Tie Fighter, Part II

Wayne Sikes

After an initial, general look at LucasArts's Tie Fighter game engine, Wayne Sikes digs deeper. This month, he scrutinizes the game's data files, as well as the structure of the Defender of the Empire add-on mission set.

This month, we conclude our review of TIE Fighter by LucasArts Entertainment Company. We've already looked at the TIE game engine in general, and I wrapped up last month by summarizing some of the data I found in the pilot file. This month, we'll go through some of the data found in the TIE mission files. LucasArts released Defender of the Empire, TIE Fighter's add-on mission disk set, just as I began writing this review, so I'll also cover some of the mission data I found in these new missions.

The data files in the Defender of the Empire mission set were somewhat different from what I expected. The add-on missions for X-Wing, the first game in the Star Wars series, contained new game engines (the primary executable plus two overlay executables) in each mission set. TIE Fighter's mission set consists primarily of the new mission files plus some graphics and sound files for the new Missile Boat vehicle. The absence of a new game engine in the TIE add-on missions implies that the original game engine had the data for add-on missions built into it—the add-on mission sets will activate this data as needed. LucasArts's strategy of creating only one game engine is good in that it forced the company to plan out the entire TIE game series from the start. This makes for cleaner and more reliable executables—as compared to game engines that must be updated with each release of new missions. Also, by not distributing new game engines with each add-on mission set, LucasArts saves money on duplicating and distributing mission disks, because the game engine files are quite sizable.

Tie Fighter add-on missions don't modify the original game engine, they just activate data already incorporated in the engine.

Listing 1. General Flight Group Parameters

STRUCTURE OFFSET (DECIMAL)	DATA TYPE*	DESCRIPTION
0-11	byte	Flight Group Name. 12-byte null-terminated array.
24-35	byte	Cargo 1 text. 12-byte null-terminated array.
36-47	byte	Cargo 2 text. 12-byte null-terminated array.
48	byte	Special Craft Position.
50	byte	Vehicle. 1=X-Wing, 2=Y-Wing, 3=A-Wing, etc.
51	byte	Vehicles Per Wave.
52	byte	Starting Configuration. 0=Normal -> 8=Shields off
53	byte	Weapons. 0=None -> Magnetic Pulse
54	byte	Beam Weapons. 0=None -> 3=Decoy Beam
55	byte	Affiliation. 0=Rebel, 1=Imperial, 2=Neutral, etc.
56	byte	Artificial Intelligence. 0=Novice -> 5=Super Ace
58	byte	Talk Flag. 0=Talk off. 1=Talk on.
60	byte	Formation. 0=Vic -> 9=Vertical
64	byte	Number Of Waves.
66	byte	Player Position.
73	byte	Difficulty. 0=All Levels -> 5=<Hard Levels
96	byte	Arrival Mother Ship Flight Group.
97	byte	Arrival Method. 0=Hyperspace. 1=Mothership
98	byte	Departure Mother Ship Flight Group.
99	byte	Departure Method. 0=Hyperspace. 1=Mothership

* "byte" references an unsigned character.

TIE Mission File Overview

The TIE Fighter mission files have a .TIE suffix. Refer to "TIE Fighter, Part I" (Chopping Block, Feb./Mar. 1995) for information on how the battle and historical mission files are named as well as for listings of the mission files found in the original game distribution. The TIE mission files range in size from about 2,000 to 22,000 bytes. Each mission file contains essentially all the flight group data, briefing text, radio messages, mission objective summaries, mission accomplishments, enemy opposition summaries, and instruction or warning messages that appear during a mission. X-Wing, on the other hand, had separate mission and briefing data files, and its mission and briefing text was nowhere as detailed as TIE Fighter's.

The data for each flight group is contained in a 292-byte structure. All flight group structures are grouped together with the first structure beginning at offset `1CA` (`hex`) in the mission file. The flight group structures, and hence the flight groups, are ordered in a one-up manner with the first flight group structure in the file as "flight group 0".

The TIE mission data is one of the most complex mission structures I have seen. Due to this complexity, I will be able to discuss only a few of the mission details. Hopefully, the mission parameters I cover here will give you a broad idea of the types of data used by the TIE game engine for setting up missions.

