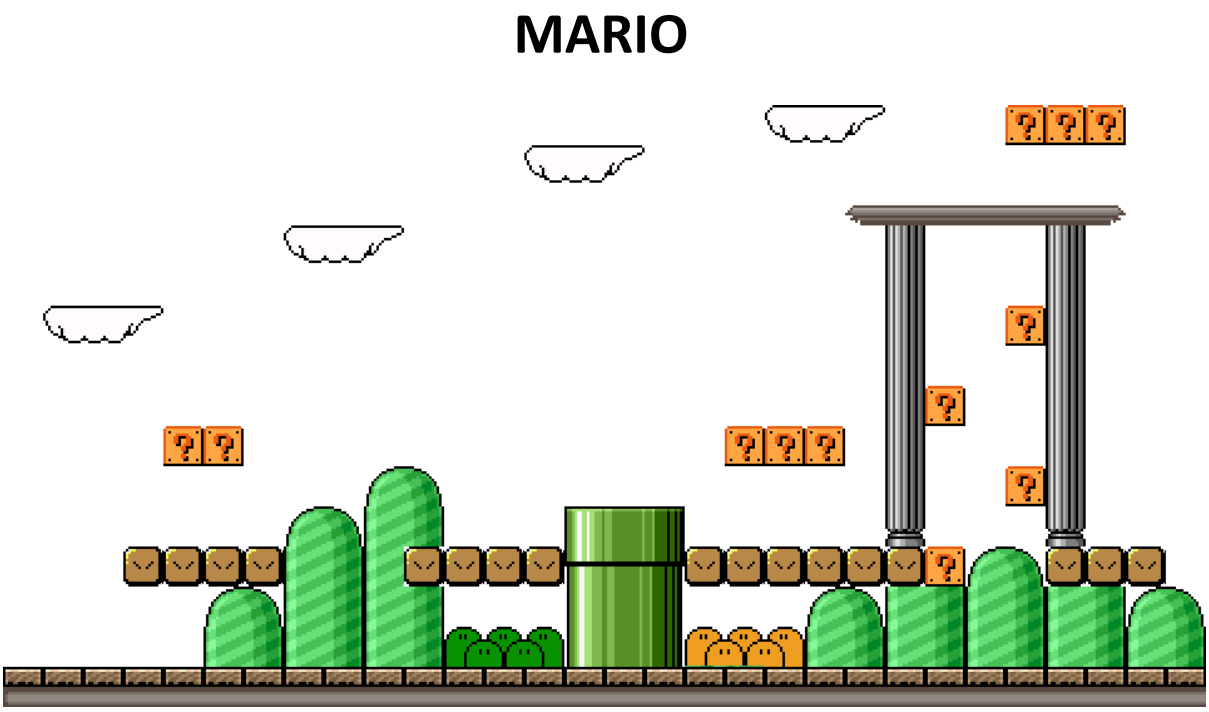
Make your own game



Introduction

This walkthrough will take you through the steps required to develop a simplified 2D Mario game using C# and XNA.

You will be able to design your own levels using a Tile Based editor, which you can then give to friends and family to play.

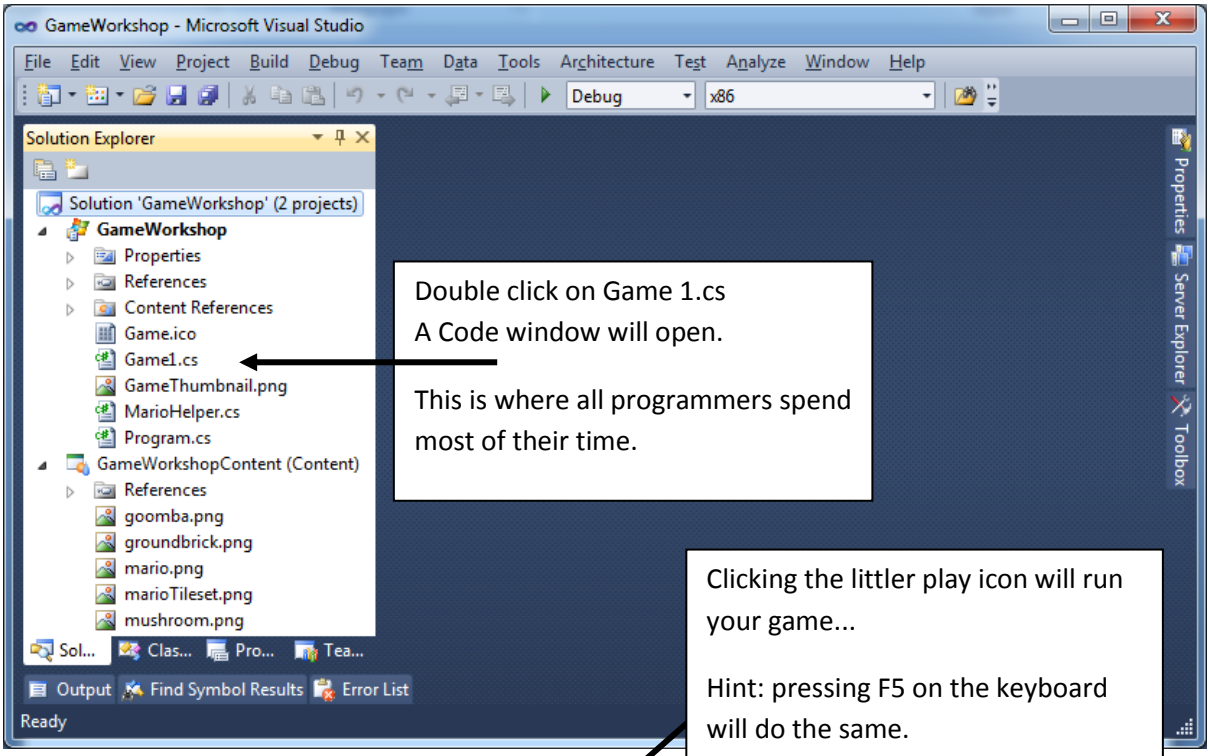
Required Software:

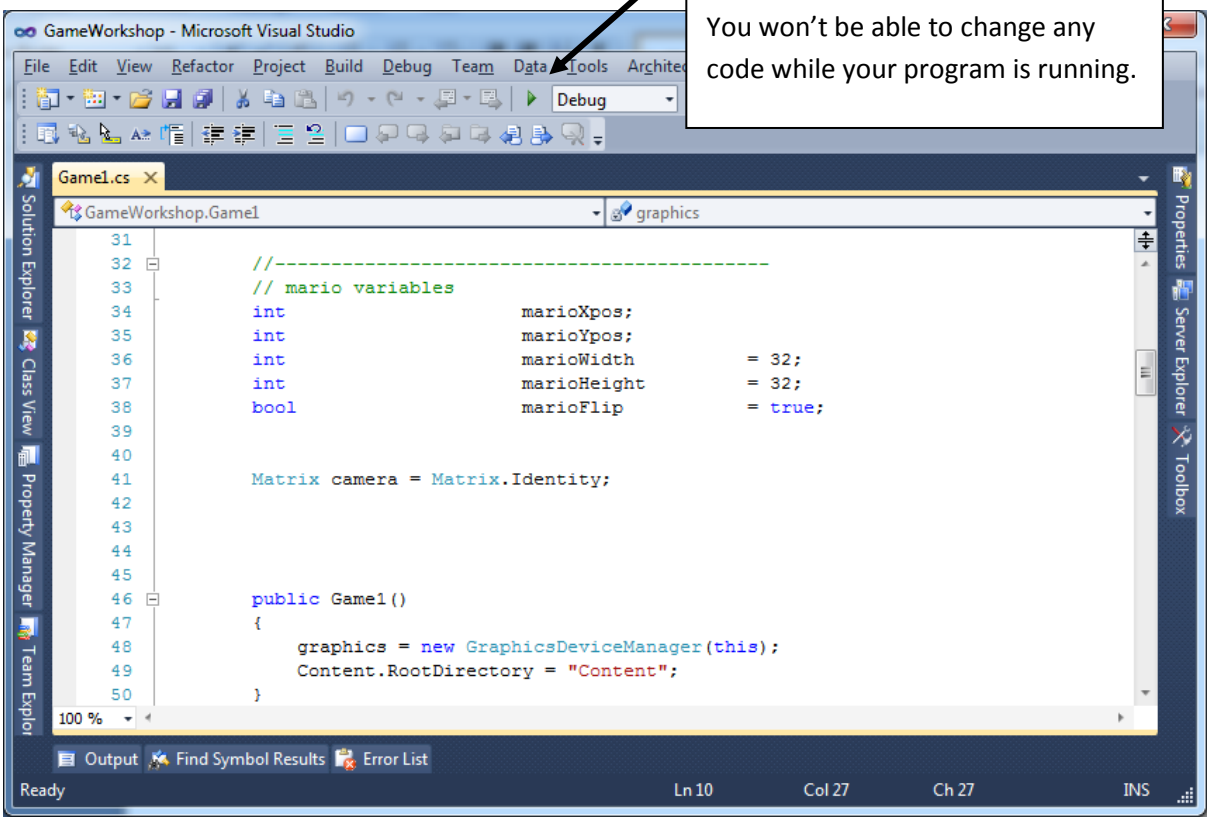
- Microsoft Visual C# Express
  www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express
- Microsoft XNA Game Studio 4.0
  www.microsoft.com/download/en/details.aspx?id=23714
- Tiled Map Editor
  www.mapeditor.org
- Gimp
  www.gimp.org

Let's Begin:

Open the Visual studio project supplied with this tutorial.

The interface:





Let's really get started:

