CSCI 1106
Lecture 16

Player Movement

Announcements

• Today’s Topics
  – Collision detection (finish off last lecture)
  – Motivation for player movement
  – Mouse Movement
  – Easing
  – Keyboard Movement
hitTestObject()

• Idea: If the bounding boxes of two objects intersect, a collision has occurred
• Pros: Fast, cheap, simple to use
• Cons:
  – Cannot determine where the collision occurred
  – Irregularly shaped objects have large bounding boxes
  – False positives
• Use:
  if(objA.hitTestObject(objB))
  { ... }
• Obs: Need finer granularity mechanism

hitTestPoint()

• Ideas:
  – Detect whether a specific point is within the shape of the object
  – Useful on vector objects (ones you draw with the rectangle tool)
  – Only the drawn part is checked for overlap with the point
  – The bounding box isn’t considered!
• Pros: Still pretty simple
• Cons:
  – Can only check one point
  – Objects comprise many points so object collisions require multiple checks
    • E.g. ball and paddle
    • Expensive and slow if many points need to be checked
  – Does not work for bitmapped graphics
• Use:
  if(obj.hitTestPoint(x,y))
  { ... }
  no collision  
  collision
Vector vs Bitmapped Graphics

- **Vector** based graphics are those that you draw using the rectangle, circle, or other tools.
- **Bitmap** based graphics are pictures that you import.
- Problem: Flash uses the bounding box for point hit detection on bitmapped graphics.

![Collision diagram](http://www.snap.ednet.ns.ca/hhs/tproffcon12/images/vector-vs-bitmap.png)

A Compound Approach

- Problem:
  - Need to use `hitTestObject()` on irregular shaped object.
  - Bounding box of object differs from object shape.
- Solution:
  - Create invisible objects within this object with smaller bounding boxes.
  - Use the smaller bounding boxes to detect collisions.

![Collision diagram](http://www.snap.ednet.ns.ca/hhs/tproffcon12/images/compound-approach.png)
Player Motion

- All interactive games have player movement
  - Players can move their character or avatar on the screen
  - Players can react to the game and move their avatar
- How the avatar moves is dictated by the game’s
  - Laws and physics of the game
  - Goals and objectives
  - Environment and level of play
- Common ways to move the avatar are through
  - Mouse
  - Keyboard
  - Dedicated game controllers and joysticks

Direct Mouse Movement

- Idea: Make the player the “mouse”
  - The avatar appears where the mouse is pointing to
  - No need to control the velocity of the avatar
  - Position and velocity is managed by the mouse movement
- How:
  - Import the Mouse class
    ```
    import flash.ui.Mouse;
    ```
  - Hide the mouse at the start of the level so that the avatar replaces the mouse pointer
    ```
    Mouse.hide();
    ```
  - Set the player object’s coordinates to the mouse coordinates at each ENTER_FRAME event
    ```
    avatar.x = mouseX;
    avatar.y = mouseY;
    ```
Direct Mouse Movement

• Pros:
  – Easy
  – Not much code required

• Cons:
  – Restrictions on movement may be needed, e.g.,
    • Disallowing movement in some dimensions (paddle)
    • Checking if mouse is over the game panel area
  – Violates most accepted laws of physics
    • Avatar can accelerate and move instantly

• How can we solve these problems?

Mouse Movement using Easing

• Idea: gradually move avatar toward the location clicked on with the mouse pointer
  – A mouse click sets the target to move toward
  – Calculate distance between the avatar and target
  – Incrementally move the avatar toward the target
  – Note: the avatar isn’t guaranteed to reach the target because
    the target will change if another location is clicked first

• Pros:
  – Makes the physics of the game more realistic
  – Restricts avatar movement by ignoring clicks on illegal areas of
    the stage

• Cons:
  – Allows only coarse-grained movement
Implementing Easing

- Declare an EASING constant
  - $0 < \text{EASING} < 1$
  - Smaller constant implies slower movement
- Declare “target” variables $(tx, ty)$
- Set $(tx, ty)$ to the avatar’s location
- On each MOUSE_DOWN event
  - Sets the target variables to the mouse location
    $$tx = \text{mouseX};$$
    $$ty = \text{mouseY};$$
- On each NEXT_FRAME event
  - Calculate the distance $d$ between avatar and target
    $$dx = tx - x$$
    $$dy = ty - y$$
    $$d = \sqrt{(dx^2 + dy^2)}$$
  - If $d > 1$ pixel, move avatar toward the target
    $$x = x + (dx \times \text{EASING})$$
    $$y = y + (dy \times \text{EASING})$$

Keyboard based Movement

- Idea: Move the player with the keyboard
  - The arrow keys control the direction that the avatar moves
  - These directions allow the player to move diagonally as well
  - Need to respond to the KEY_DOWN and KEY_UP events
  - More than one key can be down at the same time
- Pros:
  - Very precise movement
- Con:
  - Requires the player to learn the control keys
Implementing Keyboard Controls

- Import the KeyboardEvent and Keyboard classes
- Use two variables, (vx,vy) to store the avatar’s velocity (initially 0)
- Listen for the KEY_DOWN, KEY_UP, and ENTER_FRAME events
- On a KEY_DOWN event
  - Check which of the arrow keys are pressed and set velocity
    if (event.keyCode == Keyboard.LEFT) vx = -5;
    if (event.keyCode == Keyboard.RIGHT) vx = 5;
    if (event.keyCode == Keyboard.UP) vy = -5;
    if (event.keyCode == Keyboard.DOWN) vy = 5;
- On a KEY_UP event
  - Check which of the arrow keys were released and reset velocity
    if (event.keyCode == Keyboard.LEFT) vx = 0;
- On the ENTER_FRAME event
  - Update the avatar’s position
    x = x + vx;
    y = y + vy;