WORKING WITH MULTIPLE ROOMS:
HOW TO USE ROOM ACTIONS IN GAMEMAKER

So far we have only used a single room in our games. However, you can have a game that uses multiple rooms. This is especially useful if you want to create a game that involves multiple levels.

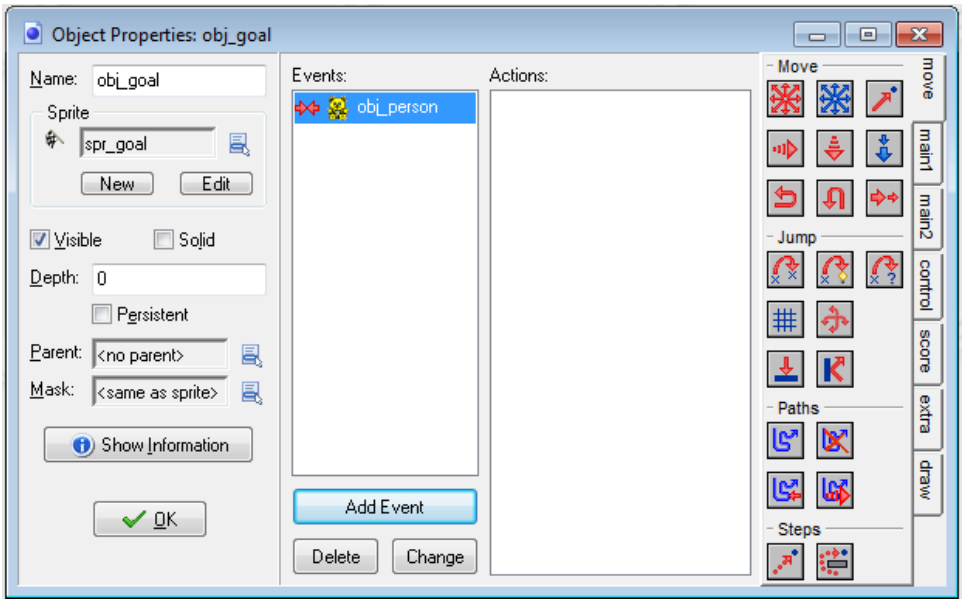
In this lesson, we will be creating a maze game that consists of multiple rooms. Each room will consist of a maze. To escape the maze the player must collect all the diamonds in the room and then reach the exit. To do so the player must solve puzzles and monsters must be avoided. Many puzzles can be created: blocks must be pushed in holes; parts of the room can be blown away using bombs, etc. It is very important to not show all these things in the first room. Gradually new items and monsters should appear to keep the game interesting.

For now, let's forget about the diamonds and just create a game in which the player simply must reach the exit. There are three crucial ingredients in the game: **the player**, **the wall**, and **the exit**. Each of these sprites and objects, together with the maze, can be found in a GameMaker file called **MazeGame.gmk** which you can find in the shared directory.

