```
1  using System; // add to allow Windows message box
2  using System.Runtime.InteropServices; // add to allow Windows message box
3
4  using Microsoft.Xna.Framework;
5  using Microsoft.Xna.Framework.Graphics;
6  using Microsoft.Xna.Framework.Input;
7  using Microsoft.Xna.Framework.Audio;
8  using System.Collections.Generic;
9
10 namespace Demo_MG_PlatformMovement
11 {
12     /// <summary>
13     /// enumeration of all possible game actions
14     /// </summary>
15     public enum GameAction
16     {
17         None,
18         PlayerRight,
19         PlayerLeft,
20         PlayerUp,
21         Quit
22     }
23
24     /// <summary>
25     /// This is the main type for your game.
26     /// </summary>
27     public class PlatformMovement : Game
28     {
29         // add code to allow Windows message boxes when running in a Windows enviro
           nment
30         [DllImport("user32.dll", CharSet = CharSet.Auto)]
31         public static extern uint MessageBox(IntPtr hWnd, String text, String
           caption, uint type);
32
33         // set the cell size in pixels
34         private const int CELL_WIDTH = 64;
35         private const int CELL_HEIGHT = 64;
36
37         // set the map size in cells
38         private const int MAP_CELL_ROW_COUNT = 8;
39         private const int MAP_CELL_COLUMN_COUNT = 10;
40
41         // set the window size
42         private const int WINDOW_WIDTH = MAP_CELL_COLUMN_COUNT * CELL_WIDTH;
43         private const int WINDOW_HEIGHT = MAP_CELL_ROW_COUNT * CELL_HEIGHT;
44
45         // wall objects
46         private Wall wall01;
47         private Wall wall02;
48
49         // player object
50         private Player player;
51
52         // variable to hold the player's current game action
53         GameAction playerGameAction;
54
55         // keyboard state objects to track a single keyboard press
56         KeyboardState newState;
57         KeyboardState oldState;
58
59         // declare a GraphicsDeviceManager object
60         GraphicsDeviceManager graphics;
61
62         // declare a SpriteBatch object
63         SpriteBatch spriteBatch;
64
```

```
65          public PlatformMovement()
66          {
67              graphics = new GraphicsDeviceManager(this);
68
69              // set the window size
70              graphics.PreferredBackBufferWidth = MAP_CELL_COLUMN_COUNT * CELL_WIDTH
                ;
71              graphics.PreferredBackBufferHeight = MAP_CELL_ROW_COUNT * CELL_HEIGHT;
72
73              Content.RootDirectory = "Content";
74          }
75
76          /// <summary>
77          /// Allows the game to perform any initialization it needs to before starti
            ng to run.
78          /// This is where it can query for any required services and load any non-g
            raphic
79          /// related content.  Calling base.Initialize will enumerate through any co
            mponents
80          /// and initialize them as well.
81          /// </summary>
82          protected override void Initialize()
83          {
84              // add floors, walls, and ceilings
85              wall01 = new Wall(Content, "wall", new Vector2(0, WINDOW_HEIGHT -
                CELL_HEIGHT));
86              wall01.Active = true;
87              wall02 = new Wall(Content, "wall", new Vector2(WINDOW_WIDTH -
                CELL_WIDTH, WINDOW_HEIGHT - CELL_HEIGHT));
88              wall02.Active = true;
89
90              // add the player
91              player = new Player(Content, new Vector2(CELL_WIDTH * 2, WINDOW_HEIGHT
                 - CELL_HEIGHT));
92              player.Active = true;
93
94              // set the player's initial speed
95              player.SpeedHorizontal = 2;
96              player.SpeedVertical = 2;
97
98              base.Initialize();
99          }
100
101         /// <summary>
102         /// LoadContent will be called once per game and is the place to load
103         /// all of your content.
104         /// </summary>
105         protected override void LoadContent()
106         {
107             // Create a new SpriteBatch, which can be used to draw textures.
