STRATEGY GUIDE

Gungriffon



ARTDINK

SONY COMPUTER ENTERTAINMENT



Table of Contents

What is the Carnage Heart Programming Strategy Guide? .....Pg. 4

Step 1: Programming Basics Pg.4

Step 2: Actual Examples of Programs Pg.20

Step 3: Creating a Program Pg.26

Step 4: Sample Programs for Different OKE Types.....Pg.40

Step 5: Analysis of an Original Program.....Pg.48

Step 6: One-Point Advice.....Pg.52

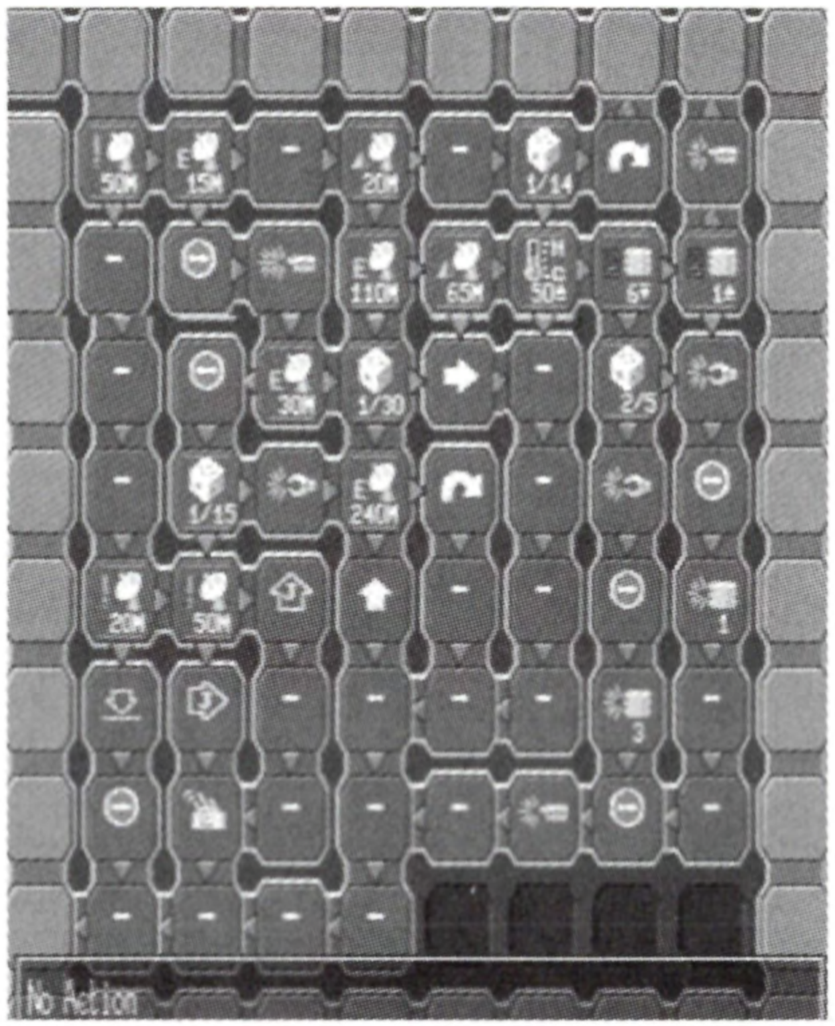
Practice Programs.....Pg.56

What is the Carnage Heart Programming Strategy Guide?

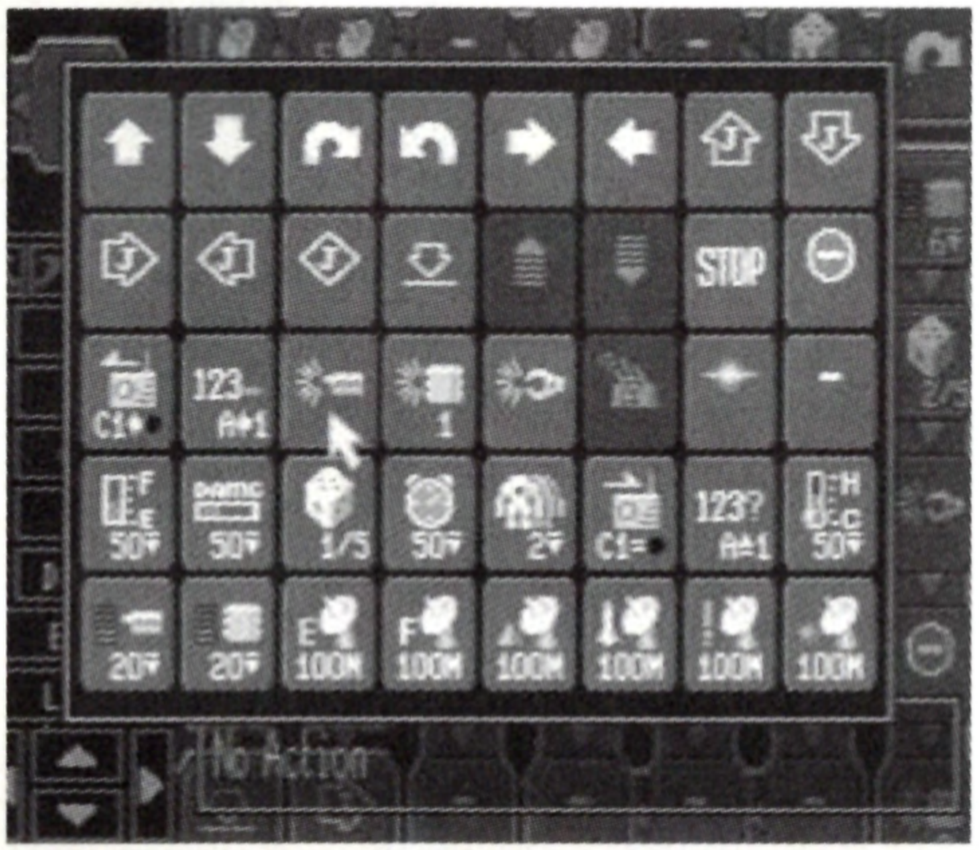
The Carnage Heart Programming Strategy guide was developed to help you understand the most innovative and challenging part of Carnage Heart, designing OKE software programs. This strategy guide will inform you of everything you need to know in order to start designing your very own OKE software programs. It is highly recommended that you read the Carnage Heart Manual and view the Carnage Heart: Mission Briefing tutorial disc before reading through the Carnage Heart Programming Strategy Guide. The Carnage Heart Programming Strategy Guide only offers help on designing OKE software programs. For any other gameplay assistance, please refer to the Carnage Heart Manual.

Programming an OKE is the most interesting part of Carnage Heart, but it is also the most challenging. You may be wondering at the start how to put together a program, so go ahead and read on to pick up a few pointers.

So What Exactly is a Program?

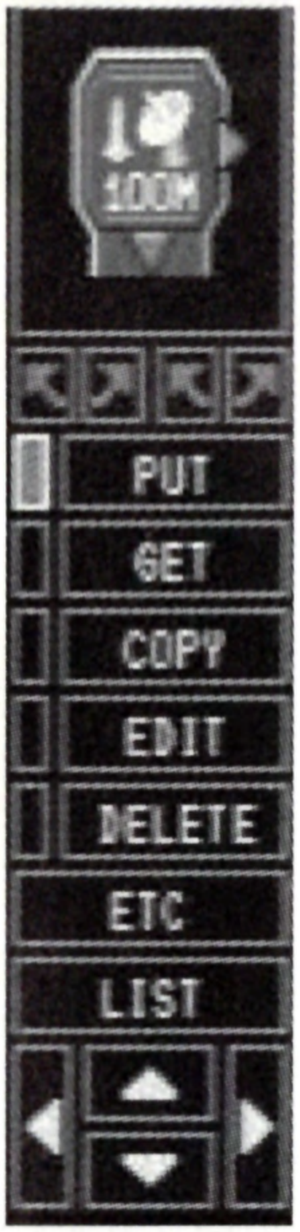


Among the various elements that make up an Over Kill Engine, what do you think is the most important? Why, it must be the program that can be called the brains of the Over Kill Engine. A program is a set of commands that determines the actions of the Over Kill Engine, such as evading an enemy's attacks or fighting back. When fighting with the enemy in a single-player game or when engaging in a Vs. Battle with the Over Kill Engine you have created, it is no overstatement to say that the outcome of the battle will be determined by the completeness of your program. The program is just that important in this game. Yet creating a program is not so difficult. You may be confused at first because this is not something that is in any other existing game, but once you get used to it you will soon be able to put together powerful programs. Once you become proficient at programming, you will see just how interesting this game can really be. So now we will show you a little theory and some of the tricks of the trade of programming. Be sure to read this section carefully.



Line up Chips to Create Your Program

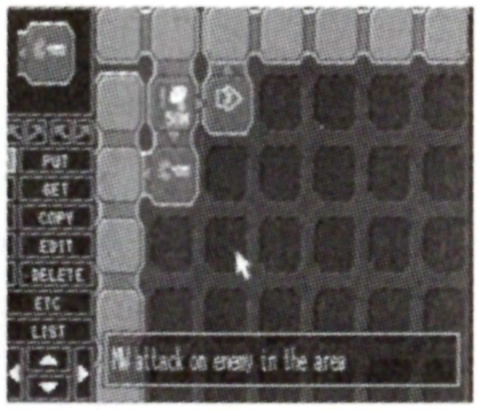
To create a program, line up the commands called chips inside the CPU frame called the chip table. There are 40 types of chips (for details, see Page 15), and the size of the chip table changes depending on the CPU you are using. We will describe detailed methods of arranging the chips and how they are processed (execution of the program) later, but the basic idea is that you build the program by stacking up toy block-like commands, such as move forward or fire a weapon, in the right order.



Creating a Program

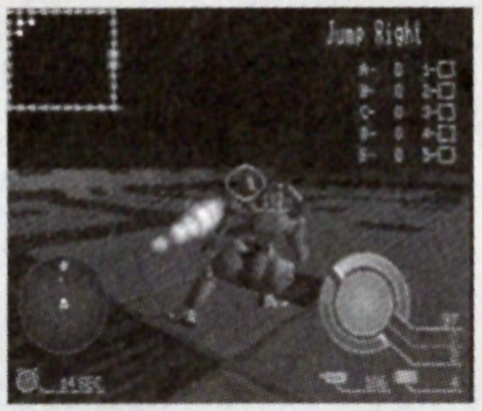
To create a program, after designing your "Hardware" from the "Design" screen, select "Software" to display the design screen. On this screen, the R2 button calls up a list of chips, so select the chips and place them on the chip table to create your program. Take particular note of the arrows on each chip, as they indicate the direction in which the processing of each chip advances. In addition, "EDIT" in the group of commands appearing on the left is a valuable command that can change the internal settings of chips you have already placed on the table.

Using Chips to Make Branches in Your Program



Although we described programming like stacking up toy blocks, just stacking them up would give you only a simple program with a one-track mind, such as one that only attacks repeatedly. The important point here is that the program itself should be responsive to external conditions, so that the program can branch to take different actions depending on the situation, such as if attacked, evade the attack or if not, attack the enemy. Program branches are generally executed using the search chips (for details, see Page 16), so if you learn to make good use of them, you will be able to create an advanced program that can handle any situation that may come up. For your information, each chip can branch into one of only two directions, one corresponding to "YES" and one for "NO."

Branching in a program.



This is an actual example of branching. This program is designed to jump to avoid a projectile if one is approaching...



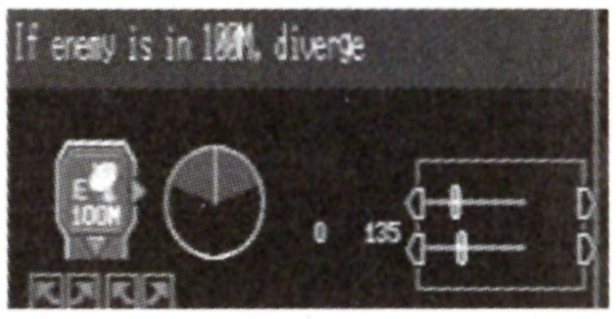
or fire the main weapon if not.

Types of Chips and How to Identify Them

The forty different types of chips can be broadly divided into about five categories. Each of the chips has a green arrow on its border to indicate the direction of processing. The search chips and others also have a red arrow which indicates the direction of the branch to be taken if the condition set within the chip is "YES." Note that the numbers and the like on the chip indicate a portion of the internal settings.

Once you select a chip to place on the table, the internal settings and other information about that chip are displayed. These settings will differ depending on the type of chip, but typically they include the effective range of this chip, the direction of arrows and the like. The internal settings are described in detail later. Note that this same screen will also appear if you select a chip already placed and use the "EDIT" command.

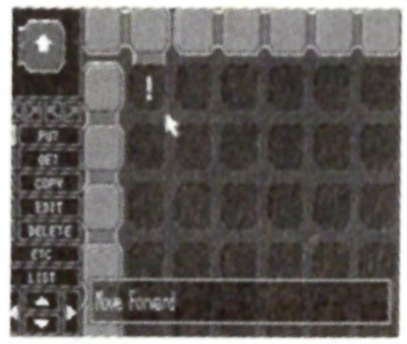


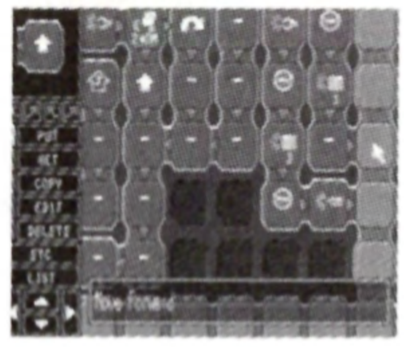


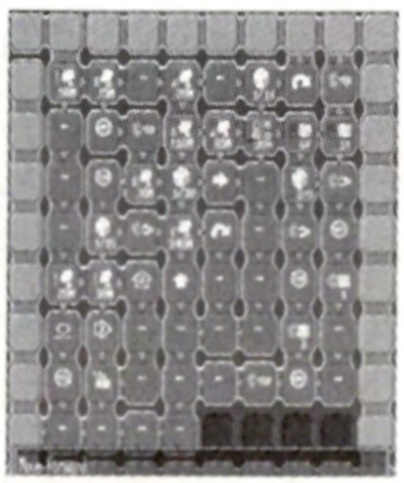
Understanding the Flow of a Program

So how exactly will a program created by placing chips on the chip table function?

First, processing starts from the upper left-hand corner of the chip table and ends at the bottom right. The program then repeats until the end of combat or until the OKE is unable to take action. Note that processing always starts from the frame at the upper left corner of the chip table. When assembling a program, always start from the upper left. Similarly, the end of processing is at the bottom right, but if you connect the end of the program to somewhere on the frame, processing will automatically proceed toward the bottom right and end.





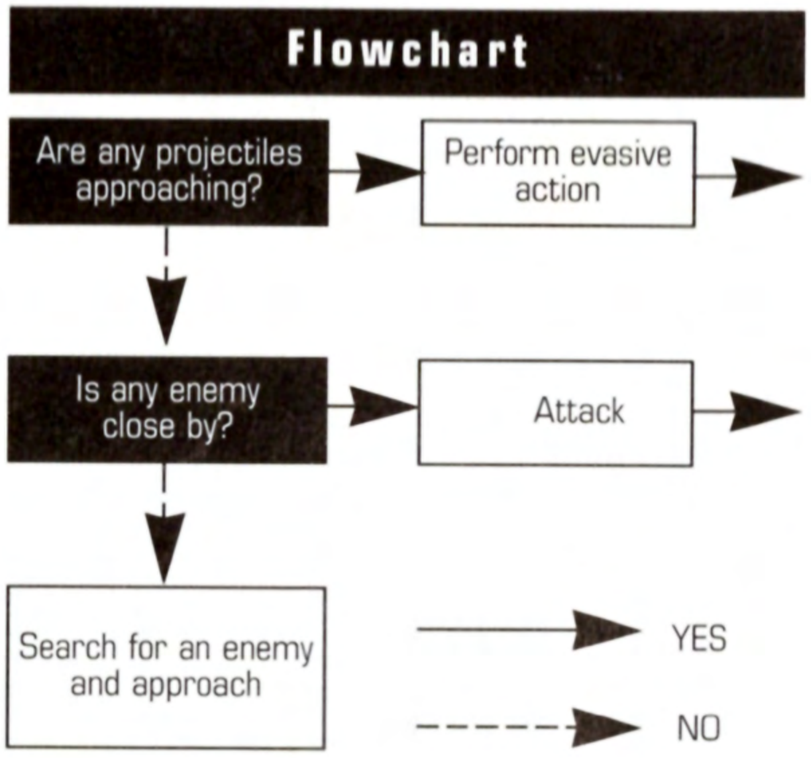


Now, to describe the workings of a program, following the program flow starting from the upper left, the content of each chip is executed. So for a search chip, the situation is checked at the time of execution. If necessary, a branch in the program occurs. Take particular care when executing a chip that performs some sort of action like attacking. Processing will continue even while the attack is in progress. So if another chip calling for another action is to be executed during this period, processing of that chip is ignored as unexecutable. You may not need to pay attention to this at the start, but you should remember this.

Using Flowcharts to Understand Programs

In this part, we use flowcharts to explain and analyze programs. A flowchart graphically illustrates the general flow of processing. One example of a flowchart appears below. A rectangular box with a white background indicates a simple action that may be performed, and a rectangular box with a black background indicates a decision about which action is to be taken depending on the situation. The solid lines point to the step to be taken if the answer to the question is YES and the dotted lines point to the NO choice. Flowcharts are also handy ways to keep notes for when you are creating your own programs, so it is helpful to be able to read and write flowcharts yourself. It is useful to plan your program with notes before you jump in and start programming.





Adjusting the Internal Settings of a Chip

How well the search chips and other branches are used is the one thing that most greatly determines how good a program is. Naturally, the internal settings are also very important. The internal settings differ greatly depending on the type of chip, as some may include only the direction of the arrows, while others such as the search chips allow you to set the effective range, effective angle, central axis and the direction of the red and green arrows. The effective range and angle can be easily changed by sliding the bars on the right. The central axis can be changed by pressing the X button within the circle in the center. In the case of status checking chips (for details, see Page 17) and special chips, be careful about the selection of the "greater than" and "less than" choices.

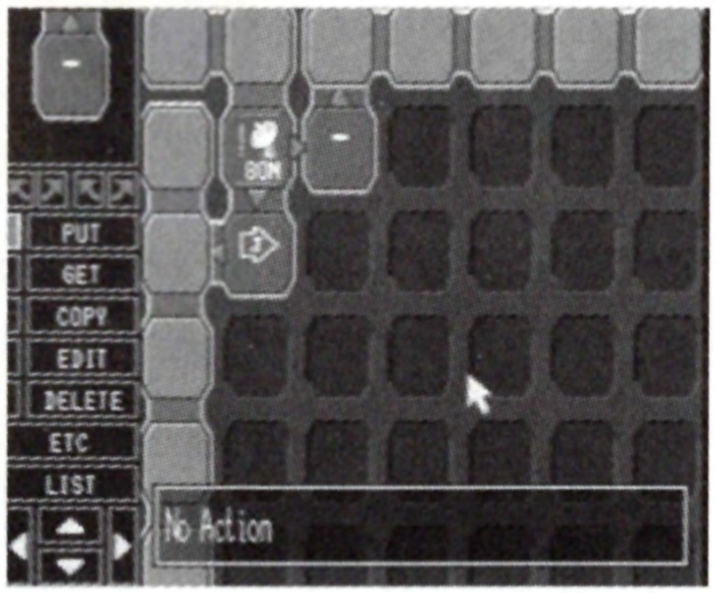
Chip internal settings. The numbers can be changed by sliding the bars on the right and the arrows can be changed using the switch icons on the left.





Change the central axis by directly manipulating the interior of the circle.

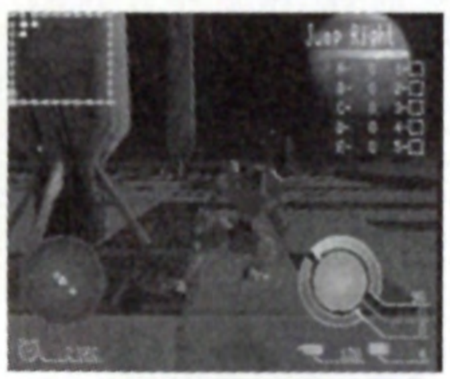




How Do Differences in Settings Affect Actions?

So how do differences in internal settings actually appear? For example, here we created a program that takes the evasive action of jumping to the right if a projectile (bullet) comes from the front, and tried changing the projectile searching range to three different values. The results shown below show that the longer the range the better the result. However, a range of 80 m may mean that the program may be excessively sensitive to projectiles, so that the evasive action is performed even in response to projectiles aimed at another of your OKEs.

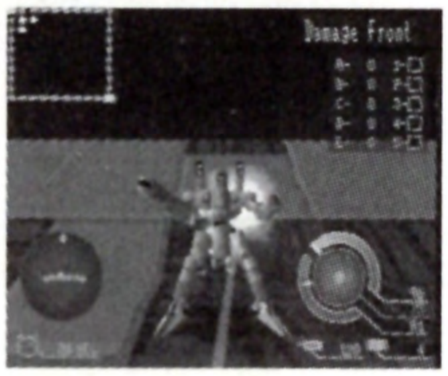
This program jumps to the right if a projectile is coming from the front. We used this program to illustrate the differences in actions that arise from different settings.



(80 m) At 80 m, projectiles can be evaded almost without fail. However, the searching range is very broad, so even projectiles not aimed at yourself may trigger evasive action, and the OKE may begin jumping relentlessly.



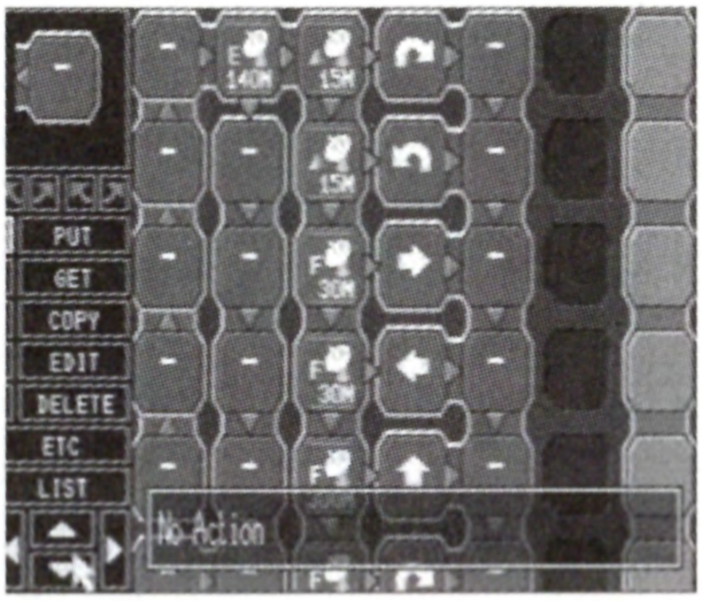
(50 m) Projectiles can be evaded safely if the enemy is armed with an assault rifle or laser. However, the problem is that you may not be able to get away in time if you are fired upon with a shotgun.



(10 m) You may try to jump but you will probably be hit due to the time lag before the jump command is executed. The time lag for actions will be described later.

