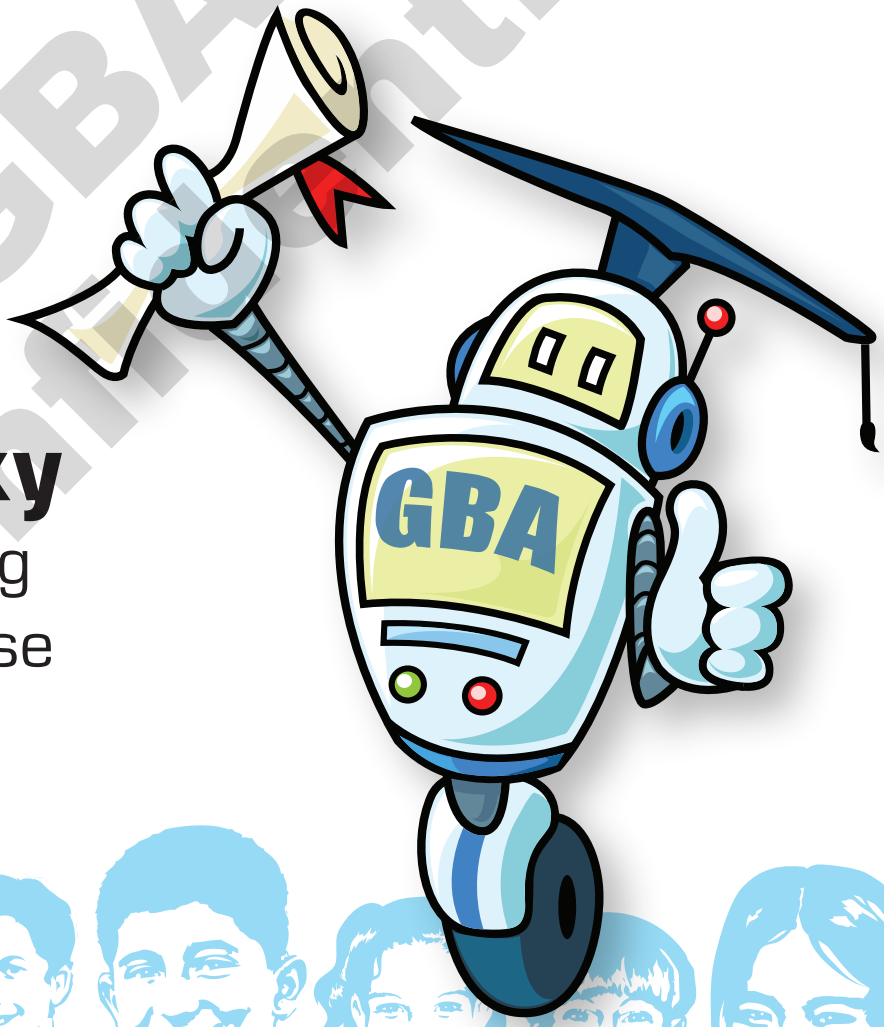
Game Builders Academy™

Learn ★ Grow ★ Have Fun ★ Succeed!

Video Game Design and Development

Level 1



Phil Lipsky
Sally Rosenberg
Michael Pugliese

TEAM GBA CORP. 35 Lace Lane, Westbury, New York 11590

Game Builders Academy™

Learn ★ Grow ★ Have Fun ★ Succeed!

Video Game Design and Development, Level 1
written by Phil Lipsky

Development Editors
Phil Lipsky, Michael Pugliese

Production Editor
Sally Brecher Rosenberg

Box/Binder Design
Michael Pugliese

Marketing and Sales
Walter Ebe, John Taylor

ISBN-13:
978-0-9815502-0-6

TEAM GBA CORP. 35 Lace Lane, Westbury, New York 11590

Project 1 Overview: MyQuest

The game that students will create is referred to as a top-down, 2-dimensional, action game. One of the earliest examples of this type of game is the original Legend of Zelda on the Nintendo Entertainment System.

Students' games will have either two or three levels (depending on time available and students' own game design choices). The game will be complete with a Game Start screen and Game Over screen, and fully developed art and gameplay functionality.

In the first level of the game, the player will be located in a room in a castle and will be faced with challenging adversaries that will chase the player. When adversaries touch the player, the player's health will be reduced, and when the player's health is fully depleted, the game ends. The player will be able to throw (food) projectiles at the adversaries to make them disappear, and the player's score will also increase when the (food) projectile touches the adversary. After defeating all the adversaries, the player will have to find the opening in the wall (sliding tiles) through which he or she can escape from the current room (level), and proceed to the next room (level).

In the second level of the game, the player will have to face regular adversaries, along with a "boss" monster - a large, very challenging adversary. To win the game, the player must defeat the boss monster. Along the way there will opportunities for the player to obtain "pickups" to restore health, add additional capabilities or score, etc. Students will be able, time permitting, to add additional features, art, sound, and animation effects. At the end of the course, students will create an "executable" file of their game. This is a true, PC gaming format file, which can be then played on any PC, with or without Game Maker being installed. Students can also post their games on the web or send them via email for friends and families to see and play.

.

Lesson 1 Overview

Modules 1-10

Module 6: Create a Controller Object and Add Controller to Room

OVERVIEW:

In this Module, students will: Create an object that will hold programming instructions for other objects within the game.

Note: In this Module and future Modules, where the details of a given Step have been covered in a previous Module, full details will not be repeated. Only the Step will be listed. Refer back to previous Modules for reference if necessary. For example, Step 1 below is to create a new object, and the details of creating an object were covered in Module 4, so the instructions will not be repeated here. A reference to the appropriate Module will be included within the given Step.

STEPS:

1. Choose Resources>Create Object. In the Object Properties window, name the object "controller"; **do not apply a sprite** (refer to Module 4).

   Brief Discussion - Controller Object: Programming instructions must be set on an object. Any object can hold programming instructions for itself, or one object can hold programming instructions for another object. In general, we want to keep our programming instructions consolidated on as few objects as possible. One very good way to implement this method is to create a separate object that is invisible to the game player, yet still exists in the game and carries programming instructions that run during the game. This object is unique because **there is no sprite applied to it**. We will name this object "controller" because it controls other objects and events.

2. In the Object Properties window, left-click the Persistent checkbox.

3. In the Library, double-click room0 to bring up the Room Properties window.

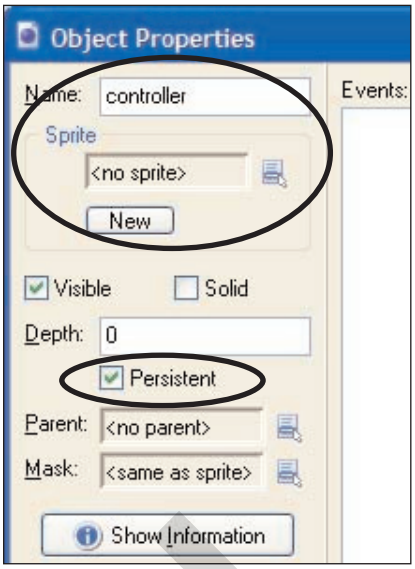
4. Put the controller object in the room (refer to Module 5).

   Note: The controller object will appear as a blue sphere with a question mark inside it. This is because no sprite was applied when the object was created. That is proper. In this case, we do not want the object (controller) to be visible to the player, so we do not apply a sprite. There are other ways to make an object invisible, but the method of not applying a sprite is best in this situation. It does not matter where you place the controller object within the room, because it will not be seen by the player when the game runs.

