Sprites

In computer graphics, a **sprite** is a two-dimensional image or animation that is integrated into a larger scene. In object oriented programming, a sprite is implemented as a class with properties and behaviors. In more specific terms, a sprite is an abstract data type that represents an image or animation in a GUI application.

Sprite Implementation

In the implementation of a sprite several properties must be considered.

- A sprite must have an image that can be rendered on the screen. If the sprite is animated it may have multiple images that could be stored as an array of images.
- A sprite must have a location property (x and y coordinates) to determine where the image should be rendered.
- A sprite should have a width and height property which would be dependent on the size of the image.
- If the sprite moves around the screen then it would require a direction property.

In an object oriented implementation of a sprite accessor and mutator methods would be required for the properties (fields) listed above.

In addition, the sprite should have a method to render itself to a Graphics context and, if it is animated or move around the screen, a method to update the sprite's image and/or location.

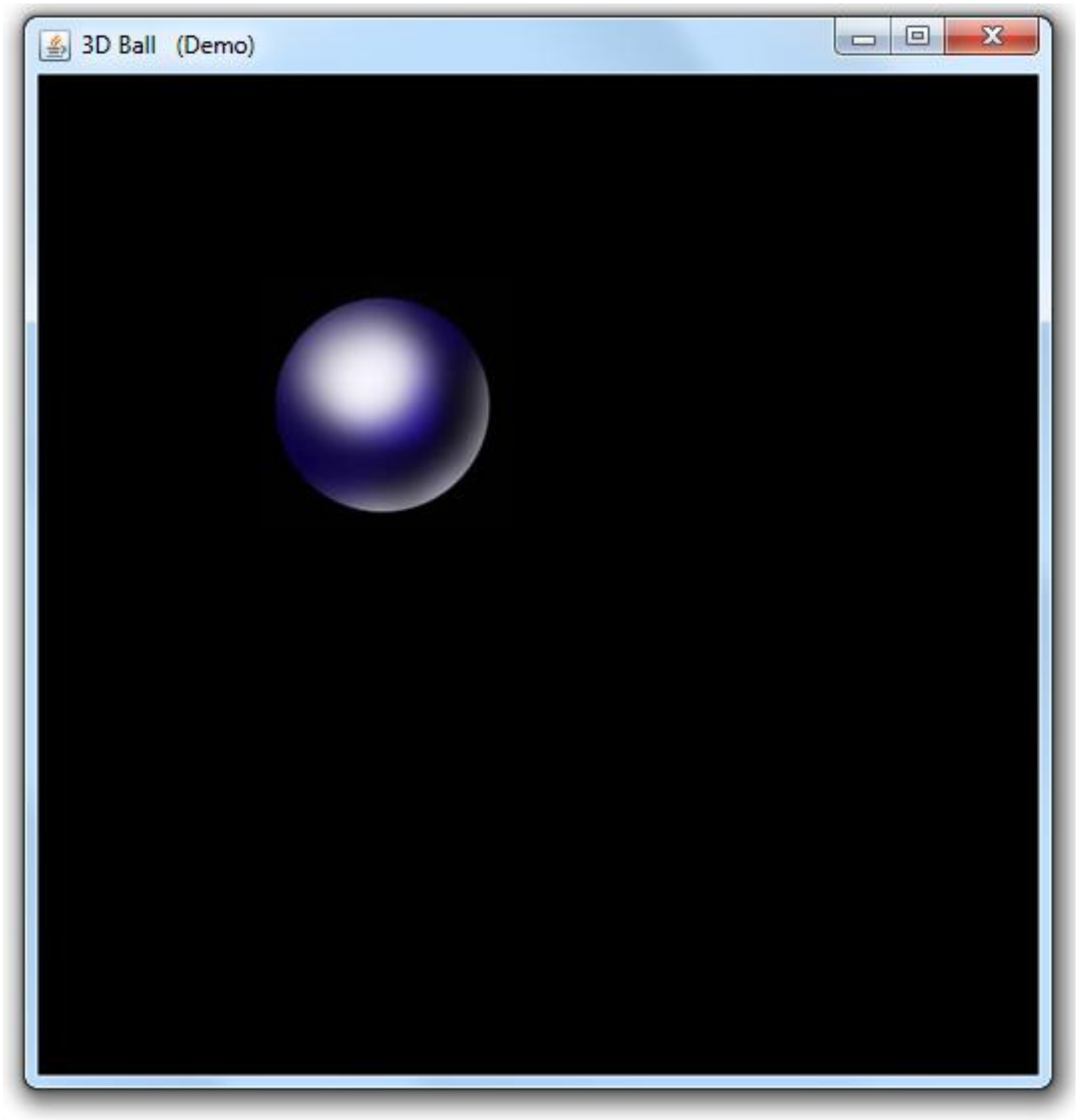
Lab 22 - ASSIGNMENT

Lab 22A - 60 points

PROGRAM OBJECTIVE

Modify the Lab22A (3D Ball) project adding the ability for the user to click above and below the ball causing the ball to move up or down. You will need to modify two classes – the Sprite class and the BallPanel class.