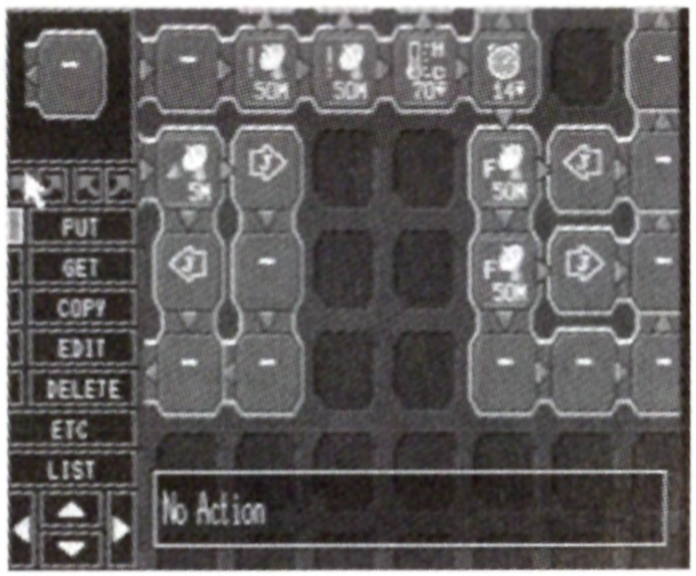
Using the Program to Change Action

A program is built by combining chips as we described before, but each chip does not in itself have much meaning. As in the example of evasive action on the previous page, only the combination of several chips makes a meaningful action. It is the method of combining chips that is the most difficult yet most important point in creating a program. The basic sequence you should always remember is search and then act.

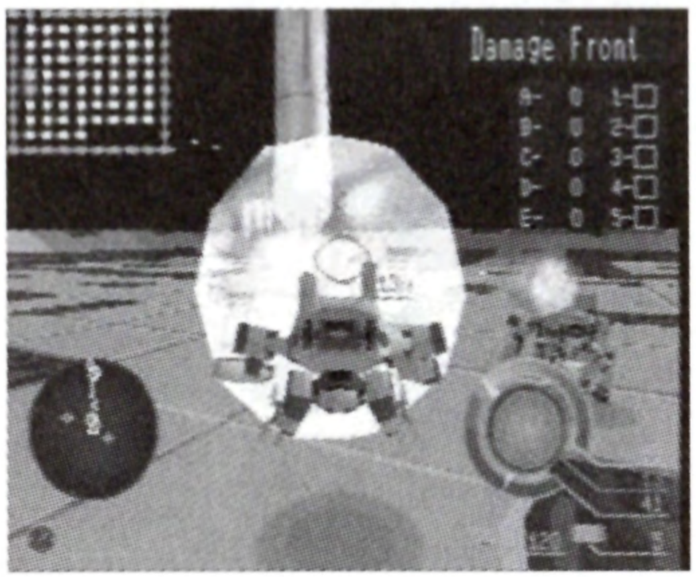


Actions Change Depending on the Combination of Chips

So now that we have seen how the chips are actually combined, let's take a look at creating a complete program. It is easiest to think of combining the chips on the basis of actions so that they can be changed more easily later on. This action-based programming style is a method of creating a program by dividing the program into four blocks corresponding to the basic actions that the Over Kill Engine can perform in battle, and then assembling the blocks to create a complete program.

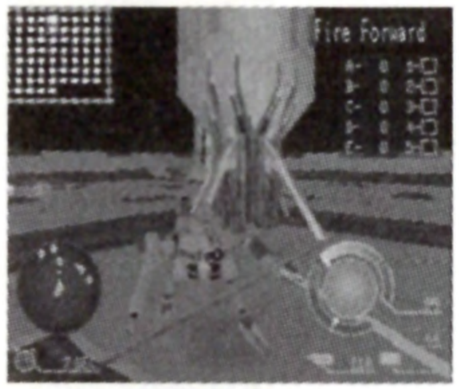


The four basic actions are: attack, evade, search for the enemy and move. To create a comprehensive OKE it is a bit more complex, but as long as these four elements are present, at least your OKE will be able to fight. Samples of these are shown on the next page.



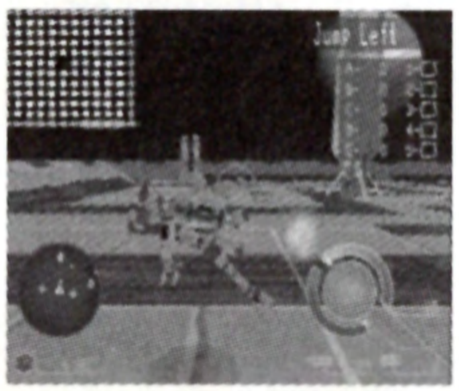
The greatest benefit of assembling a program in an action-based manner is that it can be thought of relatively simply.

A drawback to this approach is that the program size becomes relatively large, and can be too simplistic.



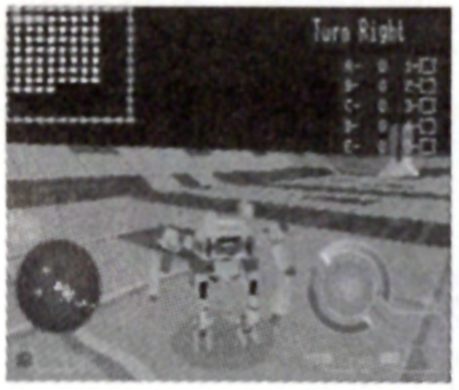
Attack

Attack is different from other actions in that it can be performed with even just a single chip. In an extreme case, it is possible to create a program that is able to fight by placing only a single chip that fires the main weapon. The problem lies in the variation of attacks.



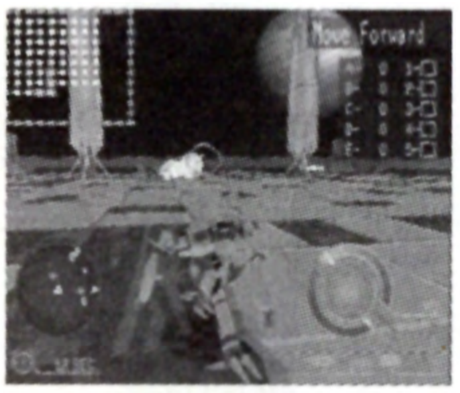
Evade

Even the most powerful Over Kill Engine may not be able to win if it continues to absorb an enemy's attacks. For this reason, it is important to know how to evade an opponent's attack. This is generally done by jumping, but this becomes more difficult for the tank and flying type OKEs.



Search for the Enemy

Searching for the enemy may not be needed at first, but it may be important depending on your fighting strategy. Note that here, "searching for the enemy" refers to not only searching for an enemy, but also advancing toward the enemy. Just searching for the enemy can be performed with the enemy search chip alone.

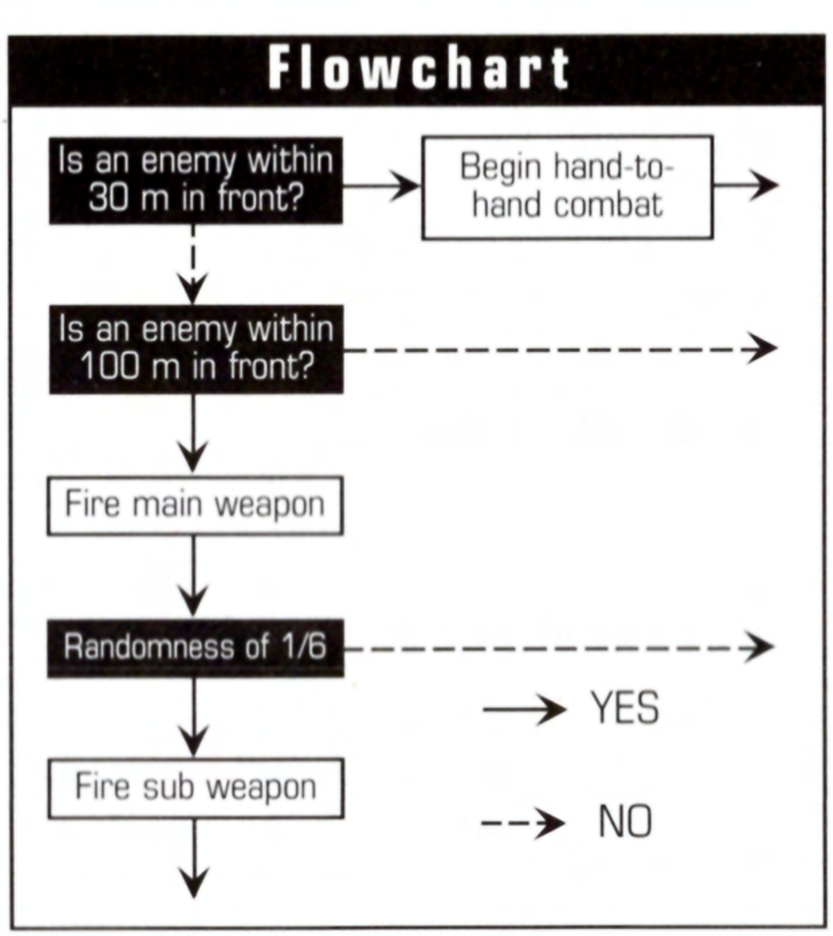


Move

Movement is important in order to approach within attack range after finding an enemy, and in order to escape. In addition, this element is also indispensable in maintaining a constant distance from your other OKEs.

1 Attack Surrounding Enemies

The first sample program is one in which the main weapon is used to attack an enemy that has approached into intermediate range. After the initial attack, the sub-weapon is fired with a probability of one time out of six. Finally, hand-to-hand combat is initiated when the enemy approaches within short range, here 30 meters.

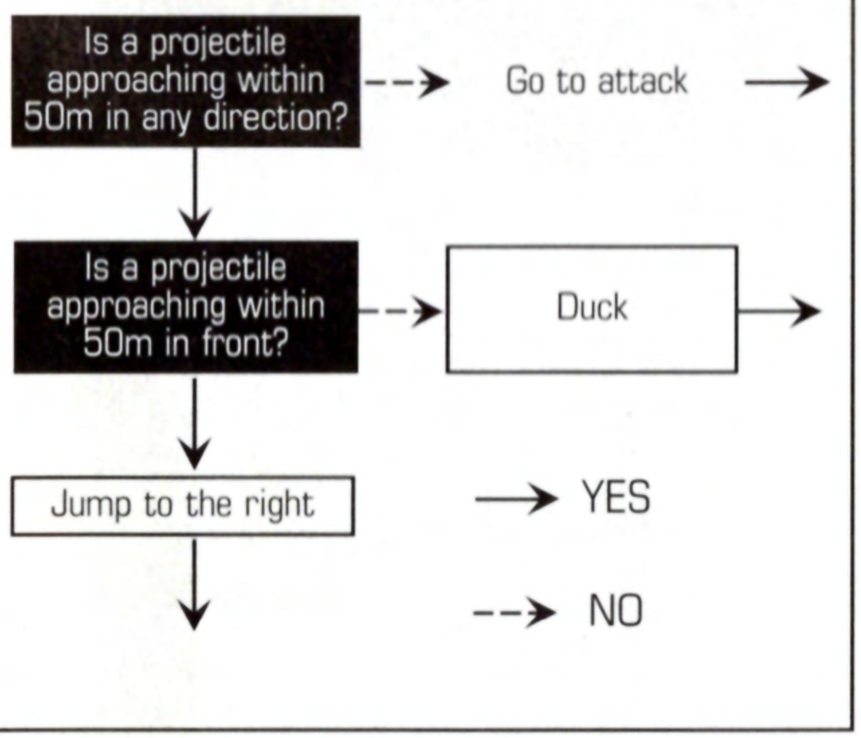


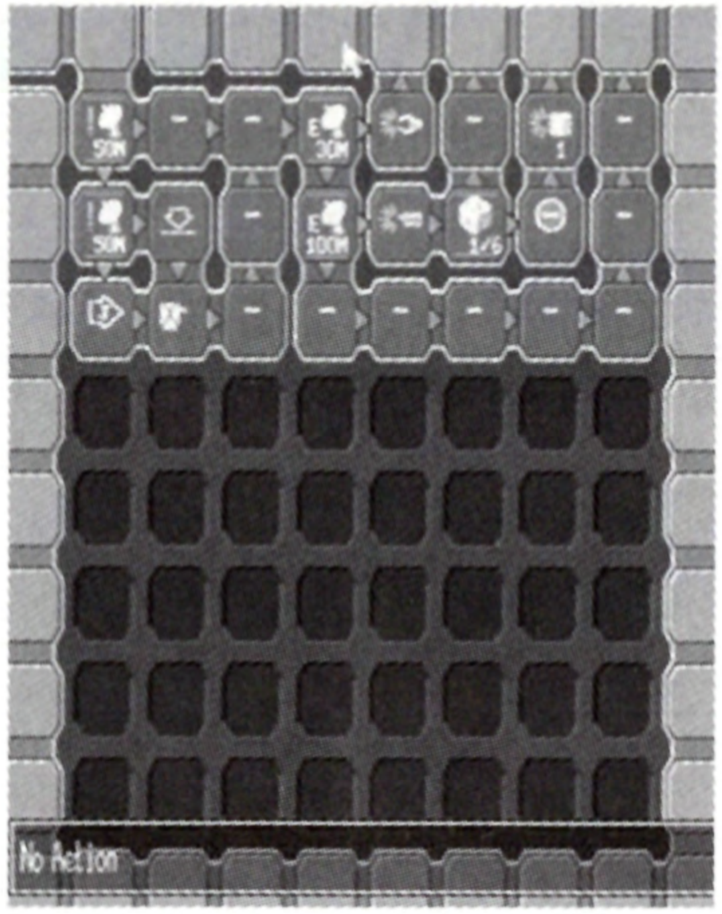


2 Evade Projectiles and Attack

Adding evasive action elements to the previous program gives us the following program. A search for projectiles is performed prior to an attack, and if the answer is YES, evasive action is taken. The evasive action is divided into two stages: jump to the right if the projectile is in front of the OKE, and duck in other cases. Note that when the evasive action is over, the program ends and then returns to the beginning, where the projectile search is restarted.

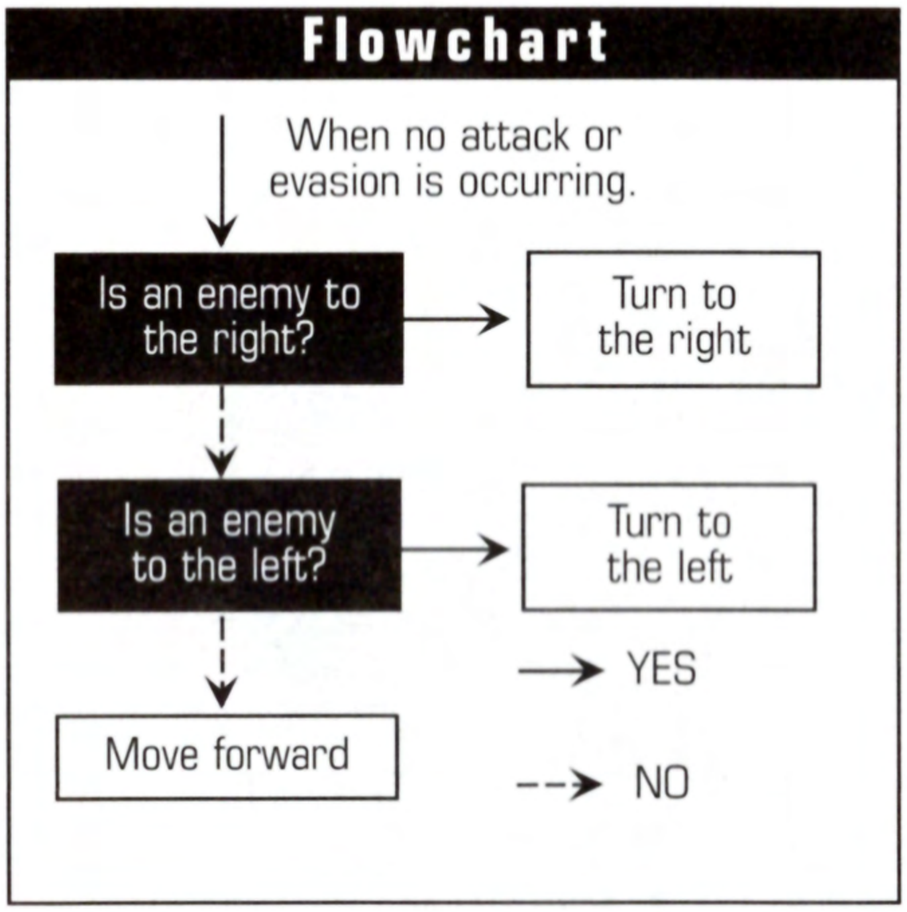
Flowchart

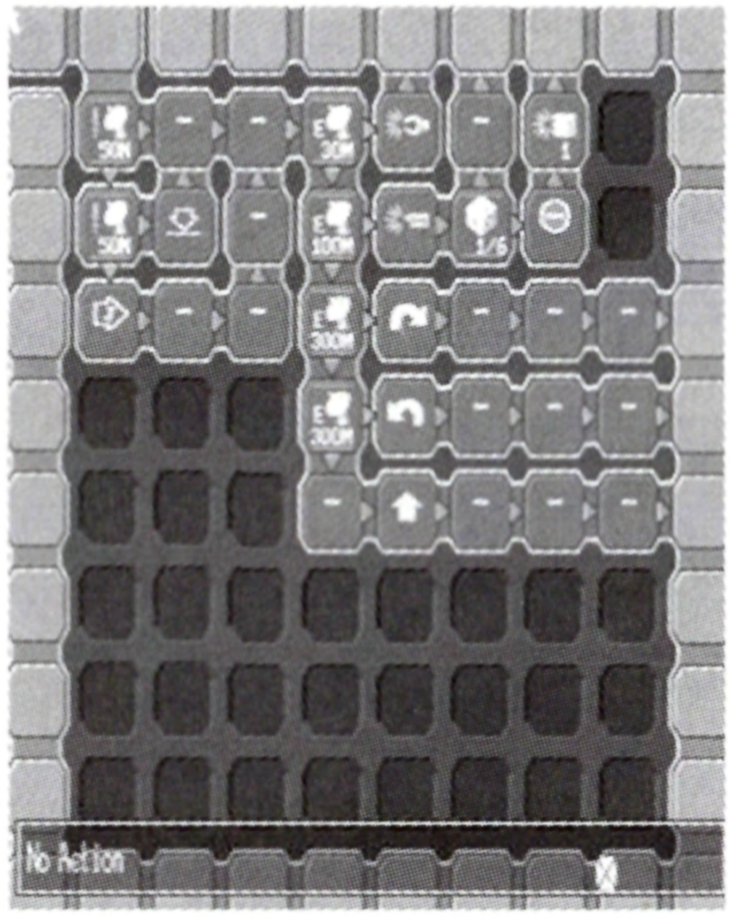




3 If There is No Enemy to Attack, Search for the Enemy and Move

In sample programs 1 and 2, if there are no projectiles approaching or enemies present within the set region, the OKE will just sit there, halted. To avoid this situation, when neither of the other conditions holds true, the OKE will search for the enemy and then move toward the enemy. Now a sequence of actions is started in which the OKE will search for the enemy on its own, avoid attacks and attack the enemy depending on the situation. This can be thought of as a basic form of a program.







Description of All Program Chips

There are 40 kinds of chips that serve as the material used to assemble programs. Although only one command is used to execute the individual chips, a single chip may include multiple commands such as the "Fire main weapon" chip which searches for the enemy and fires the weapon. Here, we have divided the chips into five groups; movement, search, attack, status check and special – and present the effects of each. Some chips have unexpected side effects, so read the descriptions carefully.

Movement Chips (Total of 13 Types)

The movement chips can be divided further into three groups; direct movements such as move forward, indirect movements such as jump forward, and change altitude that can be used only by flying OKEs. When a movement chip other than the jump chips is executed, the command remains in effect until the next movement is started.



Direct Movement:
There are six types – move forward, move back, turn right, turn left, move to the right (move laterally to the right) and move to the left (move laterally to the left). Note that tank OKEs cannot make lateral movements.



Indirect Movement:
These jumps are usable only by the two-leg and multi-leg OKEs. There are five types – jump straight up, jump forward, jump back, jump to the right and jump to the left.



Change Altitude:
These altitude changes are usable only by flying OKEs. There are two types – ascend and descend. A special ability of these chips is that they can be used together with other movement chips. For example, it is possible to ascend while moving forward.

Search Chips (Total of 6 Types)

The search chips are useful in letting the OKE determine what is present in a set region centered around itself, and branching the program depending on the results. These chips search for projectiles, allies, explosives and the like. So they are probably the type of chips used most frequently.



Search for enemies. Searches to see whether or not an enemy is present within the set region.



Search for allies. Searches to see whether or not another of your OKEs is present within the set region.



Search for obstacles. Searches to see whether or not an obstacle is present within the set region. Note that only the center point of the obstacle is found. It is wise to make the search angle a bit on the wide side.



Search for projectiles. Searches to see whether or not a projectile is present within the set region. Projectiles include bullets, including friendly fire. However, friendly-fire bullets fired from very close range are excepted.



Search for missiles. Searches to see whether or not a missile is present within the set region. However, missiles are also treated as projectiles. For this reason, this chip is mainly used for missle guidance jamming (with a jammer such as the Omni P1 ID).



Search for explosives. Searches to see whether or not an explosive is present within the set region. Explosives include landmines and floating mines, but unfortunately, enemy OKEs that are about to explode are excluded.

Attack Chips (Total of 3 Types)

The attack chips offer the highest performance among all of the chips. One of these chips alone can search for an enemy and mount an attack on the enemy relatively accurately. Note that when one of these chips is executed, the weapon is fired after a fixed period of time needed for preliminary actions. Even if an enemy may be present in the rearward direction within the set region, the weapon would be fired while the OKE is turning around, so there will be no hit.



If an enemy is present within the set region, this chip fires the weapon provided as the main weapon. Note that the Max. enemy search distance is fixed at 200m.



If an enemy is present within the set region, this chip fires the weapon provided as the sub weapon. Note that the Max. enemy search distance is fixed at 200m. In addition, the number of rounds to be fired can be set so that up to five rounds are fired simultaneously.



Usable only by two-leg and multi-leg OKEs, this chip begins hand-to-hand combat if an enemy is present in front of the OKE. The Max. enemy search distance is fixed at 30m. Depending on the type of OKE, hand-to-hand combat may be the strongest of all of the attacks.

Status Check Chips (Total of 5 Types)

These chips perform checks on the amount of ammunition remaining, the temperature of the OKE and other aspects of the state of your OKE. While these chips may not be needed at the start, they will become more important as you become more familiar with the game. They can be used in programs that change the strategy used depending on the situation; for example, if the damage to the OKE exceeds a certain value, the OKE may fire off the remaining missiles randomly, or get in close to the enemy and self-destruct.



Checks the amount of fuel remaining, reporting whether it is greater or less than a percentage in the range from 1-100%. Note that fuel is consumed only during some sort of action; when the fuel reaches 0, the OKE becomes disabled.



Checks the amount of damage to the OKE, reporting whether it is greater or less than a percentage in the range from 1-100%. This chip is useful in deciding when to start a desperation attack just before the OKE is about to be destroyed.



Checks the temperature of the OKE, reporting whether it is greater or less than a percentage in the range from 1-100%. When the temperature reaches 80% or higher the OKE will begin taking damage, so this chip is useful in avoiding such situations.



Checks the number of rounds of ammunition remaining for the main weapon, reporting whether it is greater, less or equal to a number in the range from 1-100. It is wise to perform this check before attacking with a weapon such as the shotgun that is equipped with a very limited number of rounds.



Checks the number of rounds of ammunition remaining for the sub weapon, reporting whether it is greater, less or equal to a number in the range from 1-100. The sub weapons are equipped with only a limited number of rounds, so be sure to perform this check before attacking.

