# A STRATEGY PLANNING PROGRAM FOR THE MIT SOLAR POWERED RACING VEHICLE

by

Bruce Eric Larson

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
at the Massachusetts Institute of Technology
May 1989



Signature redacted

Author
Department of Electrical Engineering and Computer Science
May 22, 1989

Signature redacted

Certified by
Gill Pratt
Thesis Supervisor

Signature redacted

Accepted by
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

# A STRATEGY PLANNING PROGRAM FOR THE MIT SOLAR POWERED RACING VEHICLE

by

Bruce Eric Larson

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1989

in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering


## Abstract

The Tour de Sol '89 Strategic Planner is a computer program written for the MIT solar car project. The program computes an optimal racing strategy to be used for competition in the week-long Tour de Sol solar-powered vehicle race. The Strategic Planner must consider factors such as energy contained in the car's battery, energy expected to be received from forecasted weather patterns, and energy expenditure varying with different speeds and slopes. By weighing these factors, the program informs the user at what speeds the car should travel for every point of the race, so that the car maintains a constant fuel efficiency and finishes with a nearly drained battery. In this manner, the car will finish the race in the least possible amount of time.


Thesis Supervisor: Gill Pratt
Title: Thesis Supervisor

Table of Contents

## List of Figures

# Chapter 1

## Introduction

The Tour de Sol '89 Strategic Planner is a computer program written in Lightspeed Pascal for the Apple Macintosh. The program's function is to plan an optimal racing strategy for the M.I.T. solar car at the six-day Tour de Sol race in Switzerland. The Tour de Sol is a landspeed competition between solar-powered vehicles, whose purpose is to demostrate the speed, efficiency, and endurance of solar-powered transit.

The M.I.T. entry in the Tour de Sol is a small, lightweight craft which carries one person only. Its motor is electric and is powered by the car's battery. The battery is recharged over the course of each day by an array of photovoltaic cells, which convert the sunlight that hits them into electric potential. The amount of power required to propel the car varies with its speed and the slope of the road. The wheels have regenerative breaks, so it is possible to actually recharge the battery on downhill slopes.

Because the Strategic Planner makes extensive use of the Macintosh user interface, a general familiarity with the Macintosh environment on the part of the reader is assumed.

# Chapter 2

# Design

## 2.1 Rules of the Tour de Sol

The Tour de Sol consists of a series of races which span six days. At the start of the first day, all vehicles have fully charged batteries. Each day of the Tour de Sol is divided into two parts: first, a road race, and second, optional laps around a track.

The race-part typically occurs on a stretch of public road connecting two cities whose distance is roughly one hundred kilometers. A different road is used for the race course each day. The winner of the Tour de Sol is determined by whichever car has the least sum of race times, so it is desirable to minimize the race time for each day. However, there are several limitations. The car's average speed for any one race can not be less than 30 kph, or else the car is disqualified. If the average speed for a race is above 45 kph, then the car is only given credit for having acheived a 45 kph average, and its recorded race time is adjusted accordingly. Furthermore, the car must obey all traffic regulations, including stop-lights and speed limit regulations.

The optional laps occur on a circular track several kilometers in circumference, and only occur after the day's race has been completed. They may occur immediately when the car finishes the race, or they may occur later in the day. On some days there may be no optional laps allowed. In any event, it is up to the driver to decide how many laps to complete, although there is a time limit after which no more laps may be performed. The benefit of performing laps is that for each lap completed, a certain amount of time is deducted from the car's final score. The disadvantage of performing laps is that they cause the battery to lose charge, which may impede the car's ability to travel at maximum speed in subsequent races. Each lap must be performed at an average speed of at least 30 kph, or else no credit is given for that lap.

## 2.2 Optimization

The problem to be solved is the following: Determine the speed at which the car must travel at every point along the race course, so that at no point is the battery emptied, and so that the final race time is minimized.

The complexity of the problem is greatly reduced if some simplifying assumptions are made. It is assumed that energy losses due to acceleration and deceleration are negligible, as are gains and losses due to wind patterns. It is also assumed that the total energy received from the

sun over the course of the day is distributed sinusoidally between 6:00 AM and 6:00 PM. Finally, it is assumed that each day may be calculated in isolation, independent of the conditions of other days of the race. This isolation is achieved by having the user specify the desired end-of-day battery charge for each day. The algorithm's inputs are thus reduced to the following: the storage capacity of the battery, the starting and ending charges for each day, the power consumed by the car as a function of velocity and slope, the terrain of the race course and its speed limit regulations, the energy received from the sun each day, and the rules of the race.

The optimization problem is further simplified if we temporarily assume that there are no laps performed on any day, and that exhaustion of the battery will not occur. The new question of interest is: At what speed should the car travel at every point along the race course, so that for a given energy depletion, the average velocity is maximized.

The solution to this question relies on this principle: race-time is minimized when energy-efficiency is maximized. If energy lost from ascending hills were a constant, independent of the velocity at which the hills were taken, then the optimal strategy would simply be to maintain a constant speed at all times, choosing the speed which will realize the desired energy depletion.

Unfortunately, the power consumption of the car is modeled by a fourth-order polynomial of velocity and slope. (See Appendix B.) Thus, for example, it is more efficient to ascend a steep slope at 40 kph and descend at 50 kph than it is to ascend and descend at 45 kph.

Fortunately, one simplifying fact is known. Because the power consumption function is nonlinear with velocity, it is always more efficient to take two stretches of road with the same slope at the same velocity, than it is to take one at a slower velocity and the other at a faster velocity (assuming a constant average speed is maintained). One may choose a velocity at which to travel for, say, a horizontal section of road, and know that it will always be maximally efficient, for the average speed achieved, to travel at that velocity on ALL horizontal stretches. Now the question is: given this horizontal velocity, what velocities should be chosen for other slopes so that a constant efficiency is maintained everywhere? In order to answer this question, one must be able to assign an efficiency rating to every pair of velocity and slope. This rating will be referred to as the efficiency criterion, whose derivation is given in Appendix A. The efficiency criterion is derived to have the property that, for any particular average velocity, the maximally efficient strategy will be the one where all stretches of the race are performed at the same efficiency rating. The rating is defined as follows:

$$\text{Efficiency}(V, \theta) = \text{Power}(V, \theta) - V * \frac{d}{dV}(\text{Power}(V, \theta))$$

where V is the car's velocity
theta is the slope of the road
Power(V, theta) is the power consumption function

Under this definition, efficiency decreases as the efficiency rating increases.

The strategy-planning algorithm calculates the strategy for each day's race in the following way: The algorithm initially chooses a horizontal velocity of 45 kph and calculates the corresponding efficiency. Velocities for all non-horizontal slopes are calculated to have the same efficiency as the horizontal velocity. If, at the end of the race, the average speed is found to be faster or slower than 45 kph, then the horizontal velocity is adjusted proportionally and the race is recalculated. When the entire race has been computed in this manner, the the final charge on the battery will either be higher, lower, or roughly equal to the user-specified end-of-day charge.

If the charges are approximately equal (within two percent of each other), then the calculation is finished.

If the final charge is too high, then it may be lowered by the expenditure of more energy where it is of the most benefit. In this case, the issue of performing

laps comes into play. If laps are not permitted that day, then nothing can be done to improve the race time. (Recall that no credit is given for finishing the race at speeds faster than 45 kph.) If, on the other hand, laps are permitted, then the program adds laps at the slowest permissable speed (30 kph), until either the final charge is brought down to the user-specified value, or until the time allocated for laps runs out. If the latter happens, the program erases the laps and re-performs them at a higher speed. This process continues until the final battery charge matches the user-specified value.

If, however, the laps are being performed at 45 kph, the battery is still overcharged at the end of the day, and laps are allowed immediately after the race, then it becomes most advantageous to raise both the race speed-average and the lap speed-average together. Even though no credit is given for the increase in the race speed-average, the increased speed provides more time to perform more laps. The two averages are increased until the actual final charge matches the user-specified final charge, or until the ceiling of 100 kph is reached, at which point the algorithm leaves the strategy unchanged.

The last case to consider is where the final battery charge is too low. In this case, the race speed-average is reduced below the optimal 45 kph, and the race is recalculated with no laps performed. This process

continues until the final charge is adequately high, or until the average speed falls to 30 kph (below which the car is disqualified). If the latter occurs, then the strategy is left as it is, and a warning is sent to the user. The same procedure is followed if the battery is drained at some point during the race. The speed-average is lowered until the drain is avoided, or until the speed-average falls below 30 kph.

This algorithm is invoked every time the strategy must be re-calculated. The following chapter discusses the environment in which the algorithm operates.

# Chapter 3

# User Interface

The Strategic Planner operates in a menu-based environment. When the program is run, it presents the user with a blank screen and four menu headings: FILE, STRATEGY, PARAMETERS, and SCREEN. The FILE menu allows the user to open and save files which contain the inputs to the strategy-calculating algorithm. The STRATEGY menu contains the functions that calculate and display the strategy. The PARAMETERS menu enables the user to edit the inputs to the strategic algorithm. The SCREEN menu allows the user to select the screen to be displayed. (Currently there is only one screen, the Strategy screen. This menu exists for future modifications which may require multiple data screens.)

## 3.1 Opening a file

When the program is first run, the only enabled menu selections are Open and Quit, both of which are in the FILE menu. The user selects Open and is presented with a dialog box requesting the name of the file to be opened. The user may either select a previously edited file or else choose the default file, dafault.dat. The file,

which contains the inputs to the strategic algorithm, is loaded into memory, and the strategy is automatically calculated. This calculation may last up to several minutes. Once the strategy has been computed, the program enables all menu selections and displays four windows. (See Figure 3-1.) Three of these windows -- Altitude, Speed, and Charge -- are graphics windows which plot, respectively, the road elevation, the recommended speed, and the estimated battery charge as a function of distance travelled. A fourth window -- the Pilot window -- displays this and other strategy-relevant information numerically.

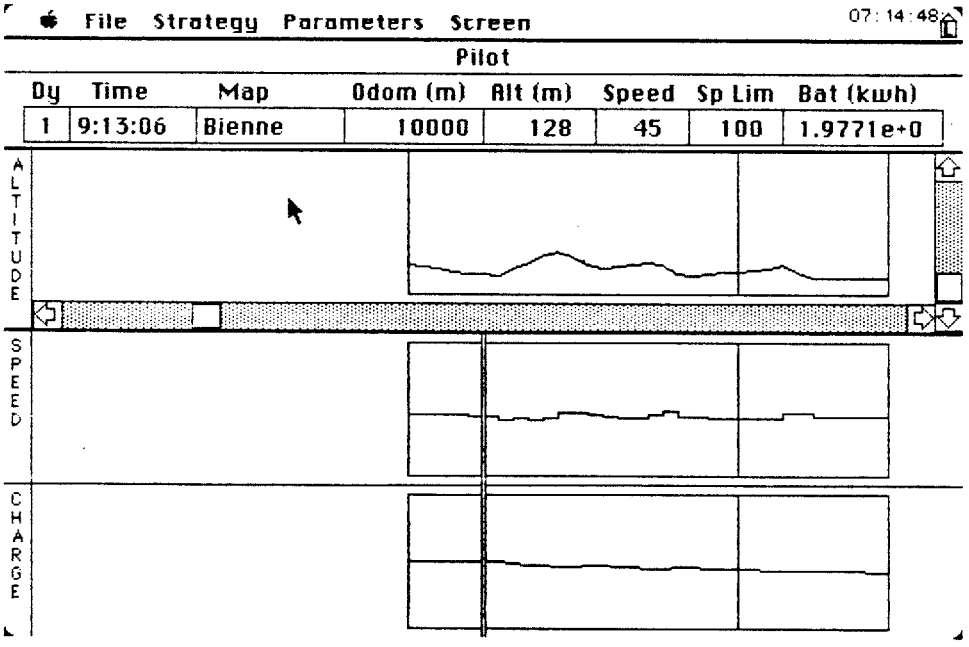


Figure 3-1: The Strategy Screen

To examine the strategy in detail, the user may slide

the Altitude graph both horizontally and vertically, using the scrollbars on the bottom and right-hand sides of the Altitude window. When the graph is moved horizontally, the Speed and Charge graphs are adjusted automatically, so that all three graphs stay in alignment. A vertical hash-line running down the center of the Speed and Charge graphs indicates the position whose information is displayed in the Pilot window. This information consists of the day and time of the indicated position, the name of the map which contains the position, the distance from the start of the map, the altitude, the recommended speed, the speed limit, and the estimated charge on the battery. When the graphs are moved horizontally, this information is updated to be consistent with the newly hashed position.

For a general overview of the strategy, the user may choose Select Results from the STRATEGY menu. Doing so will display the Results window, which indicates, for each day of the Tour de Sol, the speed-average of the race and laps, the efficiency rating of the race and laps, and the number of laps to be performed. (See Figure 3-2.) This window disappears when the OK button is clicked. The user may also increase (or decrease) the horizontal scale of the Altitude, Speed, and Charge graphs by choosing Expand X (or Contract X) from the STRATEGY menu. The Altitude graph may also be scaled vertically, with the Expand Y and Contract Y options in the STRATEGY menu.

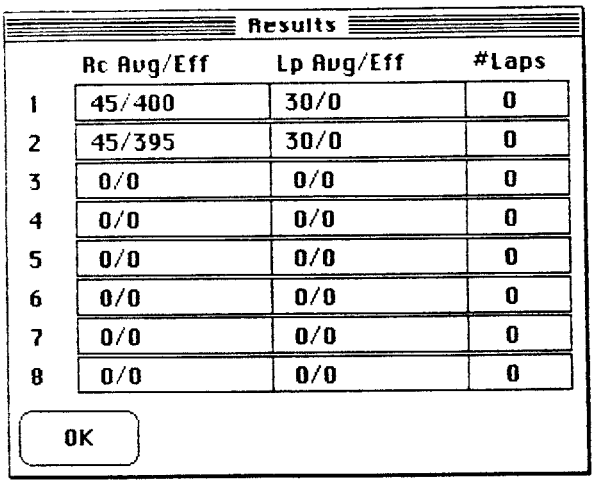


Figure 3-2: The Results Window

## 3.2 Editing the parameters

Now the user may wish to alter the inputs to the strategy-calculating algorithm. All the editing tools are located in the PARAMETERS menu. The Limits option opens a dialog box showing the limitations on speed and battery charge. (See Figure 3-3.) To edit a value, the user clicks on the appropriate box. A cursor appears in the box, and the user types in the new value. When all the desired changes have been made, the user clicks the OK button and the window is closed. If instead the user decides the alterations are inappropriate, then clicking the Cancel button will close the window and retain the old values.

The Calibration option opens a dialog box showing the details of the calibration point. (See Figure 3-4.) This

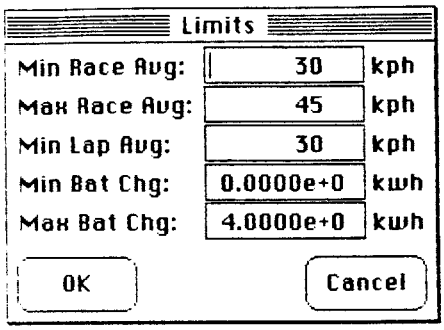


Figure 3-3: The Limits Window

point is the starting location from which the strategic algorithm calculates the strategy. The user selects the day and position at which to begin the calculation, and indicates the time of day and the battery charge at that time. All times must be in military (24 hour) notation and written in the form of hour:minute or hour:minute:second. Before the race begins, the calibration should be set to the start of the race. During the race, the calibration should be set to a point recently passed, so that the calculated strategy will be as accurate as possible.

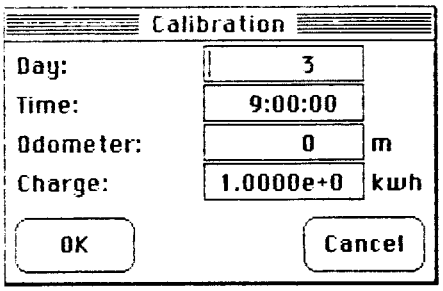


Figure 3-4: The Calibration Window

The Watt Function option opens a window containing

the coefficients to the power consumption function. (See Figure 3-5 and Appendix B.) The power function determines how much power the car consumes as a function of velocity and slope of the road.

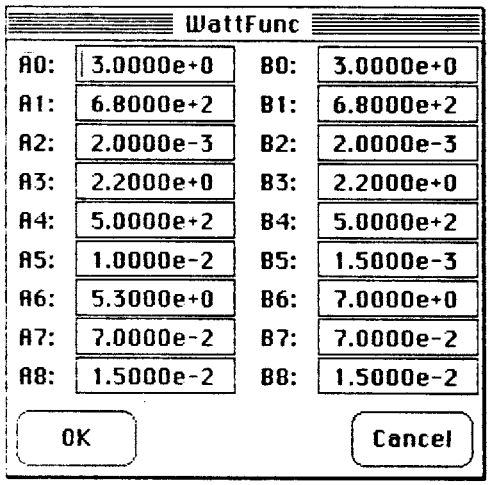


Figure 3-5: The Watt Function Window

The Maps option is used for editing the map collection. Each map corresponds to one section of the Tour de Sol, either one race or one lap. The divisions between these maps are indicated on the Altitude window by single vertical lines. A map does not denote the compass directions of the road, but rather gives a vertical profile of the road in a series of altitude measurements. Selecting the Maps option opens a window with sixteen buttons, each containing the name of a map (Figure 3-6). By clicking one of these buttons, the user opens the Map Editor window for the corresponding map. The Map Editor allows the user to alter the map's name and the distance

between its altitude measurements. Clicking the Edit Profile button opens the Profile window. Profile is a text-editing window which enumerates the altitude measurements in the following form: One measurement appears on each line, with no blank lines in between them. The measurements are in units of meters above sea level and must be integers. If the speed limit changes at a particular measurement, then the measurement is followed immediately by a semicolon, a space, and the new speed limit. A speed limit must be given after the first altitude measurement. The end of the map is indicated by the word "end" followed by a carraige return. To stop editing the profile, the user clicks the go-away box in the upper left-hand corner of the window.

The Itinerary option opens a window containing the Tour de Sol itinerary. (See Figure 3-7.) For each day, the following is indicated: the starting time of the race, the time interval when laps are permitted (optional), the amount of energy expected from the sun, the names of the race map and lap maps (optional), and the time deduction given for each lap completed (optional). The lap-related entries are left blank if no laps are allowed on that day. To indicate that a lap starts immediately after the race, the user sets the Lap Begins field equal to the Race Begins field. The end of the Tour de Sol is indicated by a the first row with the Day field left blank.