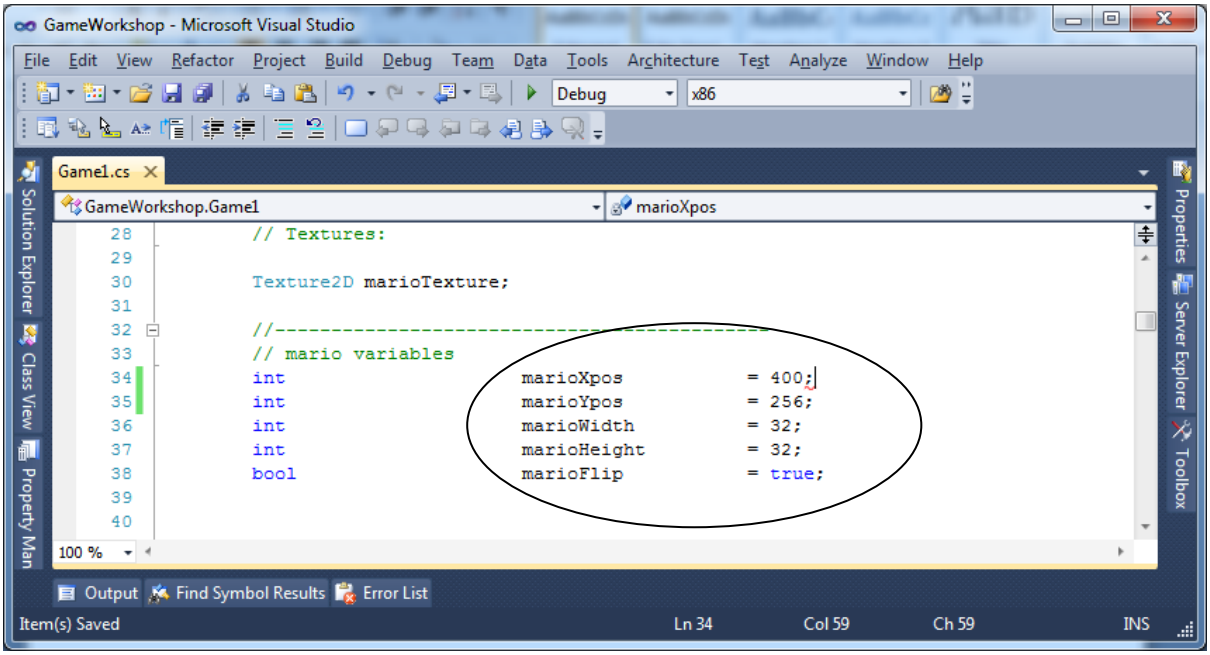
You will get more familiar with the code the more you use it... just follow along for now.

Changing variables:

Scroll down until you see the following window.

Here we have defined a few variables to represent Mario's position and size on the screen.

Changing these values will alter where Mario is drawn in out window, change the values and click run to see your results.



Continue on next page

Moving Mario:

Where going to start by getting Mario to move around the screen. We will move him in a direction by an amount.

Copy the following underneath the update function...

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    //---------------------------------
    // Mario Update


    // End Mario Update
    //---------------------------------

    base.Update(gameTime);
}

bool MoveMario(Direction dir, int amount)
{
    switch (dir)
    {
        case Direction.LEFT:    marioXpos -= amount; break;
        case Direction.RIGHT:   marioXpos += amount; break;
        case Direction.UP:      marioYpos -= amount; break;
        case Direction.DOWN:    marioYpos += amount; break;
    }

    return false;
}
```

Now to make Mario move left, right, up or down, add the following to the update functions.

```
//---------------------------------
// Mario Update

MoveMario(Direction.LEFT, 5);

// End Mario Update
//---------------------------------
```

Continue on next page

Keyboard Input:

With some modifications, we will get Mario to move left and right when we press keys on the keyboard.

Scroll down to our update function, and type the following

```
//--------------------------------
// Mario Update
    if (Keyboard.GetState().IsKeyDown(Keys.Left))
    {
        MoveMario(Direction.LEFT, 5);
        marioFlip = true;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        MoveMario(Direction.RIGHT, 5);
        marioFlip = false;
    }
// End Mario Update
//--------------------------------
```

Not using magic numbers:

If we want to change Mario's speed then we need to change the value of 5 to something else for both left and right key presses.

We're going to be using a variable instead.

Scroll to the top and create a new variable called marioMoveSpeed, then change our value of 5 to marioMoveSpeed.

```
//-----------------------------------------
// mario variables
int                     marioXpos           = 400;
int                     marioYpos           = 256;
int                     marioWidth          = 32;
int                     marioHeight         = 32;
bool                    marioFlip           = true;

int                    marioMoveSpeed       = 10;
```

```
//--------------------------------
// Mario Update
    if (Keyboard.GetState().IsKeyDown(Keys.Left))
    {
        MoveMario(Direction.LEFT, marioMoveSpeed);
        marioFlip = true;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        MoveMario(Direction.RIGHT, marioMoveSpeed);
        marioFlip = false;
    }
// End Mario Update
//--------------------------------
```

Continue on next page

Adding gravity:

Create another variable at the top called gravity and while where at it, 2 other variables for jumping. We will use those next.

```
//-----------------------------------------
// mario variables
int                     marioXpos           = 400;
int                     marioYpos           = 256;
int                     marioWidth          = 32;
int                     marioHeight         = 32;
bool                    marioFlip           = true;

int                     marioMoveSpeed      = 10;

int                     marioJumpForce      = 20;
int                     marioMaxJumpForce   = 20;
int                     gravity             = 5;
```

We want Mario to give the illusion of falling, and stopping when he hits the ground.
Add the following back in the update function.

```
...

    // apply gravity
    MoveMario(Direction.DOWN, gravity);

// End Mario Update
//------------------------------
```

We also need to modify the Move Mario Function

```
bool MoveMario(Direction dir, int amount)
{
    switch (dir)
    {
        case Direction.LEFT:    marioXpos -= amount; break;
        case Direction.RIGHT:   marioXpos += amount; break;
        case Direction.UP:      marioYpos -= amount; break;
        case Direction.DOWN:    marioYpos += amount; break;
    }

    if (marioYpos > windowHeight - marioHeight)
    {
        marioYpos = windowHeight - marioHeight;
        return true; // return true as mario collided.
    }

    return false;
}
```

Don’t forget to run the game, make sure there are no errors before continuing.

Continue on next page

Jumping:

When we press space make Mario move up, then reduce his jumpForce for next frame. Eventually Mario will start to fall.

When he collides with the ground, reset his jumpForce back to his max jump force, so he can jump again.

