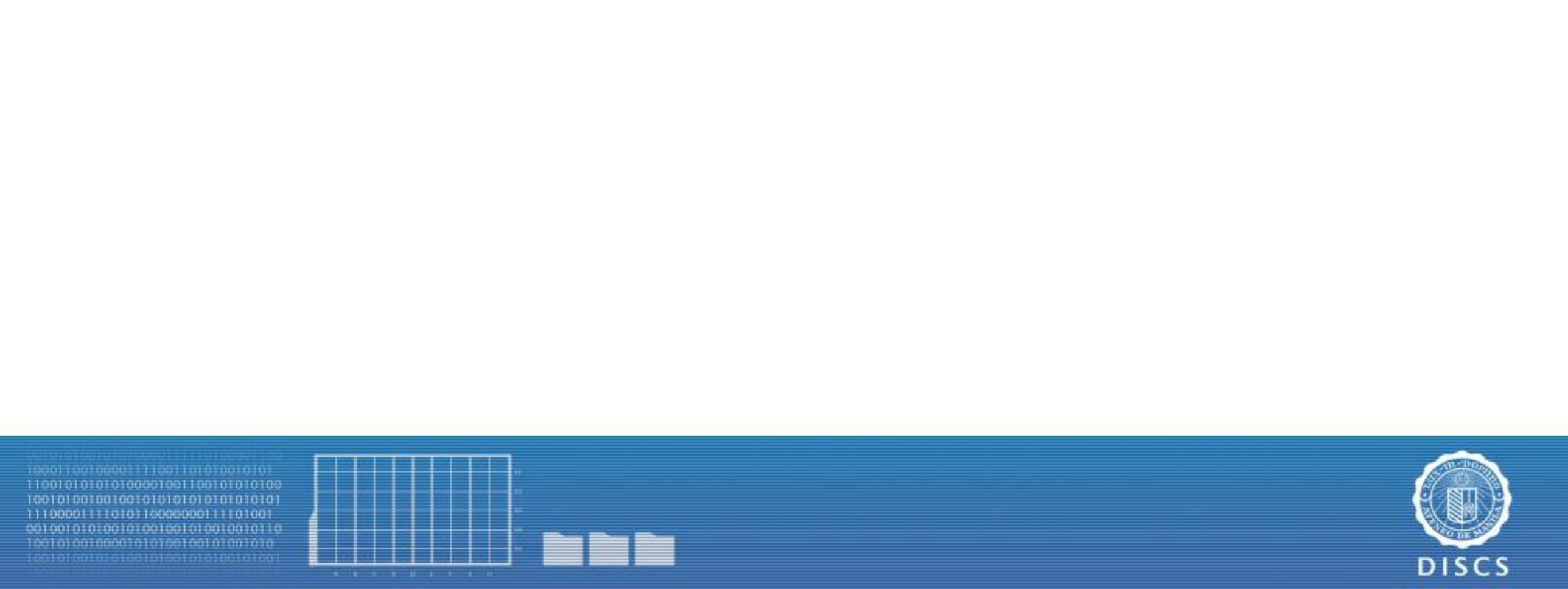

DEPARTMENT OF INFORMATION SYSTEMS AND COMPUTER SCIENCE

# Player Input

## In SFML

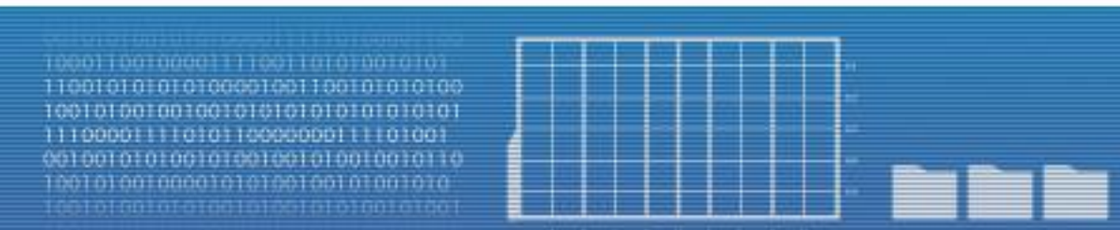
# Lecture Time!