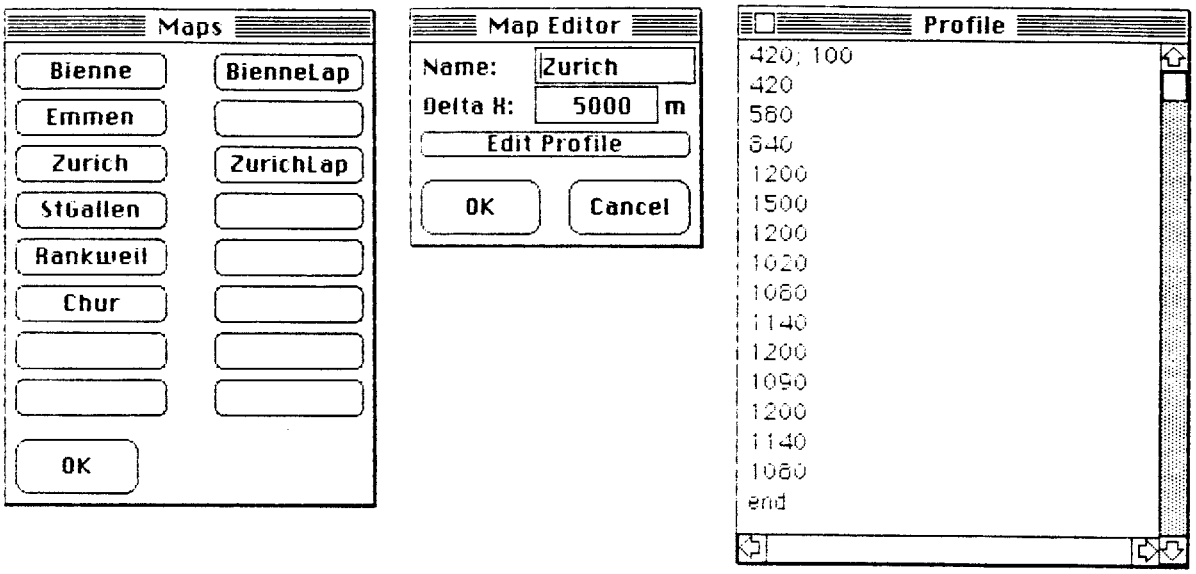


Figure 3-6: The Map Editing Windows

Itinerary

| Dy | Rce Bgn | Lap Bgn | Lap End | Kwh Rcv | Race Map | Lap Map | S/Lap |
|---|---|---|---|---|---|---|---|
| 1 | 9:00 | 12:00 | 14:00 | 4.000e+0 | Bienne | BienneLap | 90 |
| 2 | 9:00 | | | 4.000e+0 | Emmen | | |
| 3 | 9:00 | 9:00 | 12:00 | 1.000e+0 | Zurich | ZurichLap | 90 |
| 4 | 9:00 | | | 3.600e+0 | StGallen | | |
| 5 | 9:00 | | | 2.800e+0 | Rankweil | | |
| 6 | 9:00 | | | 2.600e+0 | Chur | | |
| | | | | | | | |
| | | | | | | | |

OK Cancel

Figure 3-7: The Itinerary Window

The End Charges option displays a window containing

the desired end-of-day battery charge for each day. (See Figure 3-8.) These values should be chosen to generate a strategy where the efficiency of the car is roughly the same for all days. The general trend should be to start with a full battery on the first day and to finish with a nearly drained battery on the last day.

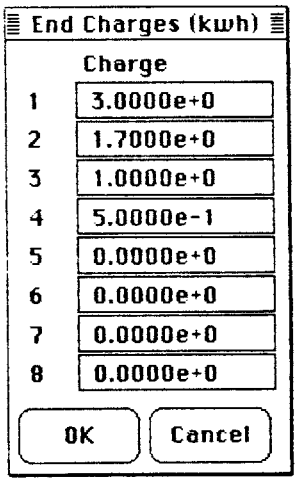


Figure 3-8: The End Charges Window

Once all the desired changes have been made, the user selects Recalculate from the STRATEGY menu. This option calculates the entire strategy from scratch, and reprints the Altitude, Speed, Charge, and Pilot windows to display the new strategy.

To save on disk the changes made to the parameters, the user selects either Save or Save As from the FILE menu. Save will write the new values to disk under the most recently specified file name. Save As opens a dialog box asking for the name of the file to be saved. To exit

the program, the user selects Quit from the FILE menu. An alert box appears, asking if the user wishes to save the data before quitting. The user may choose either Quit, Save and Quit, or Cancel.

### 3.3 Error messages

During the course of editing and calculating, the program may encounter errors. In the event of an error, one of the following messages will be signalled to the user:

Non-numeric data entered in a numeric field.

Non-integral data entered in an integer field.

Value out of range (signalled for integers greater than 1e+9).

Invalid time format (times must be entered in the form of hour:minute or hour:minute:second).

Undeclared map name (signalled if the Itinerary makes reference to a map not contained in the map collection).

Name or value exceeds 13 characters.

No initial speed limit specified (each map's profile must indicate the speed limit after the first altitude measurement).

Altitude or speed value out of range (signalled for altitudes or speeds greater that 32,767).

Invalid odometer setting in calibration (signalled if odometer setting is negative or is greater than the length of the race).

Calculated average falls below minimum speed (signalled if a race speed-average falls below 30 kph, disqualifying the car from the Tour de Sol).

When an error occurs, the procedure being executed is halted, and an alert box containing the appropriate message is presented to the user. The user must click on the box in order to proceed, and may then act to correct the problem.

# Chapter 4

## Results

The Strategic Planner program was run with a variety of test cases. In this chapter are shown six examples of the algorithm's behaviour under varying conditions.

The first example demonstrates how the strategy was calculated for a day when there are no laps allowed, and when a large amount of battery charge was allocated for that day. For this case, the initial battery charge was 2 kwh and the final battery charge was 0.1 kwh, allowing the battery to be almost drained at the end of the day. In theory, the program should have planned a strategy where the average speed was 45 kph and a constant efficiency was maintained. Because sufficient charge was allocated, the battery should not have been drained at the end of the race.

The strategy graphs produced from this test case are shown in Figure 4-1. As expected, the charge declined over the course of the race but did not drop to zero. The variations in the recommended speed logically corresponded to the contours of the road; the speeds were slower for uphill slopes and faster for downhill slopes, achieving constant efficiency. The program indicated that the

average velocity was, in fact, 45 kph, and that the efficiency rating was 434.

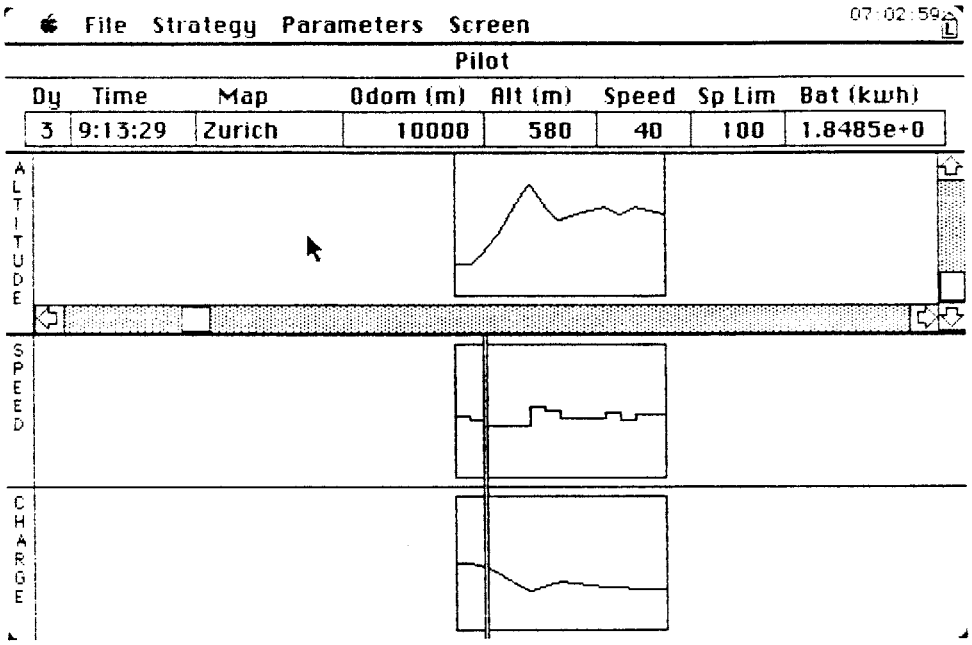


Figure 4-1: Results of Test Case #1

The second test case demonstrated how the algorithm behaved when there was insufficient charge allocated to maintain a 45 kph average throughout the race. To achieve this situation, the initial battery charge was dropped to 0.8 kwh. The program should have computed the strategy so that the average speed would be less than 45 kph, and so that the battery would be almost drained at the end of the race.

The output graphs for this case are shown in Figure 4-2. Again, the charge diminished over the course of the race, and this time the battery was left almost empty. The program indicated that the average speed was 36 kph,

and that the efficiency was 227. This efficiency rating was lower than the previous efficiency rating, indicating that the new strategy was more efficient.

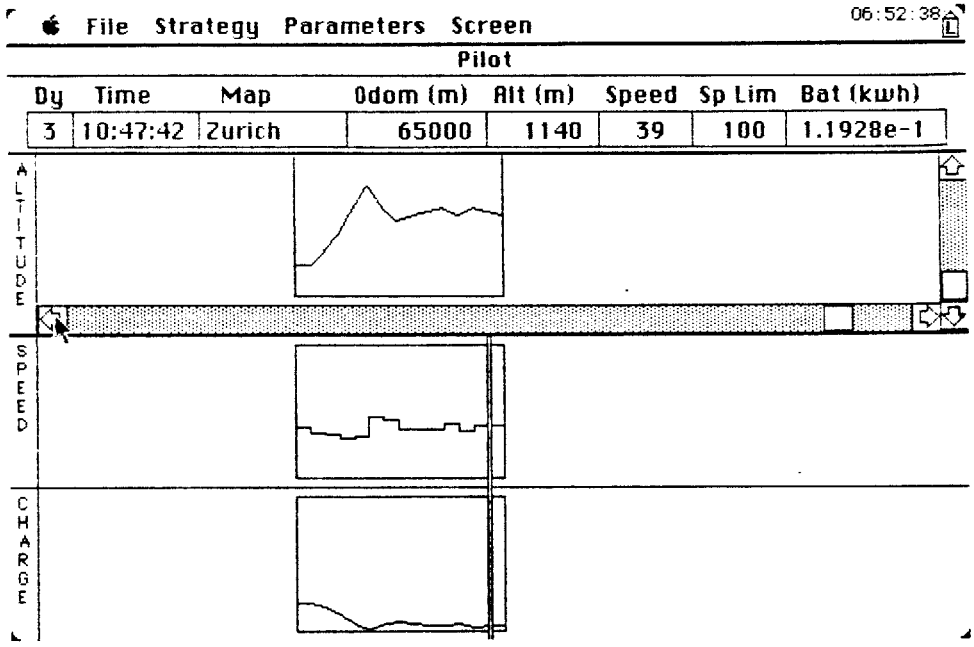


Figure 4-2: Results of Test Case #2

A third test case showed how the algorithm calculated the strategy for a day when laps were permitted. In this example, the initial battery charge was set to 0.9 kwh, and the race began at 9:00. Laps were allowed to be performed immediately after the race was finished and up until 12:00.

Figure 4-3 shows the results of this test case. The average velocity during the race was the optimal 45 kph. There were two laps performed at the minimum speed of 30 kph. After the second lap, the battery was almost empty, so the program ceased to schedule laps even though there

was more time for them. If there had been more initial charge on the battery, then it is expected that laps would have been performed over the entire period in which they were permitted. It is this contingency which the next example explored.

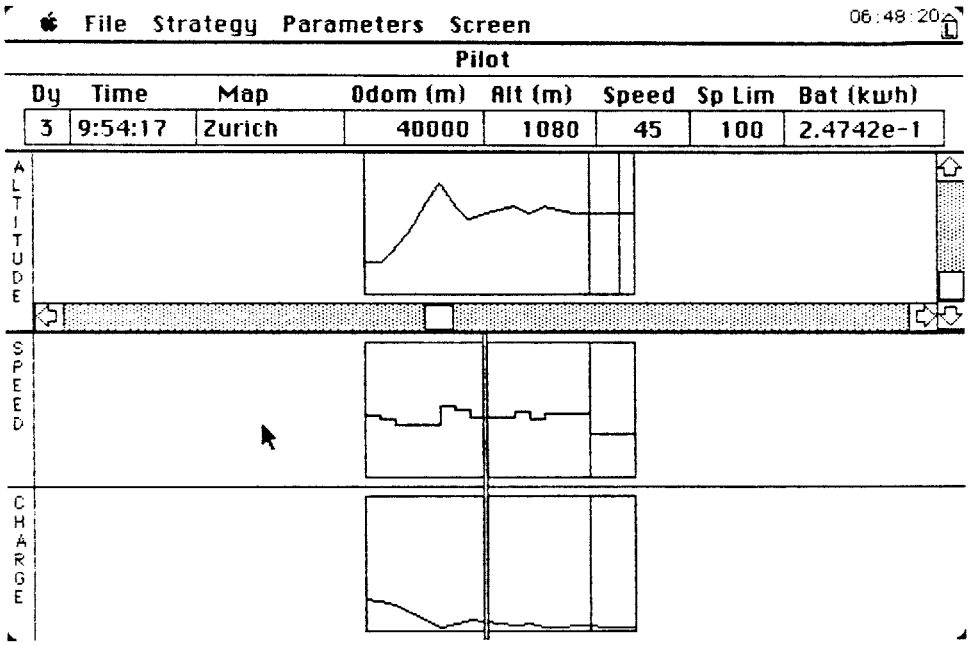


Figure 4-3: Results of Test Case #3

In the fourth test case, the initial battery charge was set to 1 kwh. The resulting strategy is shown in Figure 4-4. The race was still performed at an average speed of 45 kph, however the laps were now performed at an average speed of 38 kph. The increase in the lap average occured because in this case, the period for performing laps ended before the battery was drained. In order to make use of the remaining battery charge, more laps had to be inserted into the strategy. By increasing the speed at

which the laps were performed, the program allowed more laps to be fit into the time allocated for laps.

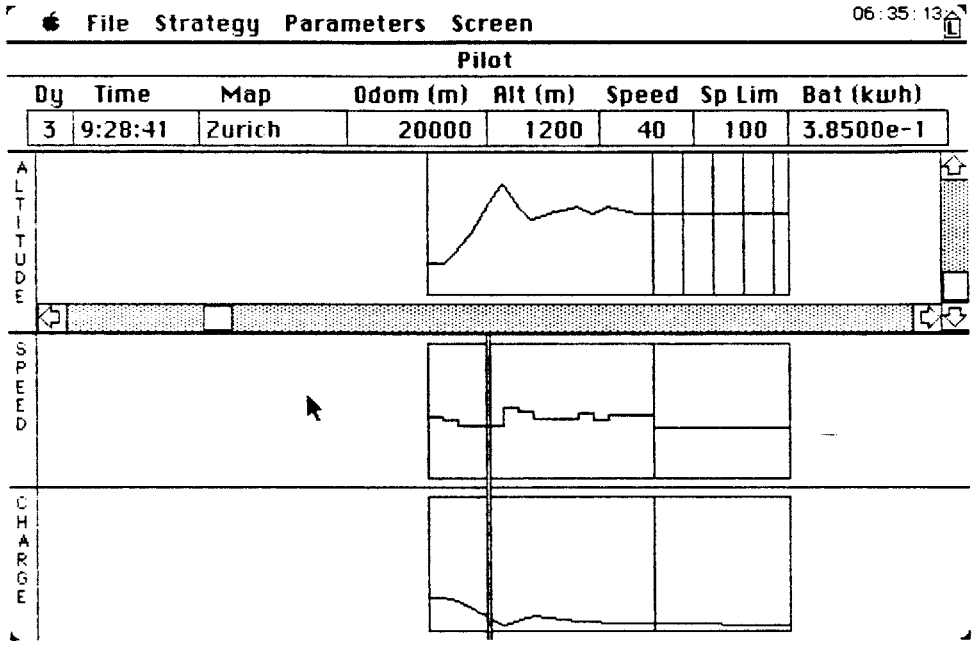


Figure 4-4: Results of Test Case #4

The fifth test case took the fourth case to its logical extreme. This time, the initial battery charge was set to 2 kwh. Figure 4-5 shows the results of this increase in allocated charge. Both the race-section and the lap-section were performed at an average speed of 59 kph. Once the lap average was raised above 45 kph, the algorithm raised both the race average and the lap average together so that they would be performed at roughly the same efficiency. Again, the purpose of increasing the average velocities was to permit more laps to be completed in the allocated time. As before, the speeds were increased until the final battery charge was almost zero.

