topics:
• event handling

references:
• https://sites.google.com/site/blendergameprojects/, by Prof Tim Hickey, Brandeis University (http://www.cs.brandeis.edu/~tim)

• events can be caused by the user in an interactive environment
• events can also be caused by activity in a virtual world
• an event handler is the code that gets executed when an event is detected
• recall that when we learned about events in JavaScript, we had to do two things:
  – define an event handler
    in JavaScript, this is a function definition like this:
    function FUNCTION_NAME ( ... ) { ... }
  – associate the event handler with the object in the interface/environment, so that the
    event handler is triggered when the registered event occurs
    in JavaScript, this is a call like this:
    OBJECT.addEventListener( EVENT, FUNCTION_NAME, ... )
• today we’ll talk about handling events in Blender

blender game engine

• the Game Engine aspect of Blender includes the Physics simulator (that we discussed
  in the last class) and a Logic Editor that lets us define how the game behaves
• four steps to creating a game:
  1. create visual elements (3D models or images, could be rendered objects)
  2. use logic editor to create behaviors for the game
     can define how user interacts with objects
     and how objects interact with each other
     and how objects interact with the environment
  3. create camera(s) from which to render the scene
  4. launch the game (e.g., create a runtime version)
• components of game logic
  – logic “bricks”
  – properties (like variables)
  – states (an object property)

game engine elements

(1) game logic (2) blender game (3) game menu (4) logic editor panel (5) properties
logic editor

- elements of game logic are defined in Blender as "logic bricks"
- three types of bricks:
  - sensors — listen for events
  - controllers — handle data provided by events
  - actuators — perform actions in response to events

(1) game property area (2) object name (3) links
(4) sensor area (5) controller area (6) actuator area

sensors

- sensors trigger game logic to be activated
- produce output when an event occurs
- sample events: user presses a key or two objects collide
- output is a "pulse" that gets sent to controller(s) connected to the sensor
- a sensor brick looks like this:

- advanced sensor options:
  - "true" level trigger — causes the sensor to fire "true" pulses, to the attached controller(s)
  - "false" level trigger — causes the sensor to fire "false" pulses, to the attached controller(s)
  - repeating trigger — repeats pulse, based on defined frequency, in units of 60 Hz (every 60th of a second)

- types of sensors:
  - Actuator: Detects when a particular actuator receives an activation pulse
  - Always: Gives a continuous output signal at regular intervals
  - Collision: Detects collisions between objects or materials
  - Delay: Delays output by a specified number of logic ticks
  - Joystick: Detects movement of specified joystick controls
  - Keyboard: Detects keyboard input
  - Message: Detects either text messages or property values
  - Mouse: Detects mouse events
  - Near: Detects objects that move to within a specific distance of themselves
  - Property: Detects changes in the properties of its owner object
  - Radar: Detects objects that move to within a specific distance of themselves, within an angle from an axis
  - Random: Generates random pulses
  - Ray: Shoots a ray in the direction of an axis and detects hits
  - Touch: Detects when the object is in contact with another object
• sample keyboard sensor

<table>
<thead>
<tr>
<th>Key</th>
<th>Modifiers</th>
<th>Log</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• controllers
  - collect data received by sensors
  - also can operate based on a specific state
  - these are like conditional statements in Java
  - a controller brick looks like this:

- types of controllers:
  
  - AND
  - OR
  - XOR
  - NAND
  - NOR
  - XNOR
  - Expression
  - Python

- controller responses based on logic operations on sensor pulse inputs:

<table>
<thead>
<tr>
<th>number of true sensors</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>NOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>one (not all)</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>some (not all)</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>all</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

• actuators
  - actuators perform actions
  - an actuator brick looks like this:
• types of actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Handles armature actions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>Has options to follow objects smoothly, primarily for camera objects.</td>
</tr>
<tr>
<td>Constraint</td>
<td>Constraints are used to limit objects locations, distance, or rotation.</td>
</tr>
<tr>
<td>Edit Object</td>
<td>Edits the objects mesh, adds objects, or destroys them.</td>
</tr>
<tr>
<td>Filter 2D</td>
<td>Filters for special effects like sephia colors or blur.</td>
</tr>
<tr>
<td>Game</td>
<td>Handles the entire game and can do things as restart, quit, load, and save.</td>
</tr>
<tr>
<td>Message</td>
<td>Sends messages, which can be received by other objects to activate them.</td>
</tr>
<tr>
<td>Motion</td>
<td>Sets object into motion and/or rotation.</td>
</tr>
<tr>
<td>Parent</td>
<td>Can set a parent to the object, or unparent it.</td>
</tr>
<tr>
<td>Property</td>
<td>Manipulates the objects properties, like assigning, adding, or copying.</td>
</tr>
<tr>
<td>Random</td>
<td>Creates random values which can be stored in properties.</td>
</tr>
<tr>
<td>Scene</td>
<td>Manage the scenes in your .blend file. These can be used as levels.</td>
</tr>
<tr>
<td>Sound</td>
<td>Used to play sounds in the game.</td>
</tr>
<tr>
<td>State</td>
<td>Changes states of the object.</td>
</tr>
<tr>
<td>Steering</td>
<td>Provides pathfinding options for the object.</td>
</tr>
<tr>
<td>Visibility</td>
<td>Changes visibility of the object.</td>
</tr>
</tbody>
</table>

• sample actuator for simple motion — adjust $dx$, $dy$, $dz$ for position and rotation:

• sample actuator for servo motion — adjust forces applied to object:

example code: moving cube

(example code)

example to try

• from https://sites.google.com/site/blendergameprojects/, by Prof Tim Hickey, Brandeis University (http://www.cs.brandeis.edu/~tim)

• Skyracer 2.61
Designed to let you experiment with the motion actuator for avatars. This game kit provides you with a vehicle that can be controlled by WASD commands, and you can experiment with the motion actuator panel to give it a better feel. You could also add more jumps, other racetracks, etc.)