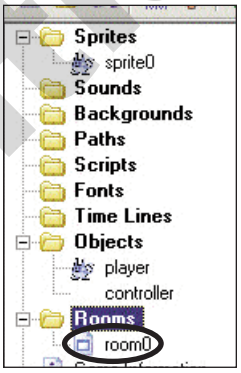
END

VOCABULARY:

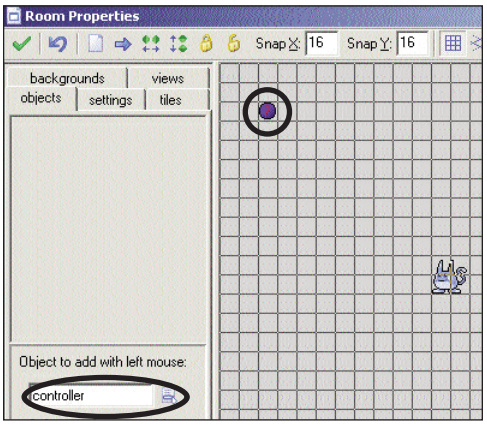
Program - A series of instructions; a sequence of events.
Object - Something that can be programmed (given instructions).



1,2. Name the object "controller" and **do not** apply any sprite. Left-click the Persistent checkbox



3. Double-click directly on room0



4. Place the controller object in the room (it will appear as a blue sphere with a question mark inside it)

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.

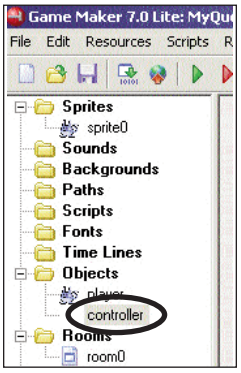
   Note: Make sure the object being programmed is the controller, and not the player object. The game now has two objects, so it's important to make sure students are always aware of which object they are working on. If students followed Step 4 (above) correctly, they should now be working on the controller 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).

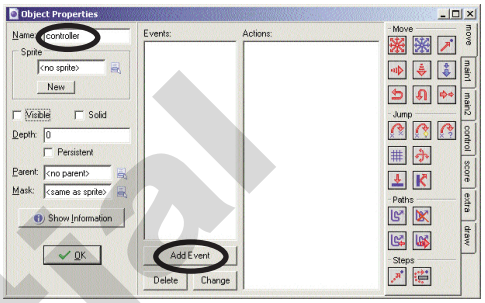
   Brief Discussion: This is the introduction of *event-driven programming*. Explain to students that event-driven programming means programming is based on an ***event*** occurring, and that event causes one or more ***actions*** to occur. For example, an event can be a key being pressed, and the resulting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

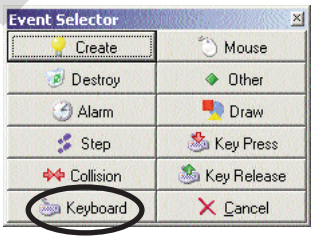
continued on following page



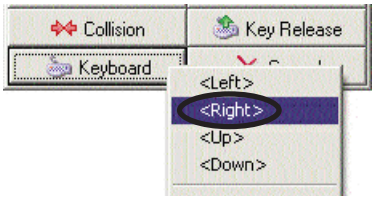
1. Double-click directly on the controller



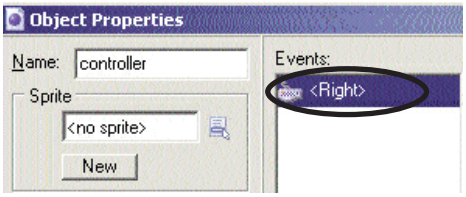
2. Left-click the Add Event button (note: make sure you're working on the controller object)



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)

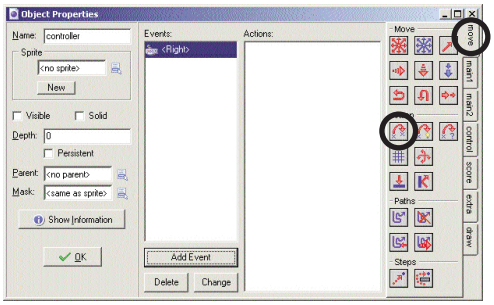
   b. Click the "famous add-thingy button" and choose the player object

   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

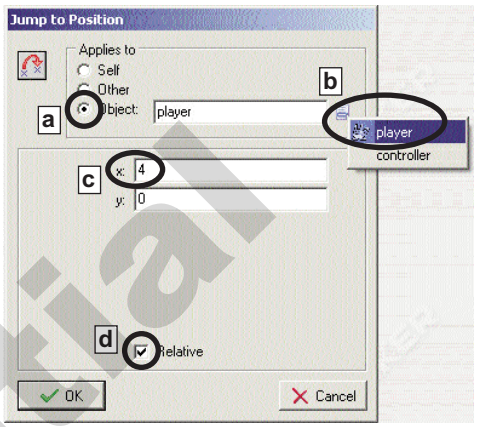
   d. At the bottom of the Jump to Position window, Click the Relative checkbox
   Note: When Relative is checked, the next new x (and/or y) position an object moves to is based on the position *relative* to the object's current position (i.e. +4 means move 4 units to the right from the current position. 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 and 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 8: Add Adversaries and Script Basic Artificial Intelligence (AI) for Adversaries

OVERVIEW:

In this Module, students will: Import art to use as adversaries; Create a new sprite and create a new object to use as adversaries; Add the adversaries to their games; Program basic Artificial Intelligence (AI) for adversaries to "chase" the player.

STEPS:

1. Download an image to use as an adversary; Create sprite; Create object; Name object "monster" (refer to Modules 2,3,4).

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

2. In the Library, double-click room0 to bring up the Room Properties window.

3. Add multiple adversaries (between 3 and 10) in room0 (refer to Module 5).

4. Test game.

   Important Discussion - note no movement of adversaries at this point:
   Ask students: Why don't the adversaries move? - we haven't programmed it yet. A computer or a computer program only does what we tell it to do - this is a vital point, and it will be repeated and reinforced throughout the classes. The computer/program only does or does not do what we tell it to or what we don't tell it to do. This is important for teaching student that they have full control over what happens in their programs; and also important for debugging, to realize that whatever is happening is due to our programming - something we have done, or something we have not done.

5. Close the game play window and return to game design mode (click the "x" in the upper right of the game play window to close the play mode).

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

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

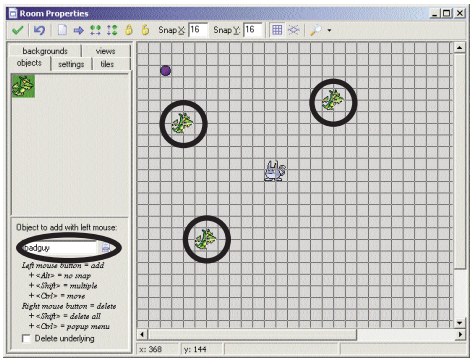
8. In the Event Selector window, left-click Step and choose Step from the flyout menu (choose Step, not Begin Step or End Step).

   Brief Discussion - Step Event: Explain to students that, in Game Maker, the Step event is a constantly running, regularly repeating event
   Ask students: What is this akin to in the human body? ***answer:*** heartbeat, et al.

9. On the right side of the Object Properties window, in the Actions panels, Left-click the Move tab (top right corner), and then right-click the Step Avoiding button (this will bring up the Step Avoiding window).

