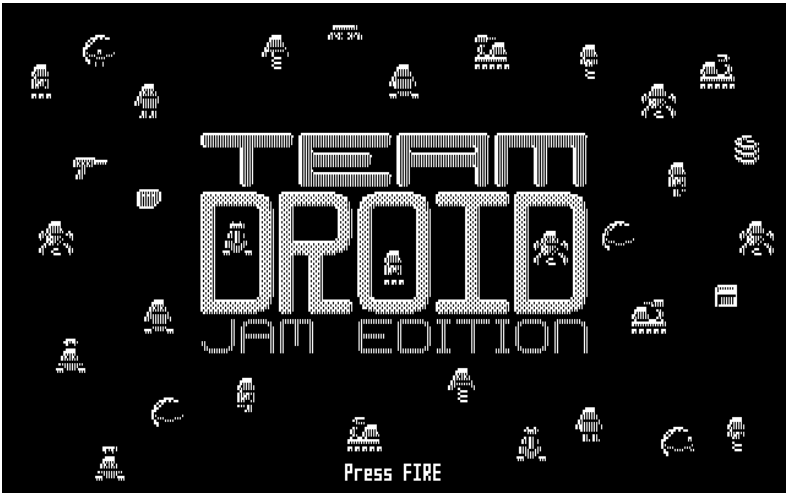

TEAM
DROID
JAM EDITION

Contents

Chapter 1

Introduction



In the far future, an industrial complex is staffed only by robots. This is more economical than employing humans, but without supervision can lead to catastrophic results. Just such a catastrophe has happened,

and you have been employed on a temporary basis to clean up the mess. The data cards needed to keep the facility running have been scattered all over the various levels, and need putting back in the data card readers.

It's dangerous for humans to enter the facility, so you have to do your job remotely. You do this with the aid of a team of specialised robots. On each level you need to choose a robot, sometimes more robots, who are best equipped to do the job. Once the robots are deployed, you instruct them to pick up the data card and take it to the reader.

That sounds easy, but there are obstacles. Contraptions like turntables, conveyor belts and teleporters may help or hinder your task. Security forcefields might block your way and need to be deactivated or otherwise bypassed. And some levels have guard robots which patrol the area; these are unaware of your important task and might destroy you if you are not careful.

Team Droid is a turn-based puzzle game that rewards careful thinking rather than fast reactions. You must examine each level to see which robots are most suited to it, and then instruct those robots turn by turn on exactly what to do. There is a library of actions you can choose from, but because of the state of chaos, not all of the actions are available each turn. So you will have to think ahead and improvise.

1.1 Installing the Game

The game runs on an IBM PC or compatible computer. It requires an 8088 processor or better, so it should run on anything from the original IBM PC to its faster clones. It supports CGA graphics and looks best when connected to an RGB colour or composite monochrome monitor. It requires 512k of memory.

Before playing the game you must install it either to another floppy disk or to a hard disk. This is because the program saves your progress, and the original disk is write-protected so your games could not be saved there.

To create a floppy disk to play the game from, have a formatted disk ready. Put the original game disk in your floppy disk drive (it is assumed your drive is drive **a:**) and type the following command:

```
A> diskcopy a: a:
```

DOS will copy all the files from the original disk to your destination disk, prompting you to swap disks as and when necessary. If you want to copy the game to your hard disk drive, you should insert the original game disk into your floppy disk drive as before, but issue the following commands instead:

```
C> mkdir \tdroid
C> copy a:\*.* \tdroid
```

Once installed, you can put the original disk away in a safe place and run the game.

1.2 Starting the Game

If you installed your game to a floppy disk, insert the disk into your disk drive (assumed in this example to be **a:**), and type the following command:

```
A> tdroid
```

If you installed the game to your hard disk, then you need to use the following commands instead (this example assumes that you installed to **c:\barren** as above):

```
C> cd \tdroid
C> tdroid
```

You will then see the *Cyningstan* logo and the game will start.

Team Droid chooses what is hoped to be an attractive colour palette for the graphics. Some monitors might not display this palette as well

as intended. If this is the case with your monitor, then you have two options. Firstly, you can play with the standard CGA palette of black, cyan, magenta and white by running the game with the **-p** option as follows:

```
A> tdroid -p
```

Alternatively, you can run the game in monochrome mode. This is obviously a good idea of you have a monochrome display, and can be done with the **-m** option as follows:

```
A> tdroid -m
```

Team Droid plays sound effects throughout the game. If these offend you, or you are playing in an environment where sound would be unwelcome, then you can use the **-q** option as follows:

```
A> tdroid -q
```

This option can be combined with **-p** or **-m** as described above. To get the maximum enjoyment out of the game, though, it is recommended to run it without any of these options and experience the game as it was intended.

Chapter 2

Starting a Game

Figure 2.1: The New Game screen.

The next few sections will introduce you to the controls, and help you

find your way around the various screens of the game. The later chapters will describe in detail the robots, items, actions and map cells that you will encounter as you play.

2.1 The Controls

Before navigating your way around the game you will need to know the game's controls. The game can be controlled entirely with the cursor movement keys ($\leftarrow \rightarrow \uparrow \downarrow$) and the *fire* key, which is your choice of *Ctrl*, *Space* or *Enter*.

On loading *Team Droid* for the first time, you will see the title screen. Once the "Press Fire" message appears, use one of the *fire* keys above to advance. You will then see the *New Game* screen. The large window on the right is where the game settings are shown. The small window at the bottom left is where a menu will appear when needed. The rest of the panel is decorative.