Special Chips (Total of 9 Types)

The chips that do not fit in the other four categories are presented here as special chips. Some of these chips have major effects depending on how they are used. So careful study is recommended.

Note that the counter check and channel check could be handled as special chips, but their effects are so special that they are handled separately.



Usable only by two-leg and multi-leg OKEs, this chip causes the OKE to duck on the spot. This is useful in evading projectiles that have approached too close for the OKE to jump in time, and for other cases. Ducking also increases the defensive capacity of the OKE.



This chip can be executed by OKEs equipped with an Interference Device (such as the Omni P1 ID). Executing this chip jams the guidance system of a missile currently locked onto (aiming at) the OKE. Note that jamming consumes a certain amount of fuel.



Halts the command currently being executed. This chip can be used to halt movement or stop an attack if the OKE's temperature gets too high.



Halts the flow of processing until the next action becomes possible. For example, an attack cannot be executed during a jump, even though the attack chip may come up. So this chip is useful in preventing the execution of such an attack and will cause the program to wait for the action to finish so that it does not skip the next action.



Self-destruct. If the OKE is in a hopeless situation, self-destruct is a last desperation measure. Self-destruct is more effective with OKEs having large numbers of parts (for example, two-leg types), as they have a better chance of striking the enemy.



Do nothing. This chip has no effect. These chips can be thought of as corridors used to connect commands.



This chip randomly branches the program. The numerator and denominator may be set in the range from 1 to 99. This can be used to prevent the OKE's actions from falling into the same pattern.



Time check. A count is started at the start of battle and this chip checks to see if the count is greater or less than a value in the range from 2-200 seconds. This chip has many uses such as for firing off the remaining missiles randomly with only 10 seconds remaining.



Check the number of your OKEs remaining. It checks whether the number of your OKEs able to function is greater, less or equal to a number in the range from 1 to 3.

Channel Check Chips (Total of 2 Types)

These chips may be the most difficult to use of all of the chips. A signal consisting of one of five colors can be sent over channels 1-5. This chip may be used for sending a signal if the damage to the OKE exceeds a certain value, to call in other OKEs for assistance or for other purposes.



Sends a signal of one of five colors over one of the channels 1-5. Naturally, a signal that is sent can also be received by the OKE that sent the signal.



Receives a signal of one of five colors on one of the channels 1-5. Processing will branch when the signal is received.

Counter Check Chips (Total of 2 Types)

The method of using these chips is also complicated. However, these chips can be useful once you get used to them. For example, it is possible to increment the counter after each jump to the right and when the counter reaches a set value, switch to jumping to the left and avoid going to the edge of the battlefield. The other possible uses are endless.



This chip executes one of the five arithmetic operations of addition, subtraction, multiplication, division or assignment using the specified number and one of the variables A through E. Here, "assignment" refers to setting the value of the variable to the specified number.



Branches to a certain command when one of the variables A through E becomes greater than or less than a certain value in the range from 1 to 100. Used wisely, a counter can be used instead of the status check commands.

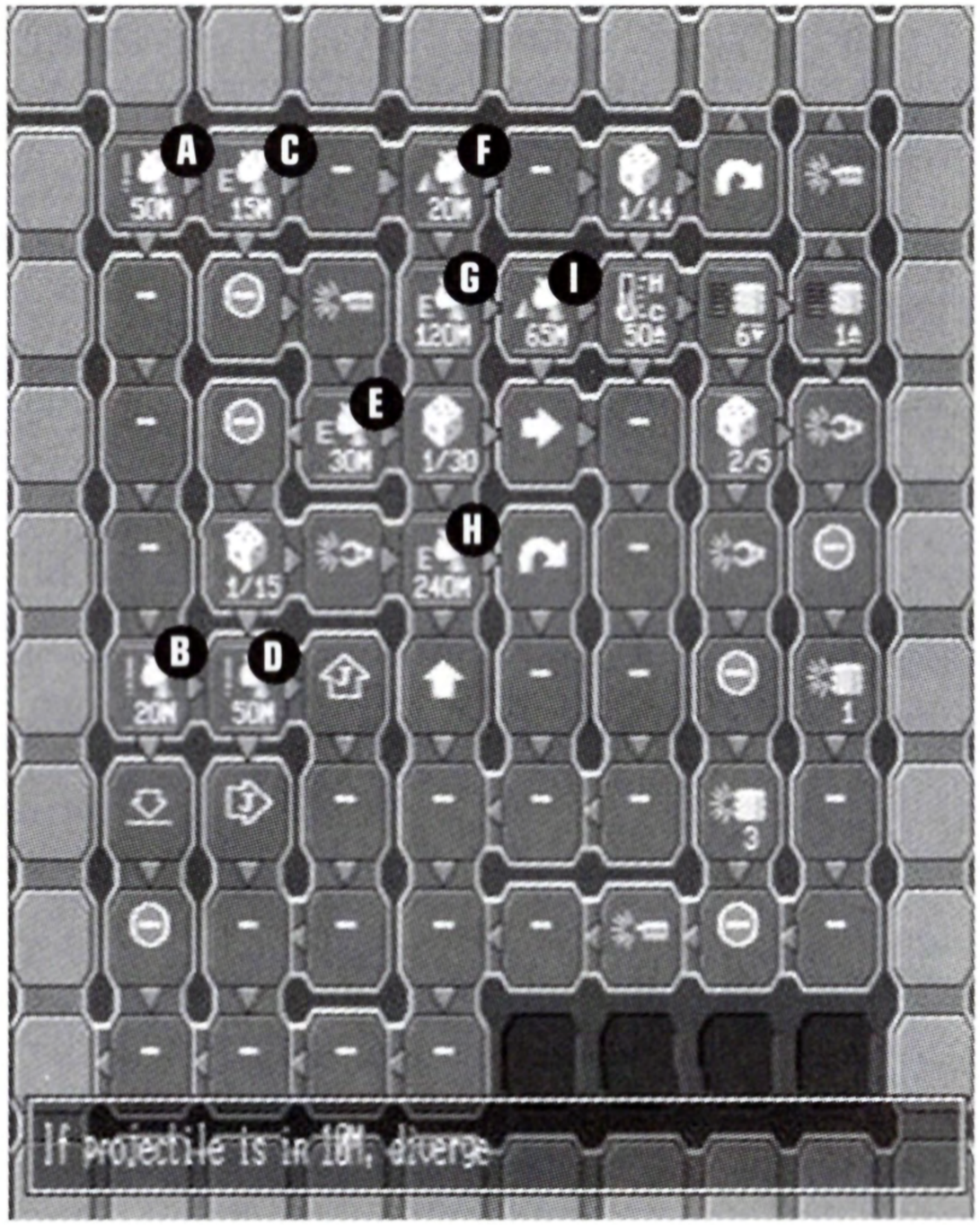
Actual Examples of Programs

So by now you should be familiar to a certain extent with programs. But just understanding the theory will not allow you to actually create a real program from scratch. So here we present a detailed description of actual programs, using the "auto" programs as examples.

Check Out the "Auto" Programs

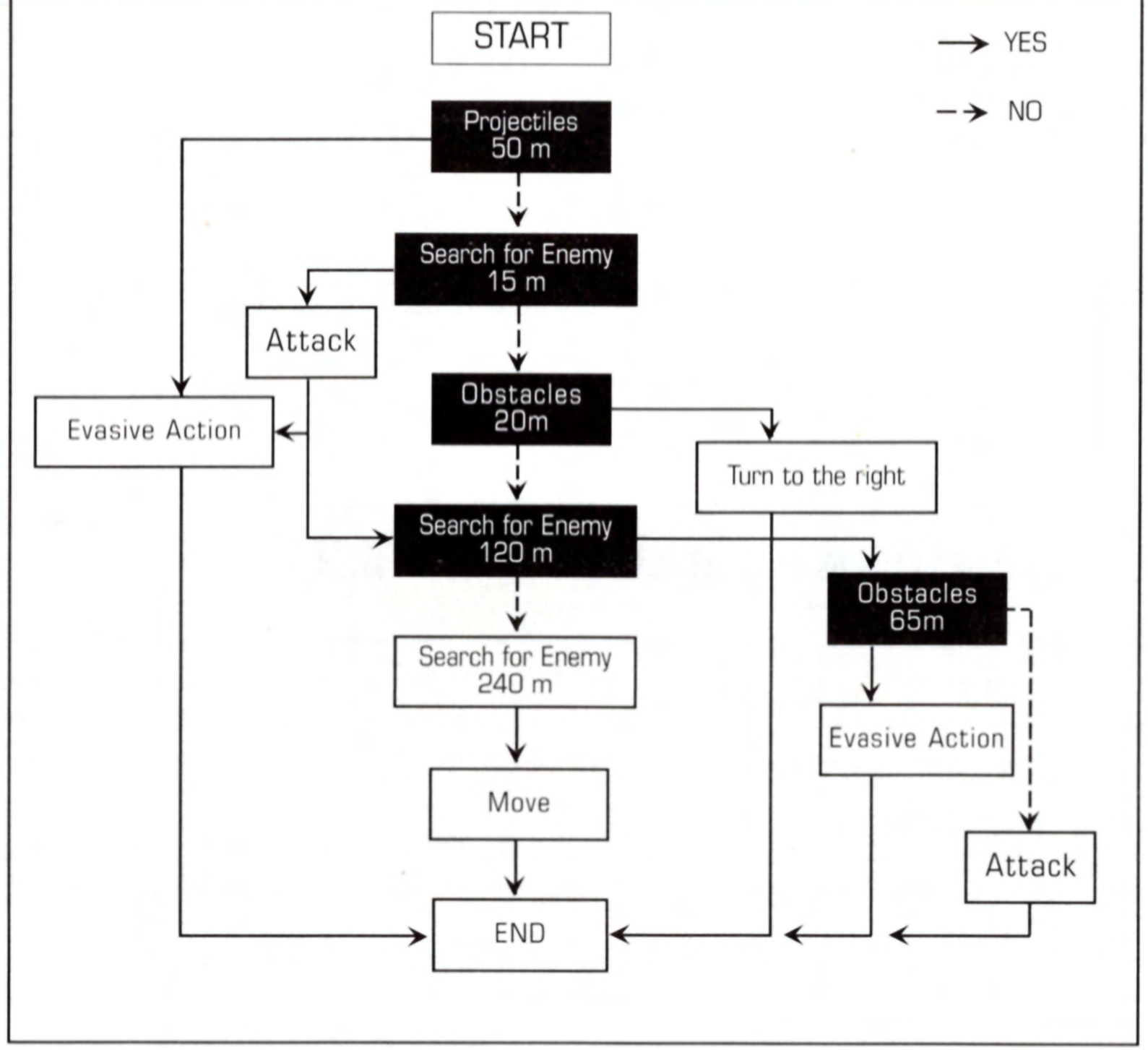
The first program that anyone uses will probably be an "auto" program. But to be perfectly honest, these programs are not very strong. They are rather weak. However, the various techniques used in these programs do serve well as learning material. So we will now present the details of these auto programs and the techniques used therein. Upon mastering these techniques, you will be on the road to mastery of programming.

Example of an auto program for a two-leg OKE.



A	50m	Axis 0° Scope 354°	B	20m	Axis 0° Scope 354°	C	15m Axis 0° Scope 354°
D	50m	Axis 0° Scope 95°	E	30m	Axis 0° Scope 67°	F	20m Axis 0° Scope 90°
G	120m	Axis 0° Scope 90°	H	240m	Axis 180° Scope 219°	I	65m Axis 0° Scope 90°

Flowchart

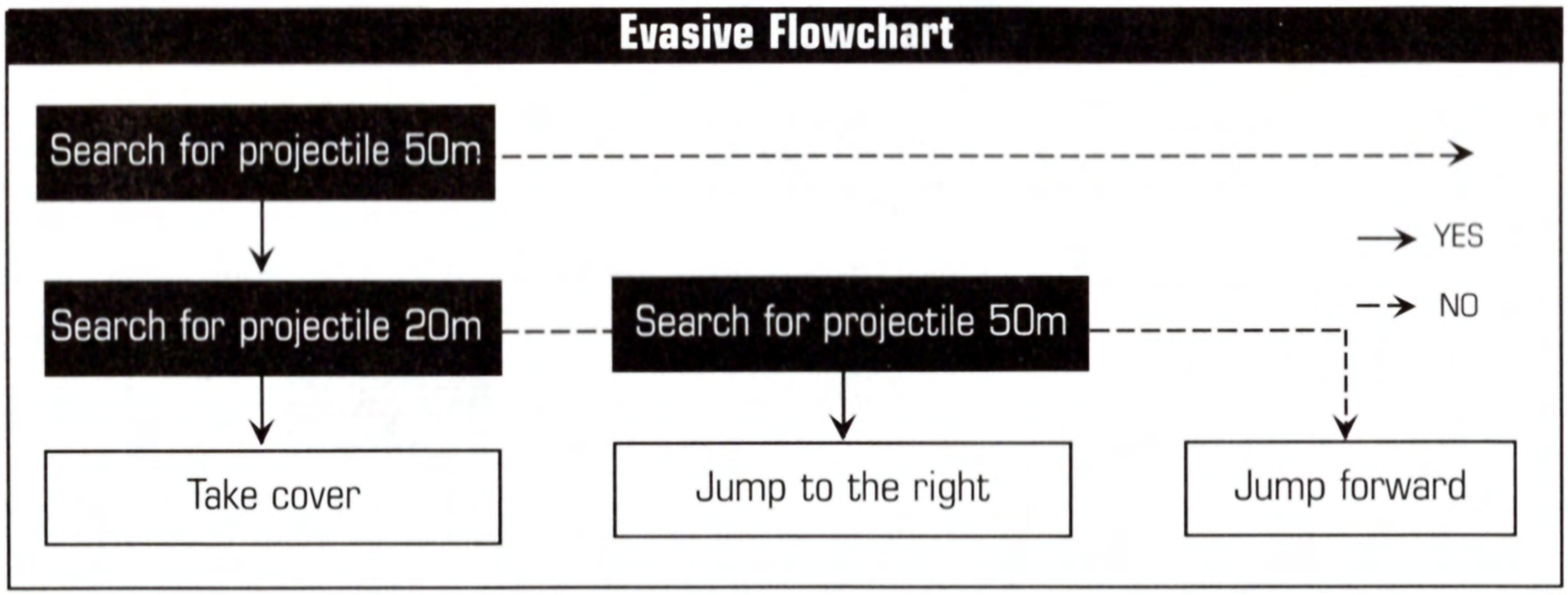


Actions Determined by the Flowchart

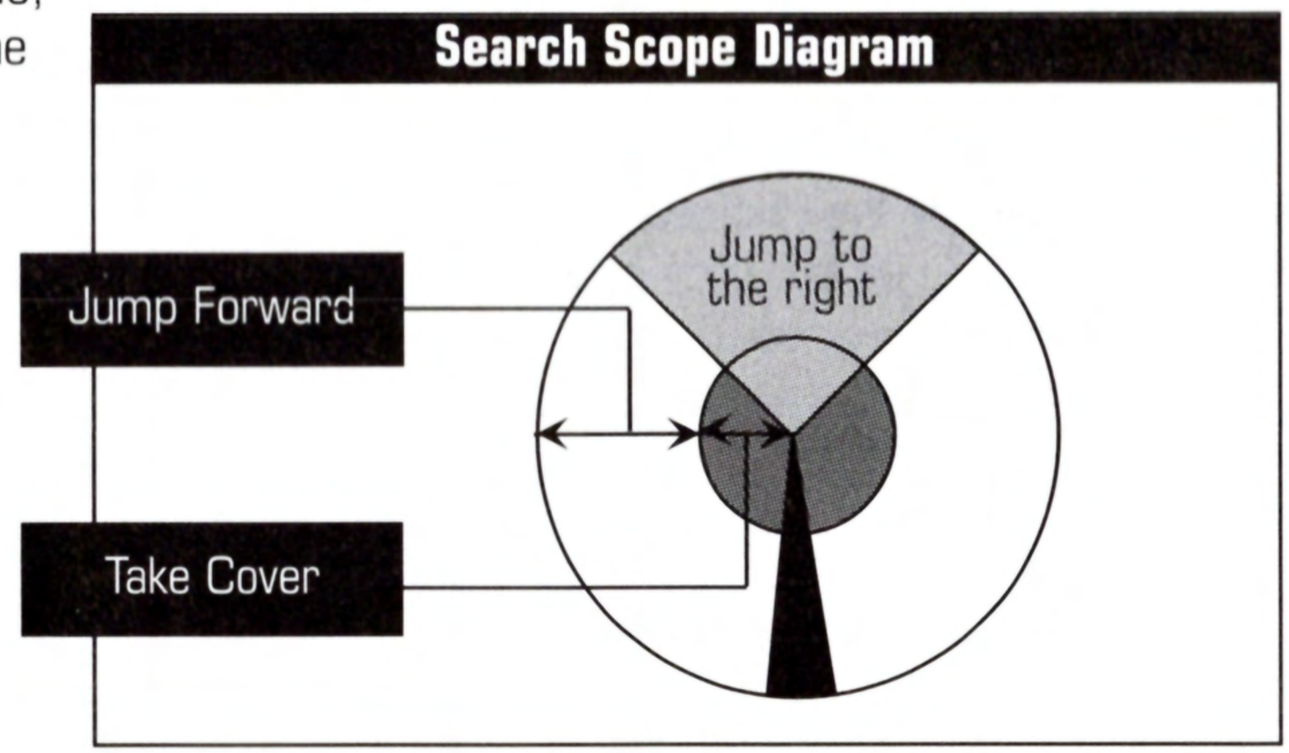
The flowchart graphically shows the auto program in general. What you can see from this is that the program is designed to waste very little time. This program evades enemy attacks and avoids obstacles while walking. In addition, the attacks are divided into two (or three) types consisting of either hand-to-hand combat or shooting attacks depending on the distance from the enemy. Also, make note of the method of checking the temperature of the OKE.

Method of Evading Enemy Attacks Depends on the Distance

The evasive action at the start of the program uses a different type of evasion depending on the location of the projectile. Specifically, looking at the flowchart you will see a search first at 50m over the full scope. If a projectile approaches within 20m, the evasive action of ducking is used. When the projectile is further than 20m away, the evasive action is jump to the right if the projectile is in front of the OKE or jump forward if not. This use of different types of evasive actions serves well as the basis for an evasive action strategy.

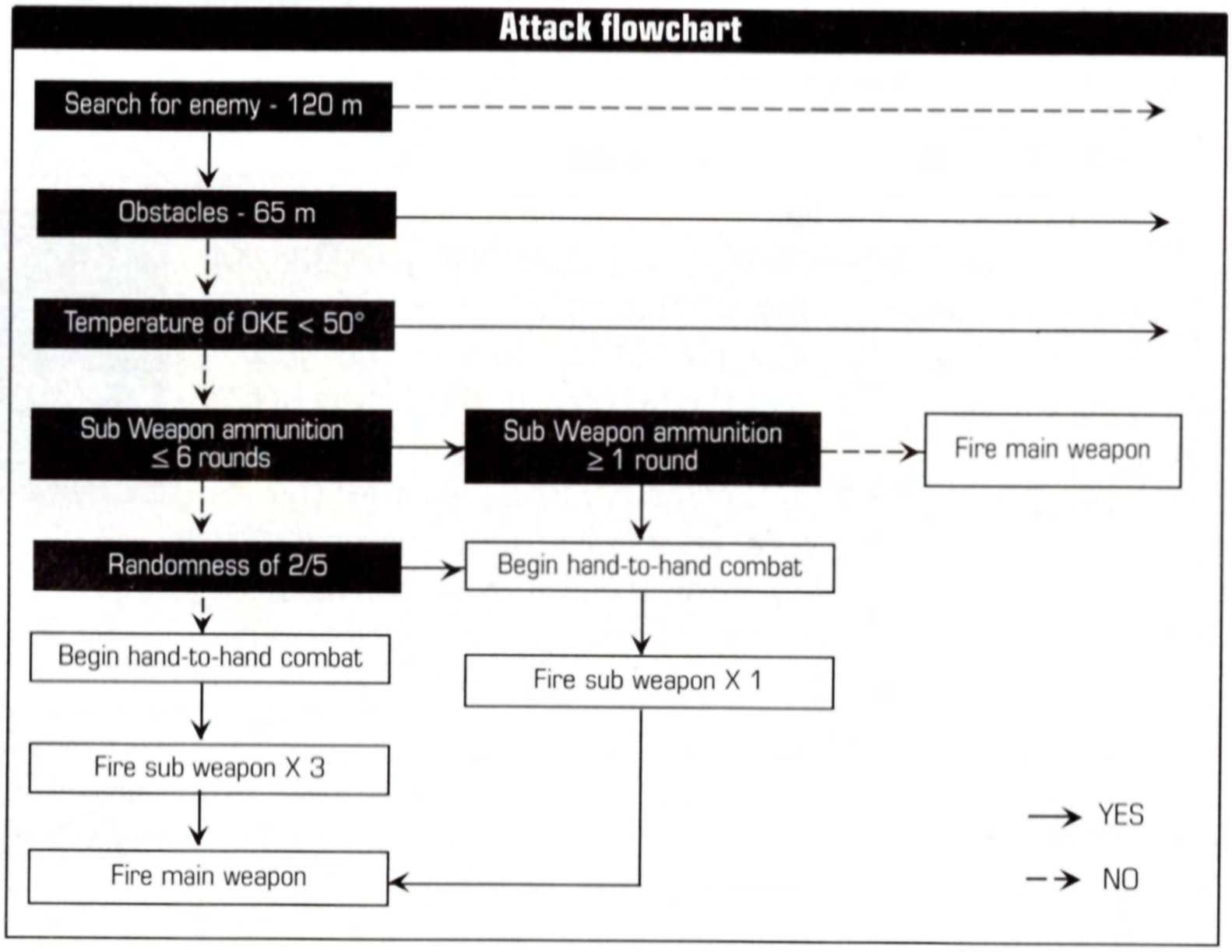


The diagram to below illustrates the scope of the projectile search. Here, the OKE jumps forward in the event of non-frontal attacks further than 20m. Although this evasive action leaves the OKE very vulnerable, it is effective in shortening the distance from the enemy.



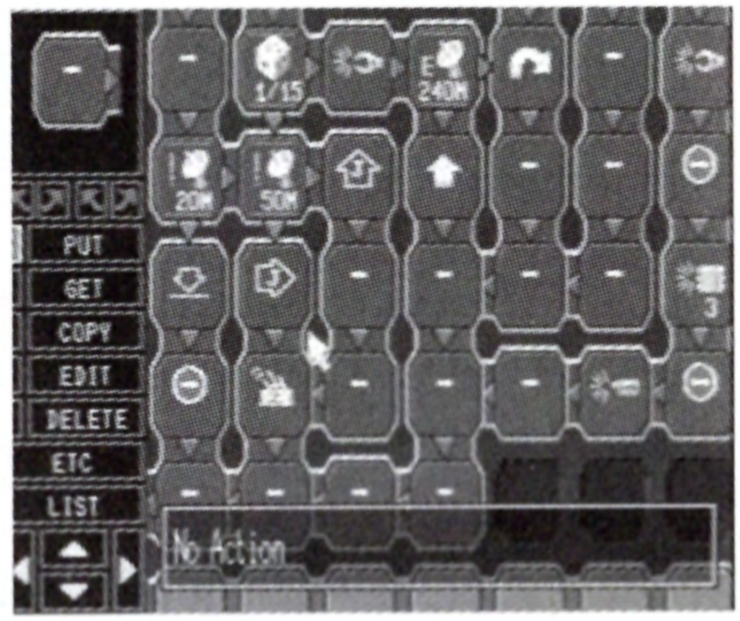
Attacks on the Enemy are Switched Depending on the Situation

As described below, the different attacks of hand-to-hand combat or shooting attacks are used selectively depending on the distance. First, if the enemy is within 15m, the OKE attacks with the main weapon and hand-to-hand combat. Next, if the enemy is within 120m, here the program has an interesting response. After checking for obstacles and the number of rounds remaining, hand-to-hand combat is first initiated before firing the sub weapon. This tactic is taken because if the enemy can be knocked over in hand-to-hand combat, the sub weapon will surely hit its mark.



