Session 2.1

The Anatomy of a Game Program

Session Overview

- Show how the program code of an XNA game is stored
- See how a game draws itself on the screen
- Create a variable that can represent a color in an XNA game
- Create a “universal color display” based on the color variable

Looking Inside A Game Program

- A good game is made up of a number of items
  - The images
  - The sounds
  - The program code
- When you make a game, you are not just writing the code. You need to think about all the other things, too
  - Although, for this course we are going to focus on the program that makes everything else work

The Microsoft Visual Studio Solution

- The Visual Studio tool will manage the creation of our game program
- It provides a view called the “Solution Explorer” that lets us look at the components of our game
- This is the view of a brand new empty game

DEMONSTRATION

1. Finding the Draw Method

- The Draw method is used by XNA to draw the display
- This shows where the method can be found

The Draw Method

- The actual drawing action for a game is performed by a method called Draw
- A method is part of a program that we can identify by name
- The XNA designers thought Draw was a good name to give the method that draws the screen
- The Draw method is part of XNA
- The Draw method is written in the C# programming language
Anatomy of a Method

- A method is a block of code with a particular name
- The name of the method is selected by the programmer and reflects what it actually does
- The block of code contains C# statements that perform the steps that the method does
- This is called the body of the method

The Draw Method Header

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- The method header contains the method name (in this case Draw)
- It also contains other information we can ignore for now

The Draw Method Body

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- The body of a method is a block of C#
- Whenever the game display needs to be drawn it is the instructions in the body of Draw that will do the job

The C# Language as a Language

- Humans can use the English language to tell each other what to do:
  “Make me a cup of coffee.”
- We don’t use English to tell computers what to do, we use a programming language such as C#
- The English language has things like full stops, sentences, and paragraphs
- We now need to learn how C# has the equivalent things

Blocks of code

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- A block of code is the statements between the open curly bracket { and the closed curly bracket }
- They are a bit like paragraphs in English

Statements

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- A statement describes a single operation
- Statements are separated by the ; character
- They are a bit like a sentence in English
Chapter 2.1: The Anatomy of a Game Program

Comments

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- This is a simple comment
- The // marks the line as not being a statement to be executed, but a comment for the programmer to read

Calling the Clear Method

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- Clear is a method that is called by a statement in the Draw method
- It is supplied with a parameter which tells it what to do
- Clear is provided by the Graphics Device

Telling Clear What to Do by Using Parameters

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- The Clear method needs to be told what color to clear the graphics device
- In programs you tell methods what to do by giving them parameters
- In this case the parameter is a Color value

Calling a Method

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
}
```

- When a method is called, the program goes off and performs the statements in that method
- It then returns to the caller and continues running from there

Calling Multiple Methods

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

- This Draw method contains two calls of Clear
  - Do you think it will work?
  - What do you think it will do?
  - Is it legal C#?

2. Drawing Different Colors

- The Clear method can be supplied with different color values to display different colors

DEMOnstration
Making Our Own Colors

- The built-in colors are all very well, but there are only a limited number
- We are making a light that can display any possible color
- We have to find out how XNA holds color, and then write some C# statements that will create our own color values

The Color Data Type

- Whenever a program wants to use a type of data, it needs to have a way of representing it
- The XNA designers had to invent a way to hold Color values in games
- They invented a new data type called Color
- C# provides built-in types for holding some things but it is designed to be extensible, and the XNA designers used this ability to make a Color type
- You will create your own types later in this course

Declaring a Color Variable in C#

```
Color backgroundColor;
```

- When you need to store a value you must create a variable to put it in
- A variable is a location which can hold something your program is going to use
- You pick the name of the variable when you write the program
- The variable in this program is called backgroundColor

Variables and Type

```
Color backgroundColor;
```

- When you create a C# variable you first give the type of data that the variable will hold
- This variable is of the type Color
- This means that it can be used to store color values and nothing else
- C# is very fussy about this. It makes sure that you can only use variables in an appropriate way
- This is called being typesafe

Variables and Identifiers

```
Color backgroundColor;
```

- The name of the variable is used to identify it
- It is called the identifier of the variable
- C# has rules about what is a legal identifier:
  - It must start with a letter and only contain letters, digits, and the _ character
  - Upper and lower case are different characters
- There is another, unwritten, rule:
  - The identifier must describe what the variable holds

Setting a Color Value