SAMPLE OUTPUT



Lab 22B - 70 points

PROGRAM OBJECTIVE

WAP that displays Pecos Bill and his horse Widow-Maker out on the prairie. Throw in a few cactus, a road-runner, some clouds and the sun.

CLASS OBJECTIVE

Add a Sprite class to the project.
This class will be used to display all the images on the frame.

INSTANCE FIELDS

- `Image image`
- `Rectangle bounds`

METHOD SUMMARY

- **constructor** – (*Arguments:* two ints representing the screen location, a float that will be used to scale the image and a BufferedImage.) A minor calculation will need to be performed to determine the sprites width and height. We have received a float called size that represents a multiplication factor. If size is 1.0 then the object will be its original size. Any value less than 1.0 will cause the image to shrink and any value greater than 1.0 will cause the object to grow proportionally. To calculate the width and height simply get the width and height of the image we received as an argument. Multiply the image **`width`** times the **`scale factor`** received as an argument to determine the width and multiply the image **`height`** times the **`scale factor`** received as an argument to determine the height. You can obtain the height and width of the image by calling the image objects **`getWidth`** and **`getHeight`** methods. Once these calculations have been made set the boundy of the Sprite (**`bounds`**). The x and y positions were received as arguments and you just calculated the width and height.
- **drawSprite** – *Arguments:* an instance of a Graphics object. Draw the image on the graphics object received as an argument. Use the overloaded **`drawImage`** method that receives as arguments the **`image`**, **`x`**, **`y`**, **`width`**, and **`height`**. Set the **`ImageObserver`** to null.
- **accessors** – provide **accessors** that return the sprite's x and y coordinates and the sprite's width and height.
- **mutators** – provide **mutators** that allow the user to change the x and y coordinates.

CLASS OBJECTIVE
Add a PecosPanel class to the project. PecosPanel extends JPanel.
INSTANCE FIELDS
• `private Sprite[] sprites;`
METHOD SUMMARY
• **constructor** – Receives an array of type Sprite as an argument. Assign the array to the instance field *sprites*. Set the background color to CYAN. Set the layout to null. Set the preferred size to 700, 500. • **paintComponent** – Set the drawing color to {200, 110,73} and draw the ground using the `fillRect` method of the `Graphics` class. Draw every sprite in the array of Sprites.

CLASS OBJECTIVE
Add a PecosBill class to the project. PecosBill extends JFrame and is the main class**.**
INSTANCE FIELDS
• `AudioClip music // "pecos.wav"` • private Sprite[] sprites; • private PecosPanel pecosPanel;
METHOD SUMMARY
• **constructor** – Add the code to play the music. 1. Set the title property to "Pecos Bill". 2. Set the icon image to be "icon.png". 3. Set the default close operation to JFrame.EXIT_ON_CLOSE. 4. Add the necessary sprites to the array of sprites. 5. Instantiate an instance of PecosPanel. 6. Set the content pane to be the instance of PecosPanel. 7. Pack the frame. 8. Play the music. • **main** – Create an instance of this class. Show the frame.

Lab 22C - 80 points

PROGRAM OBJECTIVE

Modify the GUI application you created for Lab25B. The clouds should move across the screen from left to right.

CLASS OBJECTIVE

Modify the Sprite class.

ADDITIONAL METHODS

- **act** – This will be an empty method. There will be no code in the **`act`** method.

CLASS OBJECTIVE

Add a Cloud class to the project. Cloud extends Sprite.
This class will be used to display all the clouds.

INSTANCE FIELDS

- **`None required.`**

METHOD SUMMARY

- **constructor** – (*Arguments:* two ints representing the screen location, a float that will be used to scale the image and a BufferedImage). Pass all the arguments on to the super constructor.
- **act** – Increment the sprite's **x** location by four. If the sprite is completely off the panel, then set the sprite's **x** location to the negative value of the sprite's width.

CLASS OBJECTIVE

Modify the PecosBill class.

ADDITIONAL INSTANCE FIELDS

- **`Timer timer – a timer used to constantly update the sprites.`**

MODIFIED METHODS
• **constructor** – Change all sprites that are drawing clouds to instances of a `Cloud` rather than instances of a Sprite. The Cloud constructor requires an additional argument. Instantiate the timer object. Set the delay for 20 milliseconds. [The ActionListener should call the act method for every sprite in the array of sprites and then repaint the panel.] After starting the music, start the timer.

Lab 22D - 90 points

PROGRAM OBJECTIVE

Modify PecosBill so that the sun moves across the screen from right to left. When the sun goes off the screen on the left it should reappear on the right. Think about what you did to make the clouds move.

Lab 22E - 100 points

PROGRAM OBJECTIVE

Modify PecosBill so that the sun moves across the screen from right to left. When the sun goes off the screen on the left it should reappear on the right as the moon! When the moon goes off the screen on the left it should reappear on the right as the sun.

Lab 22F – Challenge Level (Expert Programmers)

PROGRAM OBJECTIVE

Modify PecosBill so that the sun moves across the screen from right to left. When the sun goes off the screen on the left it should reappear on the right as the moon *and the sky should turn BLACK!* When the moon goes off the screen on the left it should reappear on the right as the sun *and the sky should turn CYAN.* ***You will need a way for the Sun/Moon to change the background color of the PecosPanel.***