On the *New Game* screen, the $\uparrow$ and $\downarrow$ controls move the higlighted bar that you can see in the top left window, allowing you access to the different settings. The $\leftarrow$ and $\rightarrow$ controls will cycle through the available options for the current setting, or in the case of the *Player* option, allow you to edit your name.

The *fire* control brings up the menu, which will appear in the bottom left window. You need to hold this down while navigating the menu using the $\uparrow$ and $\downarrow$ controls, and release it when the option you want to use is highlighted. Some menus have more than four options, and allow you to scroll up and down past those you can see. If you can't see the *Exit Game* option, then the menu has further options to which you can scroll.

Every menu starts with a *Cancel menu* option that does nothing; it's there for those occasions when you bring up the menu by mistake. All menus also have an *Exit game* option that allows you to quit to DOS at any point. Your current state is saved, including any game in progress, and the program will return to this exact point when you run it again. All

menus other than the one on the *New Game* screen have a *New Game* option which saves your current game and brings you to this screen.

The program tries to be intelligent when it brings up the menu, highlighting the choice that it thinks you will want at this point. This allows you to choose the most common option by just tapping *fire*. On the *New Game* screen the default option is *Start Game*, so on this screen you can just tap *fire* to accept the options shown and start your game.

2.2 Entering Your Name

The first thing you will want to do is change the player name from *Cyningstan* to your own name. This name will be entered on a score table when you've finished all the levels. You cannot change the name used in a game after the game has started.

To enter your name, move the cursor bar down to the *Player* option and start typing. Your name will replace the one that is already there. Press *Enter* when you're done. This is the only time in the game that you will need to use keys other than the directional controls and *fire*.

As mentioned before, you can also access this option with the $\leftarrow$ and $\rightarrow$ controls. When you do so, the existing name is not wiped, but you can edit it. The editing controls are rudimentary, just the *Backspace* and *Enter* are available.

2.3 On to Play

When you have told the program who you are, start the game by selecting the default *Start Game* option from the menu. As mentioned already, just tapping *fire* will do this.

The first screen you will see is the list of twelve best scores, which on your first play will be empty. In *Team Droid* your score is the number of turns you take to complete all the levels, so lower scores are better. The score table remembers the number of turns you took on each level.

Figure 2.2: The Best Score screen after one game has already been played through.

There's not much of interest on the score table, so tap *fire* to bring up the menu and select the default option, which is *Proceed*. This will take you to the *Robot Deployment* screen, where you can view the first level and choose a robot to complete it.

Chapter 3

Deploying Robots

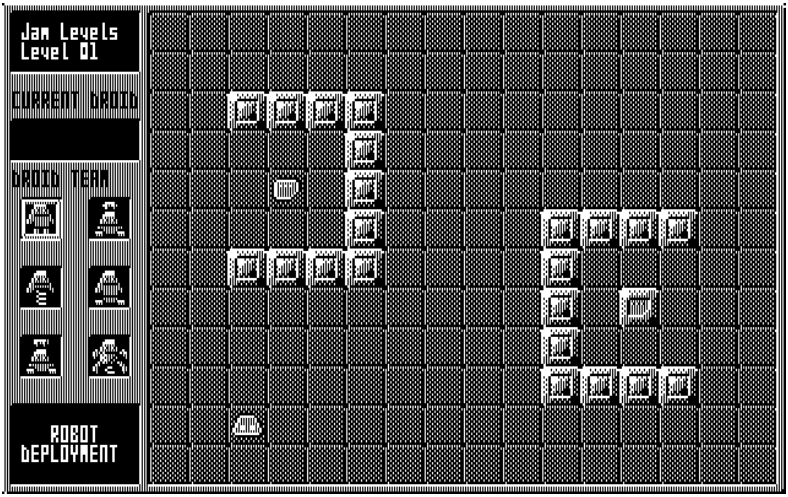


Figure 3.1: The Deployment screen, ready to deploy a robot.

The “team” in *Team Droid* consists of six robots, all with different capa-

bilities. In the *Robot Deployment* screen, you get to choose which of them will complete the current task. You get to choose a different robot (or sometimes, robots) depending on which you think will complete the levels more quickly.

The robots all have their strengths and weaknesses. Some are easier to control, others better at shooting or jumping over obstacles and dangers, or carrying objects around. The *Robot Reference* chapter will give their capabilities in detail later, but one of the robots will be introduced first, so we can get to playing.

Use the directional keys to browse the team on the deployment panel at the left of the screen. If you tap *fire* to choose the default *Select robot* menu option, you'll see that robot's name and sprite at the top of the panel. To start off with, select *Strider*, the one at the top left of the team. While *Strider* has its weeknesses, it is the most straightforward to control and move around.

Once *Strider* is the selected robot, move the cursor right and it will slide straight on to the map. The place(s) you can put a robot are marked with a little robot head symbol. In most of the levels supplied with the game there is only one place you can deploy the robot. Move the cursor to the robot head symbol and press *fire* to choose the default *Deploy robot* option. The robot head symbol will be replaced with the selected robot.

You'll notice when the robot appears that it is side on. The facing of the robot is significant when it comes to perform actions. Robots can face in the four cardinal directions, and the initial facing of the robot will be towards the centre of the level. This, as in the first level here, is not always helpful.

Now that the *Strider* robot is selected, it's time to start programming! Hold *fire* and use $\downarrow$ to select the *Proceed* option on the menu.

