境の上で簡単なプログラムを作って実行してみます。

1.1　RPGOS とは

RPGOS は、「RPG のような世界で動く OS」というコンセプトで制作している OS です。一見、黒背景に白文字の、CUI のみの OS のように見えますが、その裏では RPG のような世界があり、そこで NPC(Non Player Character) が動くことで OS の機能が成り立っている、というものです。

これは、PLiCy というゲームサイトでブラウザで動くものを公開しています。公開場所は以下の通りです。

- `https://plicy.net/GamePlay/202689`

なお、URL でのアクセスだけでなく、PLiCy のサイト内で「RPGOS」で検索しても出てくると思います。

上記のページを開くと、「大画面でゲームを遊ぶ」というボタンがあります。これをクリックすると別ウィンドウで起動します。

RPGOS について詳しくは以下の同人誌で解説しています。ご興味あればご覧ください。

- 同人誌「RPG のような世界で動く OS の本」
  - 筆者ウェブサイトよりどうぞ（`https://yuma.ohgami.jp/`）

7

1.2　NPC プログラミング環境とは

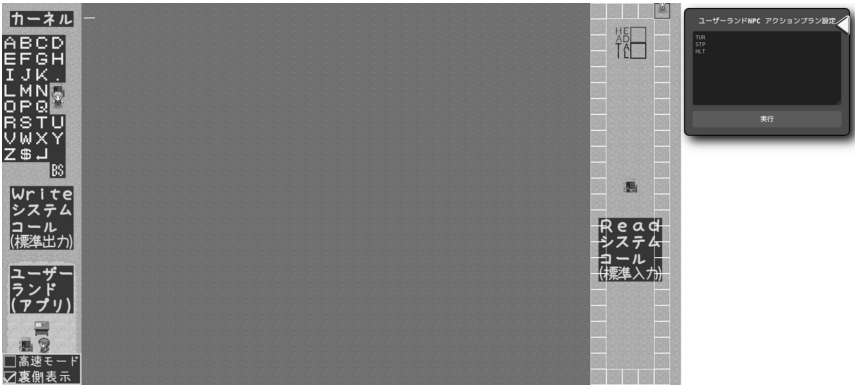
RPGOS 上の NPC は、現実のコンピュータにおける CPU に相当する存在だと考えています。そこで、現実の CPU がマシン語の命令を一つずつ解釈して実行していくように、「アクション」と呼ぶこの世界独自の命令を NPC が解釈し実行するように開発しています。できることはまだ限定的ですが、このコンセプトを体験してもらうくらいのことはできるようになりましたので、RPGOS 自体の画面のとなりにアクションを用いたプログラミングが行えるフォームを用意して NPC のプログラミングを行えるようにしたものが「NPC プログラミング環境」です。

こちらも PLiCy で公開しています。公開場所は以下の通りです。

- `https://plicy.net/GamePlay/209620`

なお、URL でのアクセスだけでなく、PLiCy のサイト内で「NPC プログラミング環境」などで検索しても出てくるかと思います。

こちらも同様に「大画面でゲームを遊ぶ」をクリックすると別ウィンドウで起動します。開いたウィンドウの左側に RPGOS の画面が、右側にテキスト入力フォームが表示されています（図 1.1）。



▲図 1.1: NPC プログラミング環境の実行画面

右側のフォームに、以降で説明するアセンブリ言語に似た独自な言語でプログラムを記述し、「実行」ボタンをクリックすると、RPGOS の画面内左下にある「ユーザーランド (アプリ)」と書かれた領域に居る NPC がそのプログラムを解釈して動きます（図 1.2）。

なお、RPGOS 部分の左下に「高速モード」と「裏側表示」という 2 つのチェック

▲図 1.2: ユーザーランドとフォーム部分の拡大図

ボックスがあります。「高速モード」は、NPC の動きを高速にするモードです。具体的には、1 つのアクションを 1 フレーム[*1]で実行完了するようになります。アクションは NPC が何らかの行動を行うものになっていて、通常時はその行動のアニメーション分のフレーム数の時間がかかりますが、高速モードではそれを 1 フレームで完了するという訳です。そして、「裏側表示」は、これをオフにすると、NPC 等の RPG 世界が黒で覆われ、画面中央のテキスト領域しか見えなくなります。これについてはまた後ほど紹介します。なお、各チェックボックスのオン/オフを切り替える際は、チェックボックスの四角の中をクリックしてください。

画面サイズは 1700x768px 以上を推奨

RPGOS 部分が 1366x768px あり、フォーム部分が 334x257px（影部分含むともう少し）あります。そのため、1700x768px 以上の画面サイズがないと、フォームが RPGOS 部分に被ってしまったり、RPGOS 部分が表示しきれなかったりします。

