Module 3: Create a Sprite from a Graphic (Image)

OVERVIEW:

In this Module, students will: Create a sprite to use as their avatar (game player character or main character) using the graphic image downloaded in the previous Module.

STEPS:

1. In Game Maker, from the top menu choices, choose Resources>Create Sprite (this will bring up the Sprite Properties window).

   Brief Discussion - sprites and objects: A *sprite* is an image that is used for an item or character in the game. A sprite doesn't actually get programmed. An *object* gets programmed (you will create an object in the next Module). A sprite is an image that goes on the object so that the object, which can be programmed, looks a certain way. A good analogy is: A sprite is like a suit of clothes, an object is like the person that wears the suit of clothes. If you want to tell a person to do something, you give instructions to the person, not the clothes. In programming, you give instructions to an object, not a sprite. But you need to put a sprite on the object so it can look like something (a robot, a boulder, a car, etc.). Without a sprite assigned to the object, the object will not be visible in the room when the game is being played.

2. In the Sprite Properties window, click Load Sprite (this will bring up the Open window).

3. In the Open window, navigate to (locate) the graphic image you want to use.

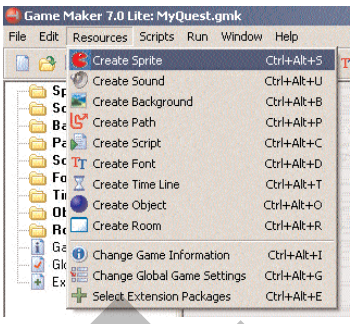
4. In the Open window, single-click on the name (or icon) of the file, and click Open (this will close the Open window and return you to the Sprite Properties window). Note: You can double-click directly on the name or icon of the file instead of selecting the file then clicking the Open button. Either method yields the same results.

5. In the Sprite Properties window, click OK (this will close the Sprite Properties window). You should now see your new sprite at the top of the Library, in the Sprites folder.

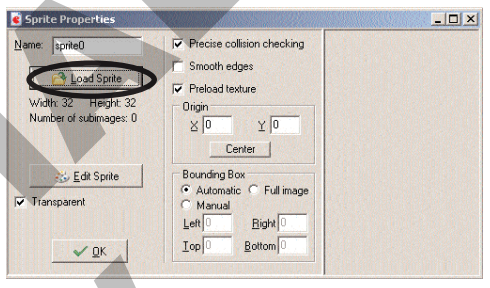
END

VOCABULARY:

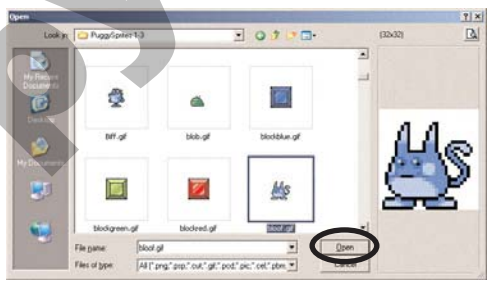
Sprite - An image used as an object or character in a game.
Avatar - Specifically, the sprite used as the player's character in a game.



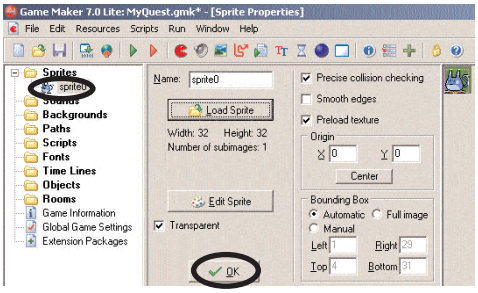
1. Choose Resources > Create Sprite



2. Click Load Sprite



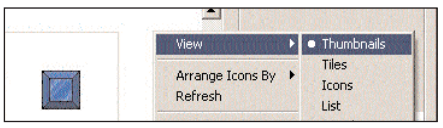
4. Select the file you want and click Open



5. Click OK

VIEW BY THUMBNAIL

When you want to choose an image from within a window (such as in Step 4 in this Module), it's helpful to have a visual preview of image files within the window. To view files as images, right-click directly on any open white area within the preview window, and choose View > Thumbnails.



