Repton Infinity

User manual

Written by
David Lawrence and David Acton

CONTENTS

  More...

  More...

1. Introduction

Welcome to Repton Infinity - a suite of four games plus all that you need to create games of your own. The authors wish you hours of pleasure pitting your wits against our puzzles and making up more for yourself!

Repton Infinity is a "fully-definable game". Like Repton 3 you can design your own levels and characters (or 'sprites'). But Repton Infinity also allows you to say how characters and objects behave - so you can really create an entirely new game! We've designed four games with Repton Infinity and you might like to try playing these first. Our four games are called "Repton 3 - Take 2", "Repton 4", "Robbo" and "Trakker".

1.1. Loading Repton Infinity

Tape users should find two tapes supplied - the GAME tape and the DATA tape. To load Repton Infinity:

- Insert the correct tape in your cassette-player. The BBC version is on side A of the GAME tape, the Electron version on side B of the GAME tape and the Master version on side B of the DATA tape.
- Rewind tape to start if necessary
- Type ***/REPTON** and press RETURN
- Press PLAY on your cassette-player

[Note: if your cassette-player has no motor-control always press STOP once loading has finished.]

Disc users should find two discs supplied - the GAME disc and the DATA disc. To load Repton Infinity, insert the GAME disc into the disc-drive and press SHIFT-BREAK. BBC and Master users will be asked which version they wish to play. BBC users should press '1' and Master users should press '2'.

After a short while the main menu will appear. To start the game, simply press RETURN. Disc users also have the option of selecting a 'data drive' - this is the drive used when loading games or data files. To select your data drive press 0, 1, 2 or 3.

1.2. Playing the game

One of the four games supplied, "Repton 3 - Take 2", is ready to play. To load one of the three other games see section 1.3, Loading Other Games below. Games are divided into 'game files' which each contain four levels. To start play just press '1' which will take you on to level 1. Pressing 2, 3 or 4 will take you on

to the other levels, but you may have to enter a password - see below. The game keys are:

Z,X,/,:	: Move left, right, down and up
RETURN	: Action
Cursor keys	: Scroll screen
M	: Display map
O	: All sound off
T	: Tune on
S	: Sound on
TAB	: End level
ESCAPE	: Lose a life
SHIFT ESCAPE	: End game
P	: Pause game
DELETE	: Resume paused game

The cursor keys are used to scroll the screen around Repton. This allows you to see more of your immediate surroundings. When Repton loses a life, the screen returns to its initial central position.

There are three levels of sound control: no sound at all (press 'O'), sound effects only (press 'S') or tune only (press 'T').

The most notable difference in game play between Repton Infinity and Repton 3 is that there is no longer any time limit for completing a screen. Also, because the game is fully definable, the aim of any particular game may not be collecting diamonds. For each Repton Infinity level there is a **minimum score** and you may not proceed to the next level until you have at least attained this score. Upon reaching the minimum score the screen will flash and a fanfare will be played. You may then press TAB to proceed to the next level or continue on the current level to score more points. There are four levels in each 'game file' and on completion of the last level you will usually be given the name of the next game file to load. The title-page and high-score table will be shown, but your score and number of spare lives will be retained so that you may load a new game file and continue play.

If a level has been designed so that the map can be viewed, pressing (and holding down) 'M' will replace the playing area with a map of the level. Unlike the map options in previous Repton games, the game continues to play when the map is viewed (but Repton is not allowed to move). This is useful if you are waiting for a monster to hatch or something similar, but remember to release 'M' when the monster starts chasing you!

The only other key you may need to use while playing the game is RETURN - the 'action' key. The use of this key is definable, so

  More...

you may have to experiment with it on any new games to see if it does anything! It is not used in "Repton 3 - Take 2" or "Repton 4", but both "Robbo" and "Trakker" use it for various purposes.

As with other Repton games, there is a password system to allow you to practise higher levels without having to play all the way through to them. However, unlike Repton 3, these passwords are generated by the computer. To start on a different level, simply press the appropriate number from the title page. If that level has a password you will be asked to type it in. If correct you will be able to start on that level and the level will be 'unlocked', meaning you will not need to enter a password for that level again. On completion of a level you will be told the password for the next.

When Repton has lost all four lives, you will be returned to the title page and, if you have gained enough points, you will be invited to enter your name for the high-score table.

Note for Electron users

Unfortunately, BBC game and data files are **not** compatible with the Electron version of Repton Infinity. For this reason, an 'e' is added to the prefix of all data files. So, for example, the sprites for the BBC version of "Trakker" are supplied on your tape/disc as a file called "S.Trak", whereas the Electron version is called "eS.Trak". You will not need to worry about adding the 'e' - it will be done for you.

1.3. Loading game files

The only other control from the title page is 'L' which loads a new game file. These are the files that the linker saves (see the sections 5.2 and 5.3, **Using the Linker**). BBC and Master game files are prefixed by 'G.', Electron games by 'eG.' (you do not need to type in the prefix when entering the filename). Press 'L' and type the name of the desired game file (see list below) followed by RETURN. If all is well the file will be loaded and the title page will return. The title of the game (if any) and the creators of the various parts of the game are shown on the title page. Each of the four games supplied has two game files. Tape users will find these on side A of the DATA tape; Disc users will find them on the DATA disc. The game files are called:

Game	Game filenames
"Repton 3 - Take 2"	G.Rep3A and G.Rep3B
"Repton 4"	G.Rep4A and G.Rep4B

"Robbo"	G.RobboA and G.RobboB
"Trakker"	G.TrakA and G.TrakB

The Electron versions of the game files ("eG.Rep3A" and so on) follow the BBC versions on side A of the DATA tape, so fast-forward your tape as necessary.

Note for Master users

Users of the enhanced Master version may make changes to a game and then try them out immediately. To assist this, an extra key is provided - 'E'. Pressing 'E' from the title screen/high score table, will return you to the main menu. (Other users must press BREAK and reload the game in the usual way.)

1.4. The rules of the games

1.4.1. "Repton 3 - Take 2"

This is the game that is loaded at the start. The rules are very similar to Repton 3. The object is to get as many points as possible by collecting the diamonds and crowns. Eggs crack when allowed to drop and, after a while, hatch out into monsters. Squash these with rocks to score more points. Spirits follow the walls and must be guided into cages whereupon they become diamonds. Safes can be opened by collecting a key. You may 'teleport' to another part of the screen by walking into a transporter, but be warned - you may only use each transporter once. Rocks fall when unsupported and roll off any curved surfaces - avoid being squashed yourself! Fungus is deadly and grows slowly when it has room to do so.

The two sets of "Repton 3 - Take 2" screens are called G.Rep3A and G.Rep3B (or eG.Rep3A and eG.Rep3B for the Electron version).

{ILLUSTRATION REQUIRED - pictures with labels of the following: Repton, transporter, skull, rock, egg, cracked egg, monster, spirit, cage, key, safe fungus, diamond, crown}

1.4.2. "Repton 4"

This game is the successor to Repton 3 and features many new objects and puzzles. Collect the jewels and banknotes to score points. Ghouls hatch out of eggs when cracked and must be squashed with rocks; spirits must be guided into cages and fungus will grow when unattended. Push three 'magiblocks' in a row to make more treasure and drop rocks through 'magic walls' for further gems to appear. Open the safes by collecting a key. You may use the

transporters to travel to other parts of the screen or push items into transporters to send them elsewhere. Ghouls and spirits will also 'teleport' if they have the chance so watch out! The photocopier will duplicate almost anything, but may only be used once.

The two sets of "Repton 4" screens are called G.Rep4A and G.Rep4B (eG.Rep4A and eG.Rep4B for Electron users).

{ILLUSTRATION REQUIRED - pictures with labels of the following: Repton, transporter, jewels, skull and crossbones, rock, magic wall, egg, cracked egg, ghoul, spirit, cage, photocopier, key, safe, fungus, banknote, magiblock}

1.4.3. "Robbo"

You are Robbo, a highly intelligent tenth-generation robot. To test your logic circuits you have been placed in a "time-space puzzle vortex". If you fail to complete all the puzzles you will probably end up on the interstellar scrapheap with all the other old robots and broken coffee-machines.

To complete each puzzle screen simply collect as many flashing orbs as you can and solve other puzzles for bonus points. When you have scored enough the fanfare will be sounded and you can proceed to the next location.

Kettles are harmless, but move right if they can; Things can only move up. A variety of puzzle objects may be moved around and, if pushed into a 'pipe' will be whisked along briskly. Use the lawnmower to cut the grass, but don't stand underneath or you'll experience a rather nasty haircut.

Repair the computer with the spanner and collect the disc as a reward. Activate the toilet with the RETURN key - place a disc at the top and you'll be able to flush out a fish! Leave the light-bulb on top of the power-station for an illuminating experience. Use RETURN to activate the Coke (R) machine, but remember to keep you cans cool in the fridge. Transform your portable phone with the machine and use the transporters to teleport yourself or other objects.

The two sets of "Robbo" screens are called G.RobboA and G.RobboB (eG.RobboA and eG.RobboB for Electron users).

Coke (R) is a registered trademark of the Coca-Cola Company.

  More...

{ILLUSTRATION REQUIRED - pictures with labels of the following: Robbo, transporter, pipes (all four directions), lawnmower, grass, spanner, light bulb, Coke (R) machine, Coke (R) can, power station, toilet, fridge (top+bottom), orb, fish, disc, kettle, 'thing', portable phone, computer, machine}

1.4.4. "Trakker"

As a driver for "J.A.F.F.A." - the "Jagga Anihilation and Fruit Flinging Associates" your responsibility is to dispose of all the hideous Jaggas and other pests that have invaded the land. To assist your task you have been supplied with the latest thing in bulldozers, numerous sticks of dynamite (with detonators) and some top-of-the-range Killafruit! Using these formidable weapons and you own ingenuity you must clear each screen of 'nasties' before proceeding to the next.

To use the dynamite guide Kevin, your associate, to the sticks you want to set off. Drive back to a detonator and press RETURN. Each detonator can only be pressed once. Hideous Jaggas may be squashed with tomatoes, but note: the only sure way of getting them is from behind. Pushing a banana at a Jagga will turn it into a Spider. These may only be squashed with bananas and it is only safe to do so from behind. Use the roadsigns to put an end to the evil Oggles. You must get them from the back though, or they may eat the roadsigns you push at them. Watch out for Repton - he's very angry that you've taken over his game! Trap him so he can't move and he'll fade away. The OOFs (Overnight Oscitant Facilities) may be pushed about to help in this task. Use the telephone boxes to transport to other parts of the screen.

The two sets of "Trakker" screens are called G.TrakA and G.TrakB (eG.TrakA and eG.TrakB for Electron users).

Killafruit (R) is a registered trademark of the J.A.F.F.A. corporation.

OOF (R) is a registered trademark of the Floyd Bedding Company.

{ILLUSTRATION REQUIRED - pictures with labels of the following: bulldozer, telephone box, Tubular Spider, Hideous Jagga, angry Repton, Oggle, OOF, banana, tomato, roadsign, detonator, dynamite, Kevin}

  END

1.5. Creating you own games

Repton Infinity allows you not only to define sprites and levels, but also lets you change how the characters move and interact with each other. You are no longer restricted to rocks that fall and monsters that chase you!

1.5.1. Using discs and tapes

There are five main programs on the disc/tapes - the main game, the three 'editors' and a file linker. Any of these programs can be selected from the main menu. As there are three distinct tasks in designing a Repton Infinity game we decided to give each task its own editor. Thus Land Scape is the screen designer, Blue Print the definitions editor and Film Strip the sprite editor.

Finally, the Linker is used to join together the files that the editors save and create one file that can be loaded into the main game. So, the business of creating your own game will go something like this:

- Design and save sprites using Film Strip
- Define and save how sprites behave using Blue Print
- Design and save levels using Land Scape
- Create a 'game file' by linking the sprite, level and definition files together with the Linker.
- Load the game file into Repton Infinity and play it!

Throughout Repton Infinity, options are selected from menus by using the up and down cursor keys to make a selection and RETURN to confirm it. In most cases ESCAPE can also be pressed to cancel a selected option.

Tape users will need to fast-forward or rewind the GAME tape as necessary when loading the editors they require. The files on the GAME tape are in the following order:

Repton	(loader)
Screen	(title screen)
Menu	(main menu)
Game, Game2	(main game)
Repton	(loader)
Menu	(main menu)
MapEdit	(Land Scape - the level editor)
Menu	(main menu)
DefEdit	(Blue Print - the definitions editor)
Menu	(main menu)
SprEdit	(Film Strip - the sprite editor)

Menu	(main menu)
Linker	(the file linker)
Menu	(main menu)
Game, Game2	(another copy of the main game)

The order of the files may seem a bit odd. Notice that the main menu is saved on the tape after each of the editors - this is so you don't have to rewind the tape all the time just to load the menu. Note also that there is a second copy of the main game saved after the Linker. This is so you can try out your new game after linking the sprite, level and definition files together.