[*1] NPC プログラミング環境は 30FPS で動作します。

1.3　いくつか簡単なプログラムを試す

♣ 一歩進んで停止するプログラム

それでは、簡単なプログラムとして「一歩進んで停止する」プログラムを書いて動かしてみましょう。使用するアクション（命令）は 2 つだけで、プログラムの全体としてはリスト 1.1 の通りです。なお、このプログラムはサンプルリポジトリの ch01 フォルダに prog_step_halt.act.txt というファイル名で置いています。

▼リスト 1.1: 一歩進んで停止するプログラム

```
STP
HLT
```

それぞれのアクションについては表 1.1 と表 1.2 の通りです。これら 2 つのアクションを実行することで、NPC は「一歩進んで停止」という動作を行います。なお、アクションは本書で紹介する他にも使用可能なものがあります。使用可能な全てのアクションについては、サンプルリポジトリの docs フォルダ内の action_set_reference.md を参照してください。

▼表 1.1: STP（STeP）アクション

書式	`STP` (引数[*2]無し)
動作	NPC が現在向いている方向に一歩前進する。

▼表 1.2: HLT（HaLT）アクション

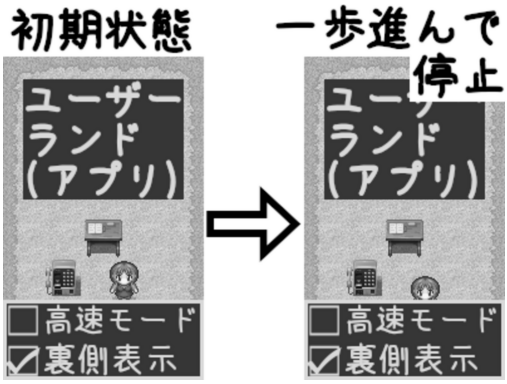
書式	`HLT` (引数無し)
動作	NPC を停止させる。

それではこれを実行してみましょう。NPC プログラミング環境の右側のフォームへリスト 1.1 を入力し、「実行」ボタンをクリックしてください。

すると、「ユーザーランド (アプリ)」の領域の NPC が一歩進んで停止します（図 1.3）[*3]。なお、一歩進むだけなので、実行するとすぐに終わります。左下の NPC に注目しつつ実行してみてください。

[*2] この独自な言語はアセンブリライクに作っているので「オペランド」と呼ぶ方が適切かもしれませんが、「引数」と呼ぶ方が多くの方に伝わりやすいかと思うので、本書ではそのように呼称します。

[*3] 前進することで頭から下がチェックボックスで隠れてしまいますが・・。



▲図 1.3:「一歩進んで停止」プログラムの実行結果

ちなみに、移動に関しては、「指定された座標へ移動する」というアクションもあります（表 1.3）。

▼表 1.3: MVC（MoVe to Coordinate）アクション

書式	`MVC <X 座標>, <Y 座標>`
動作	NPC を指定座標に移動させる。

♣ 右旋回してから一歩進んで停止するプログラム

次に、NPC が自身から見て右に向きを変えてから一歩進んで停止するプログラムを作成してみましょう。先程のプログラムにアクションを一つ追加するだけです（リスト 1.2）。このプログラムもサンプルリポジトリの ch01 フォルダに prog_turn_step.act.txt というファイル名で置いています。

▼リスト 1.2: 右旋回してから一歩進んで停止するプログラム

```
TUR  ←追加
STP
HLT
```

追加したアクションについては表 1.4 の通りです。これにより、NPC は「右旋回してから一歩進んで停止」という動作を行います。

リスト 1.2 を実行すると、NPC が 90 度右旋回してから一歩進むため、図 1.4 のように電話の正面で電話の方を向いた状態になります。

▼表 1.4: TUR（TUrn Right）アクション

書式	TUR（引数無し）
動作	NPC が右に 90 度旋回する。

▲図 1.4: 「右旋回してから一歩進んで停止」プログラムの実行結果

▼表 1.5: TUL（TUrn Left）アクション

書式	TUL（引数無し）
動作	NPC が左に 90 度旋回する。

ちなみに、逆に左旋回するアクションもあります（表 1.5）。

♣ 四角形を描くように移動する

また新たなアクションを用いて、四角形を描くように移動するようにしてみましょう。プログラムとしてはリスト 1.3 の通りです。このプログラムもサンプルリポジトリの ch01 フォルダに prog_square.act.txt というファイル名で置いています。

