

Tutorial: Your first game

Copyright 2003-2004, Mark Overmars

Last changed: September 1, 2004

Uses: version 6.0, simple mode

Level: Beginner

This tutorial is meant for those that have problems getting started with *Game Maker*. It will lead you step by step through the process of making your first game. Realize that this is the most difficult part. To make your first game you have to understand a number of the basic aspects of *Game Maker*. So please read this tutorial carefully and try to understand all the steps. Once you finished your first game the second one is a lot easier.

The game idea

It is important that we first write a brief description of the game we are going to make. Because this is going to be our first game we better design something simple. It should keep the player interested for just a short time. Our game is going to be a little action game that we will name *Catch the Clown*. (Always try to come up with a nice name for you game.) Here is our description of the game:

Catch the Clown

Catch the Clown is a little action game. In this game a clown moves around in a playing field. The goal of the player is to catch the clown by clicking with the mouse on him. If the player progresses through the game the clown starts moving faster and it becomes more difficult to catch him. For each catch the score is raised and the goal is to get the highest possible score. Expected playing time is just a few minutes.

Clearly, a game like this will have limited appeal. But we have to start simple. Later we can add some features to the game to make it more interesting.

A design document

The second step in creating a game is to write a more precise design document. You are recommended to always do this before making your game, even if it is very simple. Here is our design document for *Catch the Clown*. (I omitted the description that was already given above.)

Catch the Clown *design document*

Game objects

There will be just two game objects: the clown and the wall. The *wall* object has a square like image. The wall surrounding the playing area is made out of these objects. The wall object does nothing. It just sits there to stop the clown from moving out of the area. The *clown* object has the image of a clown face. It moves with a fixed speed. Whenever it hits a wall object it bounces. When the player clicks on the clown with the mouse the score is raised with 10 points. The clown jumps to a random place and the speed is increased with a small amount.

Sounds

We will use two sounds in this game. A bounce sound that is used when the clown hits a wall, and a click sound that is used when the player manages to click with the mouse on the clown.

Controls

The only control the player has is the mouse. Clicking with the left mouse button on the clown will catch it.

Game flow

At the start of the game the score is set to 0. The room with the moving clown is shown. The game immediately begins. When the player presses the <Esc> key the game ends.

Levels

There is just one level. The difficulty of the game increases because the speed of the clown increases after each successful catch.

That should be good enough for the moment. We can now start creating the game. So start up *Game Maker* and let's get going. Note that this tutorial uses version 6.0 of *Game Maker*. If you use a different version, the images look a bit different. It also assumes the program runs in simple mode. You can switch between simple and advanced mode by clicking on the menu item **Advanced Mode** in the **File** menu. In advanced mode there are many more options in the different menus and forms but we won't need these for our simple game.

Actually, the game we are going to create is given in the folder `example` that comes with this tutorial, together with all the sprites, sounds, etc. You can load it from there but you are recommended to recreate it by following the steps described below. In this way you will better understand how a game is being made in *Game Maker*.

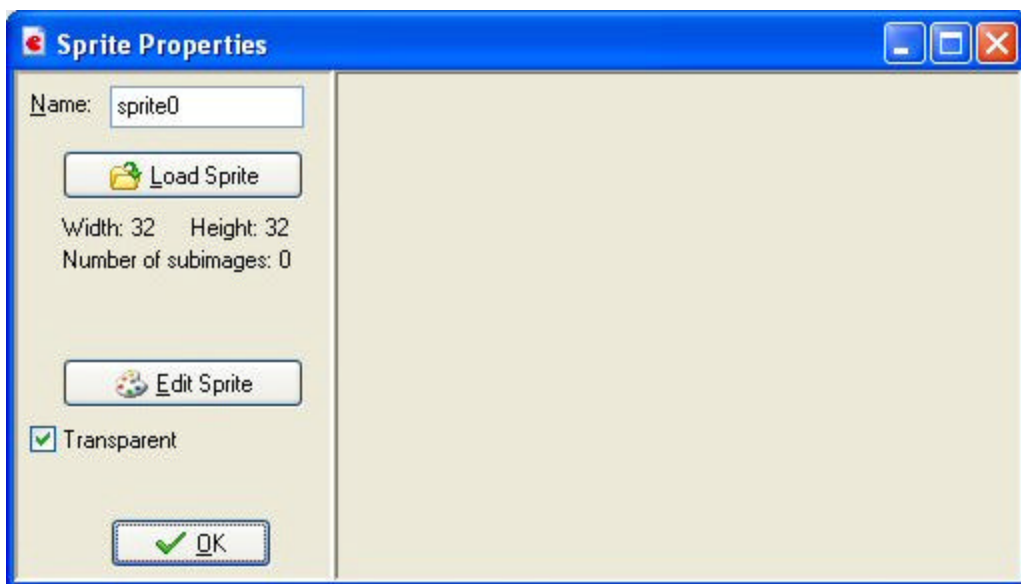
Adding sprites and sounds

As the game design document describes we will need two images for the two game objects. Such images are called sprites in *Game Maker*. There is a lot to know about sprites but for the moment, simply think of them as little images. So we need to make or find such images. For making the images you can use any drawing program you like, for example the paint program that is part of any Windows system. But *Game Maker* also has

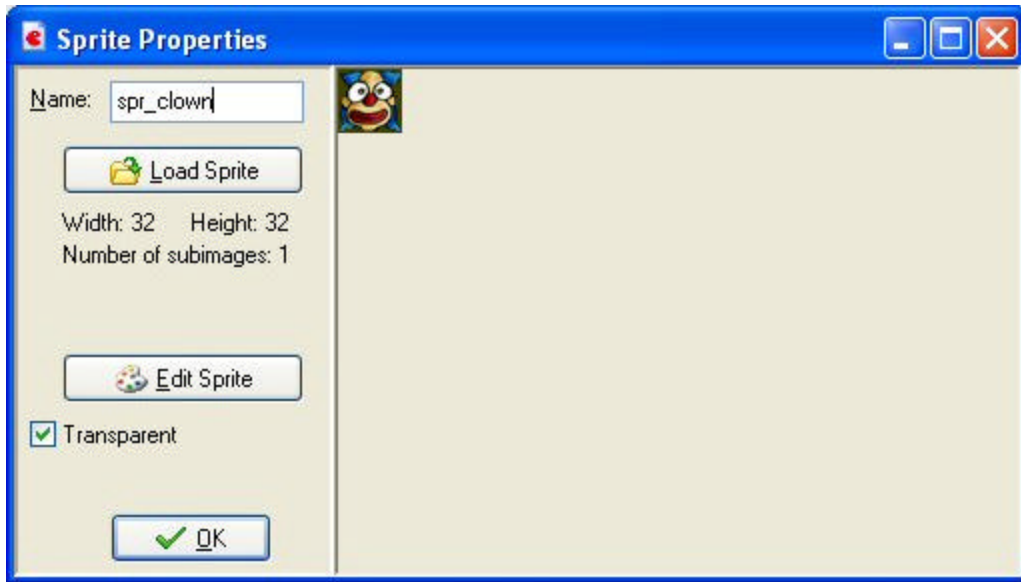
a simple built-in drawing program for this purpose. Creating nice-looking sprites is an art that requires a lot of practice. But fortunately there are large collections of images of all sorts available for free. *Game Maker* contains a number of these and on the *Game Maker* web site (www.gamemaker.nl) you can find many more. Alternatively, search the web and you are bound to find images in large quantities. So for our little game we take two sprites from the default collection.

