The *Model / View / Controller* Design Pattern

When designing software for interactive applications such as video games, desktop word processing, financial services, and thousands of others, a common way to design the software is called the *Model / View / Controller* design pattern, or *MVC*. When using this approach, the application is sub–divided into three very distinct and somewhat independent parts. Those are:

1. The *Model* class or set of functions is responsible for maintaining all of the data needed by the application. For a video game for example, the model might have information about each player in the game, their location, their collection of weapons or other belongings, and the background terrain. The key point in the design of the model is that it is completely unaware of how the information is to be displayed (or viewed) by the user, nor is it aware of how the model changes over time. Those responsibilites are assigned to the other two parts of the MVC design.

2. The *View* class or set of functions is responsible for using the information found in the model and creating a graphical image on the computer screen that represents the data in the model. Again using the video game example, the view class would render an image of each player at the appropriate location, and render that player's set of possessions. Further, the view class would render the background images with buildings, terrain, etc. Further, the view class typically would detect user actions such as mouse clicks, joystick moves, and button presses; but the responsibility for responding to those actions is delegate to the *Controller*.

3. The *Controller* class or set of functions is responsible for responding to user actions and updating the state of the model in some reasonable way. An example might be updating a given player's location in the model given his acceleration and velocity. Another might be changing the state of a player from *alive* to *dead* in response to some fatal action. Again, the key point is that the controller has no knowledge about the visual representation of the view, nor the internal representation of the objects in the model.

To illustrate this approach, study the design of the simple tic-tac-toe game given in the listings below. There are three classes defined in `ttt.h`, representing the model, view and controller. The implementations for the three classes are found in `ttt.cc` and the short main program that simply instantiates the three classes is found in `TicTacToe.cc`.

```
1  // Demonstrate the Model/View/Controller design pattern using TicTacToe
2  // George F. Riley, Georgia Tech, Fall 2011
3
4  // Define three classes for the Tic Tac Toe game
5  // 1) The Model class keeps track of the state of the board
6  // 2) The View class displays the model
7  // 3) The controller responds to actions and updates the model as needed.
8
9  #define N_SQUARE 9
10 #define N_ROW    3
11 #define N_COL    3
12 #define N_DIAG   2
13
14 class TTTController;
15 class TTTModel
16 {
17 public:
18   TTTModel();
19   void Move(int square, char player);
20   bool LegalMove(int square);   // True of square is empty and game not over
21   bool GameOver();              // True if game is over
22   void MoveRow(int r);          // Move 'o' to empty cell on row r
23   void MoveCol(int c);          // Move 'o' to empty cell on col c
24   void MoveDiag(int d);         // Move 'o' to empty cell on diag d
25   void Clear();                 // New game
26   int  GetSquare(int r, int c); // Get square number from row/col
27   int  GetCol(int s);           // Get col number from square
28   int  GetRow(int s);           // Get row number from square
29   // Helpers for controller
30   int  FindEmptyRow(int c);     // Find empty row in specified col
31   int  FindEmptyCol(int r);     // Find empty col in specified row
32   int  FindEmptyDiag(int d);    // Find empty cell in specified diag
33   int  FindEmptyCorner();       // Find an empty corner
34   int  FindEmptySide();         // Find empty side
35 public:
36   // Maintain the board in several different representations
37   // to ease the next move decision process
38   int  nMoves;             // Total number of squares occupied
39   int  xRowCount[N_ROW];   // number x moves on each row
40   int  oRowCount[N_ROW];   // Number o moves on each row
41   int  xColCount[N_COL];   // number x moves on each col
42   int  oColCount[N_COL];   // Number o moves on each col
43   int  xDiagCount[N_DIAG]; // number x moves on each diag
44   int  oDiagCount[N_DIAG]; // Number o moves on each diag
45   char board[N_SQUARE];    // State of board
46   bool draw;               // True if draw
47   bool xWins;              // True if x wins
48   bool oWins;              // True if o wins
49   int  winRow;             // Row number of winning row
50   int  winCol;             // Col number of winning col
51   int  winDiag;            // Diag number of winning diag
52 };
53
54 class TTTView
55 { // Resposible for displaying the board
56 public:
```