▼リスト 1.3: 四角形を描くように移動するプログラム

```
TUR
STP
~~HLT~~      ←削除
JMP -2   ←追加
```

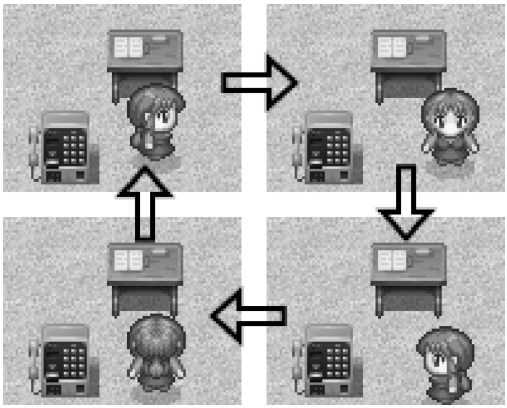
追加したアクションについては表 1.6 の通りです。これにより、NPC は「右旋回してから一歩進んで停止」という動作をしてから、2 つ前のアクションに戻るため、その動作を繰り返します。

リスト 1.3 を実行すると、図 1.5 のように NPC が四角形を描くような移動を繰り返します。

ここまで、NPC プログラミング環境の使い方の紹介として NPC に簡単な動きを

▼表 1.6: JMP（JuMP）アクション

書式	`JMP <オフセット>`
動作	NPC が次に実行するアクションのインデックスにオフセットを加算する。



▲図 1.5: 「四角形を描くように移動」プログラムの実行結果

させるプログラムを作ってみました。次の章では、このような NPC のプログラミングでシステムコールを行う方法について紹介します。

第 2 章　システムコール

この章では、NPC プログラミング環境でのシステムコールについて解説します。システムコールで使用する新しいアクションを段階的に紹介し、それらを使って Write システムコールで文字列を出力するプログラムを段階的に作り上げていきます。

2.1　システムコールとは

システムコールは、ユーザーランドのプログラムがカーネルの機能を利用するための仕組みです。RPGOS では、システムコールを「電話」で表現しています（図 2.1）。

▲図 2.1: NPC プログラミング環境上の電話

それはどういうことかというと、まず、前章でも紹介した地面の色を少し変えて「ユーザーランド (アプリ)」というラベルを付けている左下の領域をユーザーランドの領域ということにしていて、そこより外側をカーネルの領域ということにしています。そして、ユーザーランドにいる NPC にも、カーネルにいる NPC にも、近くに電話があります。ユーザーランドにいる NPC はカーネルにいる NPC へ電話をかけることで、カーネルの機能を利用できます。このようにして、RPGOS では、システムコールを「電話」で表現しています。詳しくは、1 章でも紹介した既刊の同人誌「RPG のような世界で動く OS の本」をご覧ください。

NPC プログラミング環境にて、現在対応しているシステムコールは以下の通りです。

- システムコール番号 0：Read システムコール（標準入力のみ対応）
- システムコール番号 1：Write システムコール（標準出力のみ対応）

NPC プログラミング環境ではファイルは未対応（標準入出力のみ）

通常、Read システムコールや Write システムコールはファイルディスクリプタにより、Open したファイルに対して読み込みや書き込みを行うこともできます。しかし、現状の NPC プログラミング環境では標準入出力にのみ対応している状況のため[*1]、ファイルディスクリプタの指定は省略しています。

以降、この章では、Write システムコールを例に、アクションを用いて NPC がシステムコールを実行する方法を紹介します。

2.2　第 1 段階：電話への移動と番号設定

それでは、Write システムコールで「HELLO」という文字列を画面に出力するプログラムを、3 つの段階に分けて実装していきます。なお、最終段階のプログラムのみ、サンプルコードリポジトリの ch02 フォルダに write_syscall_complete.act.txt というファイル名で置いています。

♣ プログラムとアクションについて

まず、システムコールを実行するために、NPC を電話へ移動させ、通話先の番号（システムコール番号）を設定する部分を実装します。プログラムとしてはリスト 2.1 の通りです。

▼リスト 2.1: 第 1 段階：電話への移動と番号設定

```
MVP
SWA 1
SFR call_to
```

それぞれのアクションについては表 2.1 と表 2.2、表 2.3 の通りです。

▼表 2.1: MVP（MoVe to my Phone）アクション

書式	`MVP` (引数無し)
動作	NPC を自身の電話へ移動させる。

[*1] RPGOS では Open 等のファイル関連のシステムコールにも対応していますが、アクションを用いた呼び出しがまだできない状況のため、NPC プログラミング環境では未対応です。