   Tell students: This will make the adversaries Step towards a certain point while avoiding other obstacles.
   Ask students: What point do we always want the adversaries to be moving towards? ***answer:*** on the player's current position - defined as player.x, player.y.

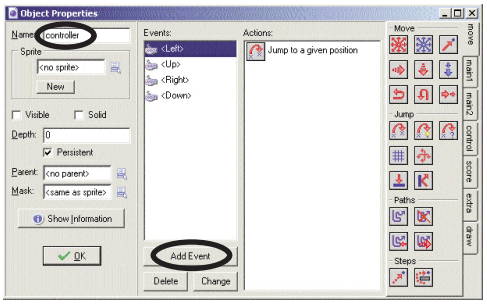
continued on following page



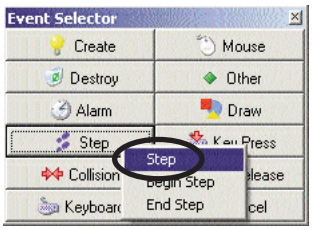
3. Add adversaries (monster) to the room

DON'T PUT TOO MANY ADVERSARIES IN THE ROOM

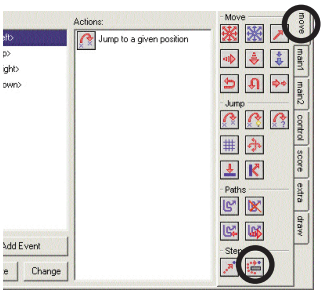
Some students will want to add many adversaries in the room (dozens or more). This can be done later in the project if desired, but at this early stage, it will cause problems. The computer may not be able to handle that many objects at once, and thus will slow down or crash. Additionally, in later Modules, when students add more items to this level (room), an inordinately high number of adversaries in the room will cause additional problems. In short, have students add no more than 10 adversaries for now.



7. Left-click the Add Event button



8. Choose the Step action from the Step Event menu



9. Left-click the Move tab and then right-click the Step Avoiding button

Module 8: *Continued from previous page*

10. Set the options in the Step Avoiding window (refer to figure 10 at right):

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

    b. Click the "famous add-thingy button" and choose the adversary (monster) object

    c. **Ask students (again):** What point do we always want the adversaries to be moving towards? ***answer:*** player's current position - defined as player.x, player.y

    Set the x and y position (towards which the object steps) to player.x and player.y, respectively

    d. Set the Speed to 2 (this keeps the adversaries moving slower than the player, thus giving the player a chance to escape adversaries by moving away - this is a gameplay choice at the moment, not a technical necessity)

    e. **Note:** Avoid is set to solid only. This will be useful to remember because it will be brought up again in an upcoming Step

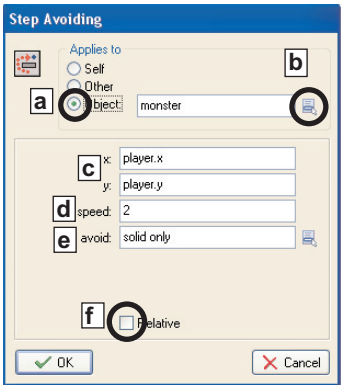
    f. At the bottom of the Step Avoiding window, uncheck the Relative checkbox
    Note: Relative should be **checked for the player** movement (Module 7), and Relative should be **unchecked for the adversary**.

11. 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.

12. Debug as needed

END



10. Set the options in the Step Avoiding window

VOCABULARY:

Adversary: An opponent: enemy
Artificial Intelligence (AI) - A broad term, most basically defined as giving non-animal entities (computers, robots, video game characters, etc.) the ability to make decisions in a logical way.
x and y (graph) coordinates - The x and y position of an object in two-dimensional space, the current position of an object.

LESSON 2 OVERVIEW

Modules 11-15

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").

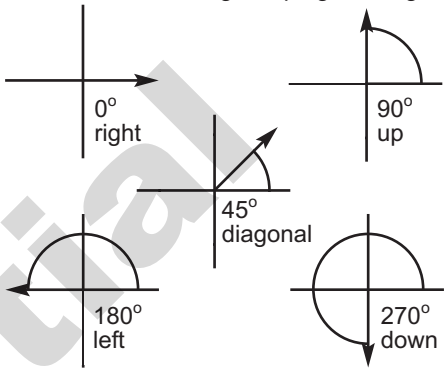
4 In the Event Selector, left-click the Key Press button (this will bring up the Key Press events menu).

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 9 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.

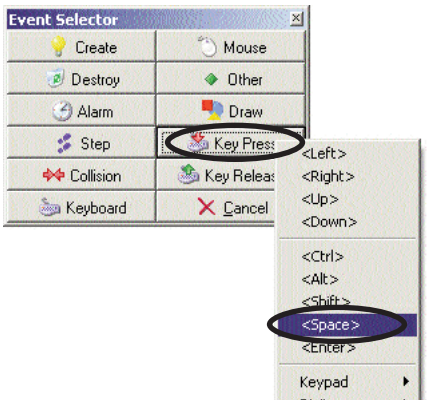
    Important Note: Students will notice that nothing happens when the projectiles hit the adversaries. **Ask students:** Why doesn't anything happen when the projectiles hit the adversaries? ***answer:*** we haven't programmed it yet. (We'll program the collision event in the next Module.)

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.

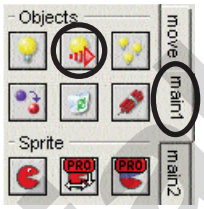
END

VOCABULARY:

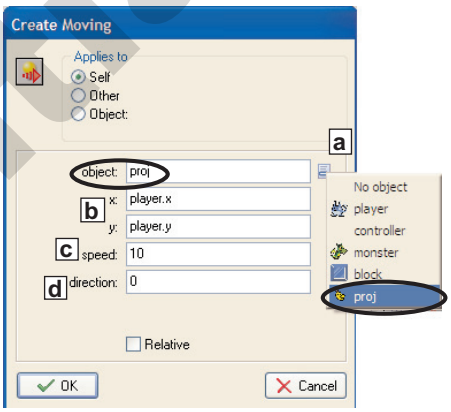
Projectile: An object that travels through space, usually (though not always) increasing in speed at first, then decreasing in speed over time.
Degrees: A mathematical unit of measure to represent various angles.



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



8. 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.

Module 12: Program Collision Event for Projectiles and Adversaries, and Projectiles and Blocks

OVERVIEW:

In this Module, students will: Program event of collision between projectile and adversary.

Ask students: What should happen when a projectile hits an adversary (tell students to think about games they play at home)?: ***answer:*** the adversary should disappear and the projectile should disappear. Additionally, scores can be incremented, sounds can play, and other effects and events can occur (we will add score functionality and sound in the next three Modules). Tell students that for now, we will start by programming the projectile and adversary to both disappear upon contact with each other (collision).

STEPS:

1. In the Library, double-click the proj (projectile) object (this will bring up the Object Properties window for the proj object).

   Note: Make sure students are programming the proj object (see sidebar on page 16, "What Object are We Currently Programming?").

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

   Ask students: Which event should we choose? ***answer:*** Collision.

3. In the Event Selector window, left-click Collision and choose monster from the flyout menu (since we are currently programming the projectile object, the other object in the collision is the monster, so we choose monster from the Collision flyout menu).

4. On the right side of the Object Properties window, in the Actions panels, left-click the main1 tab (second tab from top).

