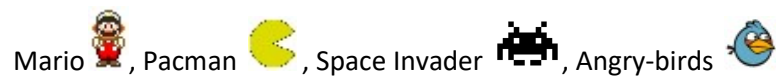


### Laboratory 3: Creating a sprite based game

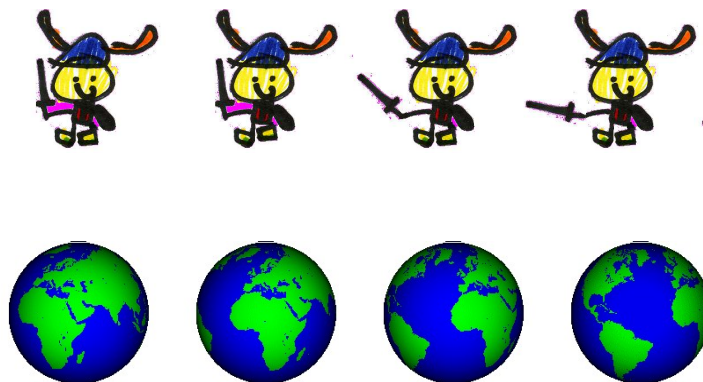
**Introduction:** It is autumn, so let us create a sprite-based game with this as a theme. You may not understand every aspect of the code in this handout. The aim of the laboratory is to allow you witness the thought processes of a competent programmer as they assemble a bigger program. You will also learn to use the tools to work on larger programs.

A sprite is a 2D bitmap object (image) that can be moved around the screen. Many popular games make use of sprites, these include




**Figure 1:** Famous sprites

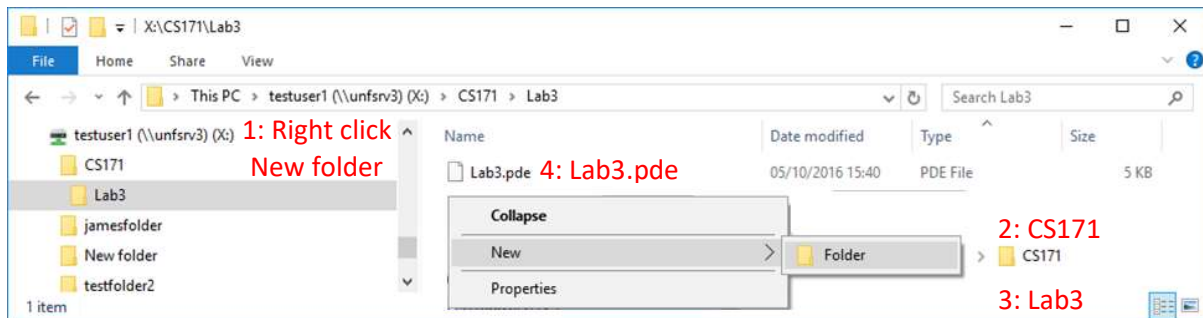
It is possible to animate sprites to give an impression of movement. It is also possible to use sprites to create 3D effects without having to embrace advanced graphics concepts.



**Figure 2:** Animation sheet for a “Happy Viking” and for 3D effect of the world turning.

### Part 1: Setting up the project folder

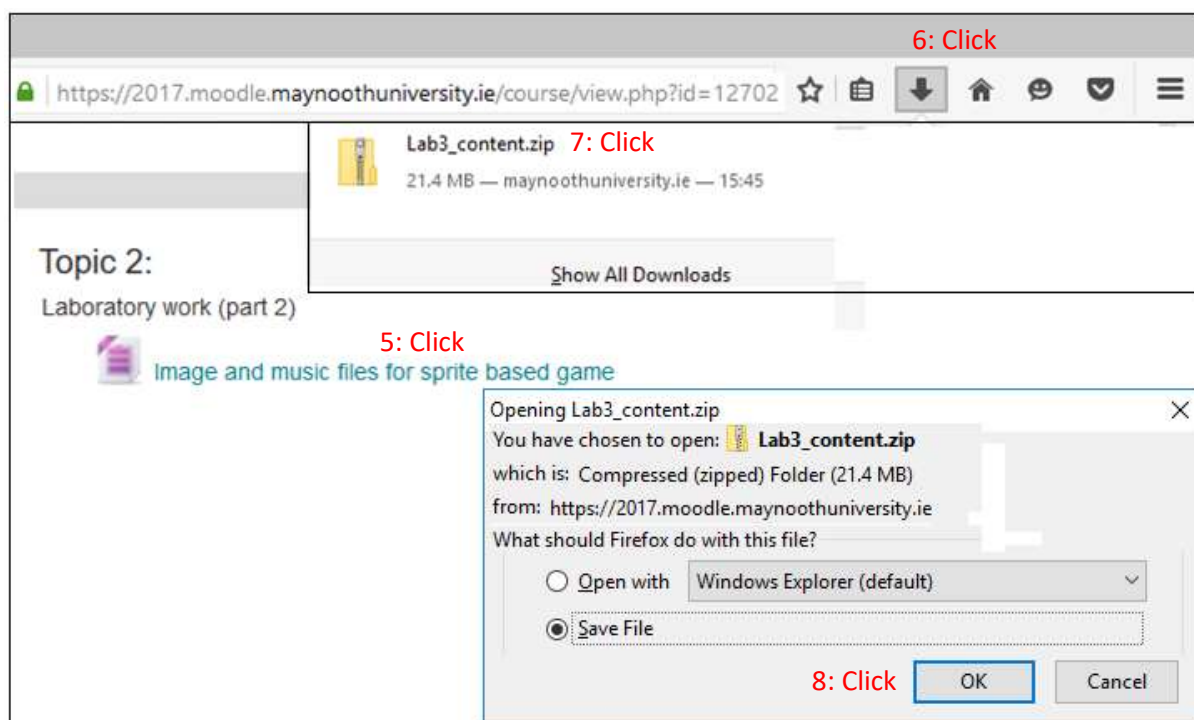
If you have an X:\ drive account, you should save project there so that you can revisit the code at a later stage (from any computer in any lab). To do this open the windows file explorer (press the Windows key, , and “e” at the same time to launch this). Create a new folder on the X:\ drive called *CS171* and a subfolder within this called *Lab3*.