Module 7: Program Player Character (Avatar) Movement

OVERVIEW:

In this Module, students will: Be introduced to event-driven programming; Program their game character to move via keyboard input from the player; Use graphing principles to control programmed movement.

INTRODUCTORY DISCUSSION:

Note: This Module presents the first application of math in this course. The Module presents the primary way that movement is handled in computer programming. Two-dimensional movement (for 2D games) is achieved by repeatedly increasing or decreasing the x position and/or the y position of an object (where the positioning is determined by coordinate positions on a graph). To move an object left or right, increment or decrement the object's x position; to move an object up or down, increment or decrement its y position; to move an object diagonally, increment or decrement both the x and the y positions at the same time.

Discuss with students: Think about games that you play at home. *Continue along these lines:* If you brought a game home from a store and put it in your console or computer, and saw your game character on the screen, what is probably the first thing you would try to do?" Move the player. **Ask students:** Since you are the programmer, you can decide which keys the player will use to move the game character. On a PC game, what keys are most common for movement? Arrow keys. **Tell students:** That's what we're going to do now. Program the game so the player can use the arrow keys to move their game character (avatar). **Note:** In this particular introductory discussion, students will know the answer to both questions above. This is not an academic topic, and not an academic discussion - it merely serves to orient the students so they begin to realize that the games they are creating are just like games they play at home. The discussion also serves as a way to help students understand the goals in this first, real programming Module.

STEPS:

1. In the Library, double-click the controller object to bring up the Object Properties window for the controller.

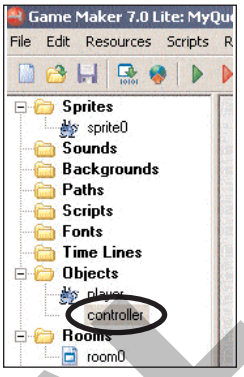
   Note: Make sure the object being programmed is the controller, and not the player object. The game now has two objects, so it's important to make sure students are always aware of which object they are working on. If students followed Step 4 (above) correctly, they should now be working on the controller object. Check the object name field to be sure. (See sidebar on page 16, "What Object are We Currently Programming?")

2. In the Object Properties window, left-click the Add Event button (this will bring up the Event Selector).

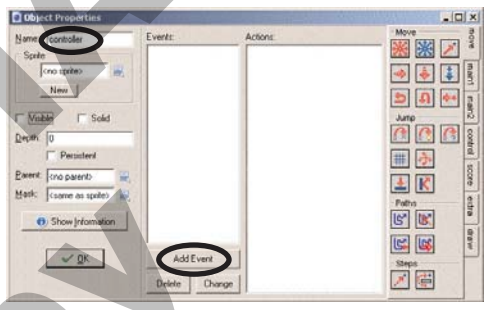
   Brief Discussion: This is the introduction of *event-driven programming*. Explain to students that event-driven programming means programming is based on an ***event*** occurring, and that event causes one or more ***actions*** to occur. For example, an event can be a key being pressed, and the resulting action can be an object moving in a specific direction.

3. In the Event Selector, left-click the Keyboard button (this will bring up the Keyboard events menu).

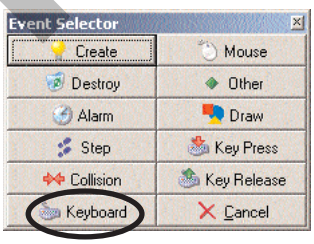
continued on following page



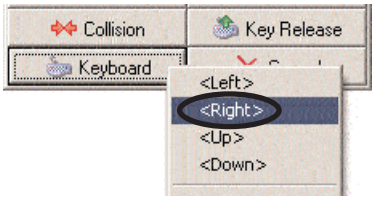
1. Double-click directly on the controller



2. Left-click the Add Event button (note: make sure you're working on the controller object)



3. Left-click the Keyboard button



4. Choose <Right> from the Keyboard events menu

4. The <Right> event now appears in the controller object Events list

Module 7: *Continued from previous page*

Note: In the Events Selector, there are two similar, but very different, events: **Keyboard** and **Key Press**. The difference is: Keyboard is a repeating event (press the given key and the programmed action occurs *repeatedly - as long as the key is held down*), Key Press is a one-time event (press the given key the programmed action occurs *once*).