- Keyboard: Keys
- Mouse: Buttons and Position



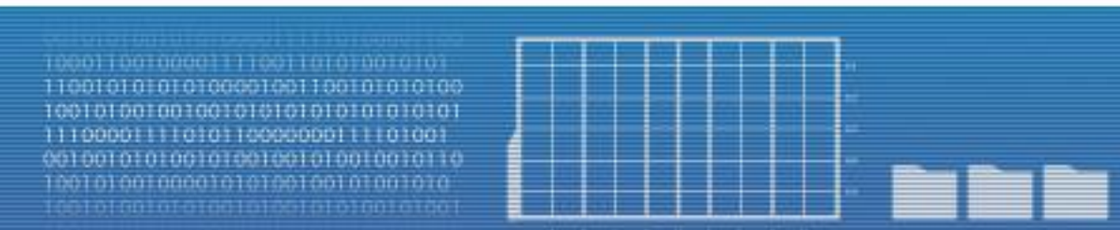
# SFML Keyboard Input

```
// optional initialization (for keybinding)
sf::Keyboard::Key keyUp = sf::Keyboard::W;
sf::Keyboard::Key keyDown = sf::Keyboard::S;
sf::Keyboard::Key keyLeft = sf::Keyboard::A;
sf::Keyboard::Key keyRight = sf::Keyboard::D;
sf::Keyboard::Key keyQuit = sf::Keyboard::Escape

// additional code / a workaround is required
// to make this work in a switch-case statement
```



# SFML Keyboard Input

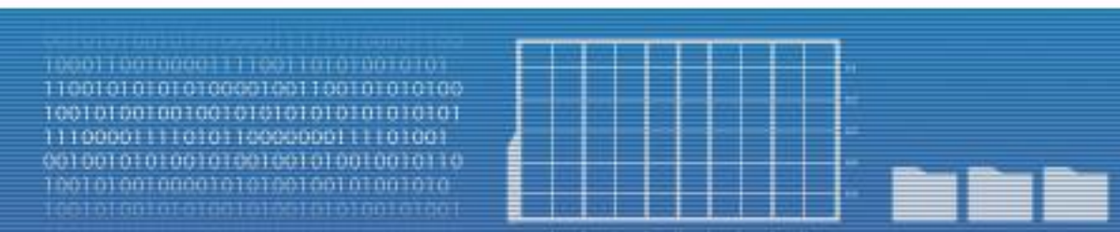
```
// in window.isOpen() loop
if( sf::Keyboard::isKeyPressed( keyQuit ) )
{
      window.close();
}
if( sf::Keyboard::isKeyPressed( keyUp ) )
{
      // ...
}
// ...
```



DISCS

# SFML Keyboard Input

- Note that the `isKeyPressed()` function will return `true` as long as the key is held down
- You will have to provide your own programming logic if you only want the "first" `true` return value to trigger something
  - Hint: Need another `bool`



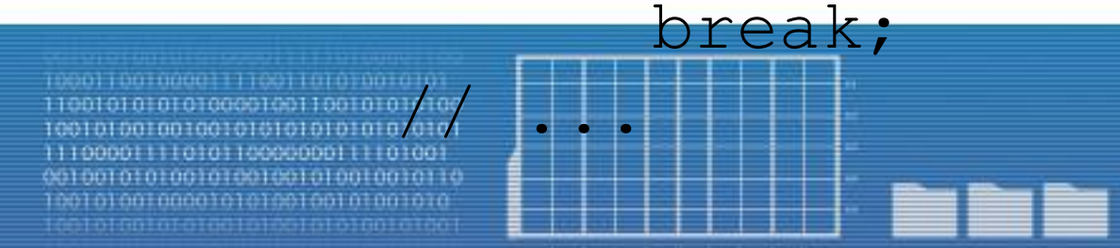




# SFML Keyboard Input

- You may also consider using events

```
while( window.pollEvent( event ) )
{
        switch( event.type )
        {
        // ...
        case sf::Event::KeyPressed:
                switch( event.key.code )
                {
                case sf::Keyboard::Escape:
                        window.close();
                        break;
                // more cases here for the other keys
                }
                break;
        //  ...
```



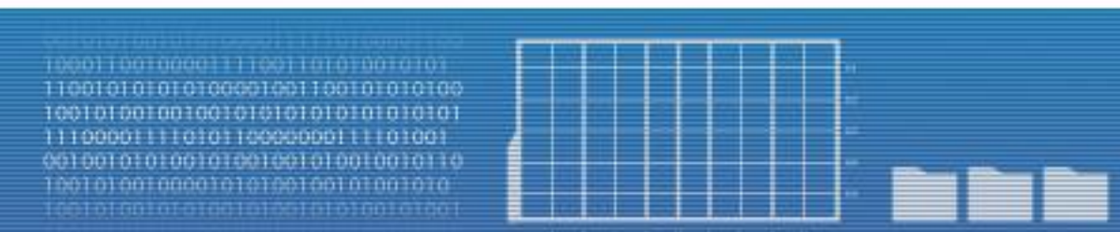
# SFML Keyboard Input

```
// event loop, continued
        // ...
        case sf::Event::KeyReleased:
                // similar to sf::Event::KeyPressed
                break;
        // ...
```