**Figure 3:** Creating a folder on the X drive called CS171 with a sub folder Lab3.

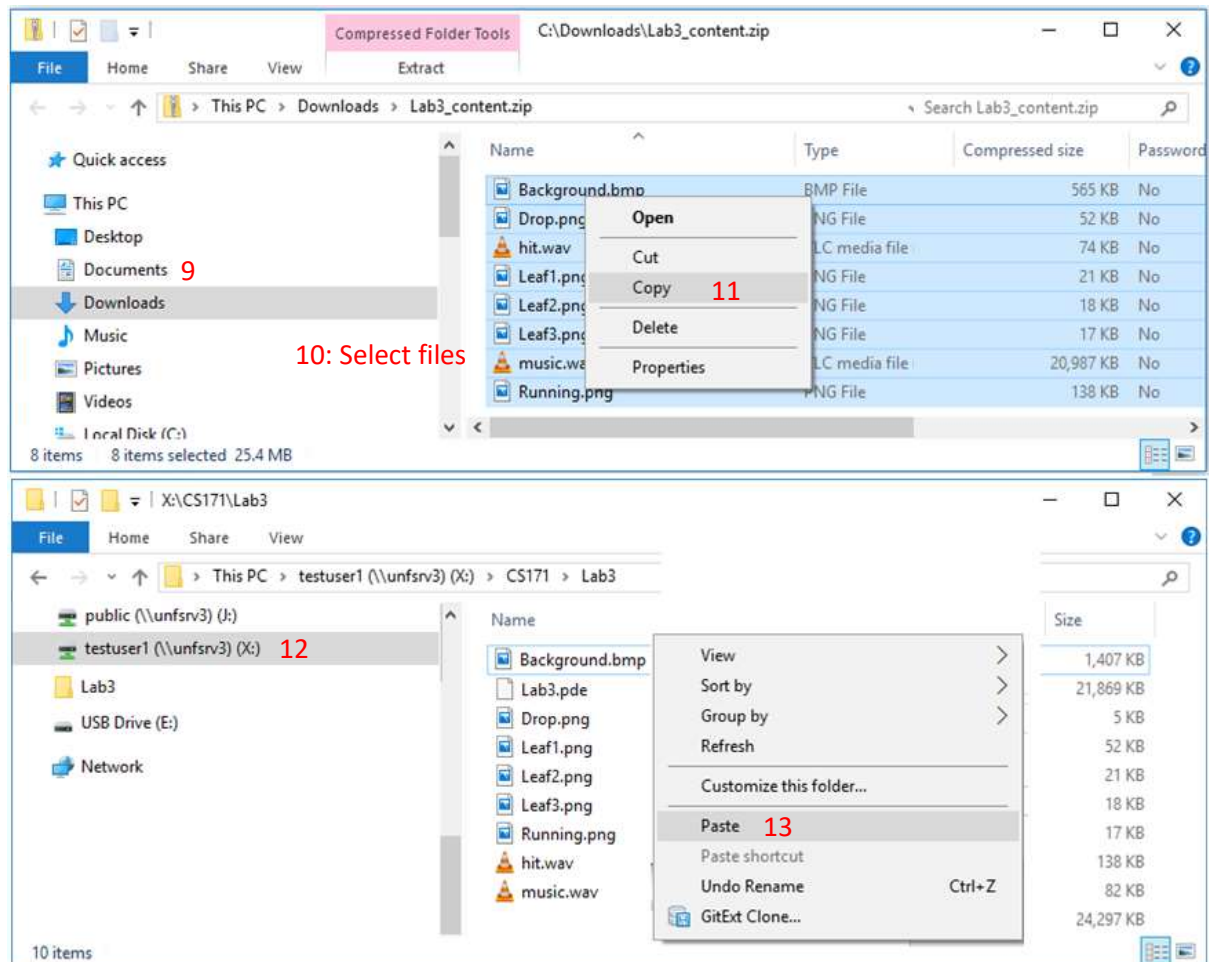
Open a new Processing project, name it (e.g. Lab3.pde), and save it to this new folder.