```
backgroundColor = new Color(0,0,0);
```

- This is an assignment statement
- It assigns the value of a new color to the variable backgroundColor
- The new color value is made up of three numbers
- Each of these numbers represents the intensity of the red, green, and blue components of the color being created
The Assignment Operator

```
backgroundColor = new Color(0,0,0);
```

- In this statement, the = character is doing an operation for the program
- It is moving (assigning) the value on the right into the variable on the left
- This means that = is called the assignment operator (I call it the gozzinta)
- Do not confuse this operation with any form of comparison

A color value is held in a variable of type Color. Each of these numbers represents the intensity of the red, green, and blue components of the color being created.

A New Draw Method

```
protected override void Draw(GameTime gameTime)
{
    Color backgroundColor;
    backgroundColor = new Color(0, 0, 0);
    GraphicsDevice.Clear(backgroundColor);
    base.Draw(gameTime);
}
```

- This version of Draw creates a new color value and sets it to red=0, green=0, blue=0
- It then uses this value to clear the screen

This method contains a call of the Fill method. By changing the values in the Color that is constructed we can create any color.

### DEMONSTRATION

3. Creating Our Own Colors

- We can use this program to find out what a color with no red, green, or blue looks like

SUMMARY

- An XNA game program contains a method called Draw that draws the display
- A method is a block of code with a particular name
- Blocks of code are made up of a sequence of statements
- The Draw method calls a method called Clear
- The Clear method gives the color to fill the display
- A color value is held in a variable of type Color
- A Color value can be constructed with intensity values of red, green, and blue, and fed into the Clear method

True/False Revision Quiz

- The Draw method is part of the C# language.
- The Draw method contains a call of the Fill method.
- The body of a method is a block of C# statements.
- Statements are terminated by the ; character.
- The equals character is used by C# as an assignment operator.
- C# contains a special type to hold Color values.
- A Color is made up of the intensity of green, orange, and teal.
Chapter 2.1: The Anatomy of a Game Program

True/False Revision Quiz

- The Draw method is part of the C# language.
- The Draw method contains a call of the Fill method.
- The body of a method is a block of C# statements.
- Statements are terminated by the ; character.
- The equals character is used by C# as an assignment operator.
- C# contains a special type to hold Color values.
- A Color is made up of the intensity of green, orange, and teal.

True/False Revision Quiz

- The Draw method is part of the C# language.
- The Draw method contains a call of the Fill method.
- The body of a method is a block of C# statements.
- Statements are terminated by the ; character.
- The equals character is used by C# as an assignment operator.
- C# contains a special type to hold Color values.
- A Color is made up of the intensity of green, orange, and teal.
Session 2.2

Storing Data in a Game Program

Session Overview
- See how a class is made up of methods and data
- Show how games are held in C# classes
- Discover how the Draw and Update methods in a game share Game World data
- Find out the difference between local variables in blocks and member variables in classes
- Use operators to update color values when the game runs
- See how programs can “overflow” data variables

Games and Classes
- In C#, programs are made up of classes
- A class is a collection of behaviors (methods) and data (variables)
- It looks after some part of a system
- The XNA designers designed a class called Game to look after a game
- We are going to create our games based on this class

The XNA Game Class
- A class brings together methods and data
- Classes can be very large
- Their name reflects what they look after
  - Game is the name for the class which XNA uses to represent a game program
  - Visual Studio creates a “starter” Game class when we make a new Game project
- We make our game by filling this class in

Drawing a Fixed Color
```csharp
protected override void Draw(GameTime gameTime)
{
    Color backgroundColor;
    backgroundColor = new Color(50, 70, 70);
    GraphicsDevice.Clear(backgroundColor);
    base.Draw(gameTime);
}
```
- At present the Draw method draws a set color
- We would like to make a game where the color that is drawn changes over time
- To do this we have to look at how games work

How Games Work: 1
- The Update method changes the game data each time the clock ticks
- Draw
  - Game Data

Game1 class
All data items are actually stored in a class called Game1.cs. This class contains data (variables) and behaviors (methods). The game data, and behaviors (methods) are all held in a C# construction called a class.

### Making a Mood Light Game Class

