Using Game Maker 8

Mike Bailey

mjb@cs.oregonstate.edu

http://cs.oregonstate.edu/~mjb/gamemaker

Oregon State University

See also:

http://cs.oregonstate.edu/~mjb/sketchup
http://cs.oregonstate.edu/~mjb/chromadepth
http://cs.oregonstate.edu/~mjb/shapes

What is Game Maker?

• **YoYo Games** produced *Game Maker* so that many people could experience the thrill of making a computer do what you ask it to do, under the guise of producing a game.

• Game Maker creates an event-driven, object-oriented simulation with a visual drag-and-drop interface.

• Game Maker program executables can be run standalone or can be run from within a web page (after loading a plug-in)

• The “Lite” Edition can be downloaded for free! There is also a “Pro Edition” that costs money. ($20)

Student Learning Objectives

1. Learn the basics of simulation software
2. Learn the step-by-step thinking that characterizes writing computer programs
3. Learn the ideas behind incremental program enhancement
4. Learn the ideas behind event-based computer programming
5. Learn the ideas behind object-oriented programming
6. If you want a head start on learning Java or C++, you can learn to use the Game Maker scripting language

Getting Game Maker for Free

Go to:

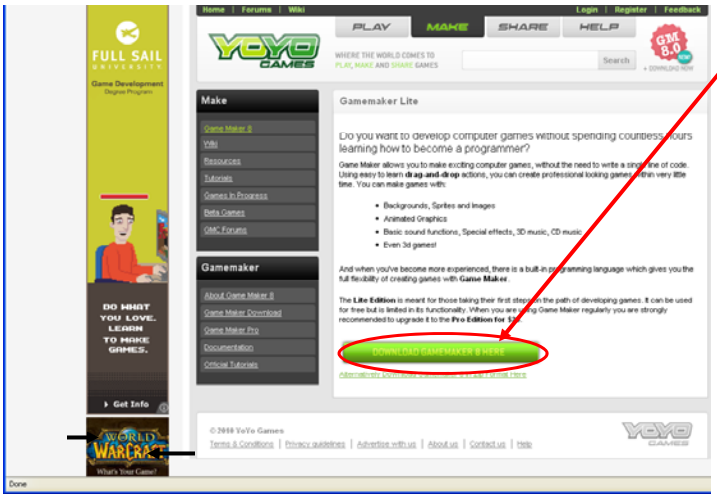
http://www.yoyogames.com/gamemaker

Follow the links to the free download (see the next page).

GameMaker comes in Windows 2000/XP/Vista/7 versions.

Getting Game Maker for Free

Getting Game Maker for Free



Good Game Maker Web Links

Main Game Maker Site:
`http://www.yoyogames.com`

These (and other) notes:
`http://cs.oregonstate.edu/~mjb/gamemaker`

Alphabetized list of Actions and what tab to find them under
`http://cs.oregonstate.edu/~mjb/gamemaker/actions.pdf`

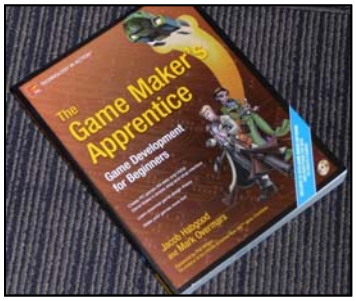
Using Game Maker for a Simple Ecological Simulation:
`http://cs.oregonstate.edu/~mjb/gamemaker/ecosim.pdf`
`http://cs.oregonstate.edu/~mjb/gamemaker/ecosim.gmk`

276-page PDF Game Maker 7 documentation:
`http://cs.oregonstate.edu/~mjb/gamemaker/gmaker.pdf`

Good Reference Books

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice*, Apress, 2006.

($27 on Amazon)

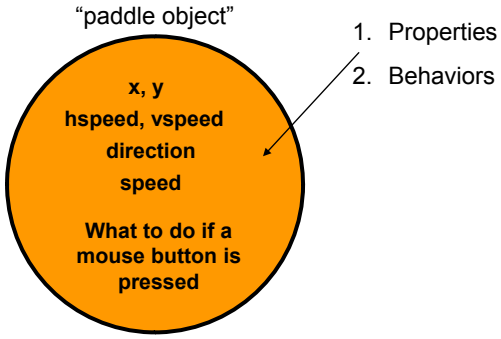


($23 on Amazon)

Jerry Lee Ford, *Getting Started with Game Maker*, Course Technology, 2010.

Game Maker Introduces Object-oriented Programming

Each object has properties and behaviors encapsulated inside of it. This entire collection can be referenced by just the object name ("Paddle") or by one property ("Paddle.hspeed") or behavior ("Paddle's Left Mouse Button Event") at a time



Game Maker Teaches Event-based Programming

Wait for some specific Event to happen

Perform some Action(s) in response to it

Some examples:

User presses a key on the Keyboard	Restart the current Room
User holds down a button on the Mouse	Move Object A to wherever the Mouse is
A new Object B is Created	Get it positioned and moving
Object C collides with Object D	Bounce Object C and play a sound
Object E collides with Object F	Destroy this instance of Object F

"Events" "Actions"

A Demonstration of Events: A Chase Simulation

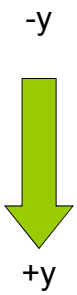
Two Objects: the Chaser and the Chasee:

1. Upon Creation, the Chasee starts at a random x and y location and heads in a random direction from 0° to 360° with a speed of 8
2. Upon Creation, the Chaser starts at a random x and y location
3. At each step, the Chasee changes its direction to a random direction from 0° to 360°
4. At each step, the Chaser takes a step towards the Chasee with a speed of 2
5. If the Chaser collides with the Chasee, a sound is played, the Chasee is obliterated, and the simulation restarts
6. If the Chaser goes outside the room, it plays a sound and bounces
7. If the Chasee goes outside the room, it wraps around to the other side of the room
8. If the 'R' key is hit on the keyboard, restart the simulation

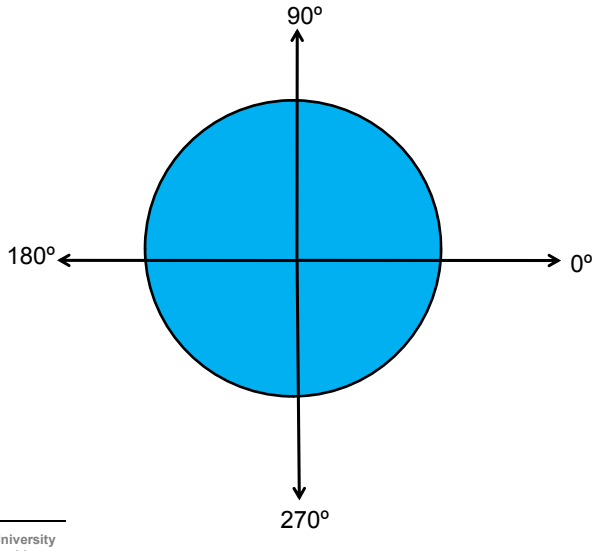
What Game Maker Means by the Y-axis

Warning: Game Maker defines **+Y** as ***down*** !

"Paddle.y - 50" is ***above*** the paddle.



What Game Maker Means by Angle Direction



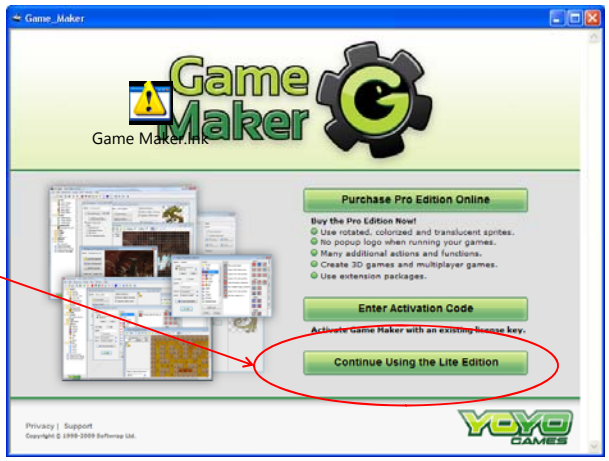
Getting Started

Double-click on the GameMaker icon

Or click on **Start** → **All Programs** → **Game Maker 8** → **Game Maker**

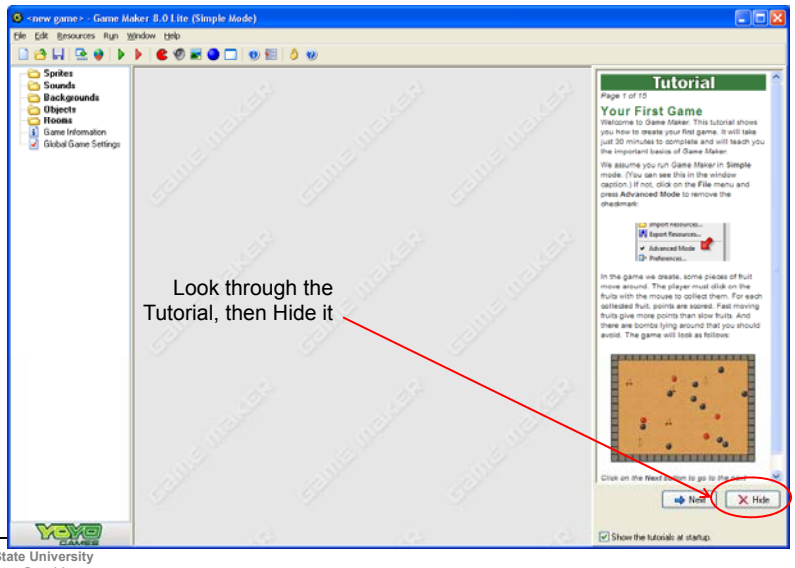
You will get a screen that looks like this:

Click here !



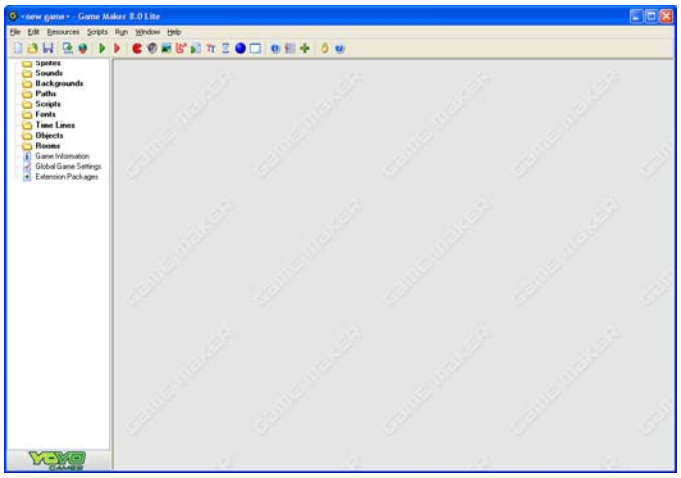
Getting Started

You will then get a start screen that looks something like this:



Getting Started

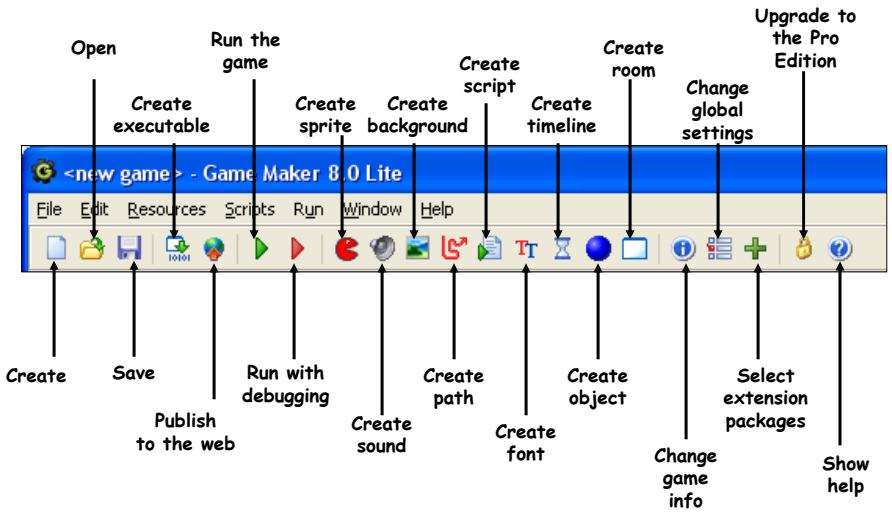
Now, click on **File→Advanced Mode**
This isn't really an advanced mode – it just brings up a few more icons, like this:



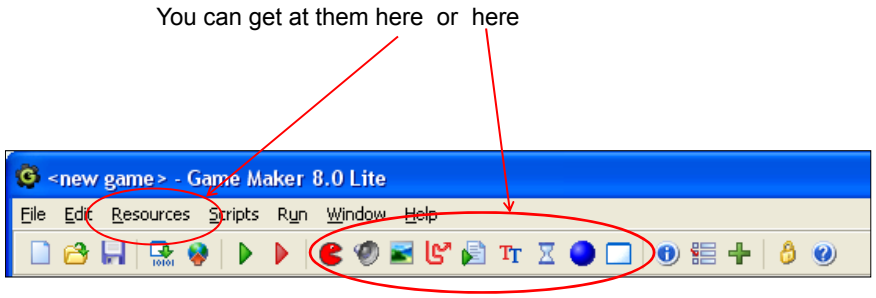
Right now, click **File→Save As** – and hit **Save** often while you are editing

Getting Started

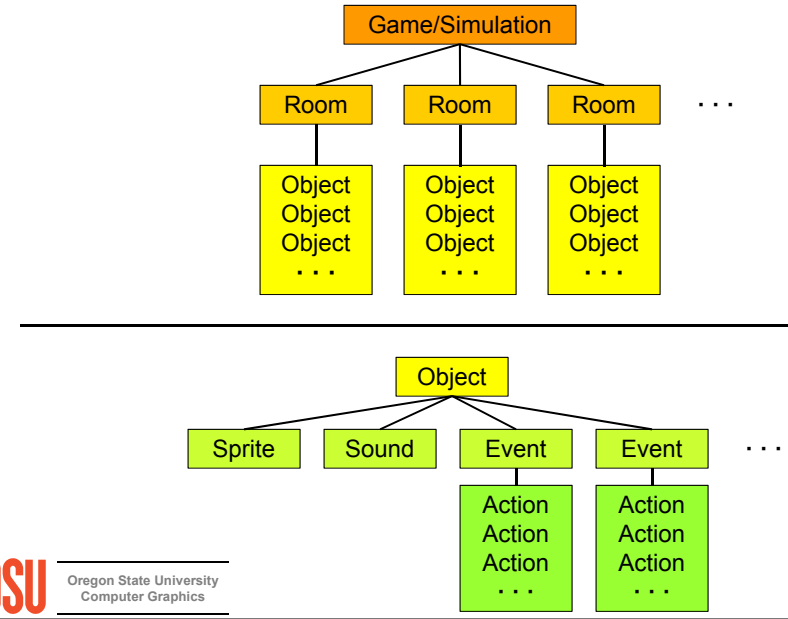
The icons across the top are *really* important:



All the things you can add to the game are called "Resources"



The Structure of a Game/Simulation



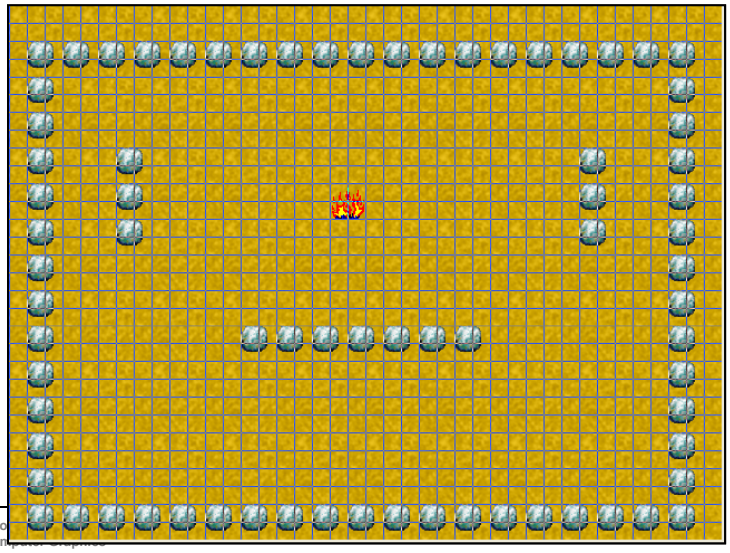
Game Maker Steps

1. Describe the game you are trying to create What is it supposed to do? What is it supposed to look like?
2. Define the sprites
3. Define the sounds
4. Define the objects themselves, **but not (yet) their events and actions**
5. Go back and define each object's events and actions
6. Define the room
7. Put the object instances in the room

It is best to define the objects first and their events and actions later because some of those actions will need to be asked for in terms of objects (that might not have been created yet)

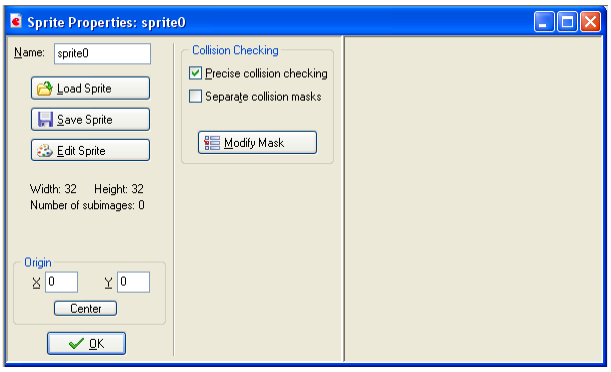
Let's Start with Just a Simulation

1. A fire bounces around off walls, forever and ever



Creating a Sprite

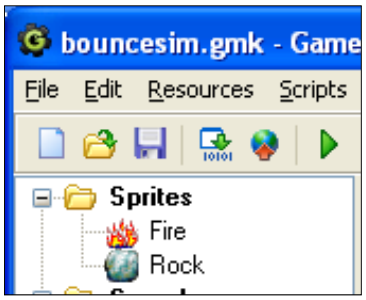
1. Select **Resources→Create Sprite**
2. Type in a name for this sprite
3. Click **Load Sprite**
4. Navigate to where your Sprite folder is (depends where you installed Game Maker)
5. Pick one
6. Click **OK**



The sprites are just images – you can create your own. (Use the .gif or .ico format.)