Download (from Moodle) the pictures and music that we are going to use in the game and copy the files to the folder that contains your Processing program (i.e. Lab3.pde). The files are called “assets” or “content” in graphics programming environments such as Unity.



**Figure 4:** Download and locate the zip file containing image and music files.

To copy the files, open the windows file explorer (press the Windows key, , and “e” at the same time to launch this), navigate to the zip file (This PC-Downloads), click on the zip file called *Lab3\_content.zip*. Highlight the files one at a time using (hold down ctrl + left click mouse on each file), select copy (right click mouse) and then open the destination folder and paste them (right click) into your project folder.



**Figure 6:** Copy files from the zip file in the download folder to your working folder on the X drive.

## Part 2: Setting up an arena in which to play

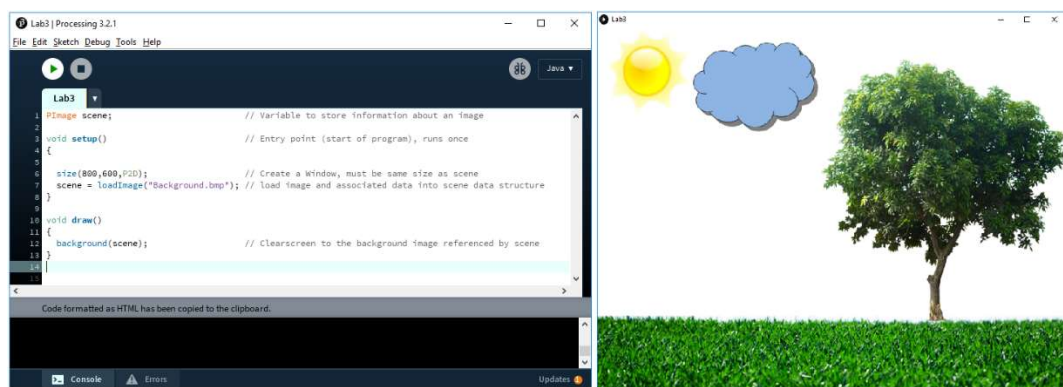
Using PowerPoint clip-art images for grass, a tree, the sun and a cloud were arranged around the slide area. The slide area was then displayed full screen and (shift Print Screen) was used to capture the image. This image was pasted into MS Paint (or Paint.NET) and saved as bitmap called *Background.bmp*.

**To do:** Add the following lines of code to your *Lab3.pde* file and run the program to see the image in your run time window. You may find that your laptop has a graphics card incapable of displaying the image, if this happens using Processing 2.0 will work. Look at the error messages if it does not work. Of course, you could use any drawing package to create your own background image. Use the image provided for now.

```
PImage scene; // Variable to store information about an image

void setup() // Entry point (start of program), runs once
{
    size(800,600,P2D); // Create a Window, must be same size as scene
    scene = loadImage("Background.bmp"); // load image and data into scene data structure
}

void draw()
{
    background(scene); // Display background image referenced by scene
}
```



**Figure 7:** Code to create a Window with a background image.

**To do:** We can now extend the code to draw a raindrop on the screen.

```
PImage scene,drop; // Variables to store information about images

int drop_x,drop_y,drop_count; // Variables to store (x,y) position of drop

void setup() // Entry point (start of program), runs once
{
    ...
    drop = loadImage("Drop.png"); // load image of rain drop into the GPU

    textureMode(NORMAL); // Scale texture Top right (0,0) to (1,1)
    blendMode(BLEND); // States how to mix a new image with the one behind it
    noStroke(); // Do not draw a line around objects

    drop_x=166+(int)random(200); // Choose drop starting position
    drop_y=90;
}

void draw()
{
    background(scene); // Clear screen to the background image, scene

    pushMatrix(); // Store current location of origin (0,0)
    translate(drop_x,drop_y); // Change origin (0,0) for drawing to (drop_x,drop_y)

    beginShape(); // Open graphics pipeline
    texture(drop); // Tell GPU to use drop to texture the polygon
    vertex(-20,-20,0,0); // Load vertex data (x,y) and (U,V) texture data into GPU
    vertex(20,-20,1,0); // Square centred on (0,0) of width 40 and height 40
    vertex(20,20,1,1); // Textured with an image of a drop
    vertex(-20,20,0,1);
    endShape(CLOSE); // Tell GPU you have loaded shape into memory.
    popMatrix(); // Recover origin(0,0) means top left hand corner again.
}
```

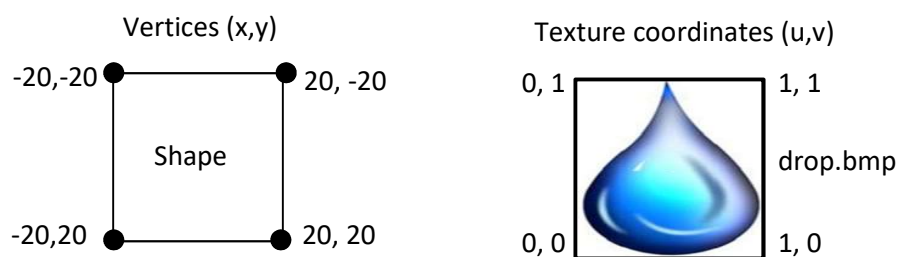
**To do:** Run the code and check that a raindrop appears on the cloud.

