Writing smaller/better code for the ISS

Many of these suggestions create small improvements in the code, but when used repeatedly, can lead to substantial code reduction, and clearer, more easily debugged code…

1. Use #define to make your code readable:

NORMAL:

```
if (state == 12) {
      do this;
      if (this is done)
            state = 38;
}
if (state == 13} etc…
```

BETTER:

```
// above all code:
#define ST_WAIT_FOR_POI    12
#define ST_TAKING_PIC      13
…etc…
#define ST_GOTO_SHADOW     38

// in the midst of your code…
if (state == ST_WAIT_FOR_POI) {
      do this;
      if (this is done)
            state = ST_GOTO_SHADOW;
}
if (state == ST_TAKING_PIC) etc…
```

2. You can use arrays of vectors, instead of duplicating code:

NORMAL:

```
float poi0[3],poi1[3],poi2[3];

game.getPOILoc(poi0,0);
game.getPOILoc[poi1,1);
game.getPOILoc(poi2,2);
```

BETTER:

```
float poi[3][3];
int i;

for (i = 0; i < 3; ++i) game.getPOILoc(poi[i],i);
```

3. You can use memcpy() to copy arrays:

memcpy(destination, source, num_of_bytes) copies memory directly

NORMAL:

```
float myState[12], velocity[3];
int i;

api.getMyZRState(myState);
for (i = 0; i < 3; ++i) velocity[i] = myState[i+3];
```

BETTER:

```
float myState[12], velocity[3];

api.getMyZRState(myState);
memcpy(velocity, myState+3, 3*sizeof(float));

//note: the multiplication “3*sizeof(float)” will be
//  done by the compiler, which will rewrite the statement as:
// memcpy(velocity, myState+3, 12);
//  and will not add work to the code itself.
```

4. Don't copy arrays at all, unless necessary. You can use array-pointer arithmetic.

NORMAL:

```
float mySpeed() {

      float myState[12], velocity[3];

      api.getMyZRState(myState);
      memcpy(velocity, myState+3, 3*sizeof(float));
      return mathVecMagnitude(velocity,3);

}
```

BETTER:

```
float mySpeed() {

      float myState[12];

      api.getMyZRState(myState);
      return mathVecMagnitude(myState+3,3);

}
```

5. Don't copy arrays. Use pointers instead. They behave just like arrays.

NORMAL:

```
void DistanceToBestPOI() {

      float poi[3][3], bestPOI[3];
      int i, whichIsBest;

      for (i = 0; i < 3; ++i) game.getPOILoc(poi[i],i);

      // decide which is the best POI… put index of the
      //  best POI into whichIsBest (an int between 0 and 2)

      memcpy(bestPOI, poi[whichIsBest], 3*sizeof(float());

      // … work with bestPOI…
```

BETTER:

```
void DistanceToBestPOI() {

      float poi[3][3], *bestPOI;
      int i, whichIsBest;

      for (i = 0; i < 3; ++i) game.getPOILoc(poi[i],i);

      // decide which is the best POI… put index of the
      //  best POI into whichIsBest (an int between 0 and 2)

      bestPOI = poi[whichIsBest];

      //  … work with bestPOI…
```

6. Put commonly used data into globals so that you don't have to ask for them in many places. Remember, we do not have a cpu time limit, we just have a cpu space limit.

The code below compiles to 4% of code size.

PROBABLY BETTER:

```
float G_myState[12];
float G_otherState[12];
float G_poi[3][3];
int G_time;
int G_nextFlare;
int G_memoryFilled;
float G_speed;  // our speed
float G_height;  // our distance from origin

void init() {

      G_Time = -1;

}

void loop() {
      int i;

      api.getMyZRState(G_myState);
      api.getOtherZRState(G_otherState);
      for (i = 0; i < 3; ++i) game.getPOILoc(G_poi[i],i);
      ++G_time;
      G_nextFlare = game.getNextFlare();
      G_memoryFilled = game.getMemoryFilled();
      G_speed = mathVecMagnitude(G_myState+3,3);
      G_height = mathVecMagnitude(G_myState,3);

      // … start the real work…
```