   Note: Have students look at the actions available in the main1 set of actions, and have them try to figure out which action should be used – ***answer:*** Destroy Instance (recycle bin icon). Students need to start exploring the various actions to become more familiar and comfortable working in Game Maker. From this point forward, students should be trying to solve problems on their own as often as possible. Give students answers only as needed, after students have looked at the various possibilities and offered suggestions.

5. In the main1 actions, right-click the Destroy Instance button (this will bring up the Destroy Instance window).

6. In the Destroy Instance window, leave the settings as they are ("Self" is checked) and click OK.

   Ask students: What is meant by "Self?" ***answer:*** "self" is always the object we are currently programming. **Ask students:** What object are we currently programming? ***answer:*** proj (see Step 1) - And since, proj is the object are we currently programming, "Self" is proj (this becomes easier to understand over time).

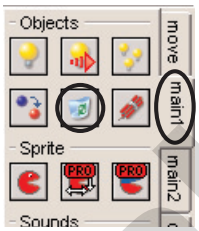
7. Test Game!

   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.

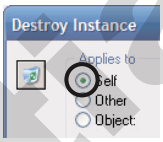
8. Debug as needed.

continued on following page

3. Choose monster from the Collision menu



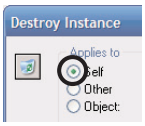
4,5. Left-click the main1 tab and then right-click the Destroy Instance button (recycle bin icon)



6. Leave the Applies to option set to "Self"

DESTROY INSTANCE: SELF, OTHER, OR OBJECT?

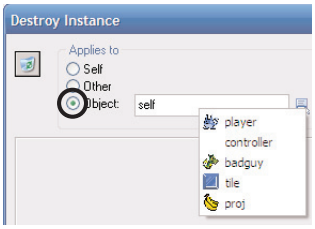
In the Destroy Instance window, there are three choices: Self, Other, and Object. Here's what they mean:



Self: The object currently being programmed.

Other: The other object involved in the collision (the object that touches, or is touched by, "self."

Object: If you click this choice, a menu appears containing all objects in the game.



Specify which object should be destroyed when the collision event occurs. **Note: All** instances of the chosen object are destroyed when you choose "Object."

In general, to make two objects disappear when they contact each other, **program one of the objects in the collision, with two separate actions: Destroy Self, and Destroy Other.**

Module 12: *Continued from previous page*

9. **Ask students:** Even though projectiles are now disappearing when they hit the adversaries, what is still **not** happening that should be happening? ***answer:*** adversaries are not disappearing **Ask students:** Why doesn't anything happen when the projectiles hit the adversaries? ***answer:*** we haven't programmed it yet.
10. Create a second Destroy instance action: In the main1 actions, right-click the Destroy Instance button (this will bring up the Destroy Instance window).
11. In the Destroy Instance window, choose "Other" and click OK.

    Ask students: What is meant by "Other?" ***answer:*** the "other" object in the collision (**not** "Self") (see sidebar on previous page, "Destroy Instance: Self, Other, or Object?" and on this page, "Either Object in the Collision can be Programmed").

    Important Note: When you choose "Object" in the Destroy Instance window (which we did not do here), **all instances of the chosen object are destroyed.** So, in Step 11, if students choose Object>monster (instead of "Other" as we are doing), **all** monsters would disappear when a projectile hits **any one** monster. The "Other" choice (which is what we chose here), destroys **only the single instance** that is involved in the collision, not all instances of the object. If a student tries his or her game and all of the monsters disappear when only one monster is hit with a projectile, the student has probably chosen Object>monster instead of Other in Step 11.
12. 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.
13. Debug as needed.
14. **Ask students:** What should happen when a projectile hits a block? ***answer:*** the projectile should be destroyed (this is a gameplay choice, not a technical necessity - some students may suggest that projectiles go through blocks and create holes, or should "splat" against the block, etc. - tell students that all this can be added later, but for now, let's just make the projectiles disappear when they hit a block). **Ask students:** Why doesn't anything happen now when the projectile hits the block? ***answer:*** we haven't programmed it yet.
15. **Students to try:** Have students try to program the projectile/block collision event on their own. Give them a few minutes to try this, then walk through it with them.
16. Repeat Steps 1 - 6, in Step 3 choose "block" from the Collision menu (instead of monster as we did in Step 3 the first time through).
17. 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.
18. Debug as needed.

END

Vocabulary:

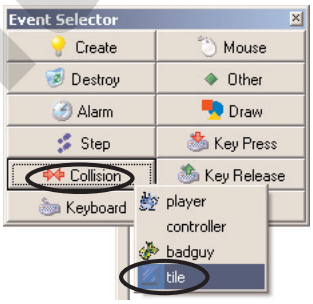
Collision (in context of programming): A collision is the event of any two objects touching each other (coming in contact with each other).

Either Object in the Collision can be Programmed

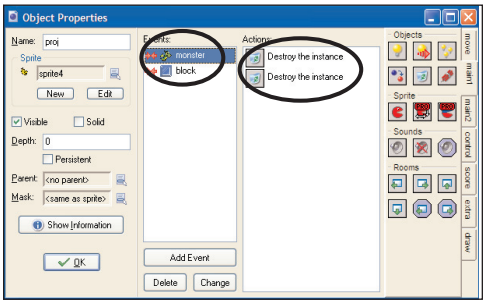
A collision event involves two objects touching each other. When you program a collision event, the programming instructions can go on **either** object in the collision.

The object you choose to program will be "Self" and the other object in the collision will be "Other" - so it doesn't really matter on which object you choose to place the programming instructions.

One object will always be "Self" and the other will always be "Other" - no matter which you choose. Think the logic through, and it should become apparent, even if it is confusing at first. (See sidebar on previous page, "Destroy Instance: Self, Other, or Object").



16. Choose block from the Collision menu



The Object Properties window should look like this after Step 16. There are two separate collision events: one between proj and monster, and the other between proj and block. The monster collision should have two separate actions on it: Destroy Self, and Destroy Other.

Module 13: Create and Display Player Score

OVERVIEW:

In this Module, students will: Add functionality to create and display the player's score.

Ask students: What else should happen when a projectile hits an adversary (tell students to think about games they play at home)?: ***answer:*** player score should increase. (This is a gameplay choice, not a technical necessity. In many games, the player's score increases when an adversary is defeated - this is an extremely common gameplay element.)

STEPS:

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

   Note: Make sure students are programming the controller object (see side-bar on page 16, "What Object are We Currently Programming?")

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

   Explain to students: The Create event is a one-time event (runs once), when the game (or game level) starts. The Create event holds programming instructions for what should happen when the game level is first created -- or when the game first begins. Tell students that since we want the score to appear immediately when the game starts (when the first level of the game is created), we will use the Create event.

3. In the Event Selector window, left-click Create.

   Ask students: Since we are trying to program the score, which tab (in the Actions tabs) do you think we should click? Have students look at the Actions panel and have them try to figure out which set of Actions (which tab) should be used. ***answer:*** score.

4. On the right side of the Object Properties window, in the Actions panels, left-click the score tab (third tab from bottom).

   Note: Have students look at the actions available in the score set of actions, and have them try to figure out which action should be used – ***answer:*** Set Score.

5. In the score actions, right-click the Set Score button (this will bring up the Set Score window).

6. In the Set Score window, leave the settings as they are (score is 0). Click OK.

   Ask students: What should the score be when the game first begins? ***answer:*** 0. This is set by default (when the Set Score window comes up), so, in this case, students do not have to make any changes in this window.

