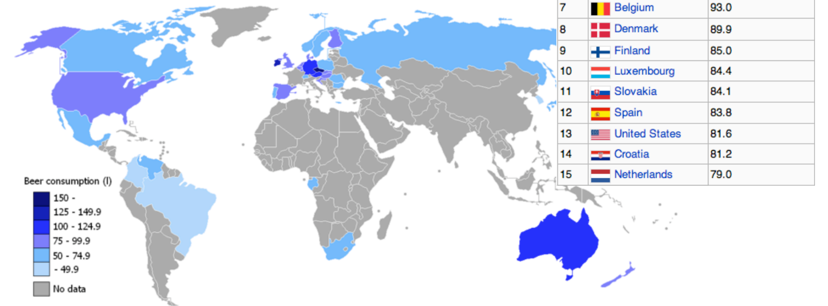
Game Engine

Lecture 3 (7/6/2007)

Useless Fact of the Day

- World beer consuption (from Wikipedia):



Rank	Country	Consumption (L/yr)
1	Czech Republic	156.9
2	Republic of Ireland	131.1
3	Germany	115.8
4	Australia	109.9
5	Austria	108.3
6	United Kingdom	99.0
7	Belgium	93.0
8	Denmark	89.9
9	Finland	85.0
10	Luxembourg	84.4
11	Slovakia	84.1
12	Spain	83.8
13	United States	81.6
14	Croatia	81.2
15	Netherlands	79.0

Topics

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

Why it's Useful

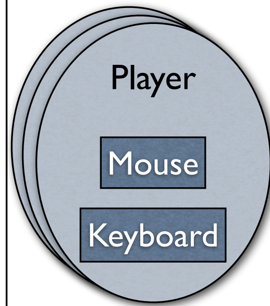
- Different games have many things in common
  - All draw things (Sprites) on the screen
  - All have a main "game loop," which runs certain code repeatedly (to move things around, check for collisions, check for user input, etc.)
  - All must be able to take user input (from the mouse or keyboard)
  - 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!

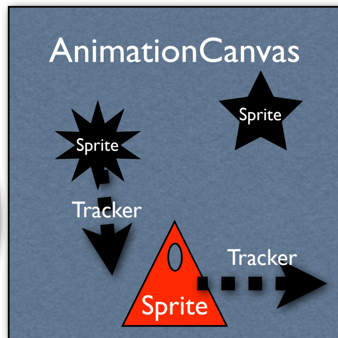
FANG Classes

GameWindow

FrameAdvancer (extends GameWindow)

GameLoop
(extends FrameAdvancer)





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

Sprite Class - Simple Example

- 1. Declare a Sprite object:

```
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:

```
pongPaddle = new RectangleSprite(1, 2);
```

- 3. Give the Sprite a location, scale, and color (and optionally, a rotation [not shown]):

```
pongPaddle.setLocation(0.03, 0.5);
pongPaddle.setScale(0.3);
pongPaddle.setColor(new Color("Red"));
```

- 4. Add the Sprite to the canvas:

```
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:

```
ImageSprite monster = new ImageSprite("resources/aMonsterPicture.jpg");
```

- A **StringSprite** has words (a string of characters)
  - The StringSprite constructor takes a string
    - Example:

```
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:

```
public class SquareSprite extends Sprite
    {
        public SquareSprite()
        {
            super();
            Rectangle square = new Rectangle(1, 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 Class

- 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)

Tracker Instance Variables

- In your custom Tracker, you'll want to declare some **instance variables:**
  - If your Tracker *moves* Sprites, you'll probably want:

```
// the sprite's velocity (direction and speed per second)
Vector velocity;
// the x and y amounts the sprite should move each frame
Coordinate translation;
```

  - If your Tracker *rotates* Sprites, you'll want:

```
// 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:

```
// the speed of scaling, in factor-change per second
double scaleSpeed;
// the factor by which you want to scale the sprite since last frame (1 doesn't change the size)
double scaleFactor;
```

- (note that the names for the variables above could be changed to anything you want!)

Tracker Methods

`void advanceTime(double timePassed)`

- This method is called repeatedly by FANG, and every time it runs, "timePassed" stores the number of seconds that have gone by since last time the method ran (this number will always be about 1/24). You should update instance variables in this method based on "timePassed" -- for example, this line updates the translation based on the velocity:

```
translation.setLocation(velocity.getXChange() * timePassed,
                        velocity.getYChange() * timePassed);
```

`Coordinate getTranslation()`
`double getScaleFactor()`
`double getRotationAddition()`

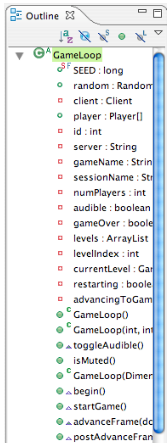
- These methods will also be called repeatedly by FANG, but they should not do any updating of variables -- instead, they should just return the current value for the translation, or scale change, or rotation change. In other words, each of these methods returns a variable that was updated in `advanceTime`. By default, the methods tell FANG to leave the Sprite alone -- `getScaleFactor()` returns 1 (scale the Sprite by 1 times its original scale, which does nothing), `getRotationAddition()` returns 0 (rotate the Sprite 0 more degrees), and `getTranslation()` returns the coordinate (0,0) (move the Sprite 0 to the right and 0 down).
- See **CircleTracker.java,** or **SpinTracker.java,** or **BounceTracker.java** for example Tracker classes

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

http://www.fangengine.org/doc

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