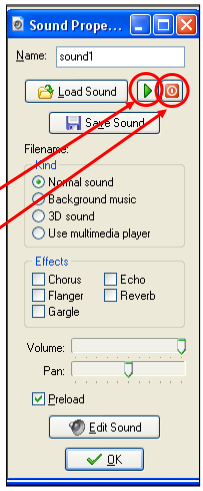
Define Two Sprites: Resources→Create Sprite

1. Fire = **Sprites → various → Fire.ico**
2. Rock = **Sprites → maze → rock.gif**



Creating a Sound

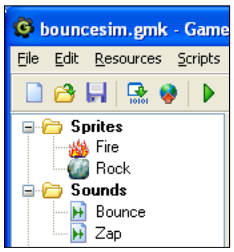
1. Select **Resources→Create Sound**
2. Type in a name for this sprite
3. Click **Load Sound**
4. Navigate to where your Sound folder is (depends where you installed Game Maker)
5. Pick one
6. If you want to check what it sounds like, click the green arrow
7. If you click the green arrow, the sound will start playing over and over (yuch). Click the red thing to turn it off.
8. Click **OK**



You can create your own sounds. Use the .wav format.

Define a Bouncing Sound : Resources→Create Sound

Bounce = **Sounds** → **boink2.wav**



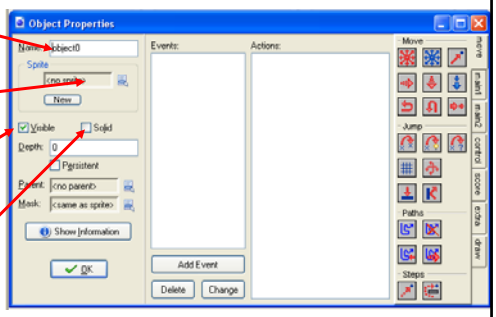
Also, while we're at it:

Define the Background: Resources→Create Background

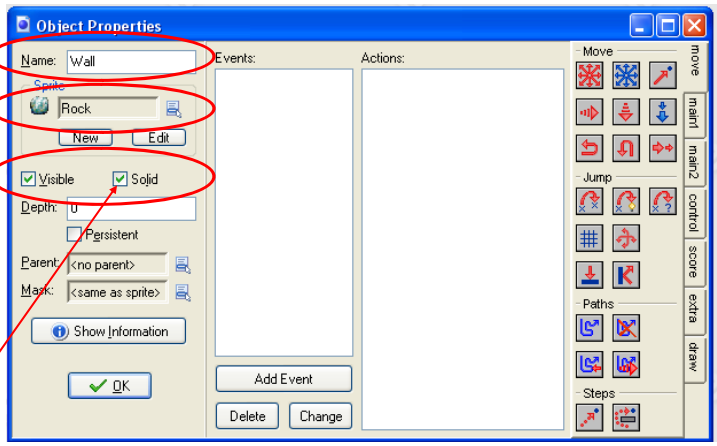
Background = **Backgrounds** → **sand1.gif**

Creating an Object

1. Select **Resources→Create Object**
2. Type in a name for this object
3. Select a sprite to represent this object from the **Sprite** pull-down menu
4. Click **Visible** if you want this object to be seen during the game
5. Click **Solid** if you want the object to be a solid that something can bounce off of, like a wall

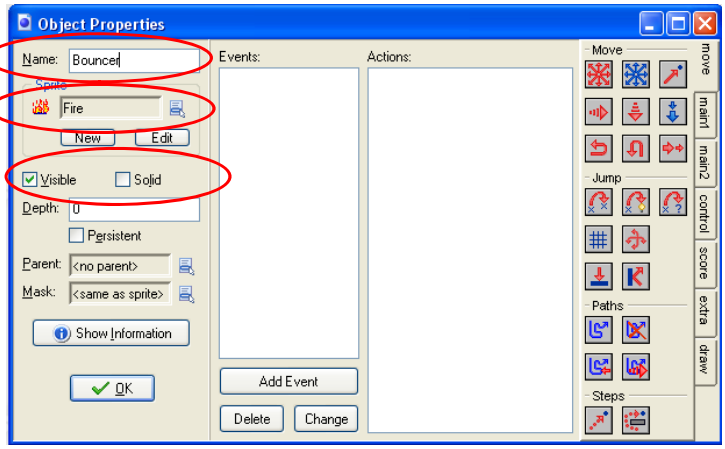


Define the Wall Object: Resources→Create Object



The wall is "Solid" because something (the fire) will need to bounce off of it

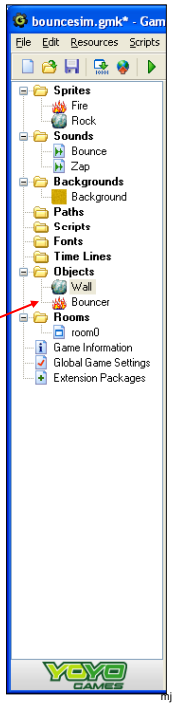
Define the Bouncer Object: Resources→Create Object



Editing something that you've created

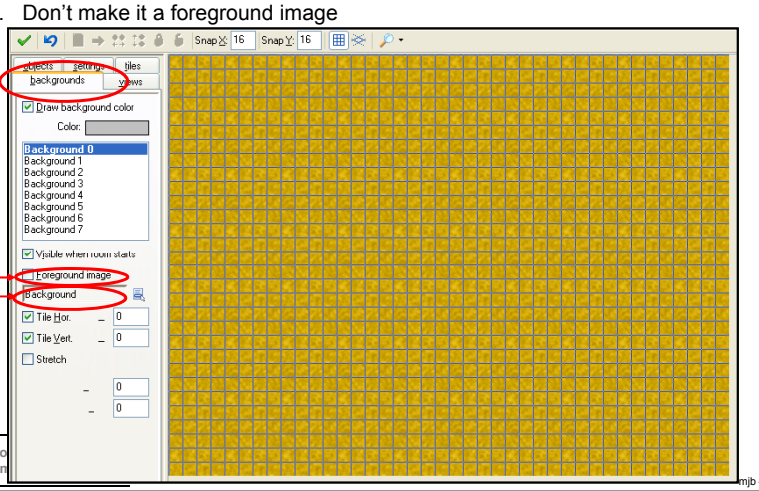
To go back and edit something that you've previously created, double-click on it in this menu area

For example, to go back and add events and actions, double-click on one of the objects



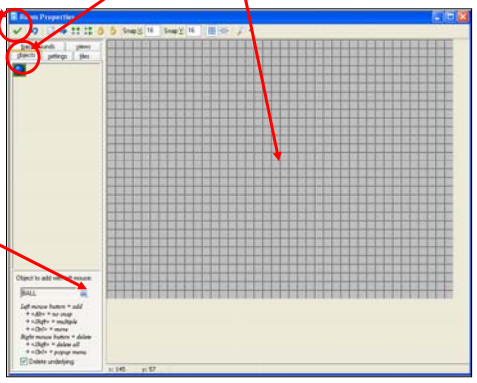
Define the Room: Resources→Create Room

1. Select **Resources→Create Room**
2. Set the background by clicking the **background** tab
3. Set the background to **Background** from the menu of backgrounds you've defined before
4. Don't make it a foreground image



Adding Objects to the Room

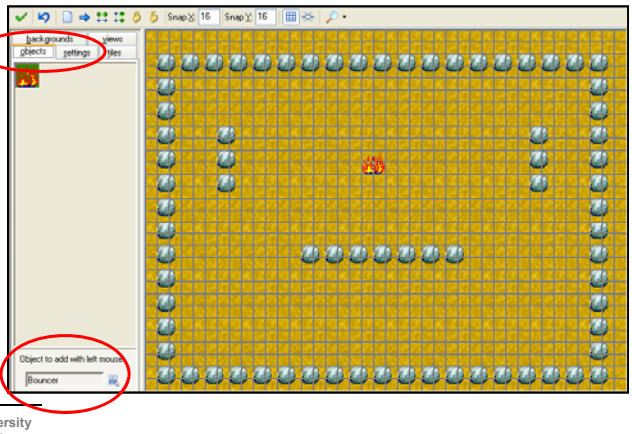
1. Click on the **Objects** tab
2. Select an object with the pull-down menu
3. Click in the room to place as many instances of them as you want
4. Click the green checkmark when you are done



Game Maker refers to each of these objects in the room as an ***Instance***.

Adding Objects to the Room

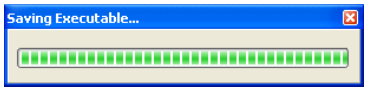
1. Position the objects by clicking the **objects** tab
2. Select an object from the pop-up menu
3. Left-click as many of them into position as you need
4. Right-click an object to delete it



Run Your Simulation!

Click on the Green Arrow in the main toolbar

Game Maker will save your executable, which looks like this:



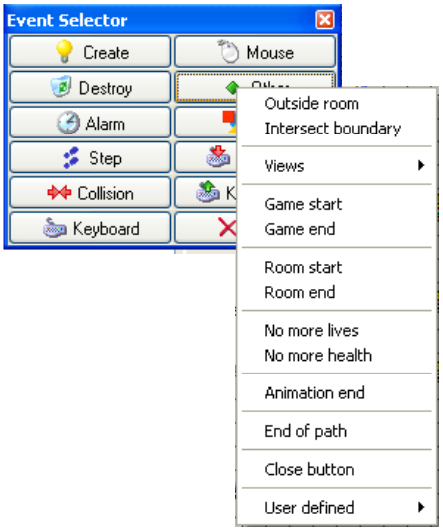
And then load it, which looks like this:

And will then execute it in a new window. Hit the keyboard **Escape key** to stop your program and return to the Game Maker main window.

Adding an Event to an Object with the Event Selector

- This menu allows you to select what will trigger this event
- Some of these events will bring up other dialog boxes to let you be more specific. For example, the **Mouse** event button will bring up another dialog box to let you specify what the mouse has to do (buttons, press/release, moving, etc.) to trigger this event.
- You then drag and drop into the **Actions area** as many actions as this Event will cause to happen

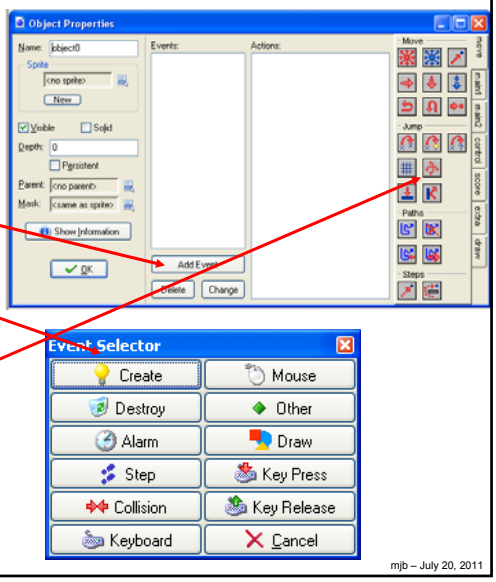
The "Other" Event List



I'm not sure why these are in a separate list instead if the main Event Selector

Creating an Object's Events and Actions

1. If you want events associated with this object click **Add Event**
2. Select what will trigger the event from the **Event Selector**
3. Drag and drop what Action(s) this Event will cause from the action icons into the **Action area**.



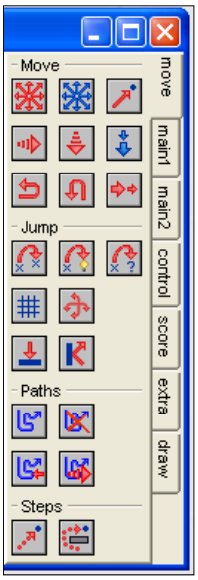
Move Actions

Move Fixed
Move Free
Move Towards
Speed Horizontal
Speed Vertical
Set Gravity
Reverse Horizontal
Reverse Vertical
Set Friction

Jump to Position
Jump to Start
Jump Random
Align to Grid
Wrap Screen
Move to Contact
Bounce

Set Path
End Path
Path Position
Path Speed

Step Toward
Step Avoiding



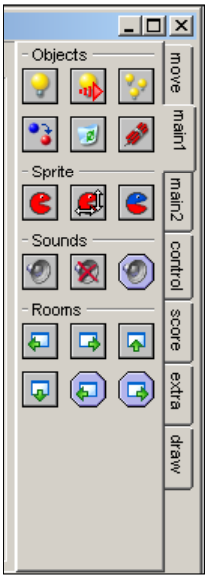
Main1 Actions

Create Instance
Create Moving
Create Random
Change Instance
Destroy Instance
Destroy at Position

Change Sprite
Transform Sprite (Pro Edition only)
Color Sprite (Pro Edition only)

Play Sound
End Sound
Check Sound

Previous Room
Next Room
Restart Room
Different Room
Check Previous
Check Next



Main2 Actions

Set Alarm
Sleep
Set Time Line
Time Line Position
Time Line Speed
Start Time Line
Pause Time Line
Stop Time Line

Display Message
Show Info
Splash Text (Pro Edition only)
Splash Image (Pro Edition only)
Splash Webpage (Pro Edition only)
Splash Video (Pro Edition only)
Splash Settings (Pro Edition only)

Restart Game
End Game
Save Game
Load Game

Replace Sprite (Pro Edition only)
Replace Sound (Pro Edition only)
Replace background (Pro Edition only)

Control Actions

Check Empty
Check Collision
Check Object
Test Instance Count
Test Chance
Test Question
Test Expression
Check Mouse
Check Grid

Start Block
Else
Exit Event
End Block
Repeat
Call Parent Event

Execute Code
Execute Script
Comment

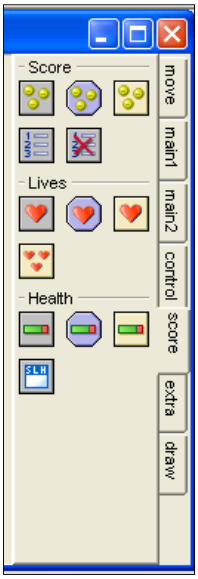
Set Variable
Test Variable
Draw Variable

Score Actions

Set Score
Test Score
Draw Score
Show Highscore
Clear Highscore

Set Lives
Test Lives
Draw Lives
Draw Life Images

Set Health
Test Health
Draw Health
Score Caption

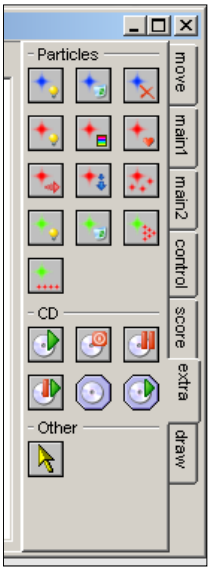


Extra Actions

Create Particle System (Pro Edition only)
Destroy Particle System (Pro Edition only)
Clear Particle System (Pro Edition only)
Create particle (Pro Edition only)
Particle Color (Pro Edition only)
Particle Life (Pro Edition only)
Particle Speed (Pro Edition only)
Particle Gravity (Pro Edition only)
Particle Secondary (Pro Edition only)
Create Emitter (Pro Edition only)
Destroy Emitter (Pro Edition only)
Burst From Emitter (Pro Edition only)
Stream from Emitter (Pro Edition only)