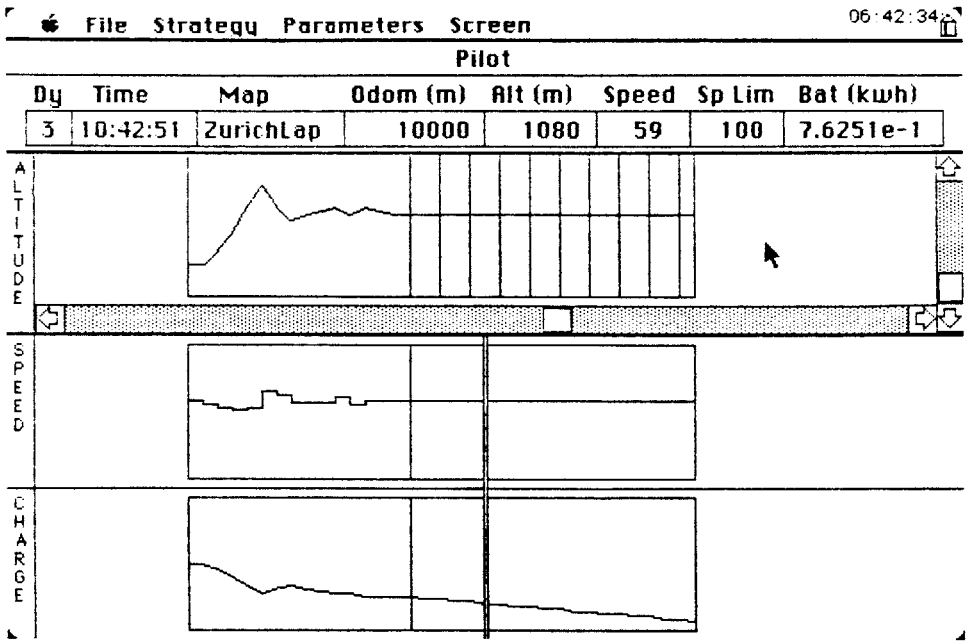


Figure 4-5: Results of Test Case #5

The sixth test case repeated the conditions of the fifth test case, except that the laps were not performed immediately after the race but were delayed until later in the day. The allocated time for laps was set between 12:00 and 15:00. Figure 4-6 shows the graphs of the computed strategy. In this case, the race average was 45 kph and the lap average was 38 kph. The interesting difference between this strategy and the previous one is that, although the lap average was increased as before, the race average was not raised above 45 kph. The reason for this discrepency is that in the sixth case, increasing the race average would not have provided more time to perform laps, since laps were not allowed until later in the day.

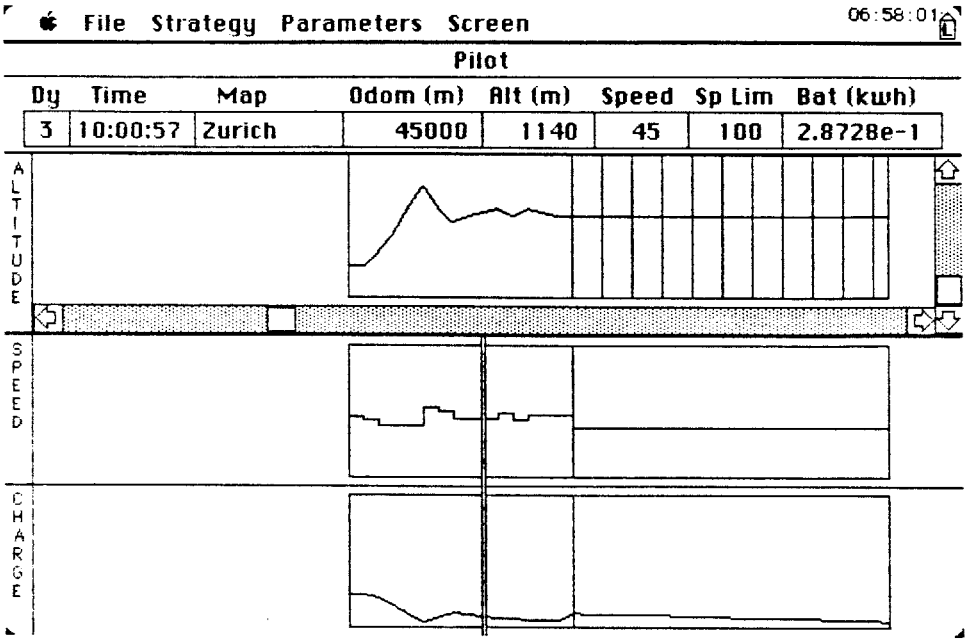


Figure 4-6: Results of Test Case #6

Each test case, containing twenty-three data points, took roughly forty seconds to be calculated. In general, the algorithm's time requirement increases linearly with the number of data points. A six day event such as the Tour de Sol, with a resolution of fifty data points per day, takes approximately ten minutes to be calculated.

# Chapter 5

## Conclusion

The task of finding an optimal racing strategy has been found to be well suited to the use of a computer algorithm. The problem of optimization is fairly simple and well-defined, and can therefore be solved in a systematic way. The strategies produced for the test cases have been shown to be both accurate and optimal. Furthermore, the strategic algorithm generally calculates the entire strategy within fifteen minutes. This time limitation lies within an acceptable range, since recalculation will probably occur only once or twice in a day. A final advantage of using the computer algorithm is that without it, the driver of the vehicle relies heavily on intuition to minimize the car's race time, and this intuition will probably not be as precise as the computer's formula. For these reasons, it is suggested that computer-designed strategies be used for all races such as the Tour de Sol.

## Appendix A

## Derivation of the Efficiency Criterion

Problem: A car must travel over two stretches of road of equal length L. The first stretch is inclined at a slope of $\theta 1$, the second at a slope of $\theta 2$. If the car travels the first stretch at velocity V1 and the second at velocity V2, what relationship must hold between v1 and v2 if the total energy consumed is the minimum possible for the time taken? (Note: $P(V, \theta)$ is the power consumed at velocity V and slope $\theta$; $E(V, \theta)$ is the energy required to travel at V and $\theta$ over distance L.)

$$E(V1,\theta 1) + E(V2,\theta 2) = E(V1+dV1,\theta 1) + E(V2+dV2,\theta 2)$$

$$P(V1,\theta 1)\frac{L}{V1} + P(V2,\theta 2)\frac{L}{V2} = P(V1+dV1,\theta 1)\left(\frac{L}{V1} + dt\right) + P(V2+dV2,\theta 2)\left(\frac{L}{V2} - dt\right)$$

$$P(V1,\theta 1)\frac{L}{V1} + P(V2,\theta 2)\frac{L}{V2} = \left[P(V1,\theta 1) + \frac{d}{dV1}P(V1,\theta 1)dV1\right]\left[\frac{L}{V1} + dt\right] + \left[P(V2,\theta 2) + \frac{d}{dV2}P(V2,\theta 2)dV2\right]\left[\frac{L}{V2} - dt\right]$$

$$P(V1,\theta 1)dt + \frac{d}{dV1}P(V1,\theta 1)\frac{L\,dV1}{V1} = P(V2,\theta 2)dt - \frac{d}{dV2}P(V2,\theta 2)\frac{L\,dV2}{V2}$$

Substitute:

$$dV1 = \frac{V1\ L}{L + V1\ dt} - V1 = \frac{(L - L - V1\ dt)\ V1}{L + V1\ dt} = \frac{-V1^2\ dt}{L}$$

$$dV2 = \frac{V2\ L}{L - V2\ dt} - V2 = \frac{(L - L + V2\ dt)\ V2}{L - V2\ dt} = \frac{-V2^2\ dt}{L}$$

$$P(V1,\Theta1)dt + \frac{d}{dV1}P(V1,\Theta1)\frac{L(-V1^2\ dt)}{V1\ L} = P(V2,\Theta2)dt - \frac{d}{dV2}P(V2,\Theta2)\frac{L(V2^2\ dt)}{V2\ L}$$

$$P(V1,\Theta1) - \frac{d}{dV1}P(V1,\Theta1)V1 = P(V2,\Theta2) - \frac{d}{dV2}P(V2,\Theta2)V2$$

$$Efficiency(V,\Theta) = P(V,\Theta) - V\left(\frac{d}{dV}P(V,\Theta)\right)$$

where Efficiency(V,Θ) is ideally constant for all segments in a speed-average zone.

## Appendix B

## Power Consumption Function of the Racing Vehicle

For a velocity V and a slope $\theta$, the power consumed by the racing vehicle is:

$$A0*V + A1*\sin\theta*V + A2*V^3 + A8*\left|\frac{A3 + A4*\sin\theta + A5*V^2}{A6*A7}\right|, \quad \theta >= 0$$

$$B0*V + B1*\sin\theta*V + B2*V^3 + B8*\left|\frac{B3 + B4*\sin\theta + B5*V^2}{B6*B7}\right|, \quad \theta < 0$$

where:

Power is in units of watt
V is in units of ft/sec

A0 = B0 = 3 (rolling friction)
A1 = B1 = 680 (gravity)
A2 = B2 = .002 (wind resistance)
A3 = B3 = 2.2 (motor/controller losses)
A4 = B4 = 500
A5 = B5 = .0015
A6 = 5.3
B6 = 7
A7 = B7 = .07
A8 = B8 = .015

These figures were derived empirically in tests conducted by Eric Vaaler.

# Appendix C

## Procedure Interface

This appendix outlines the data structures and procedures that make up the Strategic Planner program. The procedures are presented in a notation similar to the interface protocol of the CLU language, indicating the procedure name, its arguments, its return value, which structures it modifies, which procedures it uses, and which errors it signals. The function of each procedure is also briefly explained.

Because the procedures build extensively on one another, they are outlined in order of increasing abstraction to facilitate their interpretation.

### C.1 Data types

The following data types were declared in the global environment:

TEXT is a record of data objects needed to create an editable text window.

PICTURE is a record of data objects needed to create a graphics window.

MAP is a record containing information relevent to a particular section of the race course. It has a Profile

field, which is a TEXT object describing the terrain profile of the race course. MAP also has fields indicating the map name, the distance between Profile's measurements, and the length of Profile in characters.

DAY is a record containing information specific to a particular day of the Tour de Sol. It indicates the ordinality of the day, the start time of race, the start and end times of the laps, the names of the race and lap maps, the time deduction per lap, and the energy expected to be received from the sun.

POINT is a record containing information relevant to a particular point on the race course. POINT indicates the altitude of the point, the recommended speed and the regulated speed limit at the point, and the predicted time and battery charge for when the car passes the point.

## C.2 Important global structures

The following are data objects which are universally accessible and modifiable.

DATA is a text object which is used in loading and storing the editable system parameters. When a file is opened, the characters are read from a text file and stored in Data, from which the system parameters then receive their values. When a file is saved, the parameter values are converted back into text and stored in Data; then Data is written to a text file.

MinRA and MaxRA indicate the limitations on the average speed that can be achieved during a race. MinLA holds the minimum average speed for a qualifying lap. MinBC and MaxBC indicate the limits of the battery's storage capacity.

CalibPt is a record containing information about the calibration point. It indicates the date, time, odometer reading, and battery charge of the calibration.

WattFunc is an array of variables representing the coefficients of the power consumption function.

MapArray is an array of sixteen MAP-type structures.

Itinerary is an array containing the racing schedule for the Tour de Sol. It consists of eight DAY-type structures.

EndCharge is an array of eight variables, which indicates the user-specified end-of-day battery charge for each day.

Points is an array of 600 POINT-type structures, which contain the data relevant to the racing strategy.

The following are arrays of eight variables: RaceStart and RaceLength indicate where in the Points array each race begins and for how many points it lasts. LapStart and LapLength contain similar information for the first lap of each day. RaceAvg and LapAvg indicate the average race and lap speeds for each day; RaceEff and LapEff hold the corresponding efficiencies. NLaps indicates the number of laps to be performed on each day.

XScale and YScale hold the scaling factors of the Altitude, Speed, and Charge graphs.

Pilot is the data structure which holds the information displayed in the pilot window. It contains the date, time, odometer reading, altitude, speed, speed limit, map, and battery charge of the scanned position.

Altitude, Speed, and Charge are the three windows which display the strategy graphs. PilotDP is a dialog pointer for the pilot text window.

Screen is a oneof-type variable, whose settings are "empty" and "strategy".

Abort is a boolean variable. Unlike CLU, Pascal has no error-signalling mechanism, so Abort performs the function of signalling. Abort's normal state is false. When an error is detected, the appropriate error message routine is invoked, which alerts the user and sets Abort true. All subsequent procedures will terminate upon finding Abort set true, until control passes to the main procedure, which sets Abort false.

SkipLaps is a boolean variable which is normally set false. When it is set true, it signals that no laps should be performed on a particular day because there is not enough energy in the battery.

DoQuit is a boolean variable, which when set true will cause the main procedure to terminate.

## C.3 Initializing procedure

The initializing procedure is invoked when the Strategic Planner program is first run.

Init
modifies: all global variables

Init is responsible for setting the global data structures to their initial state. Init's primary functions are creating the menus and windows, setting up an empty screen, opening the resource file, and initializing the global variables to their default values.

## C.4 Error handling procedures

These procedures are invoked when an error occurs. They alert the user to the problem and cause subsequent procedures to terminate.

NumErr
modifies: Abort

NumErr opens an alert box with the message "Non-numeric data entered in a numeric field." The user must click on the box before proceeding. The global Abort is set to true.

IntErr
modifies: Abort

IntErr opens an alert box with the message "Non-integral data entered in an integer field." The user must click on the box before proceeding. The global Abort is set to true.

RangeErr
modifies: Abort

RangeErr opens an alert box with the message "Value

out of range." The user must click on the box before proceeding. The global Abort is set to true.

TimeErr
    modifies: Abort

    TimeErr opens an alert box with the message "Invalid time format." The user must click on the box before proceeding. The global Abort is set to true.

UndMapErr
    modifies: Abort

    UndMapErr opens an alert box with the message "Undeclared map name." The user must click on the box before proceeding. The global Abort is set to true.

LengthErr
    modifies: Abort

    LengthErr opens an alert box with the message "Name or value exceeds 13 characters." The user must click on the box before proceeding. The global Abort is set to true.

SpeedLimErr
    modifies: Abort

    SpeedLimErr opens an alert box with the message "No initial speed limit specified." The user must click on the box before proceeding. The global Abort is set to true.

IntRangeErr
    modifies: Abort

    IntRangeErr opens an alert box with the message "Altitude or speed value out of range." The user must click on the box before proceeding. The global Abort is set to true.

OdomErr
    modifies: Abort

    OdomErr opens an alert box with the message "Invalid

odometer setting in calibration." The user must click on the box before proceeding. The global Abort is set to true.

LowAvgErr
    modifies: Abort

    LowAvgErr opens an alert box with the message "Calculated average falls below minimum speed." The user must click on the box before proceeding. The global Abort is set to true.

C.5 Utility functions

These procedures perform a variety of commonly needed operations.

Trim(DataString: str255)
    returns: str255
    uses: LengthErr
    signals: LengthErr

    Trim returns a string equivalent to DataString with all leading and trailing spaces removed.

Integral(IntString: str255)
    returns: boolean
    uses: Trim

    Integral returns true if and only if IntString denotes an integer.

InRange(IntString: str255)
    returns: boolean
    uses: Trim, Integral

    InRange returns true if and only if IntString denotes an integer less than 1e+9.

Numeric(NumString: str255)
    returns: boolean
    uses: Trim, Integral

Numeric returns true if and only if NumString denotes an integer or real number.

```
ReadIntString(IntString: str255;  var Int: longint)
    uses: Trim, Integral, InRange, RangeErr, IntErr
    signals: IntErr, RangeErr
```

ReadIntString modifies Int by setting it to the value denoted by IntString. IntErr is signalled if IntString does not denote an integer, and RangeErr is signalled if the integer is out of range.

```
ReadExtString(ExtString: str255;  var Ext: extended)
    modifies: Ext
    uses: Trim, Numeric, NumErr
    signals: NumErr
```

ReadExtString modifies Ext by setting it to the value denoted by ExtString. NumErr is signalled if ExtString does not denote a real number.

```
IntToLongTime(Int: longint)
    returns: str255
```

IntToLongTime converts Int to a string in long-time format and returns this string. Int denotes a time in units of seconds after midnight. Long-time format consists of the corresponding hours, minutes, and remaining seconds, separated by colons.

```
IntToShortTime(Int: longint)
    returns: str255
```

IntToShortTime converts Int to a string in short-time format and returns this string. Int denotes a time in units of seconds after midnight. Short-time format consists of the corresponding hours and minutes, separated by a colon.

```
TimeToInt(TimeString: str255)
    returns: longint
    uses: Trim, TimeErr, ReadIntString
    signals: TimeErr
```

TimeToInt converts TimeString from either long- or short-time format to an integer whose value is the time

in seconds after midnight. This integer is returned. TimeErr is signalled if TimeString is not properly formatted.

TextLength(TEH: TEHandle)
    returns: longint

TextLength returns the number of characters in the text-edit record pointed to by TEH.

KPH(Vel: extended)
    returns: extended

KPH converts the velocity Vel from units of meters/sec to km/hr, and returns the converted value.

MPS(Vel: extended)
    returns: extended

MPS converts the velocity Vel from units of km/hr to meters/sec, and returns the converted value.

FPS(Vel: extended)
    returns: extended

FPS converts the velocity Vel from units of km/hr to ft/sec, and returns the converted value.

## C.6 Screen maintenance procedures

These procedures are responsible for altering and updating the windows displayed on the screen.

UpdatePilot
    modifies: PilotDP

UpdatePilot updates the values in the PilotDP window with data from the Pilot record.

StrategyScreen

modifies: PilotDP, Altitude, Speed, Charge, Screen
uses: UpdatePilot

StrategyScreen displays the four strategy-relevant windows (Strategy, Altitude, Speed, and Charge) and enables all menus. The global Screen is set to strategy.

BlankScreen
modifies: PilotDP, Altitude, Speed, Charge, Screen

BlankScreen creates a blank video screen by hiding all visible windows and disabling all menus. The global Screen is set to empty.

Instructions

Instructions displays the Instruction window, which briefs the user on the basics of using the Planner program.

## C.7 Strategy calculation procedures

The following procedures are used to compute the racing strategy.

Power(V, theta: extended)
returns: extended
uses: FPS

Power uses the coefficients in the WattFunc record to compute the power consumed by the car travelling at velocity V and at slope theta. (The formula is provided in Appendix B.) This power value is returned.