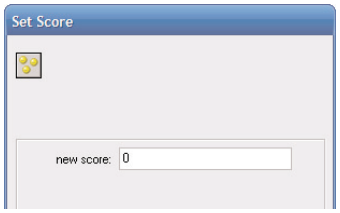
   Important Note: Steps 1-6 just **create** a score (actually a score variable), but they do not **display** the score. The score will not show until you add a **Draw event** and **Draw action** to display the score. Here's the tricky part - ***the Draw action must go on the Draw event, not on the Create event.*** Follow the next Steps precisely and everything should function as expected.

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

continued on following page

3. Left-click the Create button

4,5. Left-click the score tab and then right-click the Set Score button



6. Leave the settings as the are when the window comes up, and click OK

Challenges and Rewards

An increased score is just one way to reward a player for accomplishments during the game. There are many different ways to reward the player.

Challenges, rewards, and satisfaction gained by attaining rewards are a tremendously important part of good gameplay design. The greater the challenge for the player (up to a certain point), the more satisfying it is simply to conquer the challenge, solve the puzzle, etc. Additionally, there is more satisfaction gained from the rewards achieved (in the form of visual effects, sounds, additional capabilities, access to new levels, etc.).

Module 13: *Continued from previous page*

Ask students: Since we have already created the score in the last Step(s), what Event do you think we need to use now to **draw** the score? ***answer:*** Draw.

8. In the Event Selector window, left-click the Draw button.

   Ask students: Since we are trying to draw the score, what action (in the available score actions) do you think we should choose? Have students look at the score actions and have them try to figure out which action should be used – ***answer:*** Draw Score.

9. In the Actions panel, on the right side of the window, right-click the Draw Score button (this will bring up the Draw Score window).

10. In the Draw Score window, set both the x and the y to 50, and leave the caption as "Score."

    Ask students: What information do you think this window is asking for when it asks for an x value and a y value? ***answer:*** the position where the score will be drawn, based on x and y coordinates (there's that graphing stuff again!)

11. Test Game!

12. Debug as needed (until score shows in upper left).

    Ask students: Even though the score now shows, it does not increase when the projectiles hit the adversaries - why not? ***answer:*** we haven't programmed it yet. (We'll do that in the next Module.)

    Have students: Experiment with different x and y values in the Draw Score window to position the score in different areas of the room (upper right, lower right, etc.)

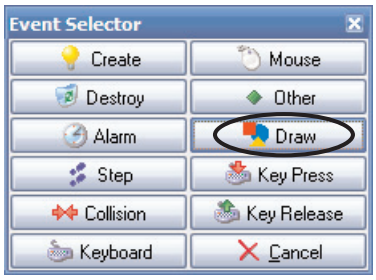
END

Discussion Topic:

What are some other ways, aside from score, to reward a player for accomplishing tasks and goals within the game?

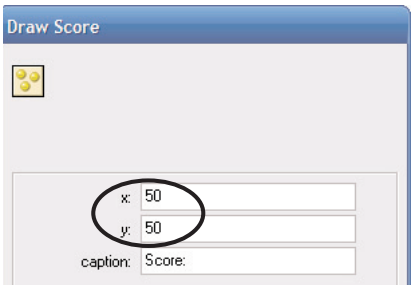
Vocabulary:

Reward: Something favorable received for accomplishing a task or goal.
Satisfaction: The good feeling gained from accomplishing a task or goal (note, that by definition, satisfaction itself is a reward).



8. Left-click the Draw button

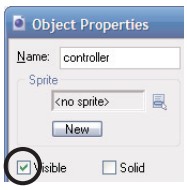
9. Right-click the Draw Score button



10. Set both the x and the y position to 50, and leave the caption as "Score"

Why isn't my score showing?

If a student's score is not showing up when he or she tests the game, check the following:



- Make sure the controller is set to Visible
- Make sure the Draw action is set on the Draw event and **not** on the Create event (be aware that this one in particular can cause confusion)
- Make sure the controller is in the Room

Module 14: Program Scoring Functionality

OVERVIEW:

In this Module, students will: Add functionality to increment the player's score when projectiles hit adversaries.

Remind students: Even though the score now shows, it does not increase when the projectiles hit the adversaries - why not? ***answer:*** we haven't programmed it yet. (We'll do that now.)

Brief Discussion - Introduction to Pseudocode: Explain to students that pseudocode is used by the developers of all the games they play at home. Before trying to program something to happen in a game (or other type of computer program), it's often helpful to think through the programming Steps using a spoken language (English, Spanish, Chinese, etc.). Once the programming Steps have been defined in plain language, then those Steps are translated into programming code (instructions). Note: There is no "right" pseudocode. The only purpose of pseudocode is to think clearly through, and define, the Steps in the programming task at hand.

Have students: Try to pseudocode the scoring functionality. (This is new to most students, if not all. Let students think about it for a few moments). Tell students there is no "right" pseudocode - pseudocode is just words that describe how to accomplish a programming task.

> **Example of pseudocode for this particular task:**
> When the projectile hits the monster, increase the player's score.
>
> **Ask students:** Now that we have the pseudocode, we take the first part, "When the projectile hits the monster" - do we have that event already in our programming? ***answer:*** yes, on the projectile object (we did this in Module 13). We have a collision event between the projectile and the monster. That is exactly "When the projectile hits the monster" so we return to that same object and event and we add to it.

STEPS:

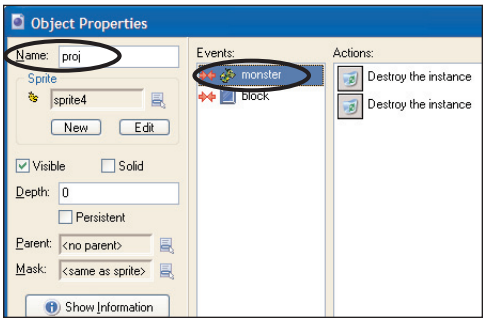
1. In the Library, double-click the proj object (this will bring up the Object Properties window for the proj object).

   Note: Make sure students are programming the proj object (see sidebar on page 16, "What Object are We Currently Programming?")

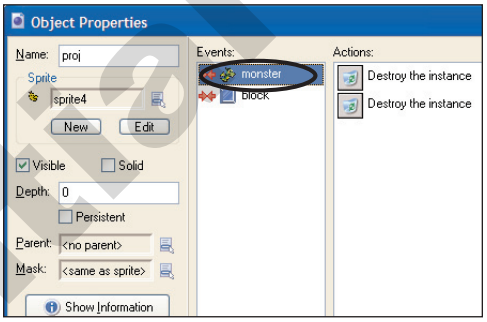
2. In the proj Object Properties window, notice that we already have a Collision event for the proj (object we're currently programming) and the monster. We will add another action to that existing event. In the Events list of the proj Object Properties, left-click on the monster collision event.
3. On the right side of the Object Properties window, in the Actions panels, left-click the score tab (third tab from bottom).

   Have students: Look at the actions available in the score set of actions, and have them try to figure out which action should be used. ***answer:*** Set Score.

continued on following page



2. Notice that on the Object Properties for the proj object, we already have a collision event between the projectile and the monster (the first part of our pseudocode - "When the projectile hits the monster")



2. Left-click on the monster Collision event

3,4. Left-click the score tab and then right-click the Set Score button

Module 14: *Continued from previous page*

4. In the score actions, right-click the Set Score button (this will bring up the Set Score window).

   Ask students: What should we do to the value of the score every time a projectile hits an adversary? **answer:** add to the current score (let students add however much they like for each "hit" - in this example we've used 10).

