

CMPS 20: Game Design Experience

XNA Game Studio

January 12, 2010
Arnav Jhala



Announcements

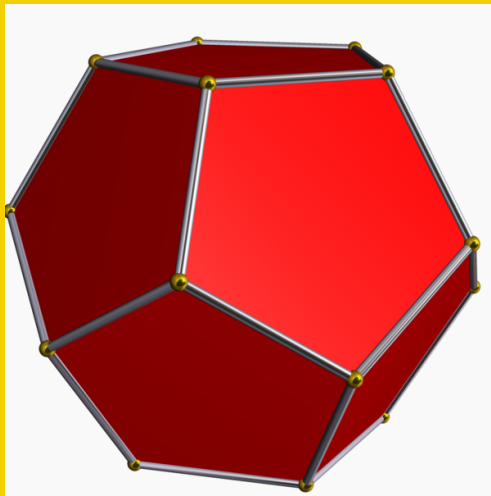
- Session schedules being finalized
- Homework #1 (Hunt the Wumpus)
 - Due Thursday, January 21
 - Detailed instructions will be up by tomorrow
- Project themes and timeline
- Check website frequently for updates
 - Come for help with Homework #1, C#, XNA

Imagine Cup

- Worldwide competition
 - Microsoft sponsored
 - Many categories, including game development
 - Registration Feb 1
 - Multiple rounds – first round ends March 15
 - Games must use XNA Game Studio 3.0
 - Games must address contest theme
 - Would be possible to take your project for CS 20, and enter it into the contest
 - <http://imaginecup.com/>

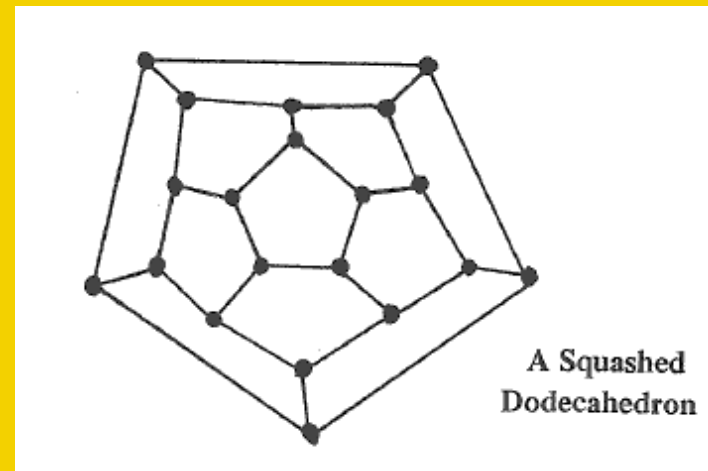
Hunt the Wumpus

- A game where you move through a dodecahedron shaped map
 - Each room is connected to three other rooms
 - Shortest non-repeating path back to same point is five moves