Efficiency(V, theta: extended)
returns: extended
uses: FPS

Efficiency computes the efficiency rating for the car travelling at velocity V and at slope theta. (The formula and its derivation is provided in Appendix A.) This efficiency rating is returned.

```
CalcSpeed(Eff, theta: extended)
    returns: extended
    uses: Efficiency
```

CalcSpeed returns the speed at which the car, travelling at slope theta, would have the efficiency rating Eff.

```
ReCharge(T1, T2: longint;  ETotal: extended)
    returns: extended
```

ReCharge returns the amount of charge received from the sun between times T1 and T2 (required to be on the same day). ReCharge assumes that the day's total recharge, ETotal, is distributed sinusoidally between 6:00 AM and 6:00 PM.

```
CalcInterval(Date: integer;  Start, Finish, dx: integer;
             IdealVel: extended)
    returns: extended
    modifies: Points
    uses: CalcSpeed, SpeedLimErr, MPS, Power, ReCharge,
          KPH
    signals: SpeedLimErr
```

CalcInterval calculates all the non-altitude fields of the Points array between the indices Start and Finish, such that the average velocity is IdealVel, and so that the energy consumed for that average is minimized. Dx is the horizontal distance between altitude measurements. The returned value is the efficiency at which the inteval has been calculated. This procedure requires that the altitude measurements between Start and Finish have already been provided. SpeedLimErr is signalled if no initial speed limit has been provided.

```
GetMap(MapName: str255)
    returns: map
    uses: UndMapErr
    signals: UndMapErr
```

GetMap searches MapArray for the map named MapName and returns that map. UndMapErr is signalled if no such map exists.

```
ReadProfile(var PtIndex: longint;  Profile: text)
    modifies: PtIndex, Points
```

uses: LengthErr, Trim, ReadIntString, IntRangeErr
signals: LengthErr, IntRangeErr

ReadProfile converts the altitude measurements contained in Profile from text to numeric values. These values are stored in the Points array starting at PtIndex, which is assumed to point to the first empty space in the array. PtIndex is then incremented to point to the new first empty space. LengthErr is signalled if any of Profile's entries exceeds 13 characters. IntRangeErr is signalled if any of the entries is out of range.

CalcRace(WhichDay: integer)
modifies: Points, RaceStart, RaceLength, RaceAvg, RaceEff, SkipLaps
uses: GetMap, OdomErr, ReCharge, CalcInterval, KPH, LowAvgErr
signals: OdomErr, LowAvgErr

CalcRace calculates all the non-altitude fields of the Points array over the interval corresponding to the race portion of WhichDay, minimizing both race time and energy consumption. Initially, CalcRace aims for an average speed of MaxRA; however, if this speed causes the battery to be completely drained, then CalcRace aims for the maximum average speed that will not exhaust the battery. OdomErr is signalled if the odometer setting in CalibPt is invalid. LowAvgErr is signalled if the average velocity falls below MinRA. This procedure requires that the altitude measurements for the race have already been provided.

CalcLap(WhichDay: integer)
modifies: Points, LapStart, LapLength, LapAvg, LapEff
uses: GetMap, ReCharge, CalcInterval, KPH

CalcLap calculates all the non-altitude fields of the Points array for the first lap of WhichDay. (Subsequent laps are computed with RepeatLap.) CalcLap aims for an average speed of MinLA in order to minimize the energy consumed. CalcLap requires that the altitude measurements for the lap have already been provided.

RepeatLap(var PtIndex: longint; WhichDay: integer)
modifies: PtIndex, Points, NLaps
uses: GetMap, Power, ReCharge

RepeatLap causes an additional lap to be scheduled for

WhichDay. The last existing lap is copied onto the end of the Points array, with the appropriate changes made to the time and charge fields. PtIndex is incremented to point to the new end of the array.

```
EraseLap(var PtIndex: longint;  WhichDay: integer)
    modifies: PtIndex, Points, NLaps
```

EraseLap erases the last lap in the Points array by setting all of its fields to zero. PtIndex is decremented to point to the new end of the array.

```
DoLaps(WhichDay: integer)
    returns: bool
```

DoLaps returns true if and only if a lap map has been scheduled on the Itinerary for WhichDay.

```
MorningCharge(PtIndex: longint;  WhichDay: integer)
    returns: extended
    uses: ReCharge
```

MorningCharge returns the amount of charge that will be in the battery when the race begins on WhichDay.

```
CalcPilot
    modifies: Pilot
    uses: GetMap
```

CalcPilot updates the values in the Pilot record, based on the current ScrollValue and current values in the Points array.

```
LowerRedraw
    modifies: Speed, Charge
    uses: DoLaps
```

LowerRedraw redraws the lower two graphics windows (Speed and Charge). Because these windows do not scroll, they must be completely redrawn whenever the Altitude window is scrolled so that the three windows will remain in alignment.

```
ReDraw
    modifies: Altitude, Speed, Charge
```

uses: DoLaps, LowerRedraw, CalcPilot, UpdatePilot

ReDraw redraws all three graphics windows (Altitude, Speed, and Charge), using data contained in the Points array. ReDraw also updates the PilotDP window.

Recalculate
modifies: Points
uses: GetMap, ReadProfile, CalcRace, DoLaps, MorningCharge, CalcLap, RepeatLap, EraseLap, ReDraw

Recalculate computes the entire racing strategy from scratch, starting from the location indicated by CalibPt. The algorithm used for this computation is described in Chapter 2.

ExpandX
modifies: XScale
uses: ReDraw

ExpandX causes the three graphs (Altitude, Speed, and Charge) to double in size horizontally, unless doing so expands the points to be drawn beyond integer range.

ExpandY
modifies: YScale
uses: ReDraw

ExpandY causes the Altitude graph to double in size vertically, unless doing so expands the points to be drawn beyond integer range.

ContractX
modifies: XScale
uses: ReDraw

ContractX contracts the three graphs (Altitude, Speed, and Charge) by a factor of two horizontally, unless XScale is equal to one.

ContractY
modifies: YScale
uses: ReDraw

ContractY contracts the Altitude graph by a factor of

two vertically, unless YScale is equal to one.

## C.8 File handling procedures

These procedures are used for loading and storing the editable parameters of the strategic algorithm.

```
ReadDataString(var DataString: str255)
    modifies: DataString
    uses: LengthErr
    signals: LengthErr
```

ReadDataString reads the first string in Data's text-edit record, and stores a copy in DataString. The original string is then deleted from Data. LengthErr is signalled if the string exceeds 13 characters.

```
ReadInt(var Int: longint)
    modifies: Int
    uses: ReadDataString, ReadIntString
```

ReadInt reads the first string in Data, converts it to an extended, and stores the value in Int.

```
ReadExt(var Ext: extended)
    modifies: Ext
    uses: ReadDataString, ReadExtString
```

ReadExt reads the first string in Data, converts it to an extended, and stores the value in Ext.

```
ReadData
    modifies: Data, MinRA, MaxRA, MinLA, MinBC, MaxBC,
              CalibPt, WattFunc, MapArray, Itinerary
    uses: ReadInt, ReadExt, ReadDataString
```

ReadData reads the editable system parameters (speed and charge limits, calibration, power function, maps, and itinerary) from Data's text-edit record. These parameters are converted from text to numeric values and are stored in the appropriate data structures.

OpenFile
    uses: ReadData, ReCalculate, StrategyScreen,
          UpdatePilot

    OpenFile opens a dialog box requesting a filename from the user. The file is opened and its characters are read into Data's text-edit record. OpenFile then reads the system parameters, recalculates the strategy, and displays the strategy windows.

WriteDataString(DataString: str255)
    modifies: Data

    WriteDataString inserts a copy of DataString at the end of Data's text-edit record.

WriteInt(Int: longint)
    uses: WriteDataString, Trim

    WriteInt converts Int to a string, trims it, adds a carriage return, and passes it to WriteDataString.

WriteExt(Ext: extended)
    uses: WriteDataString, Trim

    WriteExt converts Ext to a string, trims it, adds a carriage return, and passes it to WriteDataString.

WriteData
    modifies: Data
    uses: WriteInt, WriteExt, WriteDataString, TextLength

    WriteData takes the editable system parameters (speed and charge limits, calibration, power function, maps, and itinerary), converts them to text, and writes them to Data's text-edit record.

SaveFile
    uses: WriteData

    SaveFile invokes WriteData and then saves Data's text-edit record to the currently active file.

SaveFileAs
    uses: WriteData

SaveFileAs is the same as SaveFile, except that it opens a dialog box requesting the name of the file to be saved.

SaveAlert
    modifies: DoQuit
    uses: SaveFile

SaveAlert opens a dialog box with three options: quit, save and quit, or cancel. If the second option is selected then SaveFile is invoked. If either the first or second option is selected, then the global DoQuit is set to true, causing the main procedure to terminate.

Quit
    modifies: Altitude, Speed, Charge
    uses: SaveAlert, BlankScreen

Quit invokes SaveAlert, and if DoQuit is set to true, then Quit closes all the windows and the resource file.

## C.9 Editing procedures

The following procedures allow the user to edit the inputs to the strategy-calculating algorithm.

SelectLimits
    modifies: MinRA, MaxRA, MinLA, MinBC, MaxBC
    uses: BlankScreen, ReadIntString, ReadExtString, StrategyScreen

SelectLimits opens a dialog box displaying the speed and charge limitations (MinRA, MaxRA, MinLA, MinBC, MaxBC). The user is free to edit these values until the OK or Cancel button is selected. OK will cause the edited values to be written back to the variables. The dialog box is then closed.

SelectCalibration
    modifies: CalibPt
    uses: BlankScreen, ReadIntString, ReadExtString, StrategyScreen, IntToLongTime, TimeToInt

SelectCalibration opens a dialog box displaying the fields of CalibPt (Date, Time, Odometer, and Charge). The user is free to edit these values until the OK or Cancel button is selected. OK will cause the edited values to be written back to CalibPt. The dialog box is then closed.

SelectWattFunc
    modifies: WattFunc
    uses: BlankScreen, ReadExtString, StrategyScreen

SelectWattFunc opens a dialog box displaying the coefficients of the power consumption function. The user is free to edit these values until the OK or Cancel button is selected. OK will cause the edited values to be written back to WattFunc. The dialog box is then closed.

EditMap(i: integer)
    modifies: MapArray
    uses: Trim, ReadIntString

EditMap opens a dialog box displaying the Name and DeltaX fields of the ith map, which the user may edit. An "Edit Profile" button is also provided, which if selected will open the Profile window and allow its data to be edited. The Profile window will close when its go-away box is clicked, and the dialog box will close when OK or Cancel is selected.

SelectMaps
    uses: BlankScreen, EditMap, StrategyScreen

SelectMaps opens a dialog box displaying up to sixteen map names. When one of these is selected, EditMap is invoked for that map. The dialog box closes when OK is selected.

SelectItinerary
    modifies: Itinerary
    uses: BlankScreen, ReadIntString, ReadExtString,
          StrategyScreen, Trim, TimeToInt, IntToShortTime

SelectItinerary opens a dialog box displaying the fields of Itinerary. The user is free to edit these values until the OK or Cancel button is selected. OK will cause the edited values to be written back to

Itinerary. The dialog box is then closed.

SelectEndCharges
    modifies: EndCharge
    uses: BlankScreen, ReadExtString, StrategyScreen

SelectEndCharges opens a dialog box displaying the desired end-of-day battery charge for each day. The user is free to edit these values until the OK or Cancel button is selected. OK will cause the edited values to be written back to EndCharge. The dialog box is then closed.

SelectResults
    uses: BlankScreen, StrategyScreen

SelectResults opens a dialog box diplaying the calculation reults (race average and efficiency, lap average and efficiency, and number of laps, for each day). These values are not editable. The dialog box is closed when OK is selected.

## C.10 The main procedure

The main procedure is the actual Strategic Planner program. It makes use of all the other procedures to execute its tasks.

Planner
    modifies: Abort
    uses: Init, Instructions, OpenFile, SaveFile,
          SaveFileAs, Quit, Recalculate, SelectResults,
          ExpandX, ContractX, ExpandY, ContractY,
          LowerRedraw, CalcPilot, UpdatePilot

When the Tour de Sol '89 Strategic Planner program is run, Planner is the first procedure to be invoked. Planner does the following: First, it calls Init to set up the data structures. It then enters a loop which sets the Abort flag false and waits for a user action. If the user selects a menu item, then Planner executes the corresponding procedure. If the user scrolls the Altitude window, then Planner scrolls the Speed and Charge windows to maintain alignment. The loop repeats until the flag

DoQuit is set true, at which point the program terminates.

