DigiPen
INSTITUTE OF TECHNOLOGY

CS 176
Advanced Scripting

Tile-Based Games

Antoine Abi Chacra
Elie Abi Chahine

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Introduction

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Introduction

Tiles were already used long-long time ago for making games. It was the time, when computers didn't have speeds of GHz and hundreds of MB memory. Slow speed and limited amount of memory meant game makers had to use their brains and create clever ways to make games look better and run faster.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Introduction

So, you want to put nice background into your game, but the picture would be too large and make the game very slow. What to do? Slice the picture into tiles!

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Introduction

Other nice feature about tiles is, when you want to replace part of your background, then you don't have to redraw everything, you can only replace 1 tile.

You can reuse the tiles with different objects too. For example you might have tile with grass and another tile with flower on the grass, then you can take same grass background and only draw the flower.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

In other words…

Tile based games are special in many ways

- Easy object creation (using a tile map)
- Fast collision checking using a binary map (or enhanced binary map which I will talk about later)
- Enhancing the rendering by only updating the tiles that changed

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Tile Map

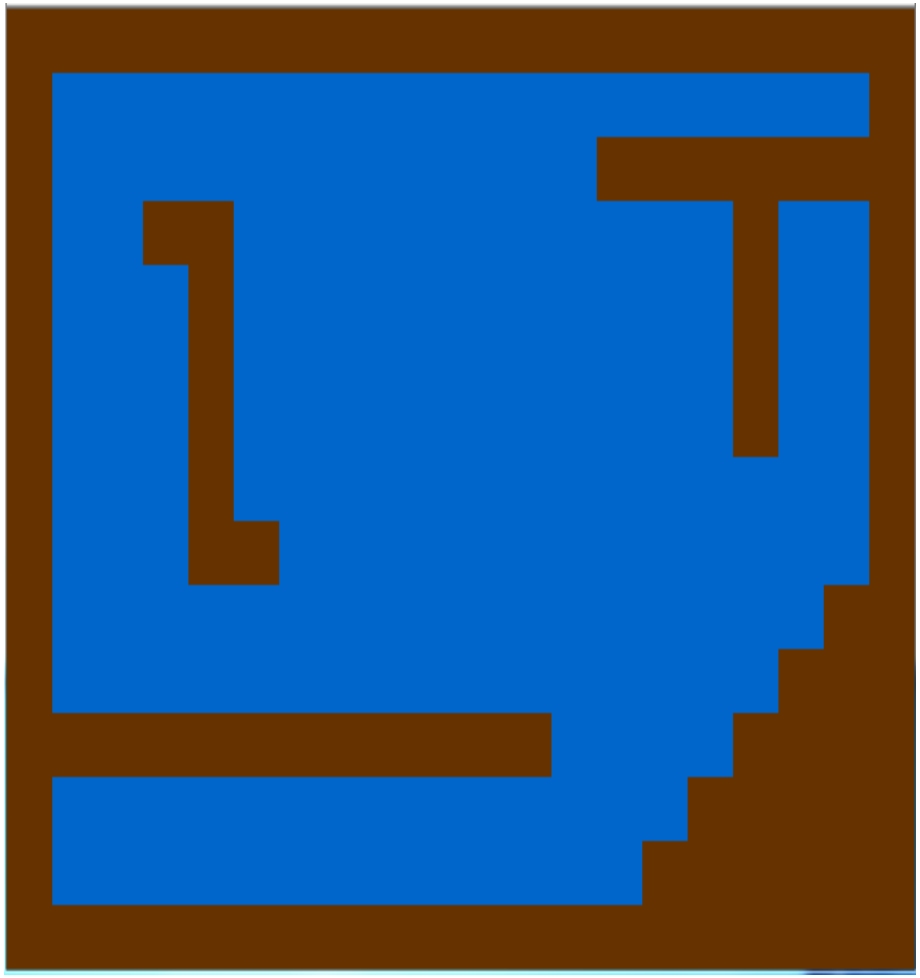
Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Object Creation

- A tile map is just another array filled with numbers
- Every number represents a certain tile or object
- Knowing the width and height of a tile and the length of the array (rows and columns) is enough to be able to place the objects on the screen