4. In the Keyboard events menu, choose <Right> (this is the event that occurs when the Right Arrow key is pressed on the keyboard). The <Right> event should now appear in the controller object Events list.

5. On the right side of the Object Properties window are the Action panels (tabs for sections and buttons for actions within each section). Left-click the Move tab (top right corner), and then right-click the Jump to Position button (this will bring up the Jump to Position window).

   Note: The term "jump," as it is used in Game Maker, actually means "move." Even though there are other actions that are specifically called "move," we will use the "Jump to Position" action here. Jump to Position means move from one position (graph point) to another.

6. Set the options in the Jump to Position window (refer to figure 6 at right):

   a. Click the Object button (since we are putting these programming instructions on the controller object, but we want to affect the player object, we have to specify which object we are referring to)

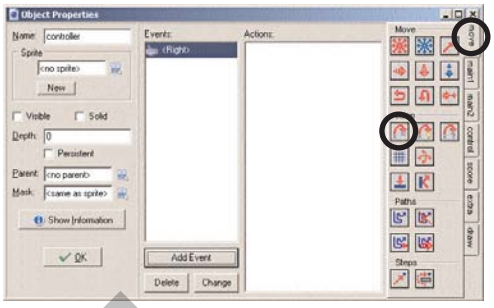
   b. Click the "famous add-thingy button" and choose the player object

   c. Set the x movement to positive 4 (can be written as 4 or +4).
   Important: This is the first significant application of math principles in this project. See ***Discussion on Programming Movement*** at the end of this Module for an important student discussion involving this fun, real-world application of Cartesian Graphing Principles

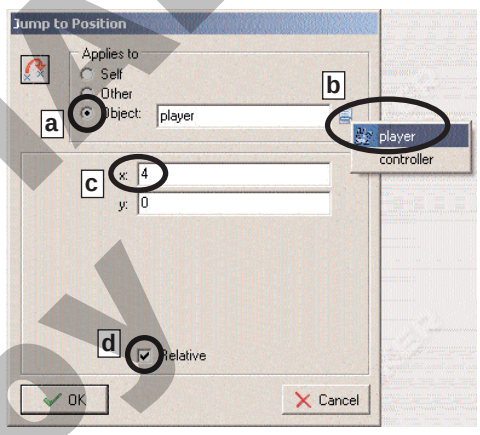
   d. At the bottom of the Jump to Position window, Click the Relative checkbox
   Note: When Relative is checked, the next new x (and/or y) position an object moves to is based on the position *relative* to the object's current position (i.e. +4 means move 4 units to the right from the current position. If an object's current position is x=6, and it moves x+4 *relative*, it's new position is x=10). When Relative is **not** checked, the next new x (and/or y) position an object moves to is based on the position in *absolute* graph coordinates (i.e. +4 means move to x=4 no matter what the current position of the object. If an object's current position is x=6, and it moves x+4 *absolute [not relative],* it's new position is x=4, **not** x=10). This will be a common stumbling point for many students. It's an easy fix, but highly frustrating for students until they understand it. For now, don't worry about explaining this to the students as it may even be overwhelming at this point. Simply instruct students to **make sure Relative is checked** for this Module. The explanation can come during another class when students are more comfortable with the basic material, and when the students are less likely to be overwhelmed by too much information at one time

7. **Test Game!** To test the game, either click the Run Game button (green arrow) at top of the Game Maker screen, or select Run>Run Normally, or press the F5 key on the keyboard.

continued on following page



5. Left-click the Move tab then right-click the Jump to Position button (left-most Jump button)



6. Set the Jump to Position options

7. Click the green arrow to test your game

Very Important Note About Testing Games

Before you have students test their games, remind them: "**What do we expect to happen when we try something new for the first time? Expect it not to work.** If it works that's gravy (bonus). Whenever we try something new, expect it not to work the first time. But be sure that you will get it to work eventually. Maybe not the first time, maybe not the second time, maybe not the hundredth time - but it will work if you stick with it!"