Chapter 4

Programming Robots

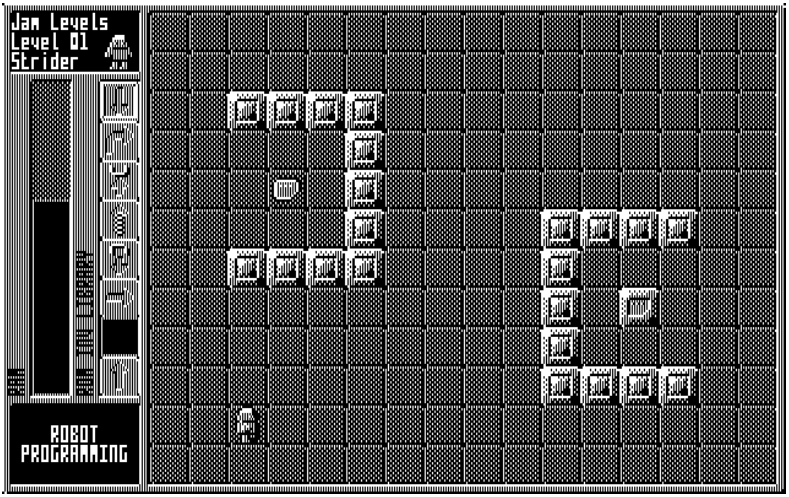


Figure 4.1: The Programming screen, with the robot ready to accept instructions.

On proceeding from the *Deployment Screen* you will see the *Programming Screen*. The team will disappear and instead the panel will focus on the robot selected for deployment. Let's take a bit of time to look around the panel.

Taking up the left-hand side of the panel is the robot's *RAM*. This is where instructions are built up for the present turn, from the bottom upwards as indicated by the direction of the text. Different robots have different amounts of RAM. *Strider* has 5 slots. The upper 3 slots of the RAM display are shaded to show that these RAM slots are unavailable.

At the top of the right hand column of the panel are six *Action* tiles, representing the actions available to the robot this turn. There are twelve actions in the *Library* in all, but only six are available each turn. At the end of a turn, the bottom action will drop off and another action will become available at the top.

The library is ordered differently each game, but the orders is not changed during the course of the game. So if you pay attention on the order in which new actions appear, you can start to predict when new actions are due to reappear and plan your programming accordingly.

Beneath the library is the *Inventory* slot. Most robots have an inventory slot allowing them to pick up and carry one item at a time. For a robot that has no inventory, this slot will be shaded in the same way as the upper RAM slots to show that it is unavailble. The inventory is always empty when starting a level.

Some robots have a single-slot *ROM*. Where present, the ROM contains a single action tile. While the library is rotated and the instructions available there vary from turn to turn, a robot's ROM is constant. *Strider* has the *Step Forward* instruction always available, making it easy to move around. For robots that have no ROM, the ROM slot will be shaded to show that it is unavailable.

One thing to note about the action tiles, especially those with arrows. The arrows do not indicate the cardinal directions of north, east, south and west. They indicate forward, backward, left and right, in relation to the current facing of the robot. Because *Strider* is facing east, the *Step Forward* action will make *Strider* move east, not north. Always keep this in mind when programming the robot: it's easy even once you

know this to instinctively misuse the action that you think points in the cardinal direction you want to move.

4.1 Adding Actions to the RAM

The way to program the robot is to select instructions and add them to the RAM. You do this by moving to an instruction in the library or the ROM, and tapping *fire* to select the *Add action* option from the menu. As mentioned already, the actions are read from the bottom up, so the first action you place will appear at the bottom of RAM.

Because strider is facing east, the *Step Forward* instruction isn't particularly helpful; it will take the robot away from the data card that you need to move. But there are other actions you can take, one of which is hopefully available in the library right now.

The *Turn Left* instruction will turn *Strider* 90°to the left, so that it faces north. If you have a *Turn Left* action available, add it to the RAM, and then add a few *Step Forward* instructions that will cause *Strider* to move a few cells to the north.

Unlike other games, where controls you press cause the immediate movement of your character, in *Team Droid* you're programming your robot for the turn ahead. The robot will not perform the selected actions until you choose the *Go!* menu option, as described later in this chapter (don't choose it yet). You will have to visualise where the robot will be as you add each instruction. This is the basic challenge of the game.

If you have no *Turn Left* action available, what are you to do? There are a number of other options that you can choose. Anyone with any knowledge of geometry knows that rotating 90°to the left is the same as rotating 270°to the right. So three consecutive *Turn Right* instructions will also turn *Strider* north.

Without any 90°rotational actions available, you're still not stuck. If a *Step Backward* action is available, then you can move *Strider* in reverse. This won't take *Strider* any closer to the card, but it will position the robot to avoid the wall that stands between it and the card. This will be useful for when *Strider* is finally available to move to the north.

One of *Strider*'s special capabilities is lateral movement, as it has the *Walker* item built in. The *Walker* is an item described in the *Item Reference* chpater, but in short it gives a robot the ability to move sideways in relation to its current facing. While *Strider* is facing east, the *Step Left* instruction will move it north, towards the data card, while *Strider* continues to face east. If you have both *Step Backward* and *Step Left* available, your best series of actions would be to *Step Backward* once and *Step Left* several times, moving you out of the way of the wall and several steps towards the data card.

One final action that could be useful if you don't have any of the above is the *Turn About* action, shown by a tile with an arrow turning back on itself. This does what the name suggests: applying it to *Strider* right now will make the robot turn about and face west, in which case a subsequent *Step Forward* action would move west out of the way of the wall, and *Step Right* actions would move north taking *Strider* towards the data card while continuing to face west.