The clown:  The wall: 

To add these sprites to the game we are creating, from the menu labeled **Add** choose **Add Sprite**. The following form will appear:



Press the button labeled **Load Sprite** and pick the file that contains the clown image. (You can find them in the folder `resources` in the `example` folder.) The sprite will be shown in the form. Fill in the **Name** field to give the sprite a nice name, like `spr_clown`. (I always let the names of the sprites start with `spr_` such that I will always know these are names of sprites. Such naming conventions are good practice.) Note that the property **Transparent** is checked. This indicates that the background color of the sprite will actually be transparent (so you will see the real background through this). You in general want your sprites to have a transparent background.

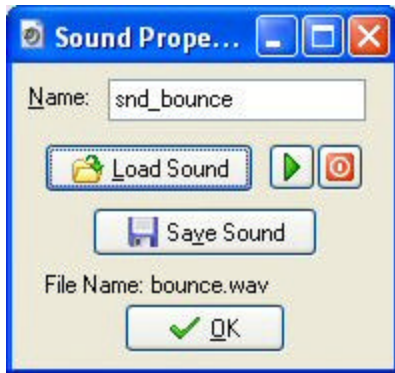


Press the button labeled **OK** and the sprite is added. You can do the same with the wall sprite, naming it `spr_wall`.

As you might have noticed, at the left of the *Game Maker* window the names of your two sprites have now appeared. Here you will always find all the sprites, sounds, objects, rooms, etc. that you have defined. Together we call them the *resources* of the game. You can select a resource by clicking on its name. Now you can use the menu **Edit** to change the resource, duplicate it, or delete it. Right-clicking on the resource name will show the same menu. This overview of resources will become crucial when you are creating more complicated games.

Now that we added the sprites we will add two sound effects. One must play when the clown hits a wall and the other must play when the clown is successfully caught with the mouse. We will use two wave files for this. Wave files are excellent for short sound effects. A number of these effects are part of the installation of *Game Maker* and many more can be found on the web. You can also create them yourself but that definitely requires some skill.

To add the first bounce sound to the game, choose **Add Sound** from the **Add** menu. A form appears in which you can change the properties of the sound. Press the button labeled **Load Sound** to load the correct sound. Also we name the sound `snd_bounce`. The sound property form now looks as follows:



You can use the play button, with the green triangle pointing to the right, to check out the sound. Press **OK** to indicate that you are ready with this sound. You can add the click sound in exactly the same way, naming it `snd_click`. Again, notice that the two sounds are shown at the left of the window in the overview of all resources

Objects and actions

Having added the sprites and sounds to the game does not mean that anything is happening. Sprites are only the images for game objects and we have not yet defined any game objects. Similar, sounds will only play if we tell them to be played. So we need to define our two game objects next.

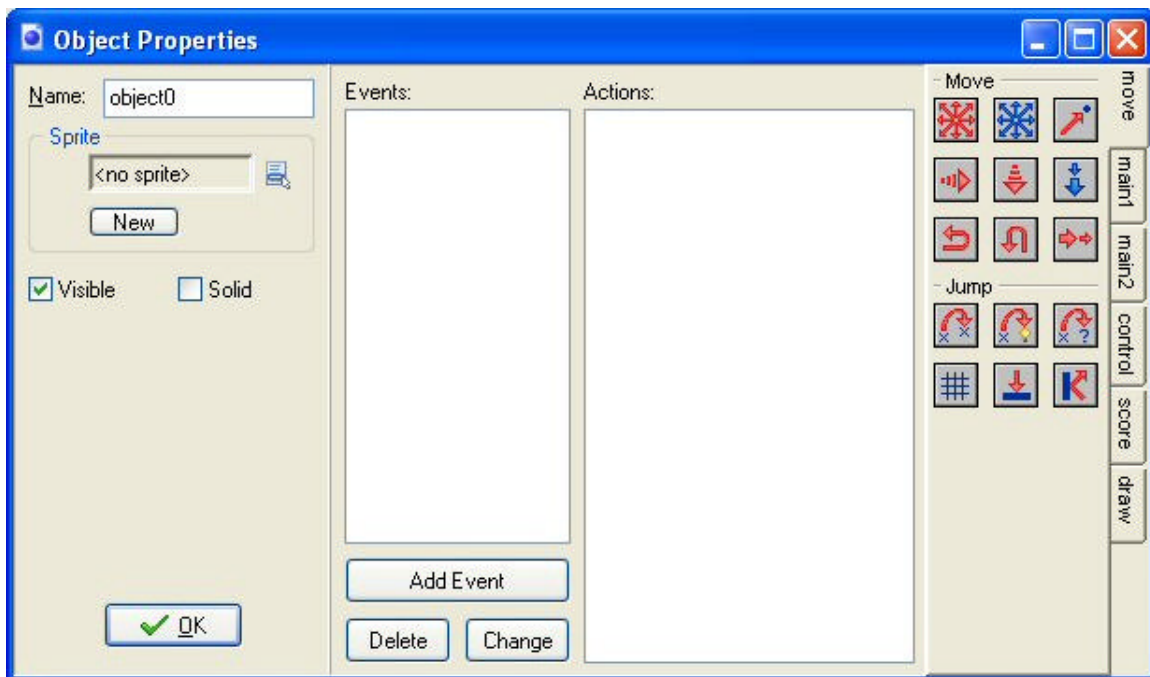
But before we will do this you will have to understand the basic way in which *Game Maker* and most other game design systems operate. As we have indicated before, in a game we have a number of different *game objects*. During the running of the game one or more *instances* of these game objects will be present on the screen or, more general, in the game world. Note that there can be multiple instances of the same game object. So for example, in our *Catch the Clown* game there will be a large number of instances of wall objects, which surround the playing field. There will be just one instance of the clown object.