Stop and think:

1/ The `pushMatrix()` allows us to store the current origin used for drawing an object on the Window. At the start of the program this is set to (0,0) representing the top left hand corner of the screen, so we can push this location on the stack to save it. We can then move the origin to where we would like to draw the water drop using `translate(drop_x, drop_y)`. This means that we don't have to add the drop (x,y) position to every co-ordinate used to draw the object (as we have been doing so far). When we have drawn the drop we can return the origin back to the top left hand corner using `popMatrix()`. Stacks are very effective data structure for temporary storage of information, "push" to save, "pop" to restore.

2/ A square (shape) described using 4 vertices is entered into the GPU pipeline between the `beginShape()` and the `endShape()` instructions.

3/ For each vertex (corner of the square on screen) we need an additional texture co-ordinate (U,V). The texture coordinates locates a point in the texture that will map image data to the vertex. The GPU can interpolate the correct image data for the space between the vertices. This allows us to cover the surface of an object with an image. The texture coordinates are always floating point numbers in the range [0,1] that map to the original bitmap (whatever its size is in pixels).



**Figure 8:** UV mapping of a texture on to a square defined by 4 vertices.

4/ Things are not draw immediately, the code in the `draw()` method pushes what is to be drawn into the graphics card. Your code is telling the computer what to draw at the next opportunity. You can consider that the graphics card only displays what you have drawn when you get to the "}" at the end of `draw()`, not as you draw it.

**To do:** Now we can add code to make the drop move down the screen until it hits the grass. When it hits the grass, a new starting position in the cloud is identified and the variable *drop\_count* is increased by one.

```
popMatrix();      // Recover the origin, (0,0) now means top left hand corner again,

drop_y+=2;        // Make "drop" move down the screen (two pixels at a time)
if(drop_y>475)    // If y value is entering the grass line
{
    drop_x=166+(int)random(200); // Restart the drop again in the cloud.
    drop_y=90;
}

} // End of draw
```

**To do:** Although a bit repetitive, we can now do the same again for a leaf on the tree.

```
PImage scene,drop,leaf;          // Variables to store information about images
...
int leaf_x,leaf_y,leaf_count;    // Variables to store (x,y) position of leaf

void setup()                     // Entry point (start of program), runs once
{
    ...
    drop = loadImage("Drop.png"); // load image of rain drop into the GPU
    leaf = loadImage("Leaf3.png"); // load image of leaf into the GPU
    ...
    float radius=random(128);      // Chose leaf starting position
    float angle=random(2*PI);      // Random position inside circle
    leaf_x=584+(int)(radius*cos(angle)); // of radius 128 centered on (584,216)
    leaf_y=216+(int)(radius*sin(angle));

}

void draw()
{
    background(scene);             // Clear screen to the image referenced by scene

    //.. draw drop

    pushMatrix(); // Draw leaf
    translate(leaf_x,leaf_y);
    rotate((float)frameCount/10);
    beginShape(); // Draw Leaf
    texture(leaf);
    vertex(-20, -20, 0, 0);
    vertex(20, -20, 1, 0);
    vertex(20, 20, 1, 1);
    vertex(-20, 20, 0, 1);
    endShape(CLOSE);
    popMatrix();

    //.. move drop

    leaf_y+=1; // Make leaf "move" down the screen
    if(leaf_y>475)
    {
        float radius=random(128); // Chose leaf starting position
        float angle=random(2*PI);
        leaf_x=584+(int)(radius*cos(angle));
        leaf_y=216+(int)(radius*sin(angle));
    }

}
```

**To do:** Run the new code and verify that you now have leaves and water drops falling down the screen.



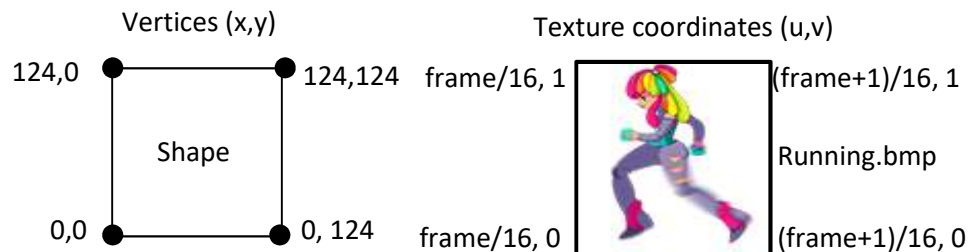
**Figure 9:** Leaf and drop moving on the screen.