General Mission Parameters

Listing 1 gives several parameters contained in the flight group structures. As you can see, each structure contains a large amount of flight group data, and we are just getting started! The flight group Name, Cargo 1, and Cargo 2 data are standard C `null`-terminated string arrays. The last character of each array must be `null (0)`, so each string can contain up to 11 characters.

The Vehicle byte specifies the vehicle used by the flight group. Each game vehicle is referenced by a specific value—a value of 1 specifies an X-Wing, 2 is a Y-Wing, 3 is an A-Wing, and so on. TIE Fighter has many more vehicles than X-Wing. The Vehicles Per Wave variable specifies the number of vehicles that will appear with each wave of craft, and the Number Of Waves byte specifies the total number of waves allowed to appear.

Several parameters specify the condition of the flight group vehicles when the group first appears in the game. The Starting Configuration specifies normal, extra weapon, damaged, and special (shields off, hyperdrive off) configurations. The Weapons variable gives the default weapon load and the Beam Weapons details any beam weapon loadout. The Beam Weapons data is especially fun to alter because in the original game only Darth Vader had a Decoy Beam weapon. Now you can give yourself one too!

The Artificial Intelligence byte tells how good the pilots in the flight group are. You can set this level from Novice (not very good) to Super Ace. (I usually avoid the Super Ace settings because I get destroyed very quickly.) The Affiliation variable sets the allegiance of the flight group (such as Rebel, Imperial, or Neutral), and the Talk Flag toggles your ability to talk to or command the flight groups. (I usually turn this flag on when programming enemy flight groups so I can tell them to "go home" when I'm being beaten badly.)

Every flight group can arrive and depart via hyperspace or a mothership. The Arrival and Departure Mother Ship Flight Group variables specify the flight group designated as the mother ships. The master controls for how vehicles arrive and depart are in the Arrival and Departure Method bytes. If you program a flight group to jump into hyperspace in the game, the vehicle you select for the flight group *must* have hyperdrive engine capability.

Flight Group Start Conditions

Listing 2 summarizes the flight group structure data that specifies when a group enters the game. I have given two start conditions the arbitrary label of Primary and Secondary Start Conditions. Both start conditions function in the same manner.

Several possible Primary and Secondary Start Conditions exist. These conditions include an "always start" condition (0), "the designated object must have arrived" (1), "the object must have been destroyed" (2), and so on. The start condition depends on actions happening to another object. This object may be another flight group, a certain type of vehicle (X-Wing, Y-Wing, and the like), or a vehicle that has a specified allegiance (Rebel, Imperial, and so on). The condition may also depend on a range of flight groups (flight groups 1 to 6, for example). The Primary and Secondary Dependency Type variables specify the type of object used for the start condition and the Primary and Secondary Start Data variables contain the data for the object. If the object is specified as a flight group, the data variables will contain the number of the flight group. If the object is a type of craft, the data will specify the craft in the dependency condition.

In addition to all the Primary and Secondary conditions, there can also be time delays that prevent a flight group from entering the game for a specified time interval. The Start Minute Delay counts in units of minutes, and the Start Second Delay counts in intervals of five seconds per data increment.

The Primary And Secondary Logic Switch is interesting in that it specifies whether the player or game must accomplish *both* the Primary and Secondary Start Conditions or just either *one* of them. When you set this switch to 0, a logical AND condition is specified and the game or player must meet both start conditions. A value of 1 specifies a logical OR condition and only one of the start conditions must be met for the flight group to enter the game.

Commanding The Flight Groups

Commanding the flight groups is one of the most interesting yet complex areas of the TIE game engine. A lot of forethought went into the design of the algorithms used for the orders that can be given to a flight group.

Each flight group can have primary, secondary, and tertiary orders. The flight group structure stores each order as an 18-byte structure with the three orders structures occurring sequentially in the flight group structure, beginning at offset 104 (decimal) and ending at offset 157 (decimal). Listing 3 summarizes some of the parameters found in each of the orders structures. Notice the detail given to each order.

The Order variable contains at least 35 commanded orders. A value of 0 commands the flight group to remain stationary, 1 instructs it to fly away, 2 commands it to fly a loop, and so on. Any details associated with the order, such as time constraints or the number of loops to fly, are found in the Indicator1, Indicator2, and Indicator3 bytes. The Commanded Speed byte details the velocity for the flight group while it is carrying out the order. The speed values increment in units of 10% of vehicle velocity.