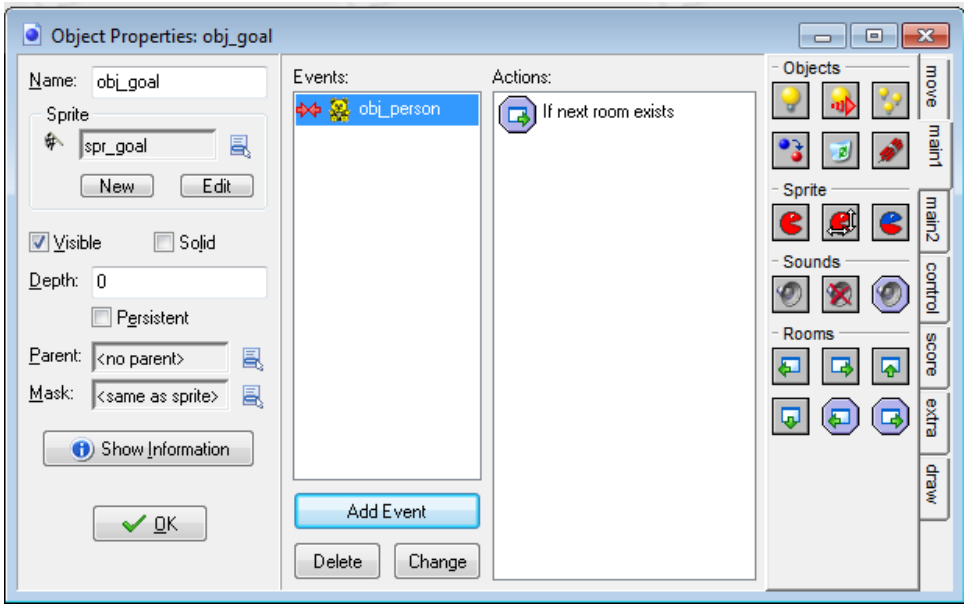
SETTING ROOM ACTIONS FOR THE GOAL OBJECT

The goal object (i.e. the finish flag) is the object the player has to reach. It is a non-solid object. When the player collides with it we need to go to the next room. So we will need to put this action in a **Collision Event**. There is, however, one drawback. It causes an error when the player has finished the last room. So we have to first check whether there is another room. If there is, we have to move the player to that room. Otherwise, we will just restart the game. So we will need to do the following:

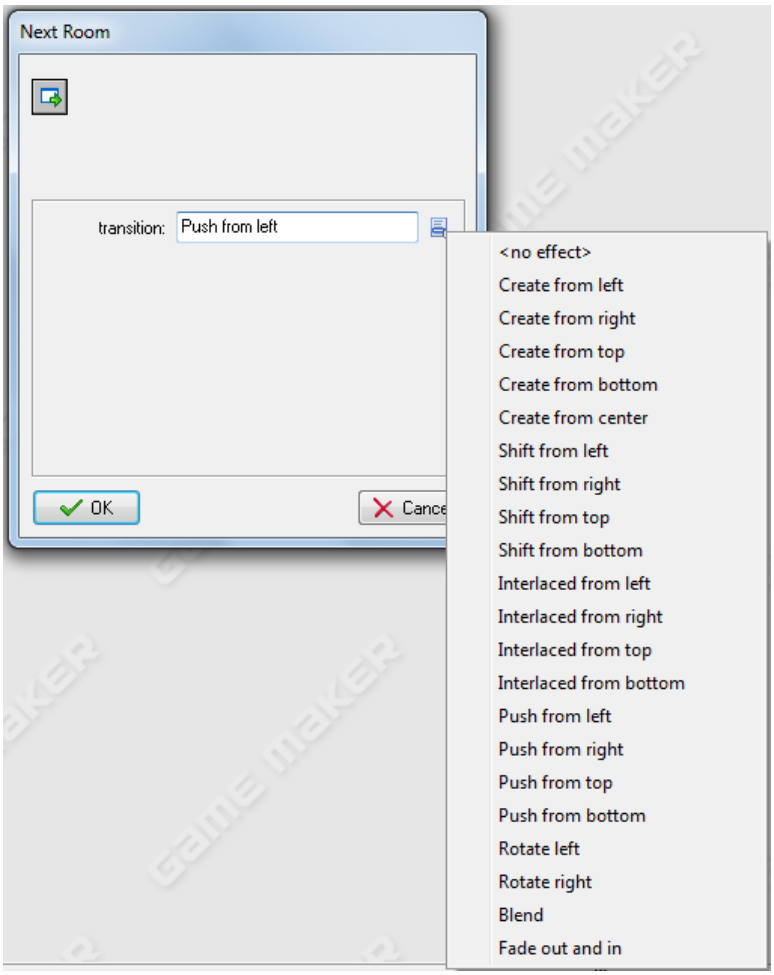
1. Double-click the **obj_person** object and add a **Collision Event**.



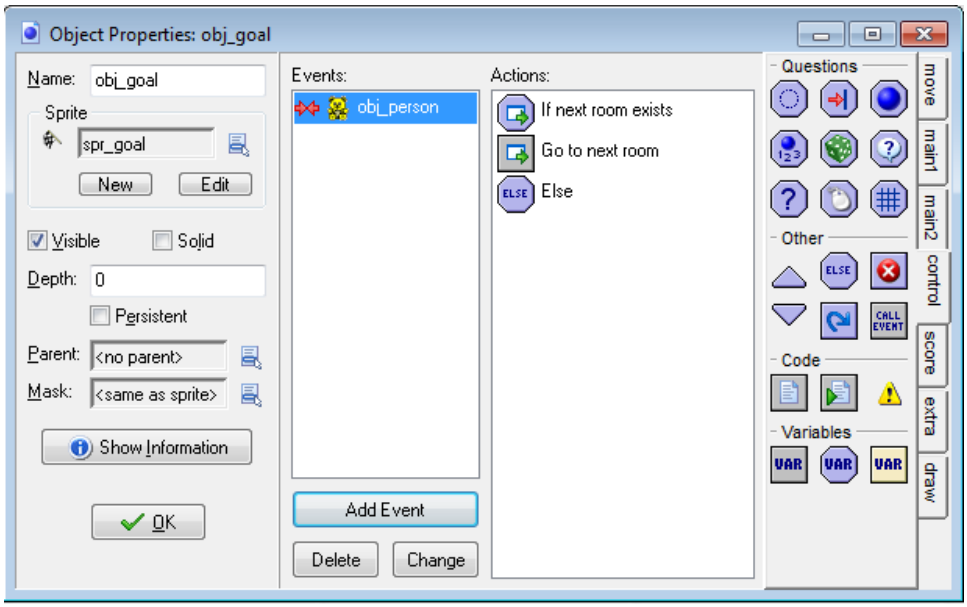
2. Add a **Check Next** action to the object which you can find in the **main1** tab. This action will check if there is another room.