A dodecahedron

Source: Wikipedia



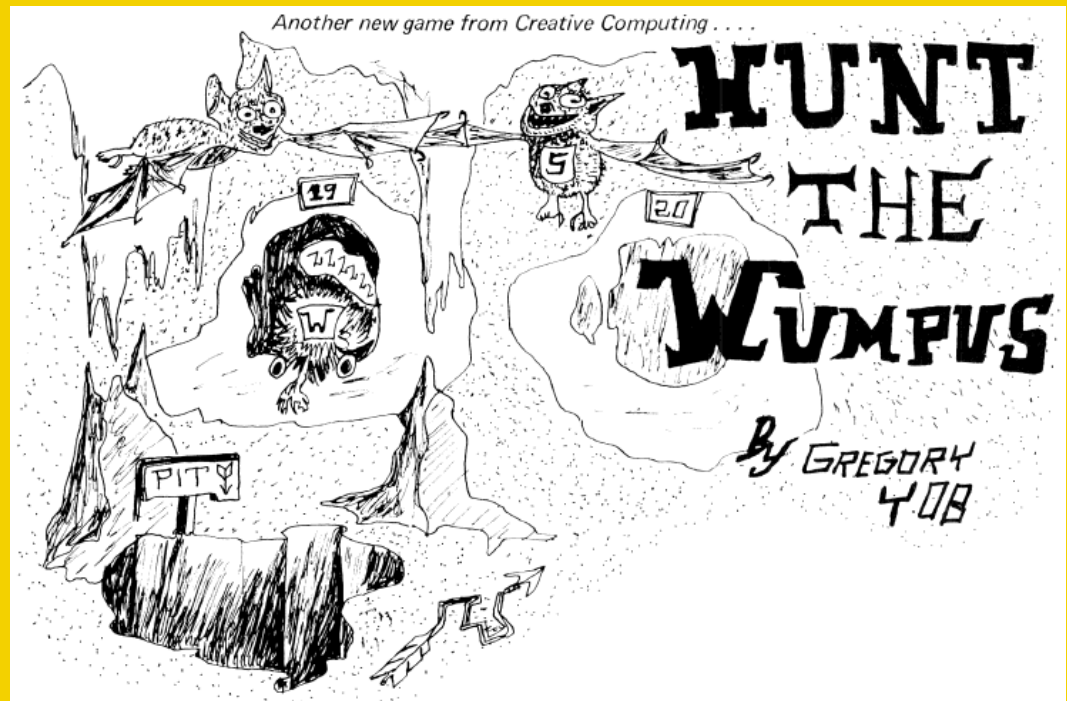
A flattened dodecahedron

Source: More BASIC Computer Games

www.atariarchives.org/morebasicgames/showpage.php?page=178

Hunt the Wumpus (cont'd)

- Turn based game
 - Each turn, player moves or shoots a crooked arrow (up to five rooms)
- Three main types of obstacles
 - Wumpus
 - If awake, kills player if both in same room at end of turn
 - “I smell a Wumpus”
 - Pits (2 instances)
 - Player falls in and dies
 - “I feel a draft”
 - Superbats
 - Places player in a random new room
 - “Bats nearby”



Source: Best of Creative Computing, Vol. 1
www.atariarchives.org/bcc1/showpage.php?page=247

Hunt the Wumpus (cont'd)

- Main challenges to the homework assignment
 - Reading and parsing input
 - How to represent game world
 - How to represent each room
 - How to represent the set of all the rooms
 - You have a fixed number of them (20)
 - How to represent connections in the map
 - Each room has a number, and each room is connected to exactly three other rooms
 - How to represent Wumpus, pits, bats
 - Each is located in the game map
 - Writing game logic
 - Main game loop
 - Display of warnings when player one room away from obstacles
 - Behavior of Wumpus, pits, bats when they interact with player
 - Handling shooting logic

Hunt the Wumpus (cont'd)

- As a rule of thumb, in object oriented design:
 - Nouns are represented as classes
 - What are some of the nouns in Hunt the Wumpus?
 - Verbs are methods on the classes
 - What are some of the actions on the nouns?
 - What can the player do?
 - What can a Wumpus do?
 - What can bats do?
 - What can pits do?
 - Also need to consider what information other classes might need
 - These will be properties
 - Location of Wumpus, bats, pits

Console input in C#

- There are three methods for reading console input in C#
 - ReadLine
 - Read a line of input into a string
 - ReadKey
 - Read the next key pressed into ConsoleKeyInfo instance
 - Provides access to whether ctrl, alt, shift were pressed
 - Read
 - Read a line of input, then gives successive characters each time you call Read
- Any of these could be used for your assignment
 - ReadLine is easiest to use

Console.ReadLine

```
public class ConsoleDemo
{
    static void Main(string[] args)
    {
        string my_input;

        System.Console.WriteLine("(M)ove or  
(S)hoot :");
        my_input = System.Console.ReadLine();
    }
}
```

- ReadLine

- Program execution blocks until user enters a line, terminated by Enter
- Line typed by user is returned as a string

Parsing Input

- In Wumpus, need to check whether player entered
 - M for move, S for shoot, Q for quit
 - That is, check the first character of the user input
 - Would like to be case sensitive
- No problem – use `string.Compare`
- `string.Compare` has 10 variations
 - An overloaded method
 - We want:
 - `public static int Compare(string strA, int indexA, string strB, int indexB, int length, bool ignoreCase);`
 - Compare two strings, **strA** and **strB**
 - ... starting at character number **indexA** in **strA**, and **indexB** in **strB**...
 - ... and comparing **length** characters in each ...
 - ... with the ability to **ignoreCase**.
 - Returns 0 if strings are the same
 - -1 if A lexically smaller than B, 1 if A lexically larger than B