# Appendix D

## Pascal Code for the Strategic Planner Program

```
unit Planner_init;

interface

  uses
    XTTypeDefs, Extender1;

  const
    checkmark = 18;
    blank = 32;
    null = 0;
    tab = 9;
    space = 32;
    return = 13;
    semi = 59;
    eof = 0;
    pi = 3.141592654;
    hr12 = 43200;
    dawn = 21600;
    dusk = 64800;
    maxPtIndex = 500;

    LimitsID = 10001;
    CalibrationID = 10002;
    WattFuncID = 10003;
    MapsID = 10004;
    MapEditID = 10005;
    ItineraryID = 10006;
    PilotID = 10007;
    ResultsID = 10008;
    EndChargesID = 10009;

    NumErrID = 20001;
    IntErrID = 20002;
    RangeErrID = 20003;
    TimeErrID = 20004;
    UndMapErrID = 20005;
    LengthErrID = 20006;
    SaveAlertID = 20007;
    SpeedLimErrID = 20008;
    IntRangeErrID = 20009;
    OdomErrID = 20010;
    LowAvgErrID = 20011;

  type

    text = record
       WPtr : WindowPtr;
       WR : WindowRecord;
       WD : WData;
       WRect, DRect, VRect : Rect;
     end;
```

```
  picture = record
      WPtr : WindowPtr;
      WR : WindowRecord;
      WD : WData;
      WRect, PRect : Rect;
      PicHndl : PicHandle;
    end;

  map = record
      Name : str255;
      DeltaX : longint;
      Length : longint;
      Profile : text;
    end;

  day = record
      Date, RaceBegins, LapBegins, LapEnds : longint;
      WattsReceived : extended;
      RaceMap, LapMap : str255;
      Deduction : longint;
    end;

  point = record
      Time : longint;
      Altitude : integer;
      Speed, SpeedLimit : integer;
      Charge : single;
    end;

var
  iBeam, cross, plus, watch : Cursor;
  AppleMenu, FileMenu, StrategyMenu, ParamMenu, ScreenMenu : MenuHandle;
  FilePresent : boolean;
  Abort : boolean;
  Reply : SFReply;
  Error : OSErr;
  FRefNum : integer;
  Data : text;
  ScrollValue : longint;
  PointValue : longint;
  HScroll, VScroll : ControlHandle;

  MinRA, MaxRA, MinLA : longint;
  MinBC, MaxBC : extended;
  CalibPt : record
      Date, Time : longint;
      Odometer : longint;
      Charge : extended;
    end;
  WattFunc : record
      A : array[0..8] of extended;
      B : array[0..8] of extended;
    end;
```

```
    MapArray : array[1..16] of map;
    Itinerary : array[1..8] of day;
    EndCharge : array[1..8] of extended;
    Pilot : record
        Date, Time : longint;
        Map : str255;
        Odometer, Altitude : longint;
        Speed, SpeedLimit : longint;
        Charge : extended;
      end;

    Points : array[0..600] of point;
    NPts : longint;
    RaceStart, RaceLength, LapStart, LapLength : array[1..8] of longint;
    RaceAvg, RaceEff, LapAvg, LapEff : array[1..8] of extended;
    NLaps : array[1..8] of longint;
    NDays : integer;

    LimitsDP, CalibrationDP, WattFuncDP, MapsDP, MapEditDP, ItineraryDP, EndChargesDP, PilotDP,
        ResultsDP : DialogPtr;
    Altitude, Speed, Charge : picture;
    AltLabel, SpdLabel, ChgLabel : picture;
    Hash : picture;

    Screen : (empty, strategy);
    DoQuit : boolean;
    SkipLaps : boolean;
    DoAllLaps : boolean;


{XScale,YScale declared in Extender1; WindowClosed in XTTypeDefs}

  procedure Init;

implementation

  procedure Init;
    var
      i : integer;
  begin
    XTendInit;
    FetchCursors(iBeam, cross, plus, watch);
    SetCursor(watch);
    HideAll;
    DoQuit := false;
    AppleMenu := NewMenu(11, chr(20));
    AppendMenu(appleMenu, '(About...;(-');
    AddResMenu(appleMenu, 'DRVR');
    InsertMenu(appleMenu, 0);
    FileMenu := NewMenu(12, 'File');
    StrategyMenu := NewMenu(21, 'Strategy');
    ParamMenu := NewMenu(22, 'Parameters');
    ScreenMenu := NewMenu(23, 'Screen');
```

```
AppendMenu(FileMenu, 'Open...;-;Save;Save As...;-;Quit');
AppendMenu(StrategyMenu, 'Recalculate;Show Results;Expand x;Contract x;Expand y;Contract y');
AppendMenu(ParamMenu, 'Limits;Calibration;Watt Func;Maps;Itinerary;End Charges');
AppendMenu(ScreenMenu, 'Strategy');
SetMenuItem(AppleMenu, 1, 'About Planner');
InsertMenu(Filemenu, 0);
InsertMenu(StrategyMenu, 0);
InsertMenu(ParamMenu, 0);
InsertMenu(ScreenMenu, 0);
DrawMenuBar;
EnableItem(Applemenu, 1);
DisableItem(FileMenu, 2);
DisableItem(FileMenu, 3);
DisableItem(FileMenu, 4);
DisableItem(FileMenu, 5);
EnableAllItems(StrategyMenu, false);
EnableAllItems(ParamMenu, false);
EnableAllItems(ScreenMenu, false);
FilePresent := false;
Abort := false;
Screen := empty;

FRefNum := OpenResFile('Planner.rsrc');
If (FRefNum < 0) then
  SysBeep(100);

SetRect(Data.WRect, 0, 0, 52, 100);
SetRect(Data.DRect, 0, 0, 32, 5000);
SetRect(Data.VRect, 0, 0, 32, 80);
Data.WPtr := CreateWindow(Data.WR, Data.WRect, 'Data', 0, false, true, true, true, true);
TEMakeNew(Data.WPtr, Data.DRect, Data.VRect);
GetWData(Data.WPtr, Data.WD);
XScale := 1;
YScale := 1;
ScrollValue := 0;
PointValue := 0;
for i := 1 to 16 do
  with MapArray[i].Profile do
    begin
      SetRect(WRect, 10, 50, 250, 330);
      SetRect(DRect, 4, 0, 220, 2000);
      SetRect(VRect, 4, 0, 220, 260);
      WPtr := CreateWindow(WR, WRect, 'Profile', 0, false, true, false, true, true);
      TEMakeNew(WPtr, DRect, VRect);
      GetWData(WPtr, WD);
    end;

with Speed do
  begin
    SetRect(WRect, 15, 179, 512, 274);
    WPtr := CreateWindow(WR, WRect, 'Speed', 2, false, false, false, true, true);
    GetWData(WPtr, WD);
    WD.hScrollBar^^.contrlVis := 0;
```

```
    WD.vScrollBar^^.contrlVis := 0;
    Error := SetEx(WPtr, EXCPTactivate, true);
  end;

with Charge do
  begin
    SetRect(WRect, 15, 261, 512, 357);
    WPtr := CreateWindow(WR, WRect, 'Charge', 2, false, false, false, true, true);
    GetWData(WPtr, WD);
    WD.hScrollBar^^.contrlVis := 0;
    WD.vScrollBar^^.contrlVis := 0;
    Error := SetEx(WPtr, EXCPTactivate, true);
  end;


with Hash do
  begin
    SetRect(WRect, 256, 179, 257, 342);
    WPtr := CreateWindow(WR, WRect, '', 2, false, false, false, false, false);
    GetWData(WPtr, WD);
    Error := SetEx(WPtr, EXCPTactivate, true);
  end;

with Altitude do
  begin
    SetRect(WRect, 15, 81, 512, 177);
    WPtr := CreateWindow(WR, WRect, 'Altitude', 2, false, false, false, true, true);
    GetWData(WPtr, WD);
  end;

with AltLabel do
  begin
    SetRect(WRect, 0, 81, 14, 177);
    WPtr := CreateWindow(WR, WRect, 'AltLabel', 2, false, false, false, false, false);
    GetWData(WPtr, WD);
    Error := SetEx(WPtr, EXCPTactivate, true);
    SetPort(WPtr);
    SetRect(PRect, 0, 0, 14, 88);
    PicHndl := OpenPicture(PRect);
    TextSize(9);
    MoveTo(4, 11);
    DrawChar('A');
    MoveTo(4, 21);
    DrawChar('L');
    MoveTo(4, 31);
    DrawChar('T');
    MoveTo(4, 41);
    DrawChar('I');
    MoveTo(4, 51);
    DrawChar('T');
    MoveTo(4, 61);
    DrawChar('U');
    MoveTo(4, 71);
```

```
      DrawChar('D');
      MoveTo(4, 81);
      DrawChar('E');
      ClosePicture;
      SetWPic(WPtr, PicHndl);
    end;

  with SpdLabel do
    begin
      SetRect(WRect, 0, 179, 14, 260);
      WPtr := CreateWindow(WR, WRect, 'SpdLabel', 2, false, false, false, false, false);
      GetWData(WPtr, WD);
      Error := SetEx(WPtr, EXCPTactivate, true);
      SetPort(WPtr);
      SetRect(PRect, 0, 0, 14, 73);
      PicHndl := OpenPicture(PRect);
      TextSize(9);
      MoveTo(4, 11);
      DrawChar('S');
      MoveTo(4, 21);
      DrawChar('P');
      MoveTo(4, 31);
      DrawChar('E');
      MoveTo(4, 41);
      DrawChar('E');
      MoveTo(4, 51);
      DrawChar('D');
      ClosePicture;
      SetWPic(WPtr, PicHndl);
    end;

  with ChgLabel do
    begin
      SetRect(WRect, 0, 261, 14, 342);
      WPtr := CreateWindow(WR, WRect, 'ChgLabel', 2, false, false, false, false, false);
      GetWData(WPtr, WD);
      Error := SetEx(WPtr, EXCPTactivate, true);
      SetPort(WPtr);
      SetRect(PRect, 0, 0, 14, 73);
      PicHndl := OpenPicture(PRect);
      TextSize(9);
      MoveTo(4, 11);
      DrawChar('C');
      MoveTo(4, 21);
      DrawChar('H');
      MoveTo(4, 31);
      DrawChar('A');
      MoveTo(4, 41);
      DrawChar('R');
      MoveTo(4, 51);
      DrawChar('G');
      MoveTo(4, 61);
      DrawChar('E');
```

```
      ClosePicture;
      SetWPic(WPtr, PicHndl);
    end;
  SelectWindow(Altitude.WPtr);
  HScroll := Altitude.WD.hScrollBar;
  VScroll := Altitude.WD.vScrollBar;

  SetCursor(arrow);
 end;
end.
```

```
unit Planner_err;

interface

  uses
    XTTypeDefs, Extender1, Planner_init;

  procedure NumErr;
  procedure IntErr;
  procedure RangeErr;
  procedure TimeErr;
  procedure UndMapErr;
  procedure LengthErr;
  procedure SpeedLimErr;
  procedure IntRangeErr;
  procedure OdomErr;
  procedure LowAvgErr;

implementation

  procedure NumErr;
    var
      DP : DialogPtr;
      Item : integer;

  begin
    SetCursor(arrow);
    DP := GetNewDialog(NumErrID, nil, pointer(-1));
    ShowWindow(DP);
    repeat
      ModalDialog(nil, Item);
      CheckDItem(DP, Item);
    until Item = 1;
    DisposDialog(DP);
    Abort := true;
    SetCursor(watch);
  end;

  procedure IntErr;
    var
      DP : DialogPtr;
      Item : integer;

  begin
    SetCursor(arrow);
    DP := GetNewDialog(IntErrID, nil, pointer(-1));
    ShowWindow(DP);
    repeat
      ModalDialog(nil, Item);
      CheckDItem(DP, Item);
    until Item = 1;
    DisposDialog(DP);
    Abort := true;
```

```
  SetCursor(watch);
end;

procedure RangeErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(RangeErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
  DisposDialog(DP);
  Abort := true;
  SetCursor(watch);
end;

procedure TimeErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(TimeErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
  DisposDialog(DP);
  Abort := true;
  SetCursor(watch);
end;

procedure UndMapErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(UndMapErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
  DisposDialog(DP);
```

```
  Abort := true;
  SetCursor(watch);
end;

procedure LengthErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(LengthErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
  DisposDialog(DP);
  Abort := true;
  SetCursor(watch);
end;

procedure SpeedLimErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(SpeedLimErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
  DisposDialog(DP);
  Abort := true;
  SetCursor(watch);
end;

procedure IntRangeErr;
  var
    DP : DialogPtr;
    Item : integer;

begin
  SetCursor(arrow);
  DP := GetNewDialog(IntRangeErrID, nil, pointer(-1));
  ShowWindow(DP);
  repeat
    ModalDialog(nil, Item);
    CheckDItem(DP, Item);
  until Item = 1;
```

```
    DisposDialog(DP);
    Abort := true;
    SetCursor(watch);
  end;

  procedure OdomErr;
    var
      DP : DialogPtr;
      Item : integer;

  begin
    SetCursor(arrow);
    DP := GetNewDialog(OdomErrID, nil, pointer(-1));
    ShowWindow(DP);
    repeat
      ModalDialog(nil, Item);
      CheckDItem(DP, Item);
    until Item = 1;
    DisposDialog(DP);
    Abort := true;
    SetCursor(watch);
  end;

  procedure LowAvgErr;
    var
      DP : DialogPtr;
      Item : integer;

  begin
    SetCursor(arrow);
    DP := GetNewDialog(LowAvgErrID, nil, pointer(-1));
    ShowWindow(DP);
    repeat
      ModalDialog(nil, Item);
      CheckDItem(DP, Item);
    until Item = 1;
    DisposDialog(DP);
    SetCursor(watch);
  end;

end.
```

```
unit Planner_misc;

interface

  uses
    XTTypeDefs, Extender1, Planner_init, Planner_err;

  procedure ReadIntString (IntString : str255;
                var Int : longint);
  procedure ReadExtString (ExtString : str255;
                var Ext : extended);
  function Integral (IntString : str255) : boolean;
  function InRange (IntString : str255) : boolean;
  function Numeric (NumString : str255) : boolean;
  function TimeToInt (TimeString : str255) : longint;
  function IntToLongTime (Int : longint) : str255;
  function IntToShortTime (Int : longint) : str255;
  function Trim (DataString : str255) : str255;
  function TextLength (TEH : TEHandle) : longint;
  function KPH (Vel : extended) : extended;
  function MPS (Vel : extended) : extended;
  function FPS (Vel : extended) : extended;
  function sign (n : extended) : integer;

implementation

  procedure ReadIntString;

  begin
    If Trim(IntString) = '' then
      Int := 0
    else if Integral(IntString) then
      if InRange(IntString) then
        ReadString(IntString, Int)
      else
        RangeErr
    else
      IntErr;
  end;

  procedure ReadExtString;

  begin
    If Trim(ExtString) = '' then
      Ext := 0
    else if Numeric(ExtString) then
      ReadString(ExtString, Ext)
    else
      NumErr;
  end;

  function Integral;
    var
```

```
    EditString : str255;
    i : integer;

begin
  EditString := Trim(IntString);
  if EditString = '' then
    Integral := false
  else
    begin
      if (EditString[1] = '-') or (EditString[1] = '+') then
        delete(EditString, 1, 1);
      Integral := true;
      for i := 1 to length(EditString) do
        if (EditString[i] < '0') or (EditString[i] > '9') then
          Integral := false;
    end;
end;

function InRange;
  var
    EditString : str255;

begin
  EditString := Trim(IntString);
  if EditString = '' then
    InRange := false
  else
    begin
      if (EditString[1] = '-') or (EditString[1] = '+') then
        delete(EditString, 1, 1);
      InRange := true;
      if not Integral(EditString) then
        InRange := false
      else if length(EditString) > 9 then
        InRange := false;
    end;
end;

function Numeric;
  var
    EditString : str255;
    EditString1, EditString2 : str255;
    i : integer;

begin
  EditString := Trim(NumString);
  if EditString = '' then
    Numeric := false
  else
    begin
      Numeric := true;
      i := pos('e', EditString);
      if i = 0 then
```

```
      begin
        i := pos('.', EditString);
        if i = 0 then
          Numeric := Integral(EditString)
        else
          Numeric := Integral(omit(EditString, i, 1));
      end
    else
      begin
        EditString1 := copy(EditString, 1, i - 1);
        EditString2 := copy(EditString, i + 1, length(EditString) - i);
        i := pos('.', EditString1);
        if i = 0 then
          Numeric := Integral(EditString1) and Integral(EditString2)
        else
          Numeric := Integral(omit(EditString1, i, 1)) and Integral(EditString2);
      end;
  end;
end;

function TimeToInt;
  var
    EditString : str255;
    hour, minute, second : longint;
    i : integer;

begin
  EditString := Trim(copy(TimeString, 1, length(TimeString)));
  if EditString = '' then
    TimeToInt := 0
  else
    begin
      i := pos(':', EditString);
      if i = 0 then
        TimeErr
      else
        begin
          ReadIntString(omit(EditString, i, length(EditString) - i + 1), hour);
          Delete(EditString, 1, i);
          i := pos(':', EditString);
          if i = 0 then
            begin
              ReadIntString(EditString, minute);
              TimeToInt := hour * 3600 + minute * 60;
            end
          else
            begin
              ReadIntString(omit(EditString, i, length(EditString) - i + 1), minute);
              Delete(EditString, 1, i);
              ReadIntString(EditString, second);
              TimeToInt := hour * 3600 + minute * 60 + second;
            end;
        end;
```

```
    end;
  end;

  function IntToLongTime;
    var
      HourString, MinuteString, SecondString : str255;

  begin
    HourString := trim(stringof(trunc(Int / 3600) : 6));
    MinuteString := trim(stringof(trunc(Int / 60 - trunc(Int / 3600) * 60) : 6));
    SecondString := trim(stringof((Int - trunc(Int / 60) * 60) : 6));
    If length(MinuteString) = 1 then
      MinuteString := concat('0', MinuteString);
    If length(SecondString) = 1 then
      SecondString := concat('0', SecondString);
    If Int = 0 then
      IntToLongTime := ''
    else
      IntToLongTime := concat(HourString, ':', MinuteString, ':', SecondString);
  end;

  function IntToShortTime;
    var
      HourString, MinuteString : str255;

  begin
    HourString := trim(stringof(trunc(Int / 3600) : 6));
    MinuteString := trim(stringof(trunc(Int / 60 - trunc(Int / 3600) * 60) : 6));
    If length(MinuteString) = 1 then
      MinuteString := concat('0', MinuteString);
    If Int = 0 then
      IntToShortTime := ''
    else
      IntToShortTime := concat(HourString, ':', MinuteString);
  end;

  function Trim;
    var
      i : integer;
      EditString : str255;

  begin
    EditString := copy(DataString, 1, length(DataString));
    If length(EditString) > 0 then
      begin
        i := 1;
        while (i < length(EditString)) and (EditString[i] = chr(space)) do
          i := i + 1;
        If i > 1 then
          delete(EditString, 1, i - 1);
        If EditString = ' ' then
          EditString := '';
      end;
```

```
  if length(EditString) > 0 then
    begin
      i := length(EditString);
      while (i > 1) and (EditString[i] = chr(space)) do
        i := i - 1;
      if i < length(EditString) then
        delete(EditString, i + 1, length(EditString) - i);
    end;
  if length(EditString) > 13 then
    LengthErr;
  Trim := EditString;
end;

function TextLength;
  var
    TextHndl : charsHandle;
    TextPtr : ptr;
    WordPtr : ^integer;
    i, off : integer;
    Byte0, Byte1 : integer;

begin
  TextHndl := TEGetText(TEH);
  TextPtr := pointer(TextHndl^);
  WordPtr := pointer(TextPtr);
