Game Engine
Lecture 3 (7/6/2007)

Topics

• Why it's useful
• Structure
• Important Classes and Methods
• Manual

Why it’s Useful

• Different games have many things in common
• All draw things (Sprites) on the screen
• All have a main “game loop,” which runs certain code repeatedly (to move things around, check for collisions, check for user input, etc.)
• All must be able to take user input (from the mouse or keyboard)
• Most make sounds
• All must be able to start/pause and turn on/off sound
• The game engine has all the basics already coded in classes that may be reused for every game!

Useless Fact of the Day

• World beer consumption (from Wikipedia):

<table>
<thead>
<tr>
<th>Rank</th>
<th>Country</th>
<th>Consumption (L/yr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Czech Republic</td>
<td>196.6</td>
</tr>
<tr>
<td>2</td>
<td>Republic of Ireland</td>
<td>131.1</td>
</tr>
<tr>
<td>3</td>
<td>Germany</td>
<td>115.5</td>
</tr>
<tr>
<td>4</td>
<td>Australia</td>
<td>109.9</td>
</tr>
<tr>
<td>5</td>
<td>Austria</td>
<td>105.5</td>
</tr>
<tr>
<td>6</td>
<td>United Kingdom</td>
<td>99.0</td>
</tr>
<tr>
<td>7</td>
<td>Belgium</td>
<td>93.0</td>
</tr>
<tr>
<td>8</td>
<td>Denmark</td>
<td>91.5</td>
</tr>
<tr>
<td>9</td>
<td>Finland</td>
<td>95.0</td>
</tr>
<tr>
<td>10</td>
<td>Luxemburg</td>
<td>94.4</td>
</tr>
<tr>
<td>11</td>
<td>Slovenia</td>
<td>94.1</td>
</tr>
<tr>
<td>12</td>
<td>Spain</td>
<td>93.6</td>
</tr>
<tr>
<td>13</td>
<td>United States</td>
<td>81.8</td>
</tr>
<tr>
<td>14</td>
<td>Croatia</td>
<td>81.2</td>
</tr>
<tr>
<td>15</td>
<td>Netherlands</td>
<td>79.0</td>
</tr>
</tbody>
</table>
Sprite Class

- Visible things on the screen (canvas)
- Has instance variables, mutator (set) and accessor (get) methods for:
  - Shape
  - Location
  - Size
  - Rotation
  - Color
- Three main classes: Sprite, StringSprite, ImageSprite

ImageSprites and StringSprites

- An ImageSprite has an image instead of a shape
  - The ImageSprite constructor takes the file name of your image
  - Example:
    ```java
    ImageSprite monster = new ImageSprite("resources/aMonsterPicture.jpg");
    ```
- A StringSprite has words (a string of characters)
  - The StringSprite constructor takes a string
  - Example:
    ```java
    StringSprite blah = new StringSprite("blah blah blah blah blah");
    ```

Sprite Class - Simple Example

1. Declare a Sprite object:
   ```java
   Sprite pongPaddle;
   ```
2. Create an EllipseSprite/RectangleSprite/LineSprite/PieSprite/etc object to store in the Sprite variable, sending the width and height (1 and 2 in this case) to the constructor:
   ```java
   pongPaddle = new RectangleSprite(1, 2);
   ```
3. Give the Sprite a location, scale, and color (and optionally, a rotation [not shown]):
   ```java
   pongPaddle.setLocation(0.03, 0.5);
   pongPaddle.setScale(0.3);
   pongPaddle.setColor(new Color("Red"));
   ```
4. Add the Sprite to the canvas:
   ```java
   canvas.addSprite(pongPaddle);
   ```

ImageSprites and StringSprites

- An ImageSprite has an image instead of a shape
  - The ImageSprite constructor takes the file name of your image
  - Example:
    ```java
    ImageSprite monster = new ImageSprite("resources/aMonsterPicture.jpg");
    ```
- A StringSprite has words (a string of characters)
  - The StringSprite constructor takes a string
  - Example:
    ```java
    StringSprite blah = new StringSprite("blah blah blah blah blah");
    ```
Custom Sprites

- You can make your own custom Sprite class by extending the standard Sprite class.
- In the constructor of your new class: (a) Call super(); (b) Make a shape (or image or string), if you extended ImageSprite or StringSprite; and (c) Call setShape(theNameOfTheShapeYouMade);
- Example:

  ```java
  public class SquareSprite extends Sprite {
      public SquareSprite() {
          super();
          Rectangle square = new Rectangle(3, 1);
          setShape(square);
      }
  }
  ```

- As another example, the EllipseSprite class does this using fewer lines of code -- it's just packing steps (a) (b) and (c) into one line! The steps have been separated here, but if you understand the other way you can do that (the method “super” calls the constructor for normal Sprites, which can optionally take a shape object as an argument, which is why the EllipseSprite constructor doesn't call “setShape”).

Tracker Instance Variables

- In your custom Tracker, you'll want to declare some instance variables:

  - If your Tracker moves Sprites, you'll probably want:
    ```java
    // the sprite's velocity (direction and speed per second)
    Vector velocity;
    // on x and y separately, the sprite should move each frame
    Coordinate translation;
    ```
  - If your Tracker rotates Sprites, you'll want:
    ```java
    // the speed of rotation, in degrees per second
    double degreesPerSecond;
    // the amount the sprite has rotated since last frame
    double rotation;
    ```
  - If your Tracker scales Sprites, you'll want:
    ```java
    // the speed of scaling, in factor-change per second
    double scaleFactorPerSecond;
    // the factor by which you want to scale the sprite since last frame (I doesn't change the size)
    double scaleFactor;
    ```

Tracker Methods

- Makes Sprites move, with some coded-in behavior
- A Tracker object is attached to the Sprite(s) it moves

  - Example: We can attach a Tracker object named “myTracker” to the Sprite named “oval” with oval.setTracker(myTracker);
  - When you write an implementation of the Tracker class, you extend the standard Tracker class (in other words, the declaration line of your new class looks like this: public class BounceTracker extends Tracker)

- You must define these methods, according to how you want your Tracker to work:

  - void advanceTime(double timePassed)
  - Coordinate getTranslation()

- You can also (optionally) define these methods:

  - double getScaleFactor()
  - double getRotationAddition()

- For an example Tracker class, see CircleTracker.java (created in class)
AnimationCanvas

- Area where all Sprites are drawn
- Can have a background color: `canvas.setBackground(new Color("Red"));`
- Sprites must be added to the canvas before they will be seen:
  - `canvas.addSprite(face);`
- Order of addition equals order the Sprites are drawn on the canvas
- If two Sprites overlap, the Sprite added to the canvas last will appear on top of the other

The Flow of a Game

- 1. `main(String[] argv)` method runs automatically (it's the first method to run in any program!). It starts up the game engine with the `runAsApplication()` method.
- 2. Game engine calls `startGame()` method.
- 3. When start button is clicked, game engine calls the method `advanceFrame(double timePassed)` in the main class.
  - We repeat this step very quickly, until the end of the game.
  - As part of this repeating step, the game engine also calls the `advanceTime(double timePassed)` method of every tracker assigned to a Sprite, and then it updates the Sprite's actual position by the translation it gets from the tracker's method `getTranslation()`.
- 4. After the game ends, if the player clicks the start button again, we repeat steps 2 and 3.

Manual

- Can also see class components in Eclipse (in the Outline)

http://www.fangengine.org/doc