So, to give you an example, let's say we wanted to change the design of a sprite and try it out in the game. The complete procedure for tape users would be something like this:

- Insert GAME tape and rewind
- Type */REPTON and press RETURN
- Press PLAY and wait for main menu
- Select Film Strip from the menu
- Fast-forward tape to file "SprEdit"
- Film Strip - the sprite editor - is loaded
- Remove GAME tape and insert own DATA tape
- Load own sprites from DATA tape
- Alter sprite and resave on DATA tape (see section 4.2, **Editing sprites**)
- Insert GAME tape again
- Quit from Film Strip and wait for menu to be loaded
- Select File Link from menu
- Fast-forward tape to file "Linker"
- Linker will be loaded
- Remove GAME tape and insert DATA tape
- Link files (see sections 5.2 and 5.3, **Using the Linker**)
- Replace GAME tape and quit from Linker
- Select Play Game from menu
- Insert your own DATA tape then load and play your new game file

On the face of it, this process seems a little complicated, but with practice it will become second nature! The above process is much easier when using the disc or Master versions of the game.

The DATA tape contains on side A the game files for our four games. On side B are the definition, object and sprite files for each of the four games supplied. These are followed by the enhanced Master version of the game. You need not worry about these data files at this stage, but when you come to design your own games (after reading the rest of this manual thoroughly!) you

might like to know the exact order of the data files on side B of the data tape. For reference the order is as follows:

Filename	Description
"E.Rep3"	"Repton 3 - Take 2" editor characters
"T.Rep3"	"Repton 3 - Take 2" definitions
"M.Rep3A"	"Repton 3 - Take 2" map file A
"O.Rep3"	BBC "Repton 3 - Take 2" object code
"S.Rep3"	BBC "Repton 3 - Take 2" sprites
"E.Rep4"	"Repton 4" editor characters
"T.Rep4"	"Repton 4" definitions
"O.Rep4"	BBC "Repton 4" object code
"S.Rep4"	BBC "Repton 4" sprites
"E.Robbo"	"Robbo" editor characters
"T.Robbo"	"Robbo" definitions
"O.Robbo"	BBC "Robbo" object code
"S.Robbo"	BBC "Robbo" sprites
"E.Trak"	"Trakker" editor characters
"T.Trak"	"Trakker" definitions
"O.Trak"	BBC "Trakker" object code
"S.Trak"	BBC "Trakker" sprites

The Electron versions of these files follow after these, in the same order. Remember that they will have 'e' prefixes - "eE.Rep3", "eT.Rep3" and so on.

Disc users will find the main game and all the editors on the GAME disc. The DATA disc contains all the data files listed above under the same names. If you have a double disc-drive you can save your own definitions on a blank disc in the other drive. The GAME disc should be inserted in drive 0 and your DATA disc in the other drive (usually drive 1). Press 1 on the main menu to specify that drive 1 contains the data disc. If you have a single drive you will be prompted to remove and insert discs as necessary; make sure that 0 is the chosen data drive on the main menu.

Master users have the benefit of an 'enhanced' version. They do not need to worry about GAME discs or tapes - the game and the three editors are all loaded at the start and need never be reloaded. Just ensure that you have inserted the correct DATA disc or tape when loading or saving sprites, levels or other data. The Linker is also slightly different on the Master version, see section 5.3, **Using the Master Linker** for more details.

Master Compact users, or indeed anybody using ADFS, will have to create some directories before saving data. Directories G, E, M, O, S, T and D are needed. So, to prepare your own ADFS data disc format a new disc and type:

```
*CDIR G
*CDIR E
*CDIR M  (and so on...)
```

1.5.2. Fundamentals for designing games

Here are a few things you should know before starting to design your own game.

Every Repton Infinity game features:

- one or more sets of screens made with Land Scape
- 48 sprites designed with Film Strip
- a set of definitions produced with Blue Print.

Of the 48 sprites, only the first 32 are 'usable'. The remaining 16 are special sprites used for animation. Animation is described in sections 3.3.1, System Flags and section 4.3, **Animated sprites**, but for now, remember that only the first 32 sprites may be placed on a level in Land Scape or defined in Blue Print. The first four sprites are also 'special'. To a certain extent their definitions are fixed (see 3.7.3, The first 4 characters for more details).

Sprite 0 is the 'blank' character. In the game, sprite 0 is put on a square when another character has moved off it.

Sprite 1 is Repton (or whatever character it is that you control in the game). Every level must contain one of these or it will be ignored by the main game. It is not sensible to have more than one Repton on a map and so Land Scape beeps if you try to add a second one.

Sprite 2 is the character that is displayed for the wall surrounding the playing area and is therefore always 'Solid' to stop Repton walking off the sides of the screen!

Sprite 3 is the **transporter** character, these are described in more detail in sections 2.3 (**Placing transporters**), 3.3.1 (System flags) and 3.7.11 (Transporters).

1.5.3. Editor characters

The sprite editor - Film Strip - is used to create not only the

main sprites and map characters used in a game, but also 'editor characters'. These are the little characters shown at the bottom of each editor screen. So that you know what's what, it is often handy to save the editor characters separately (see section 4.4, **Loading and saving sprites** for details) and load them in to the other editors by selecting the Chars option on the editors' menus. Once loaded, a set of Editor characters will be remembered between editors until you press BREAK or load another set. When first loaded, there is a default set of editor characters for "Repton 3 - Take 2" built in.

It is not compulsory to load the 'correct' set of editor characters, but it does make editing games easier. The editor characters are remembered between the three editors, so if a set is loaded into one editor, that set will be used by subsequent editors automatically without reloading it.

On the Electron Version of Blue Print, the editor characters are only displayed in monochrome.

1.5.4. Adding your name

Each editor allows you to enter your name, this will be saved with the data that the editor normally creates, and displayed on the title screen of the game when finally linked together and loaded. You can also give your games titles and 'end messages'; this is dealt with in sections 5.2.1 and 5.3.1, Entering Information.

2. Land Scape - the Screen Designer

2.1. Introduction to Land Scape

This is the screen editor (or more accurately, level designer) of Repton Infinity. The screen is divided into four main sections: the main edit area that takes up most of the screen, the character box below it that shows the available sprites, a small menu in the top right hand corner and an information area in the bottom right. As usual, options are selected with up and down cursor keys and confirmed with RETURN. The information area contains various data for the current level; the level number (1-4), the minimum score set for this level and indications of whether there is a password set and if the map is available while playing the game. Below this are four colour boxes that show the current colours available.

{ILLUSTRATION REQUIRED : Picture of Land Scape in use, annotated}

2.2. Editing screens

The **Edit** option should be selected when you wish to edit a game screen. A flashing cursor will appear in the main editing area. Move the cursor with either the cursor keys, or Z,X,: and /. To select a character to place on the screen, use the cursor keys while holding down SHIFT. To place the currently selected character on the screen press RETURN. Hold down RETURN and move the cursor to produce a 'trail' of characters. Characters can be deleted either by selecting the space character or by pressing DELETE. The whole level can be cleared by selecting the space character and then pressing CTRL C. There is no limit to the number of each type of character that can be placed on the map (except transporters and Repton - see below).

The keys 1-4 select which level to edit. Each level can have its own set of screen colours, map option, password and minimum score. Colours are selected with the function keys f0-f3: each key cycles through the eight available colours. If the map is 'on', then it is possible to press M in the game and view the map. Pressing f6 switches the map on and f7 switches it off again. There are no restrictions on which levels can have maps. Passwords in the game are generated by the computer so setting a password merely involves telling the map editor that you wish there to be one - the game does the rest. Key f5 sets 'pass' meaning there is a password and f4 sets 'none' meaning there isn't. Fairly obviously, you cannot set a password on level 1.

To set the minimum score for the current level press f8. A window will pop up and prompt you to type in a number. This can be any

number between 0 and 9999. Do so and press RETURN.

Land Scape cannot calculate how many points are available on a particular level as it would need detailed knowledge of the behaviour of the characters. However, as an aid, press f9 or 'S' when in normal edit mode and a simple score calculator will appear. This displays the first 32 characters and with each one, a number (these will all be 0 to start with) which represents any possible score that that character can give. The block cursor can be moved bewteen the characters with the cursor keys, and the numbers increased and decreased with SHIFT up and down. The score for each character on the level is added to a total displayed at the bottom of the screen. Note that as some characters may change into others, the original characters must also be given a score value. For example, cages in "Repton 3 - Take 2" have the same score as diamonds while eggs must be given the score for killing a monster. Press ESCAPE to leave the calculator.

2.3. Placing transporters

Unlike the other characters, you are limited to six transporters per level. To set up a transporter, select the transporter character and move to where you want the transporter to be. When RETURN is pressed (to place the transporter on the map), the cursor will be replaced by a flashing 'T': this is the transporter's destination. It can be moved in the normal way with any of the cursor movement keys. Move the 'T' to the desired destination point and press RETURN. The 'T' will change colour to indicate the transporter has been placed. You are now free to continue editing in the normal way.

Whenever you move the edit cursor on top of an existing transporter, the 'T' cursor will indicate the transporter's destination. Once the limit of six has been reached, the editor will beep if any attempt is made to place additional transporters. Transporters are deleted in the usual way, either by pressing DELETE, or by placing another character on top of them (This can also be a transporter, in which case the old destination is discarded and you must enter a new one with the 'T' cursor.)

At any point, ESCAPE will leave edit mode and return you to the the Land Scape menu on the right of the screen.

2.4. Loading map files

When **Load** is selected, a window will pop up and you will be asked for a filename to load. Map filenames are prefixed with 'M.' (or 'eM.' on the Electron), but this does not need to be entered.

  More...

Enter the name and press RETURN. (See section 1.5.1, Using discs and tapes for more details.) If all is well, the map file will be loaded and you will be returned to the main menu. A sample map file - the first set of screens for "Repton 3 - Take 2" can be found on your DATA tape or disc. The screens are saved as "M.Rep3A". Master users will find this set of screens already loaded when they first select Land Scape from the main menu. Electron users should remember that their version of the screens is saved as "eM.Rep3A" on their DATA tape.

2.5. Saving map files

Save behaves in a similar way to **Load**, except it saves a file. As with Load, RETURN confirms and ESCAPE aborts. The editor remembers filenames, so unless you wish to change the filename, you only need to press RETURN to confirm the old one.

2.6. Entering your name

When **Name** is selected, a window will pop up and prompt you for your name. This allows you an extra degree of personalisation in the game. This name will be shown next to 'Scenery' on the main game title page.

2.7. Loading editor characters

The **Chars** option allows you to load a new set of editor characters. See section 1.5.3, Editor characters for more details.

2.8. Leaving Land Scape

Finally, the **Quit** option will leave the editor and return you to the main Repton Infinity menu. Insert the GAME disc or tape as necessary.

3. Blue Print - the definitions editor

3.1. Introduction to Blue Print

Blue Print is an editor and 'compiler' that allows you to define how the characters in Repton Infinity move and interact with each other. Definitions are typed in using **Reptol** - a language that is similar in format to Basic or 'C'. These textual definitions are then 'compiled' by Blue Print into an 'object file' which can then be linked with a sprite and map file and then loaded into the main game.

The Blue Print screen is divided into five main sections. The majority of the screen is taken up by the edit area: this is where definitions are entered and edited. Below this is the usual character box that shows the available characters. The right hand section of the screen contains the Blue Print menu at the top, 'memory-used' indicators (text on the left, code on the right) and the insert/overtype indicator. Finally, the box at the top of the screen is used to display messages and error.

{ILLUSTRATION REQUIRED : Annotated picture of Blue Print in use}

This description of Blue Print is divided into three parts. Section 3.1 deals with the mechanics of entering, editing and compiling definitions. Sections 3.2 to 3.6 contain detailed information about the Reptol language itself and how to use it. Finally section 3.7 contains hints and tips for getting the most out of Blue Print and the game as a whole. Note that the definition files for the four supplied games are included for you to look at and experiment with. Disc users will find them on the DATA disc, and tape users on side B of the DATA tape. The object code files for all four games are also provided for you to create you own screens for the supplied games. These can be loaded straight into the Linker. The definition files are prefixed by 'T.' and the object files by 'O.' ('eT.' and 'eO.' on the Electron).

3.1.1. Using Blue Print

To enter definitions, first select **Edit** from the menu, a block cursor will appear in the top left hand corner of the edit box. The cursor can be moved with the cursor keys. If pressed in conjunction with CTRL, it will move up or down a screenfull of text or to the beginning or end of a line. SHIFT cursor keys are used to select which character is being edited. The box cursor at the bottom of the screen indicates the current one.

  More...

3.1.2. Entering text

Definitions are entered simply by typing in text. Pressing RETURN will move the cursor to a new line. Mistakes can be corrected by pressing DELETE which deletes characters to the left of the cursor. If you prefer you can use f9 or CTRL 'A', either of which deletes characters to the right of the cursor. By default the editor is in insert mode, meaning that text typed will be inserted in the current line. f4 toggles between INSERT and OVERTYPE mode. When in OVERTYPE (or indeed, INSERT) mode, f8 or CTRL 'Q' can be used to insert blank spaces at the cursor position.

As you type in text, the text memory used indicator (the bar on the left) will slowly rise. When the bar reaches the top no more text can be entered and you'll have to shorten or delete some definitions.

Blank lines can be inserted with f6 and unwanted lines deleted with f7. Because of the structured nature of the language, it is natural to indent the definitions. This can become tedious when typing in commands at a fairly deep level. Instead of typing lots of spaces at the start of each line, TAB can be used. This moves the cursor to the same level of indentation as the line above it. Electrons users will probably have noticed the absence of a TAB key on their keyboards! They should press CTRL 'I' instead. When you are happy with your definitions, press ESCAPE to leave edit mode.