Module 7: *Continued from previous page*

8. **Debug as Necessary** (click the x in the upper right corner of the game window to close the game play mode and return to the game design mode to debug and/or continue working). This is included as a Step in itself, because it is expected that much of the time, debugging will be necessary when adding new programming instructions or game features (see sidebar on previous page, "Very Important Note About Testing Games").

Important Discussion on Programming Movement: At this point in the project, it is an excellent time to discuss x and y planes and coordinates, and other graphing principles. Take this discussion in whatever directions are best for your students at their given level.

9. Repeat Steps 2-7; Change the settings in Steps 4 and 6 as indicated below:
   - Step 4: In the Keyboard events menu, choose <Left>
   - Step 6 (c): Set the x movement to **negative** 4 (-4)
10. **Have students try to program y movement on their own.**

    Note: See sidebar at right, "Which Way is Up?" for an important note about programming y movement.
11. After an appropriate amount of time, go through programming y movement together with students. Repeat Steps 2-7; Change the settings in Steps 4 and 6 as indicated below:

    For Up Movement:
    - Step 4: In the Keyboard events menu, choose <Up>
    - Step 6 (c): Set the y movement to negative 4 (-4)

    For Down Movement:
    - Step 4: In the Keyboard events menu, choose <Down>
    - Step 6 (c): Set the y movement to positive 4 (can be written as 4 or +4)
12. **Test Game!**
13. **Debug as Necessary.**

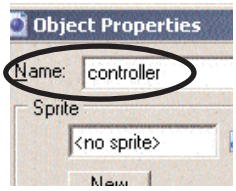
END

VOCABULARY:
Event-driven programming - Programming that is based on an *event* occurring, and that event causing one or more *actions* to occur.
Cartesian Coordinate (Graphing) System - System invented by René Descartes to indicate position in two-dimensional space, based on x and y (horizontal and vertical) positions on a grid or graph.
x-axis, y-axis - Horizontal and vertical planes (respectively) of the Cartesian Coordinate System.

WHAT OBJECT ARE WE CURRENTLY PROGRAMMING?

Students will often think they are programming (or working on) one object, when they are actually programming another. This can, of course, lead to significant problems.

To avoid this confusion, it's important to consistently remind students to make sure they are programming the correct object. The single best way to know which object is currently being programmed is to check the name field of the Object Properties window.



To reinforce this to students, frequently ask students (at random) "What object are we currently programming?" The answer is whichever object's name appears in the object name field. For this particular Module, students should be working on (programming) the controller object.

WHICH WAY IS UP?

In math, y values increase as you go further **up** the y-axis. But in programming, y values increase as you go further **down** the y-axis. Additionally, in programming, the origin of a graph (computer window area) is in the upper left corner, as opposed to in the center as it is in traditional mathematics.

Programmers originally set y values to function this way adn originally set the origin in the upper left to ensure the entire visible area is in positive x and y values.

Note: x-values work the same in math and in programming; that is, they increase when going further right, and they decrease when going further left.

Module 11: Create Projectile Sprite and Object, and Program Projectile Functionality

OVERVIEW:

In this Module, students will: Import art to use as projectiles for the player to throw at adversaries; Create a new sprite and object for use as projectiles; Program functionality for player to throw projectiles in a specific direction (using angles); Discuss basic ethics in dealing with adversaries.

IMPORTANT: This particular Module comprises a number of fundamental academic and social subject areas. At it's fullest, this Module includes: a primary application of math via use of angles to determine direction; an introduction to basic physics via creating projectiles with motion; a fundamental discussion about ethics when dealing with adversaries (in a game and in the real world). This Module starts with a fun and useful discussion with the students, described below.