These seven actions are all the actions that would be useful to *Strider* at this point. At least one of them will be present in the library at a time, so there is certainly something you can do this turn to help *Strider* approach the data card.

Because your score is measured in turns taken, it is always best to squeeze as many useful actions into a turn as possible. If you're unlucky, only *Turn about* is available among the options described, and you'll be limited to two actions *Turn About* then *Step Forward* to move *Strider* west a bit in order to avoid the wall to the north. A luckier start will allow you to fill up the RAM with instructions that will take *Strider* further towards its goal.

4.2 Go!

Once you've done all you can in adding useful actions to the robot's RAM, it's time to execute your program. Do this by holding *fire* to bring up the menu, pressing ↓ till the *Go!* option is highlighted, and releasing *fire* to choose it. This will take you to the *Action* screen: prepare to

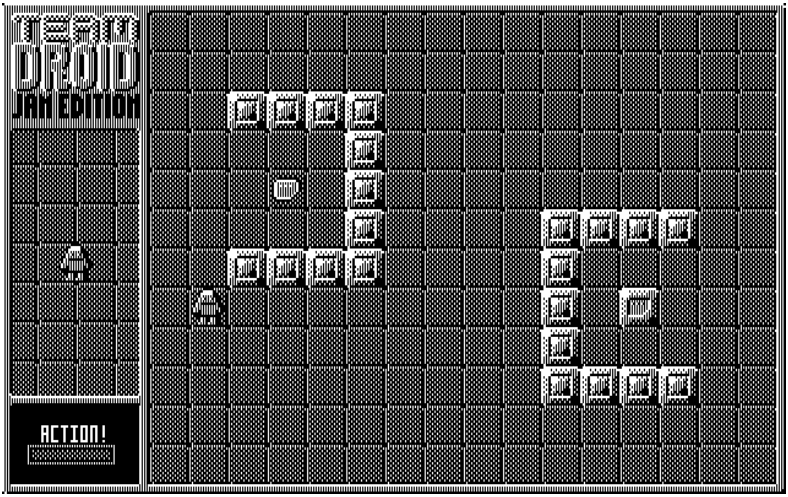


Figure 4.2: The Action screen after the robot has moved.

watch carefully as the action starts immediately.

The *Action* screen has virtually nothing on the panel apart from a space for the screen title, progress bar and the menu. On the right, you will watch as *Strider* performs the actions you gave it, which might differ from the actions you *thought* you gave it. Once the progress bar is full, all actions have been performed and you can continue to the next turn. Do this by tapping *fire* to accept the default *Proceed* menu option.

4.3 Obtaining the Data Card

In your next turn, you will notice the library tiles have all shifted down by one. The tile that was at the bottom will be gone, to return in six turns. A new action has appeared at the top, which may or may not be useful. As with the first turn, continue to add actions to the RAM that will guide *Strider* towards the data card. If none of the actions you

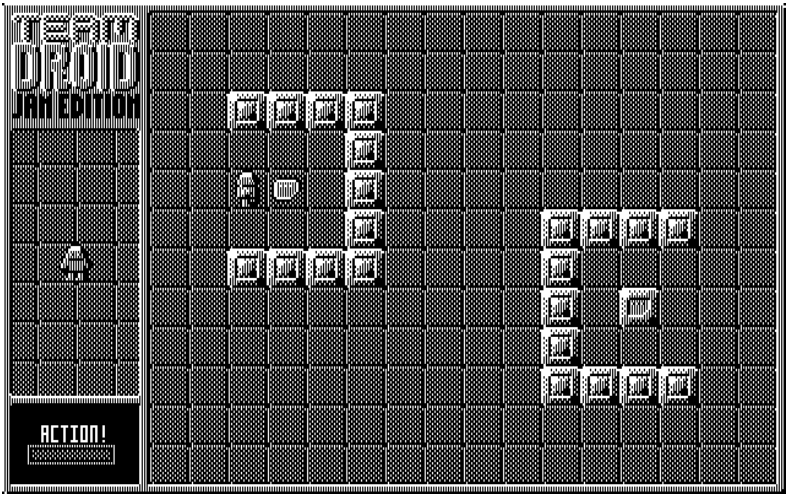


Figure 4.3: Strider is ready to pick up the data card.

need are available, you can miss a turn by choosing *Go!* from the menu without having added any instructions to the RAM.

Once *Strider* is adjacent to the data card, and facing it, you can instruct the robot to pick up the card. This is done with the *Take item* action, whose tile looks like a container with an arrow pointing up towards it.

Of course the *Take item* action may not be available when you arrive at the data card. You have two ways to deal with this. One is to miss turns, as described above, until the action becomes available. This might be the best action if you've figured out the order of the library and you know that *Take item* is going to appear in the next turn or two.

If *Take item* has only just dropped off the list of available actions, you know there's a while to wait before it comes back again. In this case, you can take advantage of the *pushing* mechanic. You can move any item around the level by getting behind it, facing it and pushing it using the *Step Forward* instruction. Robots other than *Strider* that have

the *Sprint* instruction available can also use that. The robots in the team are not strong enough to push items backwards or to the side: you must be moving forwards to push an item. Using the pushing mechanic you can start to edge the data card towards the reader, making some progress until the *Take item* action comes available.