3.1.3. 'Making' the object code

To turn your definitions into 'object code' which may be used in the game select **Make** from the menu. The top box will say 'Please wait...' and a small box will highlight each of the characters in the box at the bottom in turn. This is to indicate which character is currently being dealt with. It is a 'two-pass compiler' (rather like the Basic assembler) and so the characters will be highlighted twice. The code-memory-used indicator (the bar on the right) will be constantly updated as the object code is compiled. The bar is actually a fair bit bigger than the maximum amount of object code that can fit in the game! This is because the code produced straight away is not particularly compact and must be 'optimised'. This is performed automatically after compilation. During this stage, the code memory indicator will actually go back down to indicate the memory saved by optimising the code. If this optimised code is still to big, an error will be generated.

During compilation, the text memory used indicator will also rise, but in a different colour. This is memory that the compiler needs

to store variables (such as sprite names). Sometimes the compiler may run out of variable space, in which case an error will be generated and compilation will be abandoned. If this happens you'll have to shorten some of your variable names.

Compilation may also be abandoned if a syntactical error is found within the definitions, e.g. 'Bad IF'. If this occurs, the cursor will be positioned on the line that caused the error, and you'll be returned to edit mode. A full list of errors can be found in section 6.3. Once the object code has compiled and optimised successfully, you'll be prompted for an 'O.' prefixed filename under which to save it. If you don't wish to save the code, press ESCAPE. Unless you are using the enhanced Master version you should save the object code for loading into the Linker later on. Master users may try out the new object code in the game immediately and without saving the 'O.' file. Electron users should remember that their object files are prefixed with 'eO.'.

3.1.4. Loading and saving

To save files select **Save** from the menu. Definition files are automatically saved with 'T.' added to the start of their names (or 'eT.' on the Electron version). As with the other editors, file names are remembered, and if one has previously been specified, it will appear as a default name. Once entered, press RETURN to confirm the name or ESCAPE to abort. Any filing errors during saving will be reported at the top of the screen.

Previously saved 'T.' files can be reloaded using the Load option. 'O.' files cannot be reloaded. Selecting **Load** will prompt for a filename. To load a new set of editor characters (into the bottom box) select **Chars** and enter the filename. It is not compulsory to load the 'correct' set of editor sprites, but it does make editing definitions easier.

When in edit mode, you can change the displayed colours by pressing f0-f3, this facility is provided so you make the editor characters 'look right'. Any changes are only temporary and will be discarded when you quit the Blue Print.

Disc users should note that all files are saved to or loaded from the drive specified on the main game menu. If this is drive 0 or 2, you'll be asked to insert the GAME and DATA discs when necessary. Do make sure that you set the data drive number correctly on the main menu.

3.1.5. Entering your name

If a name is entered using the Name option, it will appear alongside the 'Screenplay' credit on the game title page.

3.1.6. Leaving Blue Print

The only other Blue Print menu option is quit: this quits the editor, and returns you to the main game menu.

3.2. Introducing Reptol

This section is designed to be an introduction and tutorial for using Blue Print. After reading this section, you should have a working knowledge of the language and should be able to design a set of your own characters.

Unlike other games designers, there are very few restrictions on creating definitions. Also, because of the 'structured' approach adopted, the language 'Reptol' is very powerful and complex objects can be created very easily. As an example we will build up the definition for a fairly stupid monster that moves across the screen, but occasionally lays Eggs that hatch into similar monsters. We'll also define a sword that can be used to kill the monsters.

The first four sprites already have their definitions fixed to a certain extent and any text typed in on these sprites (expect TYPE information) will be ignored, except Repton's ACTION definition. Our monster cannot therefore be one of these. It would be nice if the monster was animated, so it would therefore be sensible to use one of the two pairs of sprites in the bottom left hand corner as the sprites on the bottom row cannot have their definitions changed either. So we'll use the second sprite on the middle row. We will also need to define an egg and a sword, as these will only be single sprites, any of the other sprites could be used, so we'll use the ones immediately to the right of the monster. (If you want, you could change the sprites and editor characters (with Film Strip) so these three sprites looked like a monster, an egg and a sword.

A character definition in Repton Infinity consists of three major sections: these are called **TYPE**, **ACTION** and **HITS** and are introduced by the keyword **DEFINE**. for example,

```
DEFINE TYPE
```

The **TYPE** section is used for setting the 'system' and 'user' flags for each character. (Flags are simply pieces of information that tell us something about a character - for example, whether it can be squashed by a rock or not.) The **ACTION** section describes how each character moves or behaves. Finally, the **HITS** section is used to decide what happens when one character hits another. The end of a section is denoted either by the end of the definition or another DEFINE.

If you load the "T.Rep3" file into Blue Print ("eT.Rep3" for

  More...

Electron users), select edit mode and have a look at some of the definitions for "Repton 3 - Take 2", you'll be able to see these sections. Note that it is not compulsory to include all three sections. Indeed some definitions have none of them. There is also a nominal fourth section, this is the **NAME**. This is used to name a character for subsequent reference. If present, **NAME** should be placed at the very top of a definition. As with the main sections, you do not have to give a character a name, but if you ever need to refer to that character from within a definition, then a name is essential. So, for example, to name our Egg, type the words:

```
NAME Egg
```

on the top line of the definition. Also name the monster in a similar way with:

```
NAME Monster
```

and of course the sword, with:

```
NAME Sword
```

Note that the editor DOES distinguish between upper and lower case, so Egg, egg and EGG are all different. Having given all our characters names in this way we can refer to them when defining others.

3.3. The TYPE section

This is used to set up various flags for a character. There are two types of information that can be entered in the **TYPE** section. These are called the 'system flags' and 'user flags'. The system flags are all pre-defined flags that are tested, and acted upon, by the game itself. The user flags, as the name implies, can be defined by the user and tested by the user's definitions. To set either sort of flag as 'true' for a definition simply include the name of the flag after the line **DEFINE TYPE**. A typical **TYPE** section might look something like:

```
DEFINE TYPE
  NotSolidToRock
  Squash
  Transport
```

where the flag 'NotSolidToRock' is a user one and the others are system flags.

  More...

3.3.1. System Flags

The system flags are used wholly internally to the game and cannot be tested by a user definition. They are used to set up important information about the character. There are 12 system flags, although only 7 of these are of general use. The last one can only be used by Repton and Transporters.

The names and functions of the 12 system flags are detailed below.

Solid - set if this character is solid to Repton. I.e. if Repton is allowed to walk on to it. The outside wall (sprite 2) is **Solid** in "Repton 3 - Take 2".

We'll make our egg **Solid** and also the sword, as Repton needs to push it. Because our monster is nasty though, we'll not make it **Solid**, but allow Repton to walk into it (and suffer the consequences!)

Deadly - set if this character kills Repton if it moves on top of Repton, or Repton moves on top of it. Monsters and spirits in "Repton 3 - Take 2" are **Deadly**.

So far, only the monster will kill Repton, so we only need to make the monster **Deadly**.

HPush - set if this character can be pushed horizontally (across the screen). The speed at which it is pushed is dependent on its movement speed (see section 3.3.2, Movement speed more information). Eggs and Rocks are **HPush** in "Repton 3 - Take 2".

VPush - set if this character can be pushed vertically (up and down the screen). See **HPush** for details about the speed. The 'magiblocks' in "Repton 4" are **VPush**.

We'll allow Repton to push the sword in any direction, so we'll set both these flags for the sword.

Squash - set if this character can have other objects pushed on to (or under, see below) it. The space (sprite 0) and monsters are **Squash** in "Repton 3 - Take 2".

The monster needs to have this flag set, as it can be killed when hit by the sword. We also need to set this flag for the space character (sprite 0) otherwise Repton won't be able to push the sword around at all!

Under - set if characters moving on to this character should

appear to move under it. This is only a visual change - it does not affect movement in any other way. In "Repton 3 - Take 2" the earth is not **Under** (Repton is seen to move over it), but cages are (spirits 'slide' underneath them before turning them into diamonds.)

We'll set the **Under** flag for our monster.

Transport - set if this character can use transporters. If this flag is set, then the character will be transported when it enters a transporter. In "Repton 3 - Take 2" only Repton himself is **Tranport**, but it would be very easy to make monsters, spirits or rocks transportable as well. Rocks are transportable in "Repton 4". Also you could deny Repton access to transporters (like in "Trakker") by making them **Solid** and not putting **Transport** into Repton's TYPE section.

We are not interested in Transporters yet, so we won't bother with this flag.

Cycle - although this can be placed in any TYPE section, it is only of use in Repton's or the transporter's. If set in Repton's, then his horizontal movement will be 'cyclic' rather than 'oscillatory'. In simple terms, if you want Repton to walk, then don't put in **Cycle** (as with "Repton 3 - Take 2"); if you want him to rotate in any way, then use the **Cycle** flag. In the "Trakker" game you control a bulldozer with rotating caterpillar tracks: this is achieved by using **Cycle**. More information on this facility can be found in section 4.3.3, Repton's animation. If **Cycle** is set in the Transporter's defintion, then the transporter will 'regenerate' after it is used, i.e. it can then be used again and again. In all the supplied games the transporters do not **Cycle**.

Again, this flag is not relevant to our definitions, so we won't use it.

Animate - if you want a character to be animated, then this flag should be set. The game will then animate this sprite with the one 'below' it. Please read the section in Film Strip about animation for more details. In "Repton 3 - Take 2" only monsters and spirits are animated.

We do want our monster to **Animate**, so we'll set this flag for him, but not for the egg or the sword.

3.3.2. Movement speed

The final three flags describe how fast characters move. The flags

  More...

are called **One**, **Two** and **Four**. These describe the speed in terms of how fast Repton moves. In other words setting the **One** flag means the character will move the same speed as Repton, **Two** means twice as fast and **Four** four times as fast. Rocks in "Repton 3 - Take 2" move at speed **Four**. The speed dictates both the 'natural' movement of the character as described in the definition and also the speed the character moves when it is pushed by Repton. If a character moves, but no speed flag is set in the TYPE section, then it defaults to speed **Four**.

As Repton is going to be pushing the Sword, we'll make it move the same speed as him, i.e. **One**, but we'll make the monster move at **Two** so he's hard to get! As the egg doesn't move, it doesn't need a speed.

3.3.3. User Flags

The eight user flags are rather like the system flags, but their names and meaning can be altered by the user. The easiest way to describe these flags is to give an example. In "Repton 3 - Take 2", some of the characters are curved to allow rocks to roll round them. This effect could be achieved by an extensive rock definition that recognised each of these curved sections and acted upon them. But by far the simplest (and shortest) way is to use two of the user flags. We will name these flags 'CurvedLeft' and 'CurvedRight' to indicate the nature of the curvature of the character. If a character is curved both ways (like a diamond) then both flags will be set for that character.

It is very easy to create a new user flag - simply enter a new name within the **TYPE** section. In other words, if Blue Print finds a word in this section that it doesn't recognise as a system flag then it creates a user flag of that name. So to create our two new flags, just type in the words 'CurvedLeft' and 'CurvedRight' under the appropriate character definitions. Alternatively, just look at the definition file for "Repton 3 - Take 2". The definition for a rock can now be greatly simplified to just testing the 'Curved' flags for the character below it. Testing user flags is dealt with in full below, in the **IF** section.

The only user flag we need in our sword/monster/egg definitions is one indicating if a monster can move on to a character. This is a bit like the 'Solid' system flag, but as that is explicitly for Repton's use, we'll have to create our own. We'll allow the monster to move over open space and Repton at the moment. The latter is because we want the monster to kill Repton if he hits him, so he must be able to move on to Repton's square. As it is likely that the majority of characters will actually be solid to

the monster, we'll call out flag 'MonsterOK', and will put it in the TYPE sections of the characters the Monster can move on to.

3.3.4. Setting the flags...

We can now enter this information into Blue Print, as we are introducing a new section, i.e. the **TYPE** section, we first need to enter (for each definition):

```
DEFINE TYPE
```

Then for each definition enter the flags required. Thus for a Sword we need: Solid, HPush, VPush and One. These words should be entered one per line under the **DEFINE TYPE** heading. It is also usual to indent the flags slightly, although this is not compulsory, it does make the definitions easier to read. The definition for the sword should now look like this:

```
NAME Sword

DEFINE TYPE
  Solid
  HPush
  VPush
  One
```

The Egg like this:

```
NAME Egg

DEFINE TYPE
  Solid
```

The monster:

```
NAME Monster

DEFINE TYPE
  Deadly
  Squash
  Under
  Animate
  Two
```

The space character (sprite 0):

```
NAME Space
```

  More...

```
DEFINE TYPE
  Squash
  MonsterOK
```

and, finally, Repton:

```
NAME Repton

DEFINE TYPE
  MonsterOK
```

3.4. The ACTION section

3.4.1. The use of ACTION

It may help to understand a little of how Repton Infinity works before appreciating what the **ACTION** section is for. When playing a Repton Infinity game the computer is continually 'scanning' the map, asking each character what it wants to do. Some characters (e.g. wall sections or spaces) don't do anything and so do not have **ACTION** sections in their definitions. Others, like monsters or rocks behave in specific ways. These ways are defined using Reptol in the **ACTION** sections of their definitions. So, for example, while playing "Repton 4" every time the computer finds a photocopier on the map it asks it what to do - the **ACTION** part of a photocopier's definition is executed.