Program ttt.h

```
57    // Since the view uses OpenGL and responds to mouse clicks
58    // the view needs to be aware of both the model (to update the board)
59    // and the controller to determine next move.  Also needs
60    // argc and argv for opengl init
61    TTTView(int argc, char** argv, TTTModel*, TTTController*);
62  public:
63    // Static member functions and variables
64    static void reshape(int, int); // Called by GLUT
65    static void display(); // Called by GLUT
66    static void mouse(int, int, int, int); // Also called by glut
67    static TTTModel*      model;
68    static TTTController* controller;
69    static int winW;    // Window width and height
70    static int winH;
71    static int squareW; // Width of each square
72    static int squareH; // Height of each square
73  };
74
75  class TTTController
76  {
77  public:
78    // Controller needs access to the model to determine next move
79    TTTController(TTTModel*);
80    void Move(int square);   // Process player (x) move
81  public:
82    TTTModel* model;
83  private:
84    // Move helpers
85    void Response1();       // Choose 'o' move in response to first move
86    void Response2();       // Choose 'o' move in response to players second move
87    void ResponseOther();   // Choose 'o' move after later moves
88    int  BlockingMove();    // Choose 'o' to block player's win
89  };
```

Program ttt.h (continued)

```
1  // Implement the model, view, and controller for the Tic Tac Toe game
2  // George F. Riley, Georgia Tech, Fall 2011
3
4  #ifdef OSX
5  #include <GLUT/glut.h>
6  #include <OpenGL/glext.h>
7  #include <OpenGL/gl.h>
8  #include <OpenGL/glu.h>
9  #else
10 #include <GL/glut.h>
11 #include <GL/glext.h>
12 #include <GL/gl.h>
13 #include <GL/glu.h>
14 #endif
15
16 #include <iostream>
17 #include <math.h>
18 #include "ttt.h"
19
20 using namespace std;
21
22 // Model implementation
23 TTTModel::TTTModel()
24 {
25   Clear();
26 }
27
28 void TTTModel::Move(int square, char player)
29 { // returns true if legal move
30   int r = square / N_COL;
31   int c = square % N_ROW;
32   if (player == 'x')
33     { // Make the moves
34       xRowCount[r]++;
35       xColCount[c]++;
36       if (square == 0 || square == 4 || square == 8) xDiagCount[0]++;
37       if (square == 2 || square == 4 || square == 6) xDiagCount[1]++;
38       if (xRowCount[r] == N_ROW || xColCount[c] == N_COL ||
39           xDiagCount[0] == 3 || xDiagCount[1] == 3)
40         {
41           xWins = true;
42           if (xRowCount[r] == N_ROW) winRow = r;
43           if (xColCount[r] == N_COL) winCol = c;
44           if (xDiagCount[0] == 3) winDiag = 0;
45           if (xDiagCount[1] == 3) winDiag = 1;
46           cout << "X Wins!" << endl;
47         }
48     }
49   if (player == 'o')
50     { // Make the moves
51       oRowCount[r]++;
52       oColCount[c]++;
53       if (square == 0 || square == 4 || square == 8) oDiagCount[0]++;
54       if (square == 2 || square == 4 || square == 6) oDiagCount[1]++;
55       if (oRowCount[r] == N_ROW || oColCount[c] == N_COL ||
56           oDiagCount[0] == 3 || oDiagCount[1] == 3)
```

Program ttt.cc

```
57          {
58            oWins = true;
59            if (oRowCount[r] == N_ROW) winRow = r;
60            if (oColCount[r] == N_COL) winCol = c;
61            if (oDiagCount[0] == 3) winDiag = 0;
62            if (oDiagCount[1] == 3) winDiag = 1;
63            cout << "O Wins!" << endl;
64          }
65      }
66    // Update board character and count moves
67    board[square] = player;
68    nMoves++;
69    if (!xWins && !oWins && (nMoves == N_SQUARE)) draw = true;
70    if (draw) cout << "Draw!" << endl;
71  }
72
73  bool TTTModel::LegalMove(int square)
74  { // Determine if legal move
75    if (board[square] != ' ') return false;
76    if (draw || oWins || xWins) return false;
77    return true;
78  }
79
80  bool TTTModel::GameOver()
81  {
82    return draw || xWins || oWins;
83  }
84
85  void TTTModel::Clear()
86  { // Set up a new game
87    nMoves = 0;
88    for (int i = 0; i < N_SQUARE; ++i) board[i] = ' ';
89    for (int i = 0; i < N_ROW; ++i)
90      {
91        xRowCount[i] = 0;
92        oRowCount[i] = 0;
93      }
94    for (int i = 0; i < N_COL; ++i)
95      {
96        xColCount[i] = 0;
97        oColCount[i] = 0;
98      }
99    for (int i = 0; i < N_DIAG; ++i)
100     {
101       xDiagCount[i] = 0;
102       oDiagCount[i] = 0;
103     }
104   draw = false;
105   xWins = false;
106   oWins = false;
107   winRow = -1;
108   winCol = -1;
109   winDiag = -1;
110 }
111
112 int TTTModel::GetSquare(int r, int c)
```