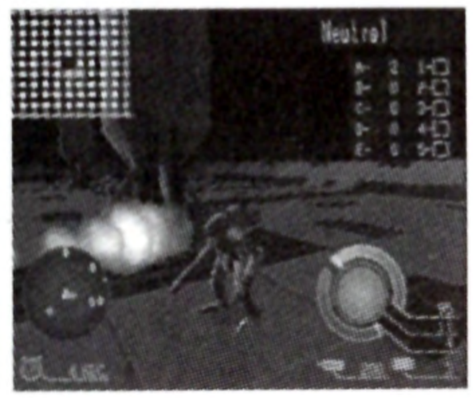
How Well Does this Program Do in Combat?

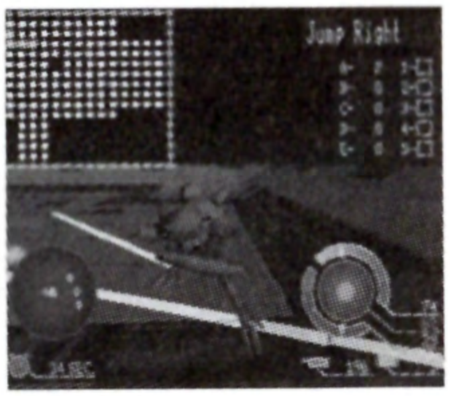
So now let's look at how well this auto program does in combat. Naturally, it will not deviate greatly from the actions determined by the flowchart. It also has deficiencies that become apparent only in battle. Take the attacks for example. The hand-to-hand combat prior to the firing of the sub weapon will hardly ever be used. This is because the sub weapon will run out of ammunition long before the enemy is close enough for hand-to-hand combat. Although the idea may be interesting, this program is not able to make good use of it.



Check Out Your Program in the Test

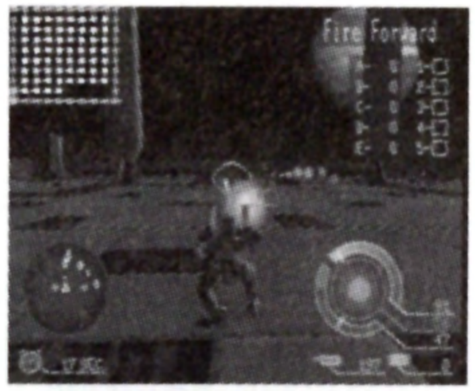
Once you have completed your hardware and software, your Over Kill Engine is complete. But if you send it into combat without testing, it may not be very effective. We recommend that you check out your program first in the Test. Note that the purpose of the Test is not to defeat the opponent but rather to make sure that the program functions as you expect it should. Don't worry about whether you defeat or are defeated by the virtual enemy, but make sure everything works correctly. The effectiveness of the program is tested most rigorously in Field No. 1 and Field No. 4 where the terrain is complex. If your program is able to fight as you expected in these fields, you know it is the real thing. In the Test, take special notice of how the OKE moves. Does it behave as you expected?





Even the Auto Programs Have Weaknesses

In addition to the inability to make use of hand-to-hand combat, other weaknesses abound. The greatest weakness is that no searching for allies is performed. So the weapons are fired even though another of your OKEs may be present in the direction of the enemy. This brings up the possibility of losses to friendly fire even before you get to the enemy. In addition, OKEs on the same side moving in unison can also induce losses to friendly fire. Excessive firing of the sub weapon is also a problem, and also the low level (narrowness) of the enemy search capacity can also result in giving the enemy a head start in attacks.





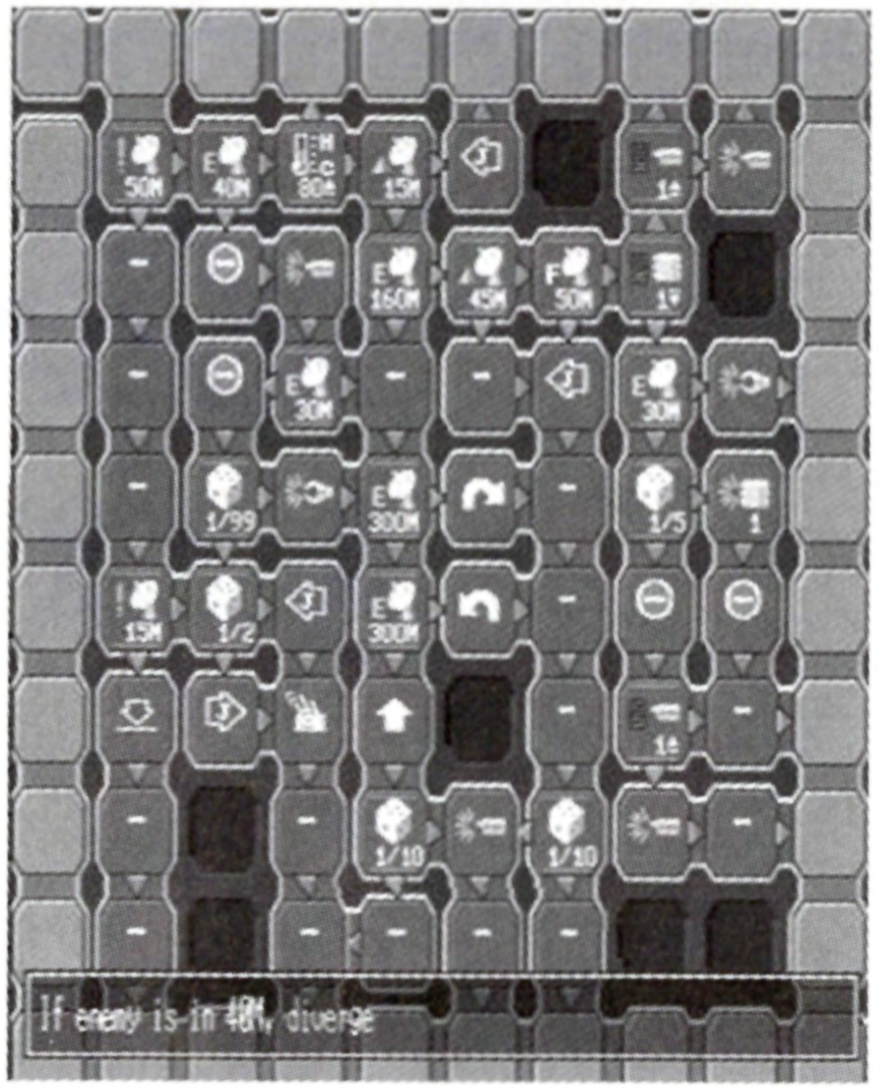
Endless right turns after losing sight of the enemy are also not very productive...Not only the auto programs, but any program may not work well at first. Keep making improvements!



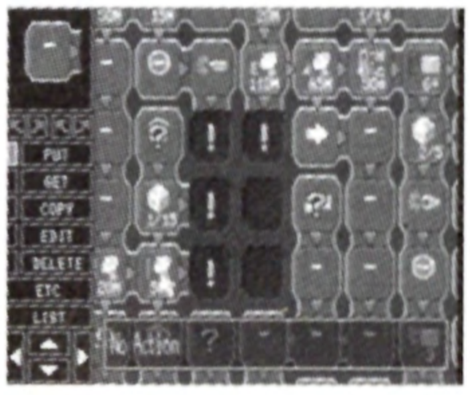
Can the Auto Programs Be Improved?

However, the problems described here are not limited to the auto programs. Without exception, the programs created by beginners have one or more of these problems. So it is exactly solving these problems that is the most difficult thing about creating a program.

So how can these problems be solved? On the right is an example of an improvement to the auto program. It is a little better than it was. However, the auto programs have a complex structure that is not easily modified, and even this improved program cannot be called particularly strong. Ultimately, there is no alternative but to build a strong program on your own.

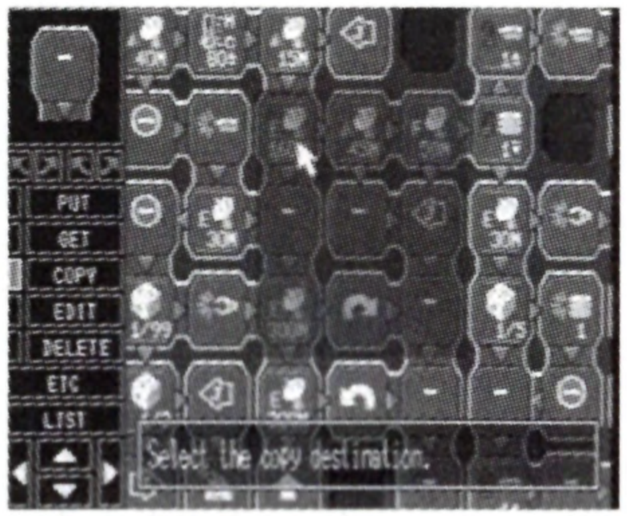


The auto programs not only take up the entire chip table but have a complex structure that cannot easily be modified as the various components are entwined with each other.



Creating a Program

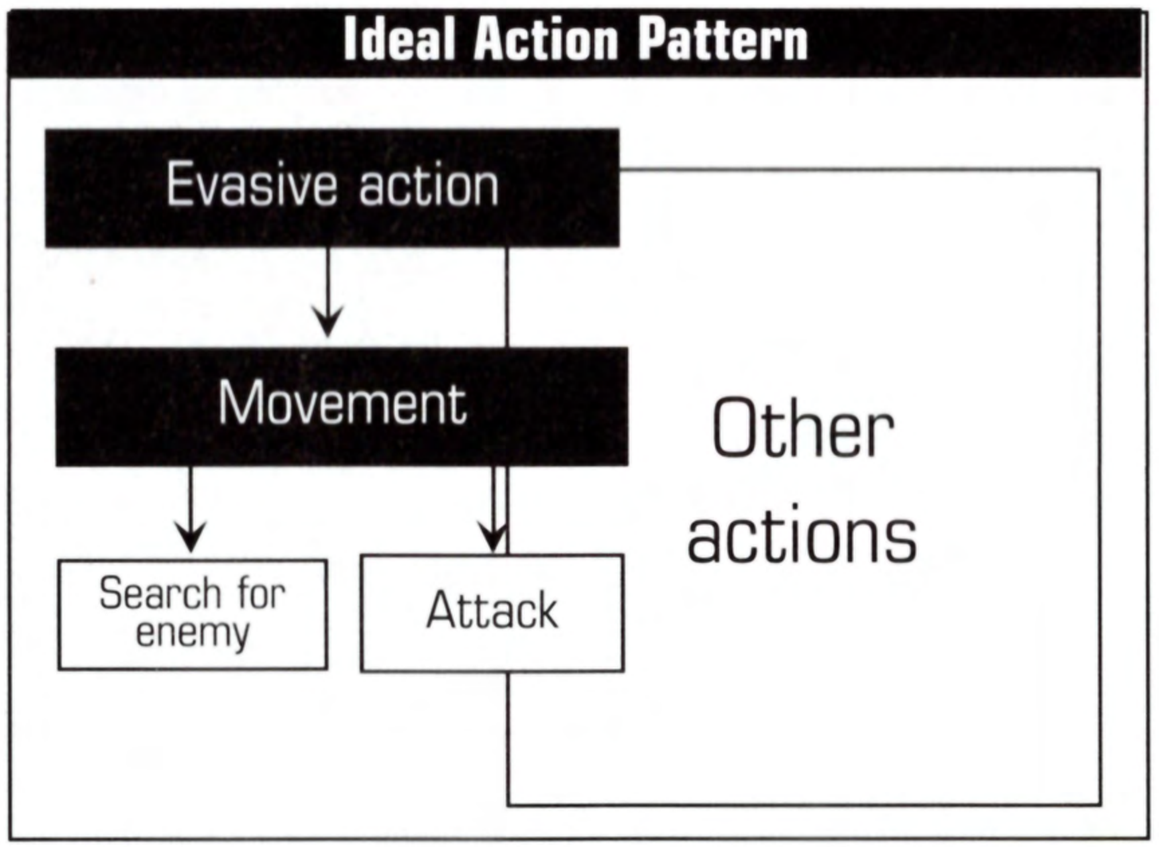
We will now describe the creation of a program, the very essence of this game. Although creating programs is not very difficult, you may be confused at the start. Remember, learn by doing. Note that the explanation in this section is mainly concerned with programs for two-leg OKEs.



Create the Program in an Action-Based Manner

When putting together a program, we recommend that you think of the program divided into portions based on the minimum of battle actions (movement, evasive action, attack, search for enemies) required for the Over Kill Engine to be able to fight well. The advantage of this method is that the program is easy to create and easy to improve. Taking the example of the auto program on the previous pages, programs with various elements intertwined can be very difficult to modify. In this regard, by putting together a program in an action-based manner, improvements are relatively simple to make, and also, the programs can be easily ported to a new CPU with a larger capacity that may appear as the game progresses.

So what to do about the sequence of actions? While this varies from person to person, considering that in a single-player scenario game, you will need to fight many battles one after another, saving your strength (by putting evasive action first) as in the sequence below may be appropriate.



Move
Movement is the only way to approach the enemy. Note that the degree of completion of a program, regardless of whether it is strong or weak, can be judged by how well it moves. For details, see page 15.

Evade
Unless you evade the opponent's attacks, an otherwise winnable battle cannot be won. The main evasive action is the jump. For details, see page 15.

Attack
In a certain sense, attacking is very simple. However, creating a satisfactory attack can be extremely difficult. It can test the sensibility of its creator. For details, see page 16.

Search for the Enemy
Searching for the enemy can have two meanings here. One is actually searching for the enemy. The other is determining the firing range. For details, see page 16.

Other Actions
It is important to learn efficient methods of using the status check chips efficiently, along with how to use the counters and channels and how to make good use of the other special chips. For details, see page 19.

Movement

Avoiding obstacles and moving independently of your other OKEs is important.
Building the perfect program is terribly difficult.

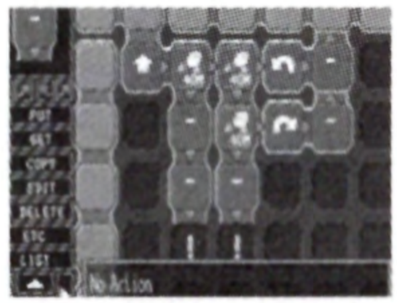
At Least, Try to Walk without Running into Obstacles
Movement is done to approach or avoid enemies. It is particularly important for OKEs that fight mainly with hand-to-hand combat.

One thing that is particularly difficult about movement is how to avoid obstacles. You must use one of the movements of jump, turn or move laterally. Using a jump is reliable but remember that it may get you into trouble depending on the circumstances.

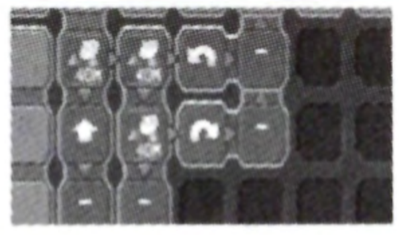
Obstacle Searching Range and Avoidance Methods

Now going on to a real program, basically this is simple. Program A below is an example of a poor design often seen in beginner's work. Since the forward motion comes first, even if a turn to the right is made to avoid an obstacle, the forward motion starts immediately, resulting in a repetitive jerky motion. Program B starts the forward motion only when no obstacle is present, smoothly avoiding obstacles. Program C is an improvement on this. These examples show avoiding obstacles, but the same thinking applies to avoiding your other OKEs.

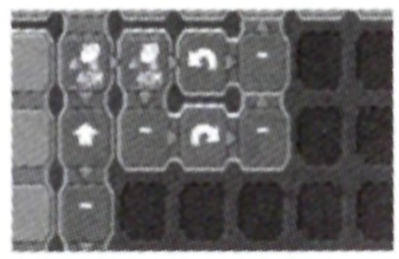
[A] Since the forward movement comes first, obstacles are not easily avoided. Note that A searches over 180° in the forward direction, B searches 90° to the right and C searches 90° to the left.



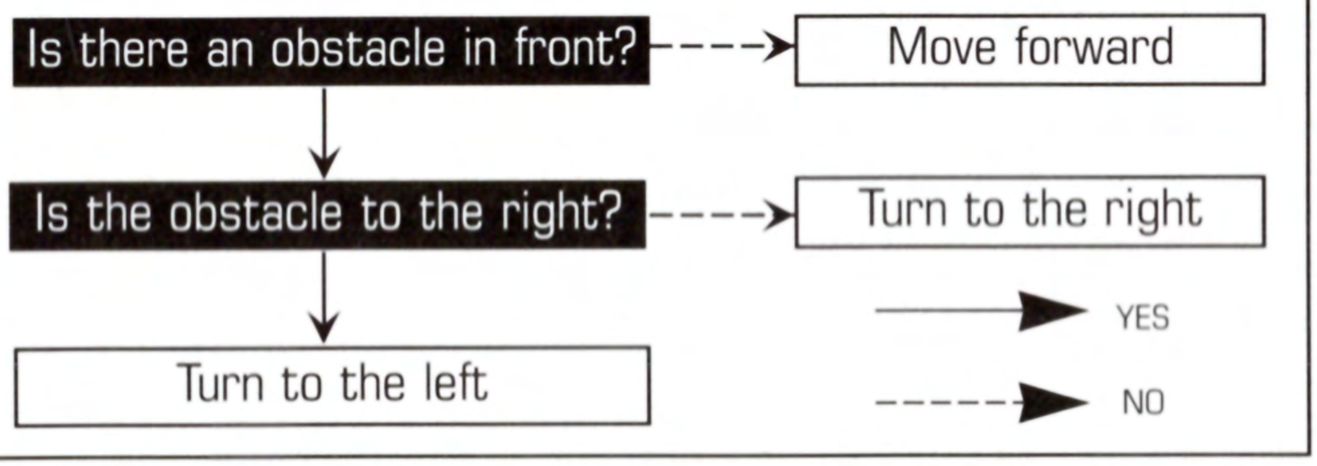
[B] This moves forward only when there is no obstacle present. Obstacles are avoided smoothly if no other commands are encountered.

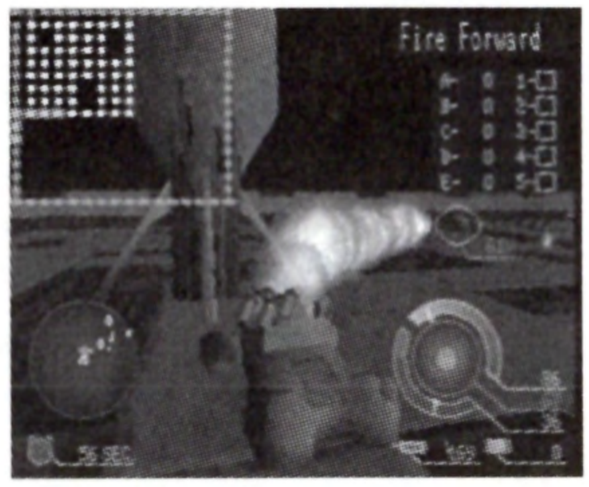


[C] An improvement to B. If the obstacle is not on the right, it must be on the left, so there is no need to take the time to search for it.



Movement flowchart

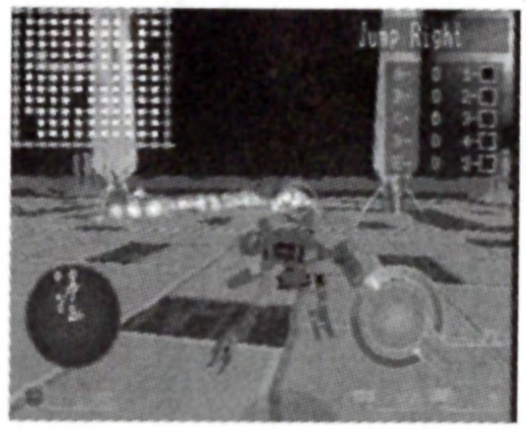




The "Stay Put" Strategy

If the object is simply to take out enemies, there is no real need to move. After all, the enemy will come to you. This is where the "stay put" strategy comes in. However, you should definitely put together a solid strategy to avoid your other OKEs. Even if you avoid your other OKEs during attacks, they will still be hit to a certain extent (especially with the shotgun and the like). Your OKEs should be spread out as much as possible.

Evasion



The evasive actions used differ greatly between the various types of OKE. Here we will describe the two-leg OKE as an example.

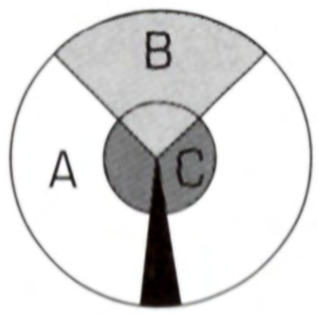
Try to Evade All Attacks

No matter how thick your armor, you will not win unless you avoid enemy attacks. Also, the battle has a time limit and when the time is up, the winner is determined by the total amount of damage inflicted. In other words, even if you don't pay much attention to attacking, as long as you do not suffer damage, there is still the possibility of victory. You can consider putting together a good evasive action plan one method of building a strong OKE.

Change Evasive Actions Depending on the Distance and Angle

In evasion, the important thing is that the evasive action be changed depending on the situation. The possible evasion actions may include jumping or ducking, but for example, jumps involve a certain time lag from when the jump chip is executed before the actual jump occurs. Therefore, you will often not be able to jump in time to avoid a projectile that has come into short range. In this case, ducking is safer than jumping because there is less of a time lag (more accurately, there is less movement). Moreover, rather than merely jumping to evade, you should change your evasive action depending on whether the projectile is coming from the front or sides. For example, jump to the right to evade a projectile from the front or jump forward to evade one from the side.

Example of Setting the Search Scope

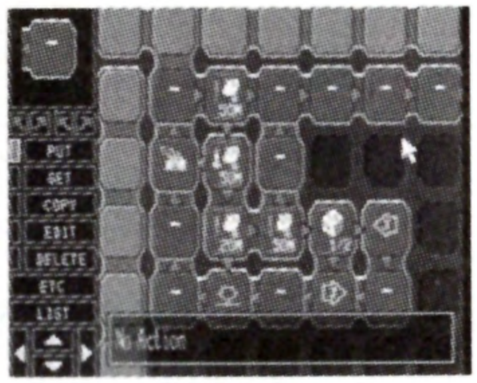


If a projectile is present in A, get into evasive action mode, if in B jump to the left or right, and if in C, duck. Note that the black portion is the so-called "blind spot." You must use two chips if you want to eliminate the blind spot. Set the distance for evasive action to, say, 50m for the outside circle and 20m for the inside circle.

Sample evasive action program. Although extremely simple, even this program can do a good job of evasive action. Note that A searches to 50m over the full scope, B searches to 20m over the full scope and C searches forward to 50m. When searching several times while changing the search scope to different ranges as in this example, be sure to search starting with the shorter range. If you start from the longer range, doing searches at different ranges is meaningless. Take care, as this is easily forgotten.

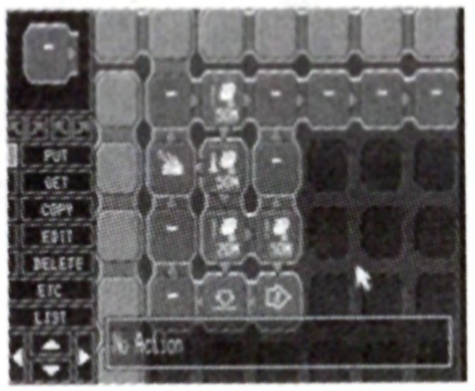


Missiles are Projectiles Too!