Parsing Input (cont'd)

```
public class ConsoleDemo
{
    static void Main(string[] args)
    {
        string my_input;

        System.Console.WriteLine("(M)ove or (S)hoot :");
        my_input = System.Console.ReadLine();

        if (string.Compare(my_input, 0, "M", 0, 1, true) == 0)
            // Move
        if (string.Compare(my_input, 0, "S", 0, 1, true) == 0)
            // Shoot
        if (string.Compare(my_input, 0, "Q", 0, 1, true) == 0)
            // Quit
        }
    }
}
```

- We want to compare the first character of user input against
 - “M” for move
 - “S” for shoot
 - “Q” for quit

Demonstration of Console input and string parsing in Visual C# 2008

Unified Modeling Language (UML)

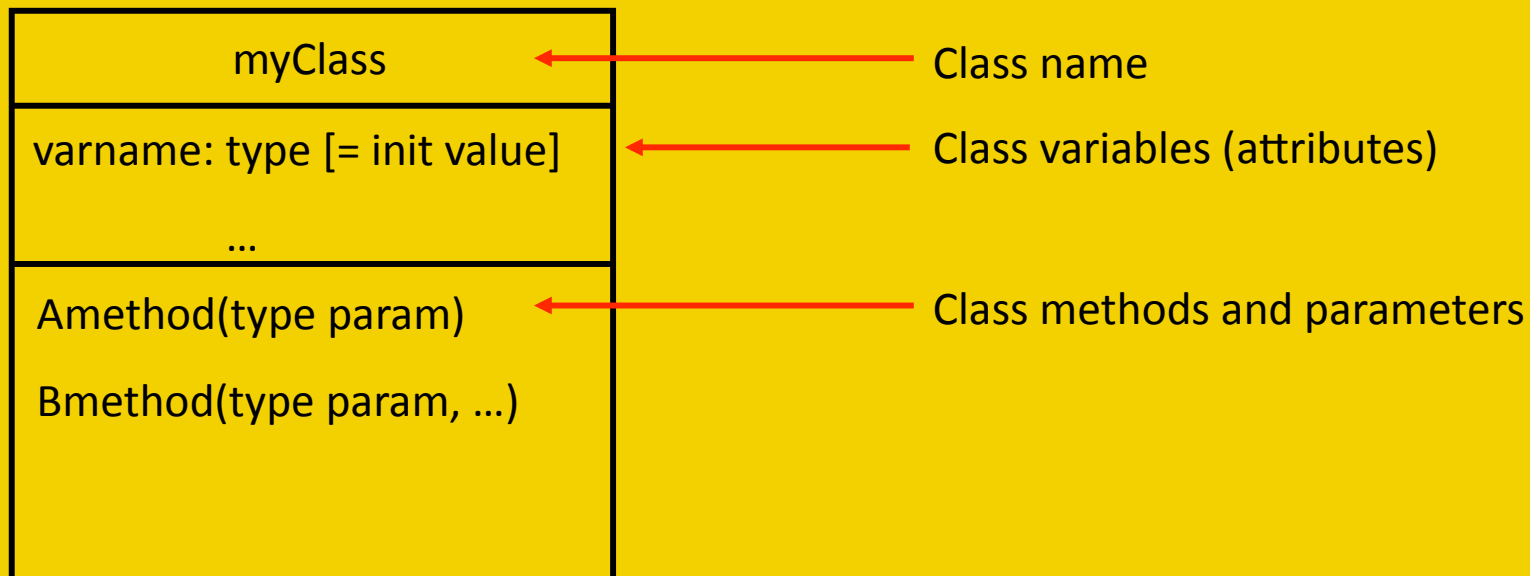
- A family of diagram types used to model object-oriented software projects
 - A standard way to graphically represent complex assemblages of objects, components, etc.
- Two useful diagram types
 - Class diagram
 - Static view of software
 - Object-oriented classes
 - Relationships
 - Inheritance
 - Containment
 - Sequence diagram
 - Dynamic view
 - Methods calling methods, and in what order