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

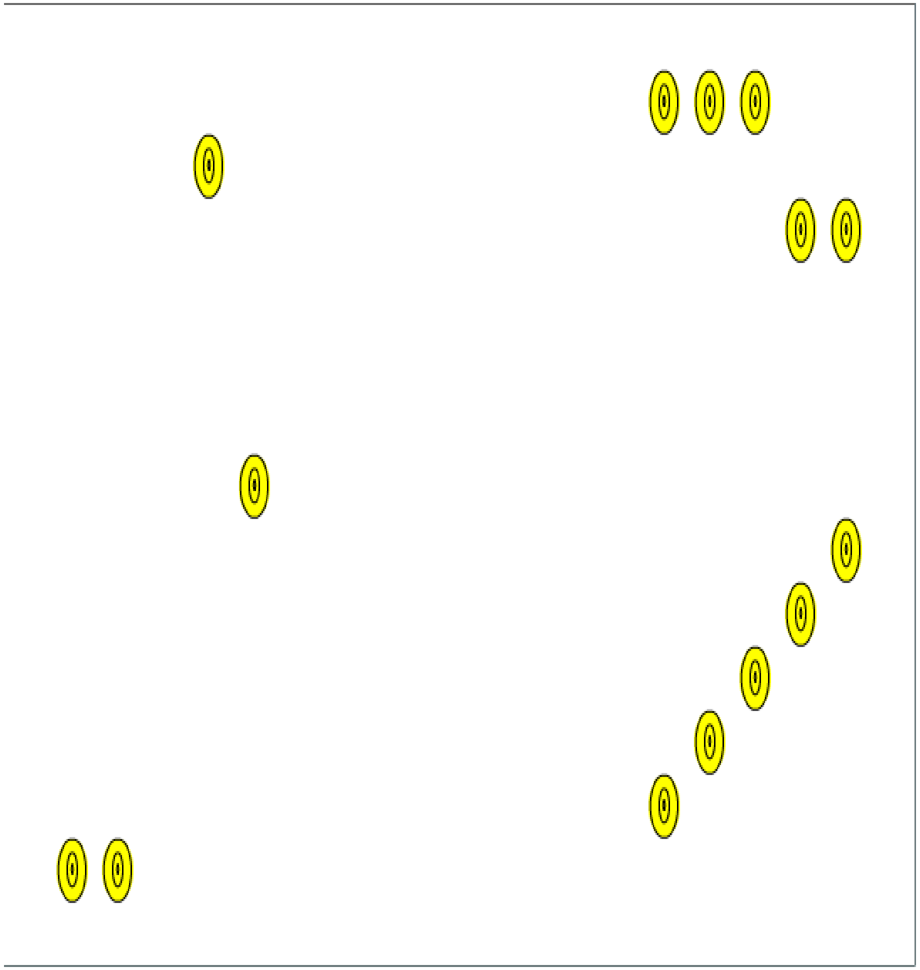
Background Creation



```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1
1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Dynamic Object Creation



```
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4, 4,-1,-1,-1
-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1
-1, 4, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

