Making Your Game Interactive

At this point we've learned about key game development ideas like *animation frames*, the *game loop*, and using *block transfer (blit)* to create the graphics for a frame. You also learned how to use variables along with the game loop in order to turn static images into dynamic animations. Of course, we can't really call what we've made a game because it lacks *interaction*.

Perhaps the most fundamental form of interaction in games is giving the player the ability to control a character through keyboard, mouse, or controller inputs. We'll focus solely on keyboard input for our game. In games, and many other interactive software systems, keyboard input is not handled through functions like python's `input` function. Instead they are handed through system *events*. In order to let the player control their character, we need to understand how pygame represents events, how we can interact with them, and then

Events, Event-Driven Programming, and PyGame

In *event-driven programming* your code does not determine the exact moment certain things happen but rather how to react, or *handle* key *events*. For example, when you use the function `input`, you're causing a user input event to occur at that very moment. In an event-driven system, you don't cause the event but you do write code to handle it.

Libraries like PyGame provide robust event detection and collection mechanisms so that programmers can focus predominately on the logic for handling the event. In particular, the event module in Pygame provides an event data type as well as functions for accessing the events detected and collected by the system. Let's start by understanding event data.

Pygame supports a standard array of event types as seen in this table taken from the PyGame event module documentation.

```
QUIT             none
ACTIVEEVENT      gain, state
KEYDOWN          key, mod, unicode, scancode
KEYUP            key, mod, unicode, scancode
MOUSEMOTION      pos, rel, buttons, touch
MOUSEBUTTONUP    pos, button, touch
MOUSEBUTTONDOWN  pos, button, touch
JOYAXISMOTION    joy (deprecated), instance_id, axis, value
JOYBALLMOTION    joy (deprecated), instance_id, ball, rel
JOYHATMOTION     joy (deprecated), instance_id, hat, value
JOYBUTTONUP      joy (deprecated), instance_id, button
JOYBUTTONDOWN    joy (deprecated), instance_id, button
VIDEORESIZE      size, w, h
VIDEOEXPOSE      none
USEREVENT        code
```

Our game loops had the following block of code at the top. You were told it handled quitting the game when the user presses the X in the window. Let's now take a closer look:

```
#  go through each event that has occurred since the last frame
for event in pygame.event.get():
        # if the current event is the QUIT event
        if event.type == QUIT:
            # then quit the game and exit the program entirely
            pygame.quit()
            sys.exit()
```

The first thing we see is a new kind of loop: `for event in pygame.event.get()`. The function call `pygame.event.get()` returns a collection of all the events in the *event queue*. This queue contains detected but unhandled events

Basic Keyboard Events

Player Motion

Your Player is a Rectangle and Not an Image

Types of Player movement

1. Fixed Grid-like steps
2. Smooth Gliding - Zero acceleration.
3. Gliding with Acceleration
4. Physical (Newtonian) Movement

Types of Movement Environments

1. Fixed space walls (min/max to clamp position changes into a range (pygame.math.clamp))
2. Wrap-around spaces (modular arithmetic)

Simple Stepping

Key Press increments/decrements location

Gliding and Sliding

Key Press increments/decrements velocity. Key release zeros velocity. Velocity changes position.

Acceleration, Jumping and Physical Movement

Key Press accelerates the player. Key release stops acceleration. Friction decelerates and stops the player. Acceleration changes velocity. Velocity changes position.

Jumping is the same but with a gravity instead of friction.