  i := 2;
  off := 1;
  repeat
    i := i + 2;
    Byte0 := trunc(WordPtr^ / 256);
    Byte1 := WordPtr^ - trunc(WordPtr^ / 256) * 256;
    WordPtr := pointer(ord(WordPtr) + 2);
  until (Byte0 > 64) or (Byte1 > 64) or (i > 10000);
  if Byte0 > 64 then
    off := 0;
  TextLength := i + off;
end;

function KPH;

begin
  KPH := Vel * 3.6;
end;

function MPS;

begin
  MPS := Vel / 3.6;
end;

function FPS;

begin
```

```
    FPS := Vel * 3.25 / 3.6;
  end;


  function sign;

  begin
    if n > 0 then
      sign := 1
    else if n < 0 then
      sign := -1
    else
      sign := 0;
  end;

end.
```

```
unit Planner_screen;

interface

  uses
    XTTypeDefs, Extender1, Planner_init, Planner_misc;

  procedure BlankScreen;
  procedure StrategyScreen;
  procedure UpdatePilot;
  procedure Instructions;

implementation

  procedure BlankScreen;
    var
      i : integer;

  begin
    if Screen <> empty then
      begin
        HideWindow(Hash.WPtr);
        DisposDialog(PilotDP);
        HideWindow(AltLabel.WPtr);
        HideWindow(SpdLabel.WPtr);
        HideWindow(ChgLabel.WPtr);
        HideWindow(Altitude.WPtr);
        HideWindow(Speed.WPtr);
        HideWindow(Charge.WPtr);
        Screen := empty;
        EnableAllItems(AppleMenu, false);
        EnableAllItems(FileMenu, false);
        EnableAllItems(StrategyMenu, false);
        EnableAllItems(ParamMenu, false);
        EnableAllItems(ScreenMenu, false);
        MarkMenuItem(ScreenMenu, 1, chr(blank));
      end;
  end;

  procedure StrategyScreen;
    var
      i : integer;

  begin
    if Screen <> strategy then
      begin
        SetCursor(watch);
        PilotDP := GetNewDialog(PilotID, nil, pointer(-1));
        Error := SetEx(PilotDP, EXCPTactivate, true);
        UpdatePilot;
        ShowWindow(PilotDP);

        Error := SetEx(Hash.WPtr, EXCPTactivate, false);
```

```
      Error := SetEx(Charge.WPtr, EXCPTactivate, false);
      SelectWindow(Charge.WPtr);
      SelectWindow(Hash.WPtr);
      SelectWindow(Altitude.WPtr);
      Error := SetEx(Charge.WPtr, EXCPTactivate, true);
      Error := SetEx(Hash.WPtr, EXCPTactivate, true);

      ShowWindow(Altitude.WPtr);
      ShowWindow(Speed.WPtr);
      ShowWindow(Charge.WPtr);
      ShowWindow(Hash.WPtr);
      ShowWindow(AltLabel.WPtr);
      ShowWindow(SpdLabel.WPtr);
      ShowWindow(ChgLabel.WPtr);

      Screen := strategy;
      EnableAllItems(AppleMenu, true);
      EnableAllItems(FileMenu, true);
      EnableAllItems(StrategyMenu, true);
      EnableAllItems(ParamMenu, true);
      EnableAllItems(ScreenMenu, true);
      MarkMenuItem(ScreenMenu, 1, chr(checkmark));
      SetCursor(arrow);
    end;
end;

procedure UpdatePilot;
  var
    Item, ItemType : integer;
    ItemHndl : Handle;
    ItemRect : Rect;

begin
  with Pilot do
    begin
      GetDItem(PilotDP, 9, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Date : 2));
      GetDItem(PilotDP, 10, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, IntToLongTime(Time));
      GetDItem(PilotDP, 11, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, Map);
      GetDItem(PilotDP, 12, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Odometer : 11));
      GetDItem(PilotDP, 13, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Altitude : 8));
      GetDItem(PilotDP, 14, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Speed : 6));
      GetDItem(PilotDP, 15, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(SpeedLimit : 6));
      GetDItem(PilotDP, 16, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Charge : 13));
    end;
end;
```

```
  procedure Instructions;

  begin
    SetCursor(watch);
    SetCursor(arrow);
  end;

end.
```

```
unit Planner_strategy;

interface

  uses
    XTTypeDefs, Extender1, Planner_init, Planner_err, Planner_misc, Planner_screen;

  procedure Recalculate;
  procedure CalcRace (WhichDay : integer);
  procedure CalcLap (WhichDay : integer);
  function DoLaps (WhichDay : integer) : boolean;
  function MorningCharge (PtIndex : longint;
                WhichDay : integer) : extended;
  procedure RepeatLap (var PtIndex : longint;
                WhichDay : integer);
  procedure EraseLap (var PtIndex : longint;
                WhichDay : integer);
  function CalcInterval (Date : integer;
                Start, Finish : longint;
                dx : longint;
                IdealVel : extended) : extended;
  function ReCharge (T1, T2 : longint;
                ETotal : extended) : extended;
  function CalcSpeed (HEC, theta : extended) : extended;
  function Efficiency (V, theta : extended) : extended;
  function Power (V, theta : extended) : extended;
  function GetMap (MapName : str255) : map;
  procedure ReadProfile (var PtIndex : longint;
                Profile : text);
  procedure ReDraw;
  procedure LowerRedraw;
  procedure CalcPilot;
  procedure ExpandX;
  procedure ContractX;
  procedure ExpandY;
  procedure ContractY;

implementation

  procedure Recalculate;
    var
      i : integer;
      j : longint;
      PtIndex : longint;
      RMap, LMap : map;
      NotDone : boolean;
      dRA, dLA : integer;
    label
      1, 2;

  begin
    SetCursor(watch);
    for i := 1 to 8 do
```

```
    begin
      RaceStart[i] := 0;
      RaceLength[i] := 0;
      RaceAvg[i] := 0;
      RaceEff[i] := 0;
      LapStart[i] := 0;
      LapLength[i] := 0;
      LapAvg[i] := 0;
      LapEff[i] := 0;
      NLaps[i] := 0;
    end;
  PtIndex := 0;
  i := 1;
  if Itinerary[1].Date > 0 then
    NotDone := true
  else
    NotDone := false;
  while NotDone and not (Abort) do
    with Itinerary[i] do
      begin
        if i >= CalibPt.Date then
          begin
            SkipLaps := false;
            DoAllLaps := false;
            dRA := 32;
            RaceAvg[i] := MaxRA;
            LapAvg[i] := MinLA;
            RaceStart[i] := PtIndex;
            RMap := GetMap(RaceMap);
            ReadProfile(PtIndex, RMap.Profile);
            RaceLength[i] := PtIndex - RaceStart[i];
1 :
            for j := RaceStart[i] to RaceStart[i] + RaceLength[i] - 1 do
              begin
                Points[j].Time := 0;
                Points[j].Speed := 0;
                Points[j].Charge := 0;
              end;
            Points[RaceStart[i]].Time := RaceBegins;
            NLaps[i] := 0;

            CalcRace(i);
            If DoLaps(i) and not SkipLaps and not Abort then
              begin
                dLA := 10;
                LapStart[i] := PtIndex;
                LMap := GetMap(LapMap);
                ReadProfile(PtIndex, LMap.Profile);
                LapLength[i] := PtIndex - LapStart[i];
2 :
                for j := LapStart[i] to LapStart[i] + LapLength[i] - 1 do
                  begin
                    Points[j].Time := 0;
```

```
            Points[j].Speed := 0;
            Points[j].Charge := 0;
          end;
        if Points[LapStart[i] - 1].Time < LapBegins then
          Points[LapStart[i]].Time := LapBegins
        else
          Points[LapStart[i]].Time := Points[LapStart[i] - 1].Time;
        CalcLap(i);
        while (Points[PtIndex - 1].Time <= Itinerary[i].LapEnds) and ((Points[PtIndex - 1].Charge
    >= EndCharge[i]) or DoAllLaps) and not Abort do
          begin
            RepeatLap(PtIndex, i);
          end;
        if (dRA = 32) and (dLA > 0) and not Abort then
          begin
            if (Points[PtIndex - 1].Time > Itinerary[i].LapEnds) then
              begin
                LapAvg[i] := LapAvg[i] + dLA;
                dLA := trunc(dLA / 2);
                PtIndex := LapStart[i] + LapLength[i];
                goto 2;
              end
            else if LapAvg[i] > MinLA then
              begin
                LapAvg[i] := LapAvg[i] - dLA;
                dLA := trunc(dLA / 2);
                PtIndex := LapStart[i] + LapLength[i];
                goto 2;
              end;
          end;
        EraseLap(PtIndex, i);
        if (LapBegins = RaceBegins) and (abs(Points[PtIndex - 1].Charge - EndCharge[i]) > MaxBC /
    50) and (LapAvg[i] > MaxRA) and (dRA > 0) and not Abort then
          begin
            RaceAvg[i] := RaceAvg[i] + dRA * sign(Points[PtIndex - 1].Charge - EndCharge[i]);
            LapAvg[i] := RaceAvg[i];
            PtIndex := RaceStart[i] + RaceLength[i];
            dRA := trunc(dRA / 2);
            DoAllLaps := true;
            goto 1;
          end;
      end;
    end;
    i := i + 1;
    if i > 8 then
      NotDone := false
    else if Itinerary[i].Date = 0 then
      NotDone := false;
  end;
NDays := i - 1;

NPts := PtIndex - 1;
while 241 + NPts * XScale >= 32768 do
```

```
    XScale := trunc(XScale / 2);
  while 4 + 72 * YScale >= 32768 do
    YScale := trunc(YScale / 2);
  ReDraw;
  SetCursor(arrow);
end;

procedure CalcRace;
  var
    RStart, REnd : longint;
    dx : longint;
    WhichMap : map;
    CalibIndex : longint;
    Redo : boolean;

begin
  WhichMap := GetMap(Itinerary[WhichDay].RaceMap);
  dx := WhichMap.DeltaX;
  if WhichDay = CalibPt.Date then
    begin
      CalibIndex := trunc(CalibPt.Odometer / dx + 0.5);
      if (CalibIndex >= RaceLength[WhichDay]) or (CalibIndex < 0) or ((CalibIndex > 0) and (CalibPt.Time
       <= Itinerary[WhichDay].RaceBegins)) then
        OdomErr
      else if (CalibIndex = 0) and (CalibPt.Time < Itinerary[WhichDay].RaceBegins) then
        begin
          RStart := RaceStart[WhichDay];
          Points[RStart].Time := Itinerary[WhichDay].RaceBegins;
          Points[RStart].Charge := CalibPt.Charge + ReCharge(CalibPt.Time,
       Itinerary[WhichDay].RaceBegins, Itinerary[WhichDay].WattsReceived);
        end
      else
        begin
          RStart := RaceStart[WhichDay] + CalibIndex;
          Points[RStart].Time := CalibPt.Time;
          Points[RStart].Charge := CalibPt.Charge;
        end
    end
  else
    RStart := RaceStart[WhichDay];
  REnd := RaceStart[WhichDay] + RaceLength[WhichDay] - 1;
  repeat
    if Points[RStart].Charge = 0 then
      Points[RStart].Charge := Points[RStart - 1].Charge + ReCharge(Points[RStart - 1].Time, dusk,
       Itinerary[WhichDay - 1].WattsReceived) + ReCharge(dawn, Points[RStart].Time,
       Itinerary[WhichDay].WattsReceived);
    if Points[RStart].Charge > MaxBC then
      Points[RStart].Charge := MaxBC;
    Redo := false;
    RaceEff[WhichDay] := CalcInterval(WhichDay, RStart, REnd, dx, RaceAvg[WhichDay]);
{RaceAvg[WhichDay] := KPH((REnd - RStart) * dx / (Points[REnd].Time - Points[RStart].Time));}
    if (Points[REnd].Charge < EndCharge[WhichDay]) and (RaceAvg[WhichDay] <= MaxRA) then
      begin
```

```
        Redo := true;
        SkipLaps := true;
      end;
    if RaceAvg[WhichDay] < MinRA then
      begin
        Redo := false;
        SkipLaps := true;
        LowAvgErr;
      end;
    RaceAvg[WhichDay] := RaceAvg[WhichDay] * 0.9;
  until (not Redo) or Abort;
{RaceAvg[WhichDay] := RaceAvg[WhichDay] / 0.9;}
  RaceAvg[WhichDay] := KPH((REnd - RStart) * dx / (Points[REnd].Time - Points[RStart].Time));
end;

procedure CalcLap;
  var
    LStart, LEnd : longint;
    dx : longint;
    WhichMap : map;

begin
  WhichMap := GetMap(Itinerary[WhichDay].LapMap);
  dx := WhichMap.DeltaX;
  LStart := LapStart[WhichDay];
  LEnd := LapStart[WhichDay] + LapLength[WhichDay] - 1;
  Points[LStart].Charge := Points[LStart - 1].Charge + ReCharge(Points[LStart - 1].Time,
      Points[LStart].Time, Itinerary[WhichDay].WattsReceived);
  if Points[LStart].Charge > MaxBC then
    Points[LStart].Charge := MaxBC;
  LapEff[WhichDay] := CalcInterval(WhichDay, LStart, LEnd, dx, LapAvg[WhichDay]);
  LapAvg[WhichDay] := KPH((LEnd - LStart) * dx / (Points[LEnd].Time - Points[LStart].Time));
  NLaps[WhichDay] := 1;
  NPts := NPts + LapLength[WhichDay];
end;

function DoLaps;

begin
  DoLaps := (Itinerary[WhichDay].LapMap > '');
end;

function MorningCharge;

begin
  MorningCharge := Points[PtIndex - 1].Charge + ReCharge(Points[PtIndex - 1].Time, dusk,
      Itinerary[WhichDay - 1].WattsReceived) + ReCharge(dawn, Itinerary[WhichDay].RaceBegins,
      Itinerary[WhichDay].WattsReceived);
end;

procedure RepeatLap;

  var
```

```
    theta, dx : extended;
    WhichMap : map;
    LapTime, i : longint;
    Charge : extended;

  begin
    WhichMap := GetMap(Itinerary[WhichDay].LapMap);
    dx := WhichMap.DeltaX;
    LapTime := Points[LapStart[WhichDay] + LapLength[WhichDay] - 1].Time -
        Points[LapStart[WhichDay]].Time;
    Charge := Points[PtIndex - 1].Charge;
    Points[PtIndex].Charge := Charge;
    for i := 0 to LapLength[WhichDay] - 1 do
      begin
        Points[PtIndex + i].Time := Points[LapStart[WhichDay] + i].Time + LapTime * NLaps[WhichDay];
        Points[PtIndex + i].Altitude := Points[LapStart[WhichDay] + i].Altitude;
        Points[PtIndex + i].Speed := Points[LapStart[WhichDay] + i].Speed;
        Points[PtIndex + i].SpeedLimit := Points[LapStart[WhichDay] + i].SpeedLimit;
        If i > 0 then
          begin
            theta := arctan((Points[PtIndex + i].Altitude - Points[PtIndex + i - 1].Altitude) / dx);
            Charge := Charge + (Power(Points[PtIndex + i - 1].Speed, theta) / 1000) * (dx / 1000) /
         Points[PtIndex + i - 1].Speed + ReCharge(Points[PtIndex + i - 1].Time, Points[PtIndex + i].Time,
          Itinerary[WhichDay].WattsReceived);
            If Charge > MaxBC then
              Charge := MaxBC;
{if Charge < MinBC then Abort := true;}
            Points[PtIndex + i].Charge := Charge;
          end;
      end;
    PtIndex := PtIndex + LapLength[WhichDay];
    NLaps[WhichDay] := NLaps[WhichDay] + 1;
    NPts := NPts + LapLength[WhichDay];
  end;

  procedure EraseLap;
    var
      i : longint;

  begin
    for i := PtIndex - LapLength[WhichDay] to PtIndex - 1 do
      with Points[i] do
        begin
          Time := 0;
          Altitude := 0;
          Speed := 0;
          SpeedLimit := 0;
          Charge := 0;
        end;
    NLaps[WhichDay] := NLaps[WhichDay] - 1;
    NPts := NPts - LapLength[WhichDay];
    PtIndex := PtIndex - LapLength[WhichDay];
  end;
```

```
function CalcInterval;
  var
    HVel, Eff : extended;
    i : longint;
    slope, theta : extended;
    Speed, Time, Charge : extended;
    AvgVel : extended;
    j : integer;

begin
  if Points[Start].Time = 0 then
    begin
      Time := Points[Start - 1].Time;
      Points[Start].Time := trunc(Time);
    end
  else
    Time := Points[Start].Time;
  if Points[Start].Charge = 0 then
    begin
      Charge := Points[Start - 1].Charge;
      Points[Start].Charge := Charge;
    end
  else
    Charge := Points[Start].Charge;
  HVel := IdealVel;
  Eff := Efficiency(HVel, 0);
  j := 0;
  repeat
    Time := Points[Start].Time;
    Charge := Points[Start].Charge;
    i := Start;
    while (i < Finish) and not Abort do
      begin
        slope := (Points[i + 1].Altitude - Points[i].Altitude) / dx;
        theta := arctan(slope);
        Speed := CalcSpeed(Eff, theta);
        If Points[i].SpeedLimit = 0 then
          SpeedLimErr
        else if Speed > Points[i].SpeedLimit then
          Speed := Points[i].SpeedLimit;
        Points[i].Speed := trunc(Speed + 0.5);
        Time := Time + dx / MPS(Speed);
        Points[i + 1].Time := trunc(Time);
        Charge := Charge + (Power(Speed, theta) / 1000) * (dx / 1000) / Speed +
       ReCharge(Points[i].Time, trunc(Time), Itinerary[Date].WattsReceived);
        If Charge > MaxBC then
          Charge := MaxBC;
{if Charge < MinBC then Abort := true;}
        Points[i + 1].Charge := Charge;
        i := i + 1;
      end;
    If not Abort then
```

```
      begin
        AvgVel := KPH((Finish - Start) * dx / (Points[Finish].Time - Points[Start].Time));
        HVel := HVel * IdealVel / AvgVel;
        Eff := Efficiency(HVel, 0);
        j := j + 1;
      end;
    until (abs((IdealVel - AvgVel) / IdealVel) < 0.02) or (j = 5) or Abort;
    Points[Finish].Speed := Points[Finish - 1].Speed;
    CalcInterval := Eff;
  end;

  function ReCharge;

  begin
    if T1 < dawn then
      T1 := dawn;
    if T2 < dawn then
      T2 := dawn;
    if T1 > dusk then
      T1 := dusk;
    if T2 > dusk then
      T2 := dusk;
    ReCharge := (ETotal / 2) * (sin(T1 * pi / hr12) - sin(T2 * pi / hr12));
  end;

  function CalcSpeed;
    var
      SEC, BestSEC : extended;
      SVel, BestSVel : integer;

  begin
    BestSEC := 1e+10;
    BestSVel := 0;
    for SVel := 20 to 100 do
      begin
        SEC := Efficiency(SVel, theta);
        if abs(HEC - SEC) < abs(HEC - BestSEC) then
          begin
            BestSEC := SEC;
            BestSVel := SVel;
          end;
      end;
    CalcSpeed := BestSVel;
  end;

  function Efficiency;
    var
      SinTheta : extended;
      X : extended;

  begin
    SinTheta := theta;{sin(theta)}
    V := FPS(V);
```

```
  with WattFunc do
    begin
      X := (A[3] + A[4] * SinTheta + A[5] * sqr(V)) / (A[6] * A[7]);
      Efficiency := sqr(V) * (2 * A[2] * V + 4 * A[5] * A[8] * X / (A[6] * A[7])); {- sqr(X) * A[8]}
    end;
end;


function Power;
  var
    SinTheta : extended;

begin
  SinTheta := theta;{sin(theta)}
  V := FPS(V);
  with WattFunc do
    begin
      Power := -((A[0] + A[1] * SinTheta + A[2] * sqr(V)) * V + sqr((A[3] + A[4] * SinTheta + A[5] *
       sqr(V)) / (A[6] * A[7])) * A[8]);
    end;
end;

function GetMap;
  var
    i : integer;

begin
  i := 0;