各 NPC には自分の電話があり、その座標を覚えています。`MVP` アクションを実行すると、NPC は自身の電話の側へ移動します。なお、この時、NPC は電話の方を向いた状態になります。

▼表 2.2: SWA（Set Work area A）アクション

書式	`SWA <数値|文字|文字列>`
動作	引数を NPC の作業領域 A へ設定する。

▼表 2.3: SFR（Set the FRont）アクション

書式	`SFR <変数名>`
動作	作業領域 A を正面にあるオブジェクトの指定された変数へ設定する。

♣ NPC の作業領域について

`SWA` アクションの動作説明で「NPC の作業領域」という話がありました。これは何かと言うと、任意のデータ（数値、文字、文字列）を格納できる領域です。主に他のアクションで使用するためのデータを一時的に保存しておく用途の作業領域です。現実の CPU にも、CPU の命令が演算などのために使用するデータを格納する領域として「レジスタ」があり、それに相当するものです。

NPC としては、A の他にも B〜D の計 4 つの作業領域を持っています。ただし、任意の値を設定するアクションとしては A の他は B しかありません（表 2.4）。[*2]

▼表 2.4: SWB（Set Work area B）アクション

書式	`SWB <数値|文字|文字列>`
動作	引数を NPC の作業領域 B へ設定する。

今回の場合、`SWA` アクションに `1` という引数を指定して実行しています。そのため、作業領域 A へ「1」が設定されることになります。この「1」というのは、Write システムコールのシステムコール番号です。

♣ オブジェクトアクセスと電話オブジェクトについて

また、`SFR` アクションの動作説明で「オブジェクトの変数へ設定する」という話がありました。これはどういうことかと言うと、まず、この世界のオブジェクトには

[*2] 単にまだ必要にならなかったからというだけです。深い理由はありません。

変数や関数を持っているものがあり、NPC はそれらを通じてオブジェクトへアクセスします。 SFR アクションは、NPC の正面にあるオブジェクトの変数へアクセスするアクションで、自身の作業領域 A の内容をアクションの引数で指定された変数へ設定します。

今回の場合、NPC の正面には自身の電話がある状態です。そして、 SFR アクションの引数に call_to を指定しています。これは、通話先の番号を設定する変数です。加えて、先ほど SWA アクションで作業領域 A へ Write システムコールのシステムコール番号（1）を設定していました。そのため、このアクションは「自身の電話オブジェクトに通話先として Write システムコールのシステムコール番号をダイヤルする」という意味になります。

なお、オブジェクトアクセスについては他にも「オブジェクトの関数を実行する」アクションや、電話オブジェクトについても他に変数や関数があったりしますが、それらはまた後ほど紹介します。

♣ まとめ

まとめると、第 1 段階で実装した 3 つのアクションにより、NPC は「自身の電話まで移動し、通話先として Write システムコール番号をダイヤルする」ということを行います。

2.3　第 2 段階：電話をかけ、相手が出るまで待機

♣ プログラムとアクションについて

第 2 段階では、電話をかけて相手が電話に出るまで待つ所までを実装します。プログラムとしてはリスト 2.2 の通りです。

▼リスト 2.2: 第 2 段階：電話をかけ、相手が出るまで待機

```
MVP
SWA 1
SFR call_to
EFR call       ←追加
EFR is_acked   ←追加
JFF -1         ←追加
```

追加したそれぞれのアクションについては表 2.5 と表 2.6 の通りです。

♣ EFR アクションと電話オブジェクトの関数について

▼表 2.5: EFR（Execute the FRont）アクション

書式	`EFR <関数名>`
動作	NPC の正面にあるオブジェクトの指定された関数を実行する。戻り値がブール値の場合、フラグに格納し、それ以外の場合、作業領域 A に格納する。

▼表 2.6: JFF（Jump if Flag is False）アクション

書式	`JFF <オフセット>`
動作	フラグが false の場合、次に実行するアクションのインデックスにオフセットを加算する。

`EFR` アクションは、正面のオブジェクトの関数を実行します。そして、その戻り値がブール値（true/false）の場合、「フラグ」に格納し、それ以外の場合は、作業領域 A に格納します。

「フラグ」とは、NPC が持つ作業領域とは別の領域で、ブール値を設定できます。現状、フラグは 1 つだけです。このフラグを使って、「実行結果に応じてジャンプ」といったことを行います。（詳しくは後述します。）