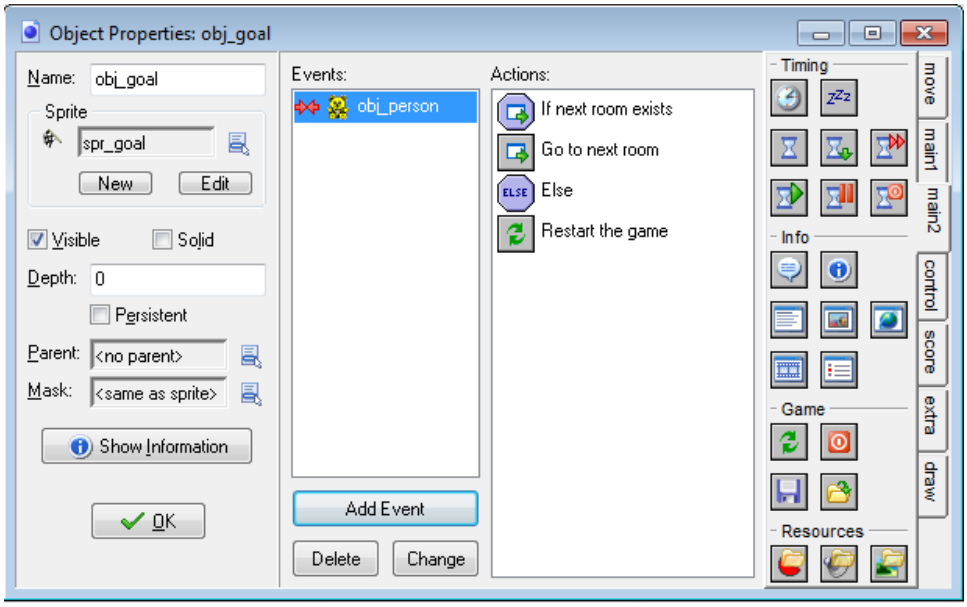
3. Add a **Next Room** action. This will bring up the next room if another room exists. You will have the option to select a transition, which determines the effect that is used when it transitions from one room to the next. Select any of the transition options.