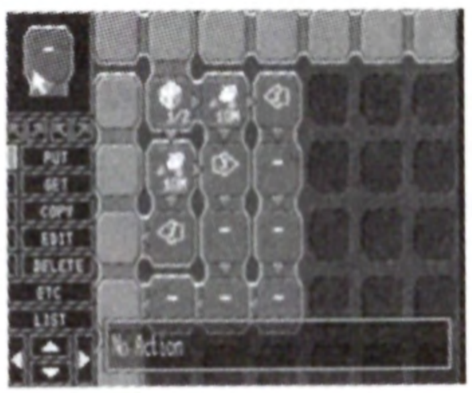
Many people may be confused by this, but missiles are also handled as projectiles. So you may ask why there is a special missile search chip, but this chip is used to identify only missiles in order to perform missile jamming using an Interference Device such as the Omni P1 ID. Thus, depending on the type of OKE, there may be no real need to use the missile search. In the extreme case, you could assume that when a projectile is coming, it is always a missile and try jamming even without using the missile search.

Use a Certain Amount of Randomness in Evading Attacks



Once you have managed to put together a reasonably good program, if you use the same program for all of the OKEs in your unit, you may suffer a fate in which all of your OKEs will move in unison and quickly get crowded together on the edge of the battlefield. Although it can be said that the same program will naturally give the same results, the main reason for this happening lies in the method of evasive action. For example, if a projectile coming from the front is evaded by jumping to the right, a continued attack from the front will result in all of the OKEs being pushed to the right side of the battlefield. A quick method of avoiding this is to add randomness to the method of evasive action. In addition, the addition of randomness also has the effect of spreading out your OKEs.

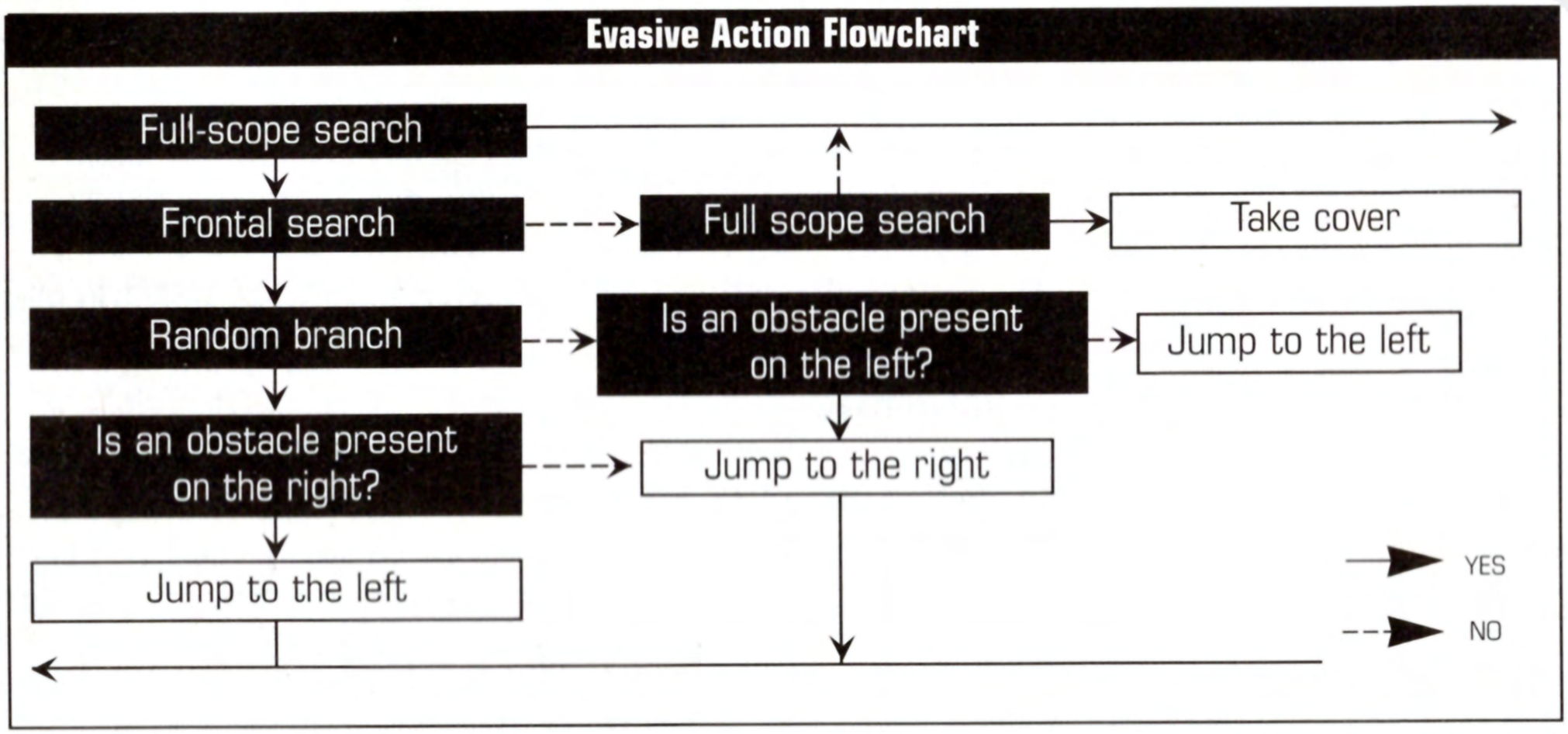
Watch out for Obstacles During Evasive Action with Jumps



One more thing that you should be aware of with evasive action using jumps in the presence of obstacles in the surroundings. If an obstacle is present on the right at the time of evasive action using a jump to the right, you will crash into the obstacle and not be able to evade the attack. To avoid this, add an obstacle search to the previous program. This will make the OKE jump in the direction away from the obstacle.



Example of an evasive action program that includes randomness. When a projectile is coming from the front, the jump goes to either the left or right on a probability of 1 out of 2. Note that after the left jump a right jump chip is encountered, but this right jump is ignored as being unexecutable. Details of such unexecutable chips will be explained later.



Evasive Actions Change Greatly Depending on the OKE Type

The place where programs differ most depending on the type of OKE is in evasive action. For example, Tank OKEs naturally cannot jump or duck and are unable to move laterally. They can only try to evade by moving forward or backward. In addition, you should try to change the evasive action somewhat depending on the type of weapon that the opponent is using. For example, considering that shotgun blasts spread out, it is safer to use a larger search range.

Attack

There are No Rules for Attacks. Originality is the Key.

Perhaps the part that most differentiates players is their approach to attacks. Attacks can be put together in so many different ways that there is no single theory for attacks. So maybe we won't try, period. But that would be irresponsible of us. So we will at least present a minimum of rules to abide by. Try some original approaches.

Minimizing Losses is Most Important

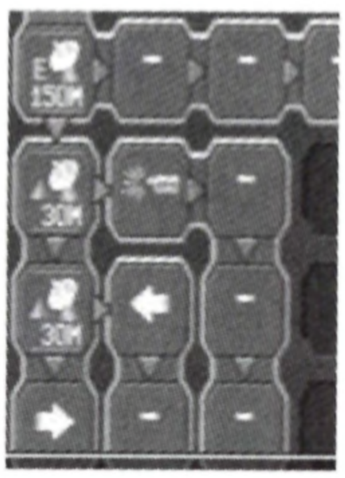
If you just want to attack the enemy, there is no need to create an attack program. The main and sub weapon firing chips themselves are extremely intelligent, so that once an enemy is sighted within 200m, a single chip on the table will cause the OKE to point in its direction and attack. So in an attack program, the most important parts are keeping from shooting obstacles and your other OKEs. Moreover, changing the attack method depending on the situation is one of the important roles of an attack program.

Avoid Shooting Obstacles and Your Other OKEs

In order to avoid shooting obstacles and your other OKEs, the most effective method is probably performing searches for both of these before an attack and halting the attack if either is found. The program below is an example of such a program. This program will certainly cut down on wasteful attacks. However, you should take the attitude that a certain level of attacks on your other OKEs or obstacles cannot be avoided. If obstacle searches are unnecessarily broad (wide in scope) there is a risk that it may become impossible to shoot at all in an area with a large number of obstacles.

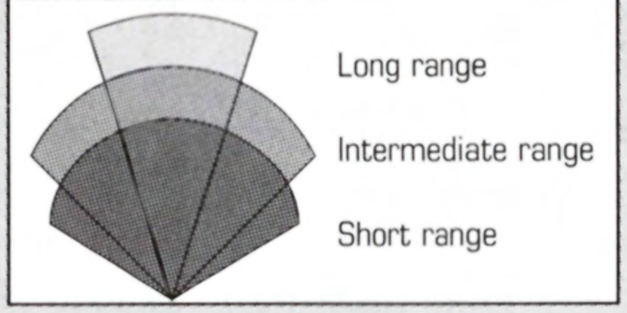


If there is an obstacle in front, do not attack. This is the basic method of avoiding obstacles. The same logic applies to your other OKEs.



This is an extension of the above program. If an obstacle is present, avoid it to the left or right depending on its position. Note that in this situation, an enemy is certain to be near so jumps should be avoided. This is because a jump leaves the OKE very vulnerable.

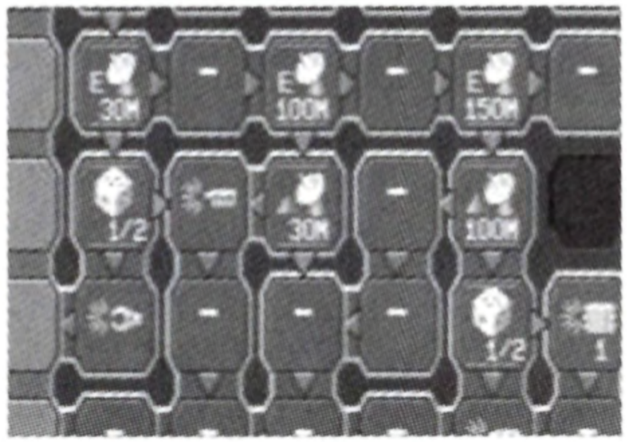
Example of Attack Ranges



By using three stages of enemy search ranges, obstacle avoidance and switching of attack types become possible. By dividing the ranges finely, even higher precision attacks become possible.

Is it Possible to Change Attacks Depending on Distance?

While the attack methods used will differ from person to person, we will present here one example. As described below, the most difficult aspect is determining the search range for avoiding obstacles and your other OKEs. For example, if the range is set to 40m in the frontal direction, even if the enemy is within this range, the OKE will go into evasive action without attacking. So how can this be avoided? By determining the range to the enemy. For example, if the enemy is within 150m, go to attack mode. Here, we will perform searches for the enemy in three stages: within 30m, within 100m and within 150m. When an enemy is more than 30m away but less than 100m away, the obstacle search can be performed to a distance of 30m. In addition, using this method permits one to change the type of attack depending on the distance. For example, hand-to-hand combat at 30m, attacks with the main and sub weapons at 100m, and a diversionary attack with the main weapon at longer range. Try this when you have acquired a large-capacity CPU.



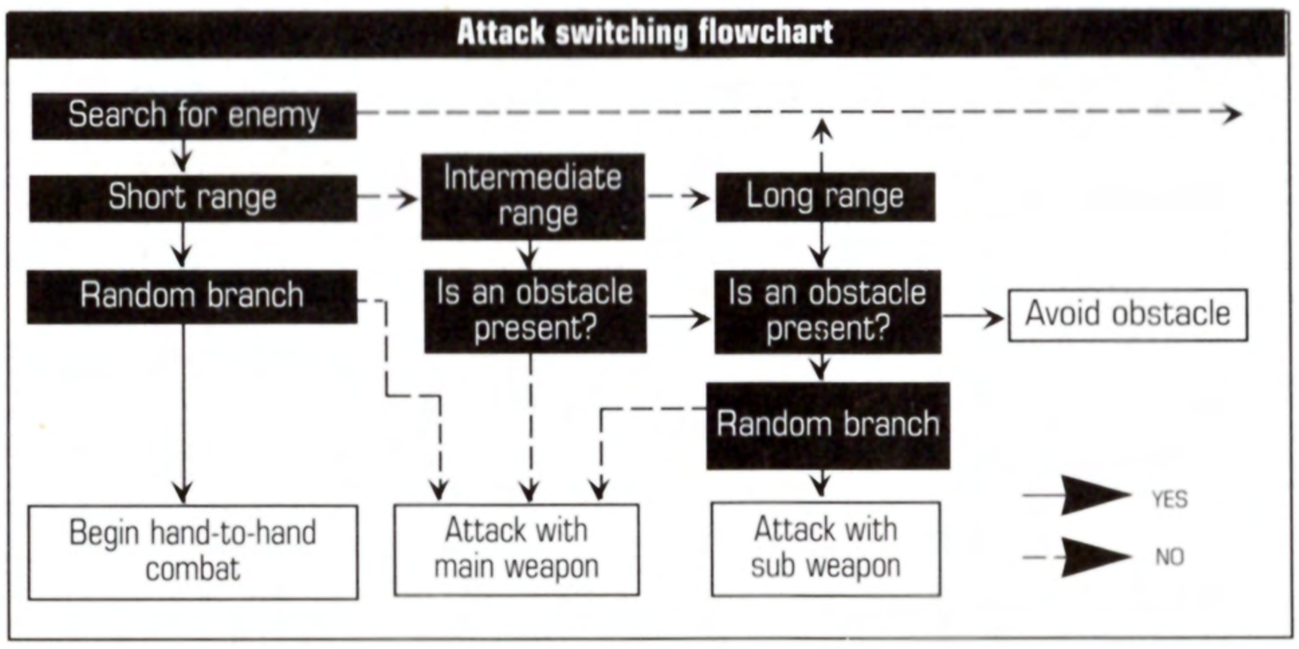
A program for switching attacks depending on the distance. Searching for allies can be performed in the same manner as for obstacles.

Which Enemy is Targeted?

The attack chip targets the enemy lying closest to the centerline (0° angle) of the search range. In other words, the enemy closest to being directly in front of your OKE is the highest priority target. The enemy search chips also react to the enemy closest to the centerline.



Attack switching flowchart



Differences in Attack Settings Depending on the Weapon Used

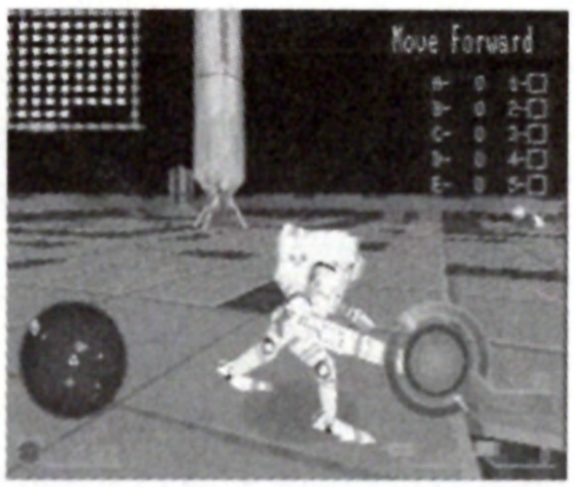
As with evasive action, the attack range should be changed depending on the weapon used. Each weapon has an effective range. So try to adjust the settings to take advantage of the effective range. Generally speaking, the shotgun is effective up to roughly 140m (but best at about 80m). The assault rifle is about 170m. The laser reaches the limit of the firing chip at 200m. It is probably best to search for the enemy and attack at about these distances.

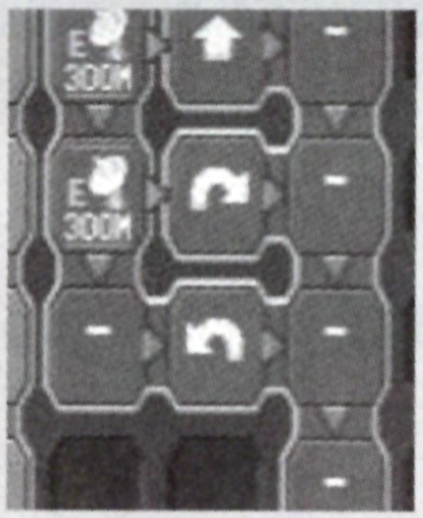
Searching for the Enemy

Searching for the enemy is used mainly for adjusting the range. This is important when using the shotgun and other weapons.

Unnecessary at the Start, Searching Becomes More Important with Strategy

Searching for the enemy is used for searching for an enemy out of firing range or for determining the range to the enemy prior to an attack. Because of this quality, you may not use the enemy search chip at the beginning, but it's an important aspect of many strategies. Here we present mainly its use in searching for the enemy, but possibly the most important use is for determining the distance to the enemy.





Sample enemy search program.

Searches to the front and right are made and if there is no response, the enemy is to the left, so turn to the left unconditionally.

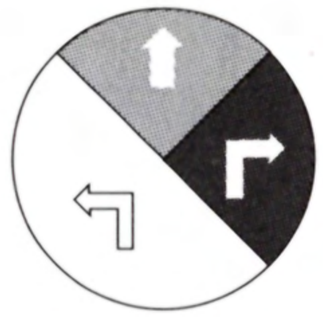
Set so that the OKE Points in the Direction of the Enemy



Searching for the position of an enemy out of firing range and pointing the OKE in that direction is the work of the enemy search chips. From a programming standpoint, their usage is extremely simple; move forward if the enemy is in front, turn to the right if on the right and turn to the left if on the left. Although narrowing the search scope in the forward direction permits the OKE to advance on the enemy more accurately, the settings should include a certain amount of leeway. If it is too narrow, depending on the situation, a conflict with the processing of the obstacle avoidance and other programs may occur, leaving the OKE unable to move.

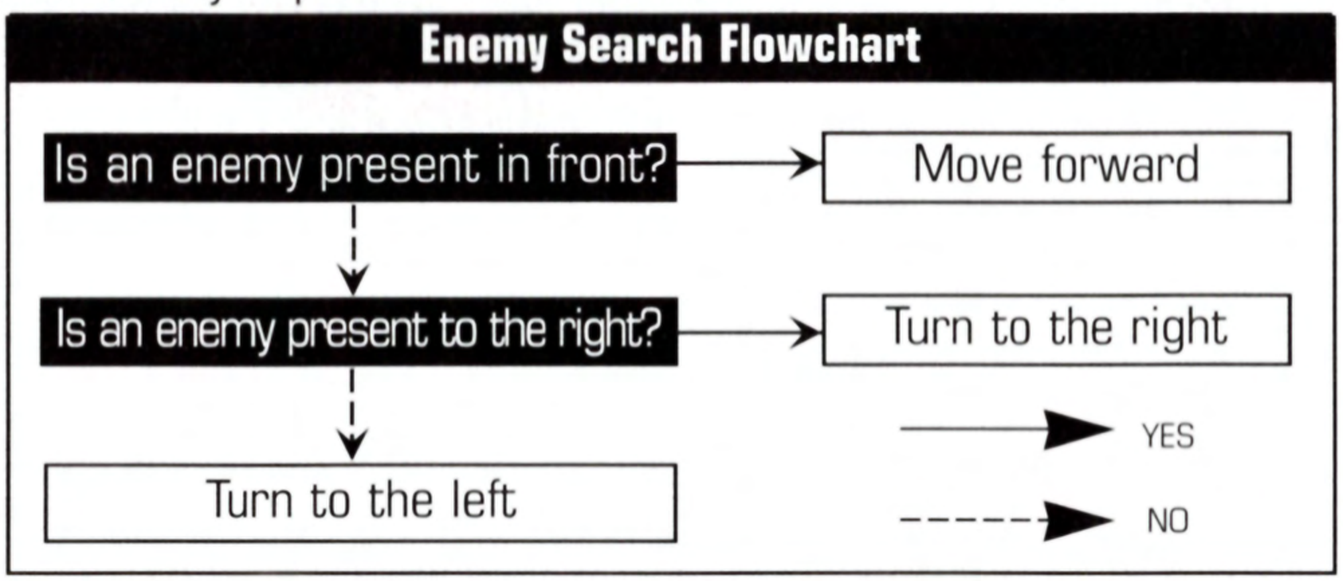
Example of the Enemy Search Scope

The search scope should probably be the same as the field of fire.
Setting the search range to the maximum distance of 300m is safe.



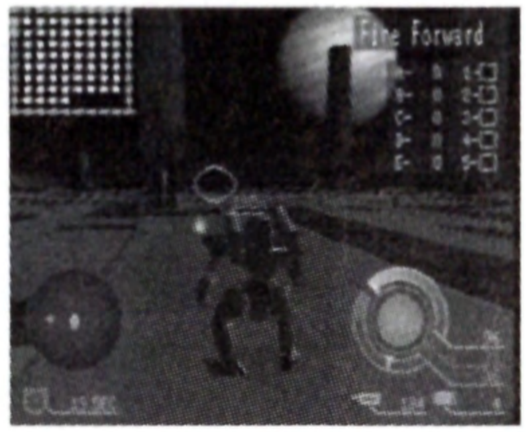
Searching for the Enemy Has a Wide Range of Uses

Although this also involves movement, if used well, searching for the enemy can also be incorporated into a program that moves the OKE while maintaining a constant distance from the enemy. This is very important for tank OKEs.



Searching for the Enemy by Attacking

Searching for the enemy need not be done by only the enemy search chips. The main and sub weapons firing chips also have the same capabilities as the enemy search chips, but with a range fixed to 200m. If you can afford to waste some shots, this method can also be used to search for the enemy.



Other



If you have learned all that we have presented so far, consider yourself to have graduated from the ranks of beginners. Now try the following two techniques.

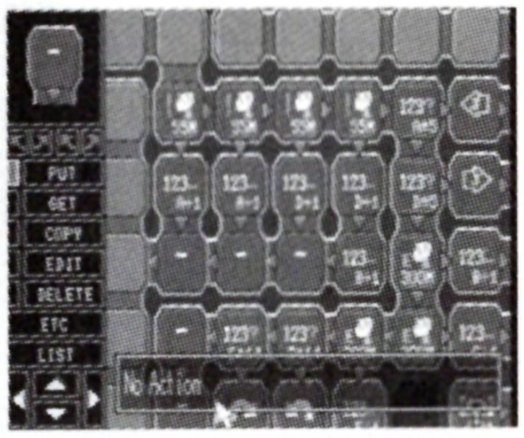
Always Check the State of Your OKE

Although some may think that the status check chips are not very important, they are important elements neither superior nor inferior to the other elements. The status check chips determine the amount of ammunition remaining, temperature of the OKE, damage to the OKE, number of your OKEs remaining and other information. The most important of these is the check on the amount of ammunition remaining. Always check the amount of ammunition remaining before firing each weapon. There is nothing more embarrassing than pulling the trigger with no round in the chamber. The temperature of the OKE should also be checked regularly as the efficiency of radiation of heat drops the thicker the armor worn. Although somewhat different, the time check chip can be used in interesting ways. It can be used effectively together with checking the number of your OKEs remaining. One use of the OKE damage check could be to launch all remaining sub weapons when the damage level reaches 80% or greater.

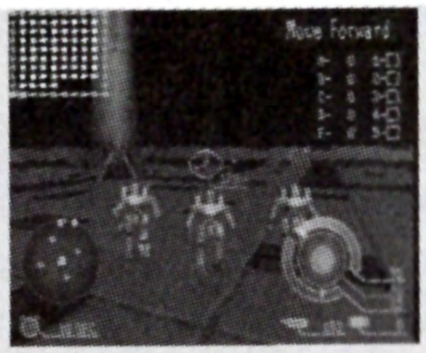
How are the Counters and Channels Used?