今回、 `EFR` アクションを使って電話オブジェクトの `call` 関数と `is_acked` 関数を実行しています。まず、 `call` 関数は、「電話をかける」関数です。これにより、ダイヤルした番号へ電話がかかります。今回の場合、Write システムコールへ電話がかかり、担当の NPC がその電話を取りに向かうことになります。なお、 `call` 関数には戻り値はありません。[*3]

そして、 `is_acked` 関数は、相手が電話に出たか否かを取得する関数です。戻り値はブール値なのでフラグへ設定されます。

♣ フラグに応じたジャンプについて

先ほど紹介した「フラグ」に応じてジャンプするアクションがあります。 `JFF` はフラグが false の場合に引数で指定したオフセットのアクションへジャンプするアクションです。

逆にフラグが true の場合にジャンプするアクションもあります（表 2.7）。

♣ まとめ

まとめると、第 2 段階で実装した 3 つのアクションにより、NPC は「Write システムコールへ電話をかけて、その担当の NPC が電話に出るまで待つ」ということを

[*3] 現状の実装ではこの場合、作業領域 A へ空文字列（""）が設定されます。

▼表 2.7: JFT（Jump if Flag is True）アクション

書式	`JFT <オフセット>`
動作	フラグが true の場合、次に実行するアクションのインデックスにオフセットを加算する。

行います。

2.4　第 3 段階：メッセージ送信

♣ プログラムとアクションについて

第 3 段階では、Write システムコールで出力したい文字列をメッセージとして送信する処理を実装します。プログラムとしてはリスト 2.3 の通りです。

▼リスト 2.3: 第 3 段階：メッセージ送信

```
MVP
SWA 1
SFR call_to
EFR call
EFR is_acked
JFF -1
SWA HELLO                ←追加
SFR message              ←追加
EFR say                  ←追加
EFR is_message_empty     ←追加
JFF -1                   ←追加
HLT                      ←追加
```

今回、新たに登場したアクションはありません。

♣ 電話オブジェクトのメッセージといくつかの関数について

今回追加した 5 つのアクションの内、まず最初の 2 つのアクションで、`SWA` アクションにより文字列"HELLO"を作業領域 A へ設定[*4]し、それを正面の電話オブジェクトの `message` 変数へ設定しています。これは、電話間で送受信するデータを配置する変数です。（以降、電話間で送受信するデータを「メッセージ」と呼称します。）そして、次のアクションで電話オブジェクトの `say` 関数を実行しています。これは、自身のメッセージの変数の内容を対向（今回の場合 Write システムコールの電話オブ

[*4] ダブルクオート等で囲っていないですが、これで文字列という扱いになります。

ジェクト）のメッセージの変数へ設定します。それにより、メッセージが通話先へ伝えられます。

そして、Write システムコールの担当 NPC は、メッセージを取得すると変数を空文字列（""）で上書きします。`is_message_empty` 関数は対向のメッセージ変数が空文字列になったかをチェックする関数です。これを実行し、`JFF` アクションによりそれが false の間繰り返すことで、Write システムコールの NPC がメッセージを確認し終わるのを待つことができます。

独自エスケープ文字で改行（~n）とバックスペース（~b）が可能

Write システムコールで出力する文字・文字列にはエスケープシーケンスが使えます。エスケープ文字はチルダ（ `~` ）です。現状で対応しているのは改行（ `~n` ）とバックスペース（ `~b` ）のみです。

例えば、 `AAA~b~b~bHELLO~nWORLD` を `message` へ設定し Write システムコールを実行すると、「"AAA"を出力した後、それらを削除し"HELLO"を出力し、改行の後に"WORLD"を出力」という動作を行います。

♣ まとめ

まとめると、第 3 段階で実装した 5 つのアクションにより、NPC は「"HELLO"というメッセージを Write システムコールの担当 NPC へ伝える」ということを行います。最後に `HLT` アクションにより NPC の動作を停止します。

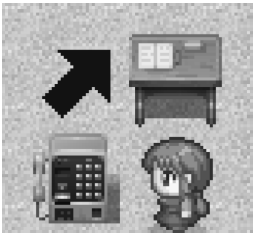
第 1 から第 3 段階までをまとめると、一連のプログラムにより「Write システムコールへ電話をかけ、出力する文字列として"HELLO"を伝えて終了する」ということを行います。

2.5　実行結果

できあがったプログラムを実行してみましょう。なお、左下の「高速モード」のチェックボックスにチェックが入っていない状態だと、「HELLO」の文字列を出力し切るのに 1、2 分ほどの時間がかかります。眺めているのがもどかしい場合は「高速モード」にチェックを入れてください。高速モードだと数秒ほどで出力が完了します。[*5]