We would now like to create an animated character under user control that can run back and forth along the grass line. To create the animation you use a sprite sheet that contains separate frames that when played back in sequence makes the sprite appear to move.



**Figure10:** Running.bmp sprite sheet used to create and animated character.

There are 16 separate images in the sprite sheet. To display the image belonging to a specific frame you use the following UV mapping.



**Figure 11:** UV mapping of a texture on to a square defined by 4 vertices,  $1/16^{\text{th}}$  of the sprite sheet.



**To do:** Add the following code to your program and run it to verify that the character appears on screen.

```
PImage scene,drop,leaf,player;           // Variables to store information about images
...
float frame=0;                           // Start on frame 0 of the sprite sheet
int player_x=400;                         // Start player in middle of screen

void setup()                             // Entry point (start of program), runs once
{
    size(800,600,P2D);                   // Create a Window, must be same size as scene
    ...
    player = loadImage("Running.png");   // load sprite sheet for player
    ...
}

void draw()
{
    background(scene);                   // Clear screen to the background image
    float left =frame/16;
    float right=(frame+1)/16;

    pushMatrix();                         // Draw player
    translate(player_x,360);
    beginShape();
    texture(player);
    vertex( 0, 0, left, 0);
    vertex(124, 0, right, 0);
    vertex(124, 124, right, 1);
    vertex( 0, 124, left, 1);
    endShape(CLOSE);
    popMatrix();                          // Restore origin (top left 0,0)

    ...
}
```



**Figure 12:** Player appears on the screen.

The player now need to change position and animate at the same time. The value in frame continuously increases up to 15 and returns to 0 after 15. The position can be adjusted in either direction in response to the keyboard.



**To Do:** Add the following code to provide keyboard control of the sprite, at the end of *draw()* before the final “}”.

```
leaf_y=216+(int)(radius*sin(angle));
}

// Move player
if (keyPressed == true)
{
    if(keyCode == RIGHT)
    {
        player_x+=8;           // Increase X position move right
        frame++;               // Every step advance the frame
        if(frame>16) frame=0; // If frame is 16 reset it to 0
    }

    if (keyCode == LEFT)
    {
        player_x-=8;           // Decrease X position move left
        frame++;
        if(frame>16) frame=0;
    }
}
}
```

**To do:** Run the program, the player should run correctly to the left but “moonwalk” to the right. To solve this problem we need to introduce a variable to store direction information and flip the sprite when moving right. The code to this is as follows.

```
... int player_x=400; // Start player in middle of screen
int direction=0;

void draw()
{
    ...
    float left =frame/16;
    float right=(frame+1)/16;

    if(direction==1) // Swap left and right UV values
    {                // to reverse direction sprite is facing
        float temp=left;
        left=right;
        right=temp;
    }

    pushMatrix(); // Draw player
    translate(player_x,360);
    beginShape();
    ...

    // Move player
    if (keyPressed == true)
    {
        if(keyCode == RIGHT)
        {
            direction=1; // Set direction to the right
            player_x+=8; // Increase X position move right
            frame++;     // Every step advance the frame
            if(frame>16) frame=0; // If frame is 16 reset it to 0
        }

        if (keyCode == LEFT)
        {
            direction=0; // Set direction to the left
            player_x-=8; // Decrease X position move left
            frame++;
            if(frame>16) frame=0;
        }
    }
}
```