The counters and channels are the most difficult chips to use. You will have to think of effective ways of using these chips on your own, but the counters are very useful. For example, when you are fighting in a single-player scenario game, a counter can be set up so that the counter is incremented each time a sub weapon is fired and then the firing is halted when four shots have been fired. This is useful if you do not want to use up all of your sub weapons considering that you may need to fight two times in a row. Another interesting use would be to loop the projectile check at a search range of 50m, incrementing a counter each time it goes around, moving to evasive action when the counter goes above a certain value. This lets you know if a projectile is approaching your OKE. The possibilities of use are endless if you use your imagination.

Now we come to the channels. They can be used to send commands to your other OKEs but using them well is very difficult. A relatively effective method of use would be to launch sub weapons (particularly landmines or other scattered weapons) in sequence to spread them over a wide area.... Try different things and find an effective use for them.



Counters can be used for a wide range of uses, limited only by the imagination. One could be in predicting the position of the enemy OKEs from the direction of the projectiles without using the enemy search at all.



Channels also can have interesting uses, but using them well can be difficult. Try using them to move three OKEs in formation.

The Theory of Program Creation

By now, you are probably familiar with the basics of the action-based programming technique. With this, your Over Kill Engine should be able to fight, but here is where it gets really difficult and interesting.

What Makes a Strong Program?

Theoretically speaking, to create a strong program you must first figure out what makes a program strong. If you know this, then you need merely put together a program that satisfies those conditions. So what is a strong program?

First, it avoids enemy attacks. Second, it reliably attacks only the enemy. Finally, it fights from a position of strength. There would be no end if we went into too much detail, but broadly speaking it is probably these three conditions that make a program strong. You may think "It can't be that easy, or I wouldn't be working so hard!" But when putting together a program, if you think about achieving these three elements, just this will let you create a strong program. We recommend that you give your program at least one clear, distinctive characteristic, whatever that may be.



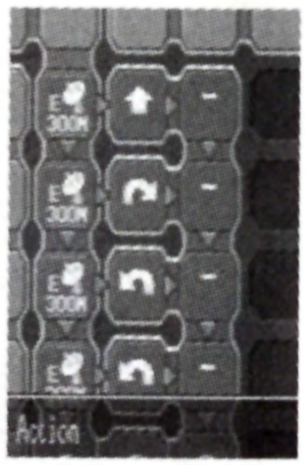
Putting a Program Together Efficiently

After strength, think of what is the ideal form for putting together a program. The action-based form presented here is ideal in the sense that it can be easily modified later, but it also has a major weakness in that the program tends to get bloated in size. Although this depends on the CPU speed, the number of chips executed per second is fixed. For this reason, the more compact the program, the faster the program itself is processed and the more accurate its movements become. In addition, the standard flow of processing (attacks, etc.) should flow smoothly from the upper left of the chip table to the lower right.

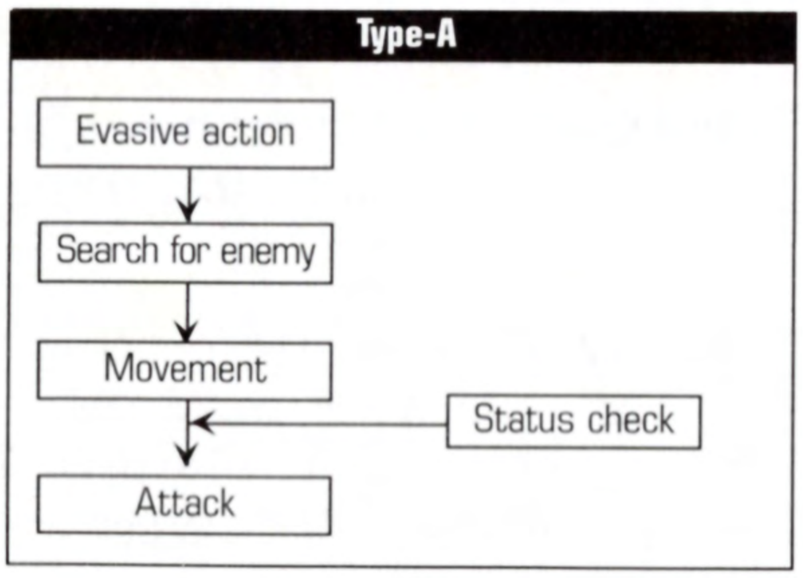


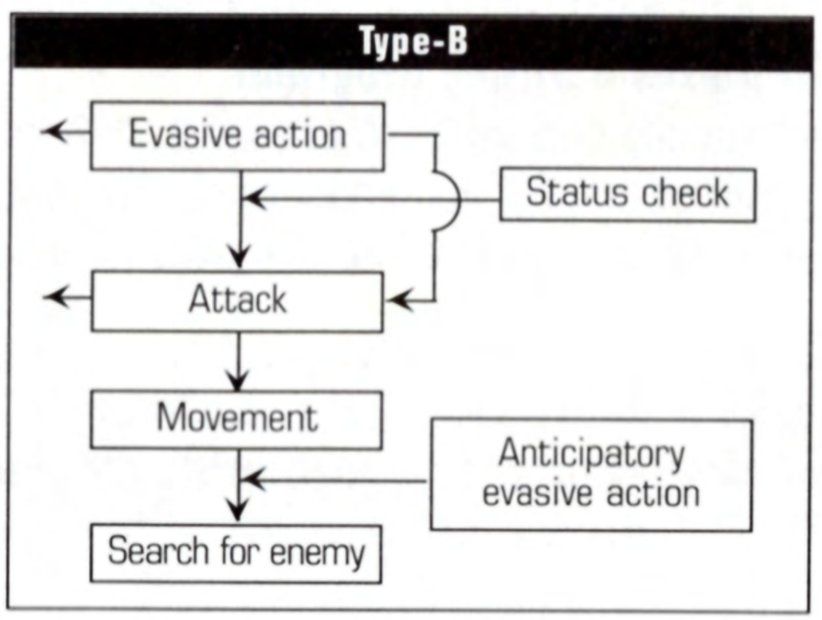
What is the Best Sequence of Actions?

The sequence in which the various actions are processed has a major effect on the strength (or more accurately, the character) of the program. In addition, whether the various actions are all processed linearly or whether the sequence branches depending on the circumstances will also have a major effect. Type A below shows linear processing. All of the actions, starting with evasive action and on to movement and attack, are performed in a single sequence. Type B branches the processing depending on the situation, and the processing of a single sequence ends when the processing of the destination of the branch ends. Either method has advantages and disadvantages but Type B may be said to be more combat-oriented.

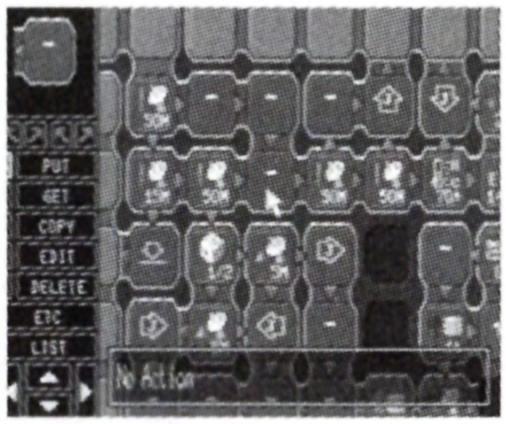


A Smaller Program will Result in Faster Processing.
Making the main flow of processing as straight as possible is also an effective method.

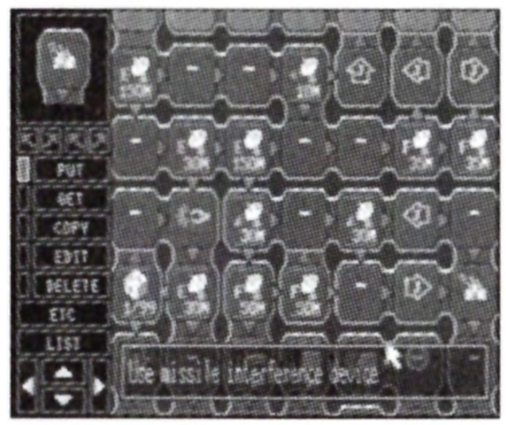




Getting back to the sequence of actions, one important decision to make is whether evasive action is to be put first or whether the attack is to be put first. Although this also depends on subsequent processing, putting the attack first generally makes the program surprisingly aggressive. Try it.



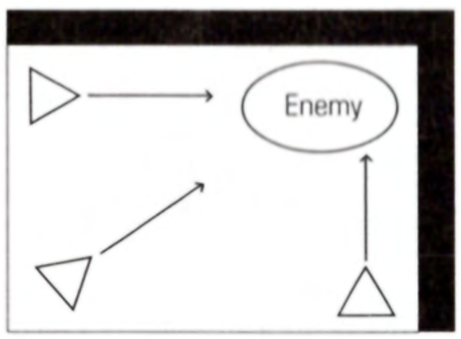
Evasive Action First
Considering the series of battles to be fought in a single-player game, a program that puts evasive action first may be able to survive longer.



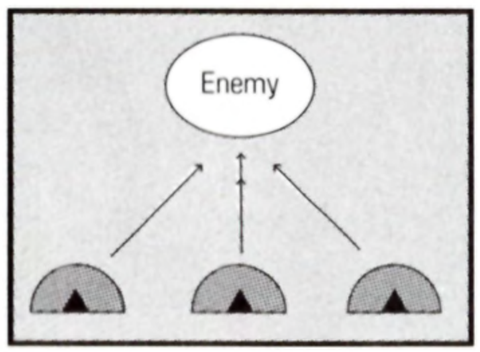
Attack First
In contrast, with a single battle to be fought in Vs. Battle, putting the attack first for an aggressive fighting stance may be better.

Aim for a Strategically Beautiful Battle Plan

If you know enough information about your opponent in battle, just winning should not be so difficult. For example, in a single-player game, you could come out with an Over Kill Engine loaded with lots of large missiles and launch them all at the start of battle. With this, you will almost certainly win. However, this is not very interesting and not much fun either. It is more interesting to try to make the OKEs fight beautifully and exactly as you want. For example, imagine a formation of three OKEs. If you can execute a battle plan in which three of your OKEs corner the enemy while keeping track of each other's positions, wouldn't that be beautiful? Anyway, this game lets you put together many different types of programs. If you can win but are still not satisfied, try creating a stronger, more beautiful program.



An attack that backs the enemy into the corner of the map. This can be done fairly simply by putting together a different movement program for each of your OKEs.



The worst is when many of your OKEs get stuck in the same place. The best is when the OKEs are spread out at equal intervals and are able to methodically hunt down the enemy.

On the previous page, we finished our general description of programs assuming the case of a two-leg OKE. You should now be almost completely familiar with the basics of programming and how to put them together. Now we will present programming techniques for other types of OKEs, as seen in the programs for the enemy side.



Tanks have the smallest selection of usable chips of any type of OKE.

Fighting Styles Differ Greatly Depending on the Type of OKE

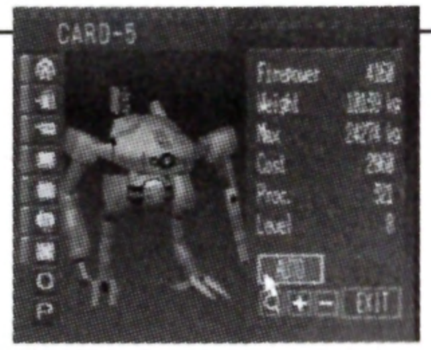
There are basically four types of OKEs. These four are: two-leg, multi-leg, tank and flying. Many aspects of programming, from the method of putting the program together to the usage of weapons and strategy, will vary greatly depending on the type of hardware used. For example, the chips used for the evasive actions of jump or duck cannot be used for tanks. In other words, there is no choice but to change.

How Well Can Programs Be Adapted to Other Types of OKEs?

So is it possible to adapt a program for use in another type of OKE? It should be no problem going from a two-leg to a multi-leg OKE. However, tanks and flying OKEs need completely original programs. Try creating the program using the programs for the enemy side as a reference.

The programs for two-leg OKEs can be adapted for multi-leg OKEs to a certain extent, but it is better to create a new program from scratch. In short, it is better not to think about adapting programs for other types of OKEs.

Two-Leg OKE
As we have already described two-leg OKEs in the previous sections, we will not cover them here. You will have to try to imagine the program used by the enemy side.



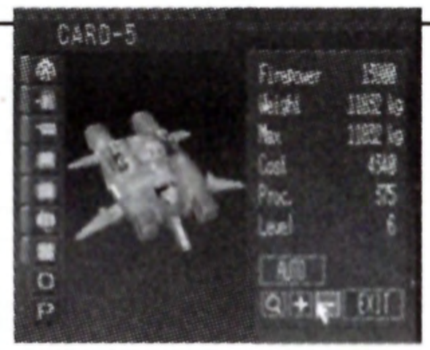
Multi-Leg OKE
The upper portion is a gun turret, so the field of fire is wider than that of the two-leg OKEs, and its powerful equipment is also attractive. However, the enormous body makes evasive action difficult.



Tank OKE
The high engine efficiency means that more equipment can be carried. However, evasive action is practically impossible. So the crucial matter is how long the tank can tough it out behind its thick armor.



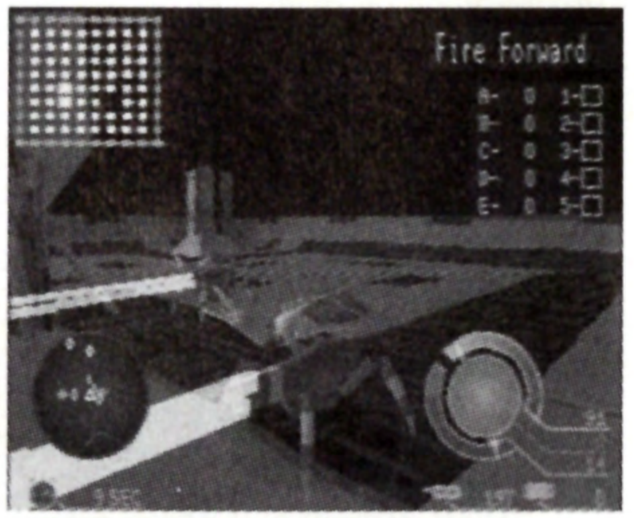
Flying OKE
The airplane requires concepts completely different than those seen in programs we have seen so far. It can be powerful if used well, and fun too. Recommended.



CONFIDENTIAL
Multi-Leg OKE

Rogue
Manticore
Livewire

This type can be equipped with powerful armament and various types of equipment.



The two-leg OKEs must turn around to attack enemies to the side, but the multi-leg OKEs can rotate its turret to start an attack.

Use the Wide Field of Fire Effectively

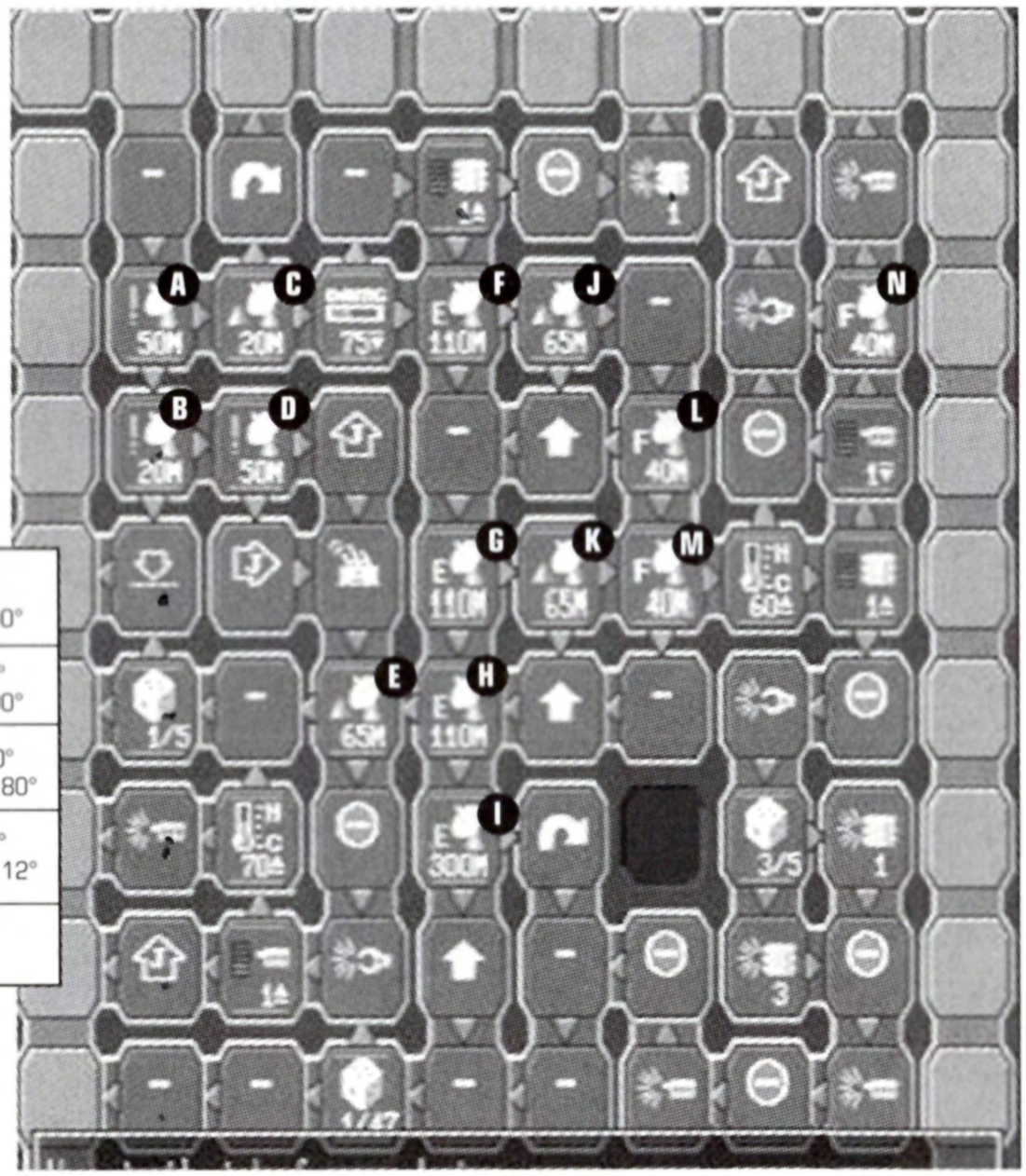
The multi-leg OKE combines the characteristics of the two-leg and tank OKEs. The engine output efficiency is also high, so it can wear very thick armor. In addition, its armaments are quite powerful, so it shows its power solidly in close-up fighting. However, the large size of the OKE makes evading projectiles by jumping more difficult than for the two-leg OKEs, and its slow movement makes it easily cornered. So the basic tactic used by the multi-leg OKE is to shoot while retreating, taking advantage of its wide field of fire. The wide field of fire comes from the upper body of the OKE, the gun turret, which can swing around so that the OKE can attack enemies behind it while moving forward.

Battle Plans for the Multi-Leg OKE as Seen in the Enemy-Side Program

Although we should have mentioned this earlier, just as there are four types of auto programs, one for each type of OKE, there are four types of programs used by the enemy army for each type of OKE. Studying the techniques used in these programs (and also the auto programs) will be very useful when putting together an actual program. The program at right is used for the enemy multi-leg OKE.

This program is distinctive in that it has attack ranges in three directions, forward, right and left. This permits the program to attack enemies to the side even during movement. The connection of a hand-to-hand combat and forward jump chips in the upper right is also a very interesting construction. Take note of it.

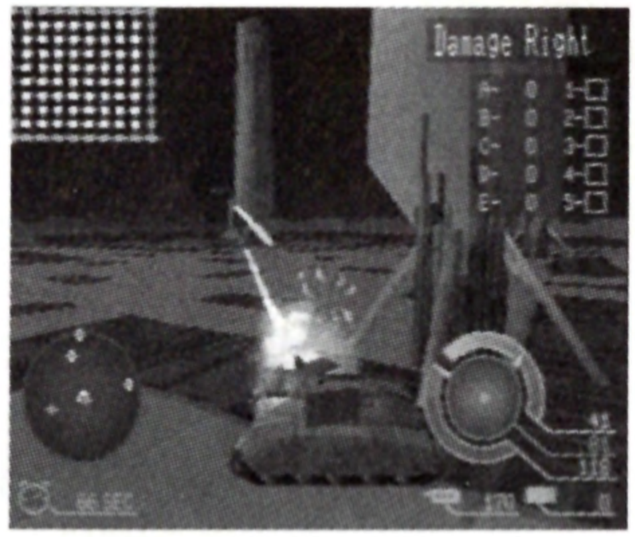
Program for Enemy Multi-leg OKEs



A	50m Axis 0° Scope 354°	B	20m Axis 0° Scope 354°	C	20m Axis 0° Scope 90°
D	50m Axis 0° Scope 95°	E	60m Axis 0° Scope 106°	F	110m Axis 90° Scope 90°
G	110mAxis -90° Scope 90°	H	110m Axis 0° Scope 90°	I	300m Axis 180° Scope 180°
J	65m Axis 90° Scope 90°	K	65m Axis-90° Scope 90°	L	40m Axis 90° Scope 112°
M	40m Axis -90° Scope 112°	N	40m Axis 0° Scope 354°		

CONFIDENTIAL
Tank OKE

Kouger
Jackhammer
Ronin



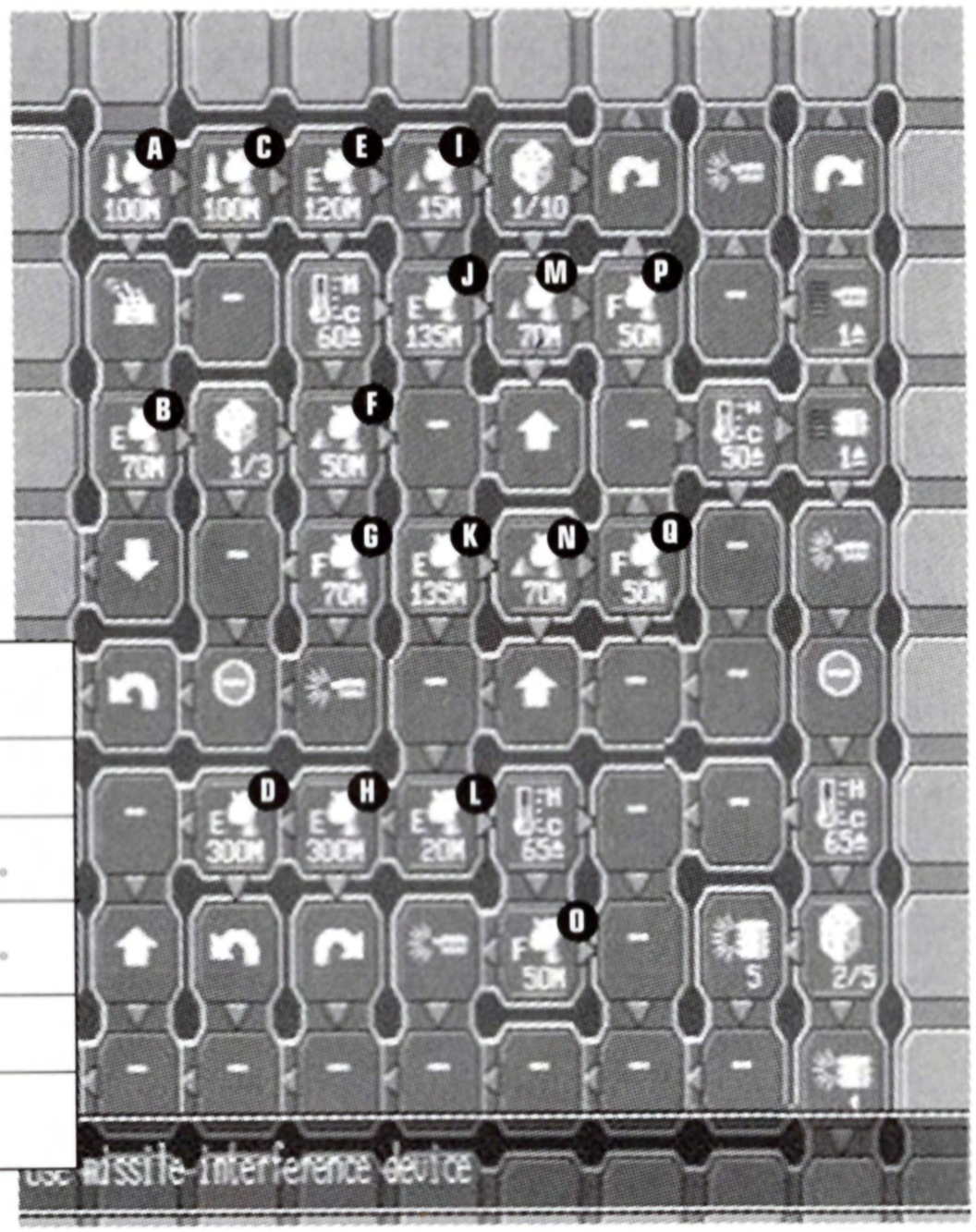
Its unexpectedly speedy movement and stubborn nature make this a really interesting type of OKE.