4. Add an **Else** action because we need to indicate what we want to happen if there is not another room.

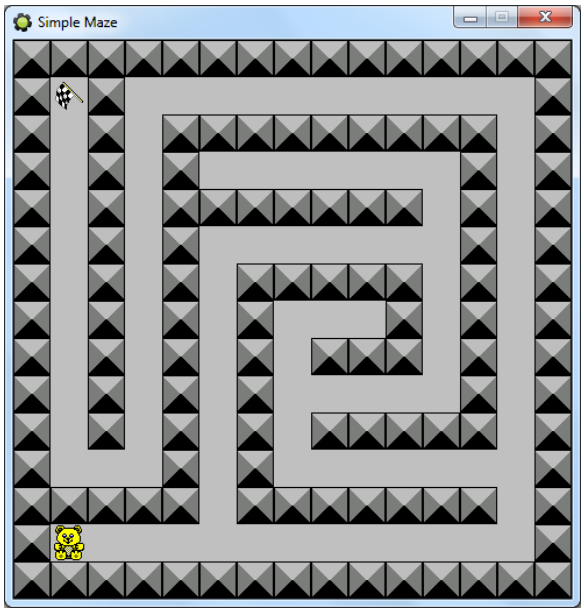


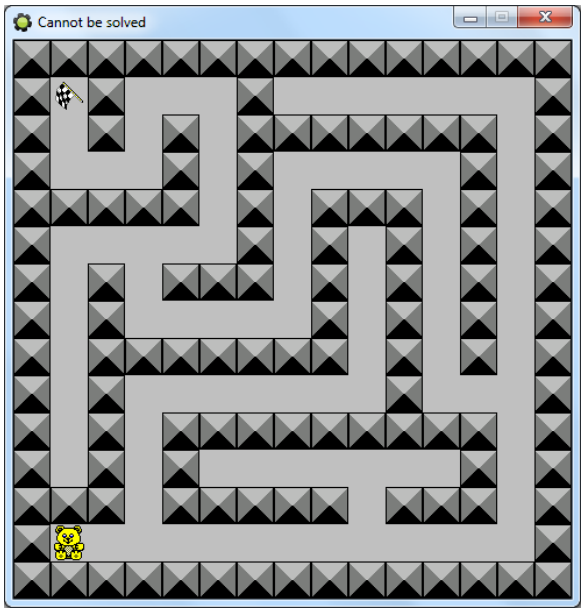
5. Finally, add a **Restart** action so the game restarts if there is no other room.



CREATING THE ROOMS

Create one or two rooms that look like a maze. In each room place the goal object at the destination and place the person object at the starting position.





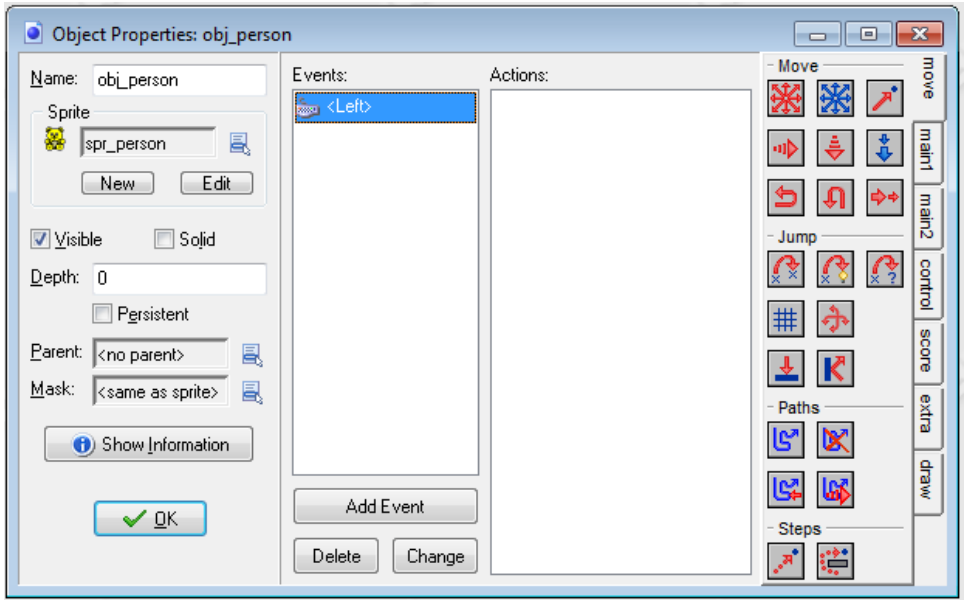
SETTING EVENTS AND ACTIONS FOR THE MAZE RUNNER

The maze runner must react to input from the user and it should not collide with a wall. We will use the arrow keys for movement. There are different ways in which we can make a character move. The easiest way is to move the player one cell at a time in the indicated direction when the player pushes the arrow key. A second way, which we will use, is that the character moves in a direction as long as the key is pressed. Another approach is to keep the player moving until another key is pressed (like in PacMan).

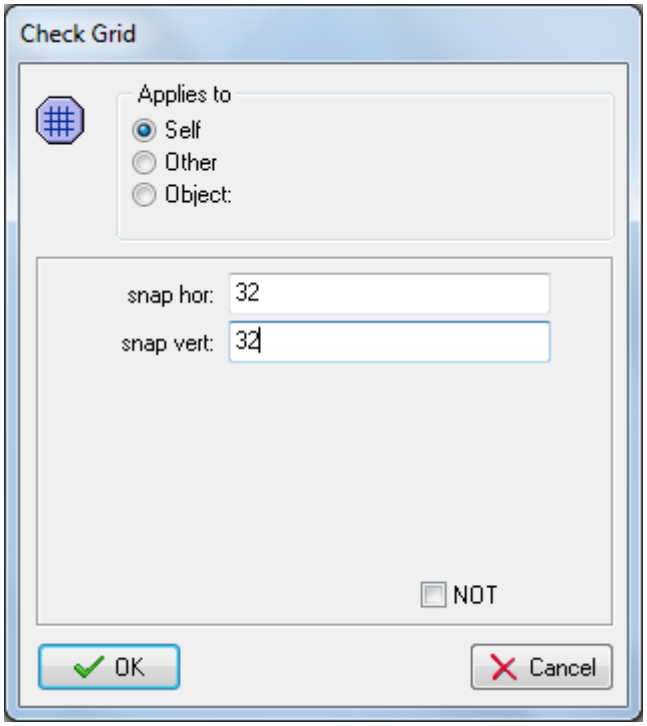
We need actions for all four arrow keys. The actions will simply set the right direction of motion. To stop when the player releases the key we will use the keyboard event for **<no key>**. This is where we will stop the motion.