Program ttt.cc (continued)

```
113 {
114   return r * N_COL + c;
115 }
116
117 // Model helpers
118 int TTTModel::FindEmptyRow(int c)
119 {
120   for (int r = 0; r < N_ROW; ++r)
121     {
122       int s = GetSquare(r, c);
123       if (board[s] == ' ') return s;
124     }
125   return -1; // Not found (should never happen)
126 }
127
128 int TTTModel::FindEmptyCol(int r)
129 {
130   for (int c = 0; c < N_COL; ++c)
131     {
132       int s = GetSquare(r, c);
133       if (board[s] == ' ') return s;
134     }
135   return -1; // Not found, should never happen
136 }
137
138 int TTTModel::FindEmptyDiag(int d)
139 {
140   if (d == 0)
141     { // square 0, 4, or 8
142       if (board[0] == ' ') return 0;
143       if (board[4] == ' ') return 4;
144       if (board[8] == ' ') return 8;
145     }
146   else
147     { // square 2, 4 or 6
148       if (board[2] == ' ') return 2;
149       if (board[4] == ' ') return 4;
150       if (board[6] == ' ') return 6;
151     }
152   return -1;
153 }
154
155 int TTTModel::FindEmptyCorner()
156 {
157   if (board[0] == ' ') return 0;
158   if (board[2] == ' ') return 2;
159   if (board[6] == ' ') return 6;
160   if (board[8] == ' ') return 8;
161 }
162
163 int TTTModel::FindEmptySide()
164 {
165   if (board[1] == ' ') return 1;
166   if (board[3] == ' ') return 3;
167   if (board[5] == ' ') return 5;
168   if (board[7] == ' ') return 7;
```

Program ttt.cc (continued)

```
169 }
170
171
172
173 // Implement the view
174
175 // Firt the static variables
176 TTTModel*      TTTView::model = 0;
177 TTTController* TTTView::controller = 0;
178 int            TTTView::winW = 300;
179 int            TTTView::winH = 300;
180 int            TTTView::squareW = winW / N_COL;
181 int            TTTView::squareH = winH / N_ROW;
182
183 TTTView::TTTView(int argc, char** argv, TTTModel* m, TTTController* c)
184 {
185   model = m;       // Save the model pointer
186   controller = c;  // Save the controller pointer
187   winW = 300;      // Window width and height
188   winH = 300;
189   squareW = winW / N_COL;
190   squareH = winH / N_ROW;
191
192   // Initialize OpenGL
193   glutInit(&argc, argv);
194   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
195   glutInitWindowSize(winW, winH);
196   glutInitWindowPosition(100, 100);
197   glutCreateWindow("Tic Tac Toe");
198   glClearColor(1.0, 1.0, 1.0, 0.0);   // white background
199   glutDisplayFunc(display);
200   glutMouseFunc(mouse);
201   glutReshapeFunc(reshape);
202 }
203
204 void TTTView::reshape(int w, int h)
205 {
206   winW = w;
207   winH = h;
208   squareW = winW / N_COL;
209   squareH = winH / N_ROW;
210
211   glViewport(0, 0, (GLsizei)winW, (GLsizei)winH );
212   glMatrixMode(GL_PROJECTION);
213   glLoadIdentity();
214   glOrtho(0, winW, winH, 0, -1, 1);
215   glMatrixMode(GL_MODELVIEW);
216   glLoadIdentity();
217 }
218
219 void TTTView::display()
220 { // Display the board
221   // First clear
222   glClear(GL_COLOR_BUFFER_BIT);
223   glMatrixMode(GL_MODELVIEW);
224   glLoadIdentity();
```