INTRODUCTORY DISCUSSION:

Tell students: In building this game, there is one main rule: We can stop an adversary but we cannot harm an adversary. Ask students: How can we do this? **Note:** This will yield many interesting answers. It will serve as a fun and engaging discussion with the students, and will also get students on track in terms of how to apply this rule in their game development. After this discussion, tell students that for the first projectiles, the player will throw food at the attacking monsters (under the logic that animals (real and fictitious, i.e. dragons, monsters, etc.) like food,) so **we throw food at them, and that "stops the adversaries without harming the adversaries."**

Ask students: Who decides which key on the keyboard the player will press to throw the projectile? ***answer:*** we do, because we are the programmers.
Ask students: In a PC game, what key is most commonly used to throw projectiles? ***answer:*** Spacebar. **Tell students:** For now, let's all program the spacebar as the projectile key to make the game easy for the player to learn quickly.

STEPS:

1. Download an image to use as projectile; Create sprite; Create object; Name it "proj" (for "projectile" - give students brief definition of "projectile" - see sidebar at right, "Projectiles - Physics Intro").

   Ask students: What two things do we always do when we create an object? - ***answer***: name the object and assign a sprite to the object, using the "famous add thingy button."

2. In the Library, double-click the controller object (this will bring up the Object Properties window for the controller).

3. In the Object Properties window, left-click the Add Event button (this will bring up the Event Selector).

   Ask students: What Event should we choose in this case?
   answer: Key Press (Keyboard can also be used - see important sidebar on next page, "Keyboard or Key Press").

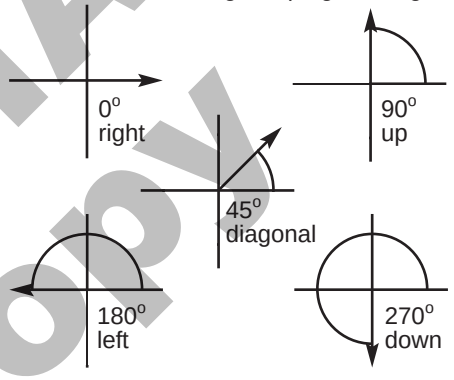
4 In the Event Selector, left-click the Key Press button (this will bring up the Key Press events menu).

5. In the Key Press events menu, choose <Spacebar> (this is the event that occurs

continued on following page

ANGLES AND DIRECTION

In Game Maker and other programming environments, direction is based on the angles within a full 360 degree circle. Right is indicated by 0 degrees (360 will work in many programming environments, but in Game Maker, only 0 degrees can be used to specify right as a direction). Left is indicated by 180 degrees; Up is indicated by 90 degrees; Down is indicated by 270 degrees. All incremental angles between 0 and 359 are usable - this allows very fine control of direction of motion in game programming.



OTHER WAYS TO INDICATE DIRECTION

In computer programming, there are two primary ways to specify direction: Degrees (based on angles) and Radians. Radians are more commonly used in higher-end programming and other mathematical and scientific applications. But, for purposes of class discussion and gameplay design, discuss with students the numerous other ways to indicate direction, such as: Compass directions (N,S,E,W); Clock directions (6 o'clock to indicate behind, 12 to indicate in front, 9 to indicate right, etc.); Longitude and latitude; most basic - left, right, up, down. There are other ways to indicate direction, but these examples should provide good jump-off points for class discussion.

PROJECTILES - PHYSICS INTRO

In Level 1, students do not apply forces such as gravity, friction, acceleration, etc. to moving objects. However, it's important to at least briefly discuss how moving objects (on earth) are affected by these forces. In Level 2, students apply these forces to moving objects. The Level 1 discussion and Level 2 application provide a good early introduction to the fun of physics.

Module 11: *Continued from previous page*

when the Spacebar is pressed). The <Spacebar> event should now appear in the controller object's Events list.

6. On the right side of the Object Properties, in the Action panels, left-click the control tab (fourth tab from top), and then right-click the Test Instance Count button (this will bring up the Test Instance Count window).

