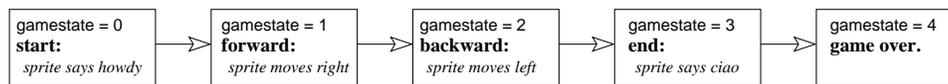


cis3.5, spring 2009, lab III.3 / prof sklar.
Understanding “Game State”

What is “game state”?

We have discussed briefly in class the notion of *game state*. The idea is that any game consists of a sequence of *states*, where each state is characterized by a combination of visual, audio and/or animation cues. Below is an example of a game with four states. Each state is given a number, starting with 0 and ending with 3 (computer scientists always start counting with 0).



- The first state, *gamestate* = 0, is when the game starts up. In this state, the screen will show a sprite (character) who goes to its starting location and says “howdy”. Then the game state changes to 1.
- In the second state, *gamestate* = 1, the sprite moves to the right, until it reaches the far right edge of its display window. Then the game state changes to 2.
- In the third state, *gamestate* = 2, the sprite moves to the left, until it reaches the far left edge of its display window. Then the game state changes to 3.
- In the fourth state, *gamestate* = 3, the sprite stops moving and says “ciao.” Then the game state changes to 4.
- In the fifth state, the game is over.

Your assignment with this lab is to create a little game with a small number of states, like the one above, and code the game in both Processing and Scratch. Before you do that, have a look at the next two pages which show how to code the above example in each environment.

→ **FIRST**, design your game states using a diagram like mine, above.

→ **SECOND**, create your animation in Processing. Use my example to get started. Type it in and make sure that it works for you and that you understand it. Then modify it to follow your game state design.

→ **THIRD**, create your animation in Scratch. Again, use my example to get started. Enter it and make sure that it works for you and that you understand it. Then modify it to follow your game state design.

You will notice in the example code for both Processing and Scratch that I use a numeric entity called a *variable* to store the “game state”. A variable is a labeled piece of memory inside the computer. You can think of it as a handy box where you can put a piece of information. You give the box a label, or name, and then you put a value inside the box. When you need to refer to the value stored inside the box, you just use its name. You can change the value inside the box, while the program runs.

In the case of my examples, I have named the box `gamestate` and given it a value of 0 to start with. As the game progresses, the program changes the value of my variable, i.e., the value stored inside the box labeled `gamestate`.

Using “game state” in Processing

On the right is the Processing code for a little animation that uses “game state”, as described on the previous page. Notice that there is a variable called `gamestate` declared at the top of the program. Inside the `setup()` function, this variable is set to 0.

The animation shows a black circle (acting as my “sprite”) that starts on the left edge of the display window and says “howdy”. This is `gamestate == 0`.

Note that in Processing, you need to use **2** equals signs, `==`, to test to see if two values (on either side of the `==` signs) are equal to each other.

The circular sprite moves, along with the text, to the right. This is `gamestate == 1`. When the sprite reaches the right edge of the display window, the value of the game state variable changes to 2.

In game state 2, the circular sprite moves, along with the text, to the left. When the sprite reaches the left edge of the display window, the value of the game state variable changes to 3.

In game state 3, the sprite stops moving and its text message changes from “howdy” to “ciao”. Then the game state changes to 4.

Game state 4 means that the game is over, and the program stops drawing.

```
int gamestate;
int x, y;
int tx, ty;
PFont font;

void setup() {
  frameRate( 10 );
  font = loadFont( "ArialMT-12.vlw" );
  textFont( font );
  gamestate = 0;
  x = 0;
  y = height-20;
}

void draw() {
  println( ‘gamestate=’ + gamestate );
  tx = x + 10;
  ty = y - 10;
  if ( gamestate == 0 ) {
    background( #ffffff );
    ellipse( x,y,10,10 );
    fill( #000000 );
    text( "howdy",tx,ty );
    gamestate = 1;
  }
  else if ( gamestate == 1 ) {
    background( #ffffff );
    ellipse( x,y,10,10 );
    fill( #000000 );
    text( "howdy",tx,ty );
    x = x + 10;
    if ( x > width ) {
      gamestate = 2;
    }
  }
  else if ( gamestate == 2 ) {
    background( #ffffff );
    ellipse( x,y,10,10 );
    fill( #000000 );
    text( "howdy",tx,ty );
    x = x - 10;
    if ( x < 10 ) {
      gamestate = 3;
    }
  }
  else if ( gamestate == 3 ) {
    background( #ffffff );
    ellipse( x,y,10,10 );
    fill( #000000 );
    text( "ciao",tx,ty );
    gamestate = 4;
  }
  else if ( gamestate == 4 ) {
    noLoop();
  }
}
```

Using “game state” in Scratch

On the right is the Scratch code for the same animation. It also uses “game state”. There is a *variable* called `gamestate` used.

Click on the Variables tab in order to create a variable in Scratch. Initially, this variable is set to 0 (see the block under



The animation shows the Scratch cat sprite starting on the left edge of the display window and saying “howdy”. This is `gamestate == 0`.

Note that in Scratch, you only need to use one equals signs, `=`, to test to see if two values (on either side of the `=` signs) are equal to each other (inside a green block).

The cat moves, along with the text, to the right. This is `gamestate == 1`. When the cat reaches the right edge of the display window, the value of the game state variable changes to 2.

In game state 2, the cat moves, along with the text, to the left. When the cat reaches the left edge of the display window, the value of the game state variable changes to 3.

In game state 3, the cat stops moving and its text message changes from “howdy” to “ciao”. Then the game state changes to 4.

Game state 4 means that the game is over, and the program stops drawing.

Note: in this example, you need to click on the green flag to start the animation .