4.4 Inserting the Data Card

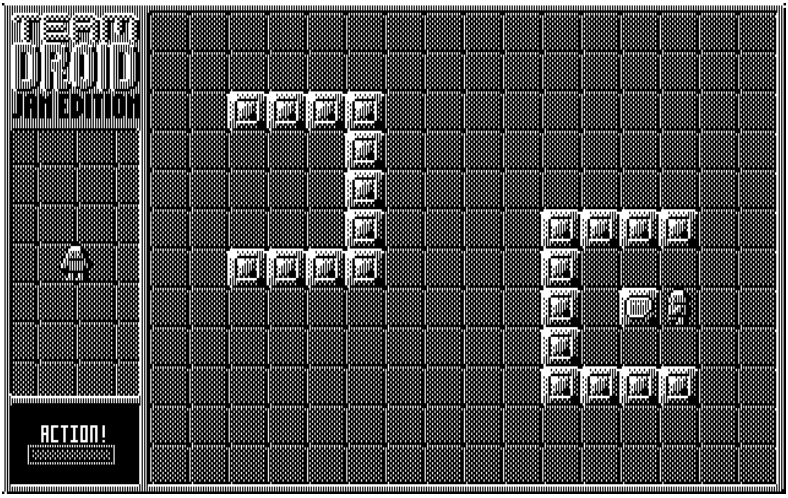


Figure 4.4: Strider has put the data card on the card reader.

All that's left to do now is to push or carry the data card to the *Card Reader* at the bottom right of the screen. Once you are adjacent to the card reader and facing towards it, the *Drop item* action will place the card on the reader. Once done, and you choose *Proceed* from the menu, the level is completed and you can move on to the next one.

Sometimes you'll find you have some distance to go before you reach the card reader, and the *Drop item* action might be about to drop

off the bottom of the available actions list. If this happens, you might elect to drop the card early, and push it the rest of the way to the card reader. Items can be pushed onto the card reader to complete the level. In fact, any robot without an inventory can *only* complete the level this way.

4.5 Moving Actions in RAM

Before leaving the subject of programming, let's look at some other programming options that you might find useful. If you've added an action by mistake, you can remove it from the RAM with the *Remove action* option on the menu. If you've added an action in the wrong order, rather than remove it, you can reorder the instructions with the *Move earlier* and *Move later*.

Notice that you can rearrange actions in the RAM so as to leave empty spaces in the middle. This is deliberate. The *Levels and Level Packs* and *Cell Reference* chapters introduce levels with more interesting elements, and sometimes you'll want to pause in your turn to allow their various effects to help or hinder your task.

Chapter 5

Completing a Level

Figure 5.1: Our current score compared to a previously finished game.

Having dropped the data card on the card reader, choosing *Proceed*

from the *Action* screen takes you to the *Level Complete* screen instead of the *Programming* screen. The *Level Complete* screen consists of a partial score table, showing the number of turns taken in all levels up to the one just completed. The total at the right is the total of turns for all those levels.

Once there are completed games in the score table, these will appear here too. For those games also, the scores will only appear up to the current level, along with totals up to that point. This allows you to compare your performance with previous games and see if you are getting better or worse.

After reviewing these scores, choose the *Proceed* option from the menu to take you to back to the *Robot Deployment* screen ready to start the next level.

5.1 Ways to Fail

It is very difficult to fail at the first level, although technically possible. If you choose *Soldier* or *Multibot*, for example, and manage to shoot the data card, then your level attempt will be swifly concluded. Instead of seeing the score table, you'll be taken back to the *Robot Deployment* screen to try the level again.

Sometimes you can get yourself into a position where the level is impossible to complete, even though the data card is still there. This is also difficult to achieve on the first level, but again it is possible. For instance, if you select *Multibot* as your chosen robot, which doesn't have an inventory, and push the data card into a corner where you can't push it out again. Or sometimes you've made so many mistakes you just want to try again to get a better score. In both instances, you can choose the *Reset level* option from the menu on the *Robot Programming* screen, and you'll be taken back to the *Robot Deployment* screen to try again. The turns wasted on this level will be discounted and you'll start again as if they never happened.

5.2 Winning the Game

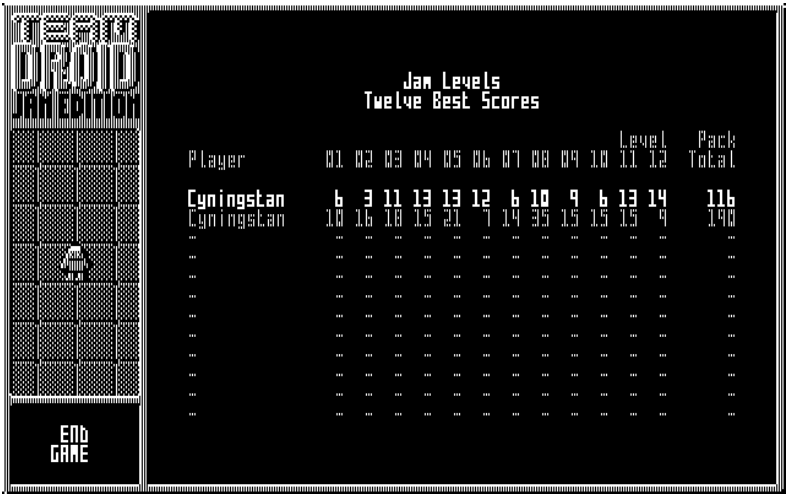


Figure 5.2: The score table and the end of a game.

There are twelve levels in a level pack, and on completion of level 12 you will win the game. You will then see the score table, with your position highlighted. This time, all the levels are shown, and your score will be saved permanently into the score table of the level pack. Your aim in future sessions will be to beat this total score.

Chapter 6

Levels and Level Packs

The first of the Jam levels is an easy introduction to the game controls, and the process of programming the robots to perform actions. The levels ahead introduce various elements that help or hinder the robots. These elements are often used in combination, requiring thought to navigate and to avoid or exploit.