7. Set the options in the Test Instance Count window as follows and click OK:

   a. object: player

   b. operation: Larger than

8. On the right side of the Object Properties, in the Action panels, left-click the main1 tab (second tab from top), and then right-click the Create Moving button (this will bring up the Create Moving window).

9. Set the options in the Create Moving window (refer to figure 7 at right):

   a. Click the "famous add-thingy button" and choose the projectile (proj) object

      Ask students: Where should projectiles be initially created (stated differently, where should the projectiles first appear)? ***answer:*** on (or from) player. **Ask students:** How do we define the player's position mathematically? ***answer:*** in terms of Cartesian coordinates - in this case, specifically, player.x, player.y

   b. Set x and y creation points to player.x and player.y, respectively

   c. Set the speed of the projectiles (the speed setting is purely a design choice - for now set it at between 10 and 20

      Note: If speed of projectile is set beyond a certain speed, the player will never see the projectile, because it will be off the screen before it can even be seen. For now, have students keep the speed setting to lower than 20.

   d. Set the direction to 0, which means the direction will be to the right (see sidebar on previous page, "Angles and Direction")

10. Test Game!

    Remind students: We're about to try something new — what do we expect to happen? Expect it not to work. If it works, that's gravy.

11. Debug as needed.

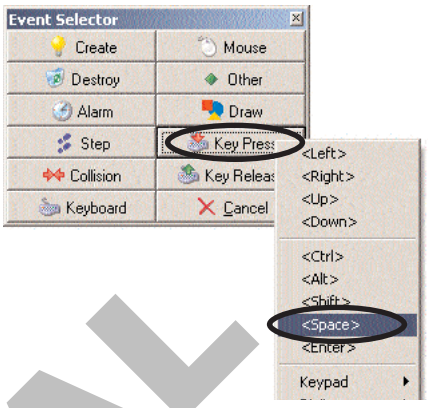
    Important Note: Students will notice that nothing happens when the projectiles hit the adversaries. **Ask students:** Why doesn't anything happen when the projectiles hit the adversaries? ***answer:*** we haven't programmed it yet. (We'll program the collision event in the next Module.)

12. **Optional (and fun):** Have students try to program projectiles to be thrown in all four straight directions: up, down, left, right (have students use W,A,S,D keys on keyboard). The respective directions (in degrees) for up, down, left, and right are 90, 270, 180, and 0.

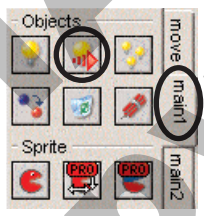
END

VOCABULARY:

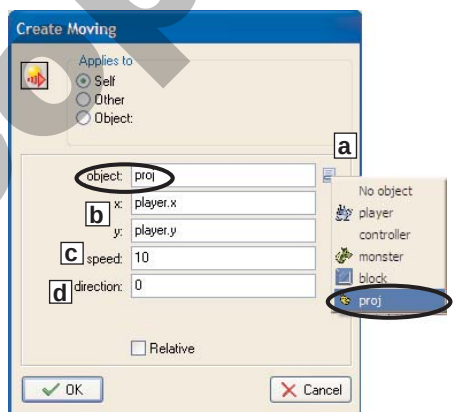
Projectile: An object that travels through space, usually (though not always) increasing in speed at first, then decreasing in speed over time.
Degrees: A mathematical unit of measure to represent various angles.



4,5. Choose <Space> from the Key Press menu



6. Left-click the Main1 tab and then right-click the Create Moving button



9. Set the options in the Create Moving window

KEYBOARD OR KEY PRESS

There is a subtle, but important difference between the **Keyboard** event and the Key Press event.

The Keyboard event is a repeating event. That means that the action(s) associated with the event happen continually, as long as the given key is held down. **The Key Press event,** on the other hand, is a one-time event. This means that the action(s) associated with the Key Press event happen only once when the key is pressed, regardless of how long the key is held down. To trigger Key Press action(s) again, the user must release the key and press it again.

In short: Key Press is a one-time event, Keyboard is a repeater.