

# Coding for Game Development

## Section Three

### XNA Game Programming

#### Game Input Devices

1/18

# Keyboard Input

- Keyboard input for games uses the simple concept that *holding down* a particular key operates some controller within the game.
- For example, a sprite might move upwards while the up arrow key is pressed and stop when the key is released.
- Within the **Update()** method, we check which keys are depressed and adjust the sprite position accordingly.
- Note: Using the keyboard to enter text is not at all easy!

2/18

# Keyboard State

- Within the **Update()** method, we obtain the keyboard state by calling:  
KeyboardState kbState = Keyboard.GetState();
- This records which keys are depressed at this particular moment.
- We can then test individual keys like this:

```
if (kbState.IsKeyDown(Keys.Up)) spriteY--;  
if (kbState.IsKeyDown(Keys.Left)) spriteX--;  
if (kbState.IsKeyDown(Keys.Down)) spriteY++;  
if (kbState.IsKeyDown(Keys.Right)) spriteX++;
```

3/18

# Key Names

- Here are some useful key names:
  - Alphabetic: Keys.A to Keys.Z
  - Numeric Pad: Keys.NumPad0 to Keys.NumPad9
  - Function Keys: Keys.F1 to Keys.F12
  - Shift Keys: Keys.LeftShift Keys.RightShift Keys.CapsLock
  - Ctrl Keys: Keys.LeftControl Keys.RightControl
  - Others: Keys.Space Keys.Escape Keys.Up Keys.Down Keys.Left Keys.Right
- For a full list see: <http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.input.keys.aspx>

4/18

# KeyboardState Methods

- **KeyboardState** has the following useful methods:
  - kbState.IsKeyDown(.. key..)
  - kbState.IsKeyUp(.. key ..)
  - Keys [] keys = kbState.GetPressedKeys ();
- The last method takes a snap shot of keys currently pressed and puts them into an array. This can be processed using a **foreach** loop:

```
foreach (Keys key in keys)  
{  
    : : : :  
}
```

5/18

# Processing Keys in a Loop

```
KeyboardState kbState = Keyboard.GetState();  
Keys [] keys = kbState.GetPressedKeys ();  
foreach (Keys key in keys)  
{  
    switch (key)  
    {  
        case Keys.Up:  
            spriteY--;  
            break;  
  
        case Keys.Down:  
            spriteY++;  
            break;  
  
        default:  
            break;  
    }  
}
```

More cases here

6/18

## Detecting a Change of State

- With a missile firing game we often want to fire just one missile for each key press. To be able to do this we need to detect when a key has changed state. At each call of **Update()** we need to save the keyboard state for next time:

```
private KeyboardState oldKbState; // Must be a field
: : : :
void Update()
{
    KeyboardState kbState = Keyboard.GetState();
    if (oldKbState == null) oldKbState = kbState;
    : : : :
    if (kbState.IsKeyDown(Keys.A) && oldKbState.IsKeyUp(Keys.A))
    {
        ... code to fire missile ...
    }
    oldKbState = kbState;
}
```

key is *down* now but *up* previously

7/18

## XBox 360 Controller

- XNA supports up to 4 controllers/game pads connected to an Xbox or PC system.
- Each controller has 14 digital controls and 4 analogue controls and optional vibration feedback.



8/18

## GamePad State

- We obtain the state of the game pad in a similar way to obtaining the keyboard state. Note that we need to specify which game pad we want to read.

```
GamePadState padState =
    GamePad.GetState(PlayerIndex.One); // One to Four
```

- We can then test individual buttons like this:  
if (padState.Buttons.A == ButtonState.Pressed) ...  
if (padState.Buttons.B == ButtonState.Pressed) ...
- Directional pad buttons are checked like this:  
if (padState.DPad.Down == ButtonState.Pressed) ...  
if (padState.DPad.Up == ButtonState.Pressed) ...
- See button names on next slide...

9/18

## Game Pad Button Names

```
padState.Buttons.A           padState.DPad.Up
padState.Buttons.B           padState.DPad.Down
padState.Buttons.X           padState.DPad.Left
padState.Buttons.Y           padState.DPad.Right

padState.Buttons.Back
padState.Buttons.Start
padState.Buttons.LeftStick
padState.Buttons.RightStick

padState.Buttons.LeftShoulder ("Left Bumper")
padState.Buttons.RightShoulder ("Right Bumper")
```

10/18

## Thumbsticks and Triggers

- The Xbox 360 controller has two analogue thumbstick and two analogue trigger controllers.
- A thumbstick position can be obtained as a **Vector2**:  
Vector2 leftThumb = padState.ThumbSticks.Left;  
Vector2 rightThumb = padState.ThumbSticks.Right;
- Each trigger value can be obtained as a **float** or **double**:  
double leftTrigger = padState.Triggers.Left;  
double rightTrigger = padState.Triggers.Right;

11/18

## Vibration Feedback

- We can specify the amount of vibration feedback sent to a controller:  
GamePad.SetVibration ( PlayerIndex playerIndex,  
float leftMotor, // Low frequency amount 0 .. 1.0  
float rightMotor) // High frequency amount 0 .. 1.0
- For example:  
GamePad.SetVibration (PlayerIndex.One, 0.5f, 1.0f);

12/18

## Disconnected Controller

- The wired version of the XBox 360 controller has an easily disconnected link to prevent physical damage to the console or computer in the event of a sudden tug on the connecting wire. This means that the controller may easily become disconnected during game play.
- We should regularly check the status of the game pad to ensure that it has not become disconnected:

```
bool connected = padState.IsConnected();
```

- We should ignore all inputs from a game pad while it is disconnected but allow them when the game pad is reconnected.

13/18

## Mouse Input

- Mouse input is not available on the Xbox 360, but can be used in games running on a standard PC.
- A typical PC mouse has three buttons, a scroll wheel and an X-Y position. We can read all of these values and use them to control a game.

14/18

## Mouse State and Position

- We can obtain the mouse state as follows:

```
MouseState mState = Mouse.GetState();
```

- We can find the X-Y position like this:

```
int mouseX = mState.X;  
int mouseY = mState.Y;
```

15/18

## Mouse Buttons

- We can find if a button is currently pressed or released:

```
if (mState.LeftButton == ButtonState.Pressed) ..  
if (mState.MiddleButton == ButtonState.Pressed) ..  
if (mState.RightButton == ButtonState.Released) ..
```

- We need to be careful with these methods, since they do not behave in the same way as a typical program.
- To simulate normal mouse behaviour, we have to detect a change of state from Pressed to Released.

```
if (mState.LeftButton == ButtonState.Released &&  
    oldMState.LeftButton == ButtonState.Pressed) ..  
oldMState = mState; // Save state for next time
```

16/18

## Scroll Wheel

- Note. The scroll wheel provides an *accumulative* value since the start of the game.

```
int scrollWheel = mState.ScrollWheelValue;
```

- To check if the scroll wheel value has changed since the last call to **Update()**, we need to compare the position in the current state against the previously stored state.

```
int diff = mState.ScrollWheelValue -  
oldMState.ScrollWheelValue;
```

17/18

18/18