Program ttt.cc (continued)

```
225     glColor3f(0, 0, 0);       // Black lines
226     glLineWidth(3);           // SLightly thick lines
227
228     // Dividing liens
229     glBegin(GL_LINES);
230     for (int v = 1; v < N_ROW; ++v)
231       {
232         GLint x = TTTView::winW / 3 * v;
233         GLint y = TTTView::winH / 50;
234         glVertex2i(x, y);
235         glVertex2i(x, TTTView::winH - y);
236       }
237     for (int v = 1; v < N_COL; ++v)
238       {
239         GLint x = TTTView::winW / 50;
240         GLint y = TTTView::winH / 3 * v;
241         glVertex2i(x, y);
242         glVertex2i(TTTView::winW - x, y);
243       }
244     glEnd();
245     // Now the occupied sauares
246     for (int s = 0; s < N_SQUARE; ++s)
247       {
248         int r = s / N_COL;
249         int c = s % N_ROW;
250         glPushMatrix();
251         // Move to center of square
252         glTranslatef(c * squareW + squareW / 2,
253                      r * squareH + squareH / 2,
254                      0);
255         if (model->board[s] == 'x')
256           { // Draw the x
257             glColor3f(1, 0, 0); // red
258             int w2 = squareW / 2;
259             int h2 = squareH / 2;
260             int x0 = w2 - w2 / 10;
261             int x1 = -x0;
262             int y0 = h2 - h2 / 10;
263             int y1 = -y0;
264             glBegin(GL_LINES);
265             glVertex2i(x0, y0);
266             glVertex2i(x1, y1);
267             glVertex2i(x1, y0);
268             glVertex2i(x0, y1);
269             glEnd();
270           }
271         else if (model->board[s] == 'o')
272           { // Draw the o
273             int radius = squareW / 2 - squareW / (2 * 10);
274             glColor3f(0, 0, 1); // blue
275             glBegin(GL_LINE_LOOP);
276             for (int i = 0; i < 360; ++i)
277               {
278                 double radians = (double)i / 360.0 * 2.0 * M_PI;
279                 double x1 = cos(radians) * radius;
280                 double y1 = sin(radians) * radius;
```

Program ttt.cc (continued)

```
281               glVertex2d(x1, y1);
282             }
283           glEnd();
284         }
285       glPopMatrix();
286     }
287   // See if winning row/col/diag
288   if (model->winRow >= 0)
289     {
290       int x0 = squareW / 10;
291       int x1 = squareW * N_COL - squareW / 10;
292       int y = squareH / 2 + model->winRow * squareW;
293       glLineWidth(5);
294       glColor4f(0.5, 0.5, 0.5, 0.5);
295       glBegin(GL_LINES);
296       glVertex2i(x0, y);
297       glVertex2i(x1, y);
298       glEnd();
299     }
300   if (model->winCol >= 0)
301     {
302       int y0 = squareH / 10;
303       int y1 = squareH * N_ROW - squareH / 10;
304       int x = squareW / 2 + model->winCol * squareH;
305       glLineWidth(5);
306       glColor4f(0.5, 0.5, 0.5, 0.5);
307       glBegin(GL_LINES);
308       glVertex2i(x, y0);
309       glVertex2i(x, y1);
310       glEnd();
311     }
312   glFlush();
313 }
314
315 void TTTView::mouse(int button, int state, int x, int y)
316 {
317   if (button == 2)
318     {
319       model->Clear();
320       glutPostRedisplay();
321       return;
322     }
323   if (button == 0 && state == 0)
324     { // Pressed, find which square
325       int c = x / squareW;
326       int r = y / squareH;
327       // Player move is always x
328       controller->Move(r * N_COL + c);
329     }
330 }
331
332 // Implement the controller
333 TTTController::TTTController(TTTModel* m)
334 {
335   model = m;
336 }
```