6.1 What to Expect of Later Levels

The easiest elements to understand are the various new map cells that will appear. These are explained in full in the *Cell Reference* chapter. But to summarise, there are cells like conveyor belts, turntables and teleporters, which move and rotate the robots in various directions which may or may not be helpful. There are forcefields that destroy robots and items on contact, although some of these will have generators that can be shot at to deactivate the forcefield.

Some levels will have items to collect in addition to the data card, which might give a robot a capability that it currently lacks. But since there is only one inventory space, these items will have to be dropped if the inventory card is to be picked up. Or the items can be retained if

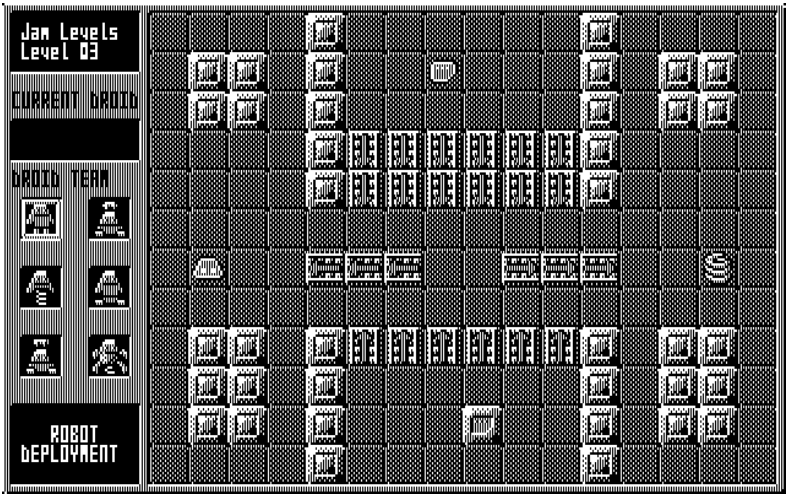


Figure 6.1: One of the later levels, with obstacles and items.

you choose to push the data card to its destination.

Another element that will appear is *Guard* robots. Contact with these robots is fatal, and often they will shoot phaser bolts which will destroy you if you are in their path. Each guard robot performs the same actions turn after turn, though, so their movements become predictable.

All the above can be combined together in such a way that certain robots are obviously better at navigating the levels than others. A few levels can only be done by a particular robot, so careful thought is needed at the deployment stage. And some levels will allow you to place more than one robot; in such levels the robots will have to work together to deliver the data card to the reader.

6.2 Future Expansion with Level Packs

One level pack is supplied with the game: *Jam Levels*. These levels are intended to introduce the various elements of the game gradually, as well as supplying entertaining puzzles to solve. But future level packs can be added for more gameplay.

Back on the *New Game* screen, the third of the game settings allows you to select a new level pack. When more than one level pack is installed, the $\leftarrow$ and $\rightarrow$ controls allow other level packs to be selected. When new level packs become available, their documentation will explain how to install them.

Although all the elements of the game are introduced in the *Jam Levels*, some combinations of them have not been explored. Only one level features multiple robots, and this is a feature that can be explored further in future level packs.

One idea that has not been explored at all is that of multiple data cards and readers. IF there are multiple data card readers, they must all have cards inserted to complete the level. Such levels might have multiple robots to achieve the tasks together, in concert or in isolation. Or one robot might be expected to insert all the data cards in sequence.

Chapter 7

Robot Reference

There are six robots in your team, all with different capabilities and limitations. You will need to know them well in order to deploy the correct one to solve a given puzzle. This chapter introduces all of the robots, and discusses their characteristics.

7.1 Strider

Robot:	*Strider*
RAM capacity:	*5*
ROM content:	*Step Forward*
Inventory:	*Yes*
Built-in:	*Walker*

Introduced in the tutorial, *Strider* is the easiest of the robots to move around, having the *Step Forward* action built into the ROM, and a built-in *Walker* item to allow lateral movement. Some levels, especially the more maze-like ones, might favour a robot that can move laterally.

One disadvantage is the relatively small RAM, allowing only five actions per turn. Another is that the *Walker* inhibits the use of the

Sprint action; since the *Walker* is permanently built-in, *Strider* may never sprint.

7.2 Bouncer

Robot:	*Bouncer*
RAM capacity:	*6*
ROM content:	*Leap*
Inventory:	*Yes*
Built-in:	*Spring*

A very excited robot, *Bouncer*'s main method of travel is to leap around. *Bouncer* has the *Spring* item built in. This gives it the ability to perform the *Leap* action without having to obtain and hold the *Spring* in its inventory. *Bouncer* also has the *Leap* action in its ROM, so that action is always available. There are some levels where the ability to leap is essential, either by collecting a *Spring* or having one built in.

While *Bouncer*'s ability to leap may mean fast movement and ways to avoid some obstacles, the *Leap* action cannot be used to push items around. Trying to leap onto an item will not work; the action will fail. Coupled with the lack of lateral movement when the *Walker* is not being carried, this means that *Bouncer* is not as manoeuvrable as *Strider*, even though it can bounce quite far in straight line.

7.3 Soldier

Robot:	*Soldier*
RAM capacity:	*6*
ROM content:	*Shoot*
Inventory:	*Yes*
Built-in:	*Phaser*

Equipped with a phaser, and containing the *Shoot* action in its rom, *Soldier* is always ready for battle. There are some levels which require

shooting either Guard robots, or forcefield generators. *Soldier*'s ability to shoot at will without having to pick up and carry the *Phaser* item, and without having to wait for the *Shoot* action to appear in the library, put it at a distinct advantage.