Nearly there, well done. To make it a game we need to add some game logic. We could lose a life each time a leaf or drop hits the grass, so when this occurs a variable called “*lives*” decrease by one. The player can stop them hitting the grass by running into them. The game terminates when you lose all your lives. You get a new life if you catch five drops or five leaves. The code for this logic is shown below.

```
...

int score=0,lives=3;           // Set initial score and number of lives
int quit_flag=0;              // This is set to 1 when game is over

void setup()                  // Entry point (start of program), runs once
{
    ...
}

void draw()
{
    background(scene);        // Clear screen to the background image
    ...

    drop_y+=2;                // Make "drop" move down the screen (two pixels at a time)
    if(drop_y>475)            // If y value is entering the grass line
    {
        drop_x=166+(int)random(200); // Restart the drop again in the cloud.
        drop_y=90;
        lives--;              // lost a life
    }

    leaf_y+=1;                // Make leaf "move" down the screen
    if(leaf_y>475)
    {
        float radius=random(128);    // Chose leaf starting position
        float angle=random(2*PI);
        leaf_x=584+(int)(radius*cos(angle));
        leaf_y=216+(int)(radius*sin(angle));
        lives--;                    // lost a life
    }

    ...

    if ((drop_y>368)&&(drop_y<470))    // If drop is on same level as player
    {
        if(abs((drop_x+10)-(player_x+62))<25) // And drop is near player
        {
            drop_count++;                // Increase drop count by one (caught)

            drop_x=166+(int)random(200); // Restart a new drop in the cloud
            drop_y=90;
        }
    }

    if ((leaf_y>368)&&(leaf_y<470))    // If leaf is on same level as player
    {
        if(abs((leaf_x+10)-(player_x+62))<25) // And leaf is near player
        {
            leaf_count++;                // Increase leaf count by one (caught)

            float radius=random(128);    // Chose leaf starting position
            float angle=random(2*PI);
            leaf_x=584+(int)(radius*cos(angle));
            leaf_y=216+(int)(radius*sin(angle));
        }
    }

    textSize(18);                // Display score information on the screen
    fill(0,0,255);
    text("Drop:"+drop_count, 540, 20);

    fill(0,255,0);
    text("Leaf:"+leaf_count, 620, 20);
}
```

```

fill(255,0,0);
text("Lives:"+lives, 700, 20);

fill(0,0,0);
text("Score:"+score, 620, 60);

// Scoring and game logic
if (lives<1) text("Game over", 120, 300); // Score of 0 display game over
score++;

if(leaf_count>5)                // Every five leaves increase  lives by one
{
    leaf_count-=5;
    lives++;
}

if(drop_count>5)                // Every five drops increase  lives by one
{
    drop_count-=5;
    lives++;
}

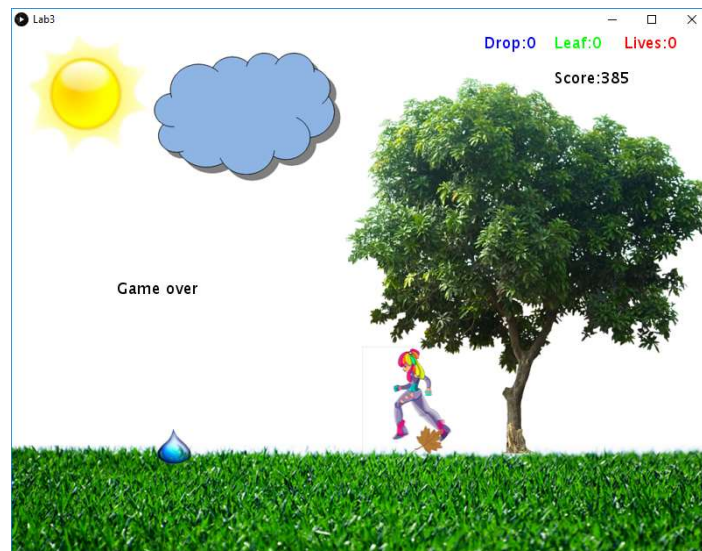
if(quit_flag==1)                // Wait five seconds before exiting
{
    delay(5000);
    exit();
}

if (lives<1) // All lives lost so game over but
{
    // return to draw one more time to
    quit_flag=1; // allow "Game Over to be displayed.
}

// Screen only drawn by graphics card at this point
// not immediately after they are entered into GPU pipeline
}

```

**To Do:** Run the program one more time to see if it plays correctly.



**Figure 13:** Final game with a low score (you could go on and add sound, see later)

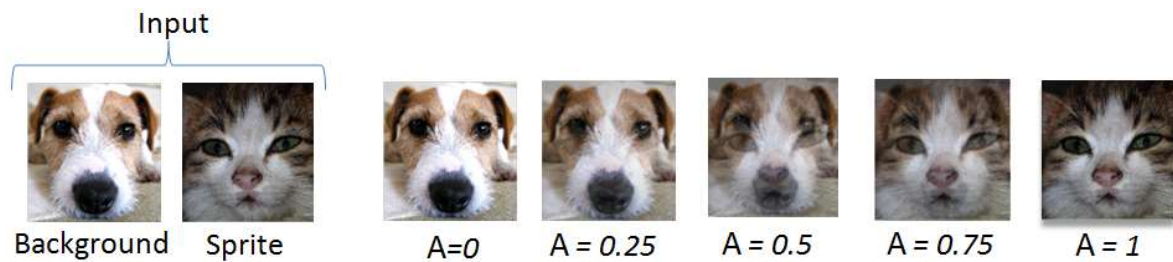
### Aside: Alpha blending

The pixels in the textures contain image information in R,G and B values. In addition, a fourth value known, as alpha is included. This sets the transparency of the pixel. The sprite sheets we used had alpha values equal to zero around there edges so the background would show through. Without alpha blending every sprite would appear as a solid rectangle with a picture on it.