To enable you to describe a wide variety of behaviours for characters, the Reptol language incorporates some of the facilities of programming languages such as Basic, C or Pascal. The facilities of Reptol can be broken down into four groups and are listed below. By the way, if you've experience in programming you should pick up the ideas of Reptol very quickly; if not, don't worry - programming in Reptol is not as difficult is it may seem!

- An **IF...ELSE...ENDIF** 'construct' for decision making.
- Testable conditions including the system and user flags.
- Commands for moving, creating and changing characters.
- Special sound and visual effects.

Most definitions consist of one or more tests followed by commands or effects: that is, an **IF** command followed by some actions dependent on the outcome of the **IF**.

As our main example, we will concentrate on the definition for the monster detailed above. We want this monster to move left and right across the screen, changing direction if it can't continue in its current direction. We also want it to 'lay' an egg every so often which will hatch into another monster. We will build this definition up slowly, starting from the top with a broad view, and working down to the details later.

3.4.2. Using the IF statement

First, let's concentrate on the movement. Obviously there are two 'states' to our monster - moving and laying an egg. Conveniently (and not entirely accidentally) Repton Infinity allows each

  More...

character on the map to have two states. The state of a character can be set explicitly, 'flipped' to the other state and tested, all from within Reptol. The two states are simply called state 0 and state 1. We will nominate state 0 as the moving state and state 1 as the egg-laying state. This is because all characters start in state 0 at the beginning of a level.

So, in the broadest terms, our definition could be thought of as being something like:

```
IF STATE(0)
  <move left/right>
ELSE
  <lay an egg>
ENDIF
```

(Note that at the moment, we have not given the specific instructions for moving or laying, this will be added later. So do not try 'making' the above definition in Blue Print!)

This program fragment introduces the **IF...ELSE...ENDIF** construct which will be familiar to programmers. What it means in English is "If we are in state 0 then move left or right; otherwise lay an egg". Note that there is no keyword 'THEN' as there is in other programming languages. Instead the IF and the condition we are testing (in this case STATE(0)) must be on a line by themselves. In fact this convention is used throughout Reptol - there is no concept of 'multiple statement lines' - put each individual instruction on its own line.

For every **IF**, there must be a matching **ENDIF**: if you miss out either Blue Print will complain when you try to 'make' the game. There doesn't have to be an **ELSE**, but you will often find things easier using **ELSE**. The definition above could have been written as:

```
IF STATE(1)
  <lay an egg>
ELSE
  <move left/right>
ENDIF
```

or even:

```
IF STATE(0)
  <move left/right>
ENDIF
IF STATE(1)
```

  More...

```
  <lay an egg>
ENDIF
```

(In fact, we will find later that this isn't **exactly** the same, because if the <move left/right> part of the definition actually changes the state to 1 (which our definition will do eventually), then the <lay an egg> part will also be executed.)

You may place one **IF...ELSE...ENDIF** inside another and may continue 'nesting' IFs in this way up to a depth of 8. We will see nested IFs later, but not that many!

You can also optionally follow the **IF** with the keyword '**NOT**', if so, the 'truth' of the condition will be reversed. For example, '**IF NOT STATE(0)**' is equivalent to '**IF STATE(1)**'. Any **IF** may be followed by a **NOT**, so all the following are valid:

```
IF NOT MonsterOK
IF NOT MOVING
IF NOT CONTENTS Repton
```

The keyword **STATE(0)** is used here as a testable condition. We will see later that it can also be used as an explicit statement, where it means 'set the state of this character to 0'. Here it is used in conjunction with IF and so means 'is the state of this character 0?".

3.4.3. LOOKing and MOVEing

We can now start on the details of the actual movement. The first stage is to think (in English) how exactly we want the monster to move: we can then translate this into Reptol. The movement could be thought of as:

```
- can I move forward?
        - yes : move forward one square
        - no  : can I move backwards?
                - yes : turn round and move forward
                - no  : I can't move, so do nothing.
```

We have already defined a user flag, **MonsterOK** that indicates whether a monster can move on to an other character, so we need to be able to see if the character in front of the monster is **MonsterOK**. The Reptol keyword to investigate a square is **LOOK**, and must be followed by a letter or two in brackets indicating a direction. This can either be a compass direction, or a 'relative' direction. There are 8 compass directions, the four cardinal points, N, E, S & W or one of the logical combinations of two of

  More...

these, i.e. NE, SE, SW or NW. North is taken to be the top of the screen, South the bottom and East and West, the right and left respectively. These latter two should not be confused with the relative directions F, L, R & B. These stand for Forward, Left, Right and Backwards. As the name implies, these directions actually vary according to the direction that the character is moving. So if a character is moving West, then 'Forward' is West, 'Left' is South, 'Backwards' is East and 'Right' is North.

At the start of a level, all characters are initialised with 'forward' meaning 'West'. Of course, these will change if the character changes direction.

We can make use of these initial settings because we want our monster to move left and right. In other words, we'll make our monster move left first! First, we must examine the square to see if the monster can move forward, and if he can, do so:

```
LOOK(F)
IF MonsterOK
  MOVE(F)
ELSE
  <rest of move>
ENDIF
```

Note the use of the user flag after the **IF**. This checks that the flag **MonsterOK** is set for the last character LOOKed at, in other words the one **F**orwards. If this character has that flag set, then the first half of the IF is executed, in this case the instruction **MOVE(F)**. This means start moving forward. The letter in the brackets indicates the direction. These directions are similar to those available when looking, but cannot be one of the directions NE, SE, SW or NW. If the character was not 'MonsterOK' then the **ELSE** part would be executed.

So, our monster will now forward one square at a time, twice as fast as Repton - remember the System Flag **Two** we set in the **TYPE** section. However, what happens when the monster **can't** move forward? We need to write some code to make him turn round.

All we need to do is check the square **B**ackwards and if it is MonsterOK, then we need to move backwards. As the monster is changing direction, the relative directions will also change, so that **F**orward is now in the opposite direction. This is not a problem, as we want the monster to return the way it came! The above code can be extended thus:

```
LOOK(F)
```

  More...

```
IF MonsterOK
  MOVE(F)
ELSE
  LOOK(B)
  IF MonsterOK
    MOVE(B)
  ENDIF
ENDIF
```

This code also caters for a 'null' move if the monster is 'trapped' - if neither **IF**s are true, then no **MOVE** is performed. Note the use of the nested **IF**s - one inside the other. Indenting the **IF**s in the way shown above makes it easier to read a definition. As a rule, make sure each **IF** is level with its corresponding **ELSE** and **ENDIF**.

It is only valid for a character to make one **MOVE** from within the **ACTION** section. If more than one is given, then only the first will be obeyed. For example in the following, the character would only move north:

```
MOVE(N)
MOVE(E)
```

3.4.4. Random programming - CHANCE

We now need to make the monster lay an egg. We will make this happen randomly, so every so often the monster will pause, lay an egg then continue moving. As we are going to put the 'lay-an-egg' code in the **ELSE** part of the **IF STATE(0)** condition, all we need to do in the move part, is swap to state 1. This can be done with another IF statement:

```
IF CHANCE(1%)
  STATE(1)
ENDIF
```

These lines should follow the previous two **ENDIF**s. There are two points to notice here. The first is the use of the condition **CHANCE**. This is followed by a percentage in brackets. This is probability of the **IF** being executed. The number can vary between 0.01% and 99.99%. In other words, a random number is generated between 0.00 and 100.00, and if it is less than the given percentage then the **IF** part is executed.

3.4.5. Changing STATE

The second point to note is the use of the command **STATE** to

  More...

explicitly set the state of the character. As with the condition **STATE**, it must be followed by a number in brackets (in this case 1): the character will be set to that state.

There is another way to change state; this is with the **FLIP** command. However, rather than setting the state to a given value, **FLIP** swaps or 'inverts' the state. This is useful if you want a definition to alternate between two actions. If **FLIP** wasn't available, then the following code would have to be written every time you wanted to change state:

```
IF STATE(0)
  STATE(1)
ELSE
  STATE(0)
ENDIF
```

FLIP will be used later on in the egg definition.

We now have the complete movement section for the monster, which looks like this:

```
IF STATE(0)
  LOOK(F)
  IF MonsterOK
    MOVE(F)
  ELSE
    LOOK(B)
    IF MonsterOK
      MOVE(B)
    ENDIF
  ENDIF
  IF CHANCE(1%)
    STATE(1)
  ENDIF
ELSE
  <lay an egg>
ENDIF
```


We can now make the Monster lay an egg. Again, it is best to think of what we want to happen in English first and then translate it into Reptol. There are two points that should be considered:

1) The monster should wait a bit before laying the egg.
2) Eggs should be laid below the monster.

The 'English' definition would therefore be something like:

- wait a bit
- look below, is there a space?
  - YES : lay an egg
  - NO : carry on moving.

This can be translated into Reptol as:

```
IF EVENT(4)
  LOOK(S)
  IF CONTENTS Space
    CREATE(Egg,S)
  ENDIF
  STATE(0)
ENDIF
```

This section of code illustrates three more features of Reptol and these will be dealt with separately.

3.4.6. Generating delays - EVENT

This is a testable condition. It is followed by a number between 1 and 7 (in brackets). The best way to explain what **EVENT** does is by way of a table:

EVENT(n)	Maximum wait (approx)
1	0.25 seconds
2	0.5 seconds
3	1 second
4	2 seconds
5	4 seconds
6	8 seconds
7	16 seconds

Imagine each event number as a hand on a clock. Hand 1 rotates at a particular speed (about four times a second), hand 2 rotates at half the speed of hand 1, hand 3 at half the speed of hand 2 and so on. When we say **IF EVENT(1)** we are really saying 'is hand 1 of the clock pointing straight up'. Clearly, hand 1 of the clock

points up twice as often as hand 2 so **EVENT(1)** is twice as likely to occur as **EVENT(2)**.

With this illustration in mind, there are a couple of things to note. First, we don't know where the hands of the clock are at any one time. So, although **IF EVENT(1)** is more likely to be true than **IF EVENT(2)** it might just so happen that hand 2 of the clock is pointing up when we look at it. To expand, if we precede a bit of our definition with **IF EVENT(7)** it is likely that it will be some time before that bit will be executed. But it may just so happen that hand 7 is pointing up and so the code after IF **EVENT(7)** is executed immediately! The timings given above are **maximum** values - if, for example, you use **IF EVENT(5)** in your definition the code that follows will be executed **some time in the next four seconds**.

The second point to note is that **IF EVENT** is only really designed for static objects. The reason for this is that while objects are in the process of moving their **ACTION** definitions are not called. But the seven hands of the clock continue to rotate. The upshot of this is that a moving object may just miss seeing a particular hand pointing upwards. Our monster isn't moving when waiting for an **EVENT** so we don't have to worry about this feature. You may use **EVENT** in the definitions of objects that move at top speed (**Four**) without any worries: this is because they, like static objects, are never 'between squares' - they are always in either one place or another.

3.4.7. The CONTENTS of squares

As mentioned before, whenever a **LOOK** is executed, the user flags can be tested to see if the character in the **LOOK**ed direction is 'suitable'. It is also possible to check if that square contains a specified character. This is where **CONTENTS** comes in. It must be followed by the name of character, i.e. the one following the **NAME** command at the top of a definition. (You must of course remember to name any characters that you might **LOOK** for.) The **IF** will then be executed if the square contained the given character. In the example above, the IF will be executed if the square South, i.e. below, contains a 'Space'.

IMPORTANT NOTE:

An **IF CONTENTS** must immediately follow a **LOOK**, or the **ELSE** of another **IF CONTENTS** statement. Otherwise it cannot be guaranteed to work.

```
LOOK(E)
IF CONTENTS Repton
```

  More...

```
  ...
ELSE
  IF CONTENTS Space
    ...
  ENDIF
ENDIF
```

is therefore valid, but:

```
LOOK(E)
MOVE(W)
IF CONTENTS Space
  ...
ENDIF
```

is **not**. The rule is: don't do anything between **LOOK** and **IF**. There is another use for the **CONTENTS** keyword: this will be seen below in the **CREATE** section.

3.4.8. CREATEing objects

CREATE is used to place characters on the map. It is followed by either one or two parameters in brackets. In the example above we see the two parameter version. Here, the first parameter is the name of a character, as given by the **NAME** command and the second is a direction. This can only be one of the eight compass directions as used by **LOOK** - it is not possible to use the four relative directions. The result of this command is, fairly obviously, to create the given character in the specified direction. In the one-parameter version no direction is given: just the character name. In this case, the new character is created on top of the old one, i.e. the old character changes into the new one. Note that this does not mean that the **ACTION** section for the created character is then executed immediately: the current definition continues to be used until the end. Next time round though, the **ACTION** section of the new character will be used instead.

A **CREATE**d character will 'hit' the character it landed on, and will call the **HITS** section of the 'squashed' character. This is dealt with in much more detail in section 3.6, **The HITS section**.

There is one other feature of **CREATE**. If the character specified to be **CREATE**d is the keyword **CONTENTS**, then the last '**LOOK**ed at' character will be created. This is used by the photocopiers in "Repton 4" and the pipes in "Robbo". As with the positions of **IF CONTENTS**, you must be careful about the position of **CREATE(CONTENTS)**s. It is only guaranteed to work immediately after

  More...

a **LOOK**, or a **LOOK** followed by an **IF** (of any sort). For example:

```
DEFINE ACTION

  LOOK(N)
  IF Copyable
    CREATE(CONTENTS,S)
  ENDIF
```

defines a character which looks upwards and, finding any character which has the use flag **Copyable** set will duplicate it below.