Something we want to keep in mind though is the need to keep the character aligned with the cells of the grid that forms the maze. If we don't, motion becomes rather difficult. For example, you would have to stop at exactly the right position to move into a corridor. This can be achieved by using the **Check Grid** action.

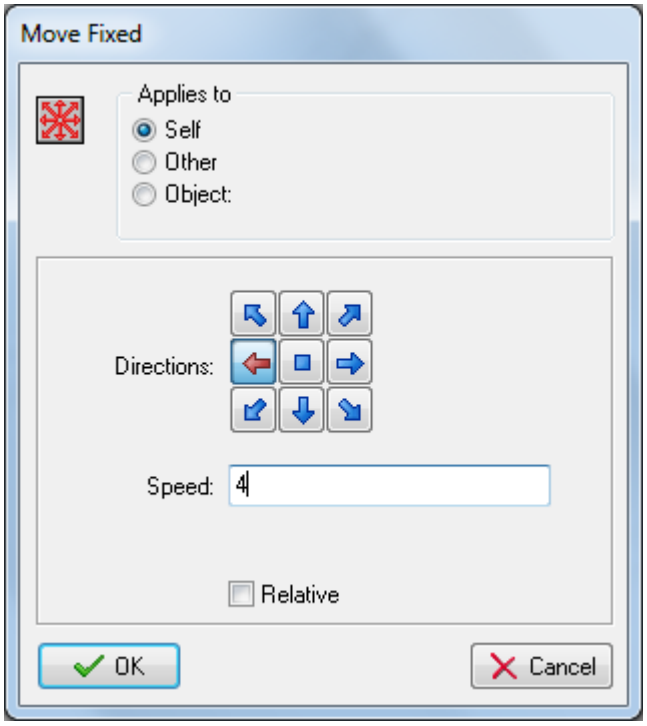
1. Open the character object (**obj_person**) and create a **<Left> Keyboard Event**.



2. In the **control** tab there is an action to test whether the object is aligned with a grid. The action is called **Check Grid**. Add the action to the keyboard event and set the parameters to **32** and **32** because that is the grid size of our maze.

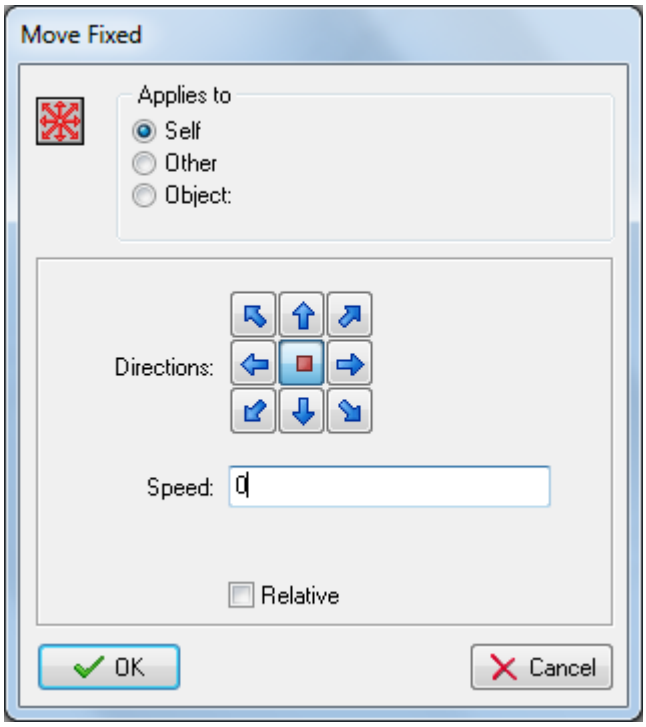


3. Now add the **Move Fixed** action and set the direction to **left** and set the speed to **4**.



4. Do the same for each arrow key event (i.e. up, down and right).

5. Finally, we also need to stop the motion when we hit a wall. For this we will need to add a **Collision Event** and stop the movement of the object.



That's it for now! Save and test the program to make sure everything works as it should.