  repeat
    i := i + 1;
  until (i = 16) or (MapName = MapArray[i].Name);
  if MapName <> MapArray[i].Name then
    UndMapErr;
  GetMap := MapArray[i];
end;

procedure ReadProfile;
  var
    TextHndl : charsHandle;
    TextPtr : ptr;
    WordPtr : ^integer;
    Word : longint;
    i : integer;
    DataString : str255;
    ReadDone, GetSpLim, Overshot : boolean;
    SpLim : longint;
    DataLong : longint;

begin
  SpLim := 0;
  ReadDone := false;
  GetSpLim := false;
  Overshot := false;
```

```
  TextHndl := TEGetText(Profile.WD.TEH);
  TextPtr := pointer(TextHndl^);
  WordPtr := pointer(TextPtr);
  repeat
    If Overshot then
      WordPtr := pointer(ord(WordPtr) - 2);
    Overshot := false;
    i := -1;
    DataString := '                    ';
    repeat
      i := i + 2;
      Word := WordPtr^;
      If Word < 0 then
        Word := Word + 65536;
      DataString[i] := chr(trunc(Word / 256));
      If (i = 1) and ((DataString[1] = chr(return)) or (DataString[1] = chr(semi))) then
        DataString[1] := chr(space);
      DataString[i + 1] := chr(Word - trunc(Word / 256) * 256);
      WordPtr := pointer(ord(WordPtr) + 2);
    until ((DataString[i] = chr(return)) or (DataString[i] = chr(semi)) or (DataString[i + 1] =
      chr(return)) or (DataString[i + 1] = chr(semi)) or (i > 15));
    If i > 15 then
      LengthErr;
    If (DataString[i] = chr(return)) and GetSpLim then
      begin
        GetSpLim := false;
        DataString[i] := chr(space);
        DataString[i + 1] := chr(space);
        DataString := Trim(DataString);
        ReadIntString(DataString, SpLim);
        If abs(SpLim) >= 32768 then
          begin
            SpLim := 0;
            IntRangeErr;
          end;
        Points[PtIndex - 1].SpeedLimit := SpLim;
        Overshot := true;
      end
    else If (DataString[i + 1] = chr(return)) and GetSpLim then
      begin
        GetSpLim := false;
        DataString[i + 1] := chr(space);
        DataString := Trim(DataString);
        ReadIntString(DataString, SpLim);
        If abs(SpLim) >= 32768 then
          begin
            SpLim := 0;
            IntRangeErr;
          end;
        Points[PtIndex - 1].SpeedLimit := SpLim;
      end
    else
      begin
```

```
      if (DataString[i] = chr(semi)) or (DataString[i + 1] = chr(semi)) then
        GetSpLim := true;
      if (DataString[i] = chr(return)) or (DataString[i] = chr(semi)) then
        begin
          DataString[i] := chr(space);
          DataString[i + 1] := chr(space);
          DataString := Trim(DataString);
          if DataString = 'end' then
            ReadDone := true
          else
            begin
              ReadIntString(DataString, DataLong);
              if abs(DataLong) >= 32768 then
                begin
                  Points[PtIndex].Altitude := 0;
                  IntRangeErr;
                end
              else
                Points[PtIndex].Altitude := DataLong;
              Points[PtIndex].SpeedLimit := SpLim;
              PtIndex := PtIndex + 1;
            end;
          Overshot := true;
        end
      else if (DataString[i + 1] = chr(return)) or (DataString[i + 1] = chr(semi)) then
        begin
          DataString[i + 1] := chr(space);
          DataString := Trim(DataString);
          if DataString = 'end' then
            ReadDone := true
          else
            begin
              ReadIntString(DataString, DataLong);
              if abs(DataLong) >= 32768 then
                begin
                  Points[PtIndex].Altitude := 0;
                  IntRangeErr;
                end
              else
                Points[PtIndex].Altitude := DataLong;
              Points[PtIndex].SpeedLimit := SpLim;
              PtIndex := PtIndex + 1;
            end;
        end;
    end;
  until ReadDone or Abort;
end;

procedure ReDraw;
  var
    i, j : longint;
    AltPort : GrafPtr;
```

```
begin
  with Altitude do
    begin
      SetRect(PRect, 0, 0, 482 + NPts * XScale, 7 + 72 * YScale);
      New(AltPort);
      OpenPort(AltPort);
      AltPort^.portbits.bounds := PRect;
      SetPort(AltPort);
      PortSize(PRect.right, PRect.bottom);
      ClipRect(PRect);

      SetRect(PRect, 0, 0, NPts * XScale, 72 * (YScale - 1) + 1);
      PicHndl := OpenPicture(PRect);
      MoveTo(241, 4);
      LineTo(241 + NPts * XScale, 4);
      LineTo(241 + NPts * XScale, 4 + 72 * YScale);
      LineTo(241, 4 + 72 * YScale);
      LineTo(241, 4);
      for i := 1 to NDays do
        begin
          DrawLine(241 + RaceStart[i] * XScale, 4, 241 + RaceStart[i] * XScale, 4 + 72 * YScale);
          if DoLaps(i) then
            for j := 1 to NLaps[i] do
              DrawLine(241 + (LapStart[i] + (j - 1) * LapLength[i]) * XScale, 4, 241 + (LapStart[i] + (j -
     1) * LapLength[i]) * XScale, 4 + 72 * YScale);
        end;
      MoveTo(241, trunc(4 + ((7200 - Points[0].Altitude) / 100) * YScale));
      for i := 1 to NPts do
        LineTo(241 + i * XScale, trunc(4 + ((7200 - Points[i].Altitude) / 100) * YScale));
      ClosePicture;
      SetPort(WPtr);
      SetWPic(WPtr, PicHndl);
      ClosePort(AltPort);

      SetCtlMin(HScroll, 0);
      SetCtlMax(HScroll, NPts * XScale);
      SetCtlMin(VScroll, 0);
      SetCtlMax(VScroll, 72 * (YScale - 1));
    end;

  ScrollValue := 0;
  PointValue := 0;
  LowerRedraw;
  CalcPilot;
  if Screen = strategy then
    UpdatePilot;
end;

procedure LowerRedraw;
  var
    i, j : longint;
    SpdPort, ChgPort : GrafPtr;
    minl, maxl : longint;
```

```
begin
  HideWindow(Speed.WPtr);
  HideWindow(Charge.WPtr);
  with Speed do
    begin
      SetRect(PRect, 0, 0, 482 + NPts * XScale, 80);
      New(SpdPort);
      OpenPort(SpdPort);
      SpdPort^.portbits.bounds := PRect;
      SetPort(SpdPort);
      PortSize(PRect.right, PRect.bottom);
      ClipRect(PRect);

      SetRect(PRect, 0, 0, NPts * XScale, 1);
      PicHndl := OpenPicture(PRect);
      MoveTo(241 - ScrollValue, 4);
      LineTo(241 + NPts * XScale - ScrollValue, 4);
      LineTo(241 + NPts * XScale - ScrollValue, 76);
      LineTo(241 - ScrollValue, 76);
      LineTo(241 - ScrollValue, 4);
      for i := 1 to NDays do
        begin
          DrawLine(241 + RaceStart[i] * XScale - ScrollValue, 4, 241 + RaceStart[i] * XScale -
       ScrollValue, 76);
          If DoLaps(i) then
            DrawLine(241 + LapStart[i] * XScale - ScrollValue, 4, 241 + LapStart[i] * XScale -
       ScrollValue, 76);
        end;
      MoveTo(241 - ScrollValue, trunc(76 - Points[0].Speed));
      minI := trunc(PointValue - 250 / XScale);
      If minI < 0 then
        minI := 0;
      maxI := trunc(PointValue + 250 / XScale + 1);
      If maxI > NPts - 1 then
        maxI := NPts - 1;
      for i := minI to maxI do
        begin
          LineTo(241 + i * XScale - ScrollValue, trunc(76 - Points[i].Speed * 0.7));
          LineTo(241 + (i + 1) * XScale - ScrollValue, trunc(76 - Points[i].Speed * 0.7));
        end;
      ClosePicture;

      KillWindow(WPtr);
      SetRect(WRect, 15, 179, 512, 274);
      WPtr := CreateWindow(WR, WRect, 'Speed', 2, false, false, false, true, true);
      GetWData(WPtr, WD);
      WD.hScrollBar^^.contrlVis := 0;
      WD.vScrollBar^^.contrlVis := 0;
      Error := SetEx(WPtr, EXCPTactivate, true);

      SetPort(WPtr);
      SetWPic(WPtr, PicHndl);
```

```
    ClosePort(SpdPort);

    SetCtlMin(WD.hScrollBar, 0);
    SetCtlMax(WD.hScrollBar, NPts * XScale);
    SetCtlMin(WD.vScrollBar, 0);
    SetCtlMax(WD.vScrollBar, 0);
  end;

 with Charge do
  begin
    SetRect(PRect, 0, 0, 482 + NPts * XScale, 80);
    New(ChgPort);
    OpenPort(ChgPort);
    ChgPort^.portbits.bounds := PRect;
    SetPort(ChgPort);
    PortSize(PRect.right, PRect.bottom);
    ClipRect(PRect);

    SetRect(PRect, 0, 0, NPts * XScale, 1);
    PicHndl := OpenPicture(PRect);
    MoveTo(241 - ScrollValue, 4);
    LineTo(241 + NPts * XScale - ScrollValue, 4);
    LineTo(241 + NPts * XScale - ScrollValue, 76);
    LineTo(241 - ScrollValue, 76);
    LineTo(241 - ScrollValue, 4);
    for i := 1 to NDays do
      begin
        DrawLine(241 + RaceStart[i] * XScale - ScrollValue, 4, 241 + RaceStart[i] * XScale -
   ScrollValue, 76);
        If DoLaps(i) then
          DrawLine(241 + LapStart[i] * XScale - ScrollValue, 4, 241 + LapStart[i] * XScale -
   ScrollValue, 76);
      end;
    MoveTo(241 - ScrollValue, trunc(4 + ((MaxBC - Points[0].Charge) / (MaxBC - MinBC)) * 72));
    minI := trunc(PointValue - 250 / XScale);
    If minI < 1 then
      minI := 1;
    maxI := trunc(PointValue + 250 / XScale + 1);
    If maxI > NPts then
      maxI := NPts;
    for i := minI to maxI do
      LineTo(241 + i * XScale - ScrollValue, trunc(4 + ((MaxBC - Points[i].Charge) / (MaxBC - MinBC))
   * 72));
    ClosePicture;

    KillWindow(WPtr);
    SetRect(WRect, 15, 261, 512, 357);
    WPtr := CreateWindow(WR, WRect, 'Charge', 2, false, false, false, true, true);
    GetWData(WPtr, WD);
    WD.hScrollBar^^.contrlVis := 0;
    WD.vScrollBar^^.contrlVis := 0;
    Error := SetEx(WPtr, EXCPTactivate, true);
```

```
      SetPort(WPtr);
      SetWPic(WPtr, PicHndl);
      ClosePort(ChgPort);

      SetCtlMin(WD.hScrollBar, 0);
      SetCtlMax(WD.hScrollBar, NPts * XScale);
      SetCtlMin(WD.vScrollBar, 0);
      SetCtlMax(WD.vScrollBar, 0);
    end;
  Error := SetEx(Hash.WPtr, EXCPTactivate, false);
  Error := SetEx(Charge.WPtr, EXCPTactivate, false);
  Error := SetEx(AltLabel.WPtr, EXCPTactivate, false);
  SelectWindow(Charge.WPtr);
  SelectWindow(Hash.WPtr);
  SelectWindow(AltLabel.WPtr);
  SelectWindow(Altitude.WPtr);
  Error := SetEx(AltLabel.WPtr, EXCPTactivate, true);
  Error := SetEx(Charge.WPtr, EXCPTactivate, true);
  Error := SetEx(Hash.WPtr, EXCPTactivate, true);
  ShowWindow(Speed.WPtr);
  ShowWindow(Charge.WPtr);
end;

procedure CalcPilot;
  var
    i : integer;
    RMap, LMap : map;

begin
  i := 1;
  while (RaceStart[i] <= PointValue) and (i <= NDays) do
    i := i + 1;
  i := i - 1;
  with Itinerary[i], Points[PointValue] do
    begin
      Pilot.Date := Date;
      Pilot.Time := Time;
      if (PointValue - RaceStart[i]) < RaceLength[i] then
        begin
          Pilot.Map := RaceMap;
          RMap := GetMap(RaceMap);
          Pilot.Odometer := (PointValue - RaceStart[i]) * RMap.DeltaX;
        end
      else
        begin
          Pilot.Map := LapMap;
          LMap := GetMap(LapMap);
          Pilot.Odometer := ((PointValue - LapStart[i]) mod LapLength[i]) * LMap.DeltaX;
        end;
      Pilot.Altitude := Altitude;
      Pilot.Speed := Speed;
      Pilot.SpeedLimit := SpeedLimit;
      Pilot.Charge := Charge;
```

```
    end;
  end;

  procedure ExpandX;

  begin
    if 241 + NPts * XScale * 2 < 32768 then
      begin
        SetCursor(watch);
        XScale := XScale * 2;
        ReDraw;
        SetCursor(arrow);
      end;
  end;

  procedure ContractX;

  begin
    if XScale > 1 then
      begin
        SetCursor(watch);
        XScale := trunc(XScale / 2);
        ReDraw;
        SetCursor(arrow);
      end;
  end;

  procedure ExpandY;

  begin
    if 4 + 72 * YScale * 2 < 32768 then
      begin
        SetCursor(watch);
        YScale := YScale * 2;
        ReDraw;
        SetCursor(arrow);
      end;
  end;

  procedure ContractY;

  begin
    if YScale > 1 then
      begin
        SetCursor(watch);
        YScale := trunc(YScale / 2);
        ReDraw;
        SetCursor(arrow);
      end;
  end;

end.
```

```
unit Planner_file;

interface

  uses
    XTTypeDefs, Extender1, Planner_init, Planner_err, Planner_misc, Planner_screen, Planner_strategy;

  procedure OpenFile;
  procedure ReadData;
  procedure ReadInt (var Int : longint);
  procedure ReadExt (var Ext : extended);
  procedure ReadDataString (var DataString : str255);
  procedure SaveFile;
  procedure SaveFileAs;
  procedure WriteData;
  procedure WriteInt (Int : longint);
  procedure WriteExt (Ext : extended);
  procedure WriteDataString (DataString : str255);
  procedure Quit;
  procedure SaveAlert;

implementation

  procedure OpenFile;

  begin
    SetCursor(watch);
    TESelectAll(Data.WD.TEH);
    TEDelete(Data.WD.TEH);
    SetCursor(arrow);
    TERecall(Data.WD.TEH, Reply);
    if Reply.good then
      begin
        SetCursor(watch);
        TESetScrollRange(Data.WPtr);
        ReadData;
        if not Abort then
          begin
            FilePresent := true;
            Recalculate;
            StrategyScreen;
            UpdatePilot;
          end;
        SetCursor(arrow);
      end;
  end;

  procedure ReadData;
    var
      i : integer;

  begin
    ReadInt(MinRA);
```

```
    ReadInt(MaxRA);
    ReadInt(MinLA);
    ReadExt(MinBC);
    ReadExt(MaxBC);

    ReadInt(CalibPt.Date);
    ReadInt(CalibPt.Time);
    ReadInt(CalibPt.Odometer);
    ReadExt(CalibPt.Charge);

    for i := 0 to 8 do
      ReadExt(WattFunc.A[i]);
    for i := 0 to 8 do
      ReadExt(WattFunc.B[i]);

    for i := 1 to 16 do
      with MapArray[i] do
        begin
          ReadDataString(Name);
          ReadInt(DeltaX);
          ReadInt(Length);
          TESelectAll(Profile.WD.TEH);
          TEDelete(Profile.WD.TEH);
          TESetSelect(32, 32 + Length, Data.WD.TEH);
          TECut(Data.WD.TEH);
          TEPaste(Profile.WD.TEH);
          TESetScrollRange(Profile.WPtr);
        end;

    for i := 1 to 8 do
      with Itinerary[i] do
        begin
          ReadInt(Date);
          ReadInt(RaceBegins);
          ReadInt(LapBegins);
          ReadInt(LapEnds);
          ReadExt(WattsReceived);
          ReadDataString(RaceMap);
          ReadDataString(LapMap);
          ReadInt(Deduction);
        end;

    for i := 1 to 8 do
      ReadExt(EndCharge[i]);

  end;

  procedure ReadInt;
    var
      DataString : str255;

  begin
    ReadDataString(DataString);
```

```
  ReadIntString(DataString, Int);
end;

procedure ReadExt;
  var
    DataString : str255;

begin
  ReadDataString(DataString);
  ReadExtString(DataString, Ext);
end;

procedure ReadDataString;
  var
    TextHndl : charsHandle;
    TextPtr : ptr;
    WordPtr : ^integer;
    Word : longint;
    i : integer;

begin
  TextHndl := TEGetText(Data.WD.TEH);
  TextPtr := pointer(TextHndl^);
  WordPtr := pointer(TextPtr);
  WordPtr := pointer(ord(WordPtr) + 32);
  i := -1;
  DataString := '                    ';
  repeat
    i := i + 2;
    Word := WordPtr^;
    if Word < 0 then
      Word := Word + 65536;
    DataString[i] := chr(trunc(Word / 256));
    DataString[i + 1] := chr(Word - trunc(Word / 256) * 256);
    WordPtr := pointer(ord(WordPtr) + 2);
  until ((DataString[i] = chr(return)) or (DataString[i + 1] = chr(return)) or (i > 15));
  if (i > 15) then
    LengthErr;
  if (DataString[i] = chr(return)) then
    begin
      DataString[i] := chr(space);
      DataString[i + 1] := chr(space);
      TESetSelect(32, i + 32, Data.WD.TEH);
      TEDelete(Data.WD.TEH);
    end
  else
    begin
      DataString[i + 1] := chr(space);
      TESetSelect(32, i + 33, Data.WD.TEH);
      TEDelete(Data.WD.TEH);
    end;
  DataString := Trim(DataString);
end;
```

```
procedure SaveFile;

begin
  SetCursor(watch);
  WriteData;
  Reply.good := true;
  SetCursor(arrow);
  TESave(Data.WD.TEH, Reply);
end;

procedure SaveFileAs;

begin
  SetCursor(watch);
  WriteData;
  Reply.good := false;
  SetCursor(arrow);
  TESave(Data.WD.TEH, Reply);
end;

procedure WriteData;
  var
    i : integer;
    Length : longint;

begin
  TESetSelect(32, 65535, Data.WD.TEH);
  TEDelete(Data.WD.TEH);

  WriteInt(MinRA);
  WriteInt(MaxRA);
  WriteInt(MinLA);
  WriteExt(MinBC);
  WriteExt(MaxBC);

  WriteInt(CalibPt.Date);
  WriteInt(CalibPt.Time);
  WriteInt(CalibPt.Odometer);
  WriteExt(CalibPt.Charge);

  for i := 0 to 8 do
    WriteExt(WattFunc.A[i]);
  for i := 0 to 8 do
    WriteExt(WattFunc.B[i]);

  for i := 1 to 16 do
    with MapArray[i] do
      begin
        WriteDataString(Name);
        WriteInt(DeltaX);
        Length := TextLength(Profile.WD.TEH);
        WriteInt(Length);
```

```
        TESetSelect(0, Length, Profile.WD.TEH);
        TECopy(Profile.WD.TEH);
        TESetSelect(65535, 65535, Data.WD.TEH);
        TEPaste(Data.WD.TEH);
      end;

  for i := 1 to 8 do
    with Itinerary[i] do
      begin
        WriteInt(Date);
        WriteInt(RaceBegins);
        WriteInt(LapBegins);
        WriteInt(LapEnds);
        WriteExt(WattsReceived);
        WriteDataString(RaceMap);
        WriteDataString(LapMap);
        WriteInt(Deduction);
      end;

  for i := 1 to 8 do
    WriteExt(EndCharge[i]);

end;

procedure WriteInt;

begin
  WriteDataString(concat(trim(StringOf(Int : 13)), chr(13)));
end;

procedure WriteExt;

begin
  WriteDataString(concat(trim(StringOf(Ext : 13)), chr(13)));
end;

procedure WriteDataString;
  var
    TextHndl : charsHandle;
    TextPtr : ptr;
    WordPtr : ^integer;
    Word : longint;
    Byte0, Byte1, i : integer;

begin
  TextHndl := TEGetText(Data.WD.TEH);
  TextPtr := pointer(TextHndl^);
  WordPtr := pointer(TextPtr);
  i := -1;
  Byte0 := 0;
  Byte1 := 0;
  repeat
    i := i + 2;
```

```
    If i < length(DataString) then
      begin
        Byte0 := ord(DataString[i]);
        Byte1 := ord(DataString[i + 1]);
      end;
    If i = length(DataString) then
      begin
        Byte0 := ord(DataString[i]);
        Byte1 := return;
      end;