```
    if (Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        MoveMario(Direction.RIGHT, marioMoveSpeed);
        marioFlip = false;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        MoveMario(Direction.UP, marioJumpForce);
        marioJumpForce -= 1;
    }

    // apply gravity
    if (MoveMario(Direction.DOWN, gravity) == true)
    {
        marioJumpForce = marioMaxJumpForce;
    }

// End Mario Update
//--------------------------------
```

Continue on next page

Adding Tiled Background:

This is where things start to get a little more interesting.

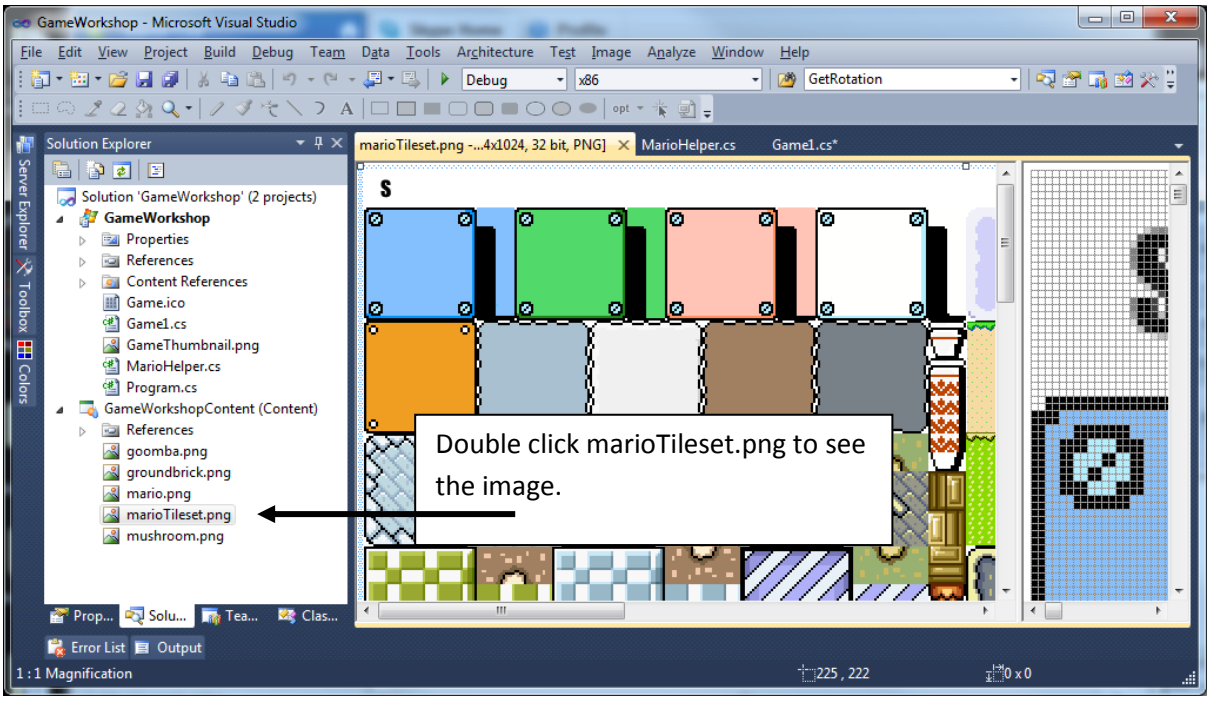
We are going to generate our levels using a tile set, something like this, or you can make your own.

Each tile in the tile set is numbered in order; we will be using those numbers in an array to display our level.

Back at the top under our variables, insert the following.

```
int[,] map =
{
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{676,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,677,6
77,677}
};
```

Before we get ahead of ourselves, we need o Load the tile-set texture into our game. This tile-set texture has already been added to our solution.



Back to the top of the file, just above our Mario variables, we need to make a new Texture2D variable to hold the image.

```
        //--------------------------------------------
        // Textures:

        Texture2D marioTexture;
        Texture2D tilesetTexture;

        //--------------------------------------------
       // mario variables
        int                     marioXpos           = 400;
        int                     marioYpos           = 256;
```

Now in the load content function, load in the texture.

```
/// <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);