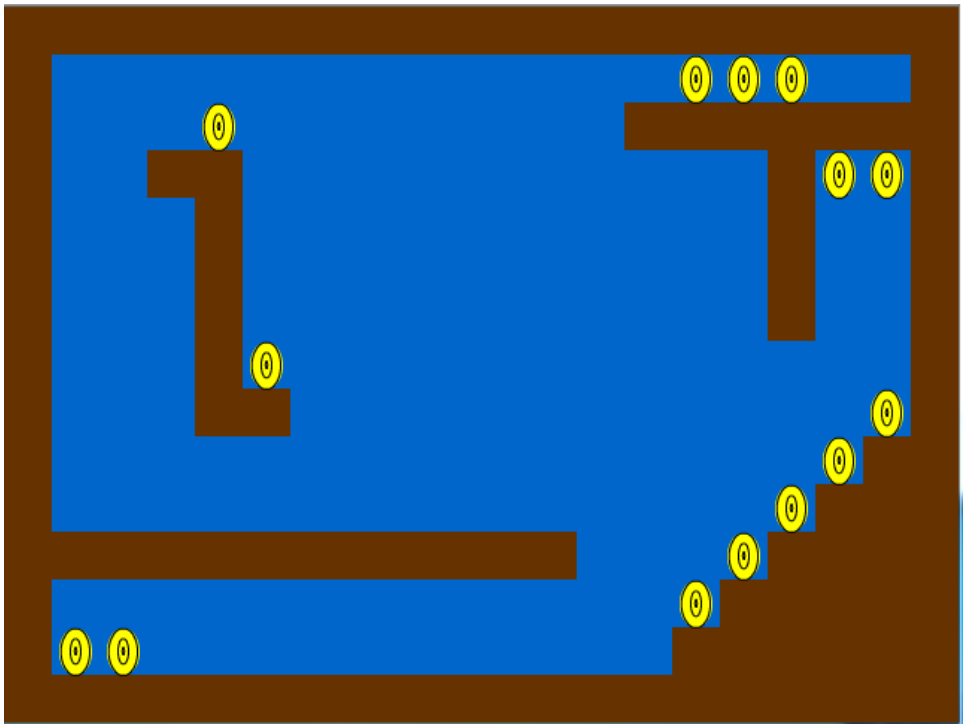
Full Representation

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1
1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

+

```
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4, 4,-1,-1,-1
-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1
-1, 4, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
```

=



Note: Both arrays are usually imported from a level file

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Our Engine

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Background & Sprite Data

Two xml files will be provided:

- one contains all the background information **(BackgroundData.xml)**
- The second contains all the dynamic sprites information **(SpriteData.xml)**

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Background Data XML

```
<Level>

  <Columns> 15 </Columns>
  <Rows> 11 </Rows>
  <TileWidth> 50 </TileWidth>
  <TileHeight> 50 </TileHeight>

  <TileIDs>
    <tile id="0" name="Air" class_name="Air" class_type="displayobject" > </tile>
    <tile id="1" name="Ground1" class_name="Ground1" class_type="displayobject"> </tile>
    <tile id="2" name="Ground2" class_name="Ground2" class_type="displayobject"> </tile>
   </TileIDs>

   <TileMap>
    <Row> 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 </Row>
    <Row> 1,0,0,0,0,0,0,0,0,0,0,0,0,0,1 </Row>
    <Row> 1,0,0,0,0,0,0,0,1,1,1,1,1,1,1 </Row>
    <Row> 1,0,0,1,1,0,0,0,0,0,0,0,0,0,1 </Row>
    <Row> 1,0,0,0,1,0,0,0,0,0,0,0,0,0,1 </Row>
    <Row> 1,0,0,0,1,0,0,0,0,0,0,0,0,0,1 </Row>
    <Row> 1,0,0,0,1,0,0,0,0,0,0,0,0,0,1 </Row>
    <Row> 1,0,0,0,1,0,0,0,0,0,0,0,0,1,1 </Row>
    <Row> 1,0,0,0,1,1,0,0,0,0,0,0,1,1,1 </Row>
    <Row> 1,0,0,0,0,0,0,0,0,0,0,1,1,1,1 </Row>
    <Row> 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 </Row>
  </TileMap>

</Level>
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Sprite Data XML

```
<SpriteData>

  <Columns> 15 </Columns>
  <Rows> 11 </Rows>
  <TileWidth> 50 </TileWidth>
  <TileHeight> 50 </TileHeight>

  <TileIDs>
    <tile id="4" name="Coin" class_name="GamePlay.Level2.CoinSP" class_type="gameobject"> </tile>
    <tile id="5" name="Hero" class_name="GamePlay.Level2.HeroSP" class_type="gameobject"> </tile>
    <tile id="6" name="Door" class_name="Door" class_type="displayobject"> </tile>
  </TileIDs>

  <TileMap>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
    <Row> -1, 6,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4, 4,-1 </Row>
    <Row> -1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1 </Row>
    <Row> -1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1, 4,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1 </Row>
    <Row> -1, 5,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1 </Row>
    <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  </TileMap>

</SpriteData>
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Data in the XML

```
<Columns> 15 </Columns>
<Rows> 11 </Rows>
<TileWidth> 50 </TileWidth>
<TileHeight> 50 </TileHeight>
```

We start by getting how many rows and columns we have in the tile map. This information is needed in order to know how many tiles we have and how we can parse it.