While *Soldier* is very useful for shooting things, it lacks any specific movement ability, and so it is at the mercy of the library for getting around. If a level has a *Phaser* available to pick up, especially if the layout has any maze-like qualities, then sometimes another robot might be a better fit to the task.

7.4 Carrier

Robot:	*Carrier*
RAM capacity:	*7*
ROM content:	*Take item*
Inventory:	*Yes*
Built-in:	*-*

Sometimes robots are waiting around for a *Take item* action so they can pick up the data card or another essential item. *Carrier* doesn't have this restriction, as *Take item* is built into its ROM. It also has 7 RAM slots, second only to *Thinker*.

Carrier doesn't have a *Drop item* action built in, so it might still have to wait around to drop an item it has picked up. It also has no specific movement ability, so it is at the mercy of the library in its attempts to move around.

7.5 Thinker

Robot:	*Thinker*
RAM capacity:	*8*
ROM content:	*-*
Inventory:	*Yes*
Built-in:	*-*

The main characteristic of *Thinker* is its large amount of RAM. At 8 slots, it is the biggest RAM of all the robots. This allows the player to pack a lot of actions into a single turn.
Thinker has no ROM for instructions, so it can only perform any action when that action is available in the library. It has no special attachments, so actions like leaping, shooting or moving laterally require the appropriate item to be carried in the inventory.

7.6 Multibot

Robot:	*Multibot*
RAM capacity:	*4*
ROM content:	*-*
Inventory:	*No*
Built-in:	*Walker, Spring, Phaser*

At first sight, *Multibot* might appear to be the most powerful robot. Its built-in *Walker*, *Spring* and *Phaser* give it the abilities of *Strider*, *Bouncer* and *Soldier* in one. There are some levels that will require these abilities and *Multibot* is the robot for them.

But coupled with these abilities are a couple of serious disadvantages. *Multibot* has only 4 slots of RAM, so it can only achieve a limited amount of progress each turn. It has the disadvantage of the built-in *Walker*, inhibiting the use of the *Sprint* action, further slowing it down. But the main disadvantage is that it has no inventory. So if *Multibot* has the task of taking the data card to the reader, it can only do so by pushing.

7.7 Guards

A seventh type of robot inhabits some of the levels, and they are definitely not on your team. The *Guard* robots patrol each level and will be a danger to your own robots. Sometimes these robots shoot, and even if not, contact with them is deadly.

Robot:	*Guard*
RAM capacity:	*8*
ROM content:	*-*
Inventory:	*Yes*
Built-in:	*Walker, Spring, Phaser*

Guards are not reliant on the library, as each guard is hard-wired with its own set of up to eight actions. They repeat these actions turn after turn, so if you watch them carefully then their behaviour can be predicted. Sometimes guards will need to be shot to progress, but in other levels they might be carefully avoided.

Chapter 8

Action Reference

The game features twelve different actions that you can use to instruct your robots. Only six of these are available at a time, although some robots have a seventh action permanently available in their ROM. Most of the instructions deal with movement, but the rest allow manipulation of items and shooting.

8.1 Step Forward

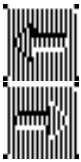
This is the simplest instruction to understand. It causes the robot to step forward one cell in the direction it is facing. If there is an item in front of the robot, and nothing blocking the path of that item, then the item is pushed forward in front of the robot. No special item is required to step forward.

8.2 Step Backward

The *Step Backward* instruction does what the name suggests: it causes the robot to step one cell backwards away from the direction it is facing. It continues to face in the same direction. No special item is required to step backward.

Unlike the *Step Forward* instruction, robots are not able to push items while stepping backward: handling items need to be done with care and the robot cannot see or steady an item that is behind it.

8.3 Step Left/Step Right



These lateral movement actions allow a robot to sidestep or strafe while continuing to face in its present direction. These instructions are very useful in navigating maze-like levels, as the robot can keep moving around obstacles and paths without constantly having to stop to turn left and right.

Lateral movement requires that the robot carry a *Walker* or to have one built in. It is not possible to push items around using lateral movement; only items that robots are facing can be pushed.

8.4 Leap

Robots may leap over a cell to land on the cell beyond using the *Leap* action. To do this, a robot must either carry the *Spring* object or have it built in. This instruction not only allows the robot to move at double speed, but also to leap over obstacles and hazards that lie in the way.

Leaping is subject to the following limitations:

- Robots cannot leap over walls, which are too tall.

- Robots cannot leap onto items or other robots. A failsafe mechanism prevents the robot from even attempting such a jump, and it will ignore the action.

8.5 Sprint

Wheeled or tracked robots can move at double speed using the *Sprint* action. This is literally a *Step Forward* instruction executed twice in one move. Sprinting robots can therefore push items at double speed. If the second step of movement is blocked, the robot merely moves one step forward. To *Sprint* requires no items to be carried or built in.

The *Sprint* action is inhibited by the *Walker*, so any robot carrying that item or having it built in cannot sprint. Robots sprinting are liable to the effects of both cells that they pass over in the course of a sprint, unlike *Leap* where the cell leapt over is completely avoided.

8.6 Turn Left/Turn Right/Turn About

The rotation instructions allow the robot to turn on the spot, either 90°to the left, 180°about, or 90°to the right. These can be used in sequence, so that if a *Turn Left* action is not available, three *Turn Right* actions, or a *Turn Right* and a *Turn About* will achieve the same effect, though in more moves. These actions require no items to be carried or built in.