Let 'em in Too Close, and it's Curtains! Don't Let the Opponent Crowd You. The tank OKE refers to both OKEs with caterpillar treads and ones with wheels. Many people may get the impression of slow-moving tanks, but all of the tank OKEs that appear in this game move very quickly and are fun to watch. The most distinctive feature of the tank OKEs is their high carrying capacity due to the good output efficiency of their engine. Their weapons are also powerful, so they are unequaled in both offensive capacity and defensive capacity. On the other hand, they cannot use the lateral movement, jump, and duck chips. So the most difficult thing is finding ways to evade the opponent's attacks. They also cannot engage in hand-to-hand combat. So they are extremely weak at short range. Against short-range enemies, since they are only able to launch missiles that spread the attack over a wide range, care must be taken to keep the enemy from getting too close. Naturally, it is fine for them to run around and try to escape.

Battle Plans for the Tank OKEs as Seen in the Enemy-Side Program

What is surprising is that no searches for projectiles are used at all. Although jamming is used for missiles, this is quite simple. In other words, the program has virtually no intent to evade attacks. In addition, even though the OKE is equipped with long-range assault rifles and lasers (tank OKEs have no shotguns), the ranges are short at only 120m for frontal enemies and 135m to the side. This may appear to go against reason at first, but it makes the OKE move around the map endlessly. In other words, it avoids the opponent's attacks by constantly moving.

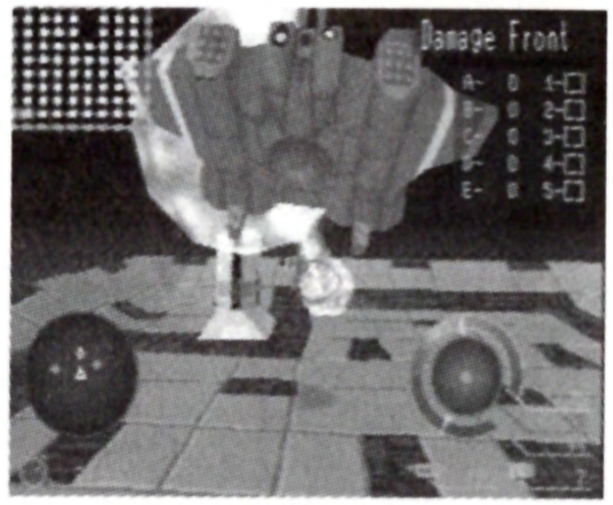
Program for Enemy Tank OKEs



A	100m Axis 90° Scope 90°	B	70m Axis 0° Scope 253°	C	100m Axis -90° Scope 90°
D	300m Axis -140° Scope 45°	E	120m Axis 0° Scope 56°	F	50m Axis 0° Scope 90°
G	70m Axis 0° Scope 61°	H	300m Axis 140° Scope 45°	I	15m Axis 0° Scope 112°
J	135m Axis 90° Scope 90°	K	135m Axis -90° Scope 90°	L	20m Axis 0° Scope 354°
M	70m Axis 90° Scope 84°	N	70m Axis-90° Scope 84°	O	50m Axis 0° Scope 67°
P	50m Axis 90° Scope 90°	Q	50m Axis-90° Scope 90°		

CONFIDENTIAL
Flying OKE

Valiant
IronDeath
Raptor



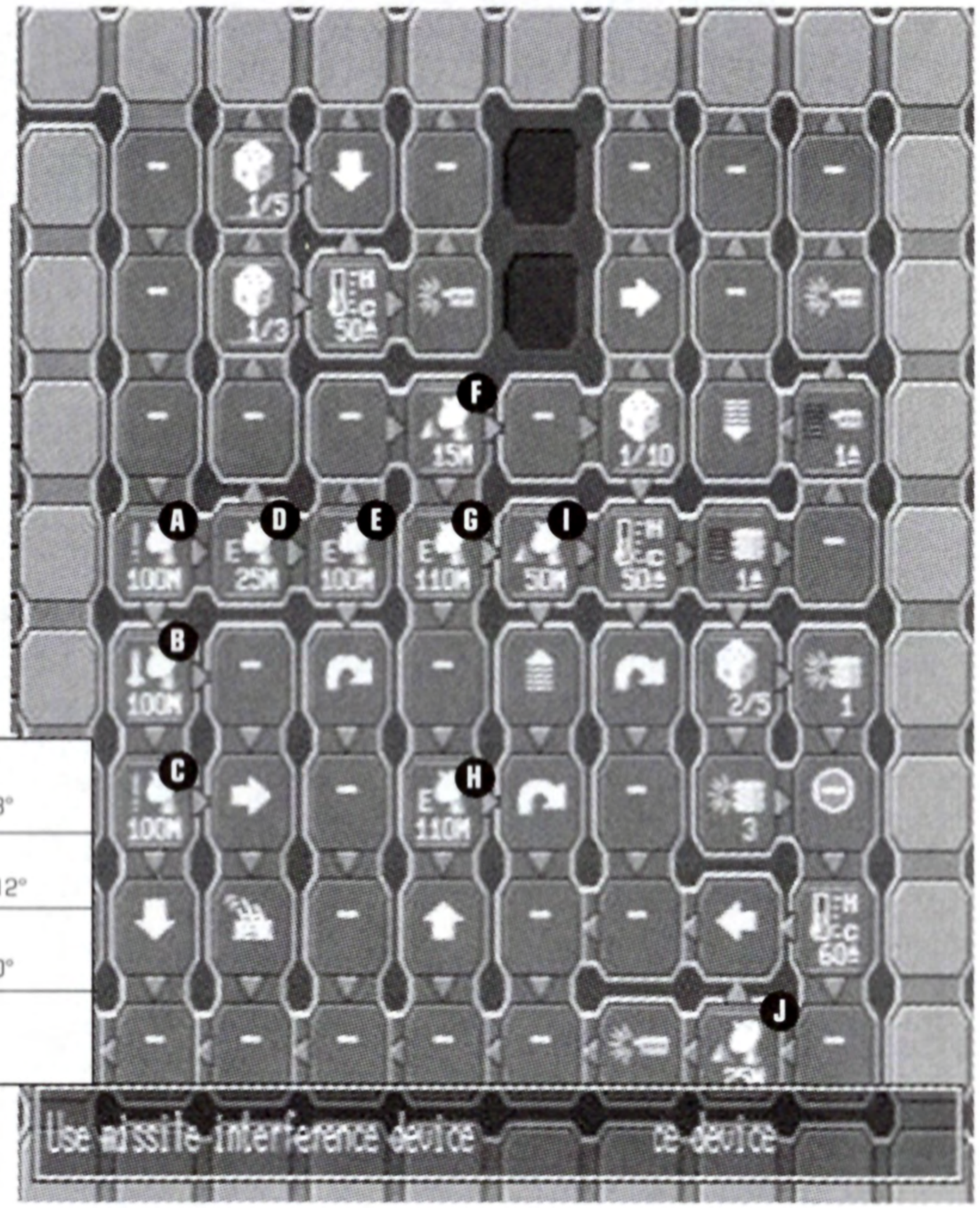
Nothing you have learned about programming so far applies to this OKE. Think of it as a brand-new challenge.

Extremely Thin Armor, but Nimble Movement Covers a Lot of Ground probably because of its low carrying capacity, the flying OKE carries very little armor. Even though you go to the trouble of building one, it may be shot down before you know what's happening. For this reason, when building a flying OKE, you must first pay close attention to evasive action. Naturally, the jump chips cannot be used, so evasive actions must be put together by deftly combining the ascend and descend chips that can be used only by flying OKEs. This said, evasive action is the only area of the flying OKEs that needs special attention, and once this is firmly in place, it should serve as a powerful Over Kill Engine. In particular, during attacks there are basically no obstacles and there is no real need to be concerned about hitting your other OKEs. Why is this? Because, although this depends on the altitude, when attacking from above, there is little chance of hitting your other OKEs even if they are present.

Battle Plans for the flying OKE as Seen in the Enemy-Side Program

Compared to the other enemy-side programs, you will notice how simple the structure is. As you will realize if you have ever actually fought against flying OKEs in the enemy army, they really are weak. The program is not strong at all. No offense. The portion related to movement has an extremely simple structure. Basically, it only points toward the enemy, without performing any searches for obstacles or allies. Very nearly the same goes for when attacking. The strength of the flying OKE lies in that it is not affected by conditions in the surroundings. So it is up to you to take advantage of this strength, but it is also the weak point of the flying OKE. Being exposed as it flies overhead, it is easily targeted by the enemy.

Program for Enemy Flying OKEs



A	100m Axis 0° Scope 354°	B	100m Axis 0° Scope 354°	C	100m Axis 0° Scope 78°
D	25m Axis 0° Scope 354°	E	100m Axis 180° Scope 180°	F	15m Axis 0° Scope 112°
G	110m Axis 0° Scope 90°	H	300m Axis 180° Scope 241°	I	50m Axis 0° Scope 50°
J	25m Axis 0° Scope 225°				

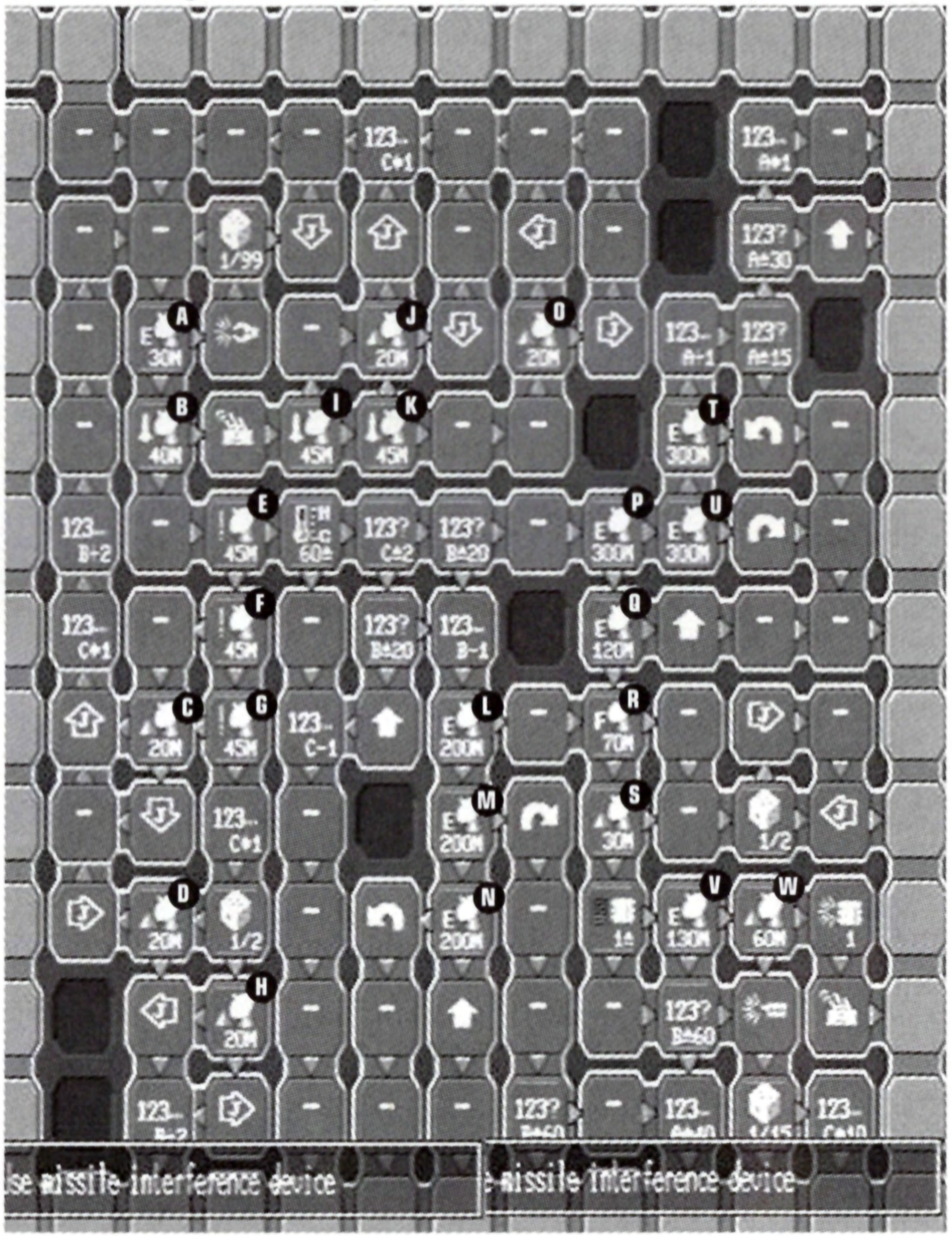


So now we come to the finale of our commentary on programs. We will now present an original program for a two-leg OKE (Prowler) developed specially for this book by the Artdink technical staff. Through this commentary, we hope to present a more profound aspect of the world of programming.

A Novel Program that Makes Liberal Use of Counters

Below is a program developed by the Artdink technical staff. You may be wondering, "What the hell?!" In the first place, almost half of the chips are counter chips. At first glance, you probably will not be able to imagine what the program does. But this program is very, er, frightfully strong. To test its strength, try actually recreating this program and fighting against it, and you will see.

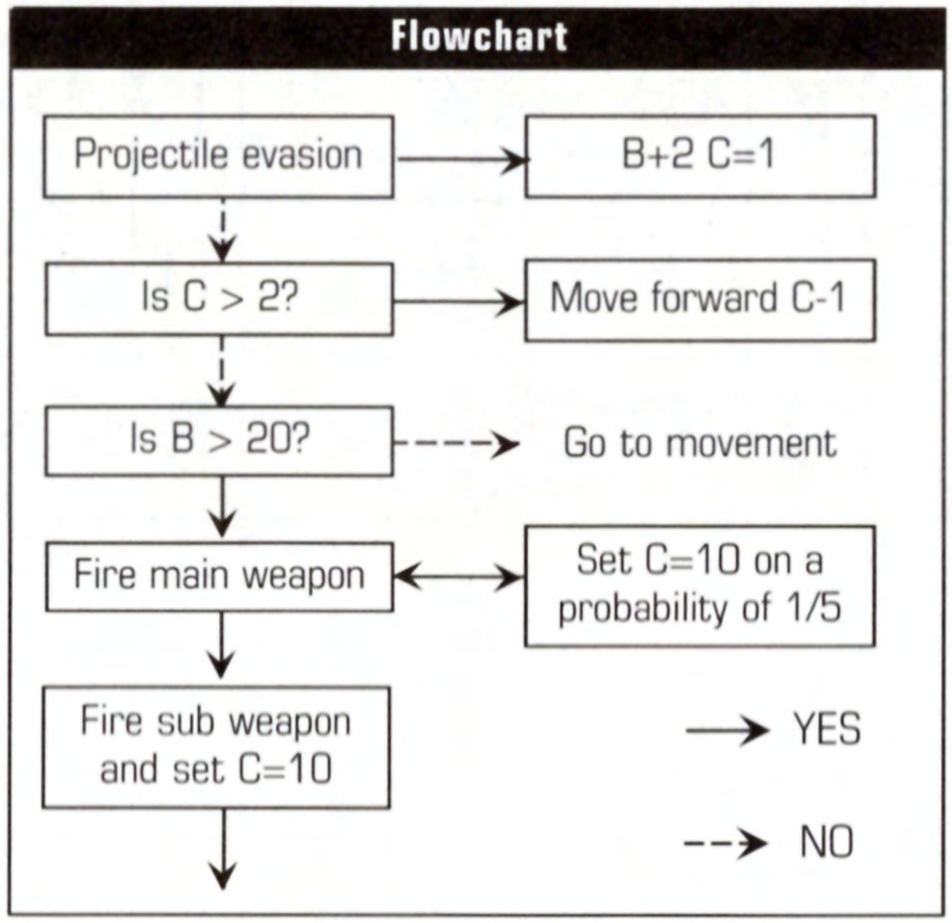
Program Developed by Artdink Technical Staff



A	30m	Axis 0° Scope 135°	B	40m	Axis 0° Scope 354°	C	20m	Axis 0° Scope 180°
D	20m	Axis 90° Scope 180°	E	45m	Axis 0° Scope 135°	F	45m	Axis -90° Scope 61°
G	45m	Axis 90° Scope 61°	H	20m	Axis -90° Scope 180°	I	45m	Axis -90° Scope 90°
J	20m	Axis 0° Scope 180°	K	45m	Axis 90° Scope 90°	L	200m	Axis 0° Scope 73°
M	200m	Axis 106° Scope 146°	N	200m	Axis-106° Scope 146°	O	20m	Axis -90° Scope 180°
P	300m	Axis 0° Scope 73°	Q	120m	Axis 0° Scope 73°	R	70m	Axis 0° Scope 90°
S	30m	Axis 0° Scope 73°	T	300m	Axis-106° Scope 146°	U	300m	Axis 106° Scope 146°
V	130m	Axis 0° Scope 73°	W	60m	Axis 0° Scope 45°			

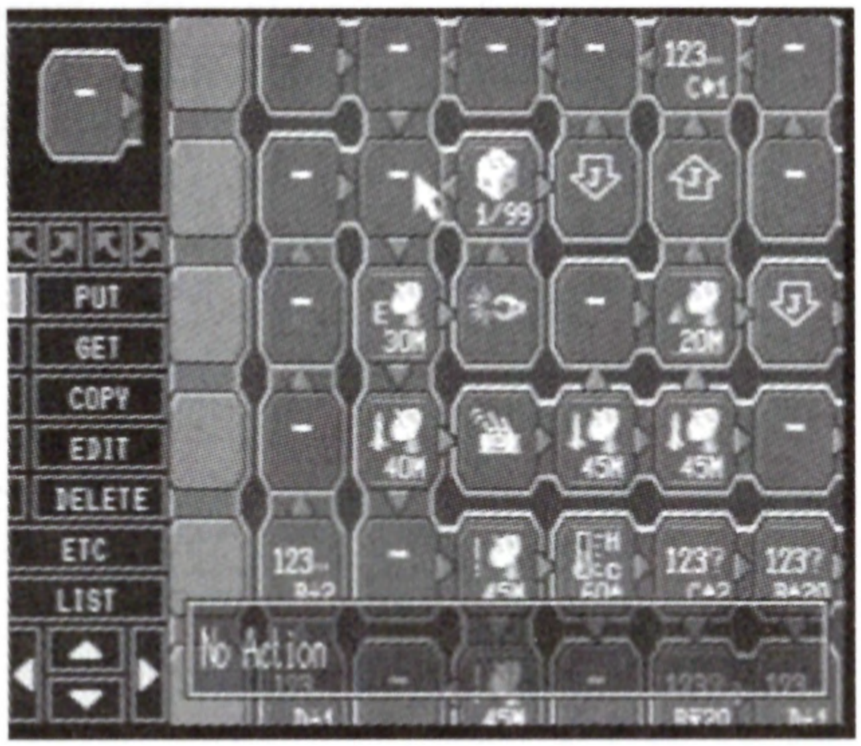
How are the Counters Used?

The details of the usage of the various counters are shown in the flowchart below, with the counters mainly used being B and C. Generally speaking, B counts the number of times enemy projectiles have approached and C counts the time elapsed since the OKE launched its own missile. Specifically, each time an enemy projectile is evaded B is incremented by 2, and when it exceeds 20, the OKE goes on the attack. In other words, by the time 10 projectiles have been evaded, the program decides that the enemy is near and starts to attack. Note that C is set to 10 when a missile is fired and thereafter reduced by -1 after each movement. When C drops below 2 again, missiles are fired (but exceptions are made). This prevents wasteful consecutive missile launches.



Check Out the Loops Used in Attack and Evasive Action!

Another interesting counter technique is the evasive action that uses a loop. The evasion techniques are not much different from those described previously. What is different is that the program does not end at the end of the evasive action, but rather, it loops around back to the start of the evasive action. This certainly permits the evasive action to be carried out faster, although not extremely so. In truth, there are also disadvantages to the loop technique, so we cannot recommend it unconditionally, but do try it out at least once. A loop is used in one other place. It is in the hand-to-hand combat, but the method is again the same as that described in the previous section. The difference is in that on a probability of 1 out of 99, it breaks out of the loop and jumps backward. Check out the effectiveness of this setting with your own eyes. It is a good example of a setting.





Now that we have explained programming at length, if you have read this far, you must be tired. In this last section, we will present some areas that we have not yet touched upon in a question and answer format.

Commands Just Don't Work Right!

You have the enemy in range and are getting ready to attack and... suddenly your OKE jumps to the right. What happened? Why would the OKE jump even though it was getting ready to attack? We call the chips that involve some sort of action "action chips." Each of these action chips has a certain priority. In this case, the jump has a priority higher than that of attack, so this may have caused this problem.

Low Priority

Ranked in priority from low to high, the various chips are ordered as follows: movement chips, attack chips, jump chips. Naturally, the highest priority is the stop chip. So if you put a jump after an attack chip, then...

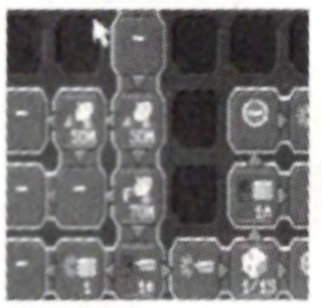




High Priority

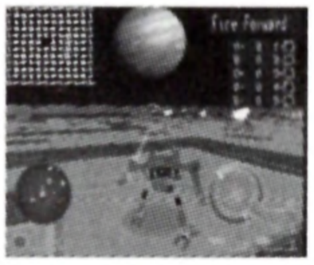
Your OKE gets ready to attack but suddenly stops and jumps. This may not necessarily be a disadvantage. There are ways to use this feature to your advantage.

