# Lecture8: Input

ETGG3802

# Some 1801 ideas

- **Device-polling**
  - Asking questions like these:
    - Is this button down?
    - What's the gamepad left-x axis value?
  - Typically you do this every frame
  - Good for:
    - Character movement
    - Moving a cursor
- Event-handling
  - A notification system that tells you when an input event first happens
    - A button is pressed
    - A button is released
    - A gamepad axis changes value
  - Good for:
    - Triggers (making attack animation start)
    - Menu navigation

# Some Unity / Unreal additions

- **Action**
  - Could be bound to
    - A single keyboard, mouse, gamepad button
  - E.g. spacebar and “A” gamepad button tied to a “Jump” action
- **Axis**
  - Could be bound to
    - Pairs of keyboard keys (e.g. “A” and “D” bound to a horizontal axis)
    - A gamepad axis (e.g. x-axis of left-analog)
- Advantages:
  - Abstraction: Gameplay designer need not know what buttons are bound to an action / axis
  - Flexibility: Player / GameDesigner can remap the bindings (possibly at run-time)

# Observer Design Pattern

- A Common Software Engineering Pattern
- Two+ entities involved:
  - The **Observed** (InputManager for us)
  - One of more **Observers** (GameObject [or a Component])
- Listener "interface class"

```
class ObserverInterface
{
    void handle_event(int data) {}
};
```

  - If you have an existing class that you want to make an observer, inherit from this
    - Our Application class inherits from OgreBites::InputListener and defines a keyPressed method
  - If you only need one type of thing to be notified, you could add this handle_event-type method there
    - I had a new ComponentInputListener component that I added and put this code there.

# Observer Design Pattern, cont.

- The Observer maintains a variable / list of listeners

```
class TheObserved
{
Std::vector<ObserverInterface*> mListeners;
    void register_listener(ObserverInterface* oi) { mListeners.push_back(oi); }
    void deregister_listener(ObserverInterface* oi) { mListeners.erase(oi); }
    void update()
    {
        if (the_event_happened())
        {
            for (ObserverInterface* i : mListeners)
                i->handle_event(data);
        }
    }
};
```

# Explore the blackboard resources

- Minimal_embeddable_zip.zip
- Updated_invader_media.zip
  - Look at the scene file (esp. script references)
  - Ship.py and invader.py
  - Init.py
- SDL_name_mappings.h
- Input_manager.h
  - You can deviate from this if you have a strong reason, but this gives you an idea what it should look like