(Back to the definition!)

Finally, whether the monster could lay an egg or not, the monster is swapped back to **STATE(0)**, so that he starts moving next go.

We therefore have a complete definition for the **ACTION** of our monster:

```
DEFINE ACTION

IF STATE(0)
  LOOK(F)
  IF MonsterOK
    MOVE(F)
  ELSE
    LOOK(B)
    IF MonsterOK
      MOVE(B)
    ENDIF
  ENDIF
  IF CHANCE(1%)
    STATE(1)
  ENDIF
ELSE
  IF EVENT(4)
    LOOK(S)
    IF CONTENTS Space
      CREATE(Egg,S)
    ENDIF
    STATE(0)
  ENDIF
ENDIF
```

3.4.9. Using STATE and EVENT together

Now we have created the egg, we need to do something with it. This

will be very simple, but will illustrate the method of using **STATE** and **EVENT** together to allow for a 'minimum time'. As all characters start off in **STATE(0)**, the following definition should suffice:

```
IF EVENT(5)
  FLIP
  IF STATE(0)
    CREATE(Monster)
  ENDIF
ENDIF
```

The first **EVENT(5)** will flip to **STATE(1)** and so the **IF STATE(0)** will not be executed. Upon the next **EVENT(5)** the **STATE** will flip back to 0 and the monster will be created. This means that the egg will wait for at least one **EVENT(5)** (about 4 seconds) before changing. This may be exactly one **EVENT(5)** or as much as two (about 8 seconds) depending on the position of hand 5 of our **EVENT** clock when the egg was created.

This definition also demonstrates the use of a single parameter **CREATE** - here, the monster will be created on top of the egg.

The sword does not do anything in itself - it is pushed by Repton and may kill a monster if it hits one. As such it does not need any **ACTION** section at all. This does not matter: as previously mentioned, characters need not have all three **DEFINE** sections.

Further programming

Before leaving this description of the **ACTION** section, there are a number of other commands and conditions available. These have not been left until last because they are unimportant; merely that they have not been needed in our tutorial example.

3.4.10. Testing movement - MOVING

It is often useful to check if the character we are dealing with is moving or not. This can be done with the condition '**MOVING**'. For example, Rocks in "Repton 3 - Take 2", must kill Repton if they land on him, but don't if he simply walks beneath them. The code to deal with this look like this:

```
LOOK(S)
IF CONTENTS Repton
  IF MOVING
    KILLREPTON
  ENDIF
```

  More...

```
ENDIF
```

(In fact, the '**MOVING**' flag is set whenever a character moves and cleared if it doesn't move. So, **IF MOVING** will be 'true' if the character moved 'last go').

The **MOVING** condition is also useful for creating objects that continue moving once pushed. This is very easily achieved:

```
IF MOVING
  LOOK(F)
  IF CONTENTS Space
    MOVE(F)
  ENDIF
ENDIF
```

This means that if the character starts moving in **any** way, then each turn, it will look forward and move on if there is nothing in the way. Otherwise, it will stop moving completely. The road signs in "Trakker" behave like this.

3.4.11. Killing Repton - KILLREPTON

The above piece of code also illustrates another feature of Reptol. Normally Repton will only lose a life if collides with a 'Deadly' character, i.e. one with the 'Deadly' system flag set. However, Rocks kill Repton not by colliding with him, but by landing on his head. The command '**KILLREPTON**' is used in such cases. This will kill Repton wherever he is on the screen - he doesn't even need to be near the character that killed him!

3.4.12. Chasing Repton - NORTHOF etc.

There is no method of determining Repton's exact position, although there are four conditions that can be used to work out which direction he is in. These are most suited for designing monsters that chase Repton, although they need not be limited to just this. The four conditions are **NORTHOF**, **EASTOF**, **SOUTHOF** and **WESTOF**. Their meaning should be self explanatory, i.e. in:

```
IF NORTHOF
  ...
```

the **IF** will be executed if this character is north of Repton, i.e. further up the screen. The other three work in a similar way, but obviously check for the other directions. In "Trakker" the hideous Jaggas and Tubular Spiders use these conditions, but in different ways, so that they chase the bulldozer slightly differently.

  More...

3.4.13. Examining the Return KEY

There is one final testable condition - **KEY** - this tests the RETURN key on the keyboard. This facility has been used in "Robbo" (to activate the Cola (R) machine and flush the toilet) and "Trakker" (to detonate the dynamite). For example, the Cola (R) machine's definition looks like this:

```
LOOK(E)
IF CONTENTS Repton
  IF KEY
    LOOK(S)
    IF CONTENTS Space
      CREATE(Can,S)
    ENDIF
  ENDIF
ENDIF
```

In other words, if Repton is standing to the east (right), RETURN is being pressed and there is a space below, create a Can there. (Exercise for reader: in fact, the real definition only allows one can to be dispensed - try changing the above definition to do this.)

3.4.14. ENDing a defintion

If you want to stop executing a definition before the end, you can use the keyword **END**. This will stop execution immediately, and move on to the next character. Although rather contrived, the following example does illustrate the use of **END**:

```
LOOK(E)
IF CONTENTS Space
  CREATE(Bag,E)
  END
ENDIF
CREATE(Cake)
```

If there is a space to the east, then this definition will create a 'Bag' there and then do nothing else. If there isn't a space, then this character will change into a 'Cake'. Note that it is perfectly valid to place an **END** in the body of an IF statement - **END** can be placed anywhere within a definition.

3.4.15. CHANGEing characters

The final command that actually affects the game directly is

CHANGE. This is followed by two character names (in brackets) separated by a comma. The command has the effect of changing all occurrences of the first character on the current level by the second. "Repton 3 - Take 2" uses:

```
CHANGE(Safe,Diamond)
```

in the definition of a key to change all safes into diamonds when it is collected.

IMPORTANT NOTE: **CHANGE** must only be used if both characters specified move at the same speed, i.e. both move at One, Two, Four or not at all. Very strange things will happen if this rule is not adhered to.

CHANGE could be used to change vicious monsters into 'scared' monsters when a 'power pill' is collected, as in Snapper.

3.4.16. GOTO and LABEL

The last two commands in this section, **GOTO** and **LABEL**, can be used for a number of purposes. Essentially they change the flow of execution of a definition. **GOTO** is not usually encouraged in normal programming, as it tends to lead to messy listings that are difficult to follow. However, as Reptol is such a small language **GOTO** is rather useful.

As there are no line numbers in Reptol, the destination of the **GOTO** must be marked in some way. This is done with the keyword **LABEL**. It is followed by string of one or more characters. This is called the name of the label. To jump to a label from within a definition, simply enter the keyword **GOTO** and follow it with the name of the label you wish to jump to. As with **END**, a **GOTO** can appear anywhere within a definition. Here's another (contrived) example:

```
LOOK(N)
IF CONTENTS Space
  MOVE(N)
  GOTO notree
ENDIF
LOOK(W)
IF CONTENTS Space
  CREATE(Tree,W)
ENDIF

LABEL notree
...
```

If the first **IF** is executed, then the character will move north, but then execution will jump to the label 'notree', therefore missing out the second **IF**.

It is perfectly valid to **GOTO** a label in another definition to save memory.

You can also place a **LABEL** in a **HITS** section and jump to it in the usual way, or indeed jump to a **LABEL** in an **ACTION** routine and jump to if from a **GOTO** in a **HITS** routine. However this will not work if a **CREATE** command is then executed. In general you shouldn't use **GOTO** to jump about between **ACTION** and **HITS** sections - this practice is only recommended when using just the 'special effect' commands or **CHANGE**.

3.5. 'Special effect' commands

There are four special commands that cannot easily be grouped under **ACTION** or **HITS** because they can be used in either section. In most games, they are more likely to be found in the **HITS** section, although this is be no means 'law'! The four commands are:

3.5.1. SCOREing points

SCORE is followed by a number between 1 and 255 in brackets, for example **SCORE(10)**. This number is added to the player's score. If this takes the score beyond the 'minimum score' for the level the screen is flashed and the fanfare played.

3.5.2. FLASHing the screen

FLASH is followed by the name of a colour in brackets. The screen's background colour will momentarily change to the one specified. 'col' can be one of the following colours:

RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE.

If you prefer, 'col' can also be a colour number, 1-7. For example, both the following are valid, and in fact, have the same effect:

FLASH(YELLOW)
FLASH(3)

3.5.3. Generating SOUNDs

SOUND is followed by a number between 0 and 255 in brackets and is used to generate a sound effect (on sound channel 1), for example **SOUND(67)**. There are four types of sound effect available and you may choose one of 64 different pitches (0 to 63) for each effect. The four effects are listed below.

Sound effect	Description
0	Short 'bell'
1	'Whizz!'
2	'Warble'
3	Extended 'bell'

To work out the parameter for a **SOUND** command simply add the desired pitch value (0=lowest pitch, 63=highest) to 64 times the sound effect number. For example, to generate a note of pitch 32

  More...

using the 'warble' effect you would use a parameter of 32+(64*2)=160. So you would include **SOUND(160)** in your definition.

Interested readers might like to know that each sound effect is produced using a different sound 'envelope'. The actual parameters of these four envelopes can be found in section 3.7.17, The envelopes

3.5.4. Making sound EFFECTs

EFFECT is very similar to **SOUND**, but uses the 'noise channel' (sound channel 0). The **EFFECT** parameter is calculated in the same way as that for **SOUND**, and the results are not dissimilar either. Here are a couple of examples to try:

EFFECT(32)	Short 'drum' noise
EFFECT(224)	'Whoosh'
EFFECT(160)	'Warbly whoosh'

Experiment to find interesting effects of your own.

Examples of all of these commands can be found in the definition files supplied for four games provided.

3.6. The HITS section

3.6.1. The use of HITS

The **HITS** section of a character's definition (usually called the 'hit routine') is executed whenever the character is hit by another character. This can happen in four ways:

1) Normal movement - when a character completes a move.
2) Creation - when a character is **CREATE**d on top of another.
3) Pushing - when a character is pushed (by Repton) on top of another.
4) Transportation - when a character enters a transporter, the hit routine for the destination square of a transporter is called.

3.6.2. Hit routine commands

Defining a hit routine is not different from defining a character's movement except you can only use a selection of Reptol commands. Also, **IF** can only be used with **HITBY** - a condition we have not met yet. The use of this condition will be explained later. The commands and keywords available are:

  More...

Commands: **CHANGE, KILLREPTON, CREATE**
Effects: **SCORE, SOUND, EFFECT, FLASH**
Structure: **IF, NOT, ELSE, ENDIF, GOTO, LABEL, END**
Conditions: **HITBY**

3.6.3. Using the HITBY condition

In our example definitions, only the monster will have a **HITS** section, although after reading this section, perhaps you would like to add one for the egg.

The monster is going to be stabbed by the sword, when this happens, we'll flash the screen, do some sort of sound effect and award some points for the player's effort. As we are also allowing Repton to move on to the Monster's square, even though he'll lose a life, we don't want to give away any points or such like! We therefore need to test which character has actually hit the monster. This is where the new condition, **HITBY** comes in. It must be followed by the name of character as given by the **NAME** command. The **IF** will be executed if that character is 'doing the hit'. Our monster's hit routine will therefore be something like:

```
DEFINE HITS

IF HITBY Sword
  FLASH(RED)
  SOUND(130)
  SCORE(20)
ENDIF
```

The actions of the 'special effect' commands in the body of the **IF** should be fairly self explanatory. If you enter the above code following straight after the **DEFINE ACTION** section of the monster, you'll have completed the definitions for our example. If you want to try them out:

- Save the textual definitions with the **Save** option from the Blue Print Menu
- Select **Make** to compile the definitions
- Quit Blue Print
- Design some sprites with Film Strip if you have not already done so
- Design a simple level containing some swords, some monsters and one Repton
- Load the Linker and link the three data files together
- Load the linked game file into the main game and try playing it!

It is quite often unnecessary to use an 'IF HITBY ...'. For example, in "Repton 3 - Take 2", by careful setting of system and user flags, Repton is the only character allowed to move on to a key. The hit routine for the key can therefore simply be:

```
DEFINE HITS
  CHANGE(Safe,Diamond)
  EFFECT(128)
```

There are a few things to beware of when using the **CREATE** command from within a hit routine: these are dealt with in section 3.7.16, Recursive programming.

For further examples of hit routines, you may like to look at the definitions of the monster, cage, key, crown and diamond in "Repton 3 - Take 2".

That completes this tour of Blue Print and the Reptol language. The best way to learn the language is to try things out - that's how we wrote the four games supplied! In section 6.2 there is a summary of the commands available in Reptol which you may like to refer to while designing your own games. If you are just beginning, then please feel free to examine the "T.Rep3" and "T.Rep4" files supplied to see how we built up the definitions. More advanced users could try and work out what the definitions are for "Robbo" and "Trakker" before taking a peek in the supplied files!

3.7. Hints and tips

This section contains a number of useful points that illustrate some of the features of Repton Infinity and the Reptol programming language. There are also some important guidelines that should be adhered to at all times. Don't worry if you do not follow all the ideas here immediately - we've included quite a lot of advanced material for when you're an expert Reptol programmer! Here then, in no particular order, all you ever wanted to know about Reptol, but were afraid to ask...