Jmgold, Flickr
www.flickr.com/photos/jmgold/2210820262/

Modeling a Class in UML

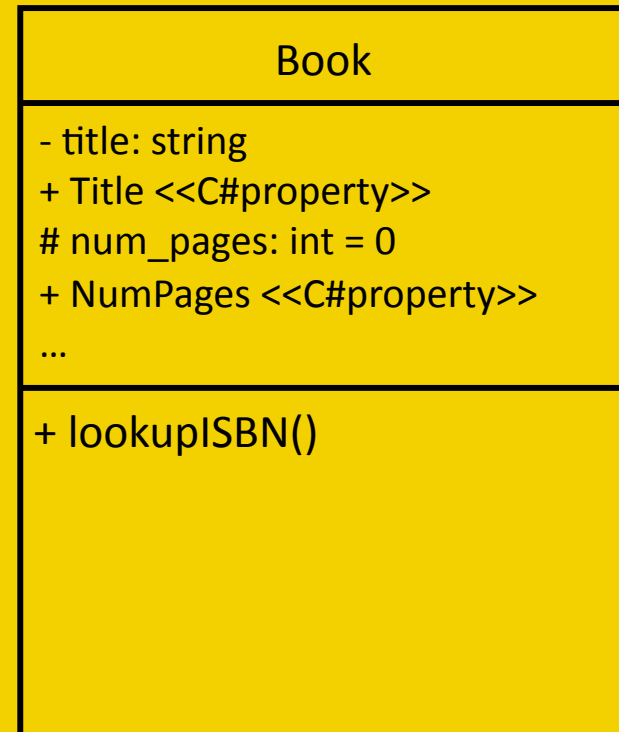
- An object oriented class contains:
 - Data: class variables
 - Type & visibility of each variable
 - Methods
 - Name, parameters, types of parameters
- UML classes can represent all of this





Modeling Visibility

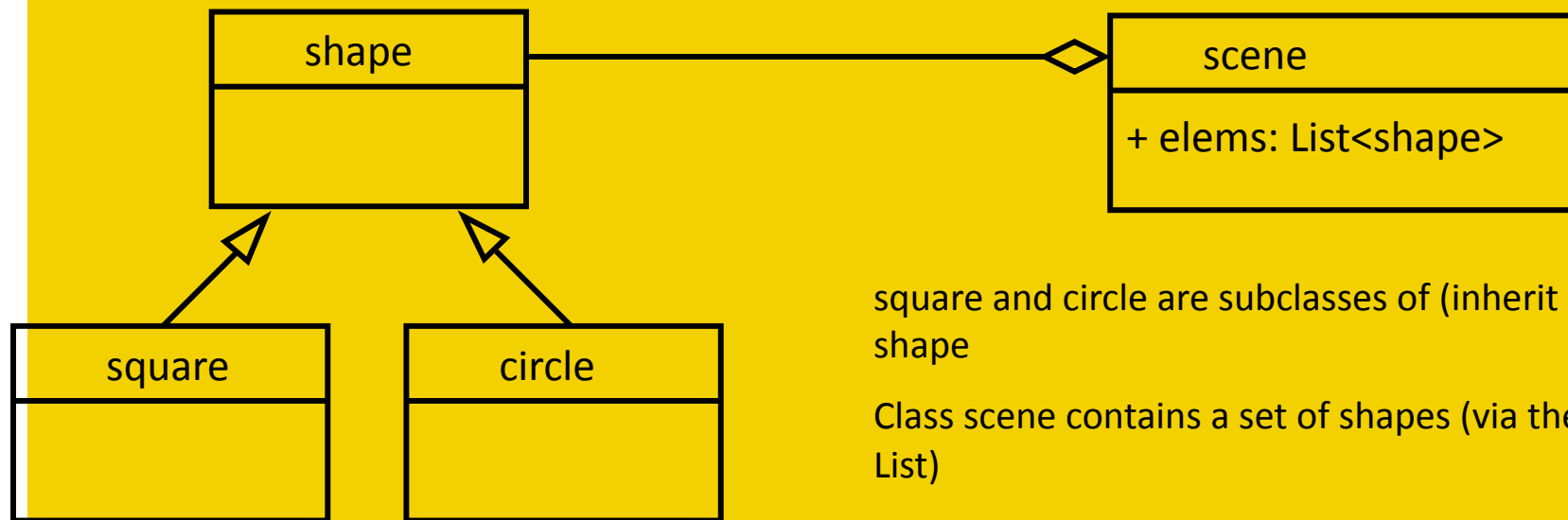
- The visibility of a variable or method can be indicated using:

- + public
- # protected
- private



Inheritance, Containment

- Two main relationships modeled in class diagrams
 - Inheritance (is-a, specialization) 
 - Containment (has-a) 



square and circle are subclasses of (inherit from) shape

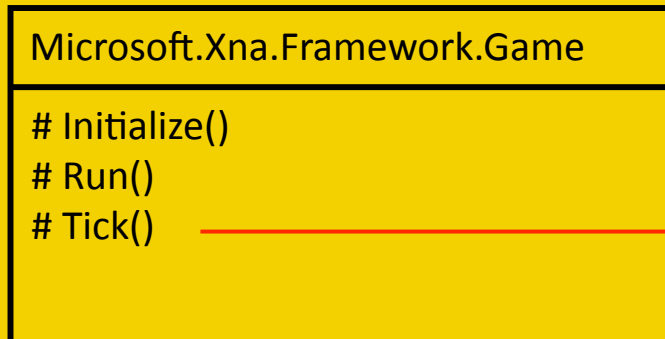
Class scene contains a set of shapes (via the elems List)

Open full triangle arrowhead significant for inheritance (a different arrowhead would not mean the same thing!)

XNA GSE Game Scaffolding

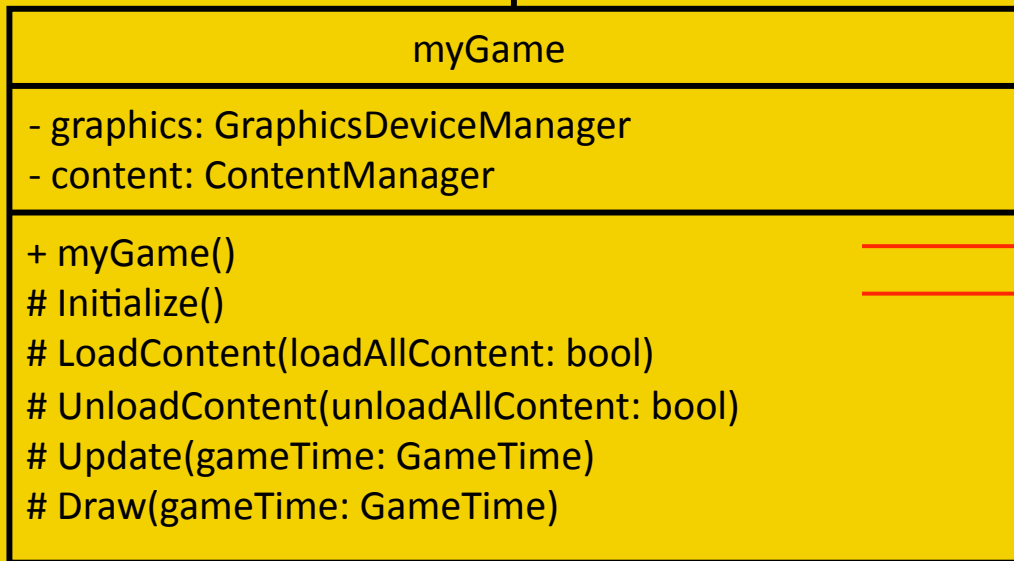
- Scaffolding for a simple XNA GSE game is created when you select a new game project in Visual C# Express
 - File ... New Project ... Windows Game (3.0)
 - Or File ... New Project ... Xbox 360 Game (3.0)
- Can fill-in this scaffolding to create your own game
- Creates a class (*myGameClass*) that includes
 - Constructor
 - Initialization
 - Initialize(), LoadContent()
 - Update
 - Update game state every clock tick
 - Draw
 - Create display every clock tick
- *Demonstration of XNA GSE scaffolding in Visual C# 2008 Express*

XNA GSE Game Scaffolding



```

Update(gameTime);
Draw(gameTime);
    
```



```

graphics = new
GraphicsDeviceManager(this);
Content.RootDirectory = "Content";
base.Initialize()
    
```

XNA GSE Game Initialization

- Create new myGame
 - Call to constructor, myGame()
 - myGame.run()
 1. Initializes game, then,
 2. Runs the main game loop & processes events
- Initialization phase of run(),
 - The following methods are called on myGame
 - Initialize()
 1. call Initialize() on parent class
 2. Initialize your game state
 1. Create player object, create enemy objects, create object to hold main game state, etc.
 - LoadContent()
 - Method used to load textures, create SpriteBatches