Play CD (Pro Edition only)
Stop CD (Pro Edition only)
Pause CD (Pro Edition only)
Resume CD (Pro Edition only)
Check CD (Pro Edition only)
Check CD Playing (Pro Edition only)

Set Cursor (Pro Edition only))

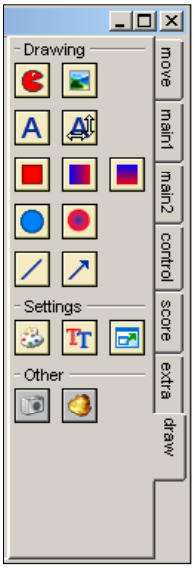


Draw Actions

Draw Sprite
Draw Background
Draw Text
Draw Scaled Text (Pro Edition only)
Draw Rectangle
Horizontal Gradient (Pro Edition only)
Vertical Gradient (Pro Edition only)
Draw Ellipse
Gradient Ellipse (Pro Edition only)
Draw Line
Draw Arrow

Set Color
Set Font
Set Full Screen

Take Snapshot (Pro Edition only)
Create Effect (Pro Edition only)



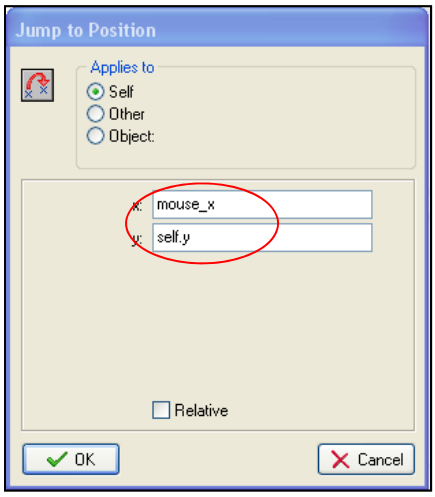
A Handy List of Game Maker Actions and What Tab to Find them Under

Get this sheet at:

http://cs.oregonstate.edu/~mjb/gamemaker/actions.pdf

Action Parameters

Most actions ask you to type in parameters. These parameters can be numbers, or they can be mathematical expressions using symbolic parameters



Object Properties

x	**Instance's current x coordinate**
y	**Instance's current y coordinate**
xstart	**where this instance started**
ystart	**Where this instance started**
xprevious	**Previous position**
yprevious	**Previous position**
hspeed	**X speed in pixels/step**
vspeed	**Y speed in pixels/step**
direction	**Current direction in degrees (0-360)**
speed	**Current speed in pixels/step**

Some of the parameters are properties of an object. When you type them in, you will ask for them by typing the object name, a period, and then the property name.
For example:

Paddle.x

Fire.y

There are some special names for objects. One of the most common is "self", designating the object that triggered this event. You can find out where it is, for example, by typing **self.x** and **self.y**

Global Names

score	**Current score**
lives	**Current number of lives**
health	**Current health of the player (0-100)**
mouse_x	**X position of the mouse**
mouse_y	**Y position of the mouse**

Some of the parameters are global names, that is, they belong to the game as a whole, not to a single object. When you type them in, you will ask for them by typing just the property name. Three of the most common are:

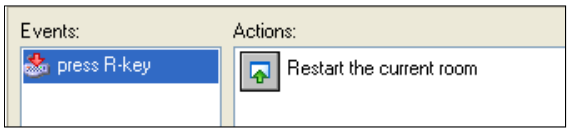
score

mouse_x

mouse_y

Note that these are spelled with an underscore not a period. These are names, not objects with properties.

Define the Wall Object Events



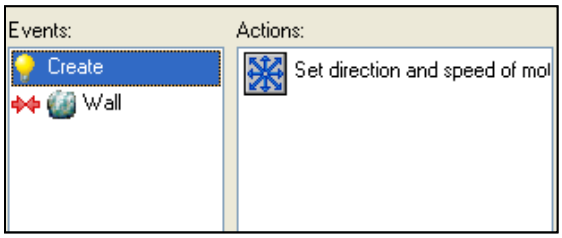
1. **main1→Restart Room** (the transition you choose is up to you)

This is the tab

This is the event

These are the parameters to select or type in

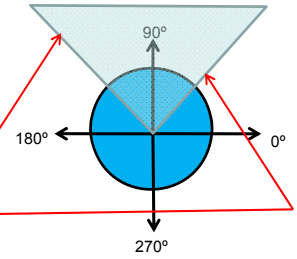
Define the Bouncer Object's Events



1. **move→Move Free:** 45+random(90), 8

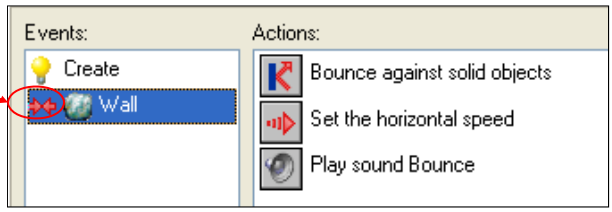
The *random(N)* function returns a random number between 0. and N.

So, the phrase "45+random(90)" will give a random number between 45°and 135°



Define the Bouncer Object's Events

The red double-arrows designate a Collision event. The picture to the right of the red arrows shows what you are checking for a collision with.



1. **move→Bounce:** Self, not precisely, solid objects
2. **move→Speed Horizontal:** Self, -2+random(4), Relative

, "-2+random(4)" gives you a random number between -2 and +2.

3. **main1 →Play sound: Bounce**

Why is this game randomly altering the fire's speed after a bounce? If you don't, there will likely be times when the fire will end up in a state where it is bouncing back and forth over the exact same path forever and ever. This action alters the fire's speed just enough to prevent that. The trick is to make it big enough to work, but small enough to be unobtrusive.

Let's Have the Fire Obliterate Something

Define Another Sprite: Resources→Create Sprite

burger = **Sprites → various → Burger.ico**

Define Another Sound : Resources→Create Sound

Zap = **Sounds → zap.wav**

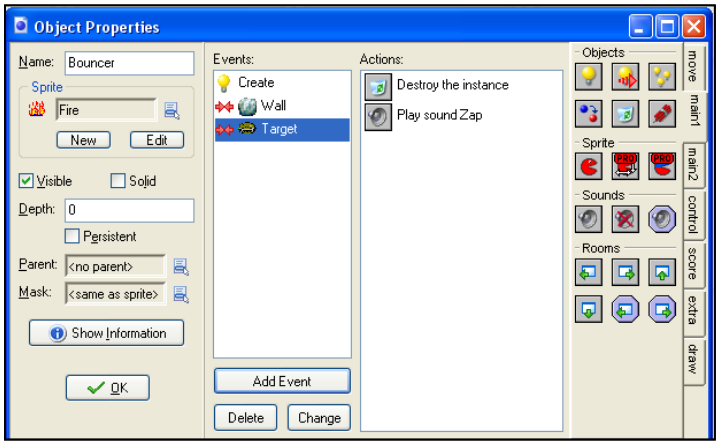
Define an Object called "Target": Resources→Create Object

Use the burger sprite

Doesn't need to be solid

The Target object doesn't need any events, we'll let the fire do the obliterating

Add Another Event to the Bouncer Object



1. **main1→Destroy Instance:** Other
2. **main1→Play Sound:** Zap, false

"Other" is one of those special names. It means the object involved in the collision that is not "Self".

Add some Targets (the burgers) to the room, then …

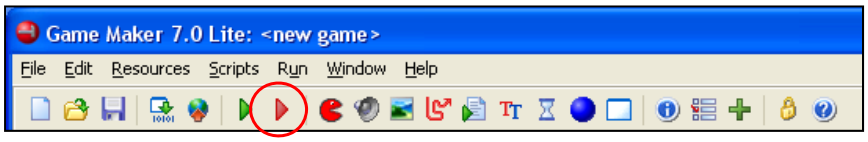
Running Your Simulation!

Try arranging the rocks from the wall differently.
Try setting different values for the starting speed and direction.
Try using *random( )* in the speed setting.

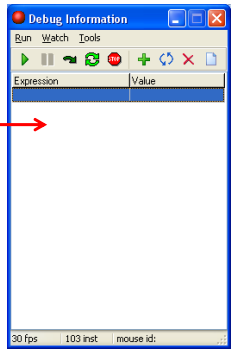
How does this affect your simulation?

Running the Simulation in Debug Mode

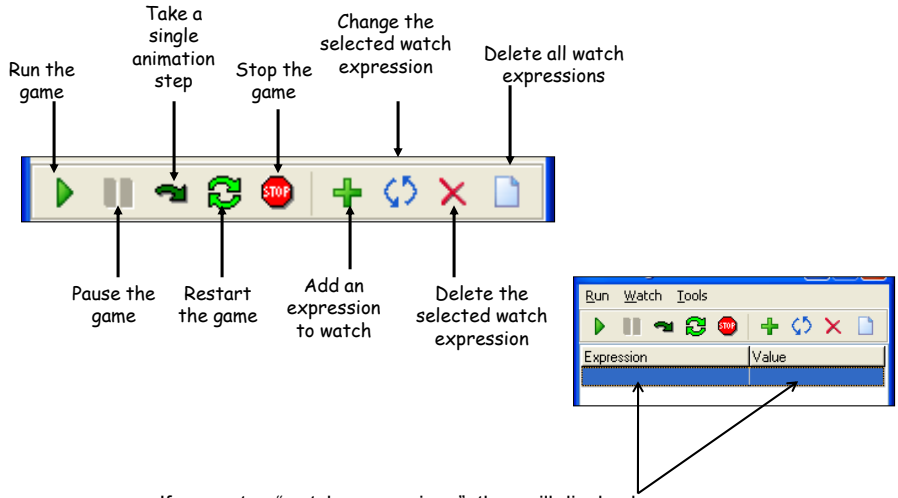
Click the **red triangle arrow** in the titlebar



This brings up a new information window



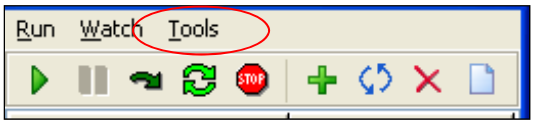
Running the Simulation in Debug Mode



If you setup "watch expressions", they will display here

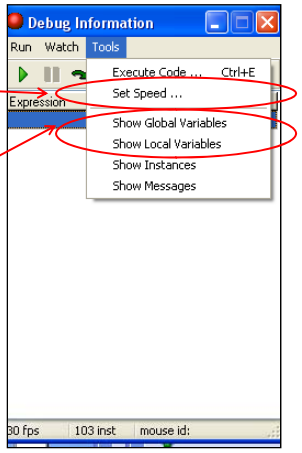
(If you want to experiment, try a watch express of: **mouse_x**)

Running the Simulation in Debug Mode



Normally, Game Maker tries to run your game at a refresh rate of 30 frames per second ("fps"). You can change that here to slow down the game play. This is useful for debugging, so that you can get a better idea what is going on.

These are useful for debugging, especially if you are using scripts



Turning the Simulation into a Game

1. Much of the time, simulations are passive, that is, we watch the world progress. If the goal is a game, we need to add some user interaction, including a way for the user to score points.
2. Let's get rid of a wall so the Bouncer can leave the room.
3. Let's also add a paddle for the for the user to try to keep the Bouncer in play.

One More Sprite: Resources→Create Sprite

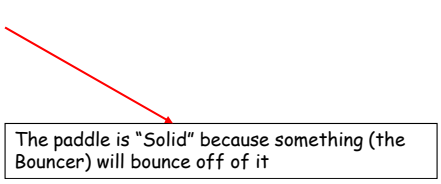
paddle = **Sprites** → **breakout** → **bat1.gif**

One More Sound: Resources→Create Sound

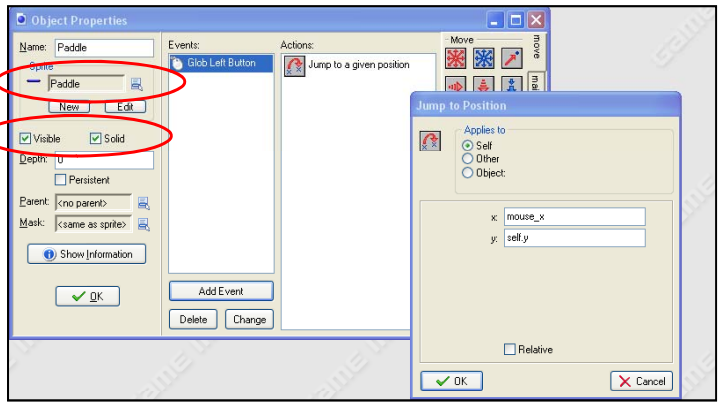
LeftRoom = **Sounds** → **beep7.wav**

Another Object: Resources→Create Object

Paddle = **Paddle sprite, Solid**



The Paddle Object Needs to Follow the Mouse X Coordinate



move→Jump to Position: Self, mouse_x, self.y

Note that the mouse x location is "**mouse_x**", *not* "**mouse.x**" ! This is because the mouse is not an object. It is just an input device with some values that your game has access to.

The Bouncer Object Needs to Bounce off the Paddle Object

1. **move→Bounce: Self, not precisely, solid objects**
2. **main1→Play Sound: Bounce**

Something Needs to Happen if the Bouncer Leaves the Room

1. **main1→Play Sound: LeftRoom**
2. **move→Jump to Position: Self, Paddle.x, Paddle.y - 50**
3. **move→Move Free: Self, 45+random(90), 8**

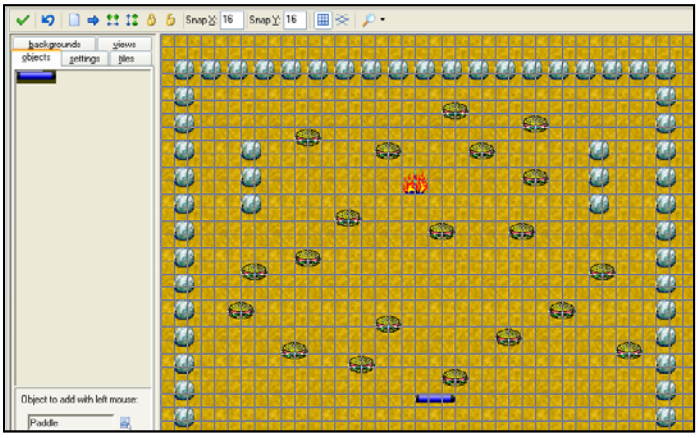
Warning: Game Maker defines **+y** as ***down*** ! "Paddle.y-50" is ***above*** the paddle.

-y

+y

Adjust the Room

1. **Add the Paddle**
2. **Remove a wall of rocks**



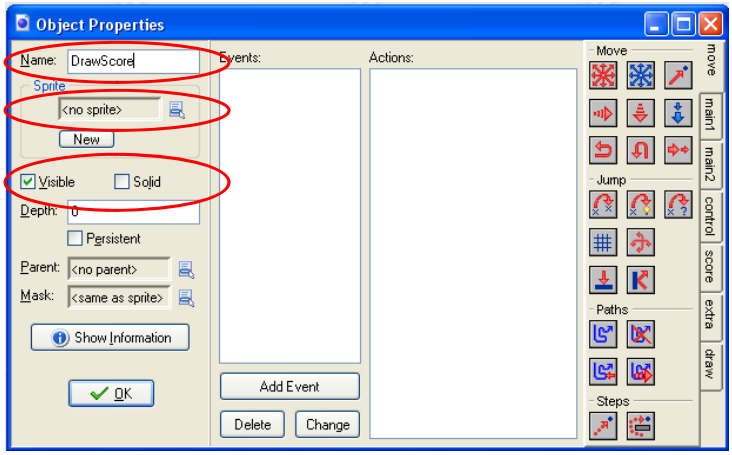
Score a Point for Every Burger Obliterated

Add to the Bouncer-Target Collision Event:

score→Set Score: 1, Relative

This is like saying "Score = Score + 1"

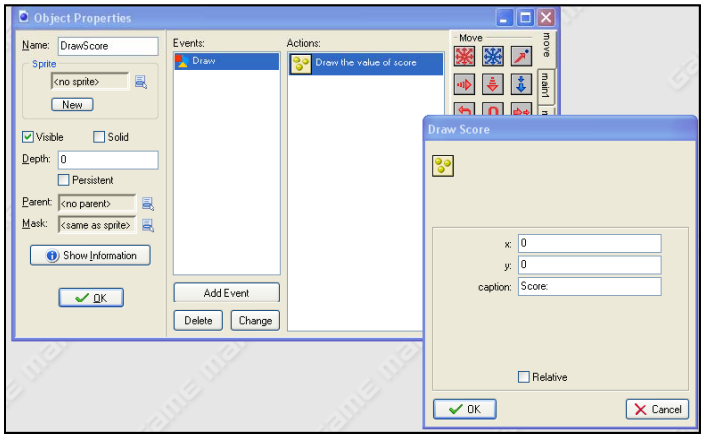
Define the DrawScore Object: Resources→Create Object

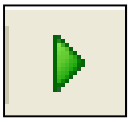


Score a Point for Every Burger Obliterated

Add a Draw Event to the DrawScore Object:

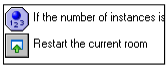
score→Draw Score, 0, 0, Score

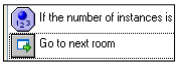




What Else Could You Add?

