IN628 2020 Practical 06 – 2D Animation Algorithms & Double Buffering

Your task is to implement class ***Sprite*** as described in lecture, and then use your class to duplicate the demo application. The movement is driven by two timers – one for the single sprite and another for the group of sprites. Use a linked list to hold the group of ***Sprites***. Create class ***SpriteList*** using your ***PelletList*** as a base, and modifying it as needed. You will need to change data types, add methods, and adjust existing methods, but the core logic will remain the same. You do not need to create a ***SpriteManager*** class now, if you don't want to. Note that the animations of the group sprites are not all in lock-step. Implement this more natural-looking behaviour by starting each of the group ***Sprites*** on a different frame of its sprite sheet.

Your ***Sprite*** will need, at a minimum, these class data members (you can, of course, use whatever variable names you prefer). Declare those data members that need a public face as properties following ***C++/CLI*** convention. As we add more complex behaviour to the class throughout the term, we will require additional data members.

Sprite Data Members

Graphics^ canvas	For the instance to draw itself to
Bitmap^ spriteSheet	A Bitmap instance containing the animation sprite sheet
Random^ rGen	Global random number generator used as needed
int nFrames	Number of frames in the sprite sheet animation
int currentFrame	Will hold the number of the frame currently being displayed
int xPos	x-coordinate of the sprite’s upper left corner
int yPos	y-coordinate of the sprite’s upper left corner
int frameWidth	Width in pixels of a single frame in the sprite sheet
int frameHeight	Height in pixels of a single frame in the sprite sheet
int xVel	Horizontal velocity
int yVel	Vertical velocity
Sprite^ Next	Pointer for holding sprites in a linked list

Sprite Methods

Constructor: At a minimum, you will need to pass in a ***Graphics^***, a ***Bitmap^***, a ***Random^*** and the ***number of frames*** in the sprite sheet. Your function prototype might look like:

```
Sprite(Graphics^ startCanvas, Bitmap^ startSpriteSheet, int startNFrames,
Random^ startRGen);
```

You might wish to include (or overload the constructor and have another version that includes), starting values for ***xPos***, ***yPos***, ***xVel*** and ***yVel***. Remember that all class data members **must** be initialised in the constructor. For those whose starting values are not passed in, you will need to come up with sensible defaults.

Note that it is not necessary to pass in values for ***frameWidth*** and ***frameHeight***. Those can (and should) be calculated using ***nFrames*** and the width and height properties of the ***Bitmap***.

void Draw();

This method needs to determine the rectangular portion of the sprite sheet that holds the current frame, and draw those pixels to the ***Graphics^***. To compute the correct rectangle of pixels to display, you will need to use ***currentFrame***, ***frameHeight*** and ***frameWidth***. See the lecture PowerPoint for more detail.

void Erase(Color eraseColour);

When we start drawing backgrounds, we won't need ***Erase*** anymore, but for now, this method should draw a rectangle with the provided colour over the current location of the ***Sprite***.

void Move();

Using vector movement, modify ***xPos*** and ***yPos*** by the corresponding velocity values. Later we will extend ***Sprite::Move()*** to include complex bounds actions. For now, you might want to just prevent the ***Sprite*** from wandering too far off the edge of the ***Form*** (optional for today's practical).

void UpdateFrame();

This method keeps ***currentFrame*** correctly initialised. To perform the animation, you must cycle through all the frames in the sprite sheet, looping back to 0 after you display the final frame. To perform this computation, you need to use ***nFrames***, the total number of frames in the sprite sheet. See the lecture PowerPoint for detail.

void SetSpriteSheet(Bitmap^ newSpriteSheet, int newNFrames);

If you change the sprite sheet, you must make sure ***nFrames***, ***frameWidth*** and ***frameHeight*** are all updated to match the new sprite sheet. Otherwise your calculated display rectangles in the ***Draw()*** method will not be correct.

void Wander();

This is not a core method of the ***Sprite*** class, but is needed to build today's application. The purpose is to make the ***Sprite*** change direction occasionally. Select some probability of change (in the demo, it is 20%). Using the ***Random^***, write code such that the ***xVel*** and ***yVel*** of the ***Sprite*** each toggle (i.e. are multiplied by -1) with that probability. You will need to give some thought to the logic required. Hint: In my ***Wander()*** method, the only call to the

Random^ is: `rGen->Next(WANDER_PROB);` where ***WANDER_PROB*** has been ***#defined*** as 5. This call occurs twice.