Session 9.1

Using the Keyboard in XNA
Session Overview

- Discover more detail on how the XNA keyboard is implemented
- Find out how to use arrays to find out which keys have been pressed on the keyboard
- See how custom type has been created in XNA to represent key presses and how to use these in your programs
- Start to create a message board program
Game Idea: “Message Board”

- We know how to create good looking text
- We can also make animated backgrounds and display a clock
- We can combine all these abilities to create a message board
- First we need to read the keyboard
Reading the Keyboard

- We have already used the keyboard in our XNA programs
- We used it to make a keyboard version of “Color Nerve”
- If the R key is pressed the red intensity value will be increased

```csharp
{
    redIntensity++;
}
```
Levels and Edges

```csharp
    keys.IsKeyDown(Keys.R))
{
    redIntensity++;
}
```

- This kind of input is managed by *level*
- Every time we test the key we will find that it is down
- If we just tested for keys in this way, we would have an “auto repeat” that worked 60 times a second
Detecting that a Key Has Been Pressed

```csharp
if (keyState.IsKeyDown(Keys.R) &&
    oldKeyState.IsKeyUp(Keys.R))
{
    // if we get here the key R has just been pressed
}
```

- To detect a key being pressed we have to do what we did before with the Button Bash game
- We have to compare the current state of the key with the previous state
- A changes in the two states will mean that a key has been pressed or released
Detecting All the Keys

- To detect all the keys on a keyboard we would have to repeat this process for every single key.
- This is possible, but would be very tedious.
- Fortunately XNA provides an easier way of finding out what keys have been pressed, using arrays.
Finding Out the State of the Keyboard

```csharp
KeyboardState keyState = Keyboard.GetState();

Keys[] pressedKeys;

pressedKeys = keyState.GetPressedKeys();
```

- To get hold of the state of the keyboard we can use the `GetState` method.
- This returns a value of type `KeyboardState` which describes the state of the keyboard at the time of the call.
Discovering Which Keys Are Currently Pressed

```csharp
KeyboardState keyState = Keyboard.GetState();

Keys[] pressedKeys;
pressedKeys = keyState.GetPressedKeys();
```

- The `GetPressedKeys` method can be called on a `KeyboardState` value.
- The method returns an array which holds a list of keys that are pressed down.
- In the code above, the array `pressedKeys` is set to the value returned by `GetPressedKeys`.
Creating a String of Key Names

```csharp
messageString = "";
for (int i = 0; i < pressedKeys.Length; i++)
{
    messageString = messageString +
    pressedKeys[i].ToString() + " " ;
}
```

- This **for** loop assembles a list of the keys that are presently pressed and puts their descriptions into a string variable called `messageString`.
- The string can be displayed by the **Draw** method.
- It does this by working through the array of pressed keys.
Creating an Empty String

```csharp
messageString = "";
for (int i = 0; i < pressedKeys.Length; i++)
{
    messageString = messageString +
    pressedKeys[i].ToString() + " " ;
}
```

- The `messageString` will hold a string of text that describes each of the keys that has been pressed.
- It must be cleared to an empty string at the start.
- This assignment sets `messageString` to an empty string.
Creating a for Loop to Work Down the Array

```csharp
messageString = "";
for (int i = 0; i < pressedKeys.Length; i++)
{
    messageString = messageString +
        pressedKeys[i].ToString() + " ";
}
```

- The `for` loop needs to know the size of the `pressedKeys` array
- All arrays provide a `Length` property which returns an integer giving the number of elements in the array
- This is used to control the `for` loop
Assembling the Message String

```csharp
messageString = "";
for (int i = 0; i < pressedKeys.Length; i++)
{
    messageString = messageString + pressedKeys[i].ToString() + " ";
}
```

- We have used the `+` operator between numbers to perform addition.
- When `+` is used between strings it actually puts one string on the end of the other.
- This statement adds a key description onto the end of the string, along with a separator space.
Getting the Key Description

```csharp
messageString = "";
for (int i = 0; i < pressedKeys.Length; i++)
{
    messageString = messageString +
    pressedKeys[i].ToString() + " " ;
}
```

- You can ask any object in a program to provide a string description of itself
- You do this by calling the `ToString` method on that object
- We are asking each key to describe itself
Displaying the Message

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin();
    spriteBatch.DrawString(font, messageString,
                           messageVector, Color.White);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

- The **Draw** method just displays the string
- `messageString` and `messageVector` are variables in the game world
1. Key Viewer

- This program displays a list of keys that are pressed

- Note that each key is uniquely identified, including the left and right shift keys
Representing Keys in XNA

- The designers of XNA needed something to represent keys on the keyboard.
- Some of the keys were printable characters, others were control keys such as shift and escape.
- They wanted users of the keyboard to be able to work with any key value that could be produced.
- To solve this problem they created an enumerated type called `Keys`.
Enumerated Types and the Keys Type

- The word “enumerate” means *to count*
- In C# an enumerated type is one that can hold a particular number of values
- In the case of the keyboard, the XNA designers wanted to hold values for every available key, but no more than that
- They therefore created an enumerated type called Keys
Why Make an Enumerated Type?

- There is no need to use an enumerated type
- The XNA designers could just have assigned some values for the keys:
  - 0 means left shift
  - 1 means right shift
  - 2 means the A key
- However programmers could then use meaningless values, which would do strange things
- An attempt to use key number 1001 might break the program
Creating Your Own Enumerated Types

- Later in the course you will create your own enumerated types
- You can regard these as variable types where you get to choose the values which are allowed
- They are used a lot in situations where items must be in one of a number of different states:
  - Bank account: open, suspended, closed
  - Video game: attract mode, playing, showing high score
  - Person: happy, sad, angry, upset
Enumerated Types and Intellisense

- Since Microsoft Visual Studio can find out what values an enumerated type can take, it can show you these automatically.
- This is an example of the way that the development tool can work with the language that you are using.
Summary

- The keyboard is used in a very similar way to the gamepad, with a `GetState` method returning an object that describes the state of the keyboard.

- Your code can ask for the state of an individual key, or it can ask for an array that provides a list of all the keys that are pressed down.

- To detect a key being pressed an XNA game must use edge detection.

- An enumerated type has a specific set of values.