```csharp
class Game1
{
    byte redIntensity;
    byte blueIntensity;
    byte greenIntensity;

    protected override void Draw(GameTime gameTime)
    {
    }

    protected override void Update(GameTime gameTime)
    {
    }
}
```

### Game World Data for the Mood Light

- The game data, Draw and Update methods are all held in a C# construction called a class.
- A class is a fundamental unit of a C# program.
- It contains data (variables) and behaviors (methods).
- The class that XNA creates for a new game is held in a file called Game1.cs.

- The Game World for our Mood Light will be the amount of red, blue, and green in the color it is displaying.
- We can store these amounts in variables of type byte, since a Color can be constructed from three byte values.
- They are declared like any other variable, with the name of the type followed by the identifier.

### Storing Data Values in Memory

- When a program runs the variables it uses are mapped onto particular memory locations.
- The run time system for the program takes care of where the data is actually stored.
- All data items are actually stored in this way.

- The Run Time System for the program takes care of where the data is actually stored.
- All data items are actually stored in this way.

- This has been happening in our games already, we have just not seen it.
- If Update changes the data each time it is called, the game can give the appearance of movement.
- To make a game, you need to fill in the Update and Draw methods.

- The XNA Framework does this for us automatically.
Using the Intensity Values in the Draw Method

```csharp
using System;

class Game1
{
    byte redIntensity;
    byte blueIntensity;
    byte greenIntensity;

    protected override void Draw(GameTime gameTime)
    {
        Color backgroundColor = new Color(redIntensity, blueIntensity, greenIntensity);
        GraphicsDevice.Clear(backgroundColor);
    }
}
```

### Classes as Offices

- You can think of a class as an office
- It contains people to perform the methods and a desk with items on it to hold the member data
- It also contains phones to call a method

### Local Variables

- We create a local variable called `backgroundColor` inside the `Update` method
- This is used to hold the color to clear the screen
- This is local to the `Update` method, and can't be used anywhere else

### Updating the Colors

- At the moment the screen is black
  - This is because the starting values of `redIntensity`, `blueIntensity`, and `greenIntensity` are all 0
- We want the screen to get progressively brighter over time
- We can do this by adding statements to the `Update` method which increase the intensity values

### Empty Update Method

```csharp
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    this.Exit();
    // TODO: Add your Update logic here
    base.Update(gameTime);
}
```

- Visual Studio creates an empty `Update` method
- We need to add `Update` statements to make the lights get brighter

### Demonstration

1. MoodLight Draw Behavior

- This version of the MoodLight uses game world variables to hold the color to be displayed
- Note that in C# a byte data member is set to 0 when it is created

### Updating the Colors

- At the moment the screen is black
  - This is because the starting values of `redIntensity`, `blueIntensity`, and `greenIntensity` are all 0
- We want the screen to get progressively brighter over time
- We can do this by adding statements to the `Update` method which increase the intensity values

### Empty Update Method

```csharp
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    this.Exit();
    // TODO: Add your Update logic here
    base.Update(gameTime);
}
```
 protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
        this.Exit();
    // TODO: Add your Update logic here
    base.Update(gameTime);
}

• The first part of the Update method tests for the Back button being pressed on Gamepad 1
• If this button is pressed the game exits

Increasing the Intensity Values
 protected override void Update(GameTime gameTime)
{
    // Increase the intensity values
    redIntensity++;
    blueIntensity++;
    greenIntensity++;
    base.Update(gameTime);
}

• This version of Update increases the intensity of red, green, and blue
• ++ is an operator that increases the value of a variable by 1

Overflow and Data Storage
• Each data item is stored in memory in a location of a particular size
• The byte type is stored in a single 8 bit location
• This can only hold values from 0 to 255
• When the program tries to put 256 into it, the value wraps round to 0

Variable Overflow
• Overflow occurs because a programmer uses a particular data type incorrectly
• Note that the program itself did not display an error when this happened
• This is the worst kind of runtime error
• Of course the computer does not “care” that it has done something wrong
• You have to program the computer to care about things like this

Summary
• An XNA game is actually a C# class that holds Game World data, a Draw and an Update method
• Game World data is made up of variables which are declared as members of the class and can be used by all the methods in it
• The Draw method draws the world, and the Update method updates it sixty times per second
• C# programs can also use variables declared in particular blocks and only used within them
• Variables are mapped onto memory locations and are of finite size
True/False Revision Quiz

- An XNA game is held in a C# class.
- In XNA, the Update method is always called before the Draw method.
- A data member is similar to an office.
- The Game World is stored in local variables.
- The byte data type is held in 8 bits of memory.
- A program will stop automatically when a variable overflows.

In XNA the Update method is always called before the Draw method.
A data member is similar to an office.
The Game World is stored in local variables.
The byte data type is held in 8 bits of memory.
A program will stop automatically when a variable overflows.
True/False Revision Quiz

- An XNA game is held in a C# class.
- In XNA the Update method is always called before the Draw method.
- A data member is similar to an office.
- The Game World is stored in local variables.
- The byte data type is held in 8 bits of memory.
- A program will stop automatically when a variable value overflows.

Session Overview

- Introduce the idea of an algorithm
- Show how a program can make logical decisions based on the value of data in variables
- Investigate how to create sensible logical constructions and how the compiler can help with this
- Create a proper MoodLight which fades smoothly between black and white
- Change the MoodLight to display a range of colors

DEMONSTRATION

1. Overflow MoodLight
   - This is the broken version of the fading MoodLight
   - The light suddenly changes from white to black when the byte variables overflow

Making a Fading MoodLight

- The problem with the fading MoodLight is that we have not told the program that we want to count down as well as up
- Every single action that we want the program to perform must be specifically described, otherwise it will not perform the action
- The computer program doesn’t actually “care” what it does, it simply follows the steps that it is given

How To Make the Lights Fade

- We can express what we want to do in English:
  “Make the value of redIntensity bigger each time that you are called. When the value reaches 255, start making it smaller each time you are called until it reaches 0, at which point you should start making it bigger again. Do the same with blue and green.”
- This would allow an intelligent human to control the lights, now we need to create some C# that will allow a computer to do this
Problems and Algorithms

- A sequence of steps that solves a problem is called an algorithm.
- You can think of a recipe as something that solves a problem “How do I bake this cake?”
- The algorithm is the starting point of the program.
- Once you have the algorithm you then convert it into a program that behaves like the algorithm says it should.
- The algorithm gives a sequence of steps to solve the problem.