1. Check the Instance Count of the Targets, and Restart the Room if it goes to 0
2. Check the Instance Count of Targets and go to the next (more difficult) room if it goes to 0
3. If the Paddle misses the Bouncer, subtract something from the Score

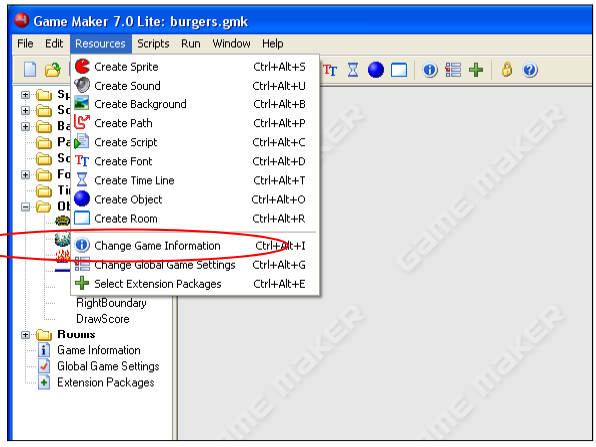




4. Move the Paddle with the Arrow Keys instead of (or in addition to) the mouse. (Hint: Use Keyboard→<Left> and Keyboard→<Right> as the Events. Use Jump to Position with relative movement for the Action.)

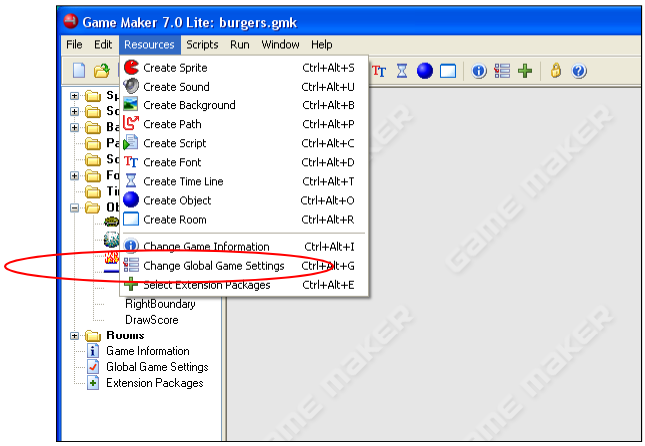
5. If the Paddle hits the Bouncer, increase the Bouncer's speed a little to make it harder to hit with the Paddle
6. What else?

Setting your Game Information



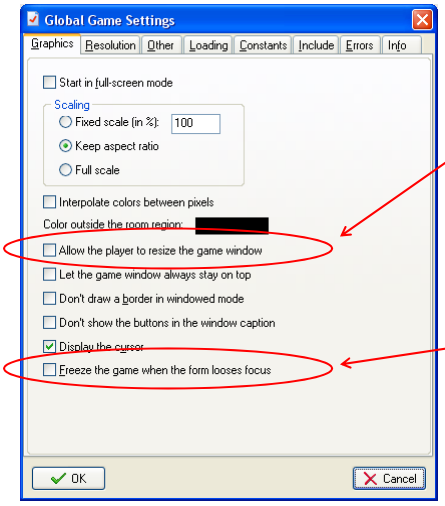
Double-click on this and enter some information. This will be shown if the player hits the <F1> key while playing your game.

Setting Global Information about Your Game



Double-click on this and a tabbed dialog box will pop up. See the next few slides to see what you can do with this.

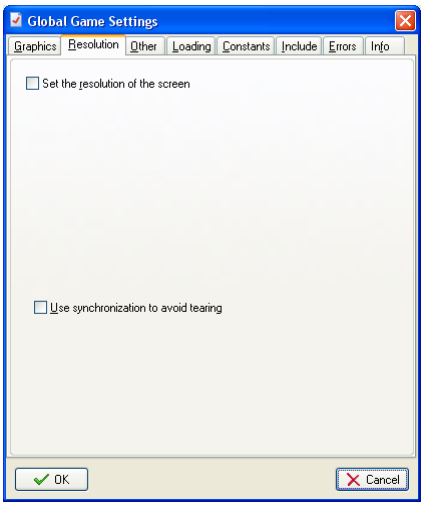
Setting Global Information about Your Game



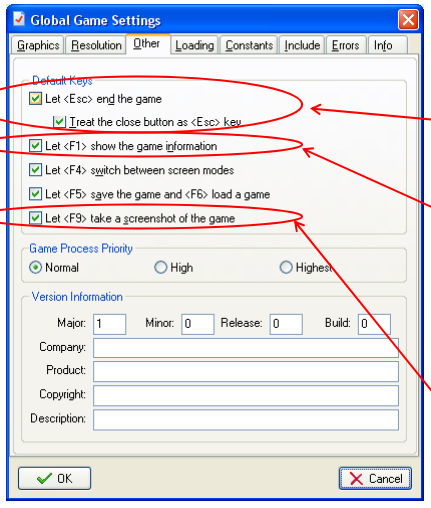
Do you want the player to be able to resize the graphics window?

Do you want the game to keep playing or freeze if you click in another window?

Setting Global Information about Your Game



Setting Global Information about Your Game

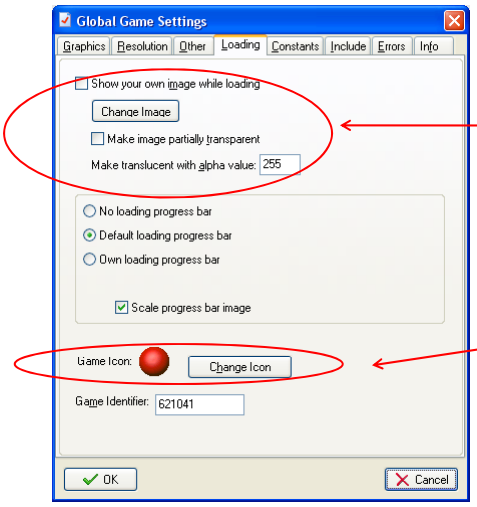


Normally the Escape key terminates the game. However, you can disable this if you want. At times this is useful if you want to force the player to save the game before exiting.

Yes, of course you want <F1> to show your game information.

This feature is *really* handy, so of course you want it enabled! Hitting the <F9> key while playing the game will put an image of the current state of the game in a file called screenshotXXX.bmp, where XXX is 100, 101, 102, etc. The files live in the same folder where your game .gmk file lives.

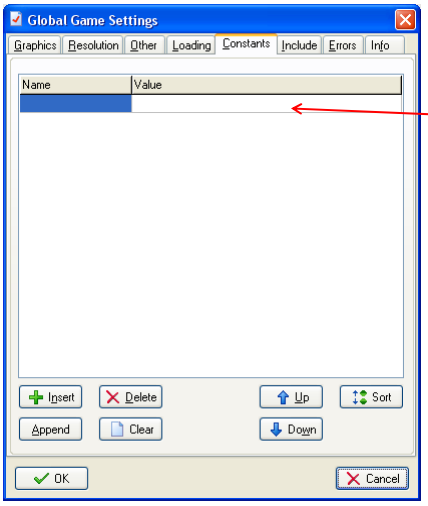
Setting Global Information about Your Game



By default, the YoYo Games logo displays during loading. You can change this to something of your own. This image can be one of around 30 different image file types.

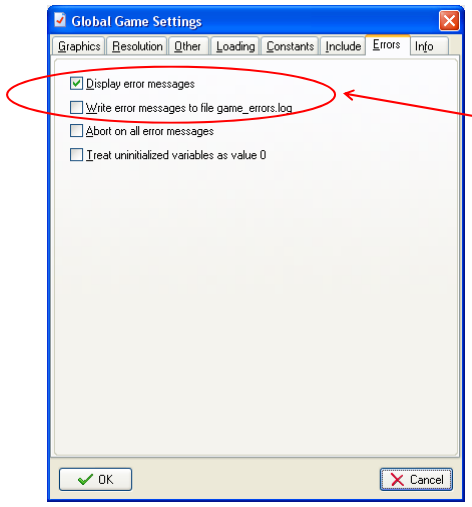
You can assign your own game program icon! This image must be in .ico format however. Many image manipulation programs are capable of producing this. Sadly, Photoshop doesn't appear to be one of them.

Setting Global Information about Your Game



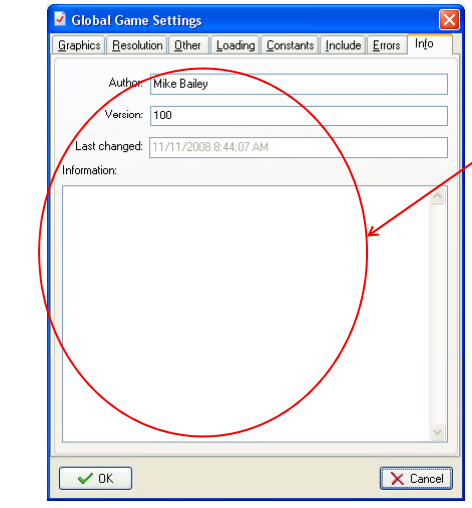
You can pre-define some constants, such as how many of something will be in your game, etc. This is probably more useful when you are writing scripts.

Setting Global Information about Your Game



Of course you want to see error messages! If you really care, you can also record them to a file for further examination or printing.

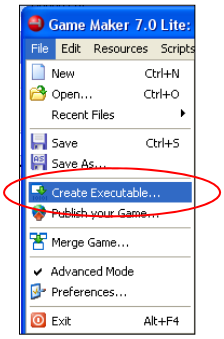
Setting Global Information about Your Game



This is more program info. This is not the same as the information that will come up when a player hits the <F1> key.

Sharing Your Game with Others

Click **File→Create Executable**



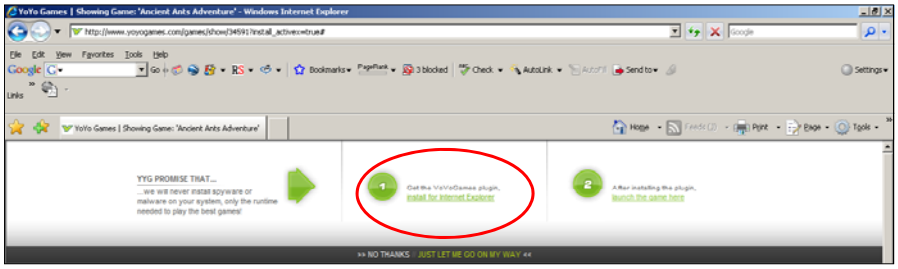
This creates a file with a **.exe** extension. This can be given (email, web page posting, memory stick, etc.) to others.

However, there is the usual warning about running a .exe file sent to you from an untrusted source!!

It's safer to send around .gmk files and read them into Game Maker !

You can also embed your game in a Web Page

You need to load a YoYo Games Internet Explorer plug-in to make this work



Game Creation Basics

From The Game Maker's Apprentice:

- Provide clear, achievable goals
- Give feedback on the player's progress
- Include both short-term and long-term goals
- Add difficulty levels and optional sub-goals for players of different abilities
- Reward the player for achieving goals and sub-goals
- Reward the player randomly
- Give the player choices that make a real difference in the game
- Don't confuse the player with too many controls
- Don't punish the player for things outside of their control
- Avoid unfair setbacks
- Give the player audio feedback about their interactions with the game

And, then one that I've always heard:

- Make the game easy to learn, but hard to master.

A Comes-At-You Game Example

"Burn-Me" – the Game

The idea is that fires come at the burger. He needs to either hit them or avoid them, depending on how you want the game play to work.

Define the Sprites: Resources→Create Sprite

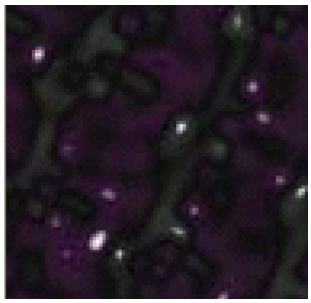
1. burger = **Sprites → various → Burger.ico**
2. fire = **Sprites → various → Fire.ico**

Define the Sounds : Resources→Create Sound

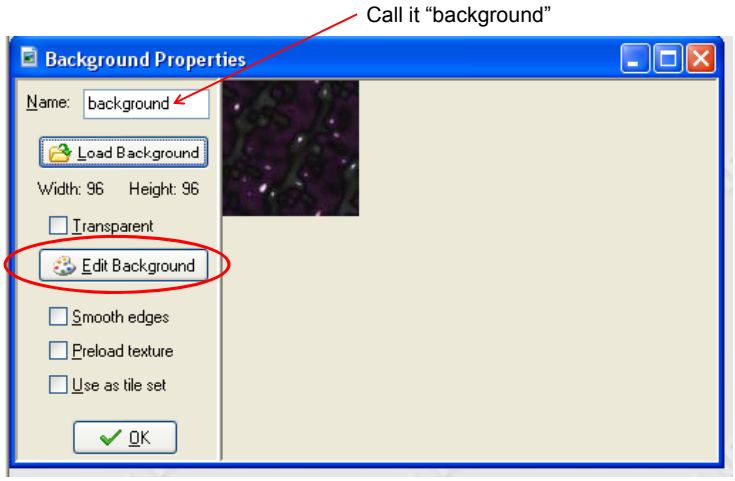
1. zap = **Sounds → zap.wav**

Define the Background: Resources→Create Background

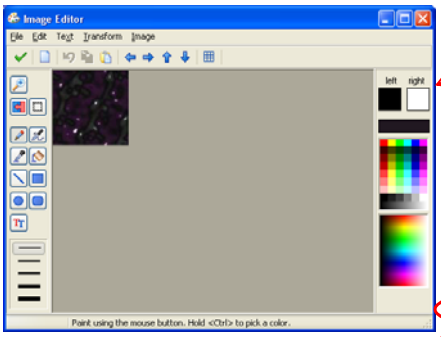
1. Background = **Backgrounds → stars.gif**



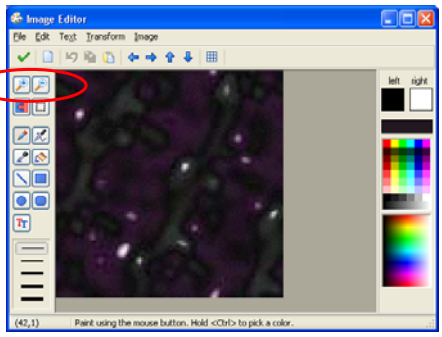
Try Editing the Background Image



Try Editing the Background Image

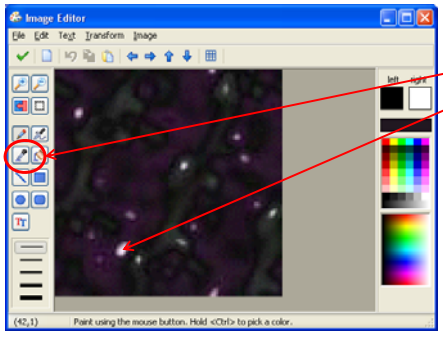


This is the background-editing window



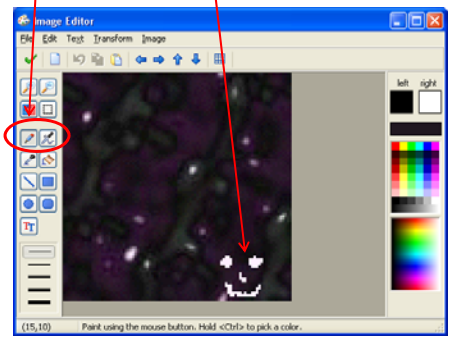
Click on the magnifying glasses to zoom in or out

Try Editing the Background Image



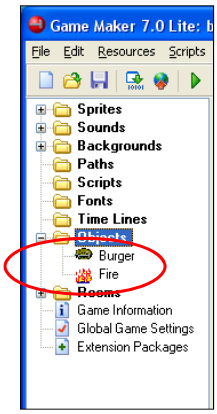
Use the eyedropper tool to select a color in the image

Use the drawing or spray painting tool to draw in the image

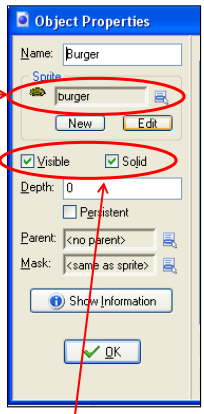


The Have-A-Nice-Day Galaxy? ☺

Setup the Burger and Fire Objects
(Leave the Events and Actions for Later)