[*5] それでも数秒ほどはかかってしまうのですが。。

実行すると、まず「ユーザーランド (アプリ)」領域の NPC が電話の正面で電話をかけます（図 2.2)。電話をかけている最中、電話の上に右上向きの矢印が出ています。

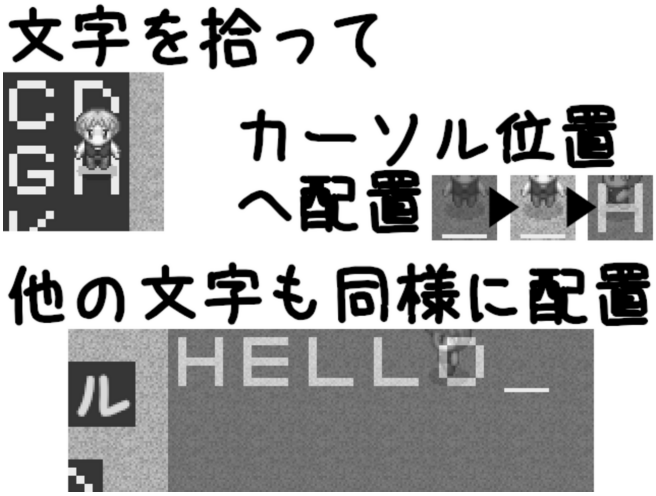


▲図 2.2: ユーザーランド NPC が電話をかける

すると、「Write システムコール (標準出力)」というラベルがある付近の電話が反応し NPC が応答します（図 2.3)。通話中、こちら側（受け側）の電話の上には左下向きの矢印が出ています。

▲図 2.3: Write システムコールの NPC が電話に応答

そして、メッセージを受け取ると、Write システムコールの NPC はその文字を一文字ずつ中央のスクリーンに配置していきます（図 2.4）[*6]。



▲図 2.4: メッセージの文字を一文字ずつ配置していく

配置し終えると、Write システムコールの NPC は定位置（自身の電話の前）へ戻ります（図 2.5）。

▲図 2.5: Write システムコール担当が電話前へ戻る

2.6　マクロアクションを使用したシステムコール実行

前節までで段階的に実装したシステムコール処理はよく使われる処理のため、専用のマクロアクション（ `SYC` ）を用意しています（表 2.8）。

[*6] 衝突判定を実装していないので、移動の際、NPC は電話を踏んで歩いていったりします・・。

▼表 2.8: SYC（SYstem Call）アクション

書式	SYC（引数無し）
動作	作業領域 A に設定されている値をシステムコール番号、作業領域 B に設定されている値をそのシステムコールの引数として、システムコールを実行する。現状で対応しているのは標準入出力に対する Read/Write システムコールのみ。 - 作業領域 A が "0" の場合： 　Read システムコールを実行（取得した内容は作業領域 B へ設定） - 作業領域 A が "1" の場合： 　Write システムコールを実行（作業領域 B の内容を出力）

SYC マクロアクションは内部で前節までで段階的に紹介したアクションを実行します。ただし異なる点として、電話オブジェクトへ与えるメッセージは作業領域 B に設定します。

このアクションを使うと前節までで作成したプログラムはリスト 2.4 のように書けます。このプログラムはサンプルコードリポジトリの ch02 フォルダに write_syscall_macro.act.txt というファイル名で置いています。

▼リスト 2.4: マクロを使用した Write システムコール

```
SWA 1
SWB HELLO
SYC
HLT
```

ここまで、Write システムコールを例にシステムコールの実行方法を紹介しました。次の章では、入力を受け取る Read システムコールと組み合わせたプログラムを紹介します。

第 3 章　エコーバック

この章では、前章で紹介した SYC マクロを用いて Read システムコールを実行する方法を紹介します。そして、入力された文字をそのまま出力する「エコーバック」プログラムを実装します。

3.1　Read システムコールを実行するには

前章で紹介した `SYC` アクションを用いて標準入力から一文字取得する Read システムコールを実装するとリスト 3.1 の通りです。

▼リスト 3.1: Read システムコールを実行する

```
SWA 0
SYC
HLT
```

書いてしまうと単に作業領域 A に Read システムコールのシステムコール番号である 0 を設定し、 `SYC` アクションを実行するだけです。これにより、Read システムコールの担当の NPC へ電話し、そこから標準入力へ入力された一文字を取得し、作業領域 B へ格納します。

なお、一応最後に `HLT` アクションを置いているので、これ単体で実行してみることができますが、動作の様子については、エコーバックまで実装した後に紹介します。

【コラム】Read システムコールをマクロを使わずに書くと

Read システムコールをマクロを使わずに書くとリスト 3.2 の通りです。これもサンプルリポジトリの ch03 フォルダに read_syscall_wo_macro.act.txt というファイル名で置いています。