Instances of game objects don't do anything unless you tell them to do something. You do this by indicating how the instances of the object must react to *events* that happen. There are many different events that can happen. The first important event is when the instance is created. This is the **Creation Event**. Probably some reaction is required here. For example we must tell instance of the clown object that it should start moving in a particular direction. Another important event happens when two instances collide with each other; a so-called **Collision Event**. For example, when the instance of the clown collides with an instance of the wall, the clown must react and change its direction of motion. Again other events happen when the player presses a key on the keyboard or clicks with mouse on an instance. For the clown we will use a **Mouse Event** to make it react to a press of the mouse on it.

To indicate what must happen in the case of an event, you specify *actions*. There are many useful actions for you to choose from. For example, there is an action that sets the instance in motion in a particular direction, there is an action to change the score, and there is an action to play sounds. So defining a game object consists of a few aspects: we can give the game object a sprite as an image, we can set some properties, and we can indicate to which events instances of the object must react and what actions they must perform.

Note that I make a distinction between *objects* and *instances* of those objects. An object defines a particular game object with its behavior (that is, reaction to events). Of this object there can be one or more instances in the game. These instances will act according to the behavior. Stated differently, an object is an abstract thing. Like in normal life, we can talk about a chair as an abstract object that you can sit on, but we can also talk about a particular chair, that is an instance of the chair object, which actually exists in our home.

So how does this work out for the game we are creating? We will need two objects. Let us first make the very simple wall object. This object needs no behavior at all. It will not react to any events. To add the object, choose the item **Add Object** from the **Add** menu. (You probably figured out by now that there are also buttons for these commands on the toolbar.) The following rather complicated form will appear:



At the left you can set some properties for the object. We set the name to `obj_wall`. Next we choose the correct sprite. To this end, you must click with the mouse on the button with the menu right below the word **Sprite**. A menu will pop up showing all available sprites (that is, the clown and the wall). Pick the wall sprite for the object. Finally, we want the wall to be solid, that is, we do not allow objects to pass through it.

So check the property **Solid** by clicking with the mouse on it. Press **OK**, and we are done with the wall object.

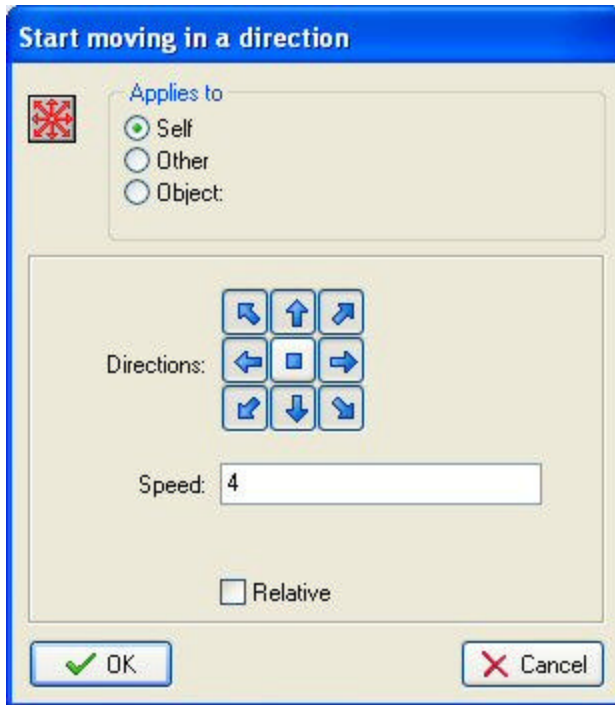
For the clown object we start in the same way. Add a new object, change the name in `obj_clown`, and pick the correct sprite. The clown does not need to be solid. But for the clown there is a lot more that needs to be done. We have to specify its behavior. For this we need the rest of the form. In the middle you see an empty list with three buttons below it. This list will contain the different events that the object must respond to. With the buttons below it you can add events, delete events or change events. There are a large number of different events but you normally need just a few in your game.

Next to the events there is an empty list of actions that must be performed for the selected event (if any). And at the right of this list that are a number of tabbed pages with little icons. These icons represent the different actions. In total there are close to 100 different actions you can choose from. If you hold your mouse above one of the icons a short description of the corresponding action is given. You can drag actions from the pages at the right to the action list to make them happen when the event occurs.

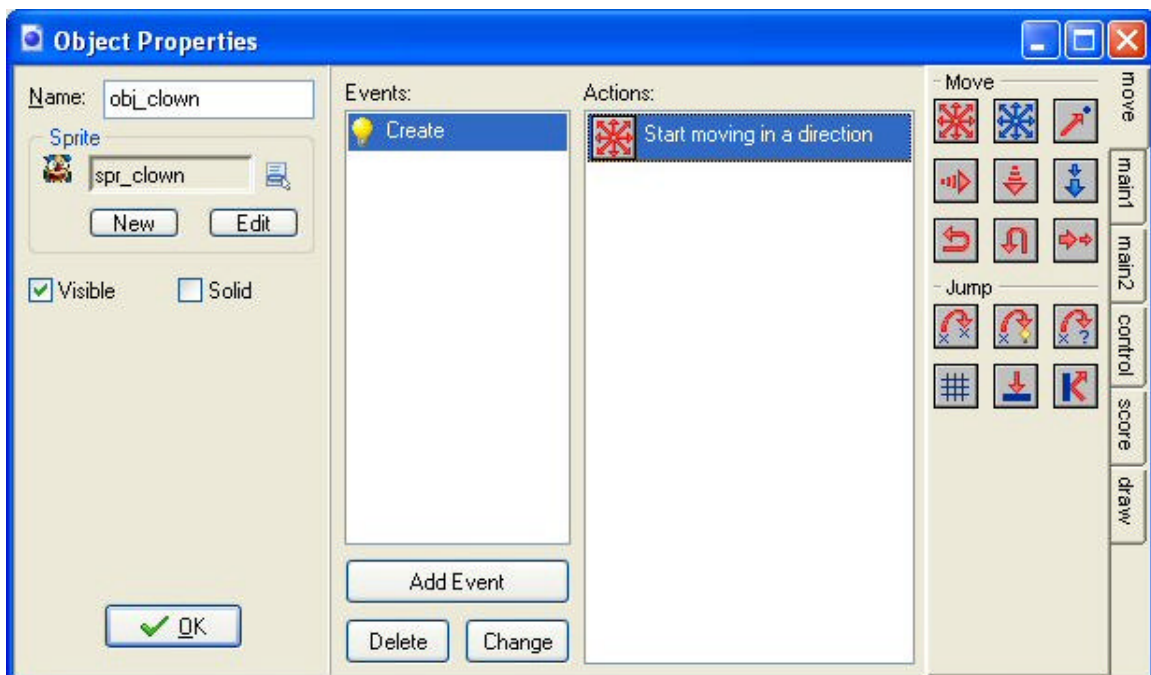
We are first going to define what should happen when an instance of the clown object is created. In this case we want the clown to start moving in an arbitrary direction. So press the button **Add Event**. The follow form pops up:



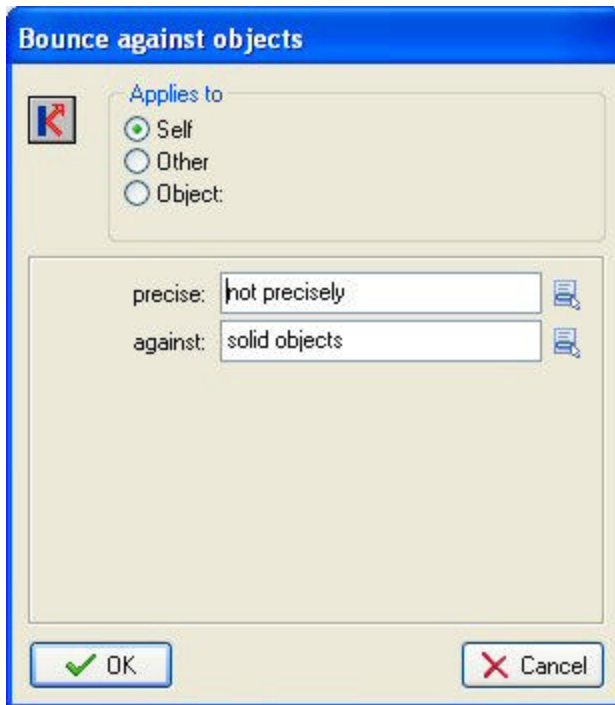
Click on the button **Create** and as you will see the create event is now added to the event list and it is selected. Now drag the icon with the eight red arrows to the list, that is, locate it in the tabbed pages at the right, click with the left mouse button on it and hold the button pressed, and drag it to the empty list of actions. This icon corresponds to the action to let the instance move in a particular direction. The following form will pop up:



Most actions have associated forms in which you can set certain properties for the action. For the moment forget the box with the label **Applies to**. We can specify two properties: the direction in which to move and the speed with which to move. Because we want an arbitrary direction, click on all eight arrows (not the central square; that will stop the motion!). This means that one out of these directions is randomly chosen. The speed we set to 4 otherwise the clown moves a bit too fast. Press **OK** to indicate that we are done with this action. The object property form should now look as follows:



The next event we need to define is a collision with a wall. Again press the button labeled **Add Event** but this time press the button labeled **Collision**. A menu will pop up showing all existing objects (the wall and the sprite). Select the wall object cause that is the one with which we want to handle the collision. We will use the bounce action for this event, which is the icon with the arrow bouncing off from the wall (right bottommost icon on the first tabbed page). Drag it to the action list and the following form will pop up:



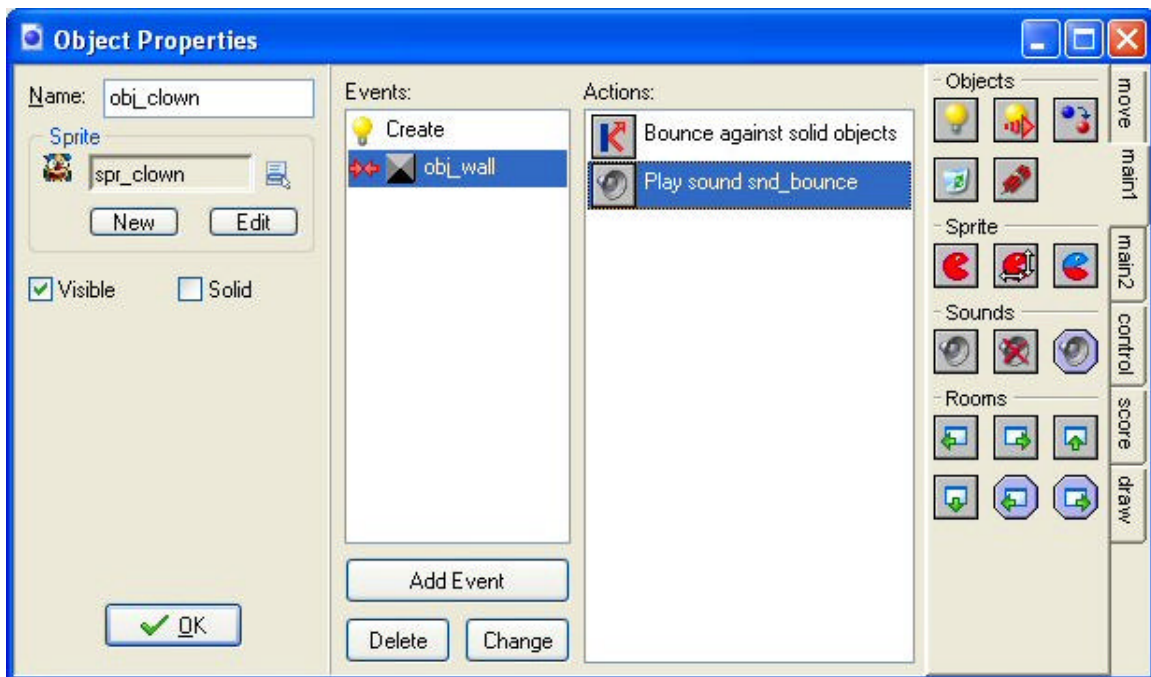
The screenshot shows a dialog box titled "Bounce against objects". It has a blue header bar. Inside, there is a small icon of a red arrow bouncing off a wall. Below this, the "Applies to" section has three radio buttons: "Self" (selected), "Other", and "Object". Below that, there are two text input fields: "precise:" with the value "not precisely" and "against:" with the value "solid objects". Each field has a small icon to its right. At the bottom, there are two buttons: "OK" with a green checkmark and "Cancel" with a red X.

There are two properties we can change but their default values are fine. We are not interested in precise bounces and we want to bounce against solid objects. (Remember that we made the wall object solid.) Press **OK** and we are done.