$$(RGB)Screen = (1-ASprite)*(RGB)Background + (ASprite)*(RGB)Sprite$$

*A= 0 New frame buffer pixel solely the existing frame buffer value*

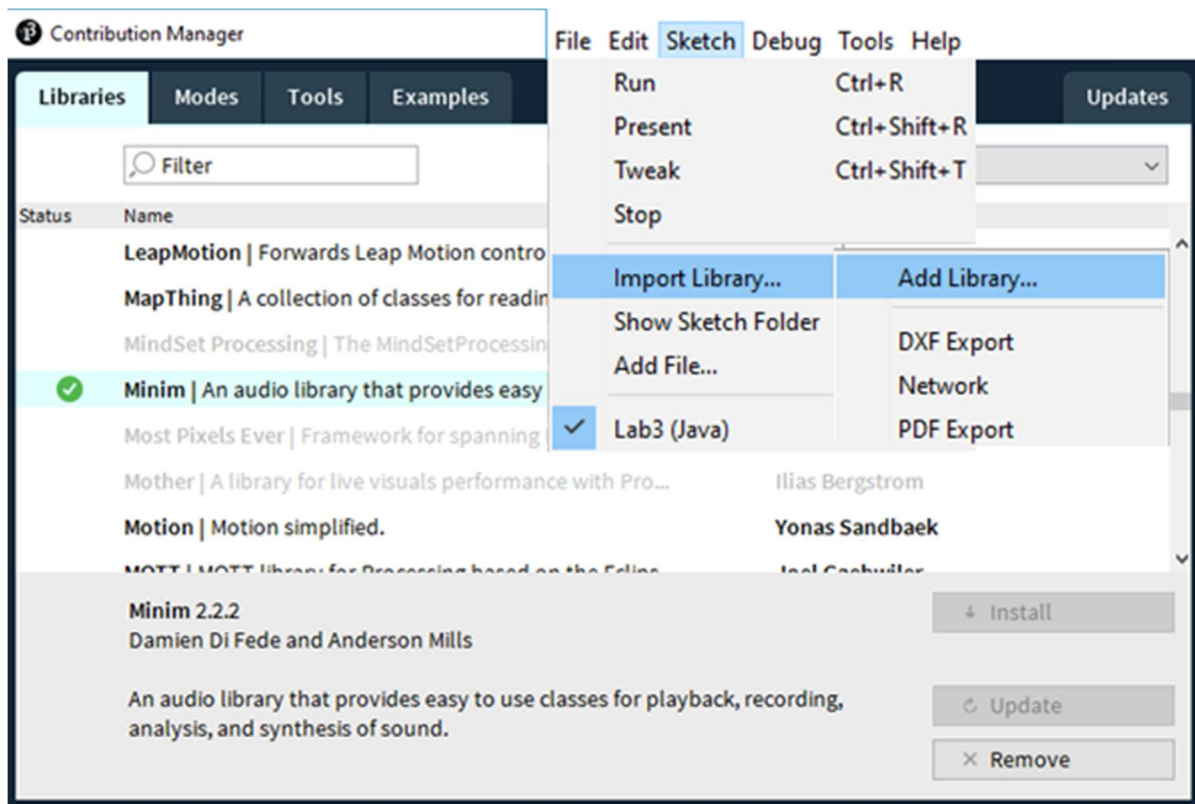
*A= 1 New frame buffer pixel solely the texture/source buffer value*



**Figure 14:** Alpha blending values changing from 0 (transparent) to 1 (opaque).

**To Do:** Finally, if you would like to add sound then you will need to add the sound library to Processing (this has not been done on your laboratory machine).

Select Sketch-Import Library-Add Library and select the Minim sound library.



**Figure 15:** Installing the Minim sound library

Assuming there is a file called *music.wav* in the same folder as the Project file *Lab3.pde* then you only need to add the following highlighted code to produce sound.

```
//SoundFile file;
import ddf.minim.analysis.*;
import ddf.minim.*;
Minim      minim;
AudioPlayer jingle;

PImage tex,back,leaf,drop;
...
void setup()
{
    size(800,600,P2D);

    // Load a soundfile from the data folder of the sketch and play it back in a loop
    minim = new Minim(this);
    jingle = minim.loadFile("music.wav");    // load the music file into memory
    jingle.loop();                          // play the file on a loop

    tex = loadImage("Running.png");
```

Add code at start of program

Add code after size();

You could investigate playing the 'hit.wav' file each time a drop or leaf is not caught.