    marioTexture = Content.Load<Texture2D>("mario");
    tilesetTexture = Content.Load<Texture2D>("marioTileset");
}
```

Finally we can draw the tile-set on our window.

Add the following function below the Draw function toward the very bottom of the file. This will draw any tile map data using any given texture.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin(SpriteSortMode.Immediate, null, null, null, null, null, camera);



    //------------------------------------------------
    // draw mario

    spriteBatch.Draw(
        marioTexture,
        new Rectangle(marioXpos, marioYpos, marioWidth, marioHeight),
        null,
        Color.White, 0.0f, Vector2.Zero,
        marioFlip ? SpriteEffects.FlipHorizontally : SpriteEffects.None, 0);

    // End draw mario
    //------------------------------------------------

    spriteBatch.End();

    base.Draw(gameTime);
}
public void DrawTileMap(Texture2D tileset, int[,] mapData, int TileWidth, int TileHeight)
{
    int yLen = mapData.GetLength(0);
    int xLen = mapData.GetLength(1);
    int numXTiles = tileset.Width / TileWidth;
    for (int y = 0; y < yLen; y++)
    {
        for (int x = 0; x < xLen; x++)
        {
            if (mapData[y, x] == 0)
                continue;

            int tileIndex = mapData[y, x];
            int tileXPos = ((tileIndex % numXTiles) - 1) * TileWidth;
            int tileYPos = ((tileIndex / numXTiles)) * TileHeight;

            Rectangle dstRect =
                 new Rectangle(x * TileWidth, y * TileHeight, TileWidth, TileHeight);

            Rectangle srcRect =
                 new Rectangle(tileXPos, tileYPos, TileWidth, TileHeight);

            spriteBatch.Draw(tileset, dstRect, srcRect, Color.White);
        }
    }
}
```

We just need to call that function now… In the Draw function, insert the following.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin(SpriteSortMode.Immediate, null, null, null, null, null, camera);


    DrawTileMap(tilesetTexture, map, marioWidth, marioHeight);

    //-------------------------------------------------
    // draw mario

    spriteBatch.Draw(
        marioTexture,
        new Rectangle(marioXpos, marioYpos, marioWidth, marioHeight),
        null,
        Color.White, 0.0f, Vector2.Zero,
        marioFlip ? SpriteEffects.FlipHorizontally : SpriteEffects.None, 0);

    // End draw mario
    //-------------------------------------------------

    spriteBatch.End();

    base.Draw(gameTime);
}
```

Assuming you have used the same tile-set, you should have something like:

Colliding with the level:

Remember that Move Mario Function we created earlier...

Where going to modify that function to check for a collision in the level, and stop Mario from moving into solid blocks.

```
bool MoveMario(Direction dir, int amount)
{
    Point p = new Point(marioXpos, marioYpos);


    if (dir == Direction.LEFT || dir == Direction.RIGHT)
    {
        p.Y += marioHeight / 2;

        if (dir == Direction.RIGHT) p.X = marioXpos + marioWidth;
    }
    else
    {
        p.X += marioWidth / 2;
        if (dir == Direction.DOWN) p.Y = marioYpos + marioHeight;
    }

    p = MovePointInDir(dir, amount, p);
    int x = p.X / marioWidth;
    int y = p.Y / marioHeight;

    if (p.X < 0) x = -1;
    if (p.Y < 0) y = -1;

    Point marioPos = new Point(marioXpos, marioYpos);
    marioPos = MovePointInDir(dir, amount, marioPos);

    bool collided = false;


    if (!(y >= map.GetLength(0) || y < 0 || x >= map.GetLength(1) || x < 0))
    {
        if (map[y,x] != 0)
        {
            // snap into place
            if (dir == Direction.UP) marioPos.Y = (y + 1) * marioHeight;
            if (dir == Direction.DOWN) marioPos.Y = (y - 1) * marioHeight;
            if (dir == Direction.LEFT) marioPos.X = (x + 1) * marioWidth;
            if (dir == Direction.RIGHT) marioPos.X = (x - 1) * marioWidth;

            collided = true;
        }
    }

    marioXpos = marioPos.X;
    marioYpos = marioPos.Y;

    return collided;
}
```

Make sure there are no errors, and run the game.

Careful, Mario will fall off the edge if you move him off the screen.

Continue on next page

Adding a camera:

Back in the update function, add the following.

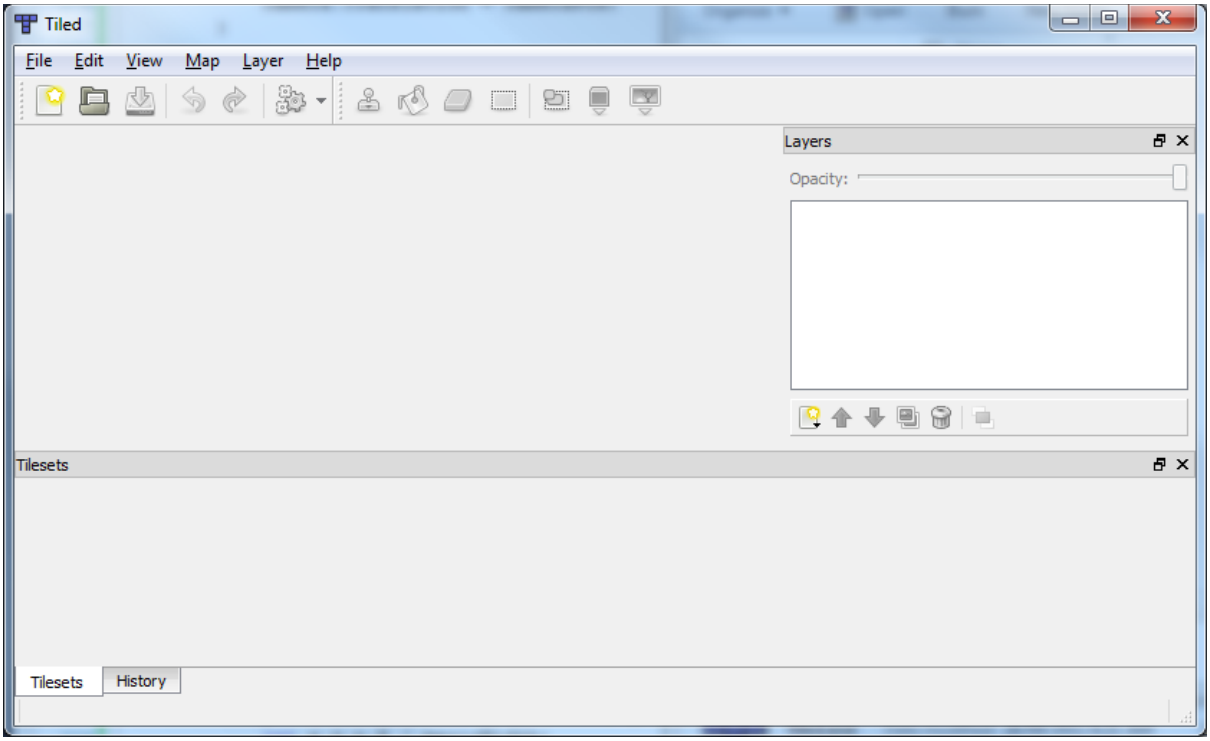
```
// apply gravity
if (MoveMario(Direction.DOWN, gravity) == true)
{
    marioJumpForce = marioMaxJumpForce;
}