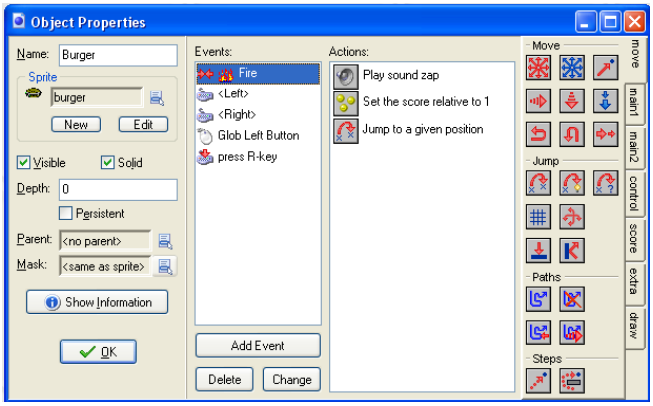


Set the appropriate sprite



Both need to be *Visible* and *Solid*

Setup the Burger Object's Events and Actions



Setup the Burger Object's Events and Actions

Collision with Fire

1. **main1→Play Sound:** zap, no looping
2. **score→Set Score:** 1, relative
3. **move→Jump to Position:** Other, random(room_width), -5

Global Left Button

1. **move →Jump to Position:** Self, mouse_x, self.y

Keyboard <Left>

1. **move →Jump to Position:** Self, -10, 0, Relative

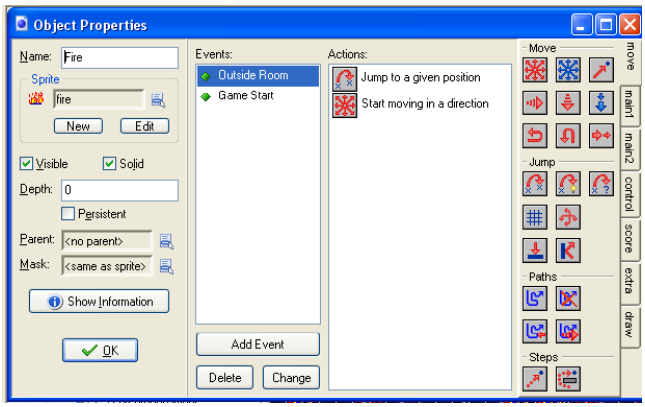
Keyboard <Right>

1. **move→Jump to Position:** Self, 10, 0, Relative

Press R key

1. **main1→Restart Room:** Fade out and in

Setup the Fire Object's Events and Actions



Setup the Fire Object's Events and Actions

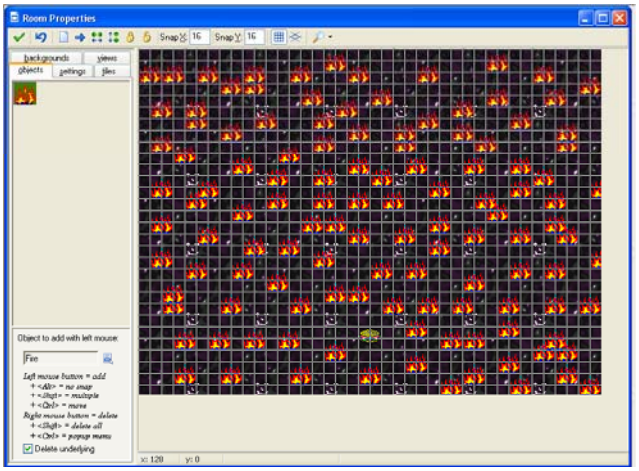
Outside Room

1. **move→Jump to Position:** Self, random(room_width), -5
2. **move→Move Fixed:** Self, Down arrow, 2

Game Start

1. **score→Set Score:** 0
2. **move→Move Fixed:** Self, Down arrow, 2

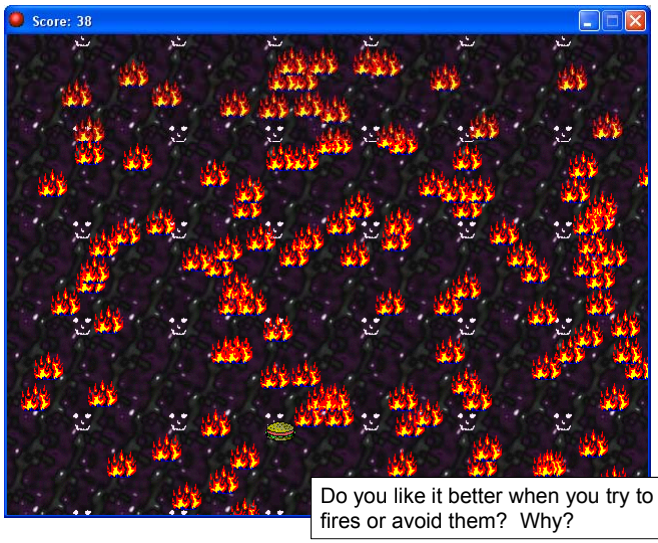
Setup the Room



Be sure to make the window big enough to see the entire room.

Put in one burger and lots of fires!

Run the Game



Do you like it better when you try to hit the fires or avoid them? Why?

An Interesting Variation

In the Fire Events, Change:

Outside Room

move→Move Fixed: Self, Down arrow, 2

To:

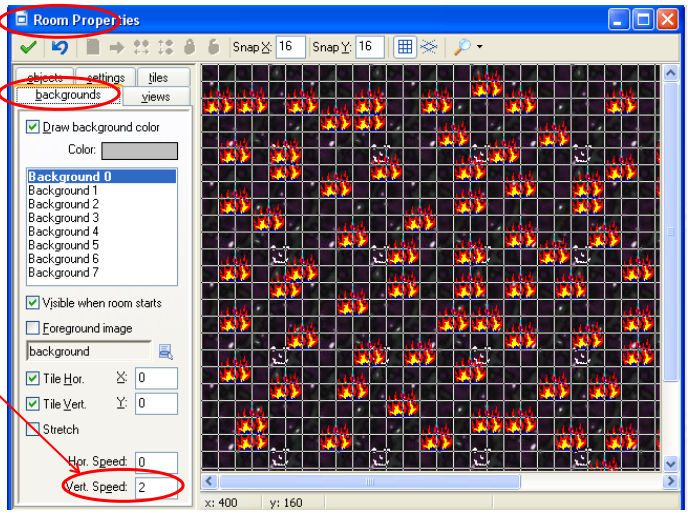
Outside Room

move→Move Free: Self, 225+random(90), 2

What will this do???

Try Making the Background Move with the Fires

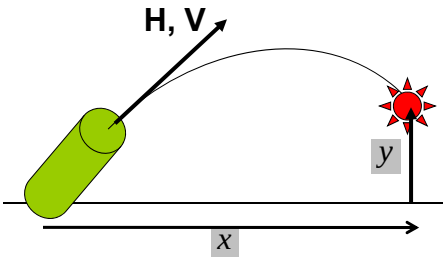
Set the background's Vertical Speed to 2 (down), same as the fires'



This now makes it look like the burger is flying through space, instead of the burger being stationary and having fires attack it.

A Physical Simulation Example

Projectile Motion Simulation



H and V are the object's Horizontal and Vertical speed

Quantities during Flight:

$$x = H \times t$$

$$y = V \times t - \frac{1}{2} \times g \times t^2$$

x,y = distances in feet
H,V = horizontal and vertical initial velocities in feet / sec
T = time in seconds
g = gravitational acceleration = 32.2 feet / sec^2

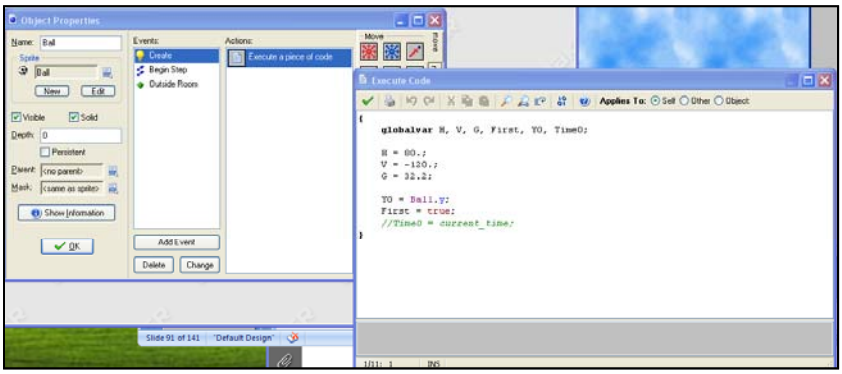
Projectile Motion Simulation Setup

1. **Ball: various→Ball1.ico**
2. **Goal: maze→Finish.gif**

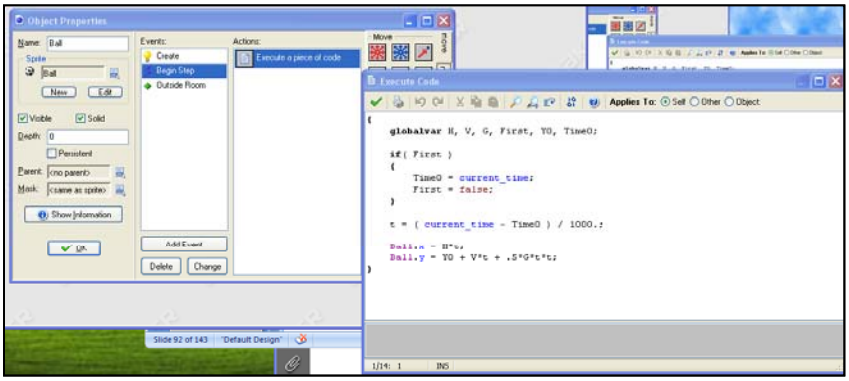
Scored: Sounds→applause.wav

Field: Backgrounds→sky.gif

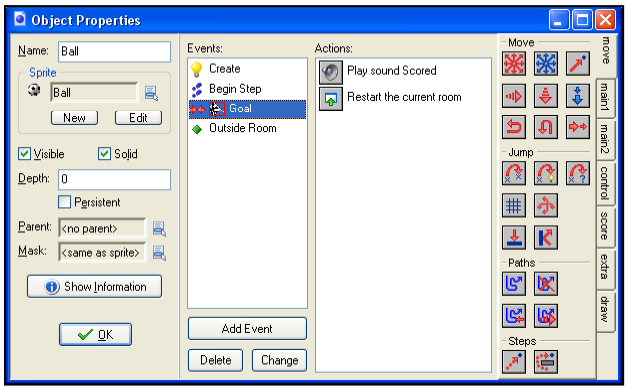
A Script to Setup Everything



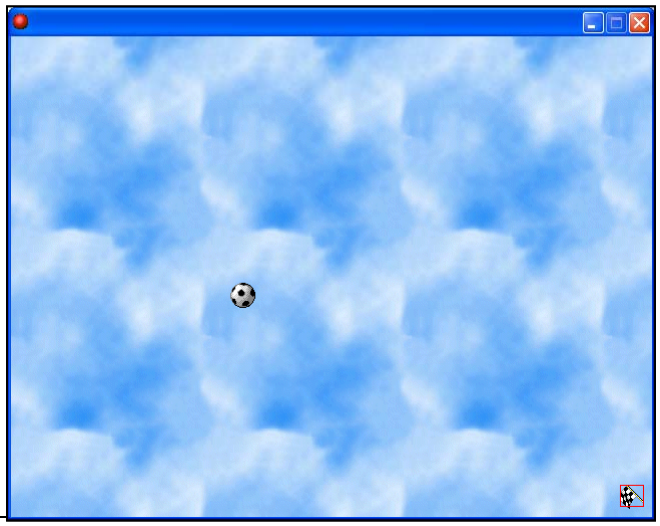
A Script to Compute X and Y



What to do if Actually Hit the Goal

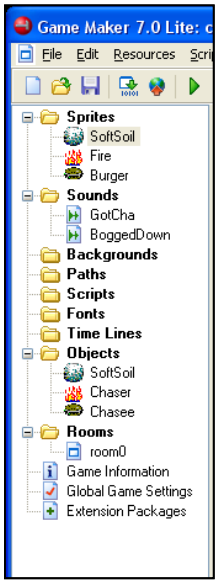


projectile.gmk



A Chases-You Game Example

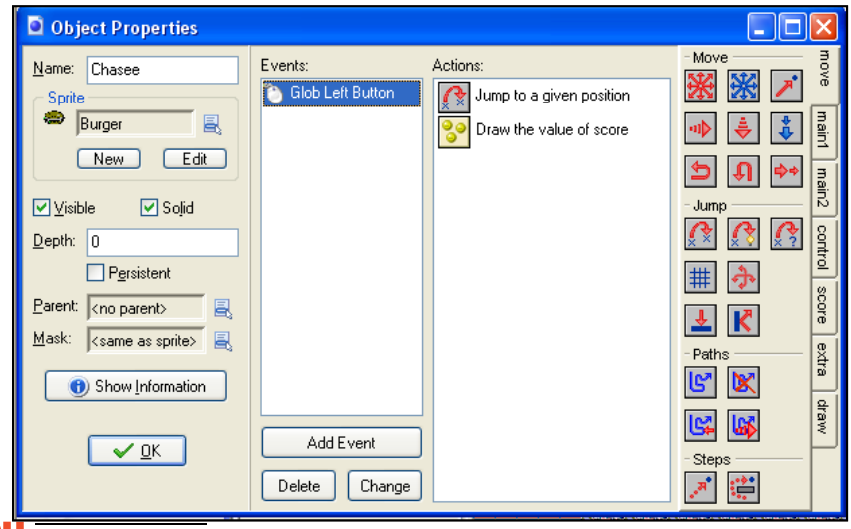
Create Sprites, Sounds, and Objects



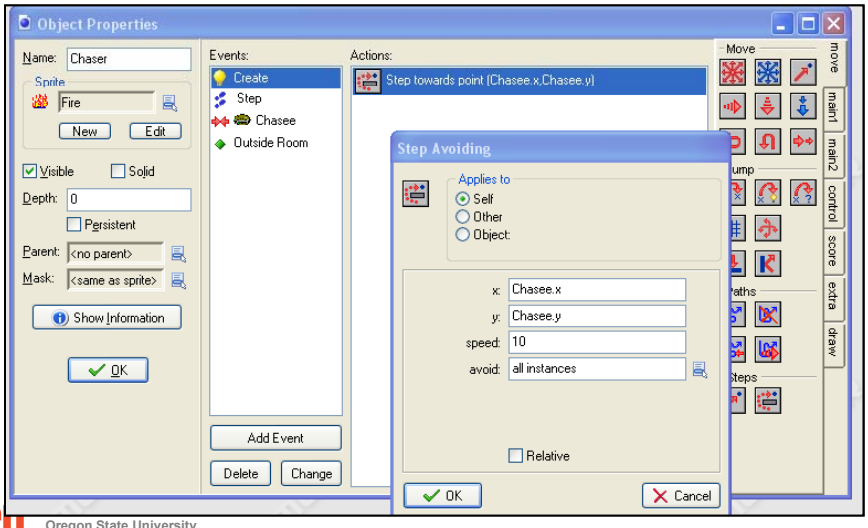
Oregon State University
Computer Graphics

mjb – July 20, 2011

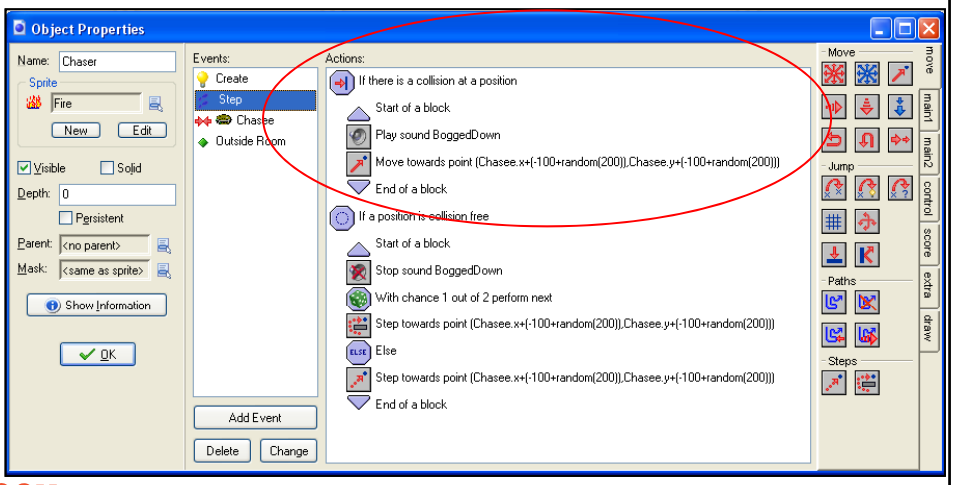
The Chasee Object should Follow the Mouse