# SFML Keyboard Input

- Note that multiple `KeyPressed` events will be generated at a rate dependent on your OS settings if a key is held down
- Can be disabled in the initialization part of your code (force a maximum of 1 `KeyPressed` event until key is released)

```
// additional initialization
window.setKeyRepeatEnabled(false);
```







# SFML Keyboard Input

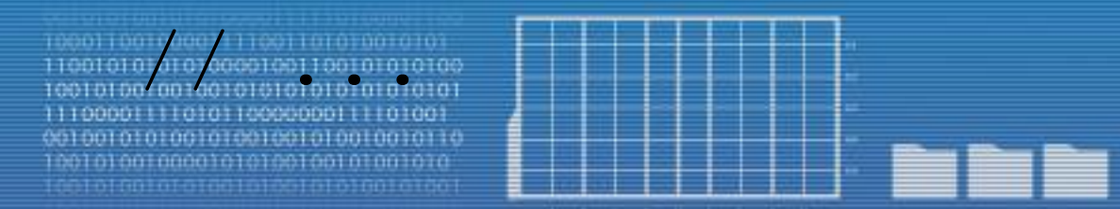
- The event system is similar to Java's
  - In other words, don't put large blocks of code in it
- For `KeyPressed` events, simply flag that the relevant key/s have been pressed (with a bit or a `bool` per key – set to true)
- For `KeyReleased` events, simply flag that the relevant key/s have been released (set to false)

# SFML Keyboard Input

- Then, in the relevant AI part of your code, simply check those flags to see what needs to be run
  - While W/A/S/D flags are true, move the player character
  - If Esc flag is true, terminate the program or open up a menu
  - Reset flag/s to false for keys that are meant to be pressed repeatedly and not held down

# SFML Mouse Input

```
// in window.isOpen() loop
// note: vector returned uses
// same coordinate system as Shape positions
// note#2: window can be omitted
// to get position relative to desktop instead

sf::Vector2i mPos =
      sf::Mouse::getPosition( window );

if( sf::Mouse::isButtonPressed(
      sf::Mouse::Left ) )

{

      // ...

}

// ...
```



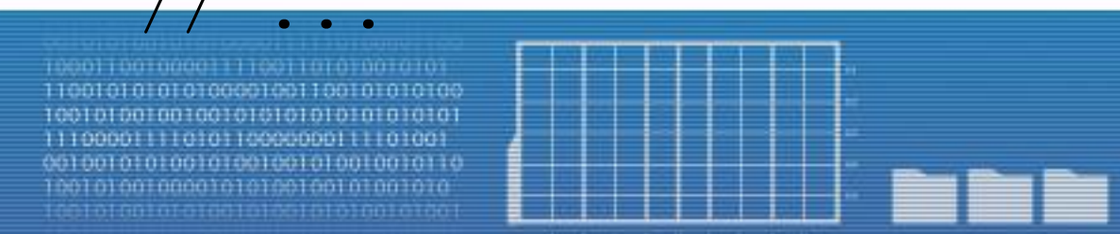


# SFML Mouse Input

- Again, the `isButtonPressed()` function will return `true` as long as the mouse button is held down
- You may also consider using events
  - Mouse wheel input can only be handled using events

# SFML Mouse Input

```
// additional cases to event loop's switch statement
case sf::Event::MouseButtonPressed:
        switch( event.mouseButton.button )
        {
        case sf::Mouse::Left:
                // set flag to true
                break;
        case sf::Mouse::Right:
                // set flag to true
                break;
        }
        break;
case sf::Event::MouseButtonReleased:
        // same, but set flags to false
        break;
// ...
```



# SFML Mouse Input

```
// additional cases to event loop's switch statement
// ...
case sf::Event::MouseMoved:
      // set mouse delta values
      // based on mouse position
      // from previous frame
      break;
// ...
```

# Exercise

- Draw 3 RectangleShapes, each with a different color (red, green, blue)
  - Clicking on one of them should "select" it and change its color to white
    - This should also deselect any RectangleShape that was selected earlier and revert it to its original color
    - Clicking on an empty space should also cause this deselect action
  - Use the keyboard (WASD keys) to move the selected RectangleShape

# Homework

- Modify your previous homework:
  - The first element in your CircleShape array should no longer drift downwards
  - It should now respond to WASD keys and move in the corresponding direction at a rate of 200 pixels per second
  - Speed should be adjusted accordingly if diagonal movement is detected

# Homework

- Modify your previous homework (continued):
  - The first element in your RectangleShape array should no longer drift to the right
  - It should now move directly towards the cursor at a rate of 200 pixels per second while the left mouse button is held down
  - Again, speed should be adjusted accordingly if movement is along both horizontal and vertical axes

DISCS