Program ttt.cc (continued)

```
337
338 void TTTController::Move(int square)
339 {
340   // Do nothing of not legal
341   if (!model->LegalMove(square)) return;
342   // First note the player's move
343   model->Move(square, 'x');
344   if (!model->GameOver())
345     {
346       if (model->nMoves == 1)
347         {
348           Response1();
349         }
350       else if (model->nMoves == 3)
351         {
352           Response2();
353         }
354       else
355         {
356           ResponseOther();
357         }
358     }
359   glutPostRedisplay();
360 }
361
362 void TTTController::Response1()
363 { // Respond to first player move is simple.
364   // If he takes middle, take corner, otherwise take middle
365   if (model->board[4] == ' ')
366     {
367       model->Move(4, 'o');
368     }
369   else
370     {
371       model->Move(0, 'o');
372     }
373 }
374
375 void TTTController::Response2()
376 { // Respond to players first move
377   // Make sure we don't need a blocking move
378   int s = BlockingMove();
379   if (s >= 0)
380     {
381       model->Move(s, 'o');
382       return;
383     }
384   // If either diag has 2 x's, we must take a side
385   if (model->xDiagCount[0] == 2 || model->xDiagCount[1] == 2)
386     {
387       int s = model->FindEmptySide();
388       model->Move(s, 'o');
389       return;
390     }
391   // Else take a corner
392   s = model->FindEmptyCorner();
```

Program ttt.cc (continued)

```
393   model->Move(s, 'o');
394   // This likely needs more work
395 }
396
397 void TTTController::ResponseOther()
398 { // Respond to players thrid and beyond moves
399   // First see if we have a winning move
400   for (int r = 0; r < N_ROW; ++r)
401     {
402       if (model->oRowCount[r] == 2 && model->xRowCount[r] == 0)
403         {
404           int s = model->FindEmptyCol(r);
405           model->Move(s, 'o');
406           return;
407         }
408     }
409   for (int c = 0; c < N_COL; ++c)
410     {
411       if (model->oColCount[c] == 2 && model->xColCount[c] == 0)
412         {
413           int s = model->FindEmptyRow(c);
414           model->Move(s, 'o');
415           return;
416         }
417     }
418   for (int d = 0; d < N_DIAG; ++d)
419     {
420       if (model->oDiagCount[d] == 2 && model->xDiagCount[d] == 0)
421         {
422           int s = model->FindEmptyDiag(d);
423           model->Move(s, 'o');
424           return;
425         }
426     }
427   // See if we need to block player's winning move
428   int s = BlockingMove();
429   if (s >= 0)
430     {
431       model->Move(s, 'o');
432       return;
433     }
434   // No immediate needs; choose a corner if availble else take a side
435   s = model->FindEmptyCorner();
436   if (s >= 0)
437     {
438       model->Move(s, 'o');
439       return;
440     }
441   s = model->FindEmptySide();
442   // No need to check -1 here as must be available
443   model->Move(s, 'o');
444 }
445
446 int TTTController::BlockingMove()
447 { // See if blocking move needed
448   for (int r = 0; r < N_ROW; ++r)
```

Program ttt.cc (continued)

```
449       {
450         if (model->xRowCount[r] == 2 && model->oRowCount[r] == 0)
451           {
452             int s = model->FindEmptyCol(r);
453             return s;
454           }
455       }
456     for (int c = 0; c < N_COL; ++c)
457       {
458         if (model->xColCount[c] == 2 && model->oColCount[c] == 0)
459           {
460             int s = model->FindEmptyRow(c);
461             return s;
462           }
463       }
464     for (int d = 0; d < N_DIAG; ++d)
465       {
466         if (model->xDiagCount[d] == 2 && model->oDiagCount[d] == 0)
467           {
468             int s = model->FindEmptyDiag(d);
469             return s;
470           }
471       }
472     return -1; // No blocking move found
473   }
```

Program ttt.cc (continued)

```
1  // Main program for model/view/controller example using Tic Tac Toe
2  // George F. Riley, Georgia Tech, Fall 2011
3
4  #ifdef OSX
5  #include <GLUT/glut.h>
6  #else
7  #include <GL/glut.h>
8  #endif
9
10 #include "ttt.h"
11
12 int main(int argc, char** argv)
13 {
14   // Instantiate the model, view and controller
15   TTTModel*      m = new TTTModel();
16   TTTController* c = new TTTController(m);
17   TTTView*       v = new TTTView(argc, argv, m, c);
18   // Opengl main loop
19   glutMainLoop();
20 }
```

Program TicTacToe.cc