Then we get the tile's dimension since that will be important when we are placing all tiles.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Data in the XML

```
<TileIDs>
  <tile id="4" name="Coin" class_name="GamePlay.Level2.CoinSP" class_type="gameobject"> </tile>
  <tile id="5" name="Hero" class_name="GamePlay.Level2.HeroSP" class_type="gameobject"> </tile>
  <tile id="6" name="Door" class_name="Door" class_type="displayobject"> </tile>
</TileIDs>
```

After that, we get the tiles information. Every tile has the following information:

- **id:** The tile map will be made out of tile ids. Every tile will have a unique id. When parsing the tile map we will know which tile to add from the id.

- **name:** Every game object in our engine has to have a name when added to the object manager. That is why every tile has to have a name which we will use when adding all the tiles to the object manager.

- **class_name:** This is the most important part in the tile's information. Every tile will have a class in our system. That class can be just a shape (display object, sprite, MovieClip, etc…) or it can be a class that we implemented that, of course, extends from GameObject. The class_name will specify the class that we need in order to instantiate one of that specific tile.

- **class_type:** The class type will specify to us if the previously provided class name is a displayobject or a class that we already created that extends from GameObject.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

Data in the XML

```
<TileMap>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  <Row> -1, 6,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4, 4, 4,-1 </Row>
  <Row> -1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1 </Row>
  <Row> -1,-1,-1,-1,-1, 4,-1,-1,-1,-1,-1,-1, 4,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1 </Row>
  <Row> -1, 5,-1,-1,-1,-1,-1,-1,-1,-1, 4,-1,-1,-1,-1 </Row>
  <Row> -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 </Row>
</TileMap>
```

Last but not least is the tile map itself. Every row is specified and contains all the tiles in that row. Every tile is specified by it's corresponding **id**.

PS: -1 means that no tile is found in this location.

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

getDefinitionByName

As it was mentioned before, we will be parsing the xml and loading tiles based on its content. In the xml though, we only have the names of the classes that represent every tile so how can we instantiate a tile at runtime from the class name?

Well, lucky for us, AS3 has a **getDefinitionByName** function that allows us to get a class object from the name of the class. Using that class object we can instantiate as many instances as we want.

Let's see an example!

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

getDefinitionByName

Assuming we have a MovieClip called “Hero”.
The following code will create an instance of Hero by using its name only.

```
var classObject:Class = getDefinitionByName("Hero") as Class;
var hero:MovieClip = new classObject();
hero.x = 100;
hero.y = 100;
stage.addChild(hero);
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

getDefinitionByName

Now let's assume we have the same Hero MovieClip in our engine and we want to add a Hero instance into the ObjectManager:

```
var classObject:Class = getDefinitionByName("Hero") as Class;
var gameObject:GameObject = new GameObject(new classObject(), 100, 100);
ObjectManager.AddObject(gameObject, "Hero", EngineGlobals.OM_STATICOBJECT);
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

getDefinitionByName

Now let's assume we have a HeroMC class that we created (of course it extends from GameObject), it's found in the GamePlay.Level1 package and we want to add a HeroMC instance into the ObjectManager:

```
var classObject:Class = getDefinitionByName("GamePlay.Level1.Hero") as Class;
var goHero:GameObject = new classObject();
goHero.displayobject.x = 100;
goHero.displayobject.y = 100;
ObjectManager.AddObject(goHero, "Hero", EngineGlobals.OM_STATICOBJECT);
```

PS: ***If the HeroMC class hasn't been used anywhere else in the code, the compiler will not be able to find it so the getDefinitionByName will return null. There is a small hack around this. You need to create a HeroMC dummy variable anywhere in your code and by that the compiler will recognize it.***

```
var dummy:HeroMC;
```

Copyright © 2011, DigiPen Institute of Technology. All Rights Reserved

The End ☺