5. In the Set Score window, set the "new score" to +10 and click the Relative checkbox at the bottom of the window, then click OK.

6. Test Game! (See if score increases when projectiles hit adversaries)

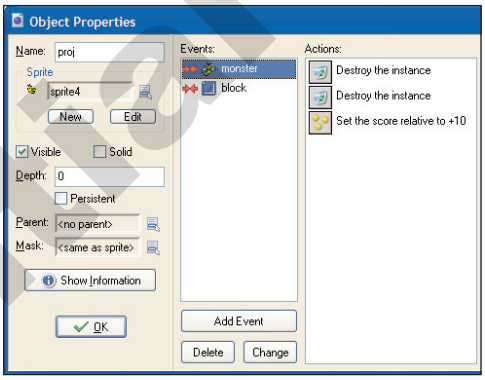
   Remind students: We're about to try something new — what do we expect to happen? Expect it not to work. If it works, that's gravy.

7. Debug as needed

END

VOCABULARY:
Pseudocode: Programming instructions defined in a spoken language.

5. Set the "new score" value to +10, and click the Relative checkbox



The Object Properties window should look like this after Step 5

RELATIVE AND ABSOLUTE

The Relative checkbox appears in a number of windows in Game Maker. In general, Relative means that any values set will be added to or subtracted from current values.

For example, if a value (score, amount of heath, x position, etc.) is currently 10, and we set a change to +5 **Relative**, the new value is 15.

However, if a value (score, amount of heath, x position, etc.) is currently 10, and we set a change to +5 **not Relative** (absolute), the new value is 5.

Absolute (not Relative) does not take into account any existing information. Relative does take into account existing information. New values are **relative** to previous values.

Remember, many aspects of game programming are confusing at first, but they will become more fluent with time.

Module 15: Import Sound and Program Sound Functionality

OVERVIEW:

In this Module, students will: Download a sound; Import the sound into the game; Program functionality for the sound to play each time the player throws a projectile.

Tell students: We will add more sounds to our games over time, including background music, but to start, we'll make a sound play whenever the player throws a projectile. The first thing we need to do is to import a sound into Game Maker.

STEPS:

1. Choose Resources>Create Sound (this will bring up the Sound Properties window).

   Tell students: Look in the Resources menu at the top of the Game Maker screen - which choice do you think we should make if we want to add a sound? ***answer:*** Resources>Create Sound.

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

3. In the Open window, navigate to the folder that contains the available sounds (either the Sounds folder in the Game Maker program folder, or the GBA Sounds folders on the included disks). Left-click on a sound in the Open window and click the Open button.

   Tell students: We have now imported the sound into Game Maker, but we still have to program the sound to play at a specific time.

   Have students: Try to pseudocode the playing of the sound.
   Example of pseudocode for this particular task:
   When the player throws the projectile, play the sound.

   Ask students: Now that we've got the pseudocode, we take the first part, "When the player throws the projectile" - do we have that event already in our programming? ***answer:*** yes, on the controller object (we did this in Module 12). We have a Key Press Spacebar event. That is "When the player throws the projectile."

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

5. In the controller Object Properties window, in the Events list, left-click on the Key Press Spacebar event (press <Space>).

   Note: If students have programmed the WASD keys, they should select each key in the Events list and follow the remaining Steps for the W, A, S, and D keys.

6. On the right side of the Object Properties window, in the Actions panels, left-click the main1 tab (second tab from top).

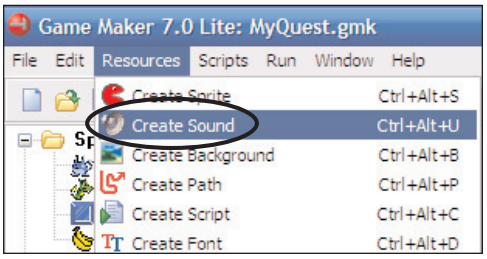
   Note: Have students look at the actions available in the main1 set of actions, and have them try to figure out which action should be used – ***answer:*** Play Sound.

7. In the main1 actions, right-click the Play Sound button (this will bring up the Play Sound window).

8. In the Play Sound window, left-click on the "famous add-thingy button" and choose sound0 (that's our new sound) from the menu, click OK (leave loop set to false).

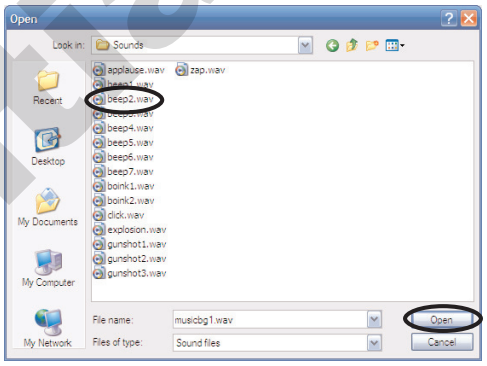
END 

END LESSON 2 

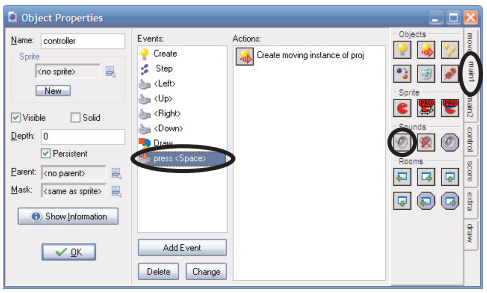


1. Choose Resources>Create Sound

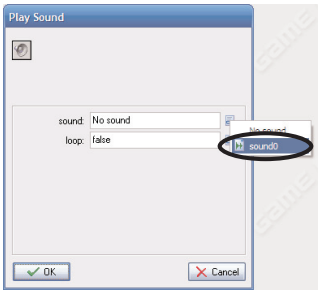
2. Click the Load Sound button



3. Left-click on a sound in the window and then click the Open button



5,6,7. Left-click on the press <Space> event, then left-click on the main1 tab, then right-click on the Play Sound button



8. Left-click directly on the "famous add-thingy button" and choose the sound to play

Project 1 Optional Modules

Module AA: Applying Classroom Subject Matter to the Game with Fact Coins (OPTIONAL)

OVERVIEW:

In this module, students will: Research a topic assigned by the teacher; find three facts about their topic; program facts to be displayed in the game.

INTRODUCTORY DISCUSSION:

Important Note: This unique module is intended for more educational settings, such as intra-curricular classes during the school day. It is not necessarily designed for camp or after-school settings. However, if you feel confident that this lesson could work in your classroom setting, please feel free to try it. Once again, this module is optional.

This module offers a unique approach in connecting the material students are learning in school to the games they are creating. Here is an explanation of how this works. First, the teacher assigns students to find three facts about a specified topic, preferably something they are currently learning about in school. Each student can be assigned to research a specific part of the topic. For example, if the topic is animals, each student would be assigned a different animal to research. Then, after their research, students will program each of those facts to be displayed in their games when the player collects a special item. We will call these special items "fact coins", and there will be three of them scattered across the levels of the game for the player to find.

Assign this research to students however you see fit, such as a homework assignment, or an in-class assignment. Do whatever you think is best for your classroom setting. Just be sure to only start this module after students have finished their research and have their three facts ready.