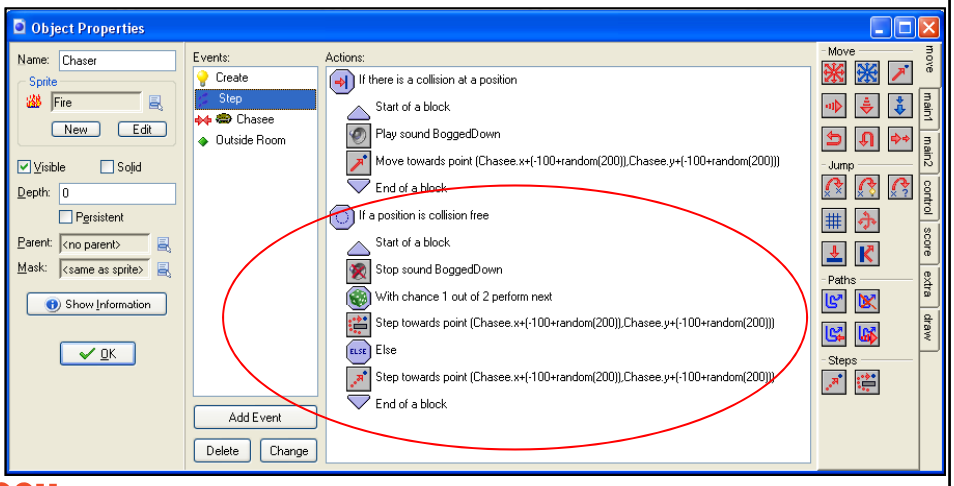
The Chaser Object Should Try to Get to the Chasee



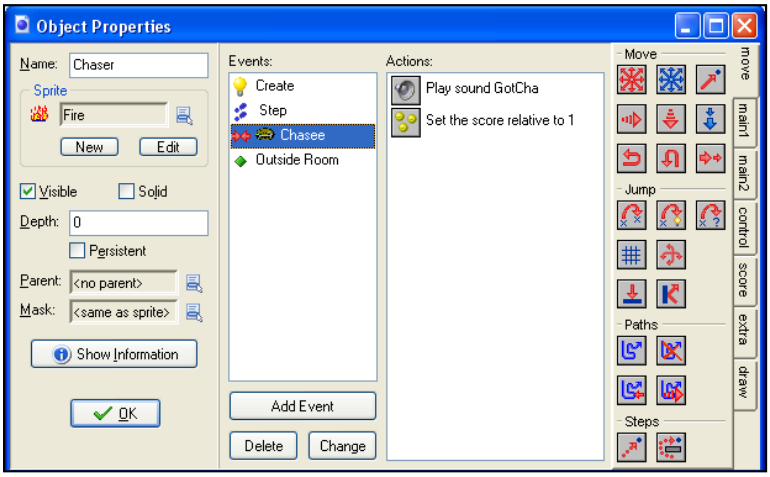
If the Chaser Object Overlaps with the Soft Soil, it Should Slog Through it Towards the Burger



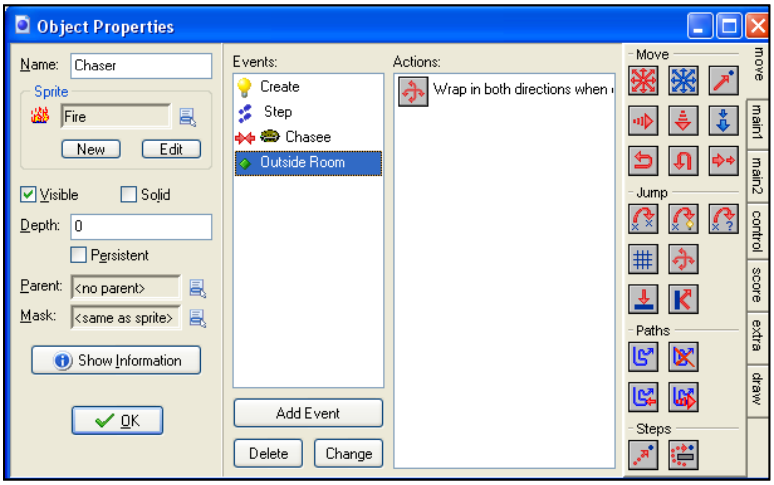
If the Chaser Object Doesn't Overlap with the Soft Soil, it Should 50% of the Time Go Around the Soft Soil and 50% of the Time Go Through it

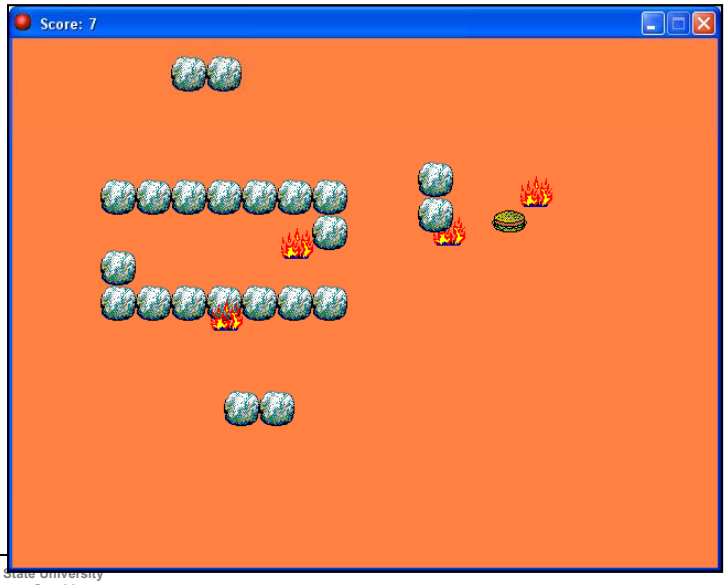


If the Chaser Catches the Chasee, Play a Sound and Add One to the Score



If the Chaser Gets Outside the Room, Wrap it Around to the Other Side (You Could Also Bounce It)





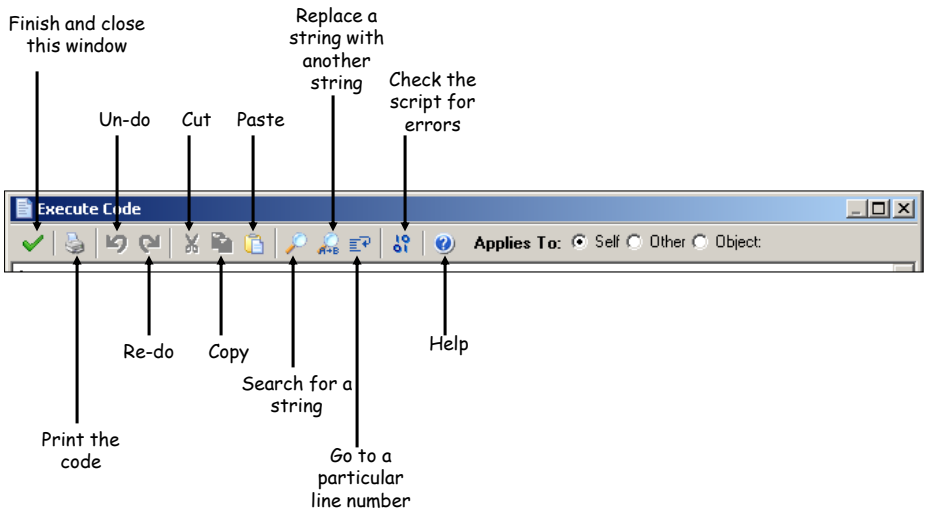
Game Maker Scripting

Scripting using the Game Maker Language (GML)

There are two neat things about using GML:

1. It allows your game to do things that the drag-and-drop features can't do by themselves
2. It looks very much like C++ and Java programming !

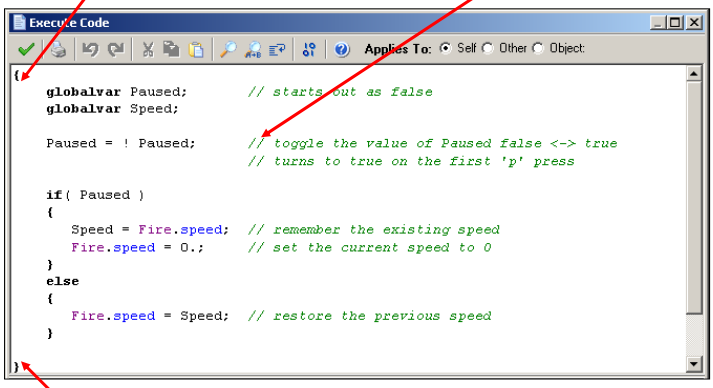
Scripting User Interface



The Structure of a Script Action

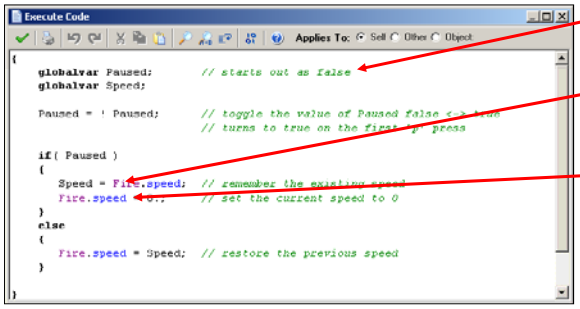
Scripts begin with a left curly brace {

Comments begin with a // and go to the end of the line



Scripts end with a right curly brace }

Pay Attention to Game Maker's Automatic Color-coding when you Enter a Script – this helps prevent typos



Comment: green

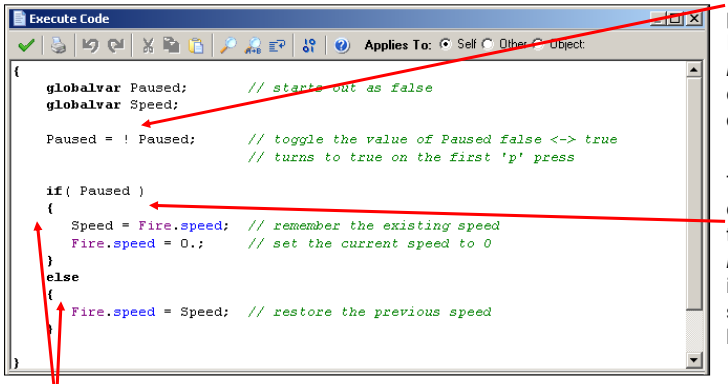
Object name: purple

Property name: blue

If these colors don't come up, then you've spelled something wrong!

Beware: names of things in scripts are all *case-sensitive*. That is, 'a' ≠'A'

Implementing a Pause feature with a Script



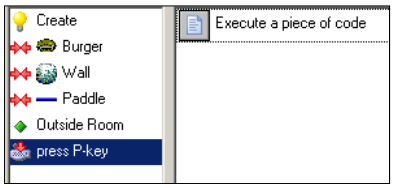
The exclamation point means "not". I.e., whatever ***Paused*** is now, change it to the other state.

The ***if*** statement causes something to happen if ***Paused*** is true. If it's not, then something ***else*** happens.

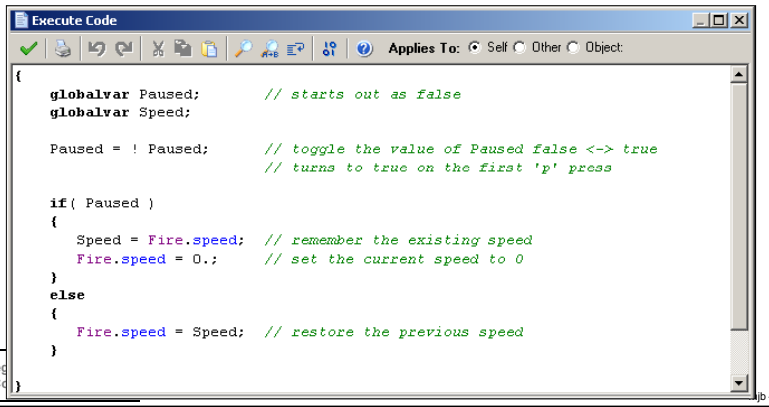
if and ***else*** statement bodies are delimited with curly braces.

The purpose of this script is to allow the 'p' key to pause the action to let you look at the state of your game. This is nice for development. When pausing, the script records the current Fire speed and sets the new Fire speed to 0. When un-pausing, the script restores the Fire speed to what it used to be.

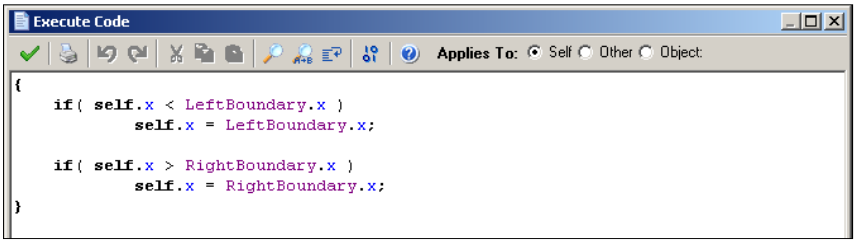
Define the Fire Object's Events



1. **control→Execute Code**

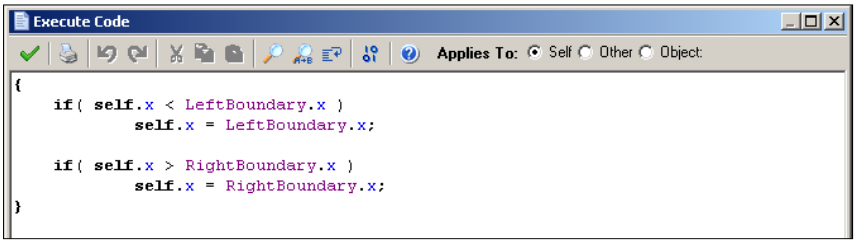


Limiting Motion with a Script

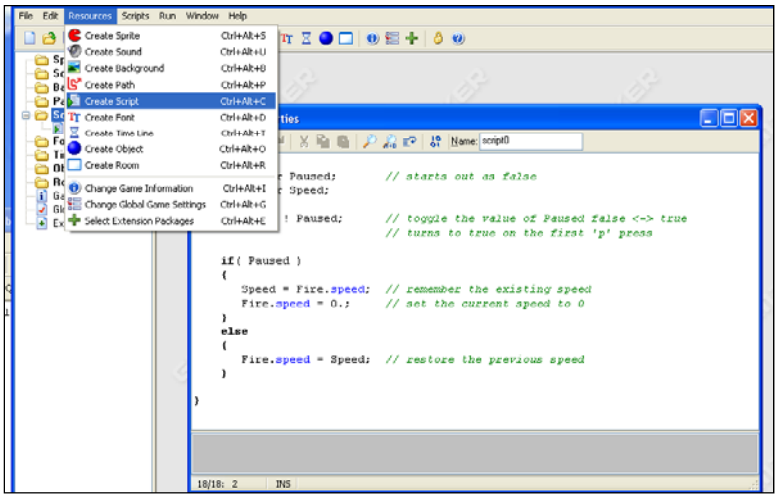


Define the Paddle Object's Events

1. **control→Execute Code**

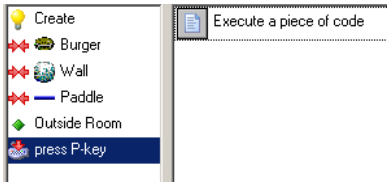


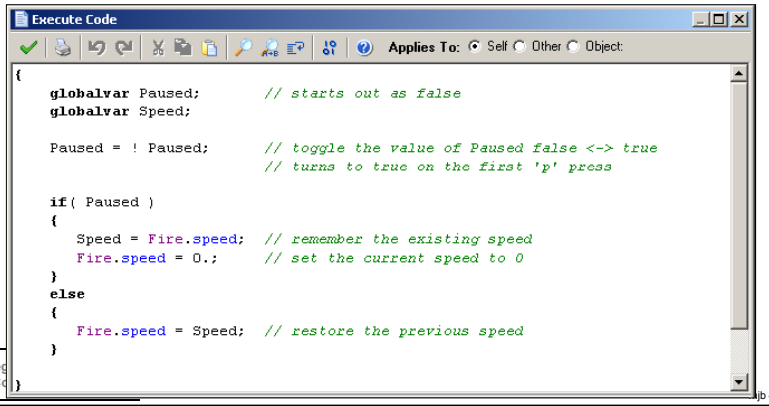
Scripts can be entered as a "Resources → Create Script"



This gives you the chance to name the script, so you can use it in multiple objects

Scripts can also be entered as an "Execute a Piece of Code" Action





General Information

- Game Maker scripts look very much like programming in C, C++, and Java
- Scripts must begin with a left curly brace ({) and end with a right curly brace (})
- Statements end with a semi-colon (;)
- Variable names consist of letters, numbers, and the underscore (_)
- Variable names must begin with a letter
- Letters are case-sensitive, that is 'A' ≠'a'

Functions you can use in Game Maker Scripts

abs(f)	Absolute value of a number
arccos(c)	Arc whose cosine is c
arcsin(s)	Arc whose sine is s
arctan(y_over_x)	Arc whose tangent is y_over_x
arctan2(y, x)	Arc whose tangent is y/x, taking signs into account
ceil(f)	Next highest whole number
cos(f)	Cosine oif r
degtorad(d)	Turn d into radians
exp(f)	e (2.71828…) raised to the f power
floor(f)	Next lowest whole number
frac(f)	Fractional (non-whole number) part of f
ln(f)	Log to the base e (2.71828…) of f
log2(f)	Log to the base 2 of f
log10(f)	Log to the base 10 od f
radtodeg(r)	Turn r into degrees
random(f)	A random number between 0. and f
round(f)	Round f to the nearest whole number
sign(f)	The sign of f (-1. or +1.)
sin(r)	The sin of r
sqr(f)	The square of f
sqrt(f)	The square root of f
tan(r)	The tangent of r

. . . and lots more . . .