XNA GSE Main Game Loop

- Time elapsed between each clock tick:
 - Fixed:
 - $1/60^{\text{th}}$ of a second (16.6667 milliseconds per tick)
 - `myGame.IsFixedTimeStep = true`
 - The default value
 - Variable:
 - Adjusts based on the time required to perform previous tick
 - `myGame.IsFixedTimeStep = false`
- Each clock tick
 - `Run()` calls `Tick()`
 - `Tick()` calls `Update()` then `Draw()`
 - You supply `Update()` and `Draw()`

Update() and Draw()

- Update()
 - Update the state of all objects
 - Receive input, move player avatar
 - Compute opponent AI, move opponent objects
 - Collision detection & consequences
 - Detect end-of-game or end-of-level condition
- Draw()
 - Re-create the on-screen scene using the up-to-date positions of player, opponent
- Advice
 - Avoid stuffing your entire game into the definition of these two methods
 - Methods become too big!
 - Have these methods call out to your player object, opponent objects, etc.
 - `foreach (Opponent o in opponentList) o.update();`

Getting a 2D Image to Appear on Screen

LoadContent()

1. Create a Texture
 - A bitmap image
2. Create a SpriteBatch
 - Collects all textures being drawn to screen

Draw()

3. Begin the SpriteBatch
4. Draw texture
 - Draw() is defined on a SpriteBatch
 - Adds texture to the SpriteBatch
5. End the SpriteBatch
 - Causes textures in SpriteBatch to be drawn to screen

Creating a Texture

- Create an instance of ContentManager
 - XNA GSE scaffolding does this for you
 - Content = new ContentManager(Services) in constructor
- Call Load<T> on ContentManager
 - For 2D sprites, type T is “Texture2D”
 - This loads an art asset that has been created by the Content Pipeline
 - In our case, conversion of a 2D bitmap image in PNG or JPG into XNA internal bitmap format
 - Give the pathname of the bitmap image (e.g., in PNG or JPG) to load
 - Path is relative to the “Content” directory of the Visual C# project
 - Note: normally need to escape slash in a string “\” → \
 - Can put “@” at beginning of string to make string “verbatim”
 - No need to escape slashes in this case
 - “\\images\\” is the same as @“\images\”

Example of creating a texture

- Create new bitmap image
 - In GIMP, Photoshop, etc.
 - Save to disk, then copy over to Visual C# project
 - Copy to
Visual Studio 2008\Projects\{*your project*}\{*your project*}\Content
 - Go to Solution Explorer in Visual C# Express
 - Right click on **Bolded Project Name**
 - Add → Add Existing Item
 - Pick filename of new bitmap image file
 - Will now appear in the project file list
 - Verify that Content Pipeline processed file by building solution (F6)
 - Build > Build Solution
- Create a Texture2D, then load the bitmap image via the content manager:

```
Protected Texture2D m_bullet = null;  
m_bullet = Content.Load<Texture2D>(@"mushi-bullet");
```

SpriteBatch

- Once a texture has been made, how does this get displayed?
 - Create a SpriteBatch
 - Within a clock tick, begin() the batch
 - Prepares the graphics device for drawing sprites
 - Draw() the texture as part of the batch
 - End() the batch
 - Causes textures to be drawn to the screen
 - Restores device to how it was before the batch
 - Typically this is performed in your game's Draw() method

SpriteBatch Example

```
protected override void LoadGraphicsContent(bool loadAllContent)
{
    if (loadAllContent)
    {
        m_batch = new SpriteBatch(graphics.GraphicsDevice); // Initialize the
        sprite batch
        m_bullet = content.Load<Texture2D>(@"mushi-bullet"); // Create Texture2D
    }
}

protected override void Draw(GameTime gameTime)
{
    Vector2 loc = new Vector2(120, 120); // Create Vector2 to give location of
    Texture2D
    m_batch.Begin(); // Start the batch
    m_batch.Draw(m_bullet, loc, Color.White); // Add Texture2D to batch. Not yet
    on screen.
    m_batch.End(); // Now Texture2D is drawn to screen.
}
```

- Draw() inside SpriteBatch is heavily overloaded
 - 7 different choices

Other Sprite features

- Depth ordering
 - Draw some sprites in front of (behind) others to give depth of field effect
- Rotation
 - Can rotate sprite image to a specific angle
- Scaling
 - Can make sprites larger or smaller
- Animated sprites
 - Need to write code that cycles the animation yourself
 - Variant of `batch.Draw()` where you specify a specific rectangle within a `Texture2D` to draw
- Warp effects
 - Deform the `Texture2D` before placing on screen