```csharp
using System; // add to allow Windows message box
using System.Runtime.InteropServices; // add to allow Windows message box

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Audio;
using System.Collections.Generic;

namespace Demo_MG_MazeGame
{
    /// <summary>
    /// enumeration of all possible game actions
    /// </summary>
    public enum GameAction
    {
        None,
        PlayerRight,
        PlayerLeft,
        Quit
    }

    /// <summary>
    /// This is the main type for your game.
    /// </summary>
    public class MazeGame : Game
    {
        // add code to allow Windows message boxes when running in a Windows environment
        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        public static extern uint MessageBox(IntPtr hWnd, String text, String caption, uint type);

        // set the cell size in pixels
        private const int CELL_WIDTH = 64;
        private const int CELL_HEIGHT = 64;

        // set the map size in cells
        private const int MAP_CELL_ROW_COUNT = 8;
        private const int MAP_CELL_COLUMN_COUNT = 10;

        // set the window size
        private const int WINDOW_WIDTH = MAP_CELL_COLUMN_COUNT * CELL_WIDTH;
        private const int WINDOW_HEIGHT = MAP_CELL_ROW_COUNT * CELL_HEIGHT;

        // wall objects
        private Wall wall01;
        private Wall wall02;

        // player object
        private Player player;

        // variable to hold the player's current game action
        GameAction playerGameAction;

        // keyboard state objects to track a single keyboard press
        KeyboardState newState;
        KeyboardState oldState;

        // declare a GraphicsDeviceManager object
        GraphicsDeviceManager graphics;

        // declare a SpriteBatch object
        SpriteBatch spriteBatch;

        public MazeGame()
        {
            graphics = new GraphicsDeviceManager(this);

            // set the window size
            graphics.PreferredBackBufferWidth = MAP_CELL_COLUMN_COUNT * CELL_WIDTH
            ;
            graphics.PreferredBackBufferHeight = MAP_CELL_ROW_COUNT * CELL_HEIGHT;


            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starti
        ng to run.
        /// This is where it can query for any required services and load any non-g
        raphic
        /// related content.  Calling base.Initialize will enumerate through any co
        mponents
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // add floors, walls, and ceilings
            wall01 = new Wall(Content, "wall", new Vector2(0, WINDOW_HEIGHT -
            CELL_HEIGHT));
            wall01.Active = true;
            wall02 = new Wall(Content, "wall", new Vector2(WINDOW_WIDTH -
            CELL_WIDTH, WINDOW_HEIGHT - CELL_HEIGHT));
            wall02.Active = true;

            // add the player
            player = new Player(Content, new Vector2(CELL_WIDTH * 2, WINDOW_HEIGHT
             - CELL_HEIGHT));
            player.Active = true;

            // set the player's initial speed
            player.SpeedHorizontal = 5;
            player.SpeedVertical = 5;

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // Note: wall and player sprites loaded when instantiated
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// game-specific content.
        /// </summary>
        protected override void UnloadContent()
        {
            // Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            // get the player's current action based on a keyboard event
            playerGameAction = GetKeyboardEvents();

            switch (playerGameAction)
            {
            case GameAction.None:
                    break;

                // move player right
            case GameAction.PlayerRight:
                    if (!PlayerHitWall(wall02))
                    {
                        player.PlayerDirection = Player.Direction.Right;
                        player.Position = new Vector2(player.Position.X + player
                    .SpeedHorizontal, player.Position.Y);
                    }
                    break;

                //move player left
            case GameAction.PlayerLeft:
                    if (!PlayerHitWall(wall01))
                    {
                        player.PlayerDirection = Player.Direction.Left;
                        player.Position = new Vector2(player.Position.X - player
                    .SpeedHorizontal, player.Position.Y);
                    }
                    break;

                // quit game
            case GameAction.Quit:
                    Exit();
                    break;

            default:
                    break;
            }

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            spriteBatch.Begin();

            wall01.Draw(spriteBatch);
            wall02.Draw(spriteBatch);

            player.Draw(spriteBatch);

            spriteBatch.End();

            base.Draw(gameTime);
        }

        /// <summary>
        /// get keyboard events
        /// </summary>
        /// <returns>GameAction</returns>
        private GameAction GetKeyboardEvents()
        {
            GameAction playerGameAction = GameAction.None;
```

```
                newState = Keyboard.GetState();

                if (CheckKey(Keys.Right) == true)
                {
                    playerGameAction = GameAction.PlayerRight;
                }
                else if (CheckKey(Keys.Left) == true)
                {
                    playerGameAction = GameAction.PlayerLeft;
                }
                else if (CheckKey(Keys.Escape) == true)
                {
                    playerGameAction = GameAction.Quit;
                }

                oldState = newState;

                return playerGameAction;
            }

            /// <summary>
            /// check the current state of the keyboard against the previous state
            /// </summary>
            /// <param name="theKey">bool new key press</param>
            /// <returns></returns>
            private bool CheckKey(Keys theKey)
            {
                // allows the key to be held down
                return newState.IsKeyDown(theKey);

                // player must continue to tap the key
                //return oldState.IsKeyDown(theKey) && newState.IsKeyUp(theKey);
            }

            /// <summary>
            /// test for player collision with a wall object
            /// </summary>
            /// <param name="wall">wall object to test</param>
            /// <returns>true if collision</returns>
            private bool PlayerHitWall(Wall wall)
            {
                return player.BoundingRectangle.Intersects(wall.BoundingRectangle);
            }
        }
    }
```