## Full listing (for reference)

```
//SoundFile file;
import ddf.minim.analysis.*;
import ddf.minim.*;
Minim      minim;
AudioPlayer jingle;

PImage scene,drop,leaf,player;           // Variables to store information about images

int drop_x,drop_y,drop_count;             // Variables to store (x,y) position of drop
int leaf_x,leaf_y,leaf_count;             // Variables to store (x,y) position of drop

float frame=0;                            // Start on frame 0 of the sprite sheet
int   player_x=400;                        // Start player in middle of screen
int   direction=0;

int score=0,lives=3;                      // Set initial score and number of lives
int quit_flag=0;                          // This is set to 1 when game is over

void setup()                              // Entry point (start of program), runs once
{
    size(800,600,P2D);                    // Create a Window, must be same size as scene

    // Load a soundfile from the data folder of the sketch and play it back in a loop
    minim = new Minim(this);
    jingle = minim.loadFile("music.wav");  // load the music file into memory
    jingle.loop();                         // play the file on a loop

    scene = loadImage("Background.bmp");   // load image and associated data into scene data structure
    drop = loadImage("Drop.png");          // load image of rain drop into the GPU
    leaf = loadImage("Leaf3.png");         // load image of leaf into the GPU
    player = loadImage("Running.png");     // load sprite sheet for player

    textureMode(NORMAL);                   // Scale texture Top right (0,0) to (1,1)
    blendMode(BLEND);                      // States how to mix a new image with the one behind it
    noStroke();                            // Don't draw a line around objects

    drop_x=166+(int)random(200);           // Choose drop starting position
    drop_y=90;

    float radius=random(128);              // Chose leaf starting position
    float angle=random(2*PI);              // Random position inside circle
    leaf_x=584+(int)(radius*cos(angle));   // of radius 128 centered on (584,216)
    leaf_y=216+(int)(radius*sin(angle));
}

void draw()
{
    background(scene);                     // Clearscreen to the background image referenced by scene

    float left =frame/16;
    float right=(frame+1)/16;

    if(direction==1)                       // Swap left and right UV values
    {                                       // to reverse direction sprite is facing
        float temp=left;
        left=right;
        right=temp;
    }
}
```

```

pushMatrix(); // Draw player
translate(player_x,360);
beginShape();
texture(player);
vertex( 0, 0, left, 0);
vertex(124, 0, right, 0);
vertex(124, 124, right, 1);
vertex( 0, 124, left, 1);
endShape(CLOSE);
popMatrix(); // Restore origin (top left 0,0)

pushMatrix(); // Store current location of origin (0,0) means top left hand corner
translate(drop_x,drop_y); // Chnage origin (0,0) for drawing to screen position (drop_x,drop_y)
beginShape(); // Open graphics pipeline
texture(drop); // Tell GPU to use drop to texture the polygon
vertex(-20, -20, 0, 0); // Load vertex position data (x,y) and (U,V) texture data into GPU
vertex(20, -20, 1, 0); // Square centered on (0,0) of width 40 and height 40
vertex(20, 20, 1, 1); // Textured with an image of a drop
vertex(-20, 20, 0, 1);
endShape(CLOSE); // Tell GPU you have loaded shape into memory.
popMatrix(); // Recover the origin, (0,0) now means top left hand corner again,

pushMatrix(); // Draw leaf
translate(leaf_x,leaf_y);
rotate((float)frameCount/10);
beginShape(); // Draw Leaf
texture(leaf);
vertex(-20, -20, 0, 0);
vertex(20, -20, 1, 0);
vertex(20, 20, 1, 1);
vertex(-20, 20, 0, 1);
endShape(CLOSE);
popMatrix();

drop_y+=2; // Make "drop" move down the screen (two pixels at a time)
if(drop_y>475) // If y value is entering the grass line
{
    drop_x=166+(int)random(200); // Restart the drop again in the cloud.
    drop_y=90;
    lives--; // lost a life
}

leaf_y+=1; // Make leaf "move" down the screen
if(leaf_y>475)
{
    float radius=random(128); // Chose leaf starting position
    float angle=random(2*PI);
    leaf_x=584+(int)(radius*cos(angle));
    leaf_y=216+(int)(radius*sin(angle));
    lives--; // lost a life
}

// Move player
if (keyPressed == true)
{
    if(keyCode == RIGHT)
    {
        direction=1; // Set direction to the right
        player_x+=8; // Increase X position move right
        frame++; // Every step advance the frame
        if(frame>16) frame=0; // If frame is 16 reset it to 0
    }

    if (keyCode == LEFT)
    {
        direction=0; // Set direction to the left
        player_x-=8; // Decrease X position move left
        frame++;
        if(frame>16) frame=0;
    }
}
}

```



```

if ((drop_y>368)&&(drop_y<470))    // If drop is on same level as player
{
    if(abs((drop_x+10)-(player_x+62))<25) // And drop is near player
    {
        drop_count++;                // Increase drop count by one (caught)

        drop_x=166+(int)random(200); // Restart a new drop in the cloud
        drop_y=90;
    }
}

if ((leaf_y>368)&&(leaf_y<470)) // If leaf is on same level as player
{
    if(abs((leaf_x+10)-(player_x+62))<25) // And leaf is near player
    {
        leaf_count++;                // Increase leaf count by one (caught)

        float radius=random(128);    // Chose leaf starting position
        float angle=random(2*PI);
        leaf_x=584+(int)(radius*cos(angle));
        leaf_y=216+(int)(radius*sin(angle));
    }
}

textSize(18);                        // Display score information on the screen
fill(0,0,255);
text("Drop:"+drop_count, 540, 20);

fill(0,255,0);
text("Leaf:"+leaf_count, 620, 20);

fill(255,0,0);
text("Lives:"+lives, 700, 20);

fill(0,0,0);
text("Score:"+score, 620, 60);

// Scoring and game logic
if (lives<1) text("Game over", 120, 300); // Score of 0 display game over
score++;

if(leaf_count>5)                    // Every five leaves increase lives by one
{
    leaf_count-=5;
    lives++;
}

if(drop_count>5)                    // Every five drops increase lives by one
{
    drop_count-=5;
    lives++;
}

if(quit_flag==1)                    // Wait five seconds before exiting
{
    delay(5000);
    exit();
}

if (lives<1) // All lives lost so game over but
{
    // return to draw one more time "
    quit_flag=1; // allow "Game Over to be displayed.
}

// Screen only drawn by graphics card at this point
// not immediatley after they are entered into GPU pipeline
}

```