General Information

- You can create conditional execution with an 'if-else' block

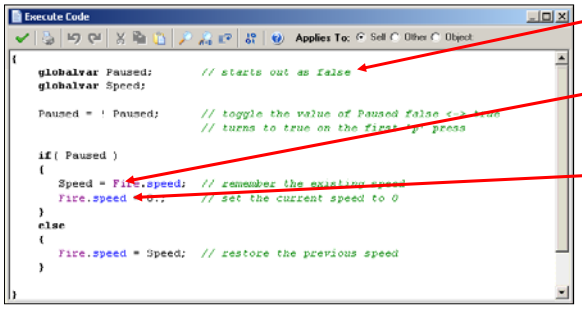
```
if( Paused )
{
    Speed = Fire.speed;  // remember the existing speed
    Fire.speed = 0.;     // set the current speed to 0
}
else
{
    Fire.speed = Speed;  // restore the previous speed
}
```

- You can create a loop with a 'for' block

```
d3d_primitive_begin( pr_linestrip );
for( angle = 0.; angle <= 1440.; angle += 10. )
{
    radians = DegreesToRadians * angle;
    x = R * cos(radians);
    z = R * sin(radians);
    y = K * angle;
    d3d_vertex( x, y, z );
}
```

" for(initial-settings ; keep-going-if-this-is-true ; do-this-in-between-loops) "

Note Game Maker's Automatic Color-coding when you Enter a Script – this helps prevent typos



Comments begin with "//" and are green

Object names are purple

Property names are blue

Special constants are red

Special variables are blue

If these colors don't come up, then you've spelled something wrong!

Beware: names of things in scripts are all *case-sensitive*. That is, 'a' ≠'A'

Particle Systems – Pro Edition Only

A ***Particle System*** is a technique used in computer graphics to simulate the appearance of wispy, fuzzy, or explosive objects with discrete particles. Think of fire, smoke, dust, sand, clouds, tornados, flowing and spraying water, rain, snow, star fields, comets, plants, etc.

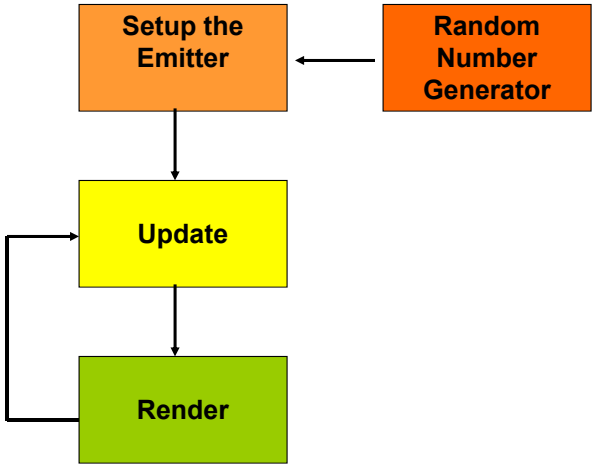
One of the first entertainment uses of a particle system was the Genesis Demo in *Star Trek II: The Wrath of Khan*.

You need the Pro version of Game Maker to use particle systems

More recently, particle systems were used to create the waterfall scene in *Cars*.

The Particle System Process

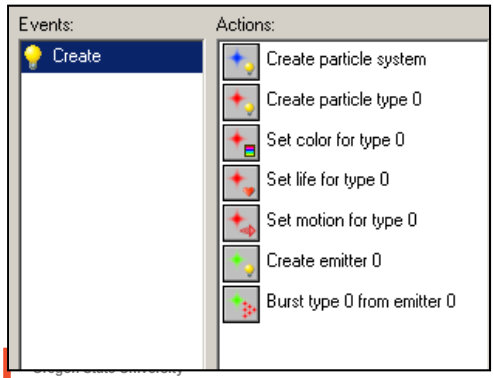
The basic process is:



Particle Systems

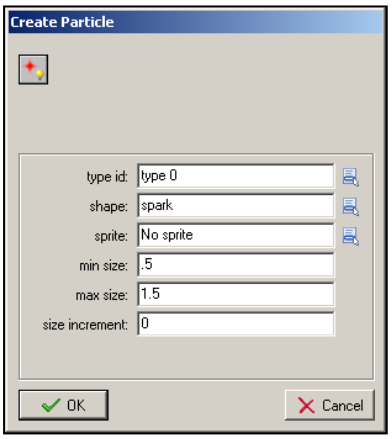
To create a Particle System:

1. Call a Create particle system (under the Extras tab) once, usually from a Create event. At the same time, set the characteristics of the particle type numbers, set the color range, the life expectancy range, and the motion type.
2. Whenever you want the particle system to start displaying, call a Create emitter and Burst or Stream type method.

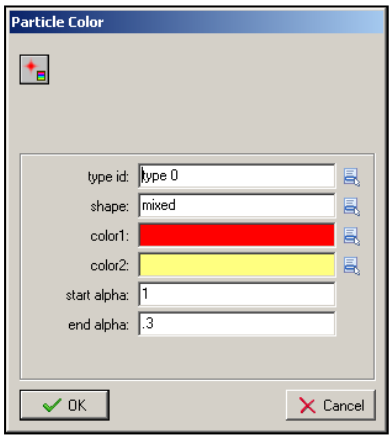


In this case, all actions were placed under this object's Create trigger so the particle system would go off right away.

Particle Systems

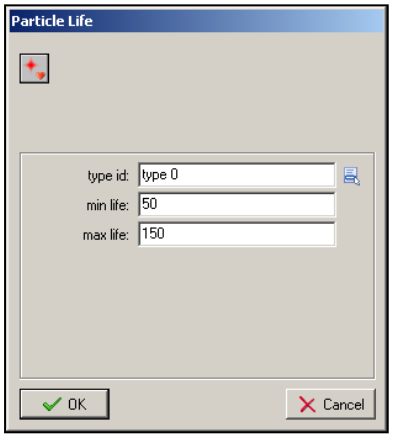


Shape, size, and growth parameters for particle type 0

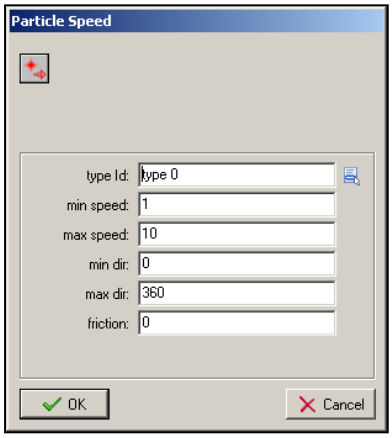


Color and transparency ranges for particle type 0

Particle Systems

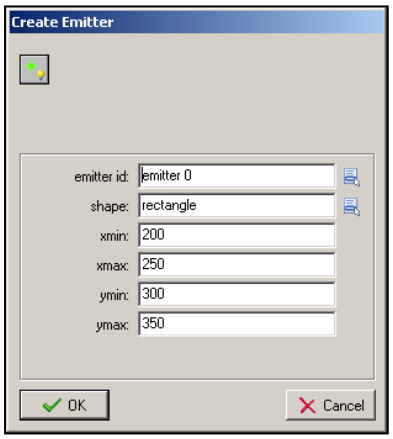


Life expectancy range for particle type 0

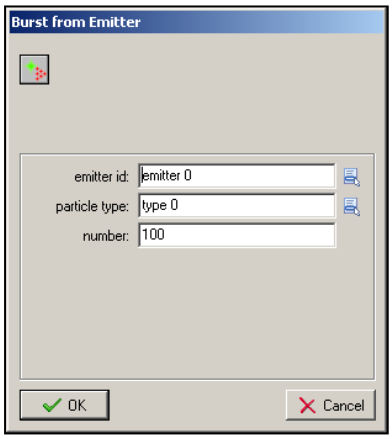


Speed and direction ranges for particle type 0

Particle Systems



Where to emit these particles from



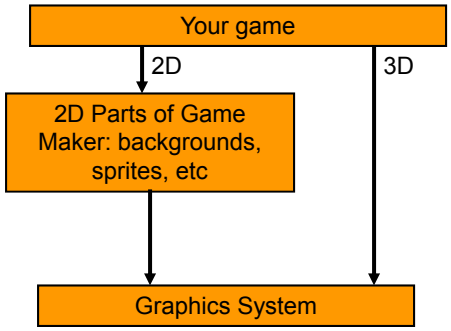
How many particles to burst forth (you can also have them continuously stream)

Particle Systems

What it looks like

Game Maker 3D – Pro Edition Only

1. Only works with the Pro version of Game Maker
2. Doesn't interoperate with the 2D graphics part of Game Maker
3. Is script-based only.
4. You place an initialization script in an object's Create event.
5. You place a re-draw script in an object's Draw event.

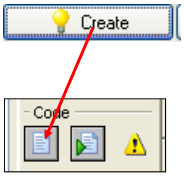


Good References for Game Maker 3D:

http://cs.oregonstate.edu/~mjb/gamemaker
http://gamemaker.wikia.com/wiki/Game_Maker_and_3D
http://tysonc.net/tutorials/gm6-d3d-from-the-ground-up

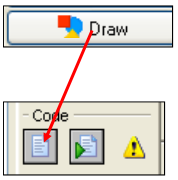
Sample Initialization Script

```
{
    globalvar RotY;

    d3d_start();

    d3d_set_projection_ortho( 0., 0., room_width, room_height, 0. );
    d3d_set_perspective( true );

    RotY = 0.;
}
```



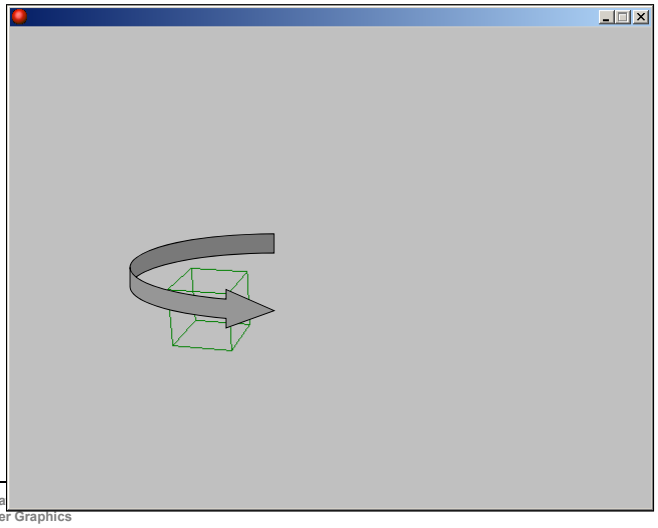
Sample Draw Script

```
{
    globalvar RotY;

    RotY += 10.;
    d3d_transform_set_rotation_y( RotY );
    d3d_transform_add_rotation_x( 20. );
    d3d_transform_add_translation( 200., 200., 0. );

    draw_set_color( c_green );
    d3d_primitive_begin( pr_linestrip );
        d3d_vertex( -30., -30., -30. );
        d3d_vertex(  30., -30., -30. );
        d3d_vertex(  30.,  30., -30. );
        d3d_vertex( -30.,  30., -30. );
        d3d_vertex( -30., -30., -30. );
        d3d_vertex( -30., -30.,  30. );
        d3d_vertex(  30., -30.,  30. );
        d3d_vertex(  30.,  30.,  30. );
        d3d_vertex( -30.,  30.,  30. );
        d3d_vertex( -30., -30.,  30. );
    d3d_primitive_end();

    d3d_primitive_begin( pr_linelist );
        d3d_vertex(  30., -30., -30. );
        d3d_vertex(  30., -30.,  30. );
        d3d_vertex(  30.,  30., -30. );
        d3d_vertex(  30.,  30.,  30. );
        d3d_vertex( -30.,  30., -30. );
        d3d_vertex( -30.,  30.,  30. );
    d3d_primitive_end();
}
```



What Does This Program Do?

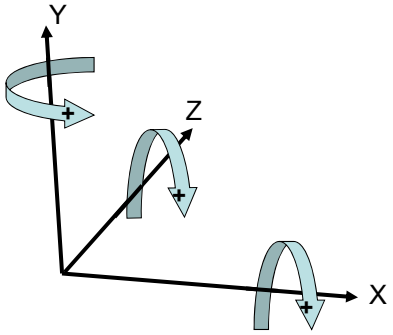
The wireframe cube rotates in 3D



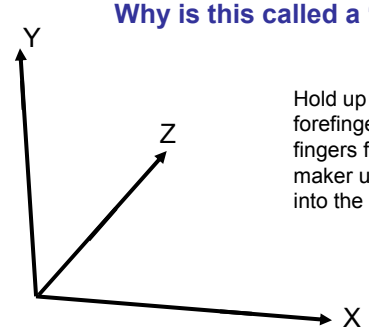
How Does it Do it?

Let's start with Game Maker's coordinate system conventions

Game Maker 3D uses a *Left-handed Coordinate System* with *Right-handed Rotations*, like this:

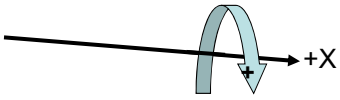


Why is this called a "Left-handed Coordinate System"?



Hold up your left hand. Think of your thumb as X, your forefinger as Y, and your middle finger as Z. Those 3 fingers form an axis system that looks like the one Game maker uses. X goes to the right, Y goes up, and Z goes into the screen.

Why is this called "Right-handed Rotations"?

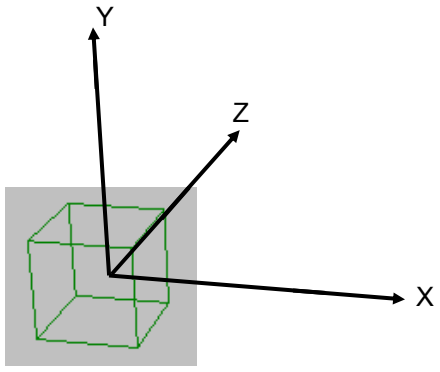


Hold up your right hand and place your thumb in the direction of the +X axis. The way your fingers naturally want to curl shows the direction that is considered a positive rotation. Try this on the other 2 axes.

These are just the conventions that Game Maker has chosen to use. There is nothing magical about them. Other graphics systems use different conventions.

Back to "How Does it Do it?"

Now draw a 3D box:

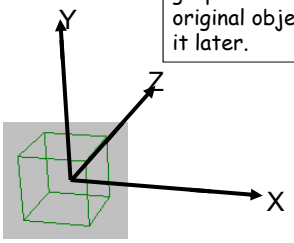


How Does it Do it?

You draw the 3D box with a 10-point *Linestrip* and 3 lines in a *Linelist*

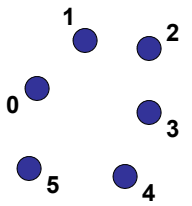
To see what a *Linestrip* and *Linelist* are, go to the next page…

Note that the box is drawn around the origin, that is, the X, Y, and Z coordinates are both positive and negative. Even if you don't want the box to end up at the origin, in computer graphics we typically draw the original object there and move it later.