    If i > length(DataString) then
      begin
        Byte0 := return;
        Byte1 := 0;
      end;
    Word := Byte0 * 256 + Byte1;
    If Word > 32767 then
      Word := Word - 65536;
    WordPtr^ := Word;
    WordPtr := pointer(ord(WordPtr) + 2);
  until ((Byte0 = return) or (Byte1 = return));
  If Byte0 = return then
    TESetSelect(0, i, Data.WD.TEH)
  else
    TESetSelect(0, i + 1, Data.WD.TEH);
  TECopy(Data.WD.TEH);
  TESetSelect(65535, 65535, Data.WD.TEH);
  TEPaste(Data.WD.TEH);
end;

procedure Quit;
  var
    i : integer;

begin
  If Screen = strategy then
    SaveAlert;
  If DoQuit then
    begin
      SetCursor(watch);
      Error := XTCloseFile(Reply);
      BlankScreen;
      SetCursor(watch);
      KillWindow(Altitude.WPtr);
      KillWindow(Speed.WPtr);
      KillWindow(Charge.WPtr);
      KillWindow(AltLabel.WPtr);
      KillWindow(SpdLabel.WPtr);
      KillWindow(ChgLabel.WPtr);
      for i := 1 to 16 do
        KillWindow(MapArray[i].Profile.WPtr);
      KillWindow(Data.WPtr);
      CloseResFile(FRefNum);
```

```
      SetCursor(arrow);
    end;
  end;

  procedure SaveAlert;
    var
      DP : DialogPtr;
      Item : integer;

  begin
    DP := GetNewDialog(SaveAlertID, nil, pointer(-1));
    ShowWindow(DP);
    repeat
      ModalDialog(nil, Item);
      CheckDItem(DP, Item);
    until (Item >= 1) and (Item <= 3);
    DisposDialog(DP);
    if Item = 1 then
      DoQuit := true;
    if Item = 2 then
      begin
        SaveFile;
        DoQuit := true;
      end;
    if Item = 3 then
      DoQuit := false;
  end;

end.
```

```
unit Planner_param;

interface

  uses
    XTTypeDefs, Extender1, Planner_init, Planner_misc, Planner_screen, Planner_file;

  procedure SelectLimits;
  procedure SelectCalibration;
  procedure SelectWattFunc;
  procedure SelectMaps;
  procedure EditMap (i : integer);
  procedure SelectItinerary;
  procedure SelectEndCharges;
  procedure SelectResults;

implementation

  procedure SelectLimits;
    var
      Item, ItemType : integer;
      ItemHndl : Handle;
      ItemRect : Rect;
      DataString : str255;

  begin
    SetCursor(watch);
    BlankScreen;
    LimitsDP := GetNewDialog(LimitsID, nil, pointer(-1));

    GetDItem(LimitsDP, 8, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, stringof(MinRA : 13));
    GetDItem(LimitsDP, 9, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, stringof(MaxRA : 13));
    GetDItem(LimitsDP, 10, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, stringof(MinLA : 13));
    GetDItem(LimitsDP, 11, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, stringof(MinBC : 13));
    GetDItem(LimitsDP, 12, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, stringof(MaxBC : 13));

    ShowWindow(LimitsDP);
    SetCursor(arrow);
    repeat
      repeat
        ModalDialog(nil, Item);
        CheckDItem(LimitsDP, Item);
      until ((Item = 1) or (Item = 2));
      SetCursor(watch);
      Abort := false;
      if Item = 1 then
        begin
          GetDItem(LimitsDP, 8, ItemType, ItemHndl, ItemRect);
```

```
        GetIText(ItemHndl, DataString);
        ReadIntString(DataString, MinRA);
        GetDItem(LimitsDP, 9, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, DataString);
        ReadIntString(DataString, MaxRA);
        GetDItem(LimitsDP, 10, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, DataString);
        ReadIntString(DataString, MinLA);
        GetDItem(LimitsDP, 11, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, DataString);
        ReadExtString(DataString, MinBC);
        GetDItem(LimitsDP, 12, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, DataString);
        ReadExtString(DataString, MaxBC);
      end;
    SetCursor(arrow);
  until not Abort;
  DisposDialog(LimitsDP);
  StrategyScreen;
end;

procedure SelectCalibration;
  var
    Item, ItemType : integer;
    ItemHndl : Handle;
    ItemRect : Rect;
    DataString : str255;

begin
  SetCursor(watch);
  BlankScreen;
  CalibrationDP := GetNewDialog(CalibrationID, nil, pointer(-1));
  with CalibPt do
    begin
      GetDItem(CalibrationDP, 7, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Date : 13));
      GetDItem(CalibrationDP, 8, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, concat('      ', IntToLongTime(Time)));
      GetDItem(CalibrationDP, 9, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Odometer : 13));
      GetDItem(CalibrationDP, 10, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(Charge : 13));

      ShowWindow(CalibrationDP);
      SetCursor(arrow);
      repeat
        repeat
          ModalDialog(nil, Item);
          CheckDItem(CalibrationDP, Item);
        until ((Item = 1) or (Item = 2));
        SetCursor(watch);
        Abort := false;
        if Item = 1 then
```

```
        begin
          GetDItem(CalibrationDP, 7, ItemType, ItemHndl, ItemRect);
          GetIText(ItemHndl, DataString);
          ReadIntString(DataString, Date);
          GetDItem(CalibrationDP, 8, ItemType, ItemHndl, ItemRect);
          GetIText(ItemHndl, DataString);
          Time := TimeToInt(DataString);
          GetDItem(CalibrationDP, 9, ItemType, ItemHndl, ItemRect);
          GetIText(ItemHndl, DataString);
          ReadIntString(DataString, Odometer);
          GetDItem(CalibrationDP, 10, ItemType, ItemHndl, ItemRect);
          GetIText(ItemHndl, DataString);
          ReadExtString(DataString, Charge);
        end;
      SetCursor(arrow);
    until not Abort;
  end;
 DisposDialog(CalibrationDP);
 StrategyScreen;
end;

procedure SelectWattFunc;
 var
  Item, ItemType : integer;
  ItemHndl : Handle;
  ItemRect : Rect;
  DataString : str255;
  i : integer;

begin
 SetCursor(watch);
 BlankScreen;
 WattFuncDP := GetNewDialog(WattFuncID, nil, pointer(-1));
 with WattFunc do
  begin
   for i := 0 to 8 do
    begin
     GetDItem(WattFuncDP, 12 + i, ItemType, ItemHndl, ItemRect);
     SetIText(ItemHndl, stringof(A[i] : 13));
     GetDItem(WattFuncDP, 30 + i, ItemType, ItemHndl, ItemRect);
     SetIText(ItemHndl, stringof(B[i] : 13));
    end;
   ShowWindow(WattFuncDP);
   SetCursor(arrow);
   repeat
    repeat
     ModalDialog(nil, Item);
     CheckDItem(WattFuncDP, Item);
    until ((Item = 1) or (Item = 2));
    SetCursor(watch);
    Abort := false;
    if Item = 1 then
     begin
```

```
          for i := 0 to 8 do
            begin
              GetDItem(WattFuncDP, 12 + i, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              ReadExtString(DataString, A[i]);
              GetDItem(WattFuncDP, 30 + i, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              ReadExtString(DataString, B[i]);
            end;
        end;
      SetCursor(arrow);
    until not Abort;
  end;
 DisposDialog(WattFuncDP);
 StrategyScreen;
end;

procedure SelectMaps;
 var
  Item, ItemType : integer;
  ItemHndl : Handle;
  ItemRect : Rect;
  i : integer;
  WPtr : WindowPtr;
  WR : WindowRecord;
  WRect, CRect : Rect;
  Button : array[0..16] of ControlHandle;
  Event : EventRecord;
  WhatHappened : EventStuff;

begin
 SetCursor(watch);
 EnableAllItems(FileMenu, false);
 EnableAllItems(StrategyMenu, false);
 EnableAllItems(ParamMenu, false);
 EnableAllItems(ScreenMenu, false);
 SetRect(WRect, 10, 50, 205, 297);
 WPtr := CreateWindow(WR, WRect, 'Maps', 0, false, false, false, false, false);
 SetRect(CRect, 5, 212, 70, 242);
 Button[0] := CreatePushButton(WPtr, CRect, 'OK');
 for i := 1 to 8 do
  begin
   SetRect(CRect, 5, -20 + i * 25, 85, i * 25);
   Button[i] := CreatePushButton(WPtr, CRect, MapArray[i].Name);
   SetRect(CRect, 110, -20 + i * 25, 190, i * 25);
   Button[i + 8] := CreatePushButton(WPtr, CRect, MapArray[i + 8].Name);
  end;
 BlankScreen;
 ShowWindow(WPtr);
 SetCursor(arrow);
 repeat
  repeat
   SystemTask;
```

```
      MaintainCursor(arrow, iBeam, arrow);
    until XTGetNextEvent(EveryEvent, event);
    HandleEvent(event, whatHappened);
    with WhatHappened do
      for i := 0 to 16 do
        if (WhichControl = Button[i]) and (i > 0) then
          begin
            HideWindow(WPtr);
            EditMap(i);
            SetCTitle(Button[i], MapArray[i].Name);
            ShowWindow(WPtr);
          end;
  until WhatHappened.WhichControl = Button[0];
  EnableAllItems(FileMenu, true);
  EnableAllItems(StrategyMenu, true);
  EnableAllItems(ParamMenu, true);
  EnableAllItems(ScreenMenu, true);
  KillWindow(WPtr);
  StrategyScreen;
end;

procedure EditMap;
  var
    Item, ItemType : integer;
    ItemHndl : Handle;
    ItemRect : Rect;
    DataString : str255;
    Event : EventRecord;
    WhatHappened : EventStuff;

begin
  SetCursor(watch);
  MapEditDP := GetNewDialog(MapEditID, nil, pointer(-1));
  with MapArray[i] do
    begin
      GetDItem(MapEditDP, 5, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, Name);
      GetDItem(MapEditDP, 6, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(DeltaX : 8));
      ShowWindow(MapEditDP);
      SetCursor(arrow);
      repeat
        repeat
          ModalDialog(nil, Item);
          CheckDItem(MapEditDP, Item);
          if Item = 8 then
            begin
              SelectWindow(Profile.WPtr);
              ShowWindow(Profile.WPtr);
              WindowClosed := false;
              repeat
                repeat
                  SystemTask;
```

```
            MaintainCursor(arrow, iBeam, arrow);
          until XTGetNextEvent(EveryEvent, event);
          HandleEvent(event, whatHappened);
        until WindowClosed;
        HideWindow(Profile.WPtr);
      end;
    until (Item = 1) or (Item = 2);
    if Item = 1 then
      begin
        SetCursor(watch);
        Abort := false;
        GetDItem(MapEditDP, 5, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, Name);
        Name := Trim(Name);
        GetDItem(MapEditDP, 6, ItemType, ItemHndl, ItemRect);
        GetIText(ItemHndl, DataString);
        ReadIntString(DataString, DeltaX);
        SetCursor(arrow);
      end;
  until not Abort;
 end;
 DisposDialog(MapEditDP);
end;

procedure SelectItinerary;
 var
  Item, ItemType : integer;
  ItemHndl : Handle;
  ItemRect : Rect;
  DataString : str255;
  i : integer;

begin
 SetCursor(watch);
 BlankScreen;
 ItineraryDP := GetNewDialog(ItineraryID, nil, pointer(-1));
 for i := 1 to 8 do
  with Itinerary[i] do
   begin
    GetDItem(ItineraryDP, i * 8 + 3, ItemType, ItemHndl, ItemRect);
    If Date = 0 then
     SetIText(ItemHndl, '')
    else
     SetIText(ItemHndl, stringof(Date : 2));
    GetDItem(ItineraryDP, i * 8 + 4, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, IntToShortTime(RaceBegins));
    GetDItem(ItineraryDP, i * 8 + 5, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, IntToShortTime(LapBegins));
    GetDItem(ItineraryDP, i * 8 + 6, ItemType, ItemHndl, ItemRect);
    SetIText(ItemHndl, IntToShortTime(LapEnds));
    GetDItem(ItineraryDP, i * 8 + 7, ItemType, ItemHndl, ItemRect);
    If WattsReceived = 0 then
     SetIText(ItemHndl, '')
```

```
      else
        SetIText(ItemHndl, stringof(WattsReceived : 12));
      GetDItem(ItineraryDP, i * 8 + 8, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, RaceMap);
      GetDItem(ItineraryDP, i * 8 + 9, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, LapMap);
      GetDItem(ItineraryDP, i * 8 + 10, ItemType, ItemHndl, ItemRect);
      If Deduction = 0 then
        SetIText(ItemHndl, '')
      else
        SetIText(ItemHndl, stringof(Deduction : 6));
    end;
  ShowWindow(ItineraryDP);
  SetCursor(arrow);
  repeat
    repeat
      ModalDialog(nil, Item);
      CheckDItem(ItineraryDP, Item);
    until ((Item = 1) or (Item = 2));
    SetCursor(watch);
    Abort := false;
    If Item = 1 then
      begin
        for i := 1 to 8 do
          with Itinerary[i] do
            begin
              GetDItem(ItineraryDP, i * 8 + 3, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              ReadIntString(DataString, Date);
              GetDItem(ItineraryDP, i * 8 + 4, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              RaceBegins := TimeToInt(DataString);
              GetDItem(ItineraryDP, i * 8 + 5, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              LapBegins := TimeToInt(DataString);
              GetDItem(ItineraryDP, i * 8 + 6, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              LapEnds := TimeToInt(DataString);
              GetDItem(ItineraryDP, i * 8 + 7, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              ReadExtString(DataString, WattsReceived);
              GetDItem(ItineraryDP, i * 8 + 8, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, RaceMap);
              RaceMap := Trim(RaceMap);
              GetDItem(ItineraryDP, i * 8 + 9, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, LapMap);
              LapMap := Trim(LapMap);
              GetDItem(ItineraryDP, i * 8 + 10, ItemType, ItemHndl, ItemRect);
              GetIText(ItemHndl, DataString);
              ReadIntString(DataString, Deduction);
              SetCursor(arrow);
            end;
      end;
```

```
  until not Abort;
  DisposDialog(ItineraryDP);
  StrategyScreen;
end;

procedure SelectEndCharges;
  var
    Item, ItemType : integer;
    ItemHndl : Handle;
    ItemRect : Rect;
    DataString : str255;
    i : integer;

begin
  SetCursor(watch);
  BlankScreen;
  EndChargesDP := GetNewDialog(EndChargesID, nil, pointer(-1));

  for i := 1 to 8 do
    begin
      GetDItem(EndChargesDP, 11 + i, ItemType, ItemHndl, ItemRect);
      SetIText(ItemHndl, stringof(EndCharge[i] : 13));
    end;

  ShowWindow(EndChargesDP);
  SetCursor(arrow);
  repeat
    repeat
      ModalDialog(nil, Item);
      CheckDItem(EndChargesDP, Item);
    until ((Item = 1) or (Item = 2));
    SetCursor(watch);
    Abort := false;
    if Item = 1 then
      for i := 1 to 8 do
        begin
          GetDItem(EndChargesDP, 11 + i, ItemType, ItemHndl, ItemRect);
          GetIText(ItemHndl, DataString);
          ReadExtString(DataString, EndCharge[i]);
        end;
    SetCursor(arrow);
  until not Abort;
  DisposDialog(EndChargesDP);
  StrategyScreen;
end;

procedure SelectResults;
  var
    Item, ItemType : integer;
    ItemHndl : Handle;
    ItemRect : Rect;
    i : integer;
    Event : EventRecord;
```

```
    WhatHappened : EventStuff;

  begin
    SetCursor(watch);
    EnableAllItems(FileMenu, false);
    EnableAllItems(StrategyMenu, false);
    EnableAllItems(ParamMenu, false);
    EnableAllItems(ScreenMenu, false);
    BlankScreen;
    ResultsDP := GetNewDialog(ResultsID, nil, pointer(-1));
    for i := 1 to 8 do
      begin
        GetDItem(ResultsDP, 12 + i, ItemType, ItemHndl, ItemRect);
        SetIText(ItemHndl, concat(stringof(trunc(RaceAvg[i] + 0.5) : 3), '/', Trim(stringof(trunc(RaceEff[i]
         + 0.5) : 5))));
        GetDItem(ResultsDP, 20 + i, ItemType, ItemHndl, ItemRect);
        SetIText(ItemHndl, concat(stringof(trunc(LapAvg[i] + 0.5) : 3), '/', Trim(stringof(trunc(LapEff[i] +
         0.5) : 5))));
        GetDItem(ResultsDP, 28 + i, ItemType, ItemHndl, ItemRect);
        SetIText(ItemHndl, stringof(trunc(NLaps[i] + 0.5) : 5));
      end;
    ShowWindow(ResultsDP);
    SetCursor(arrow);
    repeat
      repeat
        SystemTask;
        MaintainCursor(arrow, iBeam, arrow);
      until XTGetNextEvent(EveryEvent, event);
      HandleEvent(event, whatHappened);
    until WhatHappened.DialogItem = 1;
    EnableAllItems(FileMenu, true);
    EnableAllItems(StrategyMenu, true);
    EnableAllItems(ParamMenu, true);
    EnableAllItems(ScreenMenu, true);
    KillWindow(ResultsDP);
    StrategyScreen;
  end;

end.
```

```
program Planner;

 uses
  XTTypeDefs, Extender1, Planner_init, Planner_screen, Planner_strategy, Planner_file, Planner_param;

 var
  Event : EventRecord;
  WhatHappened : EventStuff;
  i : integer;

begin {main}
 Init;
 repeat
  repeat
   SystemTask;
   MaintainCursor(arrow, iBeam, arrow);
  until XTGetNextEvent(EveryEvent, event);
  HandleEvent(event, whatHappened);
  reply.good := false;
  Abort := false;
  with whatHappened do
   begin
    if WhichWindow = PilotDP then
     SelectWindow(Altitude.WPtr);
    if WhichControl = HScroll then
     begin
      ScrollValue := GetCtlValue(HScroll);
      PointValue := trunc(ScrollValue / XScale);
      LowerRedraw;
      CalcPilot;
      UpdatePilot;
     end;
    case MenuNum of
     11 : {apple menu}
      if Itemnum = 1 then
       Instructions;

     12 : {file menu}
      case Itemnum of
       1 :
        OpenFile;
       3 :
        SaveFile;
       4 :
        SaveFileAs;
       6 :
        Quit;
      end;

     21 : {strategy menu}
      case Itemnum of
       1 :
        begin
```

```
            Recalculate;
            UpdatePilot;
          end;
        2 :
          SelectResults;
        3 :
          ExpandX;
        4 :
          ContractX;
        5 :
          ExpandY;
        6 :
          ContractY;
      end;

    22 : {param menu}
      case Itemnum of
        1 :
          SelectLimits;
        2 :
          SelectCalibration;
        3 :
          SelectWattFunc;
        4 :
          SelectMaps;
        5 :
          SelectItinerary;
        6 :
          SelectEndCharges;
      end;

    23 : {screen menu}
      case Itemnum of
        1 :
          StrategyScreen;
      end;
    otherwise
      ;
  end;
 end;
until DoQuit;
end.
```