Up to four targets can be associated with each order. Each of these targets is specified in the Target X Data and Target X Type variables (where X is targets 1 to 4). The Target X Type variable specifies the type of target and the Target X Data variable contains the data for the target. Among the available target types are flight groups, types of craft (X-Wing, A-Wing, and so on), or craft having a specified allegiance (Rebel, Imperial, and the like). The Global Player is a special target type. The Global Player type and its associated data are "modifiers" of other target objects (I'll discuss global data types in more detail later.). The data for each target (Target X Data)

LucasArts's single game engine—forced the company to plan out the whole game from the start, which makes for more reliable executables.

Listing 2. Start Condition Data

```
STRUCTURE          DATA      DESCRIPTION
OFFSET (DECIMAL)   TYPE*

74                 byte      Primary Start Condition. 0=Always. 1=Arrived, etc.
75                 byte      Primary Dependency Type. 0=no dependence, 1=flight
                             group dependence, 2=vehicle type dependence, etc.
76                 byte      Primary Start Data.
78                 byte      Secondary Start Condition. 0=Always. 1=Arrived, etc.
79                 byte      Secondary Dependency Type. 0=no dependence, 1=flight
                             group dependence, 2=vehicle type dependence, etc.
80                 byte      Secondary Start Data.
82                 byte      Primary And Secondary Logic Switch. 0=AND. 1=OR.
84                 byte      Start Minute Delay.
85                 byte      Start Second Delay.

*  "byte" references an unsigned character.
```

Listing 3. The Orders Structure

STRUCTURE OFFSET (DECIMAL)	DATA TYPE*	DESCRIPTION
0	byte	Order. 0=remain stationary, 1=fly home, 2=fly loop, 3=fly loop and evade, 4=rendezvous, etc.
1	byte	Commanded Speed. 0=stopped -> 10=100% velocity
2	byte	Indicator1. General timer, loop counter, etc.
3	byte	Indicator2. General timer, loop counter, etc.
4	byte	Indicator3. General timer, loop counter, etc.
6	byte	Target 1 Type. 1=flight group, 2=type of craft, 5=allegiance, 7=global, 8=flight group range
7	byte	Target 2 Type. 1=flight group, 2=type of craft, 5=allegiance, 7=global, 8=flight group range
8	byte	Target 1 Data. flight group, vehicle, etc.
9	byte	Target 2 Data. flight group, vehicle, etc.
10	byte	Targets 1 And 2 Logic Flag. 0=AND, 1=OR
12	byte	Target 3 Type. 1=flight group, 2=type of craft, 5=allegiance, 7=global, 8=flight group range
13	byte	Target 3 Data. flight group, vehicle, etc.
14	byte	Target 4 Type. 1=flight group, 2=type of craft, 5=allegiance, 7=global, 8=flight group range
15	byte	Target 4 Data. flight group, vehicle, etc.
16	byte	Targets 3 And 4 Logic Flag. 0=AND, 1=OR

* "byte" references an unsigned character.

can be a flight group number (assuming Target X Type specified flight groups), a vehicle type (assuming Target X Type called for a type of craft), an allegiance (assuming Target X Type called for allegiance types), and so on.

If this discussion of targets hasn't already been complex enough, let's add in the logic switches and global types. In Listing 3, the data for targets 1 and 2 is grouped together, as is the data for targets 3 and 4. These data were not placed in the orders structure randomly. Targets 1 and 2 can be grouped together into a single target specification, and the same goes for targets 3 and 4. The grouping is done by using two Boolean flags—the Targets 1 And 2 Logic Flag and the Targets 3 And 4 Logic Flag variables. A value of 0 commands that the two targets function in an AND manner (both groups work together), and a value of 1 signals an OR condition (the two target groups work independently). As I previously mentioned, the Global Player target type is a modifier of other targets. It is used in conjunction with these logic flags to modify the actions a flight group takes against a target.

Let's look at an example to help simplify this targeting information. Let's assume you are flying a TIE Fighter and you want to program a flight group to attack all TIE Fighter vehicles—that is, all TIE Fighter craft except yours. How would you do this? First of all, program the flight group with the Attack Flight Group orders (give the Order byte a value of 7). Let's arbitarily use targets 1 and 2 for this order. We would set Target 1 Type to specify a type of craft (a value of 2 commands for a type of craft), and next we would set Target 1 Data to a value of 4, which would specify TIE Fighters.

