The missile knows where it is at all times. [1]
It knows[2] this because it knows where it isn't[3].
By subtracting where it is from where it isn't, or where it isn't from where it is (whichever is greater), it obtains a difference, or deviation[4].
The guidance subsystem uses deviations to generate corrective commands[5] to drive the missile from a position where it is to a position where it isn't, and arriving at a position where it wasn't, it now is. Consequently, the position where it is, is now the position that it wasn't, and it follows that the position that it was, is now the position that it isn't.[6]
In the event that the position that it is in is not the position that it wasn't, the system has acquired a variation[7], the variation being the difference between where the missile is, and where it wasn't. If variation is considered to be a significant[8] factor, it too may be corrected by the GEA[9] . However, the missile must also know where it was.
The missile guidance computer scenario works as follows. Because a variation has modified some of the information the missile has obtained, it is not sure just where it is. However, it is sure where it isn't, within reason, and it knows where it was. It now subtracts where it should be from where it wasn't, or vice-versa, and by differentiating this from the algebraic sum of where it shouldn't be, and where it was, it is able to obtain the deviation and its variation, which is called error.

---

[1] From this we can deduce that it knows it's location initially, at time = 0
[2] To calculate where it is, at least to a large degree of accuracy, we take all of the points where it isn't, and all of the other points are possible positions of the missile. It then selects the most plausible of these positions as it's estimated position.
[3] It knows this because it knows all of it's previous positions, and it is in constant motion, thus it is not in any of these previous positions. Also, it is capable of "seeing" (by some means of which we are unaware, but which are also unimportant) it's surroundings, and thus where it isn't.
[4] Where x y and a are the coordinates in space of the missile, and xnot ynot and znot are the positions of the point where it is not (which is arbitrary):
Deviation = absoluteValue((x-xnot)+(y-ynot)+(z-znot))
[5] The term corrective commands is vague so we will have to create said commands, probably using PID
[6] In other words, it changes the missile's position and restarts the process
[7] In other words if it doesn't move, or moves incorrectly
[8] If the variation is above a certain threshold, or if it moved more than a certain amount off course
[9] Again probably using PID

Flowchart:
1). Program starts by finding the object's position using the PositionFromWhereItsNot class[10]
2). If the object's y coordinate is less than the y value of the desired destination, then we set the coordinates of the next point as (obj_x + dx, obj_y + dy) where obj_x and obj_y are the coordinates of the object, and dx and dy are small numbers. Else if the y coordinate is too high, then set the next point as (obj_x + dx, obj_y - dy). Else the new point is (obj_x + dx, obj_y)
3). Now move the object using MoveObject class[11]
4). Now we will see how far off we are from the desired point, once again getting where we are, and subtracting it from the desired point. This is our variation. If it is larger than an arbitrary threshold, then we use the Move object class again, but we restrict the random integers to the range 25 to -25, narrowing us in even further.

Repeat...

[10] The structure of this class is basically:
1. see how far off the ground the object is (y distance to bottom of screen)
2. see how far away the target (right side of the screen) the object is (x distance to right side of screen)
3. these are the places that the rocket is not, so the point after where it is not (x_dist + some_constant, y_dist + some_constant) is where it is, because this is not where it is not
4. Return this as the position of the rocket

[11] The structure of this class is:
1. Generate two separate random integers from 100 to -100, and scale roughly down to the size of dx and dy.
2. Now move by (dx + first_rand, dy + second_rand)

This moves the object in the right general direction but semi-randomly, and it corresponds to finding the Deviation, and closing it.