camera.Translation =
    new Vector3(-marioXpos + (windowWidth) / 2,
                -marioYpos + (windowHeight - (marioHeight * 2)), 0);

// End Mario Update
//---------------------------------
```

Continue on next page

More Elaborate Levels:

Where going to make some levels using a program called Tiled. It's a free tile based map making program, available from http://www.mapeditor.org

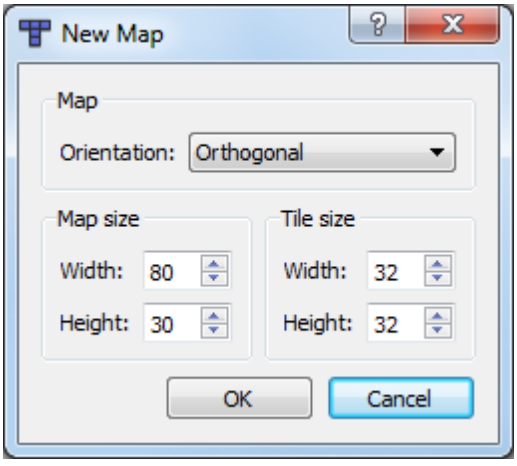
Open the program, available with this tutorial, or download from the web site above.



Click File->New and set the Map Width and Height to about 80 and 30.

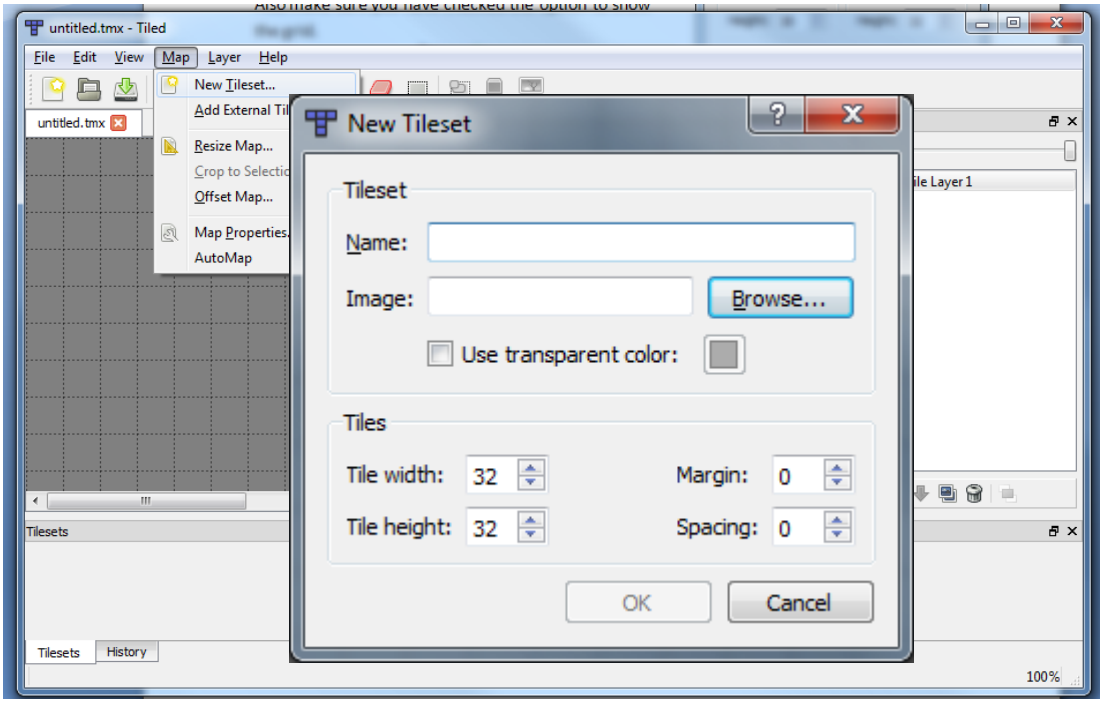
You can use any dimensions you like though.

Its best to save the project next, save it in the maps folder with this doc, this allows us to manage our project more easily.

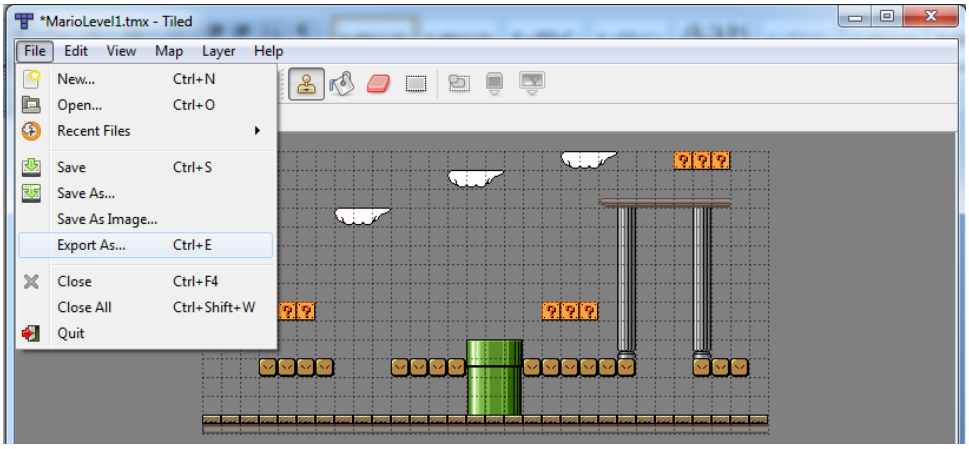


We want to add a new Tile set to this map. Click map->New Tile Set...

Browse for the image you want to use. We used the one in the images folder contained with this document. Then click OK.



Play around the program for a while, create some maps, and find something you're happy with. When you're done, we will put it in our game...



Click File->Export As.
Make sure to select the save as type "Lua Files (*.lua)"

Now open this file in Notepad++ or visual studio. (Not windows notepad)

Scroll to the bottom, and copy to the clipboard all the numbers from the opening bracket to the closing bracket.