108             spriteBatch = new SpriteBatch(GraphicsDevice);
109
110             // Note: wall and player sprites loaded when instantiated
111         }
112
113         /// <summary>
114         /// UnloadContent will be called once per game and is the place to unload
115         /// game-specific content.
116         /// </summary>
117         protected override void UnloadContent()
118         {
119             // Unload any non ContentManager content here
120         }
121
```

```
122         /// <summary>
123         /// Allows the game to run logic such as updating the world,
124         /// checking for collisions, gathering input, and playing audio.
125         /// </summary>
126         /// <param name="gameTime">Provides a snapshot of timing values.</param>
127         protected override void Update(GameTime gameTime)
128         {
129             // get the player's current action based on a keyboard event
130             playerGameAction = GetKeyboardEvents();
131
132             switch (playerGameAction)
133             {
134             case GameAction.None:
135                     break;
136
137                 // move player right
138             case GameAction.PlayerRight:
139                     if (!PlayerHitWall(wall02))
140                     {
141                         player.PlayerDirection = Player.Direction.Right;
142                         player.Position = new Vector2(player.Position.X + player
                     .SpeedHorizontal, player.Position.Y);
143                     }
144                     break;
145
146                 //move player left
147             case GameAction.PlayerLeft:
148                     if (!PlayerHitWall(wall01))
149                     {
150                         player.PlayerDirection = Player.Direction.Left;
151                         player.Position = new Vector2(player.Position.X - player
                     .SpeedHorizontal, player.Position.Y);
152                     }
153                     break;
154
155             case GameAction.PlayerUp:
156                     break;
157
158                 // quit game
159             case GameAction.Quit:
160                     Exit();
161                     break;
162
163             default:
164                     break;
165             }
166
167             base.Update(gameTime);
168         }
169
170         /// <summary>
171         /// This is called when the game should draw itself.
172         /// </summary>
173         /// <param name="gameTime">Provides a snapshot of timing values.</param>
174         protected override void Draw(GameTime gameTime)
175         {
176             GraphicsDevice.Clear(Color.CornflowerBlue);
177
178             spriteBatch.Begin();
179
180             wall01.Draw(spriteBatch);
181             wall02.Draw(spriteBatch);
182
183             player.Draw(spriteBatch);
184
185             spriteBatch.End();
186
187             base.Draw(gameTime);
188         }
```

```
189 
190         /// <summary>
191         /// get keyboard events
192         /// </summary>
193         /// <returns>GameAction</returns>
194         private GameAction GetKeyboardEvents()
195         {
196             GameAction playerGameAction = GameAction.None;
197 
198             newState = Keyboard.GetState();
199 
200 
201             if (CheckKey(Keys.Right) == true)
202             {
203                 playerGameAction = GameAction.PlayerRight;
204             }
205             else if (CheckKey(Keys.Left) == true)
206             {
207                 playerGameAction = GameAction.PlayerLeft;
208             }
209             else if (CheckKey(Keys.Escape) == true)
210             {
211                 playerGameAction = GameAction.Quit;
212             }
213 
214             oldState = newState;
215 
216             return playerGameAction;
217         }
218 
219         /// <summary>
220         /// check the current state of the keyboard against the previous state
221         /// </summary>
222         /// <param name="theKey">bool new key press</param>
223         /// <returns></returns>
224         private bool CheckKey(Keys theKey)
225         {
226             // allows the key to be held down
227             return newState.IsKeyDown(theKey);
228 
229             // player must continue to tap the key
230             //return oldState.IsKeyDown(theKey) && newState.IsKeyUp(theKey);
231         }
232 
233         /// <summary>
234         /// test for player collision with a wall object
235         /// </summary>
236         /// <param name="wall">wall object to test</param>
237         /// <returns>true if collision</returns>
238         private bool PlayerHitWall(Wall wall)
239         {
240             return player.BoundingRectangle.Intersects(wall.BoundingRectangle);
241         }
242     }
243 }
```