STEPS:

> **Note:** Before following these steps, students should have already researched their three facts as described above, and should have them ready, as they will be used shortly in the steps below.
>
> **Tell Students:** Now it's time to program all three facts from your research into the game. To do this, we will make special items called "fact coins" that will display a fact when the player touches it.
>
> **Ask Students:** What two things do we need to create our first fact coin? : A Sprite and an Object

1. Download or create an image to use as fact coin; create sprite; create object; name object "factcoin1."
2. In the factcoin1 Object Properties window, left-click the Add Event button (this will bring up the Event Selector window).

   Ask Students: When should the fact coin's message be displayed? answer: On collision between the player and fact coin.

3. In the Event Selector window, left-click Collision and choose player from the fly-out menu (since we are currently programming the factcoin1 object, the other

continued on following page

Module AA: *Continued from previous page*

object in the collision is the player, so we choose player from the Collision flyout menu).

Ask Students: Before we program the message to appear, let's think ahead. What else should happen to the fact coin after the player collects it? **answer**: The fact coin should disappear.

4. On the right side of the Object Properties window, in the Actions panels, left-click the main1 tab.

5. In the main1 actions, right-click the Destroy Instance button (this will bring up the Destroy Instance window).

   Remind students: We are programming on the factcoin1 object.

6. In the Destroy Instance window, leave "Applies to" as "Self" and click OK.

   Tell students: Now its time to program the fact to show up on the screen.

7. On the right side of the Object Properties window, in the Actions panels, left-click the main2 tab.

   Note: Have students look at the actions available, have them try to figure out which action should be used - answer: Display Message.

8. In the main2 actions, right-click the Display Message button (this will bring up the Display Message window).

   Note: Have students take out their facts from their research if they haven't done so already.

9. In the Display Message window, type in the first fact from your research, and click OK.

   Ask Students: Do you think we can test our game yet? answer: No, we need to add the fact coin to the room first.

10. In the Library, double-click (room) level1 (this will bring up the Room Properties window).

11. In the upper left of the Room Properties window, click the objects tab.

12. In the lower left of the Room Properties window, left-click the "famous add-thingy button" and select the factcoin1 object.

13. In the lower left of the Room Properties window, make sure Delete Underlying is unchecked, then left-click to add the factcoin1 object to the room.

    Remind Students: Place just 1 fact coin to the room, anywhere you'd like. Who decides what our game looks like? **answer:** We do, we are the designers.

14. **Test Game.** (Have students move the player to the fact coin to see the displayed message. To exit the message, you can click the OK button which appears under the text.)

    Important Note: If while testing the game, the message appears and continuously keeps reappearing after pressing OK, then there is an error in the programming that must be fixed. Although it appears impossible to exit back

continued on following page

Module AA: *Continued from previous page*

to game maker, holding down the escape (Esc) key on the keyboard for about 10 seconds should work. Once back in game maker, open the factcoin1 object and check to see if the Destroy Instance action is in the actions list. If not, follow steps 4-6 to fix this.

Note: In this next step, students will create the second fact coin by duplicating the first fact coin object, and making slight changes. Duplicating objects is very helpful when creating similar objects, as it saves a lot of time spent on programming.

15. In the Library, right-click the factcoin1 object and choose Duplicate.

    Tell Students: By duplicating factcoin1, this new object already contains all of the programming from the first fact coin. Now all we have to change is the name of the object and the message.

16. Name the new object "factcoin2"

17. In the Events list, left-click on the existing collision with player event.

    Tell Students: Instead of making new events and actions, we are just using and modifying the ones that already exist.

18. In the Actions list, double-click on the existing "Display a message" action. (this will bring up the Display Message window)

19. In the Display Message window, erase the current message, type in the second fact from your research, and click OK.

    Note: Although it is instructed to place all of the fact coins in the first room, students with multiple rooms can place them wherever they want.

20. In the Library, double-click (room) level1 (this will bring up the Room Properties window).

21. In the lower left of the Room Properties window, left-click the "famous add-thingy button" and select the factcoin2 object.

22. In the lower left of the Room Properties window, make sure Delete Underlying is unchecked, then left-click to add the factcoin2 object to the room.

23. Test Game.

    Tell Students: We will repeat this process one more time to create the third and final fact coin

24. In the Library, right-click the factcoin2 object and choose Duplicate.

25. Name the new object "factcoin3"

26. In the Events list, left-click on the existing collision with player event.

27. In the Actions list, double-click on the existing "Display a message" action. (this will bring up the Display Message window)

28. In the Display Message window, erase the current message, type in the third fact from your research, and click OK.

continued on following page

Module AA: *Continued from previous page*

29. In the Library, double-click (room) level1 (this will bring up the Room Properties window).
30. In the lower left of the Room Properties window, left-click the "famous add-thingy button" and select the factcoin3 object.
31. In the lower left of the Room Properties window, make sure Delete Underlying is unchecked, then left-click to add the factcoin3 object to the room.
32. Test Game. (All 3 facts should now be accessible to the person playing the game)

BONUS ACTIVITY: FACT WORKSHEET

This activity makes great use of the research and facts that students have programmed into their games, and in a very fun way. Using the facts provided by students, the teacher can create a worksheet with every student's facts in question form. (for example, "The giraffe is from Africa" would turn into "Where does the giraffe come from?"). In order to answer all of the questions on the worksheet, students must go on a "scavenger hunt" for the fact coins in their classmates games. While having fun going from computer to computer and playing all of the games, students are actually learning from their classmates. This is the culmination of all the research and programming students put into their games.

END 

Module BB: Using a Background for the gamestart, gamewin, or gameover rooms

OVERVIEW:

In this Module, students will: Draw a background and apply it to a room;

Introductory Discussion: Students have a choice of drawing or loading a background for any room. This is an alternative way to draw a message for the gamestart, gamewin, and gameover rooms. Rather than creating a sprite and an object, students create something called a background. While very similar to drawing sprites, backgrounds save a few steps and are often easier to work with for this purpose. Just as sprites are assigned to objects, backgrounds are assigned to rooms.

STEPS:

1. From the top menu choices, choose Resources>Create Background (this will bring up the Background Properties window).
2. In the Background Properties window, click the Edit Background button (this will bring up the Image Editor window).
3. In the Image Editor window, choose Transform>Stretch, (this will bring up the Stretch Image window).
4. In the Stretch Image window, uncheck the Keep Aspect Ratio button, then set the width to 640 pixels and the height to 480 pixels. (This will set the size of the background graphic to 640 x 480 pixels - the default size of a room). Click OK.

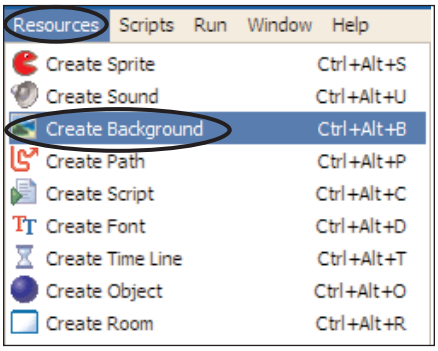
   Note: Make sure students are setting the pixels and not the percent. Students will often mistakenly change percent values instead of pixel values.

   Tell Students: Because we made this background bigger, we don't have to zoom in. So, make sure you are zoomed out all the way. First, let's fill the background in with a color.