```
data =
{

/* SELECT ALL NUMBERS BETWEEN THE OPENING AND CLOSING BRACKET */
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 540, 540, 540, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 612, 613, 613, 613, 613, 613, 613, 613, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 540, 540, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 540, 540, 540, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 427, 428, 429, 0, 0, 0, 0, 0, 979, 0, 0, 0, 979, 0, 0, 0,
0, 0, 0, 634, 634, 634, 634, 634, 0, 0, 0, 634, 634, 634, 634, 634, 459, 460, 461, 634, 634, 634, 634, 634, 634, 0, 0, 0, 634, 634, 634, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 491, 492, 493, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
676, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677
}
```

This information represents the same information we need in our map array defined earlier. Unfortunately, it's not quite formatted properly for our use.

Back in visual studio - Paste it over our map array.
at the beginning of each line – insert a {
at the end of each line, before the last comma – insert a }

It should now simular to the following:

```
int[,] map =
{
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 540, 540, 540, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 612, 613, 613, 613, 613, 613, 613, 613, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 941, 942, 943, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 540, 540, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 540, 540, 540, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 947, 0, 0, 0, 947, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 427, 428, 429, 0, 0, 0, 0, 0, 979, 0, 0, 0, 979, 0, 0, 0},
{0, 0, 0, 634, 634, 634, 634, 0, 0, 0, 634, 634, 634, 634, 459, 460, 461, 634, 634, 634, 634, 634, 634, 0,
0, 0, 634, 634, 634, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 491, 492, 493, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 491, 492, 493, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{676, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677, 677,
677, 677, 677, 677, 677, 677, 677, 677, 677}
};
```

Continue on next page

Creating a background:

There are a few things we can do to create a background for our level.

- Use Photoshop to make background image
- Create a new layer in Tiles, and save the layer as an image (preferred method)

However you want to create your background, it's always a good idea to make it the same size as our level.

For example – if our level is 80 tiles wide, and each tile is 32 pixels wide. Then our background size is 80 * 32 = 2560 pixels.

Same for our height. 30 * 32 = 960 pixels high;

Once you have created a background image,

Continue on next page.

Adding the background to our project:

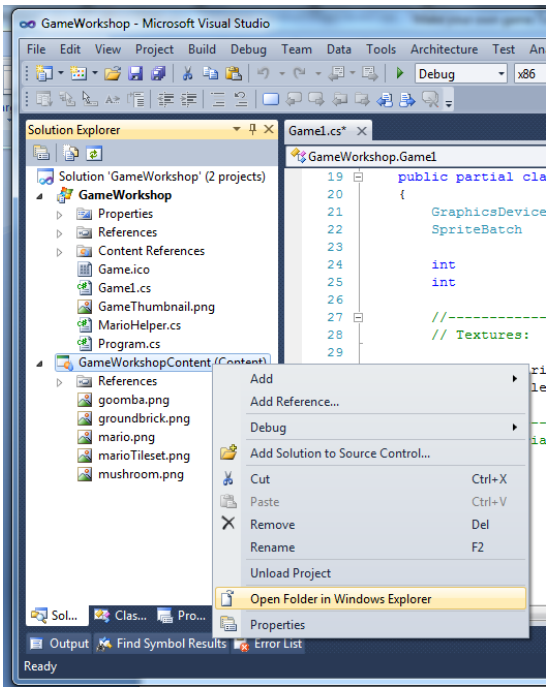
Right click on our the Content project and select “Open Directory in Explorer”

Save your background image into this folder.

Now once again, right click on the content project and click Add->Existing Item...

Select the image you just saved, and click ok.

At the top of our project, add a new texture variable.