3.7.1. Continual FLASH

If a definition continually **FLASH**es, then the background will actually keep a steady colour - it will not return to black. This is used by the light bulbs in "Robbo".

3.7.2. Speeding things up

The more complicated the definitions, and the more characters on the map, the slower the game will run. This is unfortunate, but inevitable. To make the game run as fast as possible, try to bear the following points in mind:

(a) Try to keep the definitions of the most common characters as simple as possible. If there are only eight monsters on the screen it doesn't really matter how complicated their definitions are. If there are 150 rocks though, adding a single command to the definition of a rock may have a noticeable effect on the speed of the game.

(b) Where possible, use the **HITS** section to define a character. For example, you could define a cupboard which continually looks to the left to see if there is a packet of biscuits there. If so, you could create a space on top of the biscuits and score 10 points. Alternatively you could define the cupboard as 'squashable' by including **Squash** in its **TYPE** section. Then you could wait until the biscuits were pushed into it. So that the cupboard didn't disappear when hit by the biscuits you might define its **HITS** section like this:

```
DEFINE HITS
  IF HITBY Biscuits
    SCORE(10)
  ENDIF
  CREATE(Cupboard)
```

In this way the 'thinking' part of the cupboard's definition is

only called when the cupboard is hit by something and not continually. This will speed up the game if you use a lot of cupboards on the map, but remember - you will be able to push other objects into cupboard as well. The choice is yours.

(c) Use quick tests before slow ones. **IF STATE** and **IF MOVING** are executed much more quickly than the other **IF**s so use them first. For example:

```
IF STATE(0)
  LOOK(N)
  IF CONTENTS Space
    ...
  ENDIF
ENDIF
```

will be faster to execute than if the **IF**s were in the other order. (Unless of course the state is always 0!)

(d) Try to keep the number of 'slow' commands to a minimum, these are **LOOK**, **IF CHANCE**, **SOUND**, **EFFECT** and **IF KEY**.

(e) The first user flag defined will actually be tested faster than the others so try to use it for the most 'important' functions.

Using 'animated' characters does not affect the speed of the game - this is purely a visual effect.

3.7.3. The first 4 characters

The Space, Repton, Wall and Transporter characters cannot have their definitions altered completely. You can give them all a **DEFINE TYPE** section to set up the relevant system and user flags. The only other thing you have control over is Repton's **ACTION** section (because Repton is dealt with separately - see below). This allows you to add to Repton's built-in definition, which moves him around, and checks if he is pushing things. So, for example, the following will allow him to open 'doors', by pressing RETURN:

```
DEFINE ACTION
  IF KEY
    LOOK(N)
    IF CONTENTS Door
      CREATE(Space,N)
    ENDIF
  ENDIF
```

It is also possible to make him move independently of pressing keys, for example, to make him 'skate' try the following:

```
DEFINE ACTION
  IF MOVING
    LOOK(F)
    IF CONTENTS Space
      MOVE(F)
    ENDIF
  ENDIF
```

Note that when **MOVE**ing in this way, Repton will not use his animation sequence: he will stand still and use the defined 'waiting sequence'.

You cannot define the **HITS** sections for any of these four characters. Blue Print will still try to compile any code you may enter, but the game will ignore it.

3.7.4. Killing Repton

Repton can be killed in four different ways. If:

(a) he walks into a 'Deadly' character, i.e. one with the 'Deadly' system flag set in the **DEFINE TYPE** section.
(b) a 'KILLREPTON' is executed from within a definition.
(c) any other character, deadly or not, moves, or is created on top of him.
(d) the ESCAPE key is pressed.

The main one to make note of here is (c). This happens because otherwise Repton would disappear from the map and you wouldn't be able to carry on playing!

3.7.5. Creating Repton

The **CREATE** command is normally used for placing a character on the map regardless. This would cause strange effects if you could **CREATE** Repton in the same way - there would be two of them on the screen and the game would not function correctly. To avoid this, if Repton is placed on the map with a **CREATE**, then he is first wiped off the map, so that after the creation there is still only one Repton. This also skirts round a possible problem with definitions such as the pipe. The pipe works by creating whatever is to its 'west', on its 'east', then creating a space to its 'west'. If **CREATE(Repton,E)** didn't remove the 'old' Repton, then the **CREATE(Space,W)** would kill him!

  More...

3.7.6. LOOK before you leap!

Always **LOOK** at a square and check if it is suitable to move on to, or **CREATE** on before actually moving or **CREATE**ing. Otherwise, two characters may attempt to enter the same square which will have unpredictable results, or worse, may move off the sides of the screen! [This is acceptable, although not encouraged, for moving off the left or right, but is not acceptable if the character moves off the top or bottom.]

Also remember to that there should be no other commands between a **LOOK** and an **IF CONTENTS**, an **IF <flag>** or a **CREATE(CONTENTS)**. In other words, any command that is related to a **LOOK** should follow the **LOOK** immediately. The following examples are NOT correct:

```
LOOK(N)
IF CONTENTS Cake
  CHANGE(Biscuit,Trifle)
  CREATE(CONTENTS,S)            [CONTENTS is invalid now]
ENDIF

LOOK(S)
IF MOVING
  IF Nasty                      [Invalid test here]
    CREATE(Splat)
  ENDIF
ENDIF
```

The correct versions are:

```
LOOK(N)
IF CONTENTS Cake
  CHANGE(Biscuit,Trifle)
  LOOK(N)
  CREATE(CONTENTS,S)
ENDIF
```

[or do the **CHANGE** after the **CREATE**]

```
IF MOVING
  LOOK(S)
  IF Nasty
    CREATE(Splat)
  ENDIF
ENDIF
```

3.7.7. Pushing

There is no method of checking the contents of a square before an object is pushed (by Repton) on to it. Pushing is dealt with internally and can only check if the square is 'Squashable'. You cannot, for example, just allow a rock to squash a monster; anything that is pushable will be able to as well.

When making things pushable and squashable, remember to set the 'Squash' flag for the space character (sprite 0), otherwise Repton wont be able to push things across empty space!

3.7.8. User flags

Take care when entering user flags. Remember that any unrecognised word under the **DEFINE TYPE** section will be made into a user flag. So if a definition is not quite behaving correctly, check that you have not made any spelling mistakes.

3.7.9. MOVEing twice

Once a character has been told to move with the **MOVE** command, it cannot be stopped or told to move in a different direction.

3.7.10. MOVEing and LOOKing

If a character definition contains a **MOVE** command and then a **LOOK** command, the **LOOK** will still refer to the same square, i.e. a **MOVE** only tells a character that it is about to move; it doesn't actually move it anywhere.

3.7.11. Transporters

Transporters in Repton Infinity have a few more facilities over those in Repton 3:

(a) If the system flag 'Cycle' is placed in the **DEFINE TYPE** section of the transporter, then they do not disappear after they are 'used' - they can be used again.

(b) The hit routine for the destination square of the transporter is called when it is used. At a simple level, this means that if Repton transports on top of diamond for example, he will 'collect' it and so score 5 points. This even works if the destination square contains another transporter, in which case Repton (or whatever is being transported) will go straight to the destination of the last transporter in the chain. Because this is inherently recursive, the 'stack' may fill up, although you will

only notice this if you make transporters 'Cycle', or make use of point (c) below, as there is enough stack space to deal with the standard six transporters.

(c) The 'Cycle' flag globally affects all six transporters. There is no easy way of making it affect only some of them. Instead, you can define a character as follows:

```
DEFINE ACTION
  LOOK(W)
  IF CONTENTS Space
    CREATE(Transporter,W)
  ENDIF
```

This can then be placed to the east of transporters (or, indeed, any other position if the 'W's are changed) that you wish to 'regenerate'. It will recreate those transporters when they are 'used'. Note that if these characters are placed anywhere else, then the transporters they create won't actually work, as they have no defined destinations.

3.7.12. AND and OR

Reptol does not support the Boolean operators AND and OR, but they can be easily simulated with **IF**s and **GOTO**s. If you wanted to write:

```
LOOK(S)
IF CONTENTS Space AND KEY
  MOVE(S)
ENDIF
```

you could write:

```
LOOK(S)
IF CONTENTS Space
  IF KEY
    MOVE(S)
  ENDIF
ENDIF
```

If you wanted to write:

```
LOOK(N)
IF MonsterOK OR NOT CONTENTS Rock
  CREATE(Tree,N)
ENDIF
```

you could write:

```
LOOK(N)
IF MonsterOK
  GOTO tree
ELSE
  IF NOT CONTENTS Rock
    GOTO tree
  ENDIF
ENDIF
END

LABEL tree
CREATE(Tree,N)
```

3.7.13. Diagonal movement

It is not possible to give **MOVE** a 'diagonal' direction, i.e. NE, SE, SW, NW are all impossible; however it is possible to simulate such movement using **CREATE**. If you wanted a character to **MOVE(NW)**, the following definition would do:

```
LOOK(NW)
IF CONTENTS Space
  CREATE(<name>,NW)
  CREATE(Space)
ENDIF
```

This will make the character move a whole square at a time, like system flag 'Four'. The only way to slow it down would be by using **EVENT**; however this will not make the movement smooth, just slower.

3.7.14. Screen scanning

The main Repton Infinity game works by 'scanning' the map, character-by-character, from the top-left down to the bottom-right. This scanning takes place about 8 times every second. In fact, two scans are made. The first looks for any characters that are ready to do something new and, finding any, executes the **ACTION** parts of their definitions. The second scan looks for any characters that are in the process of moving and, finding any, moves them on a bit more (according to their speed). The second scan is also responsible for dealing with collisions between characters and executes the **HITS** section of any character that is hit by another.

The reason why we are telling you all this becomes clear when you

look at a definition like:

```
NAME Pipe

DEFINE ACTION
LOOK(W)
IF NOT Fixed
  CREATE(CONTENTS,E)
  CREATE(Space,W)
ENDIF
```

This is in fact based on the definition of one of the pipes in "Robbo". Imagine a horizontal line of these, separated by spaces (like in "Robbo") with a non 'Fixed' character on the left of the leftmost pipe. When the first screen scan takes places, the leftmost pipe would move the character to its right, the next pipe would then be scanned and move the character again, as would the next pipe and so on. The final result is that the character is moved down the length of the whole pipe in one screen scan (an eight of a second)! The same effect happens for pipes 'pointing' down the screen. The way to get round this is to use **STATE**.

```
DEFINE ACTION
LOOK(W)
IF Fixed
  STATE(0)
ELSE
  FLIP
  IF STATE(0)
    CREATE(CONTENTS,E)
    CREATE(Space,W)
  ENDIF
ENDIF
```

This is the actual definition for a 'right' pipe, as in "Robbo". This problem only comes to light with objects that act on other objects. Characters that 'create themselves', like fungus in "Repton 3 - Take 2" are dealt with slightly differently. Take the definition for a very simple one-dimensional fungus:

```
DEFINE ACTION
LOOK(E)
IF FungusOK
  CREATE(Fungus,E)
ENDIF
```

You might think that a similar problem would occur, but there is a subtlety to **CREATE** - if a character is **CREATE**d it has an

(internal) flag set that means 'don't process this character until next go'. This prevents the proper fungus from filling the screen in a single screen scan!

3.7.15. Repton - a special case

Another point to note is that Repton is dealt with as a separate case in terms of screen scanning. In fact his movement is done before that of any other characters. This may help you to understand some features of Repton Infinity. For example, unlike Repton 3, in "Repton 3 - Take 2" you are allowed to move out from under a rock and then back again without the rock falling. This also explains why you can add to Repton's definition. Because Repton's movement and the pushing of objects is dealt with separately before anything else, Repton's **ACTION** section proper is vacant and so may be used like any other. If you include an **ACTION** section for Repton (like the door example above) this is executed along with all the other **ACTION** sections during the main screen scan.

3.7.16. Recursive programming

Believe it or not, Reptol actually allows definitions to be recursive (that is, for definitions to 'call themselves')! Take for example, the 'Magic Wall' in "Repton 4":

```
DEFINE HITS
  CREATE(Rock,S)
  CREATE(Crown)
```

Imagine two of these on top of each other, and a rock 'hits' the top one. This will then create a rock below it (on top of the other magic wall) and thus call the 'hit' routine for the magic wall again, creating another rock below that. Finally the recursion 'unwinds' and creates the crowns in the appropriate places. You occasionally have to be quite careful about such definitions - especially when **CREATE** is used with no direction. For example:

```
NAME Coal

DEFINE HITS
  SCORE(5)
  CREATE(Gold)
```

Imagine some Coal is hit by a rock: the hit routine will be executed, 5 points awarded, and then 'Gold' will be created ON THE SAME SQUARE. This may cause some confusion and so we recommend the

following rule of practice. If any character may be 'squashed' by more than one other type of character, its **HITS** section should use **IF HITBY** to distinguish between the sorts of things that may do the squashing. The above definition should really be written as:

```
NAME Coal

DEFINE HITS
  IF HITBY Rock
    SCORE(5)
    CREATE(Gold)
  ENDIF
```

Because of the limited 'stack' size, the recursion will eventually 'fall out' - i.e. another call could not be made because the stack is full. If this occurs, then the routine will return without doing anything. This can be seen if you pile up too many magic walls on top of each other. A rock falling in the top will change about 10 into crowns, but then will simply re-appear and not create the crown.