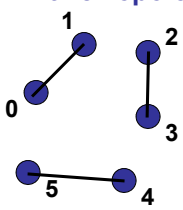



```
d3d_primitive_begin( pr_linestrip );
    d3d_vertex( -30., -30., -30. );
    d3d_vertex(  30., -30., -30. );
    d3d_vertex(  30.,  30., -30. );
    d3d_vertex( -30.,  30., -30. );
    d3d_vertex( -30., -30., -30. );
    d3d_vertex( -30., -30.,  30. );
    d3d_vertex(  30., -30.,  30. );
    d3d_vertex(  30.,  30.,  30. );
    d3d_vertex( -30.,  30.,  30. );
    d3d_vertex( -30., -30.,  30. );
d3d_primitive_end();

d3d_primitive_begin( pr_linelist );
    d3d_vertex(  30., -30., -30. );
    d3d_vertex(  30., -30.,  30. );
    d3d_vertex(  30.,  30., -30. );
    d3d_vertex(  30.,  30.,  30. );
    d3d_vertex( -30.,  30., -30. );
    d3d_vertex( -30.,  30.,  30. );
d3d_primitive_end();
```

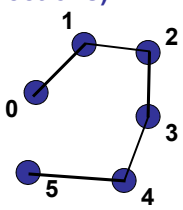

Geometric Primitive Topologies (Connections)



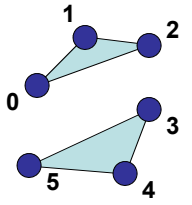
pr_pointlist



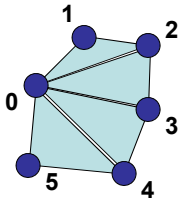
pr_linelist



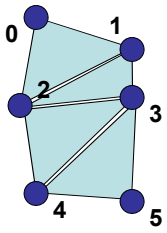
pr_linestrip



pr_trianglelist



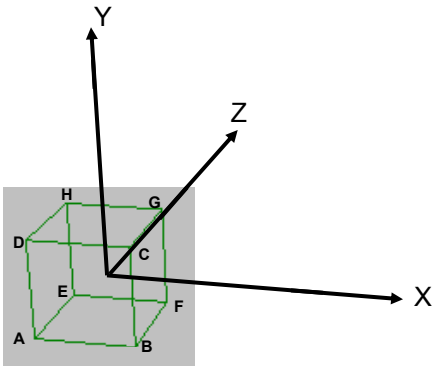
pr_trianglefan



pr_trianglestrip


```
d3d_primitive_begin( pr_linestrip );
```

Connecting the Dots



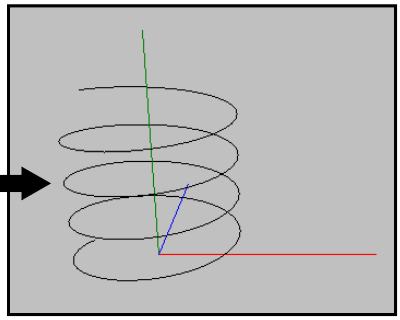

```
d3d_primitive_begin( pr_linestrip );
  A d3d_vertex( -30., -30., -30. );
  B d3d_vertex(  30., -30., -30. );
  C d3d_vertex(  30.,  30., -30. );
  D d3d_vertex( -30.,  30., -30. );
  A d3d_vertex( -30., -30., -30. );
  E d3d_vertex( -30., -30.,  30. );
  F d3d_vertex(  30., -30.,  30. );
  G d3d_vertex(  30.,  30.,  30. );
  H d3d_vertex( -30.,  30.,  30. );
  E d3d_vertex( -30., -30.,  30. );
d3d_primitive_end();

d3d_primitive_begin( pr_linelist );
  B d3d_vertex(  30., -30., -30. );
  F d3d_vertex(  30., -30.,  30. );
  C d3d_vertex(  30.,  30., -30. );
  G d3d_vertex(  30.,  30.,  30. );
  D d3d_vertex( -30.,  30., -30. );
  H d3d_vertex( -30.,  30.,  30. );
d3d_primitive_end();
```


You Don't Have to Manually List All Coordinates

Try computing them to make a spiral!

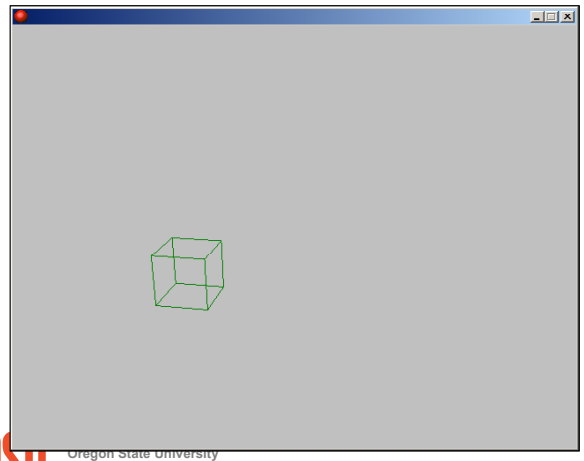
```
{
    R =  75.;           // radius of the spiral
    K = 0.10;           // how much to spiral up
    L = 200.            // axis length
    PI = 3.14159265;
    DegreesToRadians = PI/180.; // convert degrees to radians


    d3d_transform_set_rotation_y( mouse_x );
    d3d_transform_add_rotation_x( mouse_y );
    d3d_transform_add_translation( 200., 200., 0. );

    draw_set_color( c_black );

    // draw the spiral:

    d3d_primitive_begin( pr_linestrip );
    for( angle = 0.; angle <= 1440.; angle += 10. )
    {
        radians = DegreesToRadians * angle;
        x = R * cos(radians);
        z = R * sin(radians);
        y = K * angle;
        d3d_vertex( x, y, z );
    }
    d3d_primitive_end();
}
```

A spiral is just a circle that gradually lifts up in the air. We use sines and cosines to get the points on the circle. Note that the arguments to the sin and cos functions are in *radians*, not *degrees*! (This is consistent with all programming languages.)

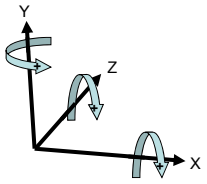


Transformations

```
globalvar RotY;

RotY += 10.;
d3d_transform_set_rotation_y( RotY );
d3d_transform_add_rotation_x( 20. );
d3d_transform_add_translation( 200., 200., 0. );
```



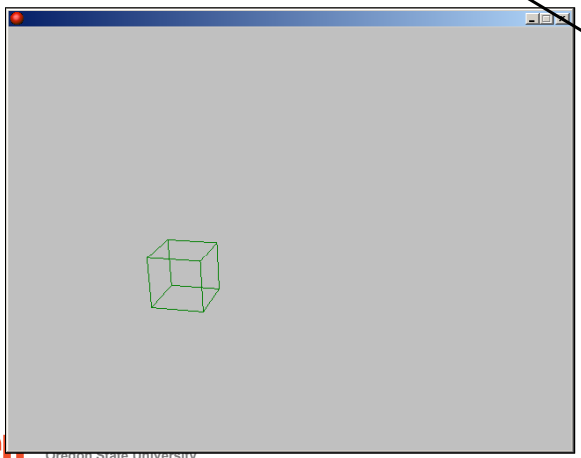
There is a computer graphics concept called the *Current Transformation* (CT) that is the accumulation of multiple movements. Every (X,Y,Z) that you ask to have drawn gets modified by the CT before it is really drawn.



Transformations

```
globalvar RotY;

RotY += 10.;
d3d_transform_set_rotation_y( RotY );
d3d_transform_add_rotation_x( 20. );
d3d_transform_add_translation( 200., 200., 0. );
```

1. Set the CT to a rotation about the Y (vertical) axis
2. Add to that a rotation about the X (horizontal) axis
3. Add to that a translation away from the corner of the screen



Because the value of *RotY* is increasing each time the box is drawn, the box will rotate on the screen.

The phrase:

RotY += 10.;

Means "increment RotY by 10." and is the same as saying:

RotY = RotY + 10.;

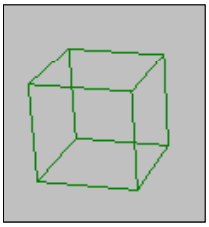
You Can Also Hook Transformations Up to the Mouse

```
d3d_transform_set_rotation_y( mouse_x );
d3d_transform_add_rotation_x( mouse_y );
d3d_transform_add_translation( 200., 200., 0. );
```

mouse_x and *mouse_y* are built-in Game Maker variables that tell you the current mouse position

Setting Colors

```
draw_set_color( c_green );
```



There are some built-in color names, such as:

c_red
c_yellow
c_green
c_blue
c_white
c_black
c_gray
c_silver

You can also create your own color mixes with a call to:

make_color_rgb(r, g, b);

where 0 ≤ r,g,b ≤ 255

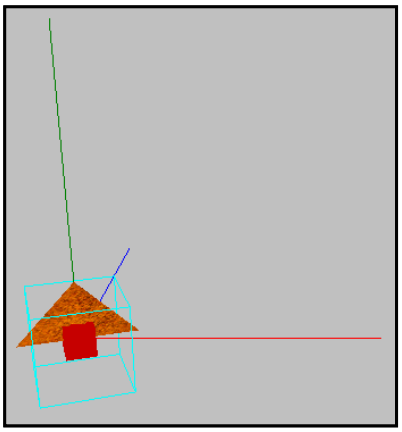
So, you could make orange by saying:

orange = make_color_rgb(255, 128, 0);
draw_set_color(orange);

A Handy Way to Envision Your 3D Scene

Create X, Y, and Z axes in red, green, and blue

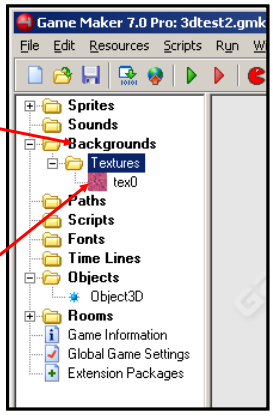
```
d3d_primitive_begin( pr_linelist );
    draw_set_color( c_red );
    d3d_vertex( 0., 0., 0. );
    d3d_vertex( L, 0., 0. );

    draw_set_color( c_green );
    d3d_vertex( 0., 0., 0. );
    d3d_vertex( 0, L, 0. );

    draw_set_color( c_blue );
    d3d_vertex( 0., 0., 0. );
    d3d_vertex( 0, 0., L );
d3d_primitive_end();
```



Textures

A computer graphics “texture” is an image that can be stretched onto a piece of geometry. It makes the scene more realistic, or at least more interesting. To create a texture in Game Maker:

1. Right-click on the *Backgrounds* word here and select *Create Group*. Provide the group name *Textures* (this keeps your texture images separate from your background images)
2. Go to *Resources→Create Background*
3. Select an image. Give it a unique name, *tex0* in this case.



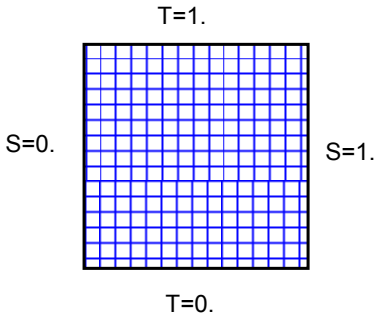
Textures

The only trick to using an image as a texture is that texture images must have pixel dimensions that are powers-of-two. For example, 64x64 or 128x64 or 256x64 would all work. 65x127 would not.

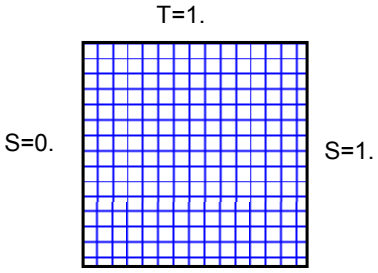
(This is a graphics card limitation, not a Game Maker limitation.)

Then, regardless of the actual pixel dimensions of the texture image:

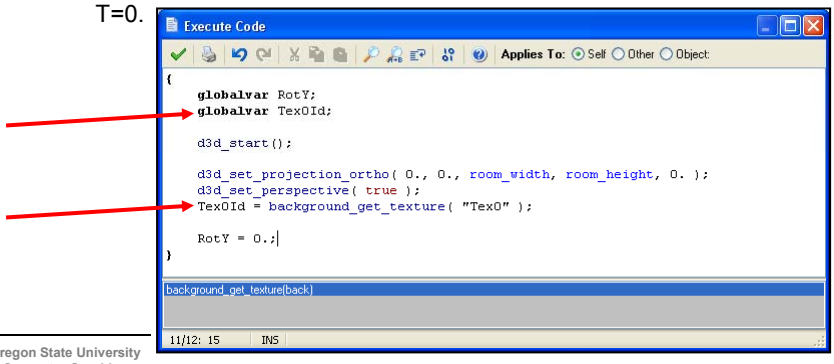
The left edge is called S = 0.
The right edge is called S = 1.
The bottom edge is called T = 0.
The top edge is called T = 1.



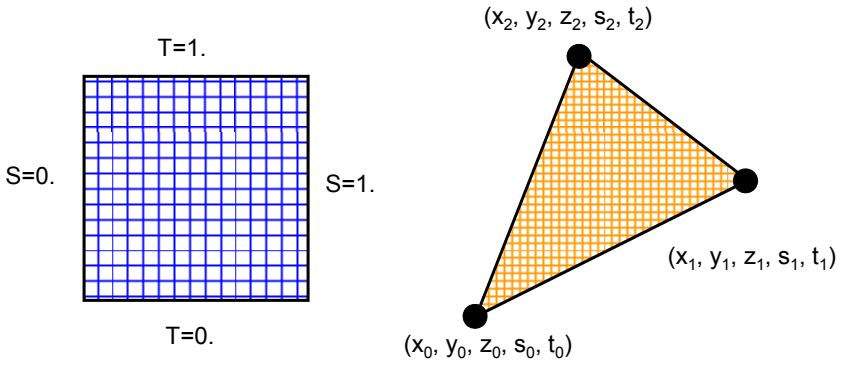
Textures



In the Create event action, define a "Texture ID" with the image you want to use.

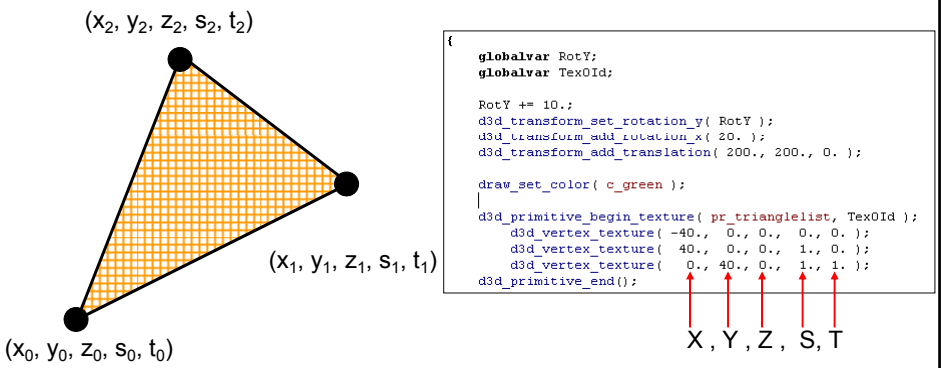


Textures



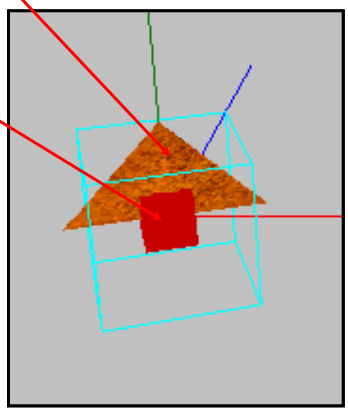
Then, when drawing an object, give each vertex an S and T coordinate in addition to X, Y, and Z

Textures



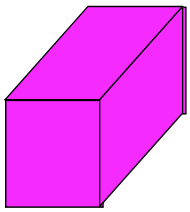
Then, when drawing an object, give each vertex an S and T coordinate in addition to X, Y, and Z

```
draw_set_color( c_yellow );
d3d_primitive_begin_texture( pr_trianglelist, TexOId );
    d3d_vertex_texture( -40.,  0., 0.,   0., 0. );
    d3d_vertex_texture(  40.,  0., 0.,   1., 0. );
    d3d_vertex_texture(   0., 40., 0.,   1., 1. );
d3d_primitive_end();

draw_set_color( c_red );
d3d_draw_block( -10., -10., -10.,  10., 10., 10.,  TexOId, 0, 0 );
```



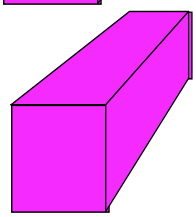
Projections

All 3D computer graphics programs need to project from the 3D of your program to the 2D of the screen. You, the programmer, need to help it do that. There are two major kinds of projections.



Orthographic – lines that are parallel in 3D remain parallel in 2D. Things don't get smaller as they get farther away from you.

Easier to line things up



Perspective – lines that are parallel in 3D don't necessarily remain parallel in 2D. Things do get smaller as they get farther away from you.

More realistic

Projections

All 3D computer graphics programs need to project from the 3D of your program to the 2D of the screen. You, the programmer, need to help it do that. There are two major kinds of projections. You usually specify the projection you want in the Create event action:

Sets size of the view

Turns perspective on