```
//----------------------------------------
// Textures:
Texture2D marioTexture;
Texture2D tilesetTexture;

Texture2D backgroundTexture;
```

Load the texture:

```
    marioTexture = Content.Load<Texture2D>("mario");
    tilesetTexture = Content.Load<Texture2D>("marioTileset");

    backgroundTexture = Content.Load<Texture2D>("Background");
}
```

Draw the background:

```
spriteBatch.Begin(SpriteSortMode.Immediate, null, null, null, null, null, camera);

spriteBatch.Draw(backgroundTexture, new Vector2(0, 0), Color.White);

DrawTileMap(tilesetTexture, map, marioWidth, marioHeight);
```

End of Day 1

Save your work for today, create new levels and backgrounds.
Instruction from your teacher will follow soon.

Tomorrow we will create enemies, pickups, and a simple HUD

Adding Game Objects

Just like when we created an array for the tiles in the world, we are going to make an array for the objects in the world, such as coins, enemies, and other objects.

For now, in our object map, a 2 will represent coins. This is because it is the second image in the tile array; likewise, a goomba will have an id of 4...

Add the following below the map array:

```
enum MarioObjectType
{
    COIN = 2,
    GOOMBA = 4
};

class MarioObject
{
    public int xPos;
    public int yPos;
    public int tileXPos;
    public int tileYPos;
    public MarioObjectType id;

    public Direction dir;
};

List<MarioObject> marioObjects = new List<MarioObject>();
```

The MarioObject above contains an objects position and id.
We have also defined a list of Mario objects, we can loop through all the Mario objects and draw them to the screen soon.

Add the following below the marioObjects List; this is an array representing where to create objects. We will be using tiled to retrieve this information, in the same way we created the levels before.

```
int[,] objectMap =
{
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

Now we need a way to populate our list of objects in the world using the information above. Add the following, in the LoadContent function.

```
backgroundTexture = Content.Load<Texture2D>("background");

for (int y = 0; y < objectMap.GetLength(0); y++)
{
    for (int x = 0; x < objectMap.GetLength(1); x++)
    {
        if (objectMap[y, x] == 0)
            continue;

        MarioObject obj = new MarioObject();
        obj.xPos = x * marioWidth;
        obj.yPos = y * marioHeight;

        int numXTiles = tilesetTexture.Width / marioWidth;

        obj.tileXPos = ((objectMap[y, x] % numXTiles) - 1) * marioWidth;
        obj.tileYPos = ((objectMap[y, x] / numXTiles)) * marioHeight;

        obj.id = (MarioObjectType)objectMap[y, x];

        obj.dir = Direction.LEFT;

        marioObjects.Add(obj);
    }
}
```

Finally we can draw this object. Move to the draw function.

```
// Draw The Level
//---------------
DrawTileMap(tilesetTexture, map, marioWidth, marioHeight);

// draw game objects
//------------------
for (int i = 0; i < marioObjects.Count; i++)
{

Rectangle dstRect = new Rectangle(marioObjects[i].xPos, marioObjects[i].yPo
s, marioWidth, marioHeight);

Rectangle srcRect = new Rectangle(marioObjects[i].tileXPos, marioObjects[i]
.tileYPos, marioWidth, marioHeight);

    spriteBatch.Draw(tilesetTexture, dstRect, srcRect, Color.White);
}

//-------------------------------------------------
// draw mario
```

Compile and run, you should see some game objects drawn on the screen, however will not update just yet.

Updating the game objects

We will create some new functions to update the game objects. UpdateMarioObjects will loop through each game object, and check if its a coin or goomba or some other type of object.

```
        ...

    // End Mario Update
    //--------------------------------


    UpdateMarioObjects(gameTime);

    base.Update(gameTime);
}

void UpdateMarioObjects(GameTime gameTime)
{
    for (int i = 0; i < marioObjects.Count; i++)
    {
        if (marioObjects[i].id == MarioObjectType.COIN)
        {
            UpdateCoin(marioObjects[i], gameTime);
        }

        if (marioObjects[i].id == MarioObjectType.GOOMBA)
        {
            UpdateGoomba(marioObjects[i], gameTime);
        }
    }
}

void UpdateCoin(MarioObject obj, GameTime gameTime)
{

}

void UpdateGoomba(MarioObject obj, GameTime gameTime)
{

}


bool MoveMario(Direction dir, int amount)
{
...
```

Currently the UpdateCoin and UpdateGoomba do not do anything, add the following to make the coin bob up and down.

```
void UpdateCoin(MarioObject obj, GameTime gameTime)
{
    obj.yPos += (float)
   (Math.Cos(gameTime.TotalGameTime.TotalSeconds * 8.0) * 0.2f);
}
```

We need another function to test if Mario collides with any of these objects, create another function... Add the following.

```
bool objectCollidesWithMario(MarioObject obj)
{
    return (marioXpos < obj.xPos + marioWidth && marioXpos + marioWidth > obj.xPos &&
            marioYpos < obj.yPos + marioHeight && marioYpos + marioHeight > obj.yPos);
}
```

TODO:
We can now check if Mario collides with a coin.

```
void UpdateCoin(MarioObject obj, GameTime gameTime)
{
    if( objectCollidesWithMario( obj )
    {
    }
}
```