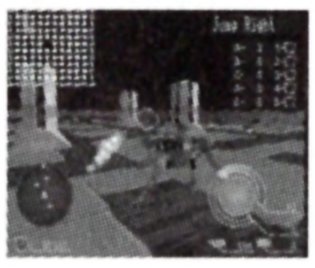




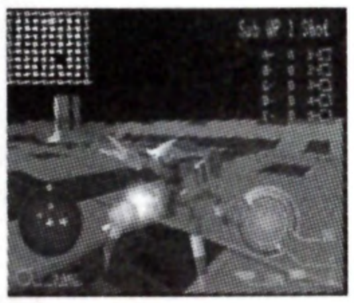
Programming Techniques that Take Advantage of the Priority of Commands

Let us explain this in more detail. Each chip has a short time lag between the time that it is processed and the time that it is executed. For example, in the case of an attack, the period when it is getting ready to attack is the time lag and the actual firing of the weapon is when the chip is said to be executed. The priority only takes effect during this time lag period. During execution, higher priority chips will be ignored if present. So how can we take advantage of this fact? For details, see the following, but this can be used for cover during an attack, the most vulnerable period among all actions.



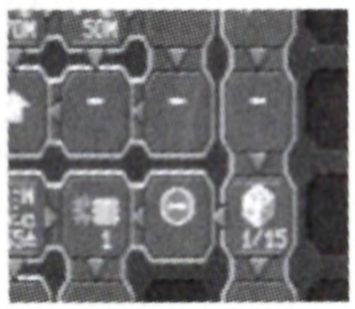


If you wish to avoid priority considerations, use the wait for action chip. Priority can be very useful depending on how you use it. Experiment with it. In one example of using priority, you could place evasive action at the start of the program and attack at the end. By doing this, even if the OKE is preparing to attack, it will still go into evasive action if a projectile approaches.



Commands Still Don't Work!

Even by no fault of priority, the chips you put together still don't work! This may certainly happen. As we touched upon previously, the action chips perform their action only after a time lag after they are processed. For example, the firing of a gun takes time. Naturally, processing will proceed during this time, and any action chips encountered during this period will be ignored as being unexecutable. Therefore, even if a fire sub weapon chip is placed after a fire main weapon chip, the sub weapon will never be fired.



While any actions other than movement are being executed, other chips encountered during that period will be ignored as being unexecutable. This is easily forgotten so be careful.

To prevent processing from proceeding during actions, two methods can be used. One is to use the wait for action chip to halt processing until action is possible. The other is to use the stop chip to stop all action.

The Wait for Action and Stop Chips



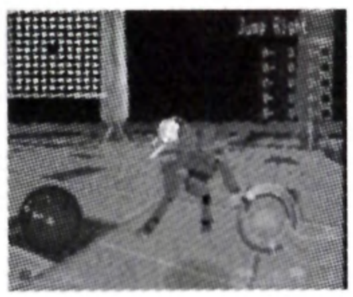
Even though this chip is called the wait for action chip, it should be more accurately called the permit action chip. This chip can be convenient.



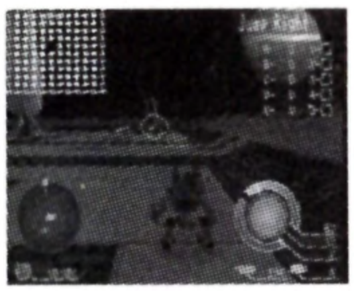
Stopping processing with the stop chip probably should not be done except in emergencies.

The OKEs on My Side All Move Together in Unison!

If the same program is used in three different OKEs, to a certain degree it cannot be helped that they will all move together. The main cause for OKEs moving together in unison is probably the evasive action used in response to a frontal attack. If this can be avoided, it will be possible to spread out your OKEs.



If a jump to the right is used to evade projectiles from the front, then ...



all of the OKEs will often be pushed to the right side of the field.

Giving your Program Randomness (Breadth)

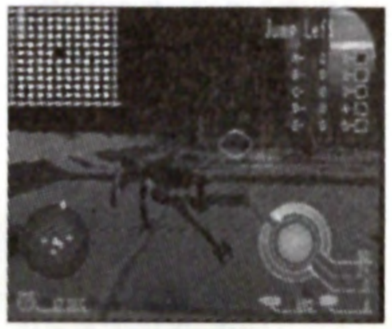




The simplest method of spreading out your OKEs in evasive action is to change the direction of the jump depending on the position of your other OKEs. However, this may not apply depending on the terrain. The safest method of avoiding this is to jump either to the left or to the right on a probability of 1 out of 2, as we described previously. Ideally, this should be followed by a search for allies or obstacles. This is probably too much to ask, though. Another method is to place a routine, like the one on the top right, at the start of processing of the program.

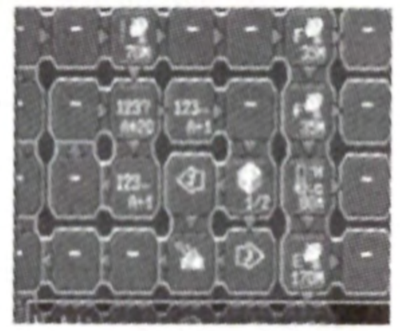
But you can't spend all your time avoiding your other OKEs; this is war after all. One way of avoiding this is a program which functions for only the first 10 seconds after the start of battle. If all goes well, your OKEs will fan out in formation.

Even Attacks by My Other OKEs Trigger Evasive Action



To be blunt, this cannot be avoided completely, but it is possible to deter this to a certain degree. For example, you can perform an obstacle search at 70m, increment a counter and then do a loop. Then go to evasive action only when the counter reaches a certain value. If there is a reaction many times in the loop, you can assume the projectile is coming at your OKE. If you evade only those attacks, you can ignore attacks by your other OKEs (or more precisely, attacks not aimed your OKE).

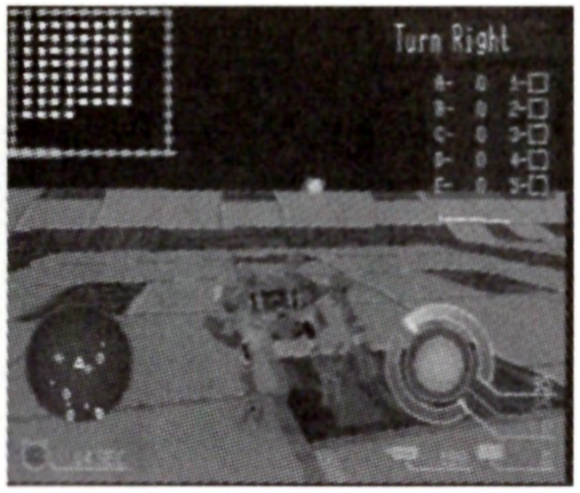
Program example. Move to evasive action if the counter is 20 or higher. However, this is not very meaningful for shotgun pellets or consecutive shots where many rounds are fired, but it is better than nothing.



One-on-One Battle Tactics

The most important aspect of the One-on-One Battle is that it is only a single battle. Thus, there is no need to conserve sub weapon ammunition. Chose rockets over missiles as your selection of sub weapon, and you will find the shotgun is very good as the main weapon. Mines can be very powerful if used well, but this is a challenge.



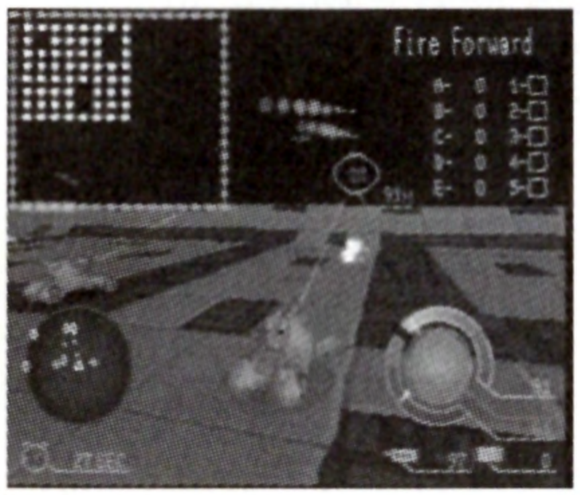


About Movement Chips

The turning and other movement chips are different from other chips in that they remain in effect until another action is executed. So by using movement together with a counter, it is possible to determine the angle of rotation. Those who are not able to avoid obstacles well should try this. In addition, the ascend and descend are special in that they can be combined with move forward or other movement chips. So you can move forward while gaining altitude, for example.

I Can't Win in the Single-Player Scenario No Matter What!

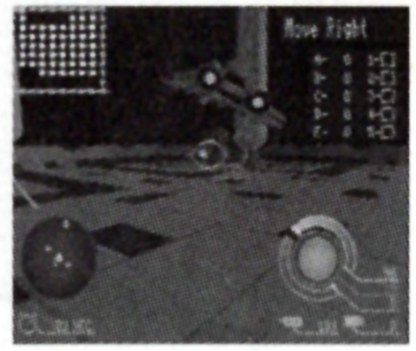
Maybe you are just not cut out for it... but that would be cruel of us so we won't say that. In fact, the enemies on Callisto are very strong and not easily defeated. If you are having a difficult time, try changing the camera mode and observe the enemy closely. You should be able to find a weakness. If you find a weakness it will not be so difficult to win. In desperation, you will find that large missiles are very effective. It can be that simple. Load up with as many large missiles as you can carry and launch them all at the start of battle. The terrain may affect the outcome and there are other aspects of luck, but maybe 70 to 80% of the time you will be greeted with only a burnt-out wasteland. But this can be rather boring, so we definitely do not recommend it.



Practice Programs

"Completely Master the Software"

Although the programming section ended on the previous page, we have included three bonus programs that have distinctive features. Try using them as practice opponents in the Test.

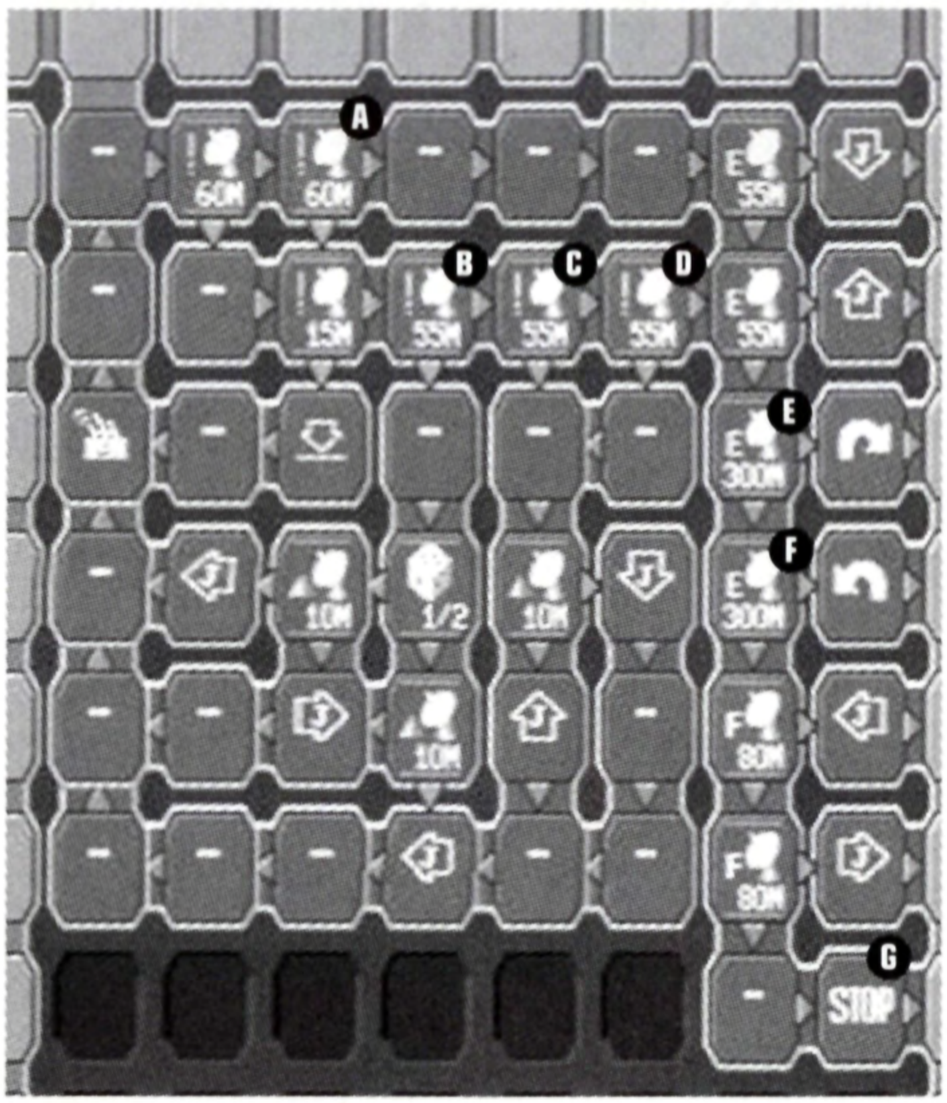


Build an Over Kill Engine That Excels at a Single Skill!

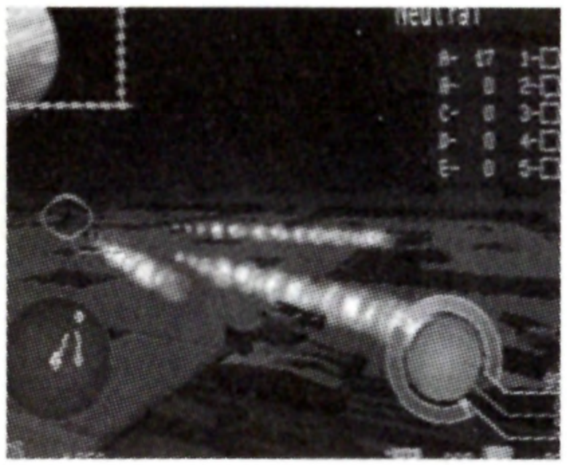
One thing that we would like to emphasize is that when putting together a program, you should give your program one clear, distinctive characteristic. If you put together several OKE like this, you will certainly be able to create a strong program. The three programs we present here were created for just that purpose.

Lesson 1 Program that Relentlessly Runs Around

As you can see, this program only runs around, trying to escape. So it is weak. The interesting aspect of it is the looping structure of the evasive action. Since this program has few other elements, there is little significance in the use of a loop, but it shows its real worth when combined with other elements. In addition, the tactic of using enemy searches to look in the direction of the enemy while running around to escape could actually be used in battle. Try pitting one of your favorite OKEs against three of these OKEs, and see how long it takes to take out all three.

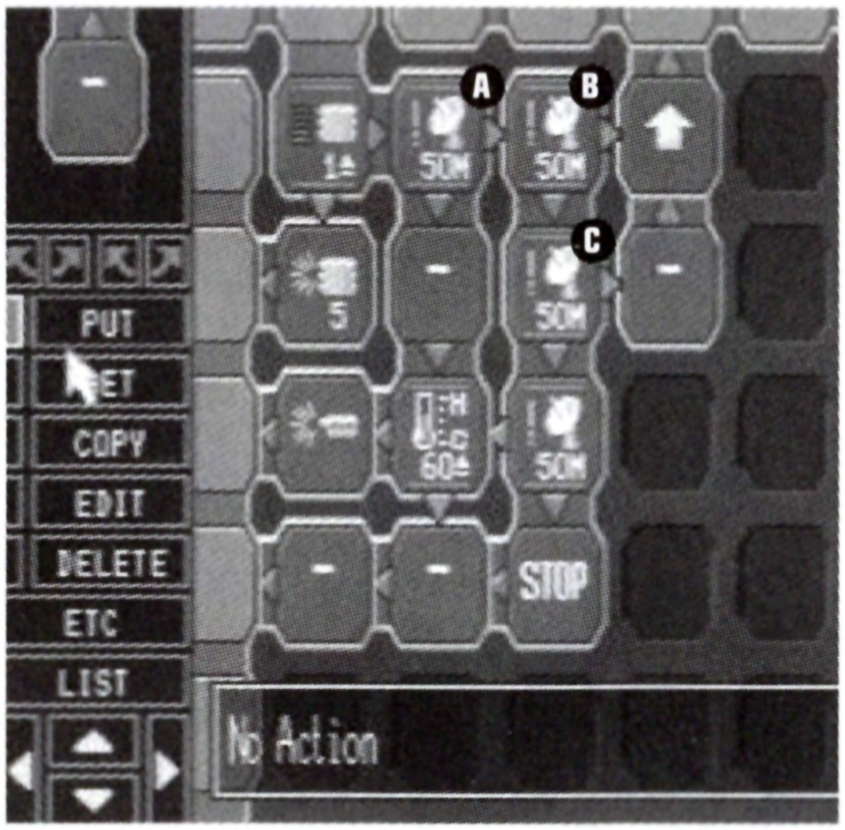


At A, two 60m searches are placed, but this is in order to eliminate the blind spot in back. B, C and D are searches to the front, right and left, respectively. E and F are searches to the right and left excluding the front that point in the respective directions. The placement of a stop chip at G is to halt the turning at E and F when the enemy is in front.



Lesson 2 Indiscriminate Large Missile Launch Program

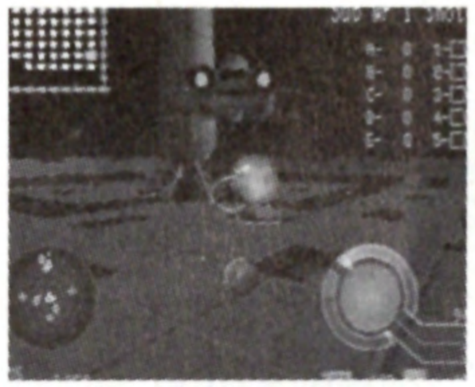
Rather than a programming technique, this program merely launches large missiles immediately at the start of battle. For the Over Kill Engine hardware, use the Jackhammer with the 8 Lg. Missile Pod. If you are fighting against this OKE, you will have no problem if your OKE is equipped with an Interference Device such as the Omni P1 ID so that it can jam missiles, but if not... Losing to this OKE would be pretty embarrassing. Try to build a program that can handle any type of enemy!



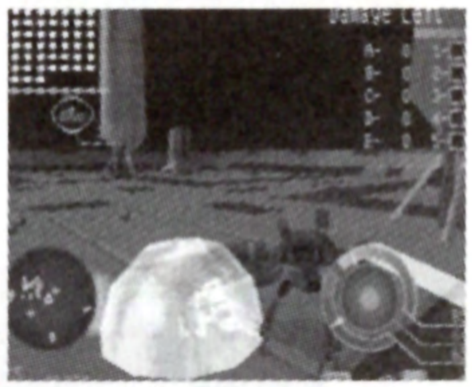
A is a search for obstacles in all directions, while B and C search to the left and right and evade by moving forward. So projectiles from the front and back are ignored. This only works for a tank OKE with thick armor. The effectiveness of heat radiation is poor during launches so a check on the OKE's temperature is performed to halt launches if it is too hot.

Lesson 3 Program that Drops a Rain of Landmines upon the Enemy

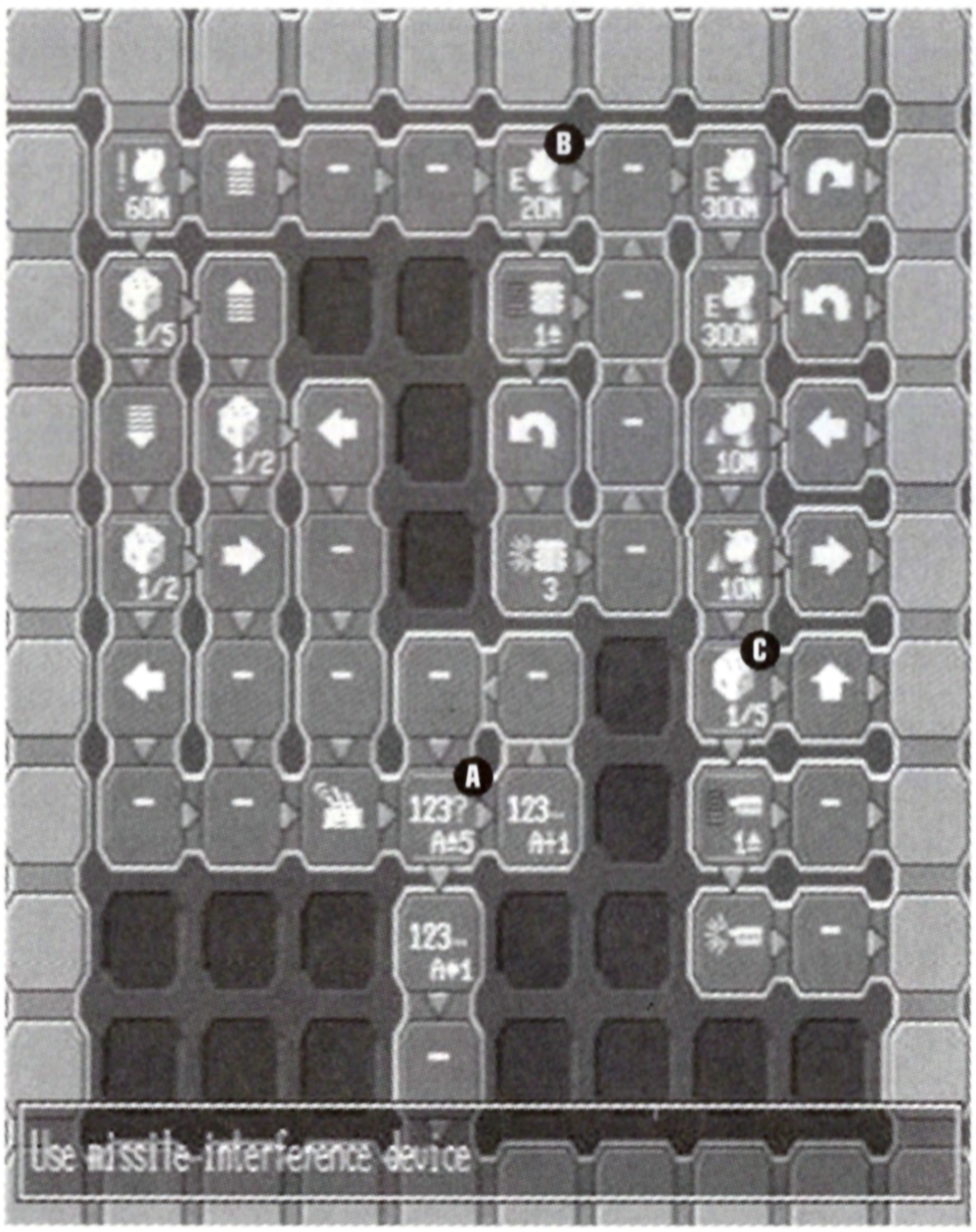
This is a special program that can only work for a flying OKE. This program relentlessly approaches the enemy while evading attacks and then when it has approached within 20m, drops landmines from above the enemy. This rain of landmines coming down all of a sudden is not easy to avoid. This is one terrifying bomber. Once you have obtained a IronDeath or other OKE able to carry landmines, definitely try to build it. This can be a terribly powerful attack if it goes well, and it also is a very entertaining battle to watch.



Landmines start dropping onto the enemy from above. The enemy below is blown away instantly.



Landmines inflict major damage in a direct hit. Even if they miss, they still function as landmines.



Thus, if you want to move a certain amount in evasive action, you can burn up time by using a loop and counter as in A. B searches for the enemy in all directions while dropping landmines. The random number at C is used to limit the number of times the main weapon is fired in order to approach the enemy.

Carnage Heart © 1997 Artdink.

Sony Computer Entertainment logo is a trademark of Sony Corporation. PlayStation and the PlayStation logos are trademarks of Sony Computer Entertainment Inc. The ratings icon is a trademark of the Interactive Digital Software Association. Manufactured and printed in the U.S.A. THIS SOFTWARE IS COMPATIBLE WITH PLAYSTATION GAME CONSOLES WITH THE NTSC U/C DESIGNATION. U.S. AND FOREIGN PATENTS PENDING.

PlayStation™

SONY COMPUTER ENTERTAINMENT