IN628 2020 Practical 07 – Directional Sprites

In this practical you will extend your Sprite class to support multiple directions of movement with corresponding sprite sheets. Your task is to duplicate the functionality of the demo application shown in class. Of course, feel free to use different images, of course. Additional images can be found in the **Resource/Extra Images** folder. Use a linked list to manage the chickens; the knight can be managed with either a linked list (which will always contain only one node) or a single **Sprite^** variable, as you prefer. With several sprites on the **Form**, you will need to use a double buffering to eliminate flicker.

To look realistic, your sprite will need different sprite sheets for movement in the different compass directions (e.g. walking left or right, walking toward or away from the viewer). Extend your sprite class to maintain a **SpriteDirections** state variable, and to display the appropriate animation for movement in its current direction. You will need to make the following changes to **Sprite.h**:

- Instead of using a single Bitmap^, you will need an array of Bitmap^. For the demo application, all sprites can move in four directions, so you need an array that can hold four Bitmap^ instances. This array needs to be correctly initialized in the Sprite constructor. In this handout, we will assume this array has been named **'spriteSheets'**, but you may of course name it whatever makes most sense to you.

- Add a new property **SpriteDirections.** Its type will either be int or an enumeration. Either declare constants EAST, SOUTH, WEST and NORTH using #define (these should have the values 0, 1, 2, and 3) or the equivalent enumeration.

- Modify the draw method to access the sprite's images from the array element **spriteSheets[spriteDirections].** Note that the sprite sheets in the Bitmap array should correspond to the direction constants (i.e. the sheet in position EAST should be the one for moving east, the sheet in position SOUTH should be the one for moving south. etc.)

- Add a **setSpriteDirection** method, or declare **SpriteDirections** as a property. Modifying this class data member at runtime should change the sprite's direction of movement. In the demo, the Knight's **SpriteDirections** property is changed in response to the arrow keys, while the Chickens' are controlled entirely by code. These Chickens simply turn around when they hit the edge of the Form, but you should feel free to implement whatever chicken behaviour you like. Think carefully about where this logic should be defined (remember: encapsulation, high cohesion and low coupling), and by whom the resulting method should be called. It is not necessary to descend a child from **Sprite**, just give your **Sprite** class some useful "turn at the edges" logic.

As discussed in lecture, **xVel** and **yVel** will now represent the **magnitude** of a single "step", so will always be non-negative. To move the sprite in the correct direction, you must multiply the velocity values by a direction term, as indicated in the PowerPoint. You can give your **Sprite** an array of Point. My example is called **velocityDirection** – which can be initialized in the **Sprite** constructor. The modification of **XPos** and **YPos** in the **Move()** method can then be done with simple equations where you multiply the velocity terms by the appropriate values from the **velocityDirection** array