If we stopped right here, then all TIE Fighter craft, including yours, would be attacked. Set Target 2 Type to 7, which invokes the Global Player type, and set Target 2 Data to 10 (decimal) which is the Global Not Player option. Finally, set the Targets 1 And 2 Logic Flag to 0 to force the AND combination of targets 1 and 2. Now all TIE Fighter craft except yours will be attacked. I realize that I have left out lots of detail here, but I hope you understand some of the logic used in commanding flight groups. (I said previously that the orders system in TIE Fighter is complex. Was I right?)

Winning the Game

There are Primary (Win 1) and Secondary (Win 2) win conditions that players must meet to win a mission. Refer to Listing 4 for a summary of the win and bonus conditions specified in the flight group structure. You'll see that I've included a possible Win 3 condition, but I do not have much evidence that indicates this condition exists.

The Win 1 Condition and Win 2 Condition variables contain the conditions that must be met. The values in these conditions are the same as those found in the start condition variables. A value of 1 specifies that the group must have arrived, 2 means the group must be destroyed, and so on. The Win 1 Detail

Listing 4. Win and Bonus Variables

STRUCTURE OFFSET (DECIMAL)	DATA TYPE*	DESCRIPTION
158	byte	Win 1 Condition. 1=created, 2=destroyed, etc.
159	byte	Win 1 Detail. 1=50%, 4=special vehicle, etc.
160	byte	Win 2 Condition. 1=created, 2=destroyed, etc.
161	byte	Win 2 Detail. 1=50%, 4=special vehicle, etc.
162	byte	Possible Win 3 Condition.
163	byte	Possible Win 3 Detail.
164	byte	Bonus Condition. 1=created, 2=destroyed, etc.
165	byte	Bonus Detail. 1=50%, 4=special vehicle, etc.
166	signed char	Bonus Points. 1 increment = 50 points.

* "byte" references an unsigned character.

and Win 2 Detail bytes function as modifiers of the conditions. For example, if the condition specifies that a flight group be destroyed, the detail might modify this situation to mean that only the special craft in the group be destroyed to win.

The Bonus Condition byte functions the same as the win condition variables and the Bonus Detail functions the same as the win detail parameters. If the bonus condition and detail are met, bonus points can be awarded. The Bonus Points variable is a `signed char` value. Each increment in the value represents 50 points. Since the bonus variable is a signed value, note that negative or penalty bonus points can occur. For example, if the bonus condition on your mother ship is that it be destroyed, a penalty of -10,000 points may be "awarded" if the enemy gets past you and takes out your mother ship.

Want to Edit Some Missions?

When analyzing the more detailed aspects of a game, I frequently write utilities that help with my analysis. The end result of my analysis of the TIE mission structures is a routine called TIEDIT. If you've read this far, then you know how complex the TIE mission data is. For this reason, I opted to write TIEDIT in MS Windows because it is relative easy to implement list boxes, edit boxes, and the like in Windows. The sheer number of mission items I could edit was enough to make me abandon any thoughts of a DOS-based TIE mission editor. Completion of TIEDIT took longer than I anticipated because every time I tested it I would get "hooked" by the new missions I was creating, and then I'd spend too much time playing the game! TIEDIT.ZIP is on CompuServe in the Flight Simulation Forum (GO FSFORUM), Space Combat Library.

We Have A Winner Here!

As I've said before, LucasArts gaming products are generally well written and executed, and TIE Fighter is no exception. The level of detail LucasArts developers have given to TIE Fighter's mission structures is fantastic, to say the least. This dedication to a product will considerably prolong TIE Fighter's life on the shelves via more add-on mission disks. I'd really like to see LucasArts release some form of the mission editor used in creating the retail missions. Obviously, the company might want to wait until it's done creating more add-on missions, but a retail LucasArts TIE mission editor could possibly extend the life of the game by years. ■

Wayne Sikes has been a computer hardware and software engineer for the last 10 years. He has an extensive backgrond in C, C++, and assembly language programming. He also has several years experience as a computer systems intelligence analyst, where he specialized in deciphering and disassembling computer code on classified government projects. He has written numerous computer gaming help utitlities. You can reach him via e-mail at 70733.1562@compuserve.com or through Game Developer.