We also need to add an action to play the bounce sound to the collision event. The sound action can be found on the tabbed page **main1** and looks like a speaker. Drag it to the list and the following form will pop up:



You can again set two properties. First you must pick the sound to play, by clicking on the menu button. We need `snd_bounce` here. (Here you start seeing why it is good to give your resources recognizable names.) The second property is whether to loop the sound, that is, play it continuously. That is obviously not what we want so leave this value to **false**. Press **OK**. The object property form should now look as follows:



There are two actions that are both performed (in the given order) when the collision occurs. If you for some reason made a mistake, you can right-click with the mouse on an action you added and for example choose **Delete** to remove the action (or press the Del key). You can also choose **Edit Values** to change the properties of the action. (Double-clicking on the action will do the same.) And you can drag them up and down to change the order in which they are executed.

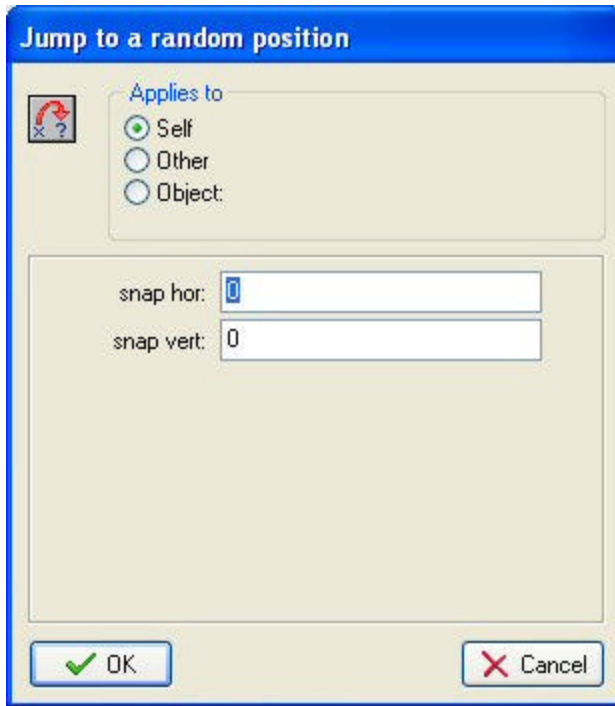
Finally we need to define what to do when the user clicks with the left mouse on the clown. Again click on the button **Add Event** and now choose the **Mouse** events. Again a pop up menu appears with many choices. The one we need is called **Left pressed**. This action happens only when the player presses the left mouse button on the instance of the object. Other mouse events happen all the time while the player keeps the mouse pressed or happen independently from where the player presses the mouse button. We need three actions here. First of all we need to increase the score. *Game Maker* automatically keeps and displays a score. To increase the score drag the action with the three coins on it from the tabbed page named **score**. The following form will appear:



Type a value of 10, as indicated above, because we will increase the score with 10 points. But this is not enough. The action sets the score to 10 but we want to add 10 to the score. This can be achieved by checking the property **Relative**. Many actions have such a relative box to increase a value rather than set it. Press **OK** to close the form.

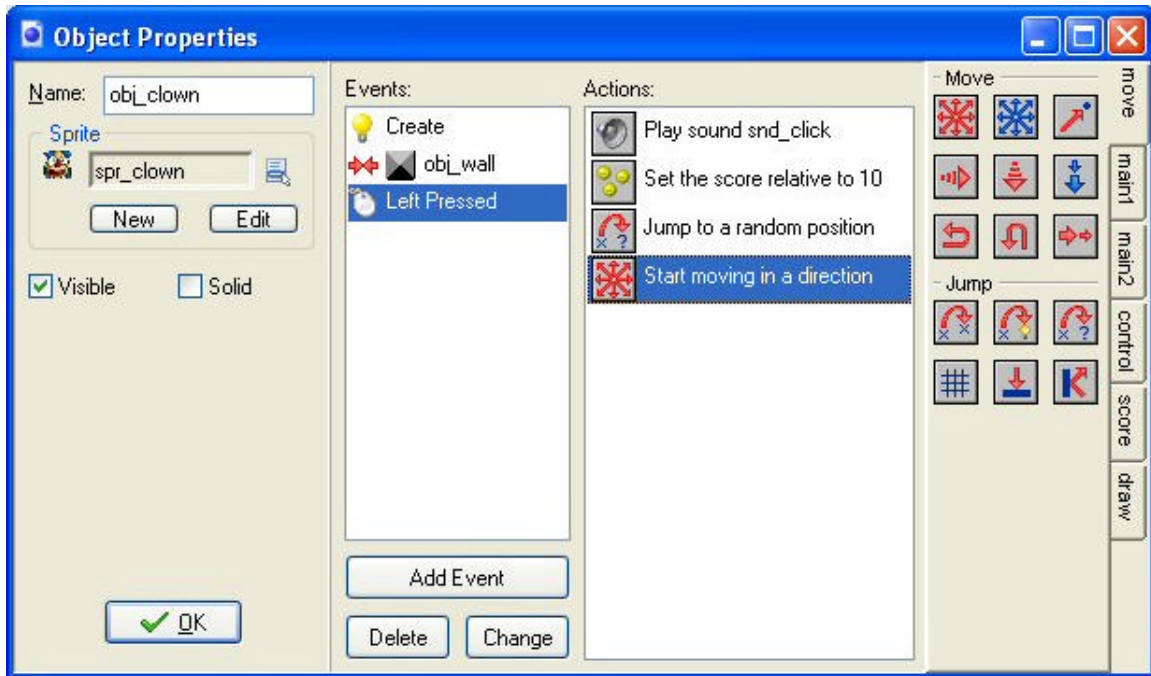
The next action is to play the click sound. This action can be added exactly as we added the bounce sound action. Next we want to make the clown jump to a new random position. This will make the game harder. On the tab page **Move** there is an action that

does this for you. Select the action with the arrow towards the question mark. The following form will pop up:



The image shows a dialog box titled "Jump to a random position". In the top-left corner, there is a small icon of a red arrow pointing to a question mark. Below this icon is a section labeled "Applies to" containing three radio buttons: "Self" (which is selected), "Other", and "Object:". Below the "Applies to" section are two input fields: "snap hor:" with a value of "0" and "snap vert:" with a value of "0". At the bottom of the dialog box are two buttons: "OK" with a green checkmark icon and "Cancel" with a red X icon.