3.7.17. The envelopes

And finally, for curious readers, here are the envelopes used in Repton Infinity:

```
ENVELOPE 1,1,0,0,0,1,1,1,100,-8,-3,-3,100,30
ENVELOPE 2,2,1,0,0,10,0,0,100,-3,-3,-5,127,80
ENVELOPE 3,5,1,-1,0,1,1,1,100,0,0,-16,80,30
ENVELOPE 4,2,0,0,0,0,0,0,90,-1,-2,-3,90,30
```

4. Film Strip - the sprite editor

4.1. Introduction to Film Strip

Film Strip is the sprite and character editor of Repton Infinity. It allows you to edit not only the main sprites used by the game, but also the characters displayed on the game map and the so called 'editor characters' that appear in the boxes at the bottom of the editor screens. Because of this, there are rather a large number of boxes on the screen!

There are three main edit 'grids' occupying most of the screen. The largest (left) of these is the game sprite grid, the next largest (centre top) is the editor character grid, and the smallest (centre middle) is the map character grid. Below these grids is the standard character box. The currently selected character is enclosed by a small square. The right hand side contains (from top to bottom) the options menu, the four available colours, the currently selected colour, the 'actual size' map character and, in the bottom right hand corner, the 'actual size' game sprite. Note that there is no unique indication for the 'actual size' editor character - it is shown in the character box at the bottom of the screen.

As with the other editors, options are selected from the Film Strip menu using the cursor keys and confirmed with RETURN.

{ILLUSTRATION REQUIRED - annotated picture of Film Strip in use}

4.2. Editing sprites

On selecting **Edit** from the menu, a flashing edit cursor will appear in one of the edit grids. The cursor can be moved using either the cursor keys or Z,X,: and /. These keys not only control the cursor movement, but also select which grid to edit - the cursor can be moved off the side of a grid into the adjacent ones.

4.2.1. Simple editing

Pixels are set by pressing RETURN and deleted with DELETE. Both these keys can be held down while moving the cursor to produce or delete a line of dots. Note that the cursor cannot be moved off the side of a grid when 'trailing' in this way.

The current colour is indicated in a box on the right of screen and other colours can be selected using keys 0-3. Colour 0 can also be selected by pressing '4'. The four available colours can be varied using keys f0-f3. These colours are saved with the

  More...

sprites and will be restored when the sprite file is reloaded. However, these colours will not be used by anything except Film Strip - the actual colours used for the sprites in the game are defined using Land Scape.

As with the other editors, the character being edited is selected by holding down SHIFT and pressing the cursor keys. There is no need to indicate that changes have been made to a sprite: all your changes are 'stored' when another character is selected.

Pressing ESCAPE will return you the Film Strip menu.

4.2.2. Advanced facilities

In addition to these standard edit keys, there are a number of additional functions to ease the editing process. To copy a sprite to another position select the sprite and press COPY. Then select the destination and press 'R' to recall the copied sprite. This will copy all three grids across. To copy just the currently selected grid press 'G' instead of 'R'. Sprites can also be reflected horizontally or vertically using 'H' and 'V' respectively. Note that 'H','V','R' and COPY act on all three grids at once. The currently selected grid can be cleared to the currently selected colour by pressing CTRL 'C'.

It is possible to create a patterned background for your sprites like the one used in "Robbo". To do this, three functions have been built in to help you. To create the background, first design the pattern in the top left hand corner of the space character (sprite 0). The pattern must not extend outside the top 4x8 pixels. Once designed, pressing f4 will copy this pattern across the whole grid. Function keys f6 (underlay) and f7 (border) are used to place this pattern 'behind' the other sprites. First select a character that needs a background; then press f6 or f7. Key f6 copies the pattern into ANY black pixels in the select sprite while f7 leaves a black border around any set pixels - giving the game a 'cartoon' look. It is inevitable that once the background has been applied some sprites may need tidying up by hand.

Finally, if you make a serious mistake while editing a sprite, f5 will restore all three grids to their original states.

4.3. Animated sprites

Repton Infinity lets you define simple animated sprites. These can have two stages (or frames) of animation, like monsters and spirits in "Repton 3 - Take 2". There are two steps to creating an

animated sprite: the system flag 'Animate' must be included in the Blue Print definition file (see section 3.3.1, System flags) and the two frames of the sprite should be edited using Film Strip.

4.3.1. What animation means

If a sprite is animated, then the game will alternate between displaying that sprite and the one directly below it in the character box. The first four sprites cannot be animated, and the 14 'Repton' sprites have their own pre-defined animation sequences. Because none of the 'bottom row' sprites can be defined, the two sprites in the bottom left hand corner are reserved for animation frames only. "Repton 3 - Take 2" uses these for the Monster and the Spirit. In addition to these two there are 12 other possible pairs of animated sprites. These are the top and middle rows to the right of the first four 'special' sprites. In short, all sprites except the first four and last 14 may be used for animation and the second image of an animated sprite should be located immediately below the first as seen in the character box at the bottom of the screen. Try it out - it's the easiest way!

{ILLUSTRATION REQUIRED - diagram of which sprites can be animated}

4.3.2. Testing out animation

As an aid, Film Strip lets you test the animation of any sprite. Pressing 'A' will attempt to animate the current sprite: if it is one of the sixteen that cannot be animated you will hear an audible warning; if not, then the animation sequence for that sprite will be displayed. Because Film Strip does not know which are the animated sprites, it will always display the two frames. This will give strange effects if the chosen sprite is not animated! Remember - you must use Blue Print to specify that a sprite is to be animated.

4.3.3. Repton's animation

In addition, 'A' will also show Repton's walking and waiting movements. Repton has four-stage left and right movement, two stage up and down movement and three-stage 'waiting'. Try selecting various Repton frames and hold down 'A' to see this. It is possible to make the left and right animation sequence 'cycle' rather than 'oscillate', i.e. if you think of the four Repton frames as 0, 1, 2 and 3, then normally the game will display this sprites in the order 0123321001233210... etc. to give a walking effect. Instead these frames can be displayed in the order 012301230123... etc. to give a cycling effect, which is useful if Repton is not an 'animal'! This change is made by setting the

'Cycle' System bit in the Blue Print definition of Repton. As Film Strip does not know if Repton should be 'Cycled' or not, an additional key, 'C' is used to show Repton's walking movements in a cycle. It is used in exactly the same way as the 'A' key.

4.4. Loading and saving sprites

There are two sorts of files that Film Strip deals with. These are 'S.' prefixed sprite files and 'E.' prefixed editor character files. 'S.' files contain the sprite data for the main game sprites and also the map characters. 'E.' files contain the editor characters as displayed in the character box at the bottom of each editor screen.

The **Load** and **Save** options from the menu have similar formats, that is a sub-menu and a prompt for a filename. As usual, previous filenames are remembered and are offered by default. At any point while entering a filename, the sub-menu bar can be moved with the up and down cursor keys. This is used to select which type of file is to be dealt with. The prefix in front of the filename will also change as the bar is moved. The first option 'Sprite' saves or loads just the 'S.' file, while 'Chars' saves or loads just the 'E.' file. The 'Both' option saves or loads both the 'S.' and 'E.' files (in that order). It does not save anything different, it is just a short cut and is mainly for use on disc.

Electron users should recall that their files are prefixed with 'eS.' and 'eE.' to distinguish them from the BBC/Master files. The 'S.' and 'E.' files for our four games can be found on your DATA tape or disc.

4.5. Entering your name

As with the other editors, the **Name** option is used to enter a name that appears on the game title screen, in this case next to the 'Casting' credit.

4.6. Quit

As usual, Quit returns you to the main game menu.

4.7. Example sprite files

To get you going, the sprite files for all four of our games are supplied. Disc users will find them on the DATA disc; tape users on side B of the DATA tape. The filenames are:

  More...

Game	Sprite file
"Repton 3 - Take 2"	"S.Rep3"
"Repton 4"	"S.Rep4"
"Robbo"	"S.Robbo"
"Trakker"	"S.Trak"

Remember that Electron sprites files are prefixed with 'eS.' instead of 'S.'

5. The Linker

5.1. Introduction to the Linker

The Linker performs the all important task of taking screen, sprite and code files and joining them together in such a way that you can subsequently load them into the game and play them.

At this point, it is assumed that you are familiar with the three editors and currently have three data files that you want linked together. If not, please read sections 2, 3 and 4 to see how to create them.

Note for Master users

The Master Linker is slightly different from the BBC and Electron versions and as such will be dealt with separately, in section 5.3. Section 5.2 deals solely with the BBC and Electron versions, however Master users may like to read it as it contains many of the fundamental concepts of the Master version.

5.2. Using the BBC Linker

The Linker screen consists of six boxes in which text can be typed. The large box at the top of the screen (with 'The Repton Infinity File Linker' in) is the 'title box'. The four smaller boxes in the central area are the 'file boxes', those on the left being 'input' files and the one on the right the 'output' file. Finally, the large box at the bottom is the 'end message' box.

5.2.1. Entering information

The cursor can be moved between the boxes by pressing the up and down cursor keys. The RETURN key has exactly the same effect as the down cursor key. The left and right cursor keys and DELETE are used to edit text. Newly typed characters will be inserted into existing text rather than overtyping it.

When first loaded, the cursor will be positioned in the 'title box'. This is where the title of the game is entered. This is the name that appears beneath the 'Repton Infinity' logo on the highscore table of the game. To enter a title, simple type it in: it will automatically be centred when loaded into the game.

Pressing RETURN (or down cursor) will move the cursor down to the 'Land Scape' file box. Here you should enter the name of the screen file you wish to link. A further RETURN will move down to 'Blue Print'. Enter the name of the object code file and press

RETURN to move down to 'Film Strip'. Now enter the name of the sprite file and again, press RETURN. At this point, the cursor will jump across to the 'Game File' box on the right of the screen. Here you should enter the name you want the final linked file to be called.

Very often all these four filenames will be the same, or at least, very similar. If this is the case, then there is a short cut you can use. Say you wanted to enter the filename "Test1" into all four boxes, move the cursor to the 'Game file' box and enter "Test1" and press COPY. This will copy "Test1" into the three boxes on the left. These copied names can be edited in the usual way. Note that the COPY key cannot be pressed if the cursor is already in one of the boxes on the left.

The final piece of information the Linker requires is entered in the bottom box. This is the message that is displayed after Level 4 of your game has been completed. Usually this is something like

"Well done! - Now try G.Rep4A!"

As with the title, this text will automatically be centred when displayed by the game.

5.2.2. Linking the files

At this point, carefully check all the filenames and your spelling. If there is anything wrong, use the up and down cursors to move to the incorrect box and amend the error using the the left and right cursor keys and DELETE. When you are happy that all the information is correct, press CTRL 'L' to link the file. If you are using a single disc drive you will be prompted to insert the DATA disc. Tape users should insert their own data tape.

The three 'input' files will now be loaded. They are loaded in the order: 'Land Scape', 'Blue Print', 'Film Strip', i.e. the order of the boxes down the left. The filenames are highlighted as they load, along with messages saying what is happening. Once loaded, the files will be linked together, and the 'G.' game file saved, if all is well, the message 'File successfully linked' will be displayed. As always, Electron users should note that all their filenames will be preceded with 'e' to distinguish them from the BBC/Master files.

You can link another set of files, or quit the editor and return to the main menu by pressing ESCAPE.

5.3. Using the Master Linker

The Master Linker is slightly more sophisticated than the BBC and Electron version, but this is only a small point. The main difference is that you do not need to use the Linker to test out alterations to a game - all the components of your game (levels, object code and sprites) are always stored in sideways RAM where the main game can find them. If, for example, you are developing a screen and you find that one part is impossible:

- Press SHIFT ESCAPE to end your game and return to the highscore table.
- Press 'E' to quit the game and return to the main menu
- Select Land Scape and alter the screen.
- Quit Land Scape - no need to save the data
- Select Play Game and test out alteration.

The only thing you must do, is use the '**Make**' option from Blue Print if you change the definitions in any way. If you load a definitions file into Blue Print you must '**Make**' it or the old object code will be still be used.

IMPORTANT: do not attempt to play the game after unsuccessfully 'making' a definitions file - if an error occurs while 'making' your object code you must correct it and select '**Make**' again until successful.

So what is the Master Linker used for, you may ask! Well, you still need to enter game titles and end messages: this is done in the Linker. It also still does the task of generating 'G.' files that can be loaded straight into the game. This is because the 'data transfer' between the editors and the game is purely a one-way process. If it wasn't you'd be able to load in a 'G.' file into the game, press 'E' and then examine all the screens, definitions and sprites to help you cheat! When developing your game it is handy to swap between the editors and the main game so you can try out new ideas and puzzles quickly. When your game is complete though, save it as a 'G.' file for others to play.

5.3.1. Entering information

The Master Linker screen is much smaller than its BBC counterpart. It consists of only three boxes and a small menu. As with the BBC Linker, the top box is the 'title box' and the bottom box is for 'end messages'. The third box, on the left, is the all that's left of the four file boxes on the BBC version!

To create a 'G.' file, enter the required title and end message in

  More...

exactly the same way as the BBC version. The name of the 'G.' file should be entered in the file box. Ignore the fact that the box has a 'D.' prefix in it for now!