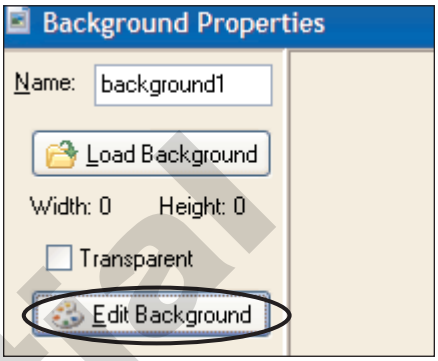
5. On the left side of the Image Editor window, in the Image Editor toolbox, left-click on the Paint bucket tool (Fill an area).
6. On the right side of the Image Editor window, in the Select Colors area, left-click to select a color for the background.
7. Position the cursor within the drawing area, then left-click to fill in the area with the selected background color. (For more information, refer to Appendix III: Art and Animation for Games)
8. On the left side of the Image Editor window, in the Image Editor toolbox, left-click on the Pencil tool, then left-click the bottom (thickest) line option.
9. Position the cursor within the room area, then press and hold the left-mouse button and drag the mouse to draw the appropriate message for the room (for details on how to use the Image Editor, refer to Appendix III: Art and Animation for Games).

   Note: At this point, students may draw any message they like depending on the room for which they are creating a background. (i.e. For the gamewin room, the student might write, "You Win")

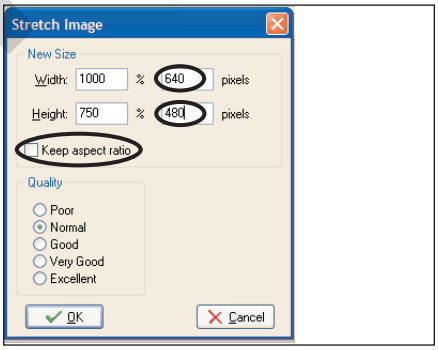
continued on following page



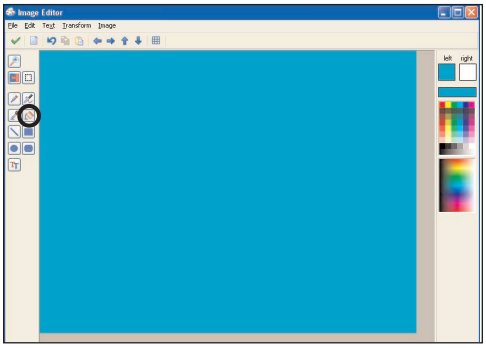
1. Choose Resources>Create Background



2. In the Background Properties window, click the Edit Background button



4. In the Stretch Image window, uncheck the Keep Aspect Ratio button, then set the width to 640 pixels and the height to 480 pixels



5, 6, 7. Use the Paint Bucket tool to select a color for your background

Module BB: Continued from previous page

10. In the upper left corner of the Image Editor window, click the green check mark (this will close the Image Editor window and return you to the Background Properties window).

    Tell Students: Now that we have the background done, let's attach it to the room.

11. In the Library, in the Rooms folder, double-click whichever room you want to add the background to in order to bring up the Room Properties window for that room.

12. In the upper left of the Room Properties window, click the "backgrounds" tab.

13. In the backgrounds tab of Room Properties window, next to the field labeled "<no background>," left-click the "famous add-thingy button." This will bring up a flyout menu of all backgrounds currently available. Select the appropriate background from the flyout menu.

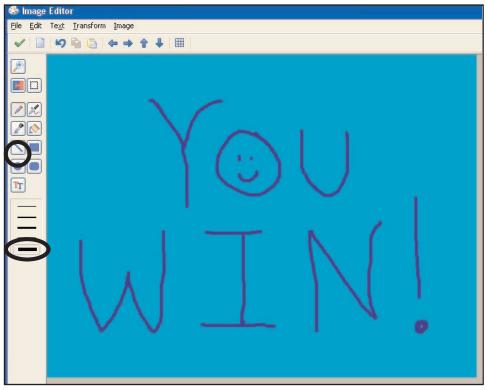
14. In the upper left corner of the Room Properties window, click the green check mark.

    Note: When students run their games, the newly assigned background should appear in the appropriate room.

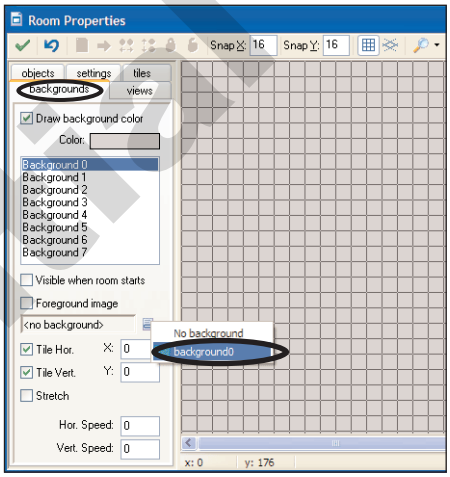
END

VOCABULARY:

Background: A non-interactive image usually set behind the objects of a room; applied directly to a room (not to an object).



8, 9. Left-click on the pencil took, then left-click the bottom (thickest) line option. Draw your message in any color that is different from the background color



12, 13. In the Rooms folder, double-click whichever room you want to add the background to in order to bring up the Room Properties window for that room. Left-click the backgrounds tab and then select the appropriate background from the flyout menu

LESSON 1 ASSESSMENT TOOL (QUIZ)

Math Section

1. Add the following numbers: 5 + 5 + 5 + 5 = _____

2. Rewrite the above addition problem as a multiplication problem: ( ) x ( ) = _____

3. Fill in the correct symbol below. (>, <, =):

a. 100 [ ] 200

b. 0 [ ] -1

c. 47 [ ] -47

d. 157 [ ] 157

4. Answer below with either "horizontal" or "vertical":

a. On a graph, the x-axis runs in a _____ direction.

b. On a graph, the y-axis runs in a _____ direction.

5. What is another way to describe "horizontal" using words? _____

6. What is another way to describe "vertical" using words? _____

7. On a graph, which of the values below are to the right of x = 40? (Circle your answers)

x = 50 x = -40 x = 34 x = 44 x = 0 x = -60

8. What is meant by the term "two-dimensional" (2D)? _____

_____

9. What is one way to describe an object's position on a two-dimensional graph? _____

_____

Quiz is continued on following page ⟹

LESSON 1 ASSESSMENT TOOL (QUIZ)

Vocabulary Section

Define the following terms (use the back of this page if you need more room):

Program

Programmer

Graphic

Download

Server

Local Hard Drive

Sprite

Avatar

Object

Object-Oriented Programming

Event-driven programming

Cartesian Coordinate (Graphing) System

x-axis, y-axis

Adversary

Artificial Intelligence (AI)

x and y coordinates

Goal

Obstacle

Quiz is continued on following page ⟹

LESSON 1 ASSESSMENT TOOL (QUIZ)

Technical Application Section *(Hands-on Computer Programming Test)*

Note to teachers: Students should be given approximately fifteen minutes to accomplish the technical application section (Steps below).

Finished instructor file can be found on the Instructor Files disk.

Instructions:
Working in Game Maker, try to accomplish the Steps listed below, and save the file (be sure to follow instructions in Step 10 for saving the file). Don't worry if you can't do all the Steps, just try your best.

1. Create a new file

2. Import art to use as avatar

3. Create sprite and object for avatar

4. Put avatar object in room

5. Program the game so that the player can move their avatar by pressing the arrow keys on the keyboard

6. Import art to use as adversary

7. Create sprite and object for adversary

8. Add adversary objects to room

9. Test your game and debug if necessary

10. Save the file to the desktop, name the file using your first name and the word "test1"
For example: A student named Phil would name his file: philtest1

Bonus:
Program Artificial Intelligence for adversaries to chase the player