The default settings are fine so press **OK** to close the form. Finally we want to let the clown move in a new direction and increase the speed a bit, as was indicated in the design document. For this we again use the action with the eight red arrows to set the motion. Click again on all eight arrows, type a value of 0.5 for the speed and click on the **Relative** box to add 0.5 to the speed. The form should now look as follows:

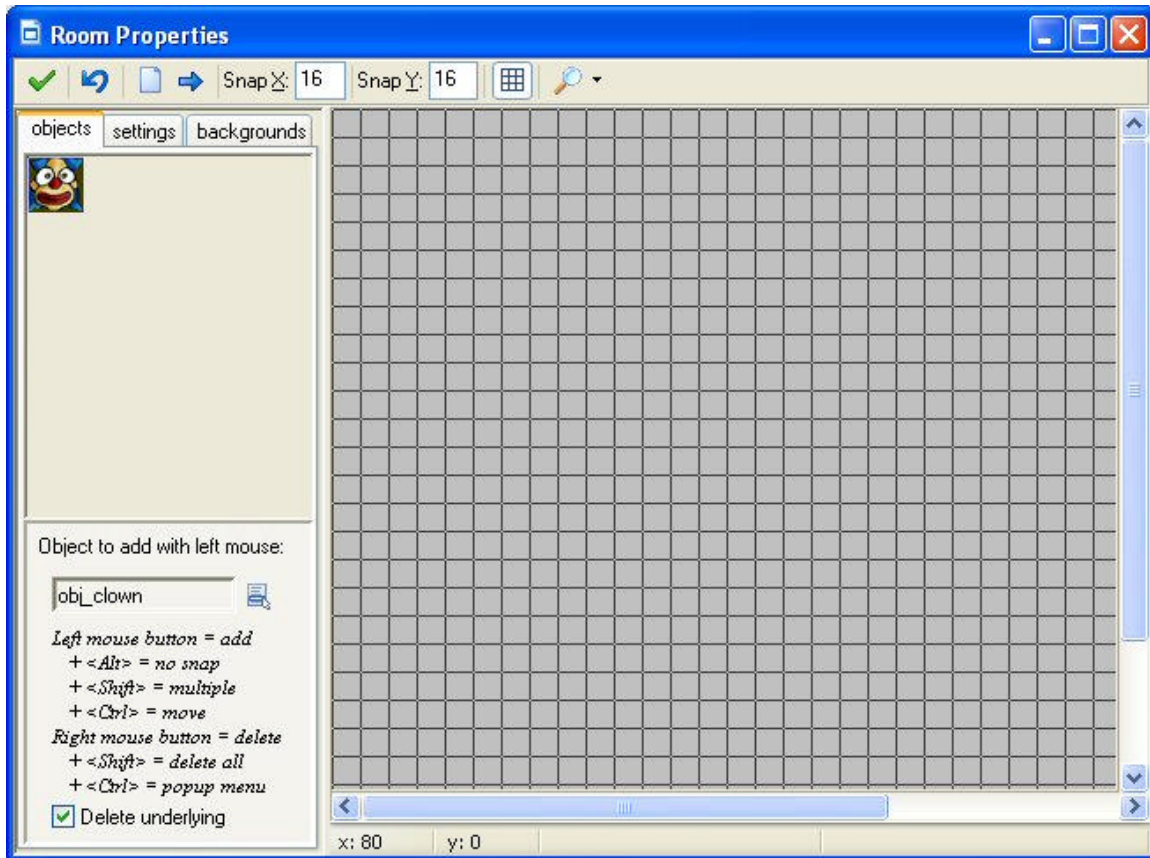


We are now ready with the clown object. We have defined actions for the three events that are important. So press the **OK** button to close the form.

Creating the room

Now that we have created the game objects there is just one more thing to do. We need to create the room in which the game takes place. For most games, designing effective rooms (often also called levels) is a time-consuming task because here we have to find the right balance and progression in the game. But for *Catch the Clown* the room is very simple: a walled area with one instance of the clown object inside it.

To create the room, choose **Add Room** from the **Add** menu. The following form will appear:



At the left you see three tabbed pages. We will only use the page **objects** here. At the right you see the current room, which is just a grey area, with a grid overlaid on it to help you place instances of objects. Because our game objects have size 32x32 we better change the grid a bit. To this end, at the top of the form change the snapping values 16 into 32. You will see that the grid lines are now spaced further apart.

To add instances of a particular object, click on the button with the menu picture. Choose the `obj_wall` object. Now use the mouse to place instances of the wall in the cells at the border of the room. You might have to enlarge the window or use the scroll bars to see the whole room. If you hold the `<Shift>` key while moving the mouse, multiple instances will be added to the room. If you made a mistake you can use the right mouse button to delete instances. Finally, select the `obj_clown` object and place one instance of it in the middle of the room. That is all. Our room is ready. So close the form with the **OK** button; that is the button with the green V sign at the left top of the form.

Saving and testing

You might not have realized it but our game is ready now. The sprites and sounds have been added, the game objects have been designed and the first (and only) room in which the game takes place has been created. Now it is time to save the game and to test it.

Saving the games works as in almost any other Windows program. Choose the command **Save** from the **File** menu, select a location and type in a name. *Game Maker* games get a file extension `.gm6`. (This stands for Game Maker version 6.) Note that you cannot directly play such game files. You can only load them in *Game Maker*. Below we will see how to make stand-alone game executables.

Next we need to test the game. Testing is crucial. You can test it yourself but you should also ask others to test it. Testing (or running the game in general) is simple; choose the command **Run normally** from the **Run** menu. The design window will disappear, the game will be loaded and, if you did not make any mistakes, the room will appear on the screen with the clown moving inside it. Try clicking on it and see whether the game behaves as expected. You should hear the correct sounds and the speed of the clown should increase. To end the game, press the <Esc> key or click on the close button at the top right of the window. The design window will reappear.



Now it is time to tune the game. You should ask yourself for example the following questions: Is the initial speed correct? Is the increase in speed correct? Is the room size correct? Did we pick effective sprites and sounds for the game? If you are not happy, change these aspects in the game and test again. Remember that you should also let someone else test the game. Because you designed the game it might be easier for you than for other people.