```csharp
if (true) redIntensity++;
```

Storing Data for the Algorithm

```csharp
// The Game World - our color values
byte redIntensity = 0;
bool redCountingUp = true;
```

- To implement the algorithm, the program needs to hold whether or not it is counting up.
- The C# language provides a type called bool (short for Boolean) which only holds the values true and false.
- Variables of type bool are declared just like the other types.

Making Decisions in Programs

```csharp
if (true) redIntensity++;
```

- The value true can be used directly in a program if you wish.
- However, this is not always useful since in the above code the intensity value will always be increased, as the condition is true.
- Note that this is not invalid C#, the compiler will not give you an error, even though the test might as well not be there.

Detecting Stupid Decisions

```csharp
if (false) redIntensity++;
```

- If the program uses the value false in a condition it “switches off” the statement that follows.
- The compiler will notice this and give a warning, since it can tell that there is a statement which will never be obeyed.
- You should always heed warnings from the compiler, as they often indicate that the program might not be correct.

```
if (true) redIntensity++;
```

- A conditional statement lets a program perform an action if a given condition is true.
- The condition must be a Boolean value, so we can use the value of a Boolean variable directly in a test.
- In the above code the intensity is only increased if the value redCountingUp is true.

```
if (true) redIntensity++;
```

- The computer program will run as a series of steps, so we need to break our instructions down into steps as well:
  1. If we are counting up, increase the value of redIntensity.
  2. If we are counting down, decrease the value of redIntensity.
  3. If redIntensity is 255, change to counting down.
  4. If redIntensity is 0, change to counting up.

```
if (false) redIntensity++;
```

- To implement the algorithm, the program needs to hold whether or not it is counting up.
- The C# language provides a type called bool (short for Boolean) which only holds the values true and false.
- Variables of type bool are declared just like the other types.

```
if (false) redIntensity++;
```

- A conditional statement lets a program perform an action if a given condition is true.
- The condition must be a Boolean value, so we can use the value of a Boolean variable directly in a test.
- In the above code the intensity is only increased if the value redCountingUp is true.
2. Conditional Red Light

- This version of MoodLight just uses the color red
- It can be controlled by a condition as we have just seen

Adding an else Part to a Conditional Statement

```csharp
if (redCountingUp) redIntensity++; else redIntensity--;
```

- An if construction can select between two statements
  - One is performed if the condition is true
  - The other is performed if the condition is false
- The above statement will perform the first two steps of the algorithm we are implementing
  - The operator `--` will reduce the value in a variable by 1

Tests in Conditions

- A condition can be a test, rather than a Boolean value
- A condition works out to either true or false
- The `==` operator in the condition above is performing a test for equality

Fun with Conditions

- Some of these conditions are stupid
- Some are not legal C# because they combine things in an incorrect way
- One of them is sensible, if not the best way to do the job
- Can you tell which is which?