Turning the robot requires a whole action in which the robot is not moving, and so is best kept to a minimum. Sometimes using lateral movement instead will allow a robot to progress further in the same amount of time as a combination of *Step Forward* and *Turn...* actions.

8.7 Take Item/Drop Item

Robots with an inventory can pick up and put down items in front of them using the *Take Item* and *Drop Item* actions respectively. This is the usual way to transport the data card to its reader, but *Take Item* is also used to pick up items like the *Spring* in order to use its capabilities when those capabilities are not built in.

An inventory consists of only one slot, so robots can carry one item at a time. If a robot is carrying a *Spring* for jumping, for example, it cannot at the same time pick up the data card. *Multibot* has no inventory at all, and cannot use these actions.

8.8 Shoot

Robots with a *Phaser* can fire it with the *Shoot* action. This fires a phaser beam in the direction the robot is facing. The beam travels in that direction direction until it hits something or goes off the map. If it hits a robot or an item, that robot or item is destroyed. If it hits a forcefield generator, all adjacent forcefields in cardinal directions will be deactivated.

It is possible to accidentally shoot the data card. This will bring about a swift end to your attempt to complete the level!

Chapter 9

Item Reference

There are six items in the game, for a variety of purposes.

9.1 Data Card

The *Data Card* is the item that you need to take to the *Card Reader* in order to complete a level. It gives the robot no special abilities when carried. If a data card is destroyed and there are not enough data cards to fill the readers on a level, then the level is automatically lost.

9.2 Walker

The *Walker* is a set of legs that can be attached to any robot to give it access to the lateral movement actions *Step Left* and *Step Right*. These legs are not as quick as tracks or wheels, though, and therefore they inhibit the *Sprint* action. *Strider* and *Multibot* have a *Walker* built in, and therefore do not need to carry this item to use lateral movement.

9.3 Spring

The *Spring* allows the robot to jump over cells using the *Leap* action. It cannot leap over walls, card readers or spawners, which are too tall. Other map cells like forcefields, teleporters, turntables and conveyor belts have a lower profile, and can be avoided with a well-placed *Leap*. *Bouncer* and *Multibot* have a built-in *Spring*, so they can leap without the need to carry one.

9.4 Phaser

The *phaser* allows a robot to shoot. This can be used to destroy guard robots, or to deactivate forcefields by destroying an adjacent generator. *Soldier* and *Multibot* have one built in, so they do not need to carry one in order to shoot.

9.5 Crate

The *crate* serves to block the path of robots and phaser bolts. Robots can push a single crate around, but a line of two or more cannot be pushed and could, for instance, block the path of a guard robot if carefully placed.

9.6 Robot Spawner

The *Robot Spawner:* an ephemeral item which is visible only during the deployment phase. It marks a spot in which a robot can be deployed. It disappears as soon as a robot is deployed there, and any unused robot spawners are cleaned up as soon as you proceed to the *Robot Programming* screen. You might not consider it an item at all, although the program does.

Chapter 10

Cell Reference

The individual squares on the level map are called *cells*, and each has a particular characteristic. There are twelve different cells in all, although that figure counts the all the variations of those cells that can act in multiple directions.

10.1 Floor

The *floor:* is the easiest cell to understand: the open floor. It does not block robots or items, and has no effect on those that travel over its surface.

10.2 Wall

The *Wall* is the opposite of the *Floor*. It blocks everything: all movement, item placement, and phaser fire. Robots cannot even jump over it. A robot wanting to get the other side will have to find a way around the wall.

10.3 Conveyor Belt

Four variations face north, east, south and west. A conveyor belt pushes a robot that moves on to or sits on it, at the rate of one cell per *move*. It will also push an item that is dropped or pushed on to it, at a rate of one cell per *turn*. Robots will not continue to be pushed once their RAM actions are exhausted. So for instance, if *Multibot* begins a turn at the start of a long conveyor belt, and does not attempt to move off it, it will be carried four cells along in a turn.

10.4 Turntable

Two variations are clockwise and anti-clockwise. A turntable turns a robot 90°in the appropriate direction after each action is performed. Like the *Conveyor Belt*, the turntable only affects a robot for as many moves as it has RAM. Turntables have no effect on items placed on them.

10.5 Teleporter

On stepping on a *teleporter*, a robot will be transported to the nearest teleporter on the level. If the robot does not immediately move off, it will be teleported again, either back to the original teleporter, or on to another teleporter if another one is nearer. Items dropped on a teleporter will be transported, but will remain on the destination teleporter for the rest of the turn. Only in the following turn will the item be transported again.

10.6 Forcefield

The *forcefield* is an energy barrier that will destroy any robot or item that moves into it. A robot with a spring can safely leap over a forcefield, as long as it does not land an another forcefield immediately beyond. Forcefields cannot be directly destroyed by phaser fire.

10.7 Generator

The *generator* supplies energy for adjacent forcefields. If a robot shoots a generator, the generator malfunctions: any adjacent forcefields *in cardinal directions only* are deactivated to give access to the open floor beneath. Some forcefields have no visible generators and cannot be deactivated.

10.8 Card Reader

The *Card Reader* is the receptacle for data cards. Robots cannot move on to the data card reader but they can push the card onto them. Other items can be placed on the card reader, but these have no effect other than to block the way for the data card to be inserted.

Chapter 11

Credit & Acknowledgements

Design, programming, art, sound:

- Cyningstan

Testing:

- Rumorsmatrix
- Ghostrider
- FractalMindMike
- Sparcie
- LadyVivianne


TEAM
DROID
JAM EDITION