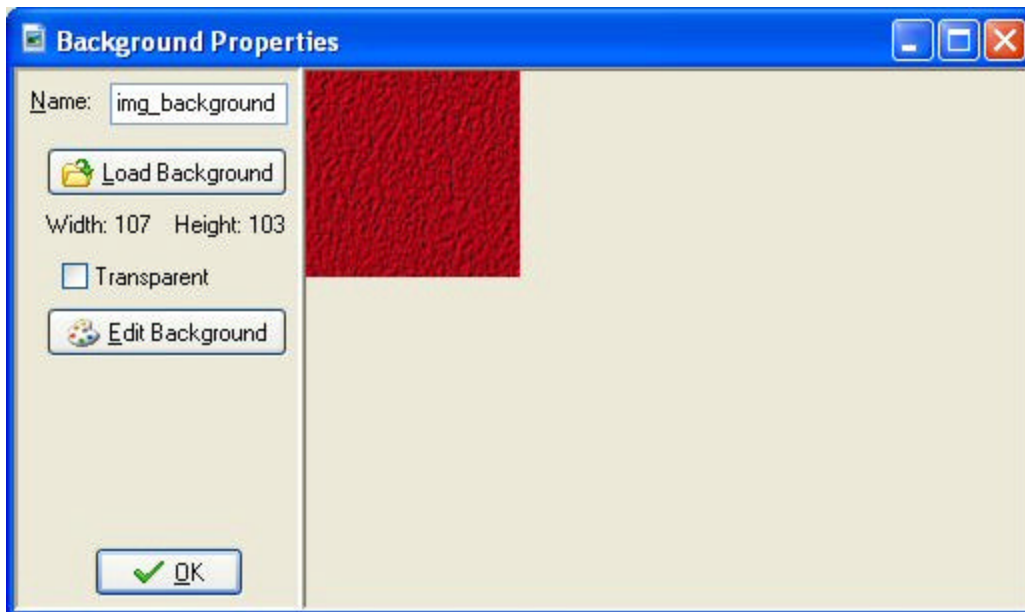
Once you are happy with your game you should create a stand-alone executable for the game. This is a version of the game that can run without the need for *Game Maker*. This is very simple. In the **File** menu choose the command **Create executable**. You have to indicate the place and name of the stand-alone executable and you are done. You can now close *Game Maker*, and run the executable game. You can also give the executable to your friends and let them play it, or put it on your website for people to download.

Finishing touches

Our first game is ready but it needs some finishing touches to make it a bit nicer.

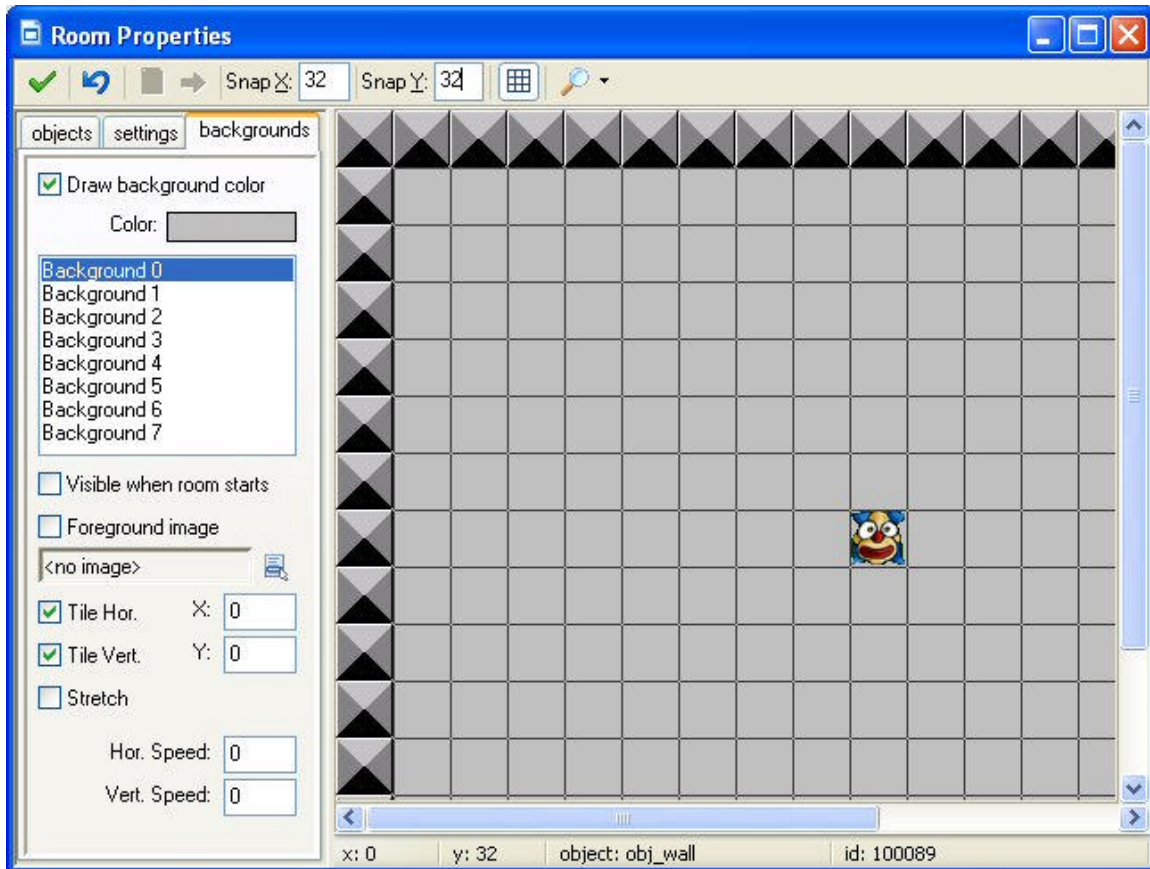
First of all we are going to add some background music. This is very easy. Again we add a sound resource to the game and this time we name it `snd_background`. For background music we don't use wave files. They use too much storage. Instead you best use midi files. Midi files are a different way of storing music. You can find lots of them on the web. Picking background music that fits the game is important. In the creation event of the clown object we add another sound action to play the background music. This time we set the property **Loop** to true to indicate that the sound should keep repeating itself forever.