5.3.2. Linking the files

Now press CTRL 'L' as with the BBC version. Now, instead of immediately linking the file, the menu on the right will be 'activated'. The first two options are dealt with below so ignore them for the moment. Press the up or down cursor keys until the 'Make Game' option is highlighted and press RETURN to confirm. (Note that the file prefix has changed to a 'G'.) This will 'dim' the menu, perform the linking and then save the 'G.' file. The menu will then return. Press ESCAPE to return to editing the Linker boxes.

5.3.3. Extra facilities

The other two menu options, 'Save Data' and 'Load Data' are provided to save time while developing your own games. 'Save Data' will save all the data for all the editors in one file. In other words this file will contain:

- Level designs
- Textual definitions
- Object code
- Game sprites
- Map characters
- Editor sprites
- Game title
- End message
- Land scape 'score calculator' values
- Filenames.

This file can subsequently be loaded back in with the 'Load Data' option. This files are prefixed by 'D.' (for Data). Note that these files cannot be loaded into anything else except the Master Linker. If you didn't have the option of saving or loading all the data together you would have to select each editor in turn and load or save the component parts individually. Remember - you cannot load a 'G.' file into the game and then edit the screens, sprites or definitions.

As with the BBC version of the Linker, ESCAPE will leave the Linker and return you to the main game menu.

6. Reference

6.1. Reptol Command Summary

In the following breakdown, each Reptol keyword/command is given a 'Section' and a 'Type'. The 'Section' lists which sections of a character definition under which the keyword can appear. The 'Type' places the keyword in of four categories:

```
 Condition : testable condition - part of an IF statement
 Command   : general command - actually does something
 Effect    : special effect
 Structure : fundamental part of Reptol
```

CHANCE(p%)
Sections: ACTION
Type: Condition

Example:
```
IF CHANCE(10%)
  FLASH(YELLOW)
ENDIF
```

Evaluates to true if a (non-integer) random number between 0.00 and 99.99 is less than the percentage 'p' given in brackets.

CHANGE(x,y)
Sections: ACTION, HITS
Type: Command

Example:
```
CHANGE(Safe,Diamond)
```

Replaces all occurrences of character 'x' with character 'y'. Both 'x' and 'y' should be character names set with **NAME**. Characters 'x' and 'y' should move at the same speed (or not at all).

CONTENTS
Sections: ACTION
Type: Condition

Example:
```
LOOK(E)
IF CONTENTS Space
  MOVE(E)
ENDIF
```

Example:
```
CREATE(CONTENTS,E)
```

Has two forms. First, it can form part of an **IF** statement following a **LOOK**, in this case the body of the conditional is

  More...

executed if the last character **LOOK**ed at is the one named after the **CONTENTS**. Second, it can be part of a **CREATE** in which case the last character **LOOK**ed at is used rather than a named character.

CREATE(x,d)
Sections: ACTION, HITS
Type: Command

Example: CREATE(Fungus,NE)

Used to put characters on to the map. The first parameter is the sprite name of the character to create; the second is optional, but if present specifies the direction of the creation. If no direction is given, then the character is placed on top of the current square. The direction is specified in terms of compass points, so any of **N, NE, E, SE, S, SW, W & NW** can be used. North is taken to be the top of the screen.

DEFINE
Sections: n/a
Type: Structure

Used to introduce a new section into a definition. There are three forms:

(a) **DEFINE TYPE**
This section describes the character in terms of the system and user flags.
(b) **DEFINE ACTION**
This section describes how the character moves or what it does (for static characters).
(c) **DEFINE HITS**
This section describes what happens when other characters land on top of this character.

EASTOF
Sections: ACTION
Type: Condition

Example:
```
IF EASTOF
  FLASH(RED)
ENDIF
```

Evaluates to true if the current character is to the east of Repton.

  More...

EFFECT(x)
Sections: ACTION, HITS
Type: Effect

Generates a noise effect on channel 0. 'x' is a number between 0 and 255, although only some values produce unique effects. The bottom 3 bits (i.e. values of 0 to 7) are used as the pitch value and the top 2 bits describe the type of effect (or the 'envelope' number).

ELSE
Sections: ACTION, HITS
Type: Structure

Forms part of an **IF** structure. The instructions following the **ELSE** are executed if the **IF** evaluated to false. See **IF** for an example.

END
Sections: ACTION, HITS
Type: Structure

Example:
```
LOOK(NE)
IF CONTENTS Wall
  END
ENDIF
```

Prevents the rest of a definition being executed.

ENDIF
Sections: ACTION, HITS
Type: Structure

Forms part of an **IF** construct. There must be a matching **ENDIF** for each **IF**. See **IF** for an example.

EVENT(x)
Sections: ACTION
Type: Condition

Example:
```
IF EVENT(2)
  LOOK(S)
  IF CONTENTS Space
    MOVE(S)
  ENDIF
ENDIF
```

Used to delay execution of a definition. 'x' can vary between 0 and 7. The delay is proportional to two to power of the value of

  More...

'x'. Thus **EVENT(1)** means 'every other time' and **EVENT(7)** means 'every 128th time'. **EVENT** is only recommended for static characters or those moving at speed 'Four'.

FLASH(x)
Sections: ACTION, HITS
Type: Effect

Example: FLASH(WHITE)

Momentarily changes the background colour to 'x'. This can either be a number between 0 and 7, or a colour name. Colours available are: RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN and WHITE.

FLIP
Sections: ACTION
Type: Command

Inverts the **STATE** bit for the current character. See **STATE** for more details.

GOTO <name>
Sections: ACTION, HITS
Type: Structure

Example:
```
GOTO left
  ...
  ...
LABEL left
```

Jumps to the named **LABEL**, which can be in another definition and/or section. IMPORTANT: you must not jump to another section if it **CREATE**s objects.

HITBY <name>
Sections: HITS
Type: Condition

Example:
```
IF HITBY Rock
  SCORE(10)
ENDIF
```

Evaluates to true if the character has been hit by the given character.

IF <cond>
Sections: ACTION, HITS
Type: Structure

  More...

Example:
```
LOOK(N)
IF CONTENTS Space
  MOVE(N)
ELSE
  LOOK(S)
  IF NOT CONTENTS Space
    EFFECT(17)
  ENDIF
ENDIF
```

Start of an **IF...ELSE...ENDIF** construction. <cond> can either be a flag name or one of the testable conditions. If the result of <cond> is true, then the instructions following the **IF** are executed. Execution stops when either an **ELSE** or an **ENDIF** is found. The construct also allows an optional **ELSE** section. The instructions following the **ELSE** will be executed if the result of <cond> is false. **IF**s can be nested up to a level of 10. The **IF** can be followed by **NOT** in which case the truth value of <cond> is 'inverted', so the **IF** executes if <cond> is false.

KEY
Sections: ACTION
Type: Condition

Example:
```
IF KEY
  GOTO opendoor
ENDIF
```

Evaluates to 'true' if the RETURN key is being pressed on the keyboard.

KILLREPTON
Sections: ACTION, HITS
Type: Command

Example:
```
IF EVENT(7)
  KILLREPTON
ENDIF
```

Kills Repton wherever he is.

LABEL <name>
Sections: ACTION, HITS
Type: Structure

Places a label at that point in the definition. This can then be jumped to be **GOTO**. See **GOTO** for an example.

LOOK(d)
Sections: ACTION
Type: Command

Example:
```
LOOK(W)
IF NOT SolidToTrifle
  MOVE(W)
ENDIF
```

Looks in the specified direction and sets the user flags according to the contents of that square. These (and the contents of that square) can then be tested using an **IF**.

The direction can either be one of the compass points as in **CREATE**, or one of F, B, L, R. These stand for Forward, Backward, Left and Right. These allow directions relative to the current direction of movement to be used. It may be necessary to **LOOK** again after an **IF**, as the results of the **IF** are not retained throughout the body of the **IF**.

MOVE(d)
Sections: ACTION
Type: Command

Example:
```
MOVE(R)
```

Moves the current character in the direction specified. Only cardinal (N,E,S,W) or relative (F,L,R,B) directions are allowed. Only one MOVE is allowed per turn: if more than one is given, then only the first is obeyed.

MOVING
Sections: ACTION
Type: Condition

Example:
```
LOOK(S)
IF CONTENTS Repton
  IF MOVING
    KILLREPTON
  ENDIF
ENDIF
```

Evaluates to true if the current character is moving. The example above comes from a rock in "Repton 3 - Take 2".

  More...

NAME <name>
Sections: Top of definition
Type: Structure

Example: NAME Fungus

Gives a character a name. Characters don't have to be named, but if you wish to refer to them elsewhere using **IF CONTENTS**, **CREATE** or **CHANGE**, then a name is essential.

NORTHOF
Sections: ACTION
Type: Condition

Same as **EASTOF**, but true if north of Repton.

NOT
Sections: ACTION, HITS
Type: Structure

Optional part of **IF**. The **IF** will be executed if <cond> is false, otherwise the **ELSE** will be executed (if present). See **IF** for an example.

SCORE(s)
Sections: ACTION, HITS
Type: Effect

Example: SCORE(10)

Adds 's' to the players score. 's' must be in the range 0 to 255.

SOUND(s)
Sections: ACTION, HITS
Type: Effect

Example: SOUND(96)

Generates a sound on channel 1. 's' must be in the range 0 to 255. The top 2 bits of 's' are used to describe the type of sound being generated (i.e. the envelope used). The bottom six bits contain the pitch where 0 is the lowest and 63 the highest.

SOUTHOF
Sections: ACTION
Type: Condition

Same as **EASTOF**, but true if south of Repton.

  More...

STATE(s)
Sections: ACTION
Type: Condition/Command

Example:
```
FLIP
IF STATE(0)
  LOOK(N)
  IF CONTENTS Space
    MOVE(N)
  ENDIF
ELSE
  FLASH(YELLOW)
ENDIF
```

Example:
```
IF NORTHOF
  STATE(1)
ENDIF
```

Can be used either as part of an **IF** to test for a specific state, or by itself to explicitly set the state. There are two states that a character can be in: 0 and 1. All characters start off in state 0. The **STATE** command can be used to vary what a character does. When used in conjuction with **FLIP**, a character can be made to alternate between two actions.

WESTOF
Sections: ACTION
Type: Condition

Same as **EASTOF**, but true if west of Repton.

6.2. Summary of system flags

Flag	Meaning...
Solid	Solid to Repton.
Deadly	Kills Repton.
HPush	Can be pushed horizontally.
VPush	Can be pushed vertically.
Under	Other characters should 'move under' this one.
Transport	Can use transporters.
Squash	Can be squashed by other characters.
Cycle	Set if Repton should cycle instead of walk, or transporters should 'regenerate'.
Animate	This sprite is animated.
One	Slowest speed - same as Repton.
Two	Twice as fast as Repton.
Four	Full speed.

6.3. Blue Print error messages

Whenever an error is caused in Blue Print, the error message will be displayed in the message box at the top of the screen and the cursor will be positioned on the line in the definition that caused the error. All possible error messages are listed in alphabetical order below:

Aborted - ESCAPE was pressed when requested to insert GAME or DATA disc.

Bad % - CHANCE parameter was not in the range 0.01% to 99.99%

Bad EVENT - EVENT parameter was not in the range 1 to 7.

Bad MOVE direction - You have tried to MOVE in a direction other than N, E, S, W, F, L, R or B.

Bad STATE - STATE parameter was not 0 or 1.

Bad definition - You have followed the keyword DEFINE with something other than TYPE, ACTION or HITS.

Bad direction - Direction parameter to LOOK was not a compass direction (N, NE, E, SE, S, SW, W, NE) or a relative direction (F, L, R, B) or the optional direction in CREATE was not a compass direction.

Bad name - Text after the NAME command was missing or invalid.

Bad numeric parameter - You have used an invalid number somewhere. Numeric parameters can only consist of the characters '0' to '9'.

Can't animate this sprite - System flag 'Animate' cannot be used in the first four sprites or the last 14 sprites on the second row. You have attempted to do so.

Can't make small enough - Blue Print could not shorten the object code sufficiently to fit it into the game. You must shorten or remove some definitions.

Line full - You have attempted to insert a character in a line that is already full.

Missing % - CHANCE parameter was not followed by a percentage (%) sign.

Missing) - You have omitted a close bracket.

  More...

Missing , - You have missed out a comma in a CHANGE command.

Mistake - Invalid text was found, probably at the end of an otherwise valid line.

Name already used - You have tried to give two sprites the same name.

No ENDIF - There are not enough ENDIFs to match all the IFs.

No IF - An ENDIF was found that couldn't be matched to an IF.

No more - You have attempted to scroll beyond the top or bottom of the current definition.

No room - This is a serious error! You have filled the memory available for definitions. You must shorten or delete definitions before entering anything new.

No such flag - Reference to a non-existent user flag was made.

No such label - You have tried to GOTO a label that does not exist.

No such sprite - Reference to a non-existent sprite was made.

Number too big - A number greater than 255 has been used where it should not.

Syntax error - You have attempted to define a flag in a section other than TYPE.

Too many IFs - More than 8 nested IFs have been used.

Too many flags - More than 8 user flags have been defined.

Type mismatch - You have mixed 'types' of variable by attempting to GOTO a sprite name, CREATE a label or similar.

Wrong section - You have used a keyword in a section where it cannot be used.