▼リスト 3.2: Read システムコール呼出をマクロを使わずに書く

```
MVP
SWA 0      ← Readシステムコール番号を作業領域Aへ設定
SFR call_to
EFR call
EFR is_acked
JFF -1
EFR is_message_empty
JFT -1     ←メッセージが空の間待つ
EFR hear   ←メッセージを作業領域Aへ取得
HLT
```

流れとしては Write システムコールの時と同じです。異なるのはダイヤルする番号として Read システムコールの番号（0）を設定している所と、相手が電話に出た後、こちらからメッセージを送るのではなく相手のメッセージを待ち、メッセージが来たらそれを取得している所です。

3.2　エコーバックプログラムの実装

エコーバックとは、入力された文字をそのまま出力することです。ユーザーがキーボードで入力した内容が、そのまま画面に表示されます。これは、Read システムコールと Write システムコールを組み合わせることで実現できます。

先ほどの Read システムコールを実行するプログラム（リスト 3.1）へ、Write システムコールを実行する処理と最初に戻って繰り返す処理を追加すると、エコーバックプログラムになります（リスト 3.3）。なお、このプログラムはサンプルコードリポジトリの ch03 フォルダに echoback_program.act.txt というファイル名で置いています。

▼リスト 3.3: エコーバックプログラム

```
SWA 0
SYC
~~HLT~~      ←削除
SWA 1    ←追加
SYC      ←追加
JMP -4   ←追加
```

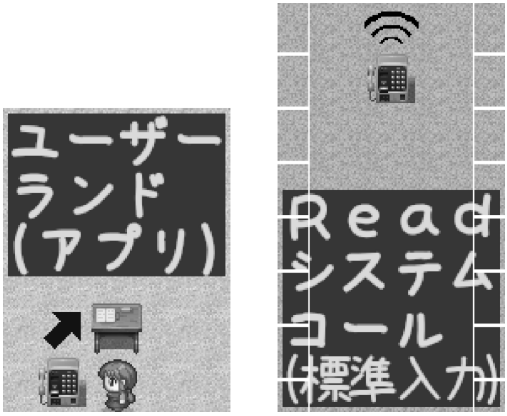
Read システムコールの実行で作業領域 B へ取得された文字を Write システムコールで出力し最初へ戻る、というプログラムになっています。

現状、バックスペースで行をまたげません

現状では、バックスペースで文字を削除する際、行をまたいで前の行の文字を削除することはできません。これは単に内部的なバックスペースの実装がまだ足りていないためです。

3.3　実行結果

リスト 3.3 を実行すると、まずユーザーランドの NPC が Read システムコールの NPC へ電話をかけます（図 3.1）。Read システムコール側の電話の上に WiFi のようなマークが出ているのが呼び出し中であることを示しています。[*1]

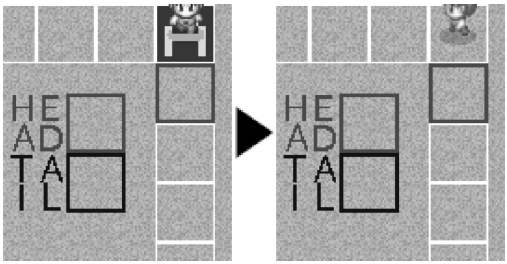


▲図 3.1: ユーザーランドの NPC が Read システムコールへ電話をかける

Read システムコール担当の NPC は白線で白い四角が描かれたリングバッファのヘッド位置で待機しています。「ユーザーが入力したキー情報を取得してから、呼

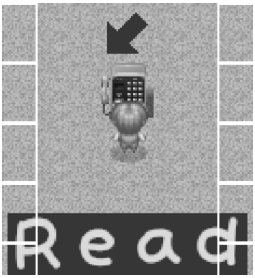
[*1] Write システムコールの時は担当の NPC が直ちに出ていたのでその瞬間が見えなかったですが、呼び出し中は電話の上にこのマークが出ます。

び出しがあれば電話へ向かう」という流れで動作するため、呼び出し中であっても、ユーザーのキー入力があるまでは動きません。キー入力を行い[*2]、入力した文字がリングバッファに現れると、NPC はそれを取得して電話へ向かいます（図 3.2）。この例では「a」キーを押下し[*3]、それが Read システムコールのリングバッファに現れ、それを NPC が取得しました。