Secondly we are going to add a background image. The grey background of the room is rather boring. To this end we use a new type of resource, the background resource. From the **Add** menu pick **Add Background**. A form appears that looks like the sprite form. Click the button **Load Background** and pick a nice image. The background form now looks as follows:



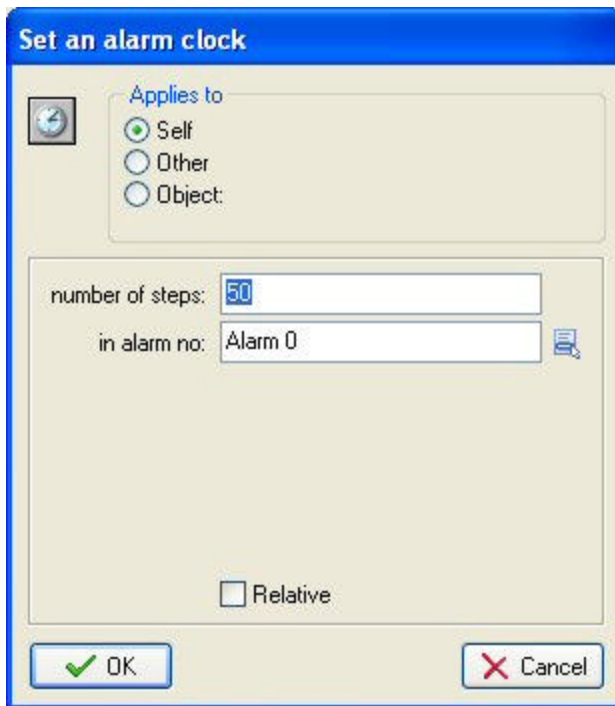
Note that for background the property **Transparent** is default off. Press **OK**.

To give the room the correct background, open the room again To this end, double click on the room resource at the left of the main window. Now use the tab **backgrounds** that is at the left top of the form. The following information is shown:



First uncheck the property **Draw background color** because we no longer need the gray background. Secondly, click on the little menu icon in the middle and pick the background we just added. As you will see, in the room we suddenly have a nice background. Note the two properties **Tile Hor.** and **Tile Vert.** They indicate that the background must be tiled horizontally and vertically, that is, repeated to fill the whole room. For this to work correctly the background image must be made such that it nicely fits against itself without showing cracks. Many such images are fortunately available.

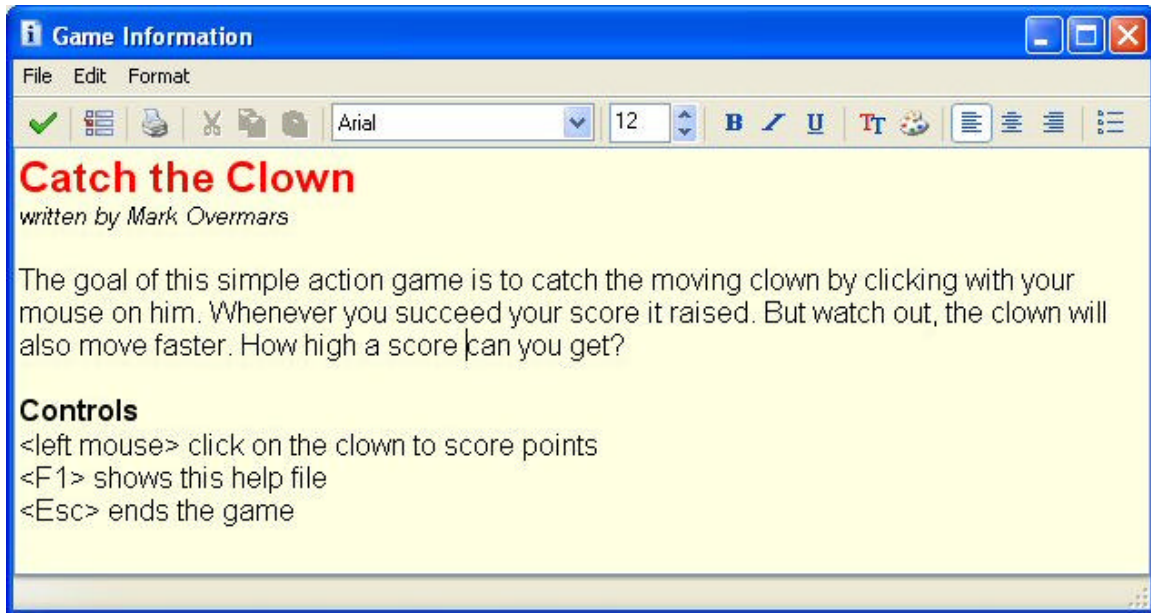
If you play the game a bit you will see that it is very easy because you know exactly where the clown is going. To make this more difficult we let the clown change its direction of motion from time to time. To this end we are going to use an alarm clock. These clocks tick down and at the moment they reach 0 an alarm event happens. Each instance of an object can have eight different alarm clocks but we need just one here. In the creation event of the clown we add the action to set an alarm clock (in the tab page **main2**). We use alarm 0 and set the number of steps to 50 (which is about 1.5 seconds because the game makes 30 steps per second; this can be changed in the **settings** tab in the room properties), as follows:



The image shows a dialog box titled "Set an alarm clock". It has a blue title bar and a light beige background. On the left, there is a small clock icon. To its right, under the heading "Applies to", are three radio buttons: "Self" (which is selected), "Other", and "Object:". Below this, there are two text input fields. The first is labeled "number of steps:" and contains the value "50". The second is labeled "in alarm no:" and contains the text "Alarm 0". To the right of the second field is a small icon of a document with a magnifying glass. At the bottom of the dialog, there is a checkbox labeled "Relative" which is currently unchecked. At the very bottom, there are two buttons: "OK" with a green checkmark icon and "Cancel" with a red X icon.

Next we must define what should happen when the alarm reaches 0. To this end, choose **Add Event**, select the **Alarm** event and pick alarm 0 from the pop up menu. In this event we set a new random direction of motion using the action with the red arrows (set the speed to 0 and check **Relative** to not change the speed). Also we set alarm 0 again to 50 to let the event happen again after 1.5 seconds. So now every 1.5 seconds the clown will change its direction of motion.

Finally, each game must tell the players what the goal is and how the user plays the game. So some help is required. *Game Maker* has a standard mechanism for this. To this end choose **Change Game Information** from the **Add** menu. An editor form will show in which you can type in some text. You can use different fonts, sizes, and colours. For example, you could type the following information:



During the game this text is automatically shown when the player presses the F1 key (like in most other programs).

Your first game is ready

Congratulations. You finished your first game. And the first game is always the most difficult one. You also saw the most important aspects of *Game Maker*: sprites, images and sounds, the game objects, events and actions, and the rooms. These are not particular to *Game Maker*. Sprites, images and sounds are important resources for any game making system. Also, in any game design system, it is customary and useful to think in terms of game objects that take actions in response to events. And of course each game design system will deal with levels or rooms as we call them in *Game Maker*.

Before continuing with a new game you might want to play a bit more with the *Catch the Clown* game we created in this tutorial. Here are some things you might want to try to add:

- Have two clowns moving around. (This is extremely easy because you can place multiple instances of the same object in a room.)
- Have a different dark clown that you should not catch because it will cost you part of your score.