```
if (false == true) redIntensity++;  // Orange lines
if (redCountingUp == 1) redIntensity++;  // Red lines
if (false == 0) redIntensity++;  // Red lines
if (redCountingUp == false) redIntensity--;  // Green lines
```

Completing the Algorithm

- This is all the steps of the algorithm converted into C#
- It would allow us to create a MoodLight that fades smoothly from black to red and back again
- This code would go into the Update method
- The Draw method would not need to change

```
if (redCountingUp) redIntensity++; else redIntensity--;
if (redIntensity == 255) redCountingUp = false;
if (redIntensity == 0) redCountingUp = true;
```
3. Fading Red Light

- This version of MoodLight just uses the color red
- It can be controlled by a condition as we have just seen

Statements and Blocks

```
if (redCountingUp)
{
    redIntensity++;
}
else
{
    redIntensity--;
}
```

- You can put a block of statements after a condition rather than a single statement
- This lets you use a single condition to select a large amount of code

Customer Question

- At this point the customer comes in with a request for our red MoodLight
  - Customers always want to change what you have made!
  - The customer wants the MoodLight to start off bright and get darker, rather than start off dark and get brighter
  - Do you have any idea how we could achieve this?

Starting with a Bright Display

```
// Game World
byte redIntensity = 255;
bool redCountingUp = false;
```

- These initial settings should make the program start with the light at full brightness (255) and then make the light get dimmer as the intensity value is reduced on each Update
- This is the direct reverse of the original starting condition

4. Start Bright Fading Light

- This version of MoodLight starts bright and then fades down
- The algorithm is just the same, but it starts from different values
C# provides a Boolean data type called Boolean.

```
bool redCountingUp = false;
byte redIntensity = 0;
```

### Spot the Bug

```csharp
// Game World
byte redIntensity = 0;
bool redCountingUp = false;
```

- These initial conditions will cause the program to do something that you might not want
- The programmer intended to make the screen get steadily brighter, but when the program runs this does not happen
- Run the program manually to see why

### Working Through the Code

```csharp
if (redCountingUp) redIntensity++; else redIntensity--;
```

- In the first statement, because redCountingUp is false, the program will try to subtract 1 from redIntensity
- This means it will subtract 1 from 0
- The byte type does not support negative values, and so it underflows to the value 255
  - This is like overflow, but heading in the other direction

### Fixing the Problem

```csharp
if (redIntensity == 0) redCountingUp = true;
if (redIntensity == 255) redCountingUp = false;
if (redCountingUp == false)
```

- The simplest way to fix this bug is to change the order of the tests and the updates
- This means that the count direction is always set correctly before the update is performed
- Remember that the order statements are performed can have a great effect on what they end up doing

### Summary

- The starting point of any program has to be an algorithm that solves the problem that the program is being written to solve
- C# provides a Boolean data type called bool that can be either true or false and be used as the basis of conditions in decisions
- The C# if construction allows a program to selectively execute program statements
- Conditional operators allow a program to compare items and make decisions on their values

### True/False Revision Quiz

- An algorithm is like a recipe.
- The C# bool type holds the values 0 and 1.
- A condition in a C# program must evaluate to true or false.
- An if construction in C# must always have an else part.
- The C# compiler will produce warnings if it thinks a program might not be correct.
- The = operator compares two values for equality.
True/False Revision Quiz

- An algorithm is like a recipe.
- The C# bool type holds the values 0 and 1.
- A condition in a C# program must evaluate to true or false.
- An if construction in C# must always have an else part.
- The C# compiler will produce warnings if it thinks a program might not be correct.
- The = operator compares two values for equality.

True/False Revision Quiz

- An algorithm is like a recipe.
- The C# bool type holds the values 0 and 1.
- A condition in a C# program must evaluate to true or false.
- An if construction in C# must always have an else part.
- The C# compiler will produce warnings if it thinks a program might not be correct.
- The = operator compares two values for equality.

True/False Revision Quiz

- An algorithm is like a recipe.
- The C# bool type holds the values 0 and 1.
- A condition in a C# program must evaluate to true or false.
- An if construction in C# must always have an else part.
- The C# compiler will produce warnings if it thinks a program might not be correct.
- The = operator compares two values for equality.