▲図 3.2: 入力された文字を取得して電話へ向かう

電話に着くと、NPC は取得した文字を電話の発信者（ユーザーランド NPC）へ伝えます（図 3.3）。



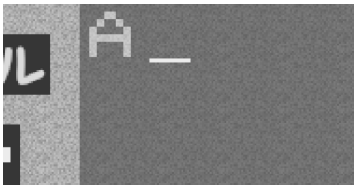
▲図 3.3: 取得した文字をユーザーランド NPC へ伝える

ユーザーランド NPC 側では前述の通り、Read システムコールから得られた文字は作業領域 B へ設定されます。そして、今度はそれを Write システムコールでスク

[*2] フォーカスが右側のフォーム側にある状態だとキー入力はそちらに取られてしまいます。一度、RPGOS の領域をクリックするか、それでもフォームのフォーカスが外れない場合は、RPGOS でもフォームでもない白背景の領域を一度クリックした上で、RPGOS の領域をクリックしてみてください。

[*3] この環境上で対応している文字は大文字のみです。小文字で入力しても大文字でリングバッファに現れます。

リーンへ出力します。その流れは 2 章で紹介した通りです。結果だけ示すと、今回入力した「a」は図 3.4 のようにスクリーンへ配置されます。



▲図 3.4: Write システムコールで入力した文字をスクリーンへ出力

そしてまたユーザーランド NPC が Read システムコールへ電話をかけてユーザーの入力文字を取得し、取得した文字を Write システムコールでスクリーンへ出力する、ということを繰り返すことでエコーバックが実現されます。

3.4　裏側表示をオフにしてみる

左下の「裏側表示」のチェックボックスをクリックし、チェックを外してみてください。すると RPGOS 部分の表示が図 3.5 のようになります。

▲図 3.5: 裏側表示をオフ（"ABC"を入力した状態）

こうして見ると、黒背景に白文字の端末上で CLI プログラムとしてエコーバックプログラムを動作させているのと見た目上は変わらなくなります。本書は「NPC のプログラミングをする」本のため、当初から RPG 世界を見せた状態で紹介していましたが、本来はまずこの状態を見てもらって、「実は裏では小人さんががんばっている」というのが RPGOS の世界観です。

おわりに

ここまで読んでいただきありがとうございます！ 本書では、「NPC プログラミング環境」を紹介し、実際にその上で簡単なプログラミングを行ってみました。まだまだできることが限られるものですが、「裏で NPC ががんばることで成り立っている」という世界観を楽しんでいただければ幸いです。

本来であれば、RPGOS 独自の実行ファイル形式（テキスト形式でアクションが羅列されている）のファイルを RPGOS のシェル上でエディタで編集し実行する、とかできれば全てを RPGOS の上で完結できて良かったのですが、新刊執筆の時期にそこまでの実装が間に合いそうになく、RPGOS 自体がブラウザ上で動くのを良い事に、NPC が解釈し実行するアクションを入力するフォームを用意するということを急ごしらえで行ったのがこの「NPC プログラミング環境」です。（そのため、RPGOS の main のブランチから分岐して作業を行っているのですが、どこかでマージしたいところです。）

経緯はともかく、NPC プログラミング環境は（わずかばかりですが）遊べるものになっているかと思いますので、本書を読んでご興味を持っていただけましたら、PLiCy のサイト上でちょろっと遊んでみていただけると嬉しいです。

[付録] シェルスクリプト製コンパイラ（もどき）

本書で紹介してきたアクションは、RPGOS の世界で CPU に相当する NPC が解釈して実行することから、この世界におけるマシン語に相当します。そうなると、高級言語からそのようなマシン語へコンパイルするコンパイラも考えてみることができます。

できることは限定的ですが、C 言語のプログラムを NPC のアクションへコンパイルするコンパイラをシェルスクリプトで作ってみたものをサンプルリポジトリの compiler フォルダに c_to_npc.sh というファイル名で置いています。このコンパイラでコンパイルできるのは、`putchar` 関数と `getchar` 関数、`puts` 関数のみで、制御構文としても使えるのは `while` による無限ループ（`while (1)`）のみです。

サンプルリポジトリの compiler フォルダにはサンプルの C 言語プログラムをいくつか配置しています。詳しくは compiler フォルダの README.md をご覧ください。

RPGOS NPC 独自アセンブリプログラミング

2025 年 10 月 26 日　ver 1.0 （第十二回技術書同人誌博覧会）

著　者　大神祐真
発行者　大神祐真
連絡先　yuma@ohgami.jp
　　https://yuma.ohgami.jp
　　@yohgami (https://x.com/yohgami)